Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA
Назва теми

КвРКІ. 190250.20.01.23 ПЗ
Шифр

Галузь знань 12 «Інформаційні технології»
Шифр, назва

Спеціальність 123 «Комп'ютерна інженерія»
Шифр, назва

Освітня програма «Комп'ютерна інженерія та програмування»
Назва

Виконав: студент IV курсу, група КІ2–20–1 _____ В.О. Шевченко
Підпис Ініціали, прізвище

Керівник _____ В.В. Яцків
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«12» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"___" _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Шевченку Владиславу Олеговичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA

Керівник проекту (роботи) Яцків В.В., професор кафедри КІІС

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 07.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області та постановка завдання

Методи перетворення даних в системи залишкових класів

Реалізація програмно-технічного модуля перетворення даних в систему залишкових класів на базі FPGA

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Функціональна схема та інтерфейс модуля перетворення

Структурна схема прямого перетворювача для СЗК(3, 5, 7) з паралельним обчисленням

Результати моделювання

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О., доцент кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_11_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	11.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – Дослідження предметної області та постановка завдання	01.03.2024	виконано
4	Робота над розділом 2 – Методи перетворення даних в системи залишкових класів	01.04.2024	виконано
5	Робота над розділом 3 – Реалізація програмно-технічного модуля перетворення даних в систему залишкових класів на базі FPGA	30.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	31.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ Шевченко В.О.
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Яцків В.В.
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ. 190250.20.01.23 ПЗ	Пояснювальна записка	58		
			Графічні матеріали			
2		КвРКІ. 190250.20.01.23 Е8	Функціональна схема та інтерфейс модуля перетворення	1		
3		КвРКІ. 190250.20.01.23 Е2	Структурна схема прямого перетворювача для СЗК(3, 5, 7) з паралельним обчисленням	1		
4		КвРКІ. 190250.20.01.23 Е4	Результати моделювання	1		

					КвРКІ. 190250.20.01.23 ВП			
Зм	Арк	№ докум	Підпис	Дата				
Розробив		Шевченко В.О.			Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA	Літера	Аркуш	Аркушів
Перевір.		Яцків В.В.				У	1	58
						ХНУ, КІ2-20-1		
Н. контр.		Засорнова І.О.						
Затв.		Говорущенко		12.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA.

Автор роботи: *Шевченко Владислав Олегович*.

Керівник роботи: *Яцків Василь Васильович*.

Пояснювальна записка: *58 с.*, 24 *рис.*, 3 *табл.*, 4 *дод.*, 49 *джерел*.

Графічна частина: 3 креслення.

МОДУЛЯ ПЕРЕТВОРЕННЯ ДАНИХ, СИСТЕМА ЗАЛИШКОВИХ КЛАСІВ

Мета кваліфікаційної роботи: розробка програмно-технічного модуля перетворення даних з двійкової системи в систему залишкових класів.

Сучасні вбудовані системи та додатки бездротового зв'язку вимагають енергоефективності та швидкого обчислення. Останнім часом було досягнуто значного прогресу у цьому напрямку в різних аспектах електронних систем, від рівня транзисторів до рівня архітектури. Однак операції на арифметичному рівні все ще недостатньо ефективні через довгі ланцюжки перенесення. Незважаючи на те, що багато методів комп'ютерної арифметики, наприклад обчислення з паралельним префіксом, було введено для зменшення затримки, перехід від традиційної позиційної системи числення з двома доповненнями до ефективної альтернативної системи числення може мати значний вплив на продуктивність обчислювальних систем.

Підпис студента  Дата 18.06.2024

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СЗК – система залишкових класів

НСЗК – надлишкова система залишкових класів

КТЗ – китайська теорема про залишки

RSD – Redundant Signed-Digit

ASIC – Application-Specific Integrated Circuit

FPGA – Field Programmable Gate Arrays

PLD – Programmable Logic Device

ВСТУП

Сучасні вбудовані системи та додатки бездротового зв'язку вимагають енергоефективності та швидкого обчислення [1, 2]. Останнім часом було досягнуто значного прогресу у цьому напрямку в різних аспектах електронних систем, від рівня транзисторів до рівня архітектури. Однак операції на арифметичному рівні все ще недостатньо ефективні через довгі ланцюжки перенесення. Незважаючи на те, що багато методів комп'ютерної арифметики, наприклад обчислення з паралельним префіксом, було введено для зменшення затримки, перехід від традиційної позиційної системи числення з двома доповненнями до ефективної альтернативної системи числення може мати значний вплив на продуктивність обчислювальних систем.

Система залишкових класів (СЗК) була найцікавішою та найважчою альтернативною системою числення в комп'ютерній арифметиці вже понад півстоліття [3 – 5]. ЇЇ головною важливою особливістю є здатність виконувати додавання, віднімання та множення без перенесення між залишками [6, 7]. Ця перевага робить мету дослідника комп'ютерної арифметики, тобто арифметику без переносів, досяжною. Однак існують певні труднощі для ефективної реалізації ділення, перетворення залишків у двійкову систему числення, виявлення знаків числа, порівняння чисел і виявлення переповнення. Таким чином, СЗК в основному використовується в програмах, де домінуючими операціями є додавання, віднімання та множення, таких як цифрова обробка сигналів [8] і криптографія [9, 10]. Крім того, RNS має багато інших цікавих властивостей, таких як виявлення/виправлення помилок і відмовостійкість [11], що робить її придатною для інших типів програм, а саме комп'ютерних мереж [12], арифметики ДНК [13] і хмарних сховищ [14]. За останні роки було досягнуто багато успіхів як у проектуванні, так і в додатках, націлених на СЗК [15].

Щоб спроектувати цифрову систему на основі СЗК, перш за все необхідно визначити динамічний діапазон, який потрібно охопити, тобто

діапазон чисел, які потрібно обробляти. Потім слід вибрати ефективний набір модулів відповідно до цього динамічного діапазону та типу цільових програм. Прямий перетворювач потрібен для інтерфейсу СЗК зі зваженим представленням чисел. Потім незалежні модулі арифметичних каналів паралельно виконують обчислення над залишками. Нарешті, зворотний конвертер перетворює отримане число СЗК у позиційне представлення, яке використовується звичайними цифровими системами.

На додаток до цих основних компонентів СЗК, деякі додаткові компоненти, такі як детектор знаків, компаратор величини та масштабування, можуть використовуватися в тракті даних СЗК відповідно до вимог програми [16].

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Основи системи залишкових класів

Система залишкових класів заснована на відношенні конгруенції, яке визначається наступним чином. Два цілі числа a і b називаються конгруентними за модулем m, якщо m точно ділить різницю a і b; для позначення цього пишуть $a \equiv b\ (mod\ m)$.

Так, наприклад, $10 \equiv 7(mod\ 3)$, $10 \equiv 4\ (mod\ 3)$, $10 \equiv 1\ (mod\ 3)$ і $10 \equiv -2\ (mod\ 3)$. Число m є модулем або основою, і ми припустимо, що його значення виключають одиницю, яка створює лише тривіальні конгруенції [15].

Якщо q і r є часткою та залишком, відповідно, від цілого ділення a на m, тобто $a = q \cdot m + r$, тоді, за визначенням, ми маємо $a \equiv r\ (mod\ m)$. Число r називається залишком a відносно m, і ми зазвичай позначаємо це $r = |a|_m$. Набір із m найменших значень, $\{0, 1, 2, \ldots, m - 1\}$, який може приймати залишок, називається множиною найменших додатних залишків за модулем m. Якщо не вказано інше, ми будемо вважати, що це єдині залишки, які використовуються.

Припустимо, що ми маємо набір $\{m_1, m_2, \ldots, m_N\}$ із N позитивних і попарно взаємно простих модулів. Нехай M – добуток модулів. Тоді кожне число $X < M$ має унікальне представлення в системі залишкових класів, яка є набором залишків $\{|X|_{m_i}: 1 \leq i \leq N\}$. Частковим доказом цього є наступне.

Припустимо, що X_1 і X_2 – два різні числа з однаковим набором залишків.

Тоді $|X_1|_{m_i} = |X_2|_{m_i}$, і тому $|X_1 - X_2|_{m_i} = 0$. Тому $X_1 - X_2$ є найменшим спільним кратним (lcm) m_i. Але якщо m_i є взаємно простими, то їх lcm дорівнює M, і має бути, що $X_1 - X_2$ є кратним M. Отже, не може бути, щоб $X_1 < M$ і $X_2 < M$. Отже, множина $\{|X|_{m_i} : 1 \leq i \leq N\}$ є унікальним і може бути прийнято як представлення X.

Запишемо представлення у вигляді $\langle x_1, x_2, \ldots, x_N \rangle$, де $x_i = |X|_{m_i}$, а зв'язок між X та його залишками вказуватимемо, що $X \cong \langle x_1, x_2, \ldots, x_N \rangle$.

Число M називається динамічним діапазоном СЗК, оскільки кількість чисел, які можна представити, дорівнює M. Для чисел без знаку цей діапазон дорівнює $[0, M - 1]$.

Представлення в системі, в якій модулі не є попарно взаємно простими, не будуть унікальними: два чи більше чисел матимуть однакове представлення. Як приклад, залишки цілих чисел від нуля до шістнадцяти відносно модулів три, п'ять і сім (які є попарно взаємно простими) подано в лівій половині таблиці 1.1. А залишки тих самих чисел відносно модулів два, чотири та вісім (які не є попарно взаємно простими) подано в правій половині тієї ж таблиці. Зверніть увагу, що в першій половині жодна послідовність залишків не повторюється, тоді як у другій є повторення.

Приклад у лівій половині таблиці 1.1 визначає систему залишкових класів, яку можна вважати стандартною і саме її будемо розглядати в першу чергу. Тим не менш, існують корисні приклади «нестандартних» СЗК, найпоширенішими з яких є надлишкова системи залишкових класів. Така система будується шляхом додавання додаткових (надлишкових) модулів до стандартної системи. Тоді динамічний діапазон складається з «легітимного» діапазону, визначеного ненадлишковими модулями, і «нелегітимного» діапазону; для арифметичних операцій початкові операнди та результати мають бути в допустимому діапазоні. Надлишкові системи числення цього типу особливо корисні у відмовостійких обчисленнях. Надлишкові модулі означають, що позиції цифр з помилками можуть бути виключені з обчислень, зберігаючи достатню частину динамічного діапазону. Крім того, можливі як виявлення, так і виправлення помилок: за допомогою k надлишкових модулів можна виявити до k помилок і виправити до $\lfloor k/2 \rfloor$ помилок. Іншу форму надлишковості можна запровадити шляхом розширення розміру набору цифр, що відповідає модулю, подібно до зайвої цифри зі знаком RSD. Для модуля m нормальним набором цифр є $\{0, 1, 2, \ldots, m - 1\}$; але якщо замість цього використовується набір цифр $\{0, 1, 2, \ldots, \tilde{m}\}$, де $\tilde{m} \geq m$, тоді деякі залишки матимуть надлишкові представлення.

Таблиця 1.1 – Залишки для різних модулів

N	Взаємно прості модулі			Взаємно непрості модулі		
	$m1 = 3$	$m1 = 5$	$m1 = 7$	$m1 = 2$	$m1 = 4$	$m1 = 8$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	0	2	2
3	0	3	3	1	3	3
4	1	4	4	0	0	4
5	2	0	5	1	1	5
6	0	1	6	0	2	6
7	1	2	0	1	3	7
8	2	3	1	0	0	0
9	0	4	2	1	1	1
10	1	0	3	0	2	2
11	2	1	4	1	3	3
12	0	2	5	0	0	4
13	1	3	6	1	1	5
14	2	4	0	0	2	6
15	0	0	1	1	3	7
16	0	1	2	0	0	0

1.1.1 Вибір модулів

З математичної точки зору найкращі модулі, це модулі які є простими числами. Особливо корисною властивістю таких модулів є «генерація». Якщо модуль m є простим, то існує принаймні один первісний корінь (або генератор), $p \leq m - 1$, такий, що множина $\{|p^i|_m : i = 0, 1, 2, \ldots, m - 2\}$ – множина всіх ненульових залишків відносно m. Наприклад, якщо $m = 7$, ми можемо взяти $p = 3$, оскільки $\{|3^0|_7 = 1, |3^1|_7 = 3, |3^2|_7 = 2, |3^3|_7 = 6, |3^4|_7 = 4, |3^5|_7 = 5\} = \{1, 2, 3, 4, 5, 6\}$; 5 також є первісним коренем. Очевидно, що для таких

модулів множення та піднесення до степеня залишків можна здійснити за допомогою простих операцій над показниками степеня первісного кореня, подібно до використання логарифмів і антилогарифмів у звичайному множенні.

Для комп'ютерних програм важливо мати набори модулів, які сприяють як ефективному представленню, так і балансу, де останнє означає, що відмінності між модулями мають бути якомога меншими. Візьмемо, наприклад, вибір 13 і 17 для модулів, які є суміжними простими числами; динамічний діапазон дорівнює 221. З простим двійковим кодуванням для представлення відповідних залишків будуть потрібні чотири біти та п'ять бітів відповідно. У першому випадку репрезентативна ефективність становить 13/16, а в другому – 17/32. Якщо натомість ми виберемо 13 і 16, ефективність представлення буде покращена — до 16/16 у другому випадку – але ціною зменшення діапазону до 208.

