Visnyk NTUU KPI Seriia – Radiotekhnika Radioaparatobuduvannia, 2023, Iss. 94, pp. 64–69

УДК 004.0569.55

Дослідження алгоритму Fletcher та розробка VHDL-моделі пристрою хешування

Гедеон Г. О., Гапак О. М., Тютюнникова Г. С., Гедеон Т. С., Маріна К. І.

Ужгородський національний університет, м. Ужгород, Україна

E-mail: *hanna.hedeon@uzhnu.edu.ua*

Актуальність апаратного моделювання алгоритму *Fletcher* пов'язана з його широким використанням для контролю автентичності документів і зображень, передачі даних між складовими різноманітних систем, а також з перевагами, які надає апаратна реалізація над програмною. У порівнянні з програмним підходом, апаратне моделювання дозволяє підвищити ефективність обробки даних завдяки оптимізації обчислень на апаратному рівні. Це призводить до значного зниження часу виконання операцій, що є критичним у великих системах обробки і передачі інформації, тому виникає необхідність у детальному вивченні характеристик цього алгоритму. У статті представлено етапи розробки пристрою хешування інформації на основі алгоритму *Fletcher*-64 у середовищі *Active-HDL*. Для реалізації моделі використано мову опису апаратури *VHDL*. Проведено тестування роботи пристрою, розглянуто його особливості проєктування. Наведено опис інтерфейсної частини пристрою із зазначенням розміру шин даних, опис архітектури об'єкта, а також показано моделювання роботи розробленої *VHDL*-моделі *Fletcher*-64. *VHDL*-модель *Fletcher*-64 обробляє інформацію 32-бітними блоками за один такт. Значення хеш-суми зберігається у шині *OUT_DATA* у *hex* форматі. Проведено порівняльну характеристику *Fletcher* з алгоритмом *Adler*-32. Визначено, що *Fletcher*-32 і *Fletcher*-64 забезпечують краще перемішування бітів, тоді як *Fletcher*-16 – поступається *Adler*-32 у виявленні помилок та перемішуванні бітів. Визначено, що використання *Fletcher*-32 для контролю цілісності даних є більш ефективним порівняно з алгоритмом *Adler*-32 завдяки кращому виявленню помилок. Проведено оцінку доцільності використання різних версій *Fletcher* для вхідних повідомлень змінної довжини із врахуванням особливостей алфавіту. У результаті проведеної роботи визначено шляхи подальших досліджень, що спрямовані на пошук колізій для алгоритмів *Fletcher*, *Adler*, *CRC*; визначено допустимі сфери використання *Fletcher* і *Adler*-32.

Ключові слова: Fletcher; *VHDL*; *HDL*; хешування; контрольна сума; модель; апаратне моделювання; *Adler*; проєктування; хеш

DOI: 10.20535/RADAP.2023.94.64-69

Вступ

У сучасному світі, де надходить надмірна кількість цифрових даних, збереження та оптимальне управління ними стають важливими завданнями. У цьому контексті розглядається використання апаратних методів для моделювання алгоритмів хешування з метою забезпечення цілісності даних та їх ефективного оброблення. Метою статті є розробка *VHDL*-моделі контрольної суми *Fletcher*-64, моделювання та дослідження роботи алгоритму хешування з використанням середовища розробки *Active-HDL*.

Для розробки моделі алгоритму *Fletcher* використано спеціалізовану мову програмування *VHDL*, яка дозволяє спроєктувати структуру, промоделювати роботу пристрою [1], а також відслідкувати особливості апаратної реалізації алгоритму. Крім того, покрокове моделювання обрахунку хешу дозволяє проаналізувати основні етапи роботи пристрою та розглянути шляхи оптимізації роботи розробленої *VHDL*-моделі.

1 Аналіз основних досліджень і публікацій

Використання алгоритму *Fletcher* забезпечує виявлення помилок, близьке до *CRC*, при менших обчислювальних ресурсах. Авторами статті [2] досліджено використання 32-бітної версії *Fletcher* для контролю автентичності сертифікатів, шляхом знаходження контрольних сум для оригінальних і модифікованих документів. Автентичність сертифіката підтверджується значеннями хешу, збереженими у системі. У випадку, якщо 32-бітний код відрізняється, це свідчить про зміни у сертифікаті. Для тестування використовуються зображення серти-

фіката у форматі *JPEG* з роздільною здатністю 902×1280 пікселів.

