УДК 004.25, 004.383

В. А. Мельник, І. І. Лопіт
Національний університет “Львівська політехніка”,
кафедра безпеки інформаційних технологій

ОПТИМІЗАЦІЯ ВІДОБРАЖЕННЯ ПАМ’ЯТІ ПРОГРАМНИХ МОДЕЛЕЙ СПЕЦІАЛІЗОВАНИХ ПРОЦЕСОРІВ В АРХІТЕКТУРУ ПЛІС

Порушено проблеми ефективного відображення в архітектуру ПЛІС пристроїв пам’яті з довільним доступом, які входять до складу програмних моделей спеціалізованих процесорів. На основі аналізу архітектури ПЛІС запропоновано підходи до ефективного відображення пам’яті, розроблено методи, алгоритми та програмні засоби.

Ключові слова: пам'ять з довільним доступом, ПЛІС, спеціалізовані процесори.

OPTIMIZATION OF MEMORY MAPPING INTO FPGA ARCHITECTURE FOR SPECIALIZED PROCESSORS' PROGRAM MODELS

The article embraces the issues of effective mapping into the FPGA architecture of the random access memory devices that are parts of application-specific processors’ program models. According to the analysis of modern FPGA architecture, the approaches of effective memory mapping are suggested; methods, algorithms and software means are developed.

Key words: random access memory, FPGA, application-specific processors.

Вступ

За сучасного стану комп’ютерної елементної бази підвищують продуктивність комп’ютерних систем переважно екстенсивним методом, тобто збільшенням кількості ядер універсальних процесорів та підвищенням частоти їх роботи. Разом з тим підхід застосування універсальних процесорів для досягнення високих показників продуктивності має принципові недоліки, головними з яких є висока споживана потужність і низька ефективність використання обладнання. Для уникнення цих недоліків створюють комп’ютерні системи із спеціалізованими процесорами, однак вони є ефективними лише для вузьких класів алгоритмів, а їх побудова потребує значних зусиль і ресурсів. Один з варіантів вирішення цієї проблеми полягає в генеруванні програмних моделей спеціалізованих процесорів з високорівневого подання алгоритму їх роботи та їх реалізації в програмовних логічних інтегральних схемах (ПЛІС). У результаті отримують спеціалізовані процесори, які використовують як прискорювачі комп’ютерних систем і функції яких можна змінювати реконфігуруванням ПЛІС і створенням у ній іншого СП.

Яскравим представником засобів цього класу є Chameleon від компанії Intron [1]. Він на основі мови високого рівня ANSI C генерує програмну модель спеціалізованого процесора (ПМСП) мовою VHDL [2]. Проте ці підходи мають свої недоліки, ринок ПЛІС швидко розвивається, їх архітектура постійно змінюється. Внаслідок низької пристосованості (генератор ПМСП не використовує всіх переваг архітектури ПЛІС) або вузької спрямованості (специфікація під конкретне сімейство або модель без можливості портування на інші) ці засоби втрачають ефективність. Згенерована модель може не відповідати сучасним вимогам ринку ПЛІС, тому актуальним є завдання оптимізації відображення компонентів спеціалізованих процесорів в архітектуру ПЛІС.

Одним з найпоширеніших і найбільш ресурсоємних компонентів ПМСП є пам'ять з довільним доступом (ПДД).

Її реалізація в ПЛІС можлива з використанням конфігуровних логічних комірок – базових компонентів ПЛІС або з використанням блоків вбудованої пам’яті. Разом з тим внаслідок відмінностей в описах пам’яті ПМСП та наявності різних типів вбудованої пам’яті в ПЛІС різних виробників і серій існує проблема ефективного відображення пам’яті ПМСП в архітектуру ПЛІС. Аналіз і вирішення цієї проблеми є предметом дослідження авторів.

