Степан Николаевич Белан, кандидат технических наук, доцент, доцент кафедри кибербезопасности и использования автоматизированных информационных систем и технологий, Институт специальной связи и защиты информации Национального технического университета Украины "Киевский политехнический институт имени Игоря Сикорского", Киев, Украина.

Андрей Андреевич Демаш, заместитель начальника научно-исследовательского отдела, Государственный научно-исследовательский институт специальной связи и защиты информации, Киев, Украина.

Stepan Bilan, candidate of technical sciences, associate professor, associate professor of cybersecurity and application of information systems and technologies academic department, Institute of special communications and information protection National technical university of Ukraine "Igor Sikorsky Kyiv polytechnic institute", Kyiv, Ukraine.

Andrii Demash, deputy head of research division, State research institute for special telecommunication and information protection, Kyiv, Ukraine.

УДК 004.032.2:512.624

ШОЛОГОН ОЛЬГА,
ШОЛОГОН ЮЛІЯ

ЗМЕНШЕННЯ ЗАГАЛЬНОЇ КІЛЬКОСТІ ЛОГІЧНИХ ЕЛЕМЕНТІВ У КЛАСИЧНОМУ ДВОКРОКОВОМУ ПОМНОЖУВАЧІ ЗАСОБАМИ VIVADO HLS

В Україні практично всі реалізації захисту інформації є програмними, основним недоліком яких, є недостатня стійкість до зламу, тому для збільшення надійності реалізації захисту інформації виникає необхідність у створенні апаратних засобів для виконання операцій над елементами скінченних полів. Однією з можливостей є реалізація на програмованих логічних інтегральних схемах. Як правило, помножувачі в полях Галуа $GF(p^m)$ будуються за допомогою засобів мови VHDL. Основним недоліком такого підходу є значні часові та апаратні затрати. В даній статті, запропоновано будувати помножувач у полях Галуа $GF(p^m)$ за допомогою середовища Vivado HLS. В роботі розглянуто метод оптимізаціїї при якому використовувались типи з визначеною точністю. В результаті досліджень, було доведено ефективність використання середовища Vivado HLS у порівнянні із засобами VHDL. Кількість найпростіших логічних елементів було зменшено у 3 рази, а також кількість тригерів із динамічним і потенційним управлінням скоротились вдвічі. Використання даного методу дає можливість розробляти помножувачі у полях Галуа $GF(p^m)$ з великим порядком.

Ключові слова: захист інформації, поля Галуа $GF(p^m)$, Vivado HLS, VHDL, класичний двокроковий алгоритм, типи з визначеною точністю

Постановка проблеми. Розробка вбудованих комп'ютерних систем нерозривно пов'язана з розвитком елементної бази, на якій виконується реалізація програми. Від ефективності використання апаратних засобів залежить швидкість реалізації алгоритмів та їх ефективність [1]. Переважно для програмування складних вбудованих систем використовуються програмовані логічні інтегральні схеми (ПЛІС) [2]. У зв'язку з цим виникає необхідність у розробці механізмів для забезпечення безпеки ПЛІС [3]. Це можна досягти шляхом виконання операцій над елементами скінченних полів [4].

Реалізація вбудованих комп'ютерних систем на ПЛІС, як правило, включає в себе написання VHDL (англ. VHSIC (Very high speed integrated circuits) Hardware Description Language) коду на рівні регістрових передач (RTL – register transfer level), який потім синтезується в логічні блоки [5]. Як альтернативний варіант розробки використовується середовище Vivado HLS [6]. Засоби високорівневого синтезу (HLS – high level synthesis) підвищують рівень абстракціїї від RTL до алгоритмічного, що дозволяє оптимізувати часові та апаратні затрати [7].

Аналіз останніх досліджень і публікацій. У роботах [8] та [9] проводився підрахунок структурної складності помножувачів у полях Галуа $GF(p^m)$, де p - характеритика поля Галуа, m- розрядність поля. Максимальний порядок поля Галуа при якому були наведені обчислення є 233. При побудові помножувача в полях Галуа $GF(p^m)$ за допомогою засобів мови VHDL потрібні значні часові та апаратні затрати [10]. Тому для оптимізаціїї апаратних витрат виникає необхідність у пошуку інших методів моделювання.

У вищезгаданих роботах [3, 4, 8, 9, 10] у зв'язку з великою апаратною складністю, не вдалося провести дослідження для полів Галуа $GF(p^m)$ з великим порядком. Тому пошук засобів для оптимізації апаратних ресурсів є актуальною задачею.