Робота [3] присвячена дослідженню модифікацій двовимірних зображень у форматі *JPG*, непомітних для розпізнання людським оком з використанням хешування *Adler*-32, що базується на *Fletcher*. Під час обробки кольорові зображення замінюються зображеннями у градаціях сірого, обраховується контрольна сума, яка і дозволяє підтвердити відсутність чи наявність змін у зображенні. Як описує автор, використання *Adler*-32 дозволяє ефективно відслідковувати маніпуляції із зображеннями, а також забезпечує високу швидкодію.

Adler-32 також використовується як інструмент для аутентифікації у проєкті [4], тоді як модифікована версія алгоритму (*M-Adler*-32) – як засіб аутентифікації транспортних засобів у системі управління транспортними мережами для контролю дорожнього руху [5]. У статті [6] наведено використання *Adler*-32 у системі охорони здоров'я, хеш-сума формується з врахуванням ідентифікатора пацієнта, зареєстрованого в системі, а також точного часу його реєстрації.

У контексті використання алгоритму *Adler*-32 для аутентифікації [4–6], виникає проблема, пов'язана із довжиною вхідних даних. Нерівномірне перемішування бітів та велике значення *mod* сприяють швидкому виявленню вразливостей при роботі з короткими обсягами даних. Це може призвести до підбору вхідних комбінацій та порушення безпеки системи. Використання *Adler*-32 у сферах, де конфіденційність даних є критично важливою (наприклад, системи управління транспортними мережами або системи охорони здоров'я), передбачає необхідність впровадження додаткових засобів безпеки або розгляду алгоритмів з вищим рівнем стійкості. Результати дослідження особливостей роботи алгоритму *Adler*-32 можуть стати підставою для вибору альтернативних методів контролю цілісності інформації.

Fletcher розглядається як інструмент контролю правильності передачі даних у комунікаційних інтерфейсах [7]. У статті наведено апаратне моделювання алгоритму *Fletcher*-16 із шиною вхідних даних 240 бітів, що включає операцію *divide*, яка вносить значну затримку обчислень. У роботі розглядаються різні підходи до реалізації алгоритму, виділяються переваги та недоліки кожного. Автор зазначає, що вибір відповідного методу повинен базуватися на індивідуальних вимогах до конструкції пристрою та використовуваній ПЛІС.

У джерелі [8] наведено проєкт *CAR4*, де алгоритм *Fletcher* використовується для перевірки правильності надісланих і отриманих даних при передачі інформації між блоком керування та пультом дистанційного керування. Передача інформації відбувається по шині *UART* з використанням блоків *UART Rx* та *UART Tx* у вигляді восьмибітних беззнакових значень. Збіг обрахованих контрольних сум на обох блоках свідчить про успішну передачу даних.

У випадку апаратного підходу до реалізації *Fletcher* [7, 8], недоліком є затримка обчислень, пов'язана з використанням версії *Fletcher*-16 та операції *divide*. Вибір 16-бітного *Fletcher* з шиною даних розміром 240 бітів може бути неоптимальним рішенням у контексті високоефективної обробки даних. 240-бітний блок вхідних даних [7] визначає хеш за 30 тактів (обробляється по 8 бітів за кожен такт). Використання 32-бітної версії алгоритму скоротить процес обрахунку вдвічі (до 15 тактів). Зменшення впливу операції *divide* на швидкодію алгоритму *Fletcher* можливе при використанні компаратора для порівняння поточного хешу із значенням *mod*. Компаратор, ґрунтуючись на простій порівняльній логіці, потребує менше обчислень порівняно з операцією *divide*. Такий підхід сприяє оптимізації використання ресурсів та підвищує ефективність алгоритму у випадках, де можливо уникнути використання операції *divide*.

2 Контрольна сума *Fletcher*

Контрольна сума *Fletcher* (*Fletcher checksum*, *FCS*) передбачає поділ вхідного повідомлення на короткі бітові блоки, які надалі використовуються для обчислення хешу. *Fletcher*-16 використовує блоки розміром 8 бітів, *Fletcher*-32 – блоки по 16 бітів і *Fletcher*-64 – блоки по 32 біти. Розмір блоку удвічі менший від розміру шуканої *FCS*.