Аналіз досліджень та публікацій

У роботі [3] подано загальні характеристики та класифікацію пам’яті з довільним доступом. У матеріалах [4] детально описано архітектуру та характеристики основних елементів ПЛІС сімейства Stratix IV, яку використано як основу для аналізу ефективних експериментальних досліджень для отримання кількісних оцінок результатів дослідження згідно із інструкціями, описаними в роботі [5]. Задача розбиття площини первинної пам’яті на частини схожа на задачу обчислення площі криволінійної трапеції методом квадратів, яку зокрема описано в [6]. У джерелі [7] подано шаблони та правила написання VHDL описів компонентів для ПЛІС сімейства Stratix IV.

Постановка проблеми

Незважаючи на наявність вбудованих блоків пам’яті, під час їх використання виникають такі проблеми:

1. САПР, що використовується для логічного синтезу ПМСП, може не розпізнати пристрій пам’яті, створений користувачем, і синтезувати його нераціонально, без використання вбудованої пам’яті.
2. САПР може розпізнати створений користувачем пристрій пам’яті, але нераціонально використати вбудовану пам’ять для його синтезу, розмістивши його в блок пам’яті більшої ємності і залишивши частину цього блоку невикористаною. Варіант розбиття первинного пристрою пам’яті на менші та їхні з’єднання за допомогою комутаційної логіки буде відкинуто, хоча це дозволило б раціональніше використовувати обмежену кількість вбудованих блоків.
3. Для великих блоків пам’яті розміром, не кратним до розміру блоків вбудованої пам’яті, САПР може не ефективно розбити блок пам’яті на підблоки.

Для вирішення цих проблем постає завдання розроблення методів розпізнавання блоків пам’яті та їх оптимального відображення в архітектуру ПЛІС, а також створення програмних засобів автоматичного виконання такого відображення на основі цих методів.

Проблеми відображення програмних моделей пам’яті в архітектуру ПЛІС

Пам’ять з довільним доступом – це один з найпоширеніших електронних компонентів сучасної обчислювальної техніки, призначений для зберігання даних. Ця пам’ять дає можливість у кожному такті звернутися до її довільної комірки, вказавши її адресу, щоб записати до неї або зчитати з неї число (дані) [3].

Своєю чергою, за здатністю до зміни вмісту ПДД поділяють на:

- Оперативний запам’ятовуючий пристрій (ОЗП, анг. RAM – Random Access Memory), який дає можливість модифікувати вміст;
- Постійний запам’ятовуючий пристрій (ПЗП, анг. ROM – Read-Only Memory), який призначений для постійного зберігання даних, і або не дозволяє їх зміни взагалі, або передбачає застосування для цього спеціального обладнання.

Основною проблемою реалізації компонентів ПДД є нераціональне використання ресурсів ПЛІС. Наприклад, у табл. 1 та 2 наведено результати генерування різноманітних блоків пам’яті, які є складовими ПМСП алгоритму швидкого перетворення Фур’є, отриманого за допомогою засобів автоматизованого високорівневого проектування ПМСП Chameleon. Для синтезу використано ПЛІС Stratix IV та САПР Quartus II версії 9 компанії Altera.

Для кращого розуміння задачі відображення програмних моделей пам’яті в архітектуру ПЛІС необхідно визначити принцип, за яким пам’ять синтезує в ПЛІС. Розглянемо, як це реалізовано в ПЛІС сімейства Stratix IV фірми Altera. ПЛІС цього сімейства являє собою матрицю однотипних

логічних комірок – Adaptive Look-Up Table (ALUT) [2]. Кожна з них складається з регістра і комбінаційної пари [2].