Метою статті є зменшення апаратної складності помножувача в полях Галуа $GF(p^m)$ за допомогою засобів Vivado HLS.

Середовище Vivado HLS. Vivado HLS є інструментом Xilinx для перетворення програмного коду в RTLімплементацію, яка в свою чергу синтезується у Xilinx ПЛІС [11]. На рис. 1 наведно принцип роботи Vivado HLS. Код може бути написаний на таких мовах програмування як: C, C++, SystemC. Це надає додатковий рівень абстракціїї до традиційного RTL кодування. Основна причина чому відбувся перехід з одного абстрактного рівня на інший є те, що за допомогою Vivado HLS набагато легше управляти складністю контрукцій.

Синтез високого рівня (HLS – High Level Synthesis) поєднує апаратну і програмну частини. А саме [12]:

– дозволяє за допомогою високого абстрактоного рівня створювати високопродуктивне апаратне обладнання;

– за допомогою нового методу компіляції надає розробникам програмного забезпечення можливість прискорити виконання алгоритмів. ПЛІС надають паралельну архітектуру з перевагами у продуктивності, ціні і потужності у порівянні з традиційними процесорами

Ці переваги досягаються за рахунок наступних чинників [5]:

– розробка алгоритмів на С-рівні дозволяє абстрагуватись від деталей імплементації;

– верифікація на С-рівні надає можливіть перевіряти функціональну правильність алгоритмів швидше ніж відомі апаратні мови;

– контроль процесу синтезу за допомогою оптимізаційного процесу дозволяє створювати високопродуктивні апаратні реалізації;

– HLS дизайн дає змогу використовувати С код на інших ПЛІС.

Відмінності синтаксису Vivado HLS у порівнянні з мовою С. Vivado HLS підтримує широкий діапазон можливостей мови С, однак деякі конструкції не можуть бути синтезовані або спричиняють помилки під час наступних етапів розробки [7]. Синтаксичні конструкції мови С, які не підтримуються у Vivado [6]:

– *системні виклики*

Системні виклики не можуть синтезуватися, тому що вони є діями, які пов'язані з виконанням деяких завдань від операційної системи, в яких працює програма С. Vivado HLS ігнорує часто використовувані системні виклики, які відображають тільки дані і не мають вплив на виконання алгоритму. Такі як: printf() і fprintf(stdout). В загальному виклики до системи не можуть бути синтезовані і повинні бути видалені перед синтезом. Це стосується таких викликів як: getc(), time(), sleep(). Всі вони містять виклики до

операційної системи. Також підчас синтезу у Vivado HLS можуть використовуватись macro _SYNTHESIS_. Код який є написаний у macro _SYNTHESIS не піддається синтезу.