З іншого боку, з кращою збалансованою парою, 15 і 16, ми мали б кращу ефективність і більший діапазон: 15/16 і 16/16 для першого, і 240 для останнього.

Також корисно мати модулі, які спрощують виконання арифметичних операцій. Це означає, що арифметика з цифрами залишку не повинна надто відхилятися від традиційної арифметики, яка є просто арифметикою за модулем степеня двійки. Загальним вибором простого модуля, який не ускладнює арифметику і має хорошу ефективність, є $m_i = 2^i - 1$. Не всі пари чисел виду $2^i - 1$ є взаємно простими, але можна показати, що $2^j - 1$ і $2^k - 1$ є взаємно простими тоді і тільки тоді, коли j і k є взаємно простими. Багато наборів модулів базуються на цих умовах, але є й інші можливості; наприклад, модулі-множини виду $\{2^n - 1, 2^n, 2^n + 1\}$ є одними з найпопулярніших у використанні.

Отже, загалом існує принаймні чотири правила, які слід брати до уваги при виборі модулів. По-перше, вибрані модулі повинні забезпечувати адекватний діапазон, одночасно забезпечуючи унікальність представлень в СЗК. По-друге, як зазначено вище, ефективність двійкових представлень; у цьому відношенні також важливий баланс між різними модулями в даному наборі модулів. По-третє, в ідеалі реалізація арифметичних блоків для СЗК повинна бути певною мірою сумісна з тими для звичайної арифметики. І по-четверте, це розмір окремих

модулів: хоча, як ми побачимо, певні СЗК-арифметичні операції не вимагають переносів між цифрами, що є однією з головних переваг СЗК це стосується лише між цифрами. Оскільки цифра в кінцевому підсумку представлена в двійковій формі, між бітами будуть переноси, і тому важливо переконатися, що цифри (не є надто великими. Цифри з низькою точністю також дозволяють реалізувати економічно ефективні реалізації арифметичних операцій у вигляді таблиці. Але, з іншого боку, якщо модулі малі, то для забезпечення достатнього динамічного діапазону може знадобитися їх велика кількість. Звичайно, зрештою зроблений вибір і те, чи корисна СЗК чи ні, залежать від конкретних програм і технологій.

1.1.2 Арифметичні операції в системі залишкових класів

Стандартні арифметичні операції додавання/віднімання та множення легко реалізуються за допомогою запису залишків, залежно від вибору модулів, але ділення набагато складніше. Останнє не дивно, у світлі наведеного вище твердження про труднощі визначення знака та порівняння величини. Додавання залишків здійснюється шляхом індивідуального додавання відповідних цифр, відносно модуля їх позиції. Тобто перенесення з однієї позиції цифри не поширюється на позицію наступної цифри.

Якщо задані модулі є $m_1, m_2, \ldots, m_N$, $X \cong \langle x_1, x_2, \ldots, x_N \rangle$ і $Y \cong \langle y_1, y_2, \ldots y_N \rangle$, тоді $x_i = |X|_{m_i}$, $y_i = |Y|_{m_i}$ тоді ми можемо визначити додавання як $Z=X + Y$. $X + Y = \langle x_1, x_2, \ldots, x_N \rangle + \langle y_1, y_2, \ldots y_N \rangle = \langle z_1, z_2, \ldots z_N \rangle \cong Z$,

де $z_i = x_i + y_i$.

Приклад. Розглянемо операцію додавання в СЗК з набором модулів {3, 5, 7, 11}. Виконаємо операцію додавання двох чисел 16 і 18 . Числа в СЗК мають вигляд: $16 = \langle 1, 1, 2, 5 \rangle$, $18 = \langle 0, 3, 4, 7 \rangle$.

Відповідно, в результаті додавання отримаємо:

$$\langle 1, 1, 2, 5 \rangle + \langle 0, 3, 4, 7 \rangle = \langle 1, 4, 6, 1 \rangle.$$

При відніманні, якщо результат менше модуля необхідно додати модуль:

$$\langle 0,3,4,7\rangle - \langle 1,1,2,5\rangle = \langle 2,2,2,2\rangle = 18 - 16 = 2.$$

Множення також можна виконати простим множенням відповідних пар цифр-залишків відносно модуля їхньої позиції; тобто множити цифри та ігнорувати або коригувати відповідну частину результату. Якщо даними модулями є:

$m_1, m_2, \ldots, m_N$, $X \cong \langle x_1, x_2, \ldots, x_N\rangle$ і $Y \cong \langle y_1, y_2, \ldots y_N\rangle$, тоді $x_i = |X|_{m_i}$, $y_i = |Y|_{m_i}$,

тоді ми можемо записати множення як $X \times Y = Z$ на

$$X \times Y \cong \langle x_1, x_2, \ldots, x_N\rangle \times \langle y_1, y_2, \ldots y_N\rangle = \langle z_1, z_2, \ldots z_N\rangle \cong Z,$$

де $z_i = |x_i \times y_i|_{m_i}$.

Приклад. Набір модулів { 3, 5, 7, 11}. Виконаємо операцію множення чисел 16 і 18.

Отже $16 = \langle 1,1,2,5\rangle$, $18 = \langle 0,3,4,7\rangle$, тоді

$16 \times 18 = \langle 1,1,2,5\rangle \times \langle 0,3,4,7\rangle = \langle 0,3,1,2\rangle = 288.$

Як і у випадку додавання, отримання модуля щодо m_i може бути реалізовано без ділення та досить ефективно, якщо m_i мають відповідну форму.

Базове ділення з фіксованою комою складається, по суті, з послідовності віднімань, порівнянь величин і вибору часток-цифр. Але порівняння в СЗК є складною операцією, оскільки СЗК не є позиційною системою.

Розглянемо, наприклад, той факт, що з набором модулів {3, 5, 7, 11} число, представлене ⟨0, 0, 1, 4⟩, майже вдвічі більше, ніж представлене ⟨2, 3, 1 , 3⟩, але це далеко не очевидно. Тому слід очікувати, що ділення буде складною операцією. Одним із способів легкого виконання ділення є перетворення операндів у

звичайну систему, використання звичайної процедури ділення, а потім перетворення результату назад у нотацію залишку. Однак перетворення займають багато часу, тому, якщо можливо, слід використовувати прямий алгоритм. Таким чином, сутністю хорошого алгоритму ділення СЗК буде відносно ефективний метод виконання порівняння величин. Однак, всі вони вимагають перетворення з СЗК, і, порівняно зі звичайними алгоритмами ділення, усі вони мають недоліки.

1.2 Загальна структура обчислювального пристрою в системі залишкових класів

Першим кроком при проектуванні системи СЗК є вибір відповідного набору модулів, враховуючи динамічний діапазон (добуток усіх модулів) і ступінь необхідної паралелізму (відповідає кількості модулів у наборі). Набір модулів включає попарно взаємно прості числа. Загалом набори модулів RNS можна організувати у дві основні групи: набори модулів, зручні для арифметики, і набори модулів, зручні для перетворення. Набори модулів, зручні для перетворення, створюють простіші структури зворотного перетворення, на відміну від наборів, зручних для арифметики. Якщо для цільової програми швидкість внутрішніх арифметичних операцій до перетворень є високою, слід вибрати арифметику дружню, і навпаки. Деякі рекомендації щодо методу вибору вибіркових наборів модулів приведені в [16–19].

Модулі 2^n і $2^n \pm 1$ є найбільш відомими та ефективними, оскільки вони дозволяють проектувати прості апаратні арифметичні блоки. На основі цих популярних чисел для СЗК було введено багато спеціальних наборів модулів. Найбільш відомим є традиційний набір модулів $\{2^n - 1,\ 2^n, 2^n + 1\}$. Незважаючи на те, що більшість досліджень СЗК присвячена цьому набору трьох модулів, наданий обмежений паралелізм перешкоджає його використанню в сучасних високопродуктивних обчислювальних системах.

Таким чином, були запропоновані розширені набори модулів СЗК, такі як $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$ і $\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} - 1\}$. Необхідно

відмітити, що на додаток до спеціальних наборів модулів СЗК існують загальні набори модулів СЗК, де кожен модуль може бути вигідно обраний розробником; наприклад, для наборів простих модулів, де всі модулі є простими, множення за модулем можна перетворити на додавання на основі ізоморфізму [20]. Набори модулів з більшою кількістю модулів зазвичай розглядаються для додатків криптографії, які працюють з дуже великими числами [21].

Загальну структуру СЗК зображено на рисунку 1.1. Прямий перетворювач отримує зважені цілі операнди та створює залишки. Наприклад, розглянемо набір модулів $\{2^n - 1, 2^n, 2^n + 1\}$ і $n = 4$.

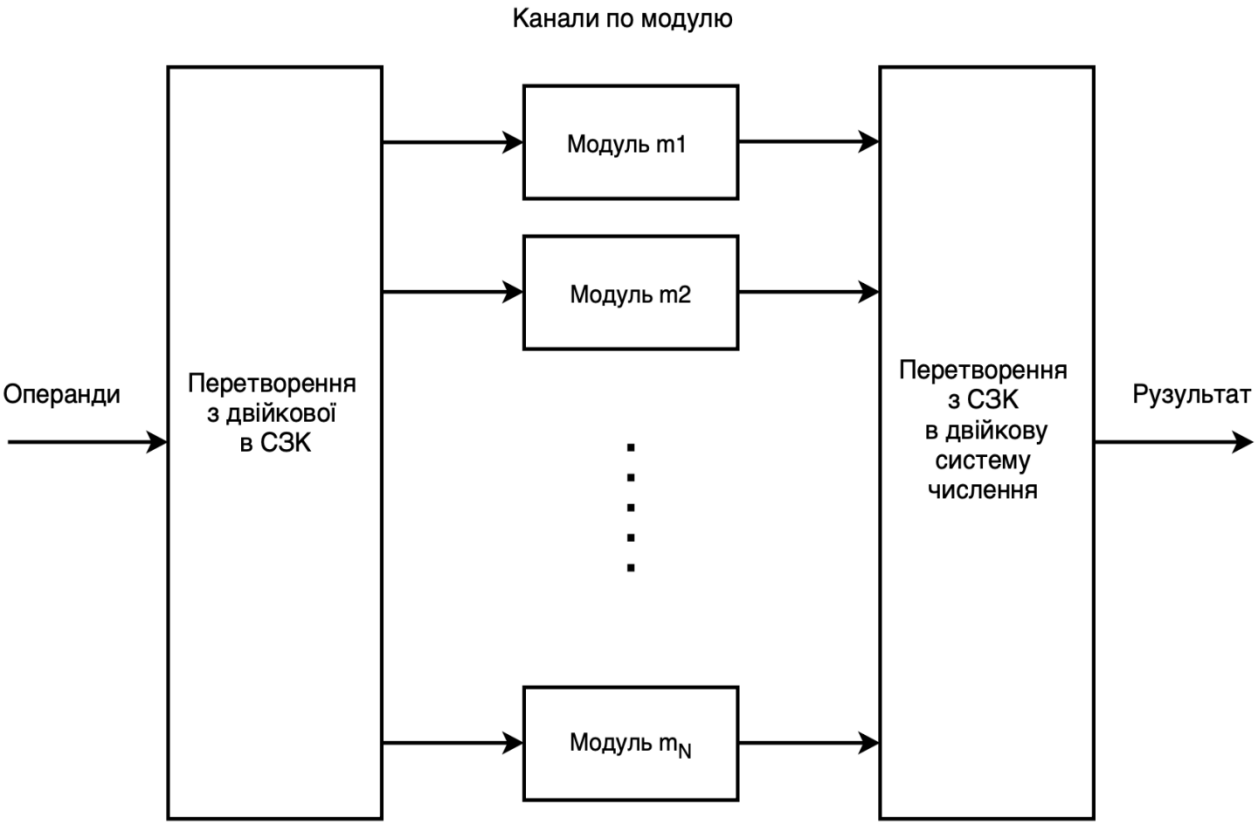


Рисунок 1.1 – Структура обчислювального пристрою в СЗК

Динамічний діапазон становить 4080, а представлення залишку початково зваженого вхідного $X = 20$ може бути досягнуто шляхом обчислення залишку від ділення X за різними модулями, в результаті чого $X = (5, 4, 3)$. На практиці не потрібно обчислювати весь процес ділення, щоб отримати залишки, але потрібні лише модульні додавання, щоб отримати їх для окремих наборів модулів [22].

Таким чином, прямі перетворювачі є паралельними архітектурами для різних залишків, які складаються з незалежних блоків, кожен з яких обчислює один залишок на основі суматорів. Проект прямого перетворювача розглядався в багатьох роботах, таких як [23–31]. Крім того, нещодавно конструкція прямих перетворювачів для загальних наборів модулів СЗК була розглянута в [32].

Конструкція зворотного перетворювача складається з п'яти етапів. На першому етапі вибирається алгоритм перетворення відповідно до набору модулів і його властивостей. На наступному етапі значення модулів разом із значеннями мультиплікативних обернених враховуються у формулах перетворення. Тоді арифметичні властивості залишків слід використовувати для роботи над рівняннями перетворення таким чином, щоб спростити обчислення. Далі спрощені рівняння реалізуються за допомогою арифметичних схем, таких як суматори та логічні вентилі. Найвідомішими алгоритмами перетворення є: китайська теорема про залишки (КТЗ), перетворення змішаного принципу (MRC) і нові КТЗ-I та КТЗ-II, похідні від оригінальної КТЗ [22]. Існує багато конструкцій зворотних перетворювачів для наборів модулів СЗК. Оскільки структура зворотного перетворювача залежить від наборів модулів, спеціальні набори модулів з обмеженою кількістю модулів зазвичай використовуються для проектування ефективних зворотних перетворювачів для оптимізації конструкції для різних функцій витрат, таких як продуктивність, вартість і енергоспоживання.