Таблиця 1

Результати генерування блоків ОЗП

№ з/п	Розрядність комірки, бітів	Кількість комірок	Ємність, бітів	Кількість регістрів	Кількість комбінаційних пар
1	18	1084	18428	19530	7651
2	18	651	11067	11736	4604
3	18	412	6970	7434	2967
4	18	91	1574	1656	597
5	18	30	510	558	193

Таблиця 2

Результати генерування блоків ПЗП

№ з/п	Розрядність комірки, бітів	Кількість комірок	Ємність, бітів	Кількість комбінаційних пар
1	190	51200	9728000	37116
2	270	25600	6912000	36774
3	1250	3200	4000000	57222
4	3555	1024	3640320	54684

Під час синтезу блоку ОЗП для зберігання кожного біта буде використано один регістр. Також буде згенеровано вхідну та вихідну комутаційну логіку. Якщо блок ОЗП має велику ємність, то затрати на комутаційну логіку будуть набагато менші, ніж на логіку для зберігання даних. Це підтверджено аналізом розподілу ресурсів для побудови комутаційної логіки в ПЛІС цієї архітектури. Так, для побудови мультиплексора 16-в-1 з шириною шини вхідних даних, що дорівнює 16 розрядів, потрібно 8 логічних комірок, а затрати обладнання збільшуються лінійно, пропорційно до $\log(LPM\_SIZE)$, де *LPM_SIZE* – кількість вхідних шин мультиплексора [8]. Отже, можна припустити, що для великих значень ємності пам’яті витрати на один біт дорівнюватимуть одному ALUT. Це є яскравим прикладом нераціонального використання архітектури ПЛІС. На відмінну від ОЗП, при побудові ПЗП використано лише комбінаційні пари ALUT, які витрачаються на побудову комутаційної логіки. Регістри в цьому випадку не потрібні – вони замінюються безпосереднім поданням констант на входи мультиплексора.

Проблема нераціонального використання логічних елементів є добре відомою – при конструюванні ПЛІС її вирішують додаванням блоків вбудованої пам’яті. Основні характеристики блоків вбудованої пам’яті ПЛІС Stratix IV наведено в табл. 3.

Таблиця 3

Основні характеристики блоків вбудованої пам’яті для ПЛІС сімейства Stratix IV фірми Altera

Тип блоку	Блок MLAB	Блок M9K	Блок M114K
Максимальна частота, МГц	600	600	540
Ємність, біт	640	9216	147456
Варіанти організації (кількість комірок х розрядність, бітів)	64 × 8 64 × 9 64 × 10 32 × 16 32 × 18 32 × 20	8K × 1 4K × 2 2K × 4 1K × 8 1K × 9 512 × 16 512 × 18 256 × 32 256 × 36	16K × 8 16K × 9 8K × 16 8K × 18 4K × 32 4K × 36 2K × 64 2K × 72

Методи розпізнавання блоків ПДД

Можна виділити два методи розпізнавання опису пристрою пам'яті мовою VHDL:

- Метод формального запису. Згідно з цим методом, користувач повинен перерахувати список сигналів, їх розрядність, і вказати тип пам'яті. За своєю суттю, користувач безпосередньо вказує, який компонент він бажає отримати.
- Метод використання взірців. Згідно з цим методом, розпізнавання здійснюється без втручання користувача на основі взірців опису пам'яті. З погляду архітектури, правильно компонент ПДД можна описати обмеженою кількістю способів – взірців (патернів, шаблонів). Програма-розпізнавач порівнює компонент поточного VHDL-файла з кожним із них, і на основі порівняння встановлює, який тип пам'яті необхідно використати. Наприклад: лінія сигналу синхронізації – це вхідний порт типу std_logic. Він обов'язково повинен бути присутнім у списку чутливості процесу, який описує пам'ять. Цей сигнал не повинен модифікуватись. Також він повинен викликатись функцією rising_edge або falling_edge. Якщо сигнал з цими атрибутами наявний в описі компонента, можна припустити, що це вхід синхронізації.