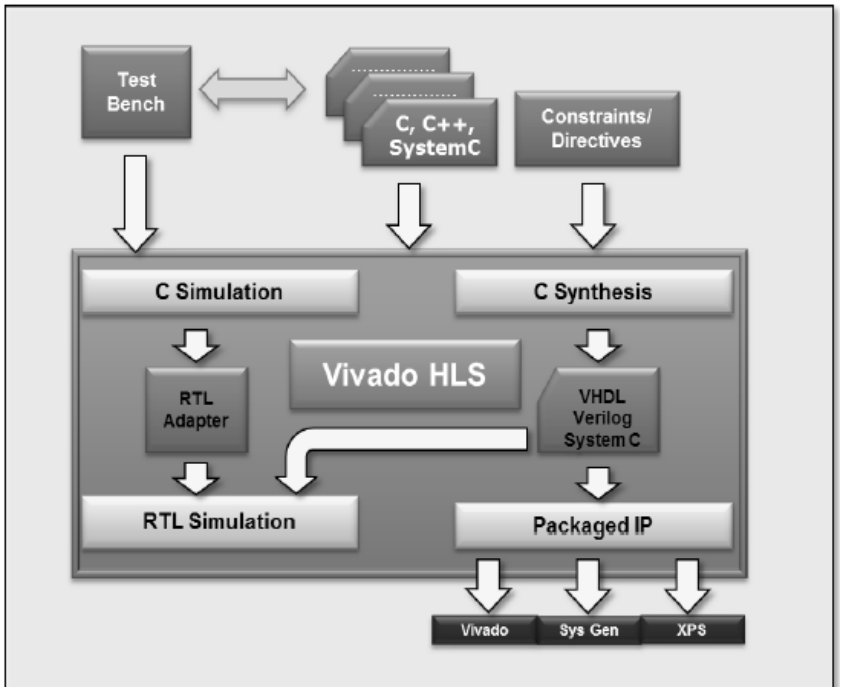

Рисунок 1 – Принцип роботи Vivado HLS

– *Динамічне використання пам'яті*

Будь-які системні виклики, які керують розподілом пам'яті в системі повинні бути повністю автономними та мати чітко визначений розподіл ресурсів. Наприклад функції malloc(), alloc(), і free() використовують ресурси, які існують в пам'яті операційної системи та створюються підчас виконання програми. Всі такі функції перед синтезом повинні бути замінені на еквівалентні з фіксованим розподілом пам'яті. Також Vivado HLS не підтримує C++ об'єкти які динамічно створюються і знищуються. Це стосується динамічного поліморфізму та динамічних віртуальних функцій.

– *Обмеження щодо вказівників*

Vivado HLS не підтримує конструкції де є звернення вказівник на вказівник. Також не синтезуються масиви вказівників, які звертаються на вказівник.

– *Рекурсивні функції*

Рекурсивні функції не можуть бути синтезовані у Vivado HLS

– *Стандартні бібліотеки шаблонів*

Багато стандартних шаблонних функцій мови С містять в собі рекурсивні виклики або динамічне виділення пам'яті. Через це вони не можуть бути синтезовані. Щоб уникнути цієї проблеми створюються подібні локальні функції, в яких відбувається заміна заборонених викликів на допустимі.

Тому, для того, щоб синтез відбувся успішно повинні бути дотримані наступні вимоги [12]:

1. Жодна функціональність не повинна використовувати системні виклики до операційної системи.
2. С конструкції повинні мати фіксований розмір.
3. Реалізація конструкцій повинна бути однозначною.

Типи з визначеною точністю (Arbitrary Precision Types). Типи даних, які використовуються в мові С, мають 8-бітні межі і мають фіксовану довжину (8, 16, 32, 64 біти). Тобто, для реалізації помножувача який займає 17 біт програмно використовуються 32 біти. З метою економії апаратних ресурсів у Vivado HLS використовуються *типи з визначеною точністю(arbitrary precision types)* [11].

Відомо два види типів з визначеною точністю [7]:

1. Цілі типи з визначеною точністю (Arbitrary Precision Integer Types).
2. Визначені типи з фіксованою комою(Arbitrary Precision Fixed Point Types).

Переваги використання цілих типів з визначеною точністю [5]:

– краща якість апаратного забезпечення. За рахунок визначених типів відбувається менше використання апаратних ресурсів;

– точний С аналіз та моделювання. Чітко визначені типи дозволяють провести моделювання використовуючи точні величини елементів. Це допомагає перевірити функціональність алгоритму перед синтезом.

Переваги використання визначених типів з фіксованою комою [12]:

– можливість представлення дробових чисел;

– вирівнювання значень після коми (коли змінна має різну кількість біт до, та після коми, використовується вирівнювання значення після коми);

– забезпечення заокруглення значень;

– керування процесом переповнення (коли результат є більшим за число бітів, його можна по різному представити).

Опис універсального помножувача у полях Галуа з використанням класичного двох-крокового алгоритму. На рис. 2 зображено структурну схему універсального помножувача в полях Галуа $GF(p^m)$ з використанням класичного двох-крокового алгоритму [9].

Помножувач має наступні вхідні параметри: вхідне значення A, вхідне значення B, основа поля Галуа p, степінь поля, нескорочуваний поліном.

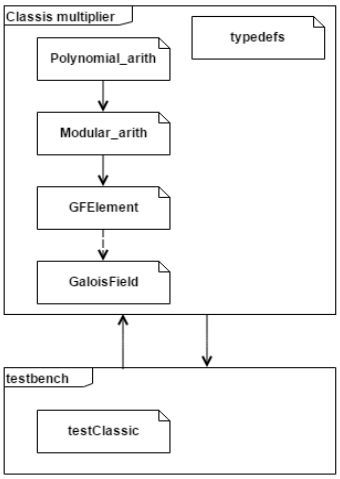

Рисунок 2 – UML діаграму компонентів універсального помножувача в полях Галуа $GF(p^m)$ з використанням класичного двох-крокового алгоритму

Розроблений у роботі універсальний помножувач складається з таких компонентів:

1. *GaloisField* – цей компонент визначає поле Галуа $GF(p^m)$, його характеристику p і степінь *m*. Також зберігає нескорочуваний поліном *p*. Атрибути *GaloisField* задаються спочатку і не можуть бути змінені. Також всі значення мають бути введені вірно, тобто *p* – просте число, *m* – відмінне від нуля, позитивне ціле число. Поліном має бути нескорочуваним. В програмі не передбачено перевірки на невірні значення.