Якщо позначити блоки $B[i]$ ($i = 1 \ldots n$, де n – загальна кількість блоків), то старша частина хешу FCS_{HIGH} розраховується як перша комплементарна сума величин $B[i]$, а молодша частина FCS_{LOW} – як перша комплементарна сума величин $(n+1-i) \times B[i]$ (перший комплемент від суми $(n \times B[1] + (n-1) \times B[2] + \ldots + B[n])$, FCS – конкатенація FCS_{HIGH} і FCS_{LOW} [9].

На кожному етапі обрахунку проміжних блоків використовується модульна арифметика. Значення модулю M залежить від розміру FCS. Для *Fletcher*-16, *Fletcher*-32 і *Fletcher*-64 використовуються модулі 255 (2^{8-1}), 65 535 (2^{16-1}) і 4 294 967 295 (2^{32-1}) відповідно.

Формули для розрахунку FCS можна записати таким чином:

$$FCS_{HIGH} = (B[1]+B[2]+B[3]+\ldots+B[n]) mod M,$$

$$FCS_{LOW} = (n \times B[1]+(n-1)B[2]+\ldots+B[n]) mod M,$$

$$FCS = FCS_{HIGH} + FCS_{LOW}.$$

Варто зауважити, контрольна сума *Fletcher* не розрізняє блоки, що складаються тільки з нулів або тільки з одиниць. Наприклад, якщо 16-розрядний блок у слові даних змінюється з $0x0000$ на $0xFFFF$, хеш *Fletcher*-32 залишається незмінним.

Приклад визначення 16-бітної контрольної суми за алгоритмом *Fletcher*. Для *Fletcher*-16 вхідне повідомлення розбивається на блоки розміром по 8 бітів, у якості модуля використовується значення 2^{8-1}. Поетапне обчислення хешу для 616263346431_{16} наведено у Таблиці 1.

Табл. 1 Приклад розрахунку *Fletcher*-16

i	$Data$	$ASCI(hex)$	FCS_{HIGH}	FCS_{LOW}
1	a	61	61_{16}	61_{16}
2	b	62	$C3_{16}$	25_{16}
3	c	63	27_{16}	$4C_{16}$
4	4	34	$5B_{16}$	$A7_{16}$
5	d	64	BF_{16}	67_{16}
6	1	31	$F0_{16}$	58_{16}
FCS			$58F0_{16}$	

Отже, для вхідного повідомлення «*abc4d1*» (код *ASCI*: 616263346431_{16}) контрольна сума *Fletcher*-16 дорівнює $58F0_{16}$.

3 Порівняння *Fletcher* з алгоритмом *Adler*-32

Хеш-функція *Adler*-32, розроблена Марком Адлером, є модифікацією алгоритму *Fletcher*-32. Контрольна сума *Adler*-32 вираховується шляхом обчисленнях двох 16-бітних частин A і B та конкатенації їх у 32-розрядний хеш. При ініціалізації $A = 1$, $B = 0$. При обчисленні A і B, так як і в алгоритмі *Fletcher*, використовується модульна арифметика. Значення модуля – 65 521. Число 65 521 – це найбільше просте число менше, ніж 2^{16} (65 536) [10].

Вибір простого числа для операції *mod* призводить до кращого перемішування бітів, тоді як зменшення розміру множини можливих значень знижує продуктивність. Результати дослідження *The Effectiveness of Checksums for Embedded Control Networks* [11] показують, що *Fletcher*-32 має перевагу у порівнянні з *Adler*-32 як у продуктивності, так і у здатності виявляти помилки. Оскільки залишок за модулем 65 535 значно простіше і швидше реалізувати, ніж залишок за модулем 65 521, контрольна сума *Fletcher*-32, зазвичай, є швидшим алгоритмом. Здатність алгоритму *Fletcher* до більш ефективного виявлення помилок порівняно з *Adler*-32 свідчить про його надійність при контролі передачі інформації між різними блоками даних чи пристроями, як у проєкті [8].

Що стосується виявлення помилок, перемішування бітів, яке реалізоване у алгоритмі *Adler*, повинно було забезпечити кращі показники, ніж у *Fletcher*. Але, за результатами дослідження *RFC* 3385 [12], *Adler*-32 поступається *Fletcher*-32. Хешування *Adler* має переваги над *Fletcher* тільки у випадку 16-бітного хешу.