Конструкції модульного суматора та помножувача також є важливим напрямком досліджень у СЗК, оскільки вони використовуються не лише в паралельних модульних арифметичних каналах, але також у прямих та зворотних перетворювачах. Існують різні архітектури для проектування суматорів і помножувачів для загальних і специфічних модулів [7], більшість з них для набору модулів $2^n - 1,\ 2^n + 1, \text{і}\ 2^n$ (для цього останнього модуля можна використовувати звичайні двійкові суматори і помножувачі [22]).

Існує багато інших операцій у СЗК, які можуть знадобитися додаткам. Одним з них є виявлення ознак. На відміну від зваженої системи числення, де

знак можна визначити за старшим бітом (MSB), у СЗК виявлення знаку є складним, вимагаючи порівняння чисел із половиною динамічного діапазону: цілі числа в першій половині динамічного діапазону вважаються позитивними, а всі інші негативними [7]. Крім того, порівняння чисел є ще однією важкою операцією в СЗК. Зазвичай напівперетворення числа СЗК у зважений еквівалент застосовується для розробки детекторів знаків СЗК і компараторів величини. Інші складні операції - масштабування та ділення. Це важливі операції, а саме для уникнення переповнення в рядах множення і додавання в СЗК, які є типовими операціями цифрової фільтрації. Коефіцієнт масштабування зазвичай розглядається як ступінь за модулем два, щоб зменшити складність операції. Переповнення, яке виникає, коли результат операції не можна представити в динамічному діапазоні, також досить важко виявити в СЗК.

1.3 Аналіз областей ефективного застосування системи залишкових класів

СЗК широко відома як альтернативна система числення, яка може прискорити швидкість програм, у яких домінує велика кількість додавання, віднімання та множення. Приклади, наведені у даному пункті, включають добре відомі програми СЗК у цифровій обробці сигналів, криптографії та модулях пам’яті, а також її менш відомі програми в мережі та хмарних обчисленнях, які використовують функції СЗК для задоволення певних критичних обмежень у безпеці, енергоспоживанні та продуктивності.

1.3.1 Хмарне сховище

Проблеми несподіваного припинення надання послуг і порушення конфіденційності даних поточними постачальниками хмарних сховищ можуть бути вирішені з допомогою НСЗК. Щоб зберегти файл у хмарі з більшою надійністю з точки зору довгострокової доступності та конфіденційності, він

розбивається на $p + r$ сегментів залишків на основі НСЗК, де r є кількістю надлишкових модулів. Кожен фрагмент кодується за допомогою BASE-64 перед тим, як він шифрується за допомогою симетричного алгоритму для інкапсуляції двійкових даних у корисному навантаженні XML-файлу-оболонки. Нарешті, кожен зашифрований фрагмент надсилається до іншого постачальника хмарного сховища. Клієнт створює та зберігає файл карти метаданих XML, що описує розташування та метод пошуку різних фрагментів. У випадку, якщо будь-який із пристроїв зберігання даних тимчасово або назавжди виходить з ладу, вихідний файл все ще може бути легко реконструйований клієнтом, використовуючи лише p фрагментів, включаючи зайві сегменти залишків. Такий підхід до зберігання не тільки захищає збережені файли від системних збоїв, але й запобігає доступу хмарного постачальника до збережених файлів, оскільки лише власник знає місця зберігання блоку та спосіб доступу до них. Крім того, паралельне завантаження окремих фрагментів від різних постачальників хмарних сховищ також призводить до ефективного використання пропускної здатності. Розмір зберігання файлу є функцією r і мінімальної кількості модулів, необхідних для реконструкції файлу. При однаковому допуску до помилок відношення обсягу пам’яті, необхідного для традиційного підходу резервування, який зберігає кілька копій, до того, якого вимагає цей підхід, було встановлено приблизно 1,75 [33].

1.3.2 Відмовостійкі обчислення

Надійність електронних схем значно погіршується через агресивне масштабування пристрою. Відмовостійкість є надзвичайно важливим параметром для надійної роботи наноелектронних схем та мінімізації втрат продуктивності та виходу з ладу приладів. Кілька методів, які включають логіку самоперевірки, реплікацію модулів, код виправлення помилок, реконфігурацію тощо, були розроблені для підвищення надійності електронних схем [34]. Ємність, властива арифметичним операціям типу СЗК. Відсутність упорядкованої значущості серед залишків СЗК означає, що помилки через шум обробки в одній цифрі залишку не

будуть забруднювати інші цифри залишку, і несправний модульний канал може бути закритий, якщо збережені канали мають адекватний динамічний діапазон. В іншому випадку можна додати додаткові модулі, щоб забезпечити бажану відмовостійкість.

Одним із застосувань, що включає код виявлення та виправлення помилок на основі НСЗК, є гібридна пам’ять [35]. У гібридній пам’яті пристрої, що не є CMOS, використовуються як комірки пам’яті разом із периферійними схемами на основі CMOS. Порівняно зі звичайними елементами пам’яті CMOS, гібридна пам’ять пропонує більшу ємність для зберігання даних, але має вищий рівень дефектів на 10% або більше через високу варіативність виробничого процесу нових нанопристроїв. Перший код НСЗК, розроблений для дефектостійких систем пам'яті, складається з шести модулів форм $\{2^n + 1, 2^n, 2^{n-1} - 1, 2^{n-2} - 1, 2^{n-3} - 1, 2^{n-4} + 1\}$, де $2^n + 1$ і 2^n - інформаційні модулі, а $2^{n-1} - 1$, $2^{n-2} - 1$, $2^{n-3} - 1, 2^{n-4} + 1$ - надлишкові модулі.

На відміну від звичайного коду НСЗК, надлишкові модулі менші, ніж інформаційні модулі, що викликає неоднозначність у виправленні помилок. Неоднозначність усувається за допомогою техніки декодування максимальної правдоподібності. У результаті за допомогою цієї схеми можна зберегти більше даних, оскільки довжина кодового слова коротша, ніж код Ріда-Соломона (RS) і звичайний код НСЗК для 16-бітної, 32-бітної та 64-бітної пам’яті. Спочатку вхідні дані перетворюються на набір залишків за допомогою кодера НСЗК. Потім залишки об’єднуються для створення кодового слова НСЗК перед його збереженням у комірках пам’яті. Щоб отримати дані, зважене число реконструюється із залишків. Якщо зважене число потрапляє в допустимий діапазон інформаційних модулів, витягуються правильні дані. В іншому випадку дані містять помилку, і помилкову цифру залишку можна знайти шляхом повторного обчислення величини представлення залишку після виключення одного залишку за раз, доки обчислена величина не потрапить у допустимий діапазон. Виключений залишок, який повертає величину до допустимого діапазону, є помилковим.

Крім того, НСЗК також використовувався для зменшення помилок у FIR-фільтрах [36]. У порівнянні з TMR, підхід на основі НСЗК споживає менше апаратної площі для тієї самої можливості виправлення помилок [37].

Підхід, заснований на НСЗК, може усунути м'які помилки, спричинені порушенням одноразової події у FIR-фільтрах з нульовою частотою пропусків помилок і використанням меншої площі апаратного забезпечення, ніж традиційні схеми пом'якшення SEU [38].

Іншим застосуванням коду на основі НСЗК є схема модуляції з кількома несучими [39]. Він адаптивно закодований у НСЗК з трьома інформаційними та шістьма надлишковими залишками. Кожен залишок має 8 біт. Порівняно з іншими кодами НСЗК з різною кількістю інформації та надлишковими залишками, код НСЗК (9, 3) забезпечує найсильнішу можливість виправлення помилок із коефіцієнтом кодування 0,33. Ця схема може ефективно протидіяти частотно-селективним ефектам завмирання, викликаним дисперсійними широкосмуговими каналами. Для цільового коефіцієнта бітових помилок 10^{-4} він перевершував усі інші схеми на основі згорткового коду щодо пропускної здатності біт на символ, коли SNR було вище 15 дБ [40]. Для підтримки більшого системного різноманітності та виграшу кодування нещодавно на основі НСЗК були сформульовані просторово-часові блокові коди [41]. Залишки відображаються безпосередньо на складні сузір'я за допомогою схем прямого, дистанційного та непрямого відображення.

Відображені символи створюються як просторово-часові блокові коди та передаються через декілька антен. У приймачі вихідна інформація відновлюється за допомогою КТЗ.

Структура коду НСЗК дає змогу зменшити складність декодування максимальної правдоподібності м'якого рішення за допомогою адаптивної схеми М-ичного зворотного відображення з відображенням з урахуванням відстані. Що ще важливіше, високомодульний систематичний НСЗК дозволяє кодувати та декодувати коди з різними параметрами однаковими обладнання. Короткі довжини блоків НСЗК також добре підходять для систем зв'язку з низькою

затримкою та реалізації стандартів бездротового зв’язку з декількома входами та кількома виходами.

1.3.3 Криптографія

Криптографія з відкритим ключем в основному використовується для забезпечення конфіденційності, автентичності та неспростовності в електронних комунікаціях, не вимагаючи прихованого каналу для обміну ключами між сторонами зв’язку. Шифрування та дешифрування в алгоритмі з відкритим ключем включає обчислювально дорогі великі модульні множення та модульне піднесення до степеня через великі ключі, необхідні з міркувань безпеки. Це має важливі наслідки для їх практичного використання в мобільних пристроях і легких програмах.

СЗК є корисним для прискорення апаратної реалізації модульного множення та модульного піднесення до степеня в алгоритмах Rivest, Shamir and Adleman (RSA) [61] та Еліптичної кривої (ECC) [42]. Будучи ядром модульного піднесення до степеня RSA та множення точки ECC, множник Монтгомері на основі СЗК є швидшим, ніж двійковий множник Монтгомері [43, 44]. Використання СЗК у RSA бере свій початок з двох добре відомих множень Монтгомері на основі СЗК у 1992 та 1995 роках [18]. Пізніше вони були розширені до повноцінної реалізації СЗК RSA [76], де всі обчислення виконуються в домені СЗК без необхідності будь-яких прямих і зворотних перетворень. Щоб досягти цього, відправник і одержувач повинні заздалегідь узгодити набір параметрів СЗК. З іншого боку, найважливішою та обчислювально інтенсивною арифметичною операцією ECC є множення точок. Будь-яке прискорення множення точки призведе до помітного покращення продуктивності ECC.

Модульне множення СЗК Montgomery виявилося найшвидшим проектним підходом для прискорення множення точок еліптичної кривої, використовуючи менше половини площ інших попередніх проектних зусиль [45]. Воно працює

краще, оскільки збільшується довжина ключа ECC. Швидкість і область множення СЗК Монтгомері були додатково покращені за допомогою використання арифметичних модулів з конвеєрною архітектурою [46].

Повністю гомоморфне шифрування (FHE) є важливою віхою в розвитку сучасної криптографії [34]. За допомогою FHE можна виконувати обчислення безпосередньо над зашифрованими текстами, не розкриваючи секретний ключ. Це може бути корисним для хмарних обчислень, щоб об'єднати різні служби, не відкриваючи дані для кожної з цих служб. Однак складність реалізації FHE ускладнила його використання на практиці.

Щоб показати масштаб проблеми, перша реалізація варіанту FHE на основі решітки зайняла 17 мегабайт відкритих ключів, більше однієї секунди для шифрування одного біта та майже півхвилини для повторного шифрування примітиву на високоякісному сервері на основі Intel Xeon. Спеціальне програмне забезпечення та апаратні методи були досліджені, щоб зробити FHE більш ефективним. Останнім часом з'явилася більш ефективна схема FHE без початкового завантаження, але для її шифрування потрібне велике множення цілочисельної матриці-вектора. СЗК можна використовувати для розкладання кожного великого цілого елемента матриці на менші залишки. Над цими меншими залишками потім виконуються векторні операції, що включають модульне множення та додавання. Кінцевий результат реконструюють із залишків за допомогою КТЗ. Оскільки процес векторних операцій споживав більшу частину часу центрального процесора (CPU), його прискорив графічний процесор (GPU). Цей метод на основі СЗК на CPU та GPU був відповідно в 7,8 рази та 273 рази швидшим, ніж звичайна бібліотека теорії чисел на CPU.

У гомоморфній схемі шифрування у формі $c = pq + m$, де c – зашифрований текст, m – повідомлення відкритого тексту, p – ключ, а q – випадкове число, функція шифрування є зворотною до операції залишку, тобто m є залишком c відносно модуля p.