2. *GFElement* – компонент який оперує елементами поля Галуа. Містить в собі поле *GaloisField*. У компоненті визначені всі операції, які можуть здійснюватись над полем *GaloisField*. Це перевизначенні оператори додавання, віднімання, множення, ділення, оператори порівняння більше/менше, оператори вводу/виводу. Також передбачено операцію знаходження оберненого елемента.

3. *Modular_arithm* – компонент містить операції які необхідні для виконання побітових операцій над полем Галуа.

4. *Polynomial_arith* – містить основні операції на полем Галуа: додавання, віднімання, множення (двокроковим алгоритмом) та допоміжні операції копіювання, встановлення нульового значення та звертання за індексом.

5. *typedfs* – містить визначення типів даних.

6. *testClassic* – основний блок виконання з тестовими вхідними параметрами.

Результати досліджень. Характеристики комп'ютера на якому проводились дослідження: Processor Intel(R) Core(TM) i5-2500 CPU @3.30GHz, RAM 8.00 GB, OS Windows 10 64-bit, середовище Xilinx Vivado HLS Дослідження проводились у на кристалі [13].

Для реалізаціїї помножувача в полях Галуа $GF(p^m)$ було створено проект в Vivado HLS, на мові C++ написано програму, проведено C Simulation та C Synthesis. Дослідження проводились на помножувачі GF(2^{251}).

Класичний двокроковий помножувач у полях Галуа $GF(p^m)$ був написаний згідно стандартних правил мови C++ і з врахуванням вимог середовища Vivado HLS.

На рис. 3 зображено вмістиме заголовочного файлу typedef.h, що містить визначення створених типів даних для класичного двокрокового помножувача у полях Галуа $GF(p^m)$. Окремо є визначенні типи: для характеристики p – це DimType, для розрядності поля m – ModType, для результату Fint. Оскільки всі дослідження проводились над додатними числами для реалізації множення був обраний тип *unsigned long*.

*typedefs.h

```
#ifndef TYPEDEFS_H
#define TYPEDEFS_H
#include <ap_int.h>
namespace shk_galoiscpp
{

    typedef unsigned long long Fint;
    typedef  unsigned long DimType;
    typedef  unsigned long ModType;

}
#endif
```

Рисунок 3 – Створені типи даних для класичного двокрокового помножувача у полях Галуа $GF(2^{251})$

В результаті синтезу було згенеровано звіт (Synthesis report) для класичного двокрокового помножувача в полях Галуа $GF(2^{251})$.Оскільки метою даної роботи є оптимізація класичного двокрокового помножувача в полях Галуа $GF(2^{251})$, на рис. 4 зображені апаратні витрати, які необхідні для реалізаціїї помножувача при використанні типу *unsigned long* у середовищі Vivado HLS.

На рис. 4 зображені такі дані:

- LUTs (Look-Up Table) – кількість найпростіших логічних елементів;
- FFs (Flip-Flop) – тригери з динамічним і потенційним управлінням;
- DSP48E (Digital Signal Processor) процесор цифрової обробки сигналів;
- BRAM_18K – (Blok Random Access Memory) – конфігурований модуль пам'яті.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	524
FIFO	-	-	-	-
Instance	-	4	2419	3209
Memory	7	-	0	0
Multiplexer	-	-	-	332
Register	-	-	730	-
ShiftMemory	-	-	-	-
Total	7	4	3149	4065
Available	530	400	157200	78600
Utilization (%)	1	1	2	5

Рисунок 4 – Загальні апаратні витрати класичного двокрокового помножувача в полях Галуа $GF(2^{251})$

Як було зазначено вище використання *arbitrary precision* типів дає можливість зменшити апаратні витрати. Спробуємо це перевірити замінивши типи даних у заголовочному файлі *typedef.h* (див. рис.5) Для оптимізації помножувача було прийнято рішення використати *arbitrary precision* типи. Для цього в файл *typedef.h* були внесені зміни.