Метод формального запису доцільно використовувати тоді, коли є безпосередній доступ до генератора ПМСП. У результуючому файлі сумісно з VHDL-кодом генератор може створити формальні записи, які прискорять розпізнання компонентів і дозволять їх розпізнати безпомилково. На відмінну від методу формального запису, метод використання взірців передбачає аналіз всього проекту з метою пошуку описів компонентів пам'яті. Цей метод дає змогу розпізнавати компоненти незалежно від наявності доступу до генератора ПМСП. Основним його недоліком є складність. Також треба зважати на те, що компоненти пам'яті можна описати по-різному, і в цьому контексті використання методу формального запису забезпечує вищі швидкість і якість розпізнавання.

Алгоритм оптимізації структури блоків ПДД

Для ефективного відображення пам'яті програмних моделей спеціалізованих процесорів в ахрітектуру ПЛІС після розпізнання пристрою пам'яті необхідно оптимізувати його структуру, враховуючи особливості архітектури цільової ПЛІС. Цей крок можна здійснити безпосередньо засобами логічного синтезу (компілятором), однак, як показує досвід, не всіма засобами це можливо, а засоби, якими можна, не завжди високоефективні. Тому автори розробили алгоритм оптимізації структури блоків ПДД, який наведено нижче.

Вхідні дані:

1. Розрядність комірки початкового блоку пам'яті в бітах;
2. Кількість комірок початкового блоку пам'яті;
3. Варіанти організації блоків вбудованої пам'яті;
4. Критерій оптимізації.

Вихідні дані: VHDL-файли з оптимізованими архітектурно залежними описами блоків пам'яті.

Критерії оптимізації:

1. Мінімальні затрати ресурсів вбудованої пам'яті.
2. Мінімальні затрати логічних елементів ПЛІС.

Список основних величин, використаних в алгоритмі: W_b – розрядність комірки початкового блоку пам'яті в бітах; H_b – кількість комірок початкового блоку пам'яті; w – розрядність комірки блоку вбудованої пам'яті в бітах; h – кількість комірок блоку вбудованої пам'яті в бітах; Wo_b – ширина кінцевого пристрою пам'яті в блоках; Ho_b – висота кінцевого пристрою пам'яті в блоках; η – коефіцієнт збільшення ємності пам'яті внаслідок заміни на блоки вбудованої пам'яті.

Алгоритм оптимізації структури блоків ПДД передбачає виконання таких кроків:

1. Виконується генерація списку можливих конфігурацій блоків вбудованої пам'яті, отриманих на основі специфікації архітектури ПЛІС.

2. Для кожної з можливих конфігурацій обчислюються:

1.1. Кількість блоків вбудованої пам’яті, яка необхідна для забезпечення потрібної довжини оптимізованої матриці пам’яті, за формулою (1):

$$Wo_b = \lceil W_b / w \rceil \quad (1)$$

1.2. Кількість блоків вбудованої пам’яті, яка необхідна для забезпечення потрібної висоти оптимізованої матриці пам’яті, за формулою (2):

$$Ho_b = \lceil H_b / h \rceil \quad (2)$$

1.3. Коефіцієнт збільшення об’єму пам’яті за формулою (3):

$$\eta = (Wo_b \cdot Ho_b \cdot w \cdot h)/(W_b \cdot H_b) \quad (3)$$

1.4. Формується список конфігурацій з обчисленими значеннями Wo_b, Ho_b та η у довільному порядку.

3. Вибірка двох конфігурацій з початку списку.

4. Якщо критерієм оптимізації є мінімальні затрати ресурсів вбудованої пам’яті, то для кожної з конфігурацій, отриманих за пунктом 2, виконуються операції порівняння їх між собою на основі значень коефіцієнта, отриманого за формулою (3). Якщо значення не однакові, то конфігурація з меншим значенням отримує вищий пріоритет у списку, перехід до п.6. В іншому випадку – порівняння їх між собою на основі значень коефіцієнта, отриманого за формулою (2). Конфігурація з меншим значенням отримає вищий пріоритет у списку.