Це принцип, який лежить в основі безпеки НОСЗК (розшифровується як Homomorphic encryption з СЗК) [60] для використання ненадійної хмари для

обчислення даних клієнта. НОСЗК генерує кілька залишків даних, а операції з ними виконуються хмарою за допомогою гомоморфного шифрування. Щоб забезпечити конфіденційність даних, дані X і динамічний діапазон M не повинні виводитися хмарою із залишків x_i та його модулів m_i. Щоб заплутати хмару, випадковий шум додається до даних. $r_x \in R_d$, а операції виконуються над $X' = X + |R_X| \cdot r_x$ замість X. Оскільки діапазон даних скорочується до M/$|R_X|$, результат будь-якого обчислення з хмари все ще можна перетворити на вихідний області даних шляхом знаходження залишку X' відносно $|R_X|$.

M можна так само затьмарити, помноживши модуль m_i на випадковий шум $r_{m_i} \in R_M$ і результати з хмари, виконані з використанням перетвореного модуля $m_i' = m_i \cdot r_{m_i}$, можуть бути перетворені на вихідний модуль за допомогою $(x_i \, mod \, m_i') \, mod \, m_i$. Щоб зменшити шанси будь-якої хмари успішно отримати дані та динамічний діапазон шляхом змови, обчислення можна розподілити між різними хмарами.

Атаки по бічних каналах є критичними загрозами для криптосистем, які здатні розкрити таємницю шляхом аналізу різних джерел витоку, таких як споживання енергії та часу під час виконання криптографічної операції на апаратних пристроях. Однією з можливих стратегій подолання цієї проблеми на арифметичному рівні є використання СЗК для маскування даних і внутрішніх обчислень у поєднанні з захищеними методами. Це основа концепції стійкої до витоку арифметики. Крім того, базова рандомізація в RSA на основі СЗК може забезпечити високу надійність проти атаки по бічних каналах без будь-яких накладних витрат у порівнянні з незахищеною звичайною реалізацією. Крім того, алгоритм AES також може бути захищений від атак збоїв поліноміальним СЗК. У цьому методі три паралельних ядра AES на основі СЗК, включаючи одне резервне ядро, використовуються для обробки СЗК-представлення вихідних даних. Після завершення обчислень залишки перетворюються назад у зважений еквівалент із подальшим механізмом виявлення переповнення, який може виявляти до 4 бітів помилок.

Доведено, що аналіз потужності є небезпечною атакою у вбудованих системах. Щоб захистити шифри відкритого ключа як від атак простого, так і від диференційованого аналізу потужності, можна скористатися перевагами розпаралелювання в багатомодульному СЗК, щоб деформувати захищену інформацію та вибрати модулі випадковим чином для кожної бітової операції ключа. Окрім покращеної безпеки, він має всі переваги СЗК, щоб швидше обробляти велику кількість вхідних даних і споживати менше енергії.

1.4 Висновки. Постановка задачі

Система залишкових класів, завдяки своїй здатності виконувати паралельні та швидкі обчислення, додає переваг сучасним системам цифрової обробки сигналів, які ставлять на перше місце вимогу енергоефективності. Проведений аналіз показав переваги СЗК та значний потенціал до вдосконалення цифрових систем у багатьох галузях. Показано, що функції СЗК можуть бути застосовані як інструмент у різноманітних програмах, щоб розширити їх функції або прискорити їх продуктивність. Завдяки аналізу практичних застосувань випливає, що розробка та дослідження перетворювачів з двійкової системи в СЗК є актуальною задачею.

Метою кваліфікаційної роботи є розробка програмно- технічного модуля перетворення даних з двійкової системи в систему залишкових класів.

2 МЕТОДИ ПЕРЕТВОРЕННЯ ДАНИХ В СИСТЕМУ ЗАЛИШКОВИХ КЛАСІВ

2.1 Табличний метод перетворення

Попереднє перетворення двійкових чисел у залишкові числа зі спеціальним набором модулів, напр. $(2^j - 1, 2^j, 2^j + 1)$ може бути реалізовано відносно легко і не потребує складних операцій, таких як обчислення мультиплікативного обернення, множення тощо [2]. Ті з довільними наборами модулів, як правило, більш складні, ніж зі спеціальним набором модулів [7]. Коли ми можемо перетворити двійкові числа в залишкові числа з довільним набором модулів за допомогою таблиць, кожен запис яких має бути попередньо обчислений і збережений.

Основна ідея використання таблиць пошуку полягає в тому, що знаходження залишку числа по відношенню до даного модуля по суті зводиться до обчислення модульної суми певних степенів двійки [7].

Приклад 3: Таблиця 2.1 показує таблицю перетворення для СЗК (3, 5, 7).

Коли ми перетворюємо двійкове число 1011100 у СЗК (3, 5, 7), ми бачимо, що біти , 2, 3, 4 і 6 мають 1.

Ми можемо обчислити $\hat{x}$, виконавши додавання за модулем 3 виділених значень комірок у таблиці 2.1, наступним чином:

$$\hat{x}_1 = |1 + 2 + 1 + 1|_3 = 2.$$

Аналогічно також можна обчислити і $\hat{x}_2$, $\hat{x}_3$:

$$\hat{x}_2 = |4 + 3 + 1 + 4|_5 = 2,$$
$$\hat{x}_3 = |4 + 1 + 2 + 1|_7 = 1.$$
$$1011100 = (2, 2, 1)_{RNS(3,5,7)}$$

Розглянемо розмір пам'яті для таблиці перетворення. Оскільки кількість рядків у таблиці дорівнює M, потрібна пам’ять $\lceil log_2 M \rceil$ адресних рядків. Розрядність кожного виходу становить $\lceil log_2 m_i \rceil$.

Таблиця 2.1 – Таблиця перетворення СЗК (3, 5, 7)

j	2^j	$\left\|2^j\right\|_3$	$\left\|2^j\right\|_5$	$\left\|2^j\right\|_7$
0	1	1	1	1
1	2	2	2	2
2	4	**1**	4	4
3	8	**2**	3	1
4	16	**1**	1	2
5	32	2	2	4
6	64	**1**	4	1

При цьому є загальний розмір пам'яті для таблиць перетворення становить:

$$V = 2^{\lceil log_2(M) \rceil} \sum_{i=1}^{n} \lceil log_2 m_i \rceil$$

Приклад 4: Для рядка адреси СЗК(3,5,7)=7, розрядність виводу дорівнює:

$$\lceil log_2 3 \rceil + \lceil log_2 5 \rceil + \lceil log_2 7 \rceil = 2 + 3 + 4 = 8.$$

Таким чином, загальний обсяг пам'яті становить:

$$2^{\lceil log_2(3*5*7) \rceil} \{ \lceil log_2 3 \rceil + \lceil log_2 5 \rceil + \lceil log_2 7 \rceil \} = 2^7 * (2 + 3 + 3) = 1024 \text{ біт.}$$

Приклад 5. Розглянемо приклад прямого перетворення, який містить 2048 біт двійкових чисел у СЗК. Припустимо, що кількість модулів дорівнює 64, кожен модуль має 32 біти. У цьому випадку загальний розмір пам’яті становить:

$$2048 * 32 * 64 = 2^{22} \text{ біт.}$$

Послідовний прямий перетворювач, показаний на рисунку 2.1, має послідовну структуру, яка складається з таблиці пошуку, яка зберігає всі значення модульного суматора $\left|2^j\right|_{mi}$, який обчислює часткові суми, лічильника та регістру накопичувача [7].

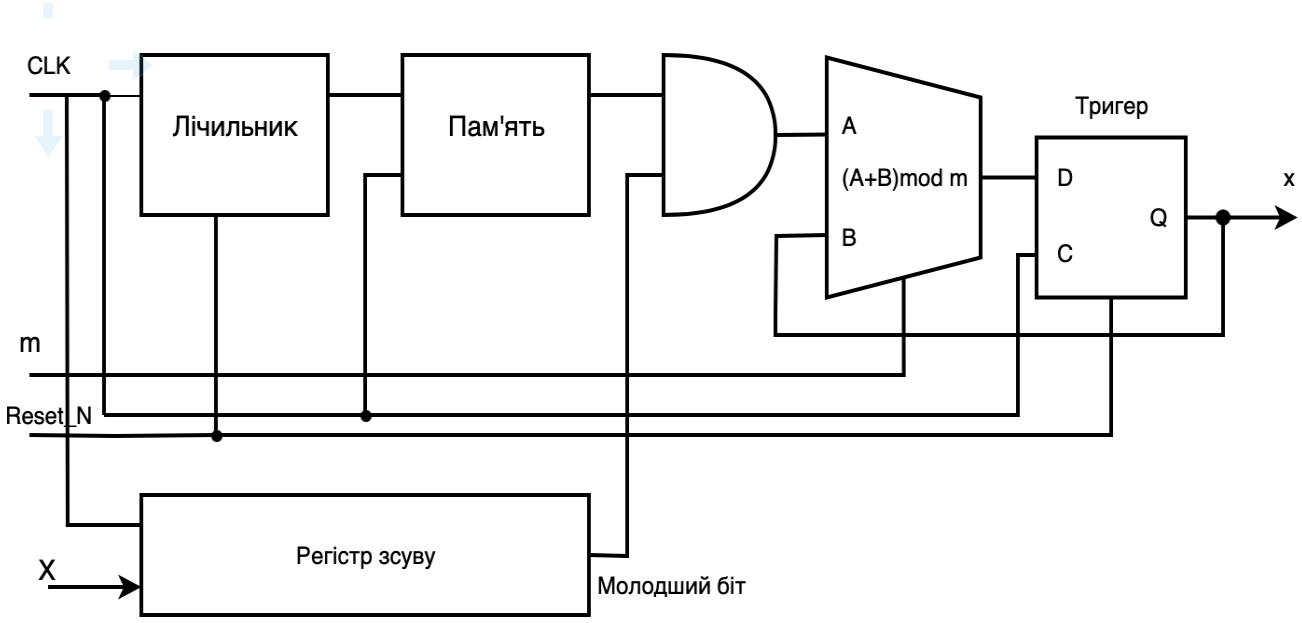


Рисунок 2.1 – Перетворювач в СЗК на основі таблиці пошуку

Приклад 6: Нехай X дорівнює $(101\_1100)_2$, а m дорівнює 5.

Давайте обчислимо X (mod 5) за допомогою послідовного прямого конвертера. Зсув X вправо на один біт за такт. Спочатку вихідний регістр очищається до 0. У тактових періодах 1, 2, і 6 LSB регістра зсуву дорівнює 0, накопичувач підтримує попередні значення. Тактові періоди 3, 4, 5 і 7. Молодші біти для таблиці перетворення дорівнюють одиниці.

Перетворити X на $\hat{x}$ можна подати наступним чином:

$$\hat{x} = ||||0 + 1|_7 + 2|_7 + 1|_7 + 4|_7 = 1.$$

Модуль пом’яті зберігає попередньо обчислені таблиці перетворення.

Лічильник подає сигнали адреси в таблицю. Присвоїмо X регістру зсуву. Коли LSB регістра зсуву дорівнює одиниці, накопичувач за модулем m і додається до виходу таблиці. Одне перетворення виконується за один такт.

Коли динамічний діапазон M є великим, послідовний перетворювач пошуку таблиці потребує пам'яті дуже великого розміру. Захищені програми можуть бути створені з використанням множинної точності, наприклад 2048 біт або більше СЗК. Для цих випадків необхідно модифікувати послідовні перетворювачі пошуку в таблиці.

2.1.1 Розрахунок таблиці послідовного перетворення

Перетворювачі на основі послідовних таблиць пошуку з великим динамічним діапазоном вимагають великого обсягу пам'яті. Відповідно, можна використати схему перетворення без пам'яті для зберігання таблиць перетворення.

Розглянемо метод обчислення таблиці схеми послідовних модульних арифметичних перетворень замість LUT. Тобто ми будемо послідовно обчислювати значення таблиці перетворення.

1. Обчислення таблиці послідовного перетворення:

$$\left|2^{j+1}\right|_m = \left|2 * \left|2^j\right|_m\right|_m, \text{ де m} \geq 3. \qquad (2.1)$$

Значення таблиці перетворення можна обчислити за формулою 2.1.

Приклад 7. Таблицю перетворення, показану в прикладі 3, можна розрахувати для модуля 7 наступним чином:

$|2^0|_7 = 1,$

$|2^1|_7 = |2 * |2^0|_7|_7 = |2 * 1|_7 = 2,$

$|2^2|_7 = |2 * |2^1|_7|_7 = |2 * 2|_7 = 4,$

$|2^3|_7 = |2 * |2^2|_7|_7 = |2 * 4|_7 = 1,$

$|2^4|_7 = |2 * |2^3|_7|_7 = |2 * 1|_7 = 2,$

$|2^5|_7 = |2 * |2^4|_7|_7 = |2 * 2|_7 = 4,$

$|2^6|_7 = |2 * |2^5|_7|_7 = |2 * 4|_7 = 1.$

Обчислення таблиці розрахунку КТЗ відбувається згідно рисунку 2.2.

На основі КТЗ можна обчислити кожне значення в таблицях перетворення в порядку зростання в кожному тактовому періоді таким чином:

1) вихідний регістр R ініціалізується на 1, тобто $|2^0|_{m_i}$, коли $m_i \geq 3$;

2) блок віднімання обчислює $2R - m_i$;

3) мультиплексор вибирає, якщо $(2R - m_i < 0)$ $Y \leftarrow 2R$, інакше $Y \leftarrow 2R - m_i$;

4) у наступний тактовий період вихідний регістр R оновлюється до Y.