*typedefs.h

```
#ifndef TYPEDEFS_H
#define TYPEDEFS_H
#include <ap_int.h>
namespace shk_galoiscpp
{

    typedef short Fint;
    typedef  ap_int<10> DimType;
    typedef  ap_int<3> ModType;

}
#endif
```

Рисунок 5 – Створені типи даних для класичного двокрокового помножувача у полях Галуа $GF(2^{251})$ після використання arbitrary precision типів

Для оптимізовного класичного двокрокового помножувача в полях Галуа $GF(2^{251})$ було згенеровано Synthesis report. На рис. 6 зображенні загальні апаратні витрати після оптимізаціїї класичного двокрокового помножувача в полях Галуа GF (2^{251}).

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	219
FIFO	-	-	-	-
Instance	1	4	816	913
Memory	7	-	0	0
Multiplexer	-	-	-	158
Register	-	-	448	-
ShiftMemory	-	-	-	-
Total	8	4	1264	1290
Available	530	400	157200	78600
Utilization (%)	1	1	~0	1

Рисунок 6 – Загальні апаратні витрати після оптимізаціїї класичного двокрокового помножувача в полях Галуа $GF\ (2^{251})$

В табл. 1 наведено порівняння апаратних і часових витрат до та після оптимізаціїї класичного двокрокового помножувача в полях Галуа $GF(2^{251})$.

Таблиця 1 – Апаратні витрати помножувача при оптимізації

Витрати	Перед оптимізацією	Після оптимізації	Space Availability
LUT	4065	1290	78600
FF	3149	1264	157200
BRAM	7	8	530
DSP48	4	4	400
Часові витрати, сек	52	48	

Як видно з табл. 1, після використання arbitrary precision типів кількість LUT зменшилась в 3 рази та кількість FF елементів зменшилась 2 два рази. Також зменшився загальний час виконання симуляції та синтезу.

Висновки. У даній роботі, було оптимізовано помножувач у полях Галуа $GF(p^m)$ на основі класичного двох крокового алгоритму, за допомогою засобів Vivado HLS на мові C++. Це дало змогу зменшити апаратні і часові витрати. У ході роботи було проведено синтез та симуляцію помножувача до та після оптимізації. Дослідження проводились на класичному двокроковому помножувачі в полях Галуа $GF(2^{251})$. На основі результатів синтезу, було порівняно апаратні і часові витрати двох помножувачів. Апаратні і часові витрати оптимізованого помножувача є менші, а саме кількість найпростіших логічних елементів (LUT) зменшилась в 3 рази та кількість тригерів з динамічним і потенційним управлінням (FF) зменшилась 2 два рази, також скоротився загальний час виконання синтезу та симуляції. Використання даного підходу дає можливість розробляти помножувачі у полях Галуа $GF(p^m)$ з великим порядком.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] W. Tan, and L. Yip, "Hardware implementation of genetic algorithms using FPGA", in *Proc. 47th IEEE International midwest symposium. Circuits and Systems*, Hiroshima, Japan, 2004, pp. 549-552.
doi: 10.1109/MWSCAS.2004.1354049.

[2] J. Rose, et al., "The VTR Project: Architecture and CAD for FPGAs from Verilog to routing", in *Proc. 20th ACM/SIGDA International symposium. Field-Programmable gate arrays*, Monterey, California, USA, February 2012, pp. 77-86.
doi: 10.1145/2145694.2145708.

[3] P. Kitsos, G. Theodoridis, and O. Koufopavlou, "An efficient reconfigurable multiplier architecture for Galois field $GF(2^m)$", *Microelectronics Journal*, vol. 34, iss. 10, pp.975-980, October 2003.
doi: 10.1016/S0026-2692(03)00172-1.

[4] Y. Li, G. Chen, and X. Xie, "Low complexity bit-parallel $GF(2^m)$ multiplier for all-one polynomials", *IACR Cryptology ePrint Archive*, pp. 414, 2012.

[5] G. Baguma "High Level Synthesis of FPGA-Based Digital Filters", M.S. thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2014.

[6] M. Zwagerman, "High Level Synthesis, a Use Case Comparison with Hardware Description Language", M.S. thesis, Grand Valley State University ScholarWorks, Allendale, Michigan, USA, 2015.