5. Якщо критерієм оптимізації є мінімальні затрати логічних елементів ПЛІС, то кожну з конфігурацій, отриманих з пункту 2, порівнюють між собою на основі значень коефіцієнта, отриманого за формулою (2). Якщо значення не рівні, то конфігурація з меншим значенням отримує вищий пріоритет в списку, перехід до п.6. В іншому випадку – порівняння їх між собою на основі значень коефіцієнта, отриманого за формулою (3). Конфігурація з меншим значенням отримає вищий пріоритет в списку.

6. Якщо не кінець списку, то перехід на одну позицію в списку і вибірка наступних двох конфігурацій і перехід до п.4. В іншому випадку – вибираємо як оптимальну останню в списку конфігурацію.

Отже, згідно із запропонованим алгоритмом, оптимізація виконується методом сортування, і після її завершення оптимальний варіант буде знаходитись в кінці списку конфігурацій. Сортування здійснюється за допомогою функцій порівняння (компараторів). Їх будова прямо залежить від критерію оптимізації.

Якщо критерієм оптимізації є мінімальне використання ресурсу вбудованої пам’яті, сортування відбуватиметься за коефіцієнтом η. Проте, якщо в декількох конфігураціях коефіцієнти рівні, доцільним буде вибрати той, у якого висота менша. Це аргументовано тим, що затрати обладнання на реалізацію пам'яті складаються з двох частин: затрати на регістри та на комутаційну логіку між ними. У випадку використання вбудованих блоків пам’яті затрати на регістри відсутні, проте, якщо кількість комірок у первинному блоці пам’яті є більшою ніж найбільша кількість комірок блоку вбудованої пам’яті, постає завдання поділу адресного простору на декілька підпросторів. Це здійснюють, розбиваючи первинний адресний простір на декілька підблоків та зв’язуючи їх за допомогою вхідної та вихідної комутаційної логіки (мультиплексора і демультиплексора). Оскільки кількість логічних елементів, які необхідні для побудови комутаційної логіки, зростає пропорційно до її розрядності, доцільним буде вибір тої конфігурації, в якої висота менша.

Функцію компаратора в псевдокоді можна записати так:

```
bool comparatorArea(left, right)
{
        if (Abs(left.etta - right.etta) < d_MAX)
        {
                return left.Hob > right.Hob;
        }
        return left.etta > right.etta;
}
```

де left, right – структури, які представляють дві конфігурації, Abs () – функція арифметичного модуля, etta – коефіцієнт збільшення обсягу пам'яті, d_MAX – допустиме значення для порівняння чисел із рухомою комою (дорівнює 0.001). Hob – висота кінцевого компонента пам'яті в блоках.

У випадку оптимізації за затратами логічних елементів ПЛІС необхідно вибрати конфігурацію з найменшим значенням коефіцієнта Ho_b. Функцію компаратора для такого випадку можна записати так:

```
bool comparatorLUT(left, right)
{
        if (left.Hob  == right.Hob)
        {
                return left.etta > right.etta;
        }
        return left.Hob > right.Hob;
}
```

де Hob – висота оптимізованої матриці в блоках.

На основі даних, отриманих в результаті розпізнавання блоків ПДД, та даних, отриманих у результаті виконання алгоритму оптимізації структури блоків ПДД, виконується генерація VHDL-файлів з оптимізованими архітектурно залежними описами блоків пам'яті.

Результати експериментальних досліджень

Щоби оцінити ефективність розробленого алгоритму оптимізації структури блоків ПДД, розроблено програмні засоби, які автоматично оптимізують структуру блоків ПДД та проведено експериментальні дослідження на ПЛІС сімейства Stratix IV компанії Altera. Синтез виконано з використанням САПР Quartus II версії 9. Як вхідні набори використано блоки ПДД, які входять до складу ПМСП алгоритму ШПФ, згенеровані системою високорівневого автоматизованого проектування ПМСП Chameleon. Результати наведено в табл. 4 і 5. У табл. 4 наведено результати синтезу у випадку оптимізації за затратами ресурсів вбудованої пам'яті. У табл. 5 – у випадку оптимізації за затратами логічних елементів ПЛІС.