Відомості отримані із робіт [2,3] підтверджують, що використання алгоритмів *Adler*-32 і *Fletcher*-32 для контролю цілісності цифрових документів, таких як сертифікати, дозволяє відстежувати заміну файлів та незначні модифікації.

Результати дослідження [13] показують, що при використанні англійського алфавіту, розмір вхідного повідомлення *Adler*-32 для перевищення частини A (значення 65 521), що складається із літер у верхньому регістрі, становить 867 байтів і для нижнього регістру – 610 байтів (у кодуванні *ASCII*). Розділові знаки та пробіли не враховуються. У статті [5] зазначено, що модифікований *Adler*-32 використовується для аутентифікації транспортного засобу з використанням ідентифікатора користувача та даних автомобіля. Це один із кроків, що служить для безпечної передачі даних. Тобто, враховуючи результати [13], виникає необхідність дослідити особливості формування хеш-суми *M-Adler*-32 та виникнення колізій, що дозволить, при наявності хешованого значення, підібрати можливі ідентифікатори для доступу у систему. Також, у якості аналога *M-Adler*-32, розглянути використання *Fletcher*, який має суттєві переваги у порівнянні з *Adler*-32 [12].

Використання *Adler*-32 для контролю доступу пацієнтів у системі охорони здоров'я, наведене у статті [6], потребує ретельного розгляду та дослідження. Це пов'язано з тим, що хеш-сума обчислюється для унікального ідентифікатора пацієнта та мітки часу. Точний час реєстрації користувача може включати дату (число, місяць, рік) і час з урахуванням годин, хвилин, секунд та *GMT* (*Greenwich Mean Time*). Наприклад, у форматі *2023-10-03, 08:13:23 GMT +2*; *October 3, 2023, 08:13:23 GMT +2* або *2023-10-03T08:13:23+02:00*, тобто до 35 байтів у кодуванні *ASCII*. Що стосується ідентифікатора пацієнта, у роботі не зазначено точний розмір, але можна припустити, що розмір унікально коду в сумі з часовою міткою не перевищує 610 байтів. Отже, значення 65 521 частини A для *Adler*-32 не досягається, що значно спрощує підбір вхідних даних і, в свою чергу, може надати доступ до особистих даних пацієнтів у системі.

Для оцінки доцільності використання алгоритму *Fletcher* та порівняння його з *Adler*-32 проведено дослідження, результати якого представлені на Рис. 1, де: A – мінімальна кількість розділових знаків та цифр, необхідних для досягнення значення модуля; B – мінімальна кількість літер верхнього регістру, необхідних для досягнення значення модуля; C – мінімальна кількість літер нижнього регістру, необхідних для досягнення значення модуля.

Згідно з Рис. 1, при обчисленні хешу за алгоритмом *Fletcher*-32 для досягнення значення в 65 535 для розділових знаків і цифр мінімальна кількість символів складає 16, тоді як для *Adler*-32 – 1985; для літер верхнього регістру – 8 і 1008 відповідно; для літер нижнього регістру – 6 і 675 відповідно.

Для алгоритму *Fletcher*-16 кількість байтів складає 4, 5, 9, проте, це пов'язано із невеликим розміром модуля – 255; для *Fletcher*-64 – 12, 16, 32 байти для розділових знаків і цифр, великих, малих літер відповідно. Пробіли і частотні характеристики алфавіту не враховані.

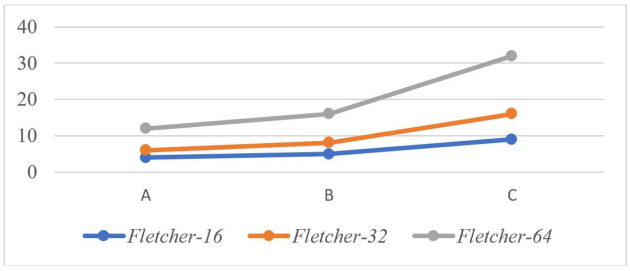

Рис. 1. Діаграма порівняння версій алгоритму *Fletcher*

Результати свідчать про те, що при хешуванні коротких повідомлень доцільно використовувати *Fletcher*-32 або *Fletcher*-64, що забезпечує рівномірний розподіл обчислень для коротких наборів інформації у порівнянні з алгоритмом *Adler*-32.