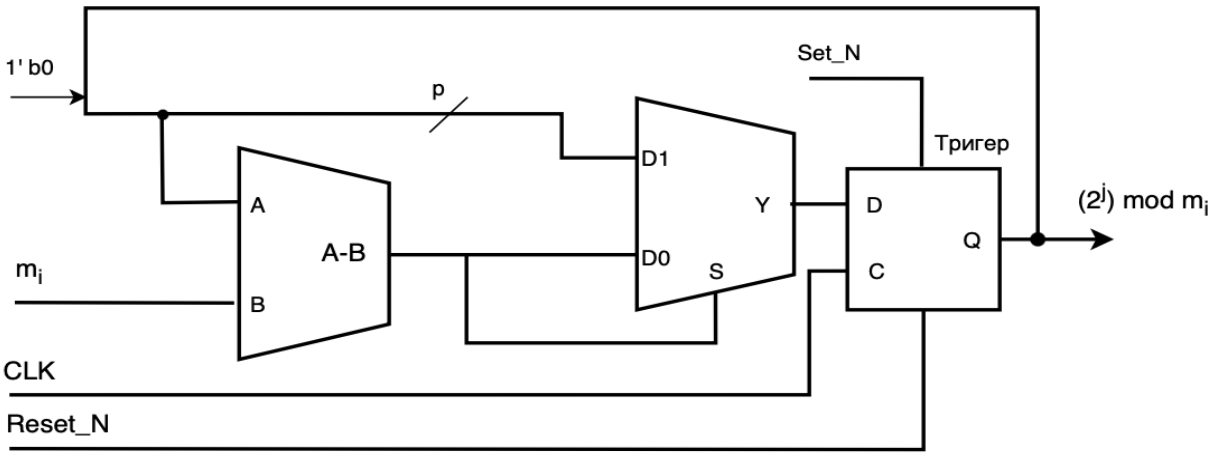


Рисунок 2.2 – Перетворення за допомогою таблиці

Прямі перетворювачі з обчисленням таблиці перетворення. Замінивши лічильник і пам’ять на рисунку 2.1 на обчислювач таблиці, як показано на рисунку 2.2, ми можемо отримати блок послідовного прямого перетворення (рисунок 2.3). Він може перетворювати двійкове число у число залишків за $\lceil log_2 M \rceil$ тактів, як показано на рисунку 2.1. При цьому він має дві особливості.

1. Не потребує пам’яті.

2. Модуль можна легко змінити, якщо ми використовуємо його для програм безпеки.

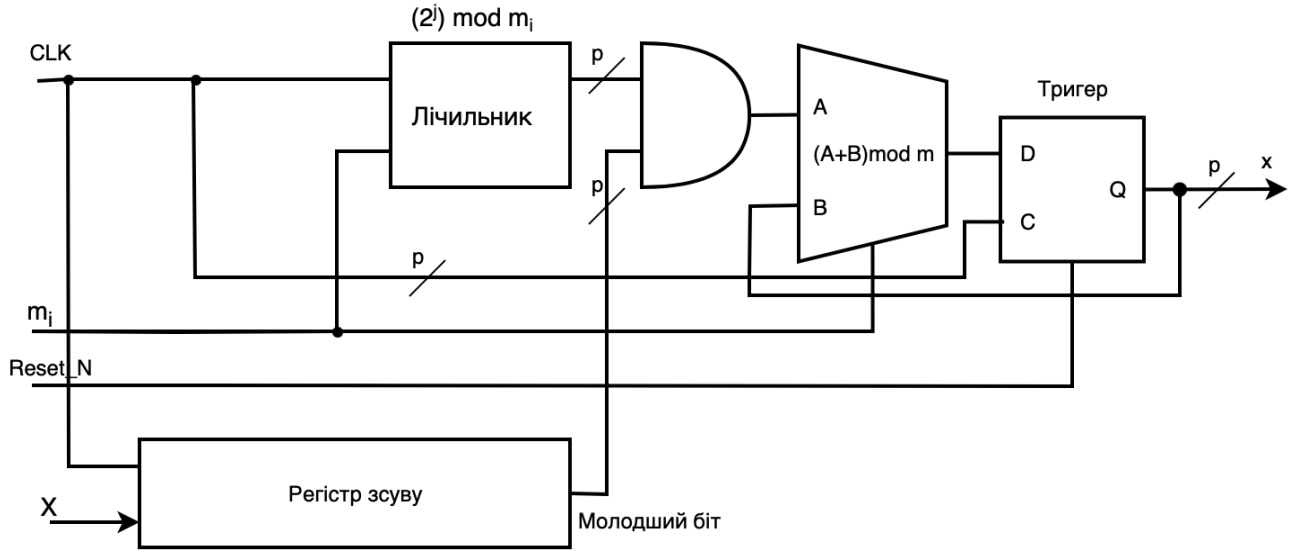


Рисунок 2.3 – Послідовний прямий перетворювач з обчисленням таблиці перетворення

Коли нам потрібен більший динамічний діапазон, цим перетворювачам не потрібна великий обсяг пам'яті.

На рисунку 2.4 показані прямі перетворювачі з n – еквівалентними блоками прямих перетворювачів. Отже, його можна легко розширити, збільшивши кількість модулів, тобто додавши послідовні перетворювачі.

2.1.2 Зменшення затримки прямих перетворювачів

Для прямого перетворювача, показаного на рисунку 2.4, потрібен час обчислення тактового сигналу $\lceil log_2 M \rceil$. Затримку можемо зменшити шляхом паралельного обчислення.

Нехай P – число розпаралелювань, а W – розрядність динамічного діапазону. Ми можемо скоротити його до $\frac{\lceil W/P \rceil}{W}$, розділивши таблицю перетворення на P таблиць.

Для цього методу ми повинні змінити таблицю перетворень, додавши функцію завантаження, як показано на рисунку 2.5.

Приклад 8. Нехай вхідне значення X буде $(92)_{10}$=(101_1100), Xh – старші біти (0101), а X_l – молодші біти (1100)

Перетворимо X в СЗК(3, 5, 7). Ми можемо обчислити цю операцію за годинниковим часом за допомогою послідовного конвертера. Давайте розділимо таблицю перетворення на дві таблиці, як показано в таблиці 2.2

Ми можемо обчислити цю операцію за 4 такти за допомогою паралельного послідовного перетворювача, показаного на рисунку 2.5. Прямі перетворювачі на рисунку 2.6 включають модифікований перетворювач на основі таблиці приведеної на рисунку 2.5.

Крок 0. Високий регістр зсуву (SRH) = 0101 і низький регістр зсуву (SRL) = 1100. $(m1, m2, m3) = (3, 5, 7)$. Прямий перетворювач i_0 обчислює $\left|2^j\right|_{m_i}$, $i = 0, \dots, 3$. Прямий перетворювач i_1 обчислює $\left|2^j\right|_{m_i}$, $i = 4, \dots, 7$.