[7] S. Brad, "Tincr. Integrating Custom CAD Tool Frameworks with the Xilinx Vivado Design Suite", M.S. thesis, Brigham Young University, Provo, Utah, USA, 2014.

[8] Ю.З. Шологон, "Оцінювання структурної складності помножувачів полів Галуа на основі елементарних перетворювачів", *Вісник Національного університету "Львівська політехніка": Комп'ютерні системи та мережі*, № 806, с. 290-296, 2014.

[9] О.З. Шологон, "Обчислення структурної складності помножувачів у поліноміальному базисі елементів полів Галуа *GF(2m)*", *Вісник Національного університету "Львівська політехніка": Комп'ютерні системи та мережі*, № 806, с. 284-289, 2014.

[10] В.С. Глухов, Р. Еліас, та А.О. Мельник, "Особливості реалізації на ПЛІС секційних помножувачів елементів полів Галуа *GF(2^m)* з надвеликим степенем", *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*, № 12, с.103-106, 2013.

[11] "Xilinx. Vivado design suite user guide high-level synthesis", Xilinx, Inc., San Jose, California, USA, Tech. rep. (v2013.4), Dec. 2013.

[12] "Xilinx. Vivado design suite user guide high-level synthesis", Xilinx, Inc., San Jose, California, USA, Tech. rep. (v2014.1), Xilinx, Inc., May 2014.

[13] "Zynq-7000 All Programmable SoC PCB Design Guide", Xilinx, Inc., San Jose, California, USA, UG933 (v1.12), Sept. 2016.

Стаття надійшла до редакції 05.09.2016.

REFERENCES

[1] W. Tan, and L. Yip, "Hardware implementation of genetic algorithms using FPGA", in *Proc. 47th IEEE International midwest symposium. Circuits and Systems*, Hiroshima, Japan, 2004, pp. 549-552.
doi: 10.1109/MWSCAS.2004.1354049.

[2] J. Rose, et al., "The VTR Project: Architecture and CAD for FPGAs from Verilog to routing", in *Proc. 20th ACM/SIGDA International symposium. Field-Programmable gate arrays*, Monterey, California, USA, February 2012, pp. 77-86.
doi: 10.1145/2145694.2145708.

[3] P. Kitsos, G. Theodoridis, and O. Koufopavlou, "An efficient reconfigurable multiplier architecture for Galois field *GF(2^m)*", *Microelectronics Journal*, vol. 34, iss. 10, pp.975-980, October 2003.
doi: 10.1016/S0026-2692(03)00172-1.

[4] Y. Li, G. Chen, and X. Xie, "Low complexity bit-parallel *GF(2^m)* multiplier for all-one polynomials", *IACR Cryptology ePrint Archive*, pp. 414, 2012.

[5] G. Baguma "High Level Synthesis of FPGA-Based Digital Filters", M.S. thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2014.

[6] M. Zwagerman, "High Level Synthesis, a Use Case Comparison with Hardware Description Language", M.S. thesis, Grand Valley State University ScholarWorks, Allendale, Michigan, USA, 2015.

[7] S. Brad, "Tincr. Integrating Custom CAD Tool Frameworks with the Xilinx Vivado Design Suite", M.S. thesis, Brigham Young University, Provo, Utah, USA, 2014.

[8] Y. Sholohon, "Evaluation of structural complexity Galois field multipliers based on the elementary transducers", *Proceedings of the national university "Lviv Polytechnic": Computer systems and networks*, no. 806, pp. 290-296, 2014.

[9] O. Sholohon, "Structural Complexity of Galois Field *GF(2^m)* Elements Multipliers in Polynomial Basis Calculation", *Proceedings of the national university "Lviv Polytechnic": Computer systems and networks*, no. 806, pp. 284-289, 2014.

[10] V.S. Hlukhov, R. Elias, ta A.O. Melnyk, "Features of the FPGA-based Galois Field GF(2m) Elements Sectional Multipliers with Extra Large Exponent", *Computer-integrated technologies: education, science and industry*, no. 12, pp. 103-106, 2013.

[11] "Xilinx. Vivado design suite user guide high-level synthesis", Xilinx, Inc., San Jose, California, USA, Tech. rep. (v2013.4), Dec. 2013.

[12] "Xilinx. Vivado design suite user guide high-level synthesis", Xilinx, Inc., San Jose, California, USA, Tech. rep. (v2014.1), Xilinx, Inc., May 2014.