Використання 16-бітного *Fletcher* з модулем 255 є неефективним, адже хеш-функція містить колізії і дозволяє швидко підібрати вхідні значення для отримання ідентичного хешу, наприклад, методом *brute-force*. Також розмір хешу *Fletcher*-16 передбачає високу ймовірність невиявлення помилок у порівнянні з 32-бітною версією алгоритму [14] та *Adler*-32.

Результати проведених досліджень показують, що вибір простого числа для операції *mod* в алгоритмі *Adler*-32 призводить до гіршої ефективності порівняно з *Fletcher*-32 та *Fletcher*-64, особливо це стосується перемішування бітів та здатності виявляти помилки. Алгоритм *Adler*-32 є неоптимальним рішенням для обробки коротких вхідних наборів через потребу в значущих розмірах даних для досягнення значення *mod*. Алгоритм *Fletcher*, в свою чергу, може використовуватись для вхідних повідомлень від 12 байтів.

4 Моделювання роботи *VHDL*-моделі *Fletcher*-64

Інтерфейс об'єкта моделювання (Рис. 2) містить n-бітну (n – розмір блоку у бітах) вхідну шину завантаження даних, керуючі сигнали (сигнал синхронізації, сигнал скидання, сигнал дозволу обробки даних), двонапрямлену шину формування контрольної суми та вихідний сигнал, що свідчить про готовність вихідних даних.

```
29  entity FCS is
30        generic (DATA_WIDTH: INTEGER := 64;
31      N: INTEGER := 32);
32        port(
33             CLK : in STD_LOGIC;
34             RESET : in STD_LOGIC;
35             ENABLE : in STD_LOGIC;
36             IN_DATA : in STD_LOGIC_VECTOR(N-1 downto 0);
37             OUT_DV : out STD_LOGIC;
38             OUT_DATA : inout STD_LOGIC_VECTOR(DATA_WIDTH-1 downto 0)
39
40             );
41  end FCS;
```

Рис. 2. *VHDL*-опис інтерфейсу модуля *Fletcher*-64

У секції *port* опису інтерфейсу об'єкта моделювання у круглих дужках розташовуються декларації всіх зовнішніх портів із зазначенням режиму роботи і типу переданих даних. У секції *generic* зазначені константні значення, які не змінюються у ході виконання програми. У цьому випадку як константа задається розмір контрольної суми у бітах.

Інтерфейс *Fletcher*-16 і *Fletcher*-32 відповідає опису *Fletcher*-64 за виключенням розмірності контрольної суми (значення *DATA_WIDTH*) та розміру блоку даних (значення N). В архітектурі розкривається функціональна сутність об'єкта моделювання, що представлений у вигляді «чорного ящика» в інтерфейсній частині *entity*.

Для обрахунку контрольної суми *Fletcher* задані внутрішні сигнали *FCS_HIGH* і *FCS_LOW* (старша та молодша частини хеш-суми), сигнал B (блок даних) та константа M (модуль). Для *Fletcher*-64 $M = 4294967295_{10}$ або $FFFFFFFF_{16}$. Для роботи з операцією *mod* (передбачає роботу виключно з числовими типами даних) використовується тип даних *signed*. Стандартом *VHDL* [15] для подання числових даних передбачений тип даних *integer* (цілий), що включає числа від –2 147 483 648 до +2 147 483 647. Зазначений діапазон є недостатнім для представлення числа $2^{32}-1$, тому виникає необхідність конвертувати значення модуля у тип *SIGNED* для роботи з операцією *mod*. Для конвертування із типу даних *STD_LOGIC_VECTOR* у тип *SIGNED* підключено пакет *NUMERIC_STD* бібліотеки *IEEE* [15]. Фрагмент коду, в якому виконується обчислення залишку від ділення для частини *FCS_HIGH*:

FCS_HIGH<=STD_LOGIC_VECTOR((SIGNED(B) + SIGNED (OUT_DATA (N-1 downto 0)) mod SIGNED (M)).

Для перевірки роботи *VHDL*-моделі *Fletcher*-64 на шину *IN_DATA* на кожному такті подаються 32-бітні двійкові набори, які у кодуванні *ASCII* відповідають рядку *abc4d1AF8377aajj*. Симуляція проєкту у середовищі *Active-HDL* виконується протягом чотирьох тактів, за кожен такт обробляється по чотири байти вхідних даних, з яких формується вихідний хеш. Результат зберігається у двонапрямленій шині *OUT_DATA*.