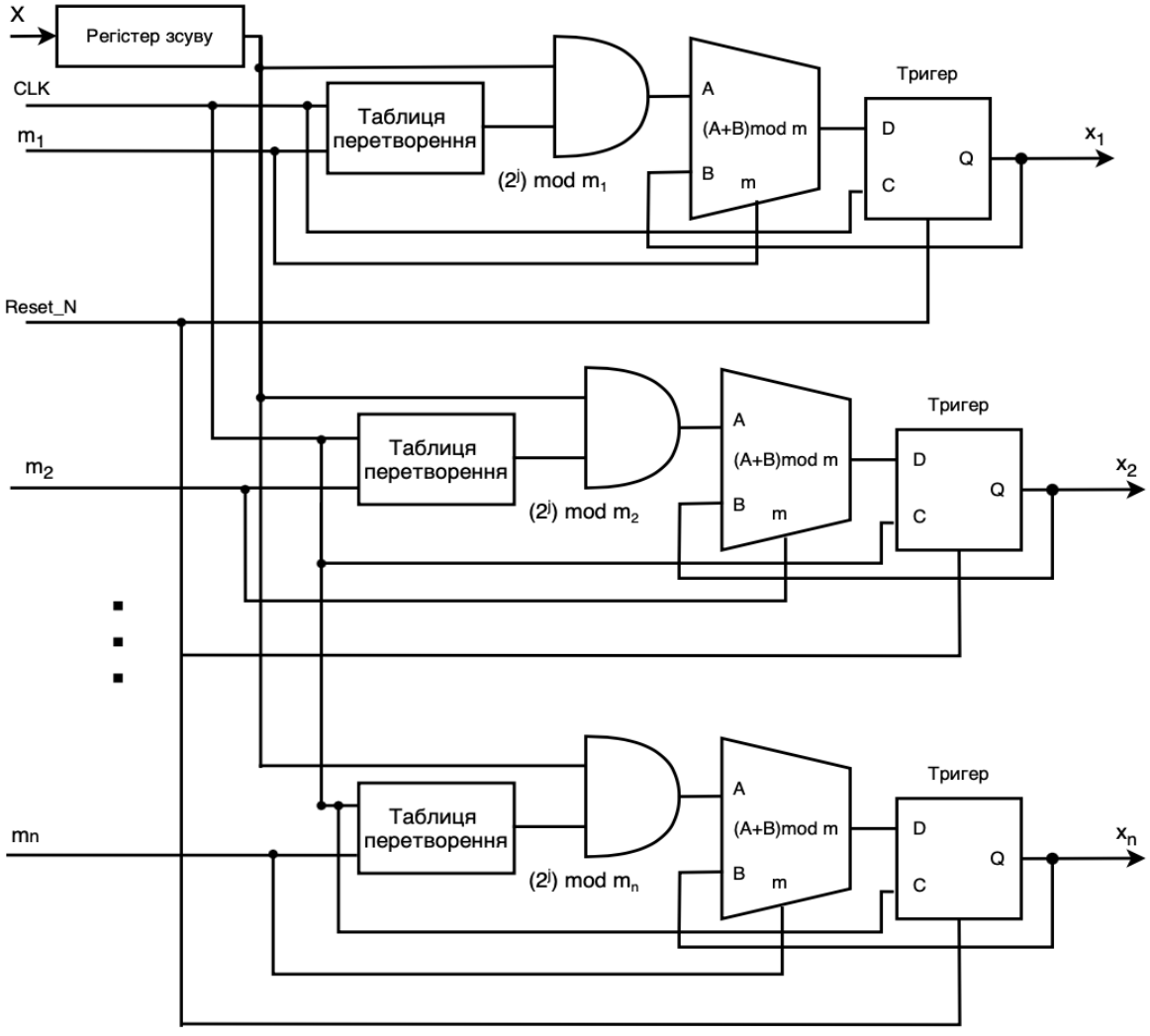


Рисунок 2.4 – Прямий перетворювач

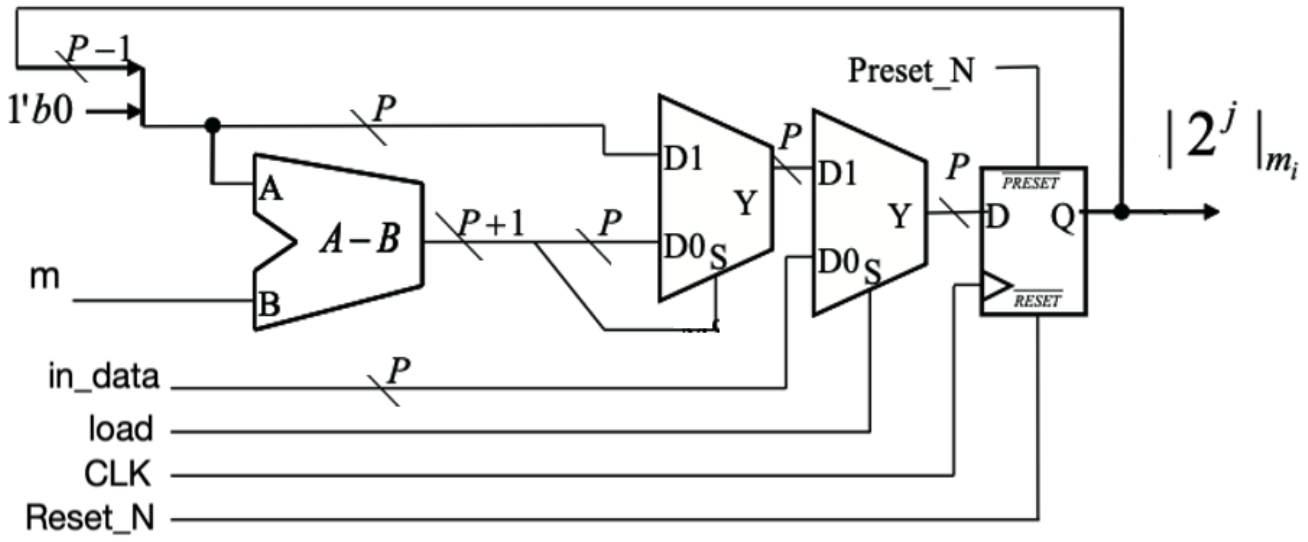


Рисунок 2.5 – Перетворювач на основі таблиць із регістром завантаження

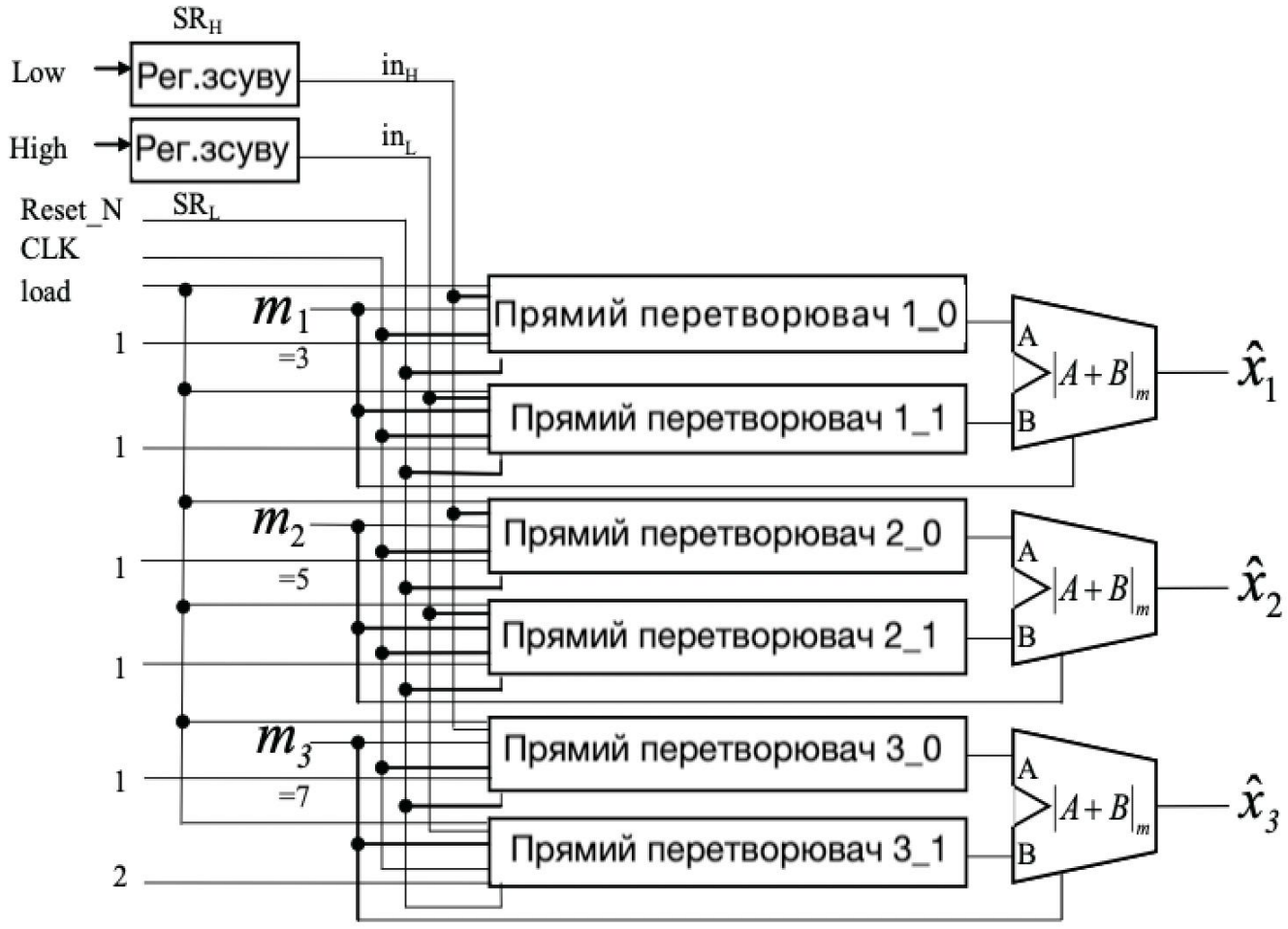


Рисунок 2.6 – Прямий перетворювач для СЗК(3, 5, 7) з паралельним обчисленням

Таблиця 2.2 – Розділення таблиці перетворення

j	$\left\|2^j\right\|_3$	$\left\|2^j\right\|_5$	$\left\|2^j\right\|_7$
0	1	1	1
1	2	2	2
2	1	4	4
3	2	3	1
4	1	1	2
5	2	2	4
6	1	4	1
7	0	0	0

2.2 Перетворення на основі спеціальної системи модулів

Розглянемо пряме перетворення для спеціальних модулів $2^n - 1$, 2^n і $2^n + 1$. Враховуючи, що арифметика за модулем 2^n є просто звичайною арифметикою, далі розглянемо лише модулі $2^n - 1$ та $2^n + 1$. Спеціальні модулі зазвичай називають недорогими модулями, оскільки перетворення до та з їхніх залишків може бути реалізовано відносно легко і не вимагає складних операцій, таких як обчислення мультиплікативних обернених, множення тощо [7].

Розглянемо обчислення залишку довільного цілого числа X за модулем m. Оскільки X можна представити як n – розрядне двійкове число, $x_n - 1\ x_n - 2\ \cdots\ x_0$, а його залишок по відношенню до m можна виразити як:

$$|X|_m = |x_{n-1}\ x_{n-2}\ x_{n-3} \cdots x_0|_m,$$

який є еквівалентний виразу:

$$|X|_m = |2^{n-1}x_{n-1} + 2^{n-1}x_{n-2} + 2^{n-1}x_{n-3} + \cdots + 2^{n-1}x_0|_m.$$

З властивостей залишків, отримаємо:

$$|X|_m = \left| \left| 2^{n-1} x_{n-1_m} \right| + |2^{n-1} x_{n-2}|_m + |2^{n-1} x_{n-3}|_m + \cdots |+2^{n-1} x_0|_m \right|_m$$

Оскільки x_i дорівнює або 0, або 1, отже для обчислення залишку X все, що потрібно, це обчислення значень $|2^i|_m$, які потім додаються за модулем.

1. Модуль $2^n - 1$. Перетворення та арифметику за модулем $2^n - 1$ досить легко здійснити. Наприклад, відносно звичайного (тобто доповнення до двох) двійкового додавання, єдиною «складністю» у додаванні за модулем $(2^n - 1)$ є необхідність іноді також додавати кінцеве перенесення. Тому можна реалізувати прості апаратні схеми з цією категорією модуля [2].

Залишки за $2^n - 1$ визначаються наступним чином:

$$|2^n|_{2^{n-1}} = |2^n - 1 + 1|_{2^{n-1}} = 1, \qquad (2.21)$$

Це рівняння можна легко розширити до добутку в степені 2^{nq} і, загалом, до довільного степеня 2:

$$|2^{nq}|_{2^{n-1}} = \left| \sum_{i=1}^{q} |2^n|_{2^{n-1}} \right|_{2^{n-1}} \qquad (2.3)$$

Отже, залишок будь-якого числа 2^m, де $m \neq n$, можна визначити за допомогою формули:

$$|2^{nq}|_{2^{n-1}} = |2^{nq+r}|_{2^{n-1}} = ||2^{nq}|_{2^{n-1}} \times |2^r|_{2^{n-1}}|_{2^{n-1}} = 1 \times |2^r|_{2^{n-1}}$$

де $q = \lfloor m/n \rfloor$, а r – залишок від ділення.

Розглянемо приклад. Нехай $X = 2^7$ і $m = 2^3 - 1$, тобто $n = 3$ і $m = 7$.

Тоді

$$|X|_7 = |2^7|_7 = |2^{2\times 3}|_7 \times ||2|_7|_7 = 2$$

У даному прикладі $q = 2$ і $r = 1$.

Отже, згідно з рівнянням 2.3,

$$|X|_7 = |1 \times 2|_7 = 2.$$

Даний підхід можна використовувати для обчислення залишків відносно іншого модуля, $2^n + 1$.

2. Модуль $2^n + 1$. Як і в попередньому випадку, спочатку розглянемо залишки 2^n відносно модуля $2^n + 1$:

$$|2^n|_{2^n+1} = |2^n + 1 - 1|_{2^n+1} = -1$$

Потім ми розширюємо це до довільного ступеня двійки, 2^m, де $m \neq n$ і

$$m = nq + r:$$

$$|2^m|_{2^n+1} = |2^{nq}|_{2^n+1} \times |2^r|_{2^n+1} = \begin{cases} 2^r - \text{якщо } q \text{ парне} \\ 2^r + 1 - 2^r - \text{інакше} \end{cases}$$

де $q = \lceil m/n \rceil$.

Якщо q непарне, $|2^{nq}|_{2^n+1}$ дорівнює -1, і тому $2^n + 1$ додають знову, щоб залишок став додатним.

Розглянемо приклад. Нехай $X = 2^7$ і $n = 4$, $m = 17$. Тоді q парне, і тому

$$|2^7|_{17} = |2^{4\times1}|_{17} \times |2^3|_{17} = 7$$

Тепер візьмемо $X = 2^7$; $n = 5$ і $m = 2^5 + 1 = 33$. Тоді q непарне, і тому

$$|2^7|_{33} = |2^{5\times1}|_{33} \times |2^2|_{33} = 2^5 + 1 - 4 = 31$$

На рисунку 2.7 показано організацію основного блоку для прямого перетворення, як описано вище. Модульні суматори можуть бути реалізовані як повністю в ПЗУ, комбінаційна логіка або комбінація обох схем.

Хоча конструкція на рисунку 2.7 є простою, модульні суматори повинні бути суматорами з повним перенесенням, і це може призвести до низької продуктивності. Існує кілька способів покращення продуктивності, але всі вимагатимуть додаткової логіки.

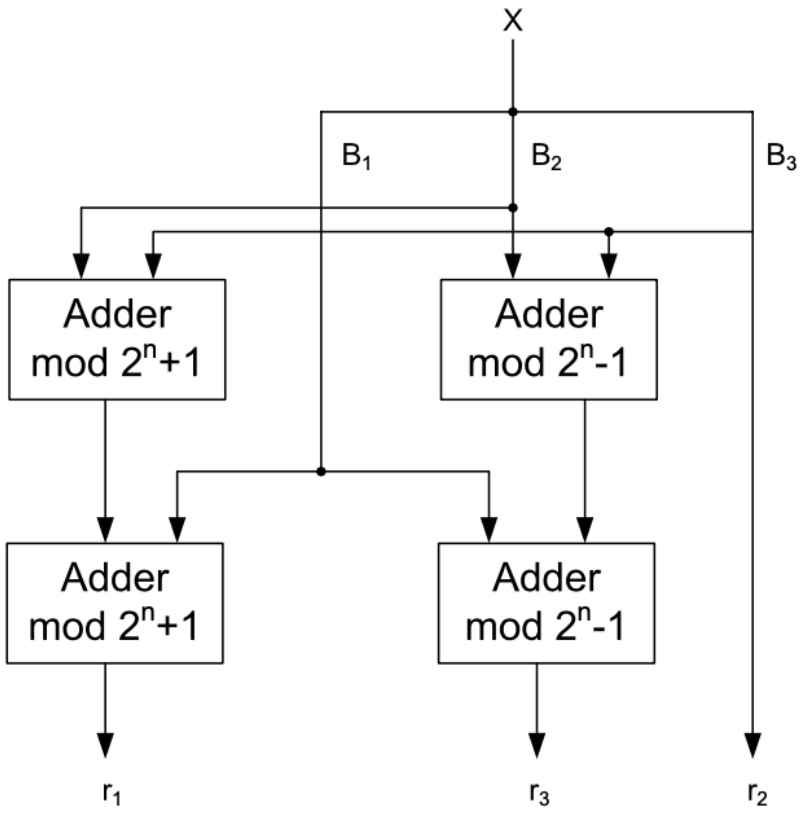


Рисунок 2.7 – Прямий перетворювач за модулями $\{2^n - 1, 2^n,\ 2^n + 1\}$

Існують дві основні техніки, які можна використати для досягнення кращої продуктивності: перша – це використання суматорів із збереженням переносу; а інший – це використання більшого паралелізму. На рисунку 2.8 показано пристрій, заснований на цих методах. Суматор із збереженням переносу (СЗП) приймає три операнди та створює два виходи, часткову суму (PS) і частковий перенос (PC), які, нарешті, повинні бути подані в суматор із поширенням

переносу (CPA), щоб перенесення передавались, для отримання результату, в звичайній формі. Обчислення $r1$ може вимагати коригувального віднімання модуля $m3$, а для $r3$ може знадобитися віднімання $m1$ або $2m1$. Усі різні можливі результати обчислюються паралельно, а правильний вибирається через мультиплексор. Слід зазначити, що хоча окремі суматори показані на рисунку 2.8, на практиці логіка суматора може бути спільною; тому тиражування буде меншим, ніж здається на перший погляд. Крім того, для певних високошвидкісних проектів СЗП і залежно від точності модулів відмінності в продуктивності між рисунком 2.7 і рисунком 2.8 можуть бути незначними.

2.3 Перетворення даних на основі розширеної системи спеціальних модулів

Описаний вище метод можна однаково добре застосувати до похідних 2^n множин модулів. Розглянемо два приклади наборів модулів у цій категорії: ті, що мають форму $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$, для непарних n, і ті, що мають форму $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$ для парних n. Вони часто зустрічаються в програмах, де потрібен великий динамічний діапазон. Використаємо, по суті, таку саму процедуру, яка описана вище; але при обчисленні залишку, що відповідає 2^{n+1}, двійкове представлення X тепер буде розділено на блоки по $n + 1$ біт кожен. Наступний приклад ілюструє процедури, які використовуються для обчислення залишків для кожного з двох наборів [5].