[13] "Zynq-7000 All Programmable SoC PCB Design Guide", Xilinx, Inc., San Jose, California, USA, UG933 (v1.12), Sept. 2016.

ШОЛОГОН ОЛЬГА,
ШОЛОГОН ЮЛИЯ

УМЕНШЕНИЕ ОБЩЕГО КОЛИЧЕСТВА ЛОГИЧЕСКИХ ЕЛЕМЕНТОВ У КЛАССИЧЕСКОМ ДВУХШАГОВОМ УМНОЖИТЕЛИ В ПОЛЯХ ГАЛУА СРЕДСТВАМИ VIVADO HLS

В Украине практически все реализации защиты информации являются программными, основным недостатком которых является недостаточная устойчивость к взлому. Поэтому для увеличения надежности реализации защиты информации возникает необходимость в создании аппаратных средств для выполнения операций над элементами конечных полей. Одной из возможностей является реализация на ПЛИС. Как правило, умножители в полях Галуа *GF(p^m)* осуществляются с помощью средств языка VHDL. Основным недостатком такого подхода является значительные временные и аппаратные

затраты. В данной статье предлагается разрабатывать умножитель в полях Галуа $GF(p^m)$ с помощью среды Vivado HLS. В работе рассмотрен метод оптимизации при котором использовались типы с определенной точностью. В результате исследований было доказано эффективность использования среды Vivado HLS по сравнению со средствами VHDL. Количество простейших логических элементов было снижено в 3 раза, а также количество триггеров с динамическим и потенциальным управлением сократилось вдвое. Использование данного метода позволяет разрабатывать умножители в полях Галуа $GF(p^m)$ с большим порядком.

Ключевые слова: защита информации, поля Галуа $GF(p^m)$, Vivado HLS, VHDL, классический двухшаговый алгоритм, типы с определенной точностью

OLHA SHOLOHON,
YULIIA SHOLOHON

DECREASING THE TOTAL NUMBER OF LOGIC ELEMENTS IN THE CLASSIS TWO-STEP MULTIPLIER WITH A HELP OF VIVADO HLS

In Ukraine, almost all implementation of information security have software realization. The main disadvantage of this approach is the insufficient resistance to fracture. So in order to increase the information security is needed to develop hardware devices which use elements of finite fields. One of appropriate realizations is implementation of field-programmable gate array (FPGA). Advantages of their usage is possibility of configuration by a customer or a designer after manufacturing. Typically, multipliers in the Galois field $GF(p^m)$ are built with the help of language VHDL. The main drawback of this approach are significant time and hardware costs. In this paper is proposed to build classic two step multiplier in Galois field $GF(p^m)$ with the help of Vivado HLS tool. The Xilinx High-Level Synthesis software Vivado HLS transforms a C specification into a Register Transfer Level (RTL) implementation that synthesizes into a Xilinx Field Programmable Gate Array (FPGA). In order to optimize classic two step multiplier in Galois field $GF(p^m)$ was used arbitrary precision types. Arbitrary precision data types help controlling both the area and resource utilization. As a result were proven better usage of Vivado HLS environment in comparison with VHDL tools. The number of the simplest logic elements were reduced three times, the number of triggers with dynamic management were reduced by half, also was reduced total execution time. Usage of this method makes it possible to develop multipliers in the Galois field $GF(p^m)$ with a large order.

Keywords: information security, Galois field $GF(p^m)$, Vivado HLS, VHDL, classic two-step algorithm, arbitrary precision types

Ольга Зіновіївна Шологон, аспірант, асистент кафедри електронних обчислювальних машин Національного університету “Львівська політехніка”, Львів, Україна.
E-mail: olha-sholohon@ukr.net.
Юлія Зіновіївна Шологон, аспірант, асистент кафедри електронних обчислювальних машин Національного університету “Львівська політехніка”, Львів, Україна.
E-mail: yulia.sholohon@gmail.com.

Ольга Зиновьевна Шологон, аспирант, ассистент кафедры электронных вычислительных машин Национального университета “Львовская политехника”, Львов, Украина.
Юлія Зиновьевна Шологон, аспирант, ассистент кафедры электронных вычислительных машин Национального университета “Львовская политехника”, Львов, Украина.

Olha Sholohon, graduate student, assistant of computer engineering academic department, National University “Lviv Polytechnic”, Lviv, Ukraine.
Yuliia Sholohon, graduate student, assistant of computer engineering academic department, National University “Lviv Polytechnic”, Lviv, Ukraine.