На першому такті *OUT_DATA* зберігає значення 3463626134636261_{16}, тоді як «0» на ви-

ході *OUT_DV* вказує про неготовність кінцевої хеш-суми (Рис. 3). На наступних тактах продовжується обчислення хешу при подачі наборів вхідних даних: 64314146_{16}; 38333737_{16}. Поява «1» на виході *OUT_DV* свідчить про готовність результату, шина *OUT_DATA* містить хеш-суму $7d29e5831c46285f_{16}$ (Рис. 4). Внутрішні сигнали *FCS_HIGH* та *FCS_LOW* дозволяють відстежувати формування старшої та молодшої частини хешу на кожному кроці.

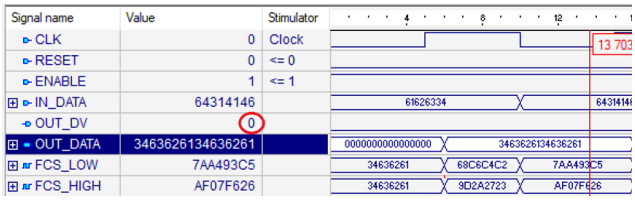

Рис. 3. Результати *Fletcher*-64 на першому такті

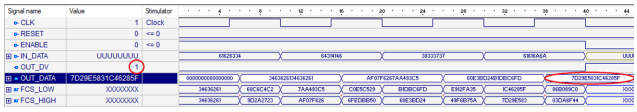

Рис. 4. Формування хеш-суми *Fletcher*-64

Для контролю обчислень використано онлайн-сервіс [16]. Обрахований хеш збігається із *hex* значенням, наведеним на Рисунку 4. Отже, розроблена *VHDL*-модель алгоритму хешування *Fletcher*-64 працює правильно.

Висновки

Результатом дослідження є реалізована *VHDL*-модель пристрою хешування інформації на основі алгоритму *Fletcher*-64. Розроблена модель дозволяє обраховувати хеш-суму для вхідного потоку інформації; обробляє інформацію 32-бітними блоками за один такт. Значення хеш-суми зберігається у шині *OUT_DATA* у *hex* форматі.

Актуальність апаратного підходу при розробці ефективних засобів контролю цілісності даних та перевірки автентичності цифрових документів пов'язана із високою швидкістю обробки інформації, яка досягається шляхом оптимізації обчислень на апаратному рівні. Проведене дослідження підтверджує, що алгоритм хешування *Fletcher* є надійним та широко використовуваним інструментом виявлення помилок при передачі даних каналами зв'язку, а також засобом контролю модифікацій двовимірних зображень, непомітних людському оку.

Подальші дослідження спрямовані на знаходження колізій у алгоритмах *Fletcher*-16, *Fletcher*-32, *Fletcher*-64, *Adler*-32, *CRC*-16, *CRC*-32, *CRC*-64. Це дозволяє визначити швидкість підбору вхідних даних для отримання ідентичних контрольних сум та розрахувати кількість колізій для кожного з алгоритмів на прикладі використання методу повного перебору з обмеженим алфавітом. Очікується, що ефективніше перемішування бітів, реалізоване в алгоритмах *CRC*, *Fletcher*, призводить до зменшення кількості виникнення колізій.

References

[1] Pecheux, F., Lallement, C., Vachoux, A. (2005). VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 2, pp. 204-225. doi: 10.1109/TCAD.2004.841071.

[2] Syifa, S. (2022). Implementasi Checksum Dengan Menggunakan Algoritma FletcherUntuk Mendeteksi Keaslian Sertifikat Rumah. *JUSSI: Jurnal Sains Dan Teknologi Informasi*, Vol 2, No. 1, pp. 1-6.

[3] Lahagu, J. (2019). Mendeteksi orisinalitas citra digital dengan menerapkan metode Adler-32. *KOMIK (Konferensi Nasional Teknologi Informasi dan Komputer)*, Vol 3, No. 1, pp. 789-797. doi: 10.30865/komik.v3i1.1694.

[4] Velmurugadass, P., Dhanasekaran, S., Shasi Anand, S., Vasudevan, V. (2022). Quality of Service aware secure data transmission model for Internet of Things assisted wireless sensor networks. *Transactions on Emerging Telecommunications Technologies*, pp. 1-25. doi: 10.1002/ett.4664.