Таблиця 4

Результати оптимізації за затратами ресурсів вбудованої пам'яті

№ з/п	Розрядність комірки, бітів	Кількість комірок	Ємність, бітів	Зростання об'єму пам'яті, %	Кількість комбінаційних пар
1	18	32	576	3,22	0
2	18	92	1656	4,34	75
3	18	416	7488	0,72	323
4	18	672	12096	3,067	529
5	18	1088	19584	0,27	438

Таблиця 5

Результати оптимізації за затратами логічних елементів ПЛІС

№ з/п	Розрядність комірки, бітів	Кількість комірок	Ємність, бітів	Зростання об'єму пам'яті,%	Кількість комбінаційних пар
1	18	32	576	3,22	0
2	18	256	4068	394,6	0
3	18	512	9216	23,97	0
4	18	1024	18432	57	0
5	18	4096	73728	277	0

Як можна зауважити з таблиць, ці методи демонструють хороші результати. З використанням критерію оптимізації за обсягом вбудованої пам'яті застосування цих методів показує зменшення використання логічних елементів на декілька порядків, а в деяких випадках

зводить їх до нуля, при цьому ємність пам’яті зростає не більше ніж на 5 %. У випадку використання оптимізації за затратами логічних елементів цей метод дав змогу звести використання логічних елементів до нуля.

Висновок

Розглянуто проблеми розпізнавання та ефективного відображення компонентів спеціалізованих процесорів в архітектуру ПЛІС. У роботі проаналізовано повний цикл для оптимального відображення блоків ПДД, а саме: розпізнавання, оптимізація та генерування адаптованого до архітектури цільової ПЛІС VHDL-опису. В статті виокремлено методи розпізнавання компонентів ПДД та запропоновано алгоритм оптимізації структури блоків ПДД,. Алгоритм дає змогу оптимізувати структуру за затратами ресурсів вбудованої пам’яті та за затратами логічних елементів ПЛІС.

Розроблено програмні засоби, які автоматично виконують оптимізацію структури блоків ПДД, та проведено експериментальні дослідження, які показали зниження затрат обладнання логічних елементів ПЛІС на декілька порядків, а в деяких випадках – зведення їх до нуля. Отже, дані засоби оптимізації можна використовувати як проміжну ланку між генератором ПМСП і засобами логічного синтезу спеціалізованих процесорів.

1. Chameleon – the System-Level Design Solution. [Електронний ресурс] / – Режим доступу: http://intron-innovations.com/?p=sld_chame. 2.IEEE, Standard VHDL Language Reference Manual. Standard 1076-1993, New York, NY: IEEE, 1993. 3. Мельник А. О., Архітектура комп'ютера. Луцьк: Волинська обласна друкарня, 2008. — 470 с. 4. Quartus II Help v12.0. Adaptive Look-Up Table (ALUT) Definition. [Електронний ресурс] / – Режим доступу: http://quartushelp.altera.com/12.0/mergedProjects/reference/glossary/def_alut.htm *5. Stratix IV Device Handbook [Електронний ресурс] / – Режим доступу:* http://www.altera.com/literature/hb/stratix-iv/stx4_5v1.pdf *6. Мудров А.Е. Численные методы для ПЭВМ на языках Бейсик, Фортран и Паскаль. Томск: МП “РАСКО”, 1991. - 272 с. 7. Recommended HDL Coding Styles [Електронний ресурс] / – Режим доступу* http://www.altera.com/literature/hb/qts/qts_qii51007.pdf *8. lpm_mux, mux and busmux Megafunction [Електронний ресурс] / – Режим доступу http://quartushelp.altera.com/13.0/mergedProjects/hdl/mega/mega_file_lpm_mux_etc.htm*