Приклад. Виберемо набір модулів {3, 4, 5, 7}. Модулі в цьому наборі є взаємно простими, а четвертий модуль має форму $2^{n+1} - 1$, з $n = 2$. Нехай X дорівнює 234. Щоб отримати залишки щодо 3, 4 і 5, ми діємо точно так само, як в останньому прикладі вище.

Ми розбиваємо двійкове представлення числа 234, тобто залишок по відношенню до 4 отримується простим зсувом двійкового еквівалента вправо на дві бітові позиції, при цьому 3 є зсунутим значенням бітів:

при $n = 2$:

$$|234\,|_3 = |11 + 10 + 10 + 10|_{2^2-1} = |3 + 1 + 1 + 1|_3 = 0.$$

Аналогічно, залишок по модулю 5 обчислюється як:

$$|234|_5 = |11 + 10 - 10 + 10|_{2^2+1} = |3 + 1 - 1 + 1|_5 = 4.$$

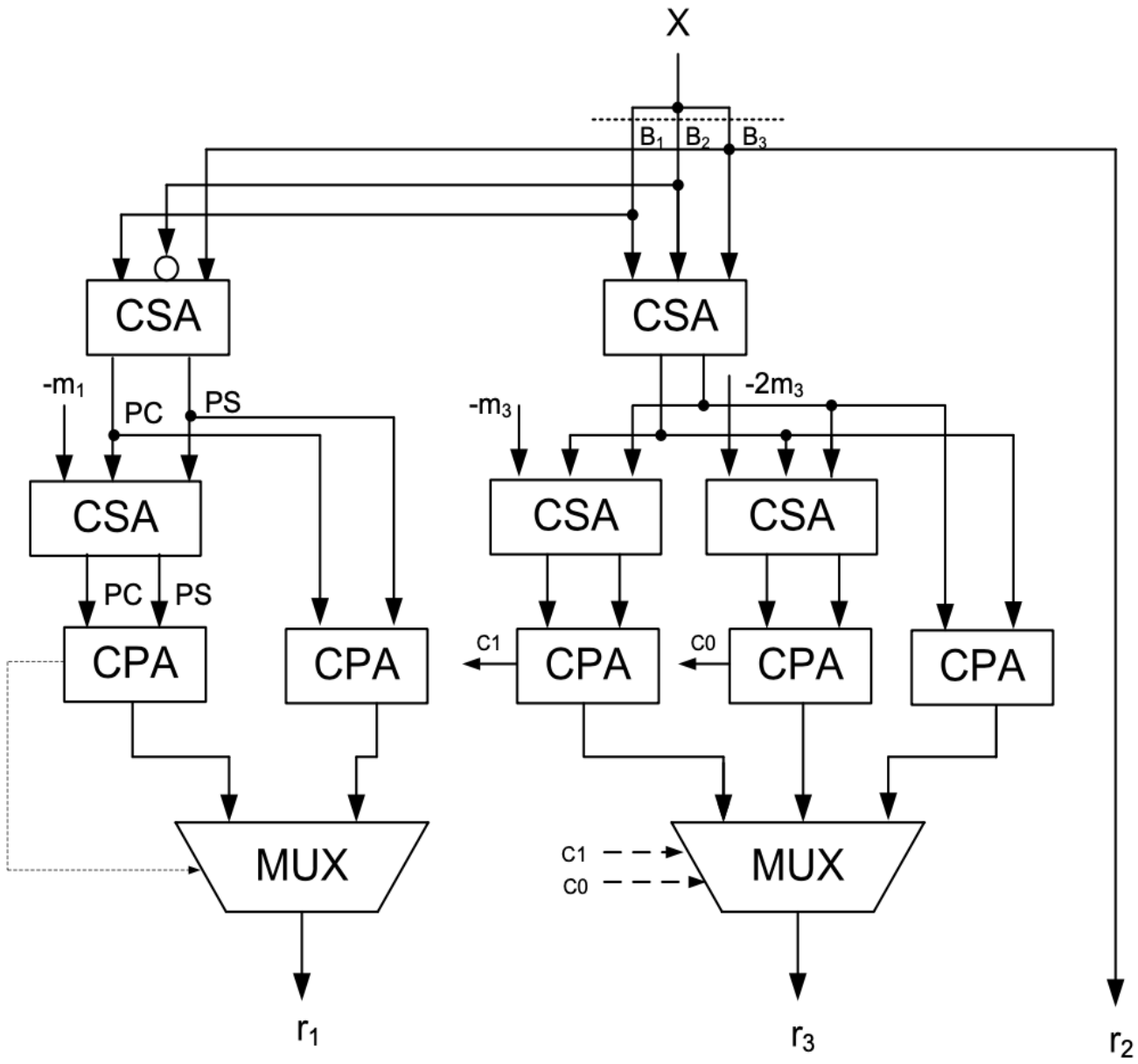


Рисунок 2.8 – Прямий перетворювач із підвищеною продуктивністю $\{2^n - 1, 2^n, 2^n + 1\}$

Щоб визначити залишок по відношенню до 7, ми розбиваємо двійкове представлення 234 на 3-бітові блоки, оскільки двійкове представлення 7 містить три біти:

$$|234|_7 = |011 + 101 + 010|_{2^3-1} = |3 + 5 + 2|_7 = 3$$

Розглянемо тепер подібний випадок, але з непарним n.

Приклад. Нехай $n = 3$. У цьому випадку набір модулів є формою $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$; тобто це буде $\{7, 8, 9, 17\}$.

$$|234|_{17} = |-1010 + 1110|_{17} = |-10 + 14|_{17} = 4$$

Залишки щодо інших модулів можна визначити, як показано в попередніх прикладах. Пряме перетворення в будь-якому розширеному наборі модулів, що складається з більш ніж чотирьох модулів, можна здійснити однаково легко шляхом розділення X відповідним чином для кожного модуля в наборі. Зі збільшенням кількості модулів у наборі складність схеми зростатиме лінійно. Складність можна зменшити, якщо перетворення виконується послідовно, тобто, якщо за один раз визначається один залишок, але це призведе до лінійного збільшення часу перетворення, із затримками, введеними в перетворювачі, що додасть накладних витрат на загальна система.

2.4 Висновки

Розглянуто табличний метод перетворення. Проведено розрахунок таблиці послідовного перетворення. Досліджено перетворення на основі спеціальної системи модулів, а також перетворення даних на основі розширеної системи спеціальних модулів

3 РЕАЛІЗАЦІЯ ПРОГРАМНО-ТЕХНІЧНОГО МОДУЛЯ ПЕРЕТВОРЕННЯ ДАНИХ В СИСТЕМУ ЗАЛИШКОВИХ КЛАСІВ НА БАЗІ FPGA

3.1 Обґрунтування вибору елементної бази

З часом світ електроніки пережив значний прогрес. Індустрія HBIC пройшла довгий шлях від транзисторів до інтегральних схем і ASIC. Далі з'явилися програмовані логічні пристрої, які зробили курс на стандартну промисловість безперервних напівпровідників. PLD з'явилися ще на початку 1970-х років, але тільки коли Xilinx представила FPGA наприкінці 1980-х років, PLD потрапили у світ ASIC [48].

FPGA – інтегральна схема, яку можна запрограмувати/налаштувати для функціонування відповідно до запланованого дизайну, маючи на увазі, що вона може функціонувати як мікропроцесор, блок шифрування, графічна карта або навіть усі ці три одночасно.

Конструкцію FPGA, що виконується як чіп, можна перепрограмувати для роботи як відеокарти в польових умовах, а не в ливарному виробництві напівпровідників.

Конструкція FPGA складається з тисяч настроюваних логічних блоків (CLB), встановлених у просторі програмованих з'єднань. Типова модель мікросхеми FPGA показана на рисунку 3.1. Мікросхема також складається з блоків введення/виведення, які розроблені та пронумеровані відповідно до функцій.

Програмовані з'єднання виконуються з горизонтальними та вертикальними каналами маршрутизації та PSM (програмованими мультиплексорами).

CLB складаються з таблиць пошуку (LUT) із фіксованою кількістю вхідних даних і побудовані на основі простої пам'яті, SRAM або Flash, які зберігають булеві функції. Таблиці пошуку (LUT) використовуються для виконання

генераторів функцій у CLB. Чотири незалежні входи надаються кожному з двох функціональних генераторів (F1-F4 і G1-G4) [48].

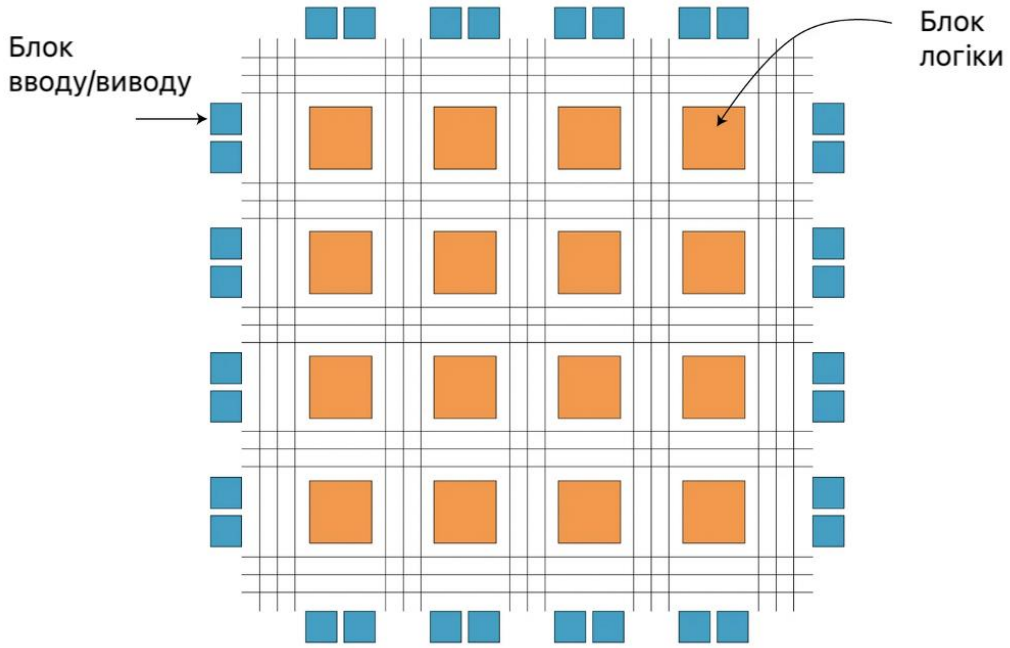


Рисунок 3.1 – Модель мікросхеми FPGA

Ці генератори функцій можуть виконувати будь-яку довільно охарактеризовану булеву функцію чотирьох входів. Для реалізації послідовних схем кожна LUT з'єднується з мультиплексором і тригером. Крім того, LUT можна комбінувати для виконання складних логічних функцій.

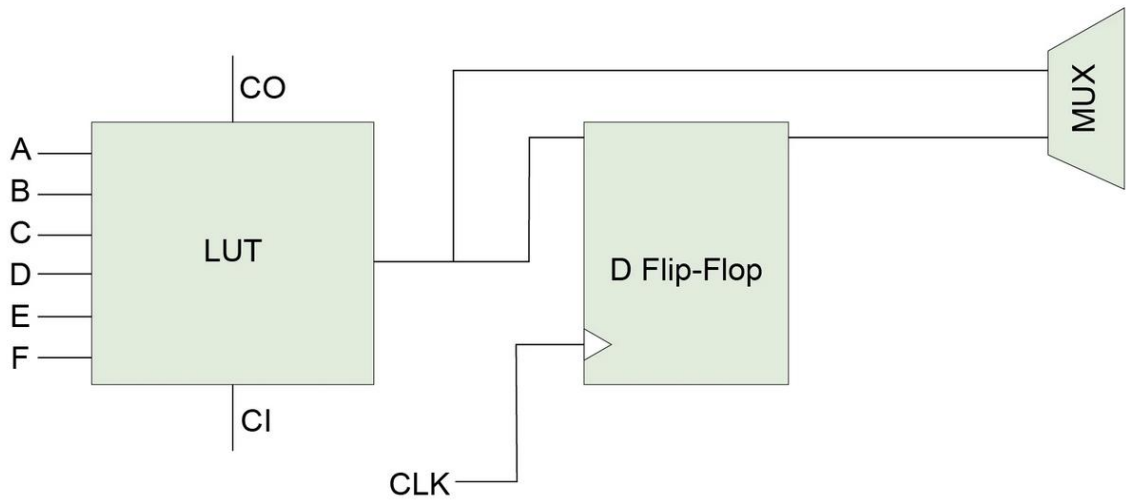


Рисунок 3.2 – Структура FPGA

Крім CLB і напрямних міжконнектів, численні FPGA також містять виділені жорсткі кремнієві блоки різної ємності, наприклад блоки DSP, блоки RAM, контролери зовнішньої пам’яті, PLL тощо. Сучасні FPGA є потужними пристроями, що підтримують багато стандартів введення/виведення, наприклад I2C, SPI, CAN, PCIe. Вводи/виводи в FPGA згруповані в банки, де кожен банк може автономно підтримувати різні стандарти вводу/виводу.

Крім того, практикується новий шаблон забезпечення процесорного центру на твердому кремнії (наприклад, ARM Cortex A9 у Xilinx Zynq) всередині тієї ж FPGA. Це дає змогу процесору виконувати звичайні, нестандартні завдання.

Кількість CLB визначає складність конструкції FPGA. Функціональність CLB і PSM розроблена Verilog, VHDL або будь-якою іншою мовою опису обладнання. Синтезатор постачальника використовує опис апаратного забезпечення, описаний розробником, щоб виявити оптимізовану схему дій активів FPGA, які виконують певні функції. Після програмування CLB і PSM розміщуються на чіпі та зв’язуються один з одним за допомогою каналів маршрутизації.

Є три головні причини вибрати дизайн FPGA замість дизайну ASIC. Перш ніж перейти дальше необхідно зрозуміти фундаментальну та критичну різницю між ASIC та FPGA Design через аналогію.

Розглянемо можливість створення палацу з блоків Lego та створення палацу з бетону. Перший аналогічний FPGA, тоді як другий аналогічний ASIC. Можна повторно використати блоки Lego для створення іншого дизайну, але бетонний палац вічний.

Існує три вагомі переваги використання дизайну FPGA перед мікропроцесором, таким як інтегральна схема, призначена для конкретного застосування (ASIC), у прототипі або в обмеженому виробництві, які позначені нижче [49].

1. Гнучкість і продуктивність. Там, де продуктивність є головною вимогою, ПЛІС залишаються в стороні в глибоко паралелізованих завданнях. Сучасні мікропроцесори виконуються на багатьох ядрах із нестандартними інструкціями,

що не дуже підходить для таких функцій, як додатки обробки великих зображень або цифрових сигналів.

Крім того, FPGA може мати багато апаратних або програмних мікропроцесорів, що працюють в одному пакеті.

Проект FPGA має більш простий цикл проектування для керування та вимагає менше ручної роботи. Програмне забезпечення автоматично обробляє більшу частину маршрутизації, розміщення та синхронізації відповідно до запрограмованих специфікацій.

2. Можливість повторного використання. Оскільки FPGA можна перепрограмувати, їх можна багаторазово використовувати, що робить їх придатними для швидкого створення прототипів, а помилки не коштують дорого.

Функція реконфігурації стандартної FPGA має переваги порівняно з ASIC. Незважаючи на складні IP-центри, розроблені для конкретного додатка, фактична цінність полягає в тому, щоб мати можливість переналаштувати (і знову налаштувати) після встановлення – те, чого ASIC не можуть зробити.

Швидке визнання – FPGA швидше виходить на ринок, оскільки вони не розроблені заздалегідь для виконання конкретних завдань. Можна придбати готову ПЛІС, а потім налаштувати її під потрібний дизайн.

Розглянемо деякі характеристики FPGA, які роблять їх кращим вибором для засобу проектування обчислювальних пристроїв.

Паралельна обробка. Одна з переваг FPGA, яка робить їх гідним вибором для роботи з вимірювальними системами та іншими обчислювальними програмами, які потребують обробки великої кількості даних, полягає в тому, що вони можуть обробляти паралельно. Навпаки, процесори працюють послідовно, що робить їх не практичним вибором.

ПЛІС виконують критичну за часом обробку. Маючи можливість паралельно обробляти дані, ПЛІС мають низьку затримку, що робить їх придатними для додатків, які потребують дуже критичних за часом обчислень, наприклад, у програмно визначених радіостанціях, медичних пристроях і авіаційних системах.

FPGA мають оптимальну продуктивність. У порівнянні з процесором або графічним процесором продуктивність на ват буде кращою з FPGA. Це низьке енергоспоживання може бути майже в 3-4 рази менше, ніж у процесора. Експлуатаційна вартість ASIC є, на жаль, найкращою, але висока початкова ціна значною мірою врівноважує це.

Процес розробки FPGA. Програмування FPGA – це планування, проектування та впровадження рішення щодо розробки FPGA. Все починається з ідеї, а потім здійснюється перетворення ідеї на код. Потім код RTL моделюється та налагоджується для функціональної перевірки. Після кожної перевірки код готовий для завантаження в проект FPGA.

Процес розробки дизайну FPGA можна умовно розділити на три етапи: проектування, перевірка та впровадження [48].

Основна задача полягає в тому, щоб перенести нашу ідею чи думку в фактичний пристрій FPGA на етапі проектування. Це включає в себе звичайну архітектуру чіпа або поділ його на менші блоки для формування цілого дизайну. Тоді можна реалізувати кожен із цих блоків, використовуючи мову HDL або іншу методологію. На етапах реалізації проект HDL перетворюється на програмний файл, який далі розміщується на проекті FPGA.

Послідовність проектування FPGA. Послідовність проектування FPGA складається з наступних кроків, включаючи введення проекту, синтез, впровадження та програмування пристрою. Нижче докладно пояснюються всі ці кроки, які беруть участь у процесі проектування.

Проектування введення даних. Введення проекту можна виконати за допомогою схем або мови опису обладнання (HDL). Доцільно також поєднувати їх обидва та використовувати найкраще з обох за допомогою інструментів, які можуть перетворювати схему в HDL і навпаки. Як правило, краще вибрати HDL для конструкції, яка більше стосується складних систем. Цей швидший мовний процес позбавляє вас від необхідності проектувати на апаратному забезпеченні нижчого рівня.

Схематичний підхід є хорошим вибором для дизайнерів, які займаються проектуванням обладнання, оскільки схематичний підхід дає їм більше видимості.

У кожного підходу є переваги та недоліки. Хоча техніку, засновану на схемі, легше прочитати та зрозуміти, вона працює лише з більш незначними проектами. З іншого боку, методи на основі HDL, як правило, швидкі та прості у впровадженні, і на сьогоднішній день є найпопулярнішим підходом при проектування FPGA.

Синтез. Після того, як проект було введено в код, проект синтезується, де код перетворюється на схему, яка містить затвори, тригери, помножувачі тощо. HDL перетворюється на список з'єднань, який включає логічні елементи та з'єднання необхідні в конкретній ієрархії.

Процес починається з перевірки синтаксису дизайну на основі HDL. Потім його оптимізують шляхом зменшення логіки, усунення надлишкової логіки та зменшення розміру дизайну. На останньому кроці дизайн з'єднується з логікою шляхом відображення технології (рисунок 3.3).

```
module and2 (c, b, a);
output c; input a, b;
assign c=a&b;
endmodule
```

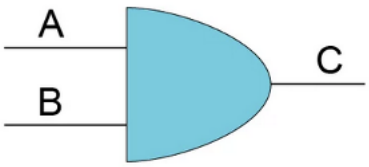


Рисунок 3.3 – Приклад опису логічного елементу І: а) опис логічного елементу І мовою Verilog; б) синтезований елемент І

Синтез FPGA виконують з використанням спеціальних інструментів. Багато компаній, такі як Cadence, Synopsys і Mentor Graphics, розробляють інструменти для синтезу FPGA.

На етапі впровадження визначається макет дизайну, який складається з трьох кроків: переклад, карта та розміщення та маршрут. Постачальник FPGA

надає інструмент, який використовується на цьому кроці, оскільки він знає, як перевести створений список з'єднань у FPGA.

Першим кроком для інструментів є збір усіх обмежень, встановлених користувачем, і файлів списку мереж. Ці обмеження містять положення контактів вводу-виводу та можуть включати вимоги щодо синхронізації, такі як максимальна затримка, помилковий шлях тощо.

Потім інструмент визначає реалізацію, порівнюючи вимоги до ресурсів, указані у файлах, із ресурсами, доступними для використовуваного дизайну FPGA. Схема розділена на логічні блоки або елементи у вигляді підблоків. У результаті весь ваш дизайн розміщується в певних логічних блоках і «відображається» в дизайні FPGA.

Наступним кроком є підключення та маршрутизація всіх сигналів відповідно до обмежень, встановлених користувачем.

Пам'ять програм FPGA. Після того, як закінчили розробку FPGA, ми створюємо програмний файл, який повідомляє про те, як налаштувати FPGA.

Проект FPGA використовує цей файл для налаштування внутрішніх LUT і взаємозв'язків, а також інших внутрішніх компонентів, таких як PLL, ядра DSP, DDR, BRAM тощо.

Сучасні пристрої FPGA використовують пам'ять типу SRAM для зберігання цієї інформації через її високу швидкість. Одним із недоліків використання пам'яті типу SRAM є те, що для зберігання програми потрібна зовнішня мікросхема пам'яті, оскільки SRAM є енергозалежною. SRAM не може зберігати свій стан після вимкнення живлення. Щоб вирішити цю проблему, деякі постачальники пропонують дизайн FPGA, який використовує флеш-пам'ять замість SRAM, оскільки цей тип пам'яті є енергонезалежним.

Однак FPGA на основі SRAM зазвичай може працювати з вищою тактовою частотою, тому вони більш популярні.

Процес перевірки FPGA. Після написання коду HDL виникає вимога його перевірити. Цей процес називається перевіркою.

Першим етапом перевірки конструкції є моделювання. Для моделювання дизайну створюється тестовий стенд для створення вхідних даних для дизайну.

Потім відповідно до функціональності перевіряються виходи проекту FPGA. Якщо результат відповідає очікуванням, то вважається, що моделювання пройдено.

Як правило, моделювання є основним процесом, який бере участь у перевірці розробленого проекту. Зазвичай ми доповнюємо це тестуванням апаратного забезпечення, щоб переконатися, що інтерфейси FPGA належним чином працюють із усіма зовнішніми схемами.

Однак, оскільки дизайн FPGA став більш складним, інші методи стали популярними. Більш сучасні види верифікації включають апаратне забезпечення в циклі (HiL) і емуляцію.

Це передбачає запуск коду на нашому цільовому пристрої та передачу даних програмному забезпеченню моделювання в обох випадках. Це дозволяє нам запускати специфічні структуровані тести на нашому пристрої майже в режимі реального часу.

Інтеграція FPGA. Передостанній крок називається системною інтеграцією перед фактичним проектуванням. Типова цифрова система, розроблена з використанням FPGA, не складається з однієї конструкції FPGA, а натомість містить кілька FPGA, логіку зв’язку, таку як PDL або TTL частини, та інші компоненти, такі як EPROM та SRAM. Модель близька до реальної структури та моделювання показує, що фізичний прототип працює добре [50].

Дизайн HDL. Початковим кроком є специфікація системи, а потім система розділяється на дві частини: програмне забезпечення та апаратне забезпечення. Апаратне забезпечення може бути ASIC, FPGA Design або PLD (програмований логічний пристрій). Крім того, для системи програмне забезпечення діє як обладнання та змушує його працювати належним чином [49].

Фундаментальним обмеженням мови HDL є те, що вона не може працювати з аналоговими сигналами, вона може приймати лише цифрові системи. Крім того, HDL-моделі можуть бути ізольовані в різних абстракціях, таких як HDL

(абстракція найвищого рівня), потім рівень воріт, Netlist і Layout (найнижчий рівень абстракції).

Ієрархію HDL можна розділити на різні рівні. Верхній модуль можна розділити на два різноманітних підмодулі, тоді як підмодуль можна додатково розділити на модулі базового рівня.

Вибір мови опису апаратних засобів. Тепер існує два різних типи мови HDL. Дизайнери мають різні думки щодо того, яка мова краща, але справа не в тому, яка мова краща; замість цього мова йде про те, якій мові ви віддаєте перевагу. Необхідно також уточнити, що існує System Verilog, але він дуже тісно пов'язаний з Verilog.

VHDL означає Very High-Speed Integrated Circuit Hardware Description Language. Однією з важливих особливостей мови VHDL є те, що це строго типізована мова, що означає, що сам VHDL має попередньо визначені типи даних (цілі числа, символи тощо). Усі значення або змінні, визначені цією мовою, описуються одним із типів даних.

VHDL є більш деталізованою, ніж Verilog, і додатково має синтаксис, відмінний від C.

Verilog є більш компактною мовою, оскільки це більше мова моделювання апаратного забезпечення. Verilog краще володіє апаратним моделюванням, але має нижчий рівень програмних конструкцій.

Verilog не такий довгий, як VHDL, тому він більш компактний. Verilog унікальний щодо VHDL. Є кілька подібностей, однак у них переважають відмінності.

3.2 Розробка модуля перетворення даних з використанням мови Verilog

Програмно - технічний модуль перетворення даних з двійкової системи числення в СЗК має забезпечувати перетворення вхідних даних в діапазоні від 0 до N, де $N < 2^{24}$. Відповідно, щоб забезпечити вказаний діапазон необхідно вибрати модулі добуток яких $P = \sum_{i=1}^{5} p_i \geq 2^{24}$.

Отже, щоб забезпечити даний робочий діапазон виберемо наступні модулі:

$p_1 = 7;\ p_2 = 23;\ p_3 = 29;\ p_4 = 59;\ p_5 = 61,$ їх добуток $P = 7 * 23 * 29 * 59 * 61 = 16803731 > 2^{24}$.

Реалізуємо модуль перетворення даних з двійкової системи числення в СЗК засобами мови Verilog.

Програмний код модуля перетворення на мові Verilog нає вигляд:

```
module convert_24_5 (input[23:0]A,
// оголошення входів, виходів та їх розрядності;
output [2:0]b1,
output [4:0]b2,
output [4:0]b3,
output [5:0]b4,
output [5:0]b5);
parameter  