[5] Ruhin Kouser R., Manikandan T. (2023). A Robust Vehicular Networking Management System for the Traffic Control Using PK-DCNN Classification And L-SSA Based Rerouting. *International Conference on Electronics, Communication and Computing Technologies (ICECCT)*, doi: 10.1109/ICECCT56650.2023.10179853.

[6] Kumar, M., Mukherjee, P., Verma, S., Kavita, K. et al. (2022). BBNSF: Blockchain-Based Novel Secure Framework Using RP2-RSA and ASR-ANN Technique for IoT Enabled Healthcare Systems. *Sensors*, Vol. 22, No. 23, 9448. doi: 10.3390/s22239448.

[7] Wawrzyn, E., Wawrzyniak, Z. M., Wojeński A. (2021). Comparison of implementation methods for data processing algorithms on FPGA. *Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments*, Vol. 12040. doi: 10.1117/12.2612247.

[8] Šůstek, J. (2020). *Design of Control Unit for the Robotic Vehicle Car4 (Bachelor's thesis)*. Brno: Brno University of Technology. Faculty of Mechanical Engineering. Institute of Mechanics, Mechatronics and Biomechanics.

[9] Olykh, O. Y. (2015). *Modern Computer Technologies. Principles of Building Computer Networks: A textbook*. Kyiv: VPC "Kyiv University", 479 p.

[10] Buriachok, V. L., Kyrychok, R. V., Skladannyy, P. M. (2018). *Fundamentals of Information and Cybersecurity: A textbook*. Kyiv, 320 p.

[11] Maxino T. C., Koopman P. J. (2009). The Effectiveness of Checksums for Embedded Control Networks. *IEEE Transactions on Dependable and Secure Computing*, Vol. 6, No. 1, pp. 59-72. doi: 10.1109/TDSC.2007.70216.

[12] Sheinwald, D., Satran, J. (2002). Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC). Checksum Considerations. *Network Working Group*, Request for Comments, RFC 3385, pp. 1-23. doi: 10.17487/RFC3385.

[13] Hapak, O. M., Hedeon, H. O. (2023). Hardware control unit of effective option of hashing module. *Science and Technology Today*, Vol. 2(16), pp. 373-380. doi: 10.52058/2786-6025-2023-2(16)-373-380.

[14] Sahu, S. K. (2019). Hardware instruction based crc32c, a better alternative to the tcp one's complement checksum. *Texas A&M University*, 86 p.

[15] IEEE Standard for VHDL Language Reference Manual. (2019). *IEEE Std 1076-2019*, doi: 10.1109/IEEESTD.2019.8938196.

[16] *CyberChef*. GitHub Repository.

Studying of the Fletcher algorithm and developing VHDL model of hashing device

Hedeon H. O., Hapak O. M., Tiutiunnykova H. S., Hedeon T. S., Marina K. I.

The relevance of hardware modelling of the Fletcher algorithm is related to its widespread use for control of document and image authentication, data transfer between components of various systems, and the advantages provided by hardware implementation over software. Compared to a software-based approach, hardware modelling can improve data processing efficiency by optimising computations at the hardware level. This leads to a significant reduction in the execution time of, so there is a need for a detailed study of the characteristics of this algorithm.

The article presents the stages of development of an information hashing device based on the Fletcher-64 algorithm in the Active-HDL environment. The VHDL hardware description language is used to implement the model. The device is tested and its design features are considered. A description of the interface part of the device with the size of the data buses, a description of the object architecture, and a simulation of the developed VHDL model of Fletcher-64 are given. The Fletcher-64 VHDL model processes information in 32-bit blocks in one cycle. The value of the hash sum is stored in the OUT_DATA bus in hex format.

A comparative characterisation of Fletcher with the Adler-32 algorithm is carried out. It is determined that Fletcher-32 and Fletcher-64 provide better bit shuffling, while Fletcher-16 is inferior to Adler-32 in error detection and bit shuffling. It is determined that the use of Fletcher-32 for data integrity control is more efficient than the Adler-32 algorithm due to better error detection. The feasibility of using different versions of Fletcher for incoming messages of variable length, taking into account the peculiarities of the alphabet, is assessed.

As a result of the work, the ways of further research aimed at finding collisions for the Fletcher, Adler, CRC algorithms are identified; the acceptable areas of use of Fletcher and Adler-32 are determined.

Keywords: Fletcher; VHDL; HDL; hashing; checksum; model; hardware modelling; Adler; design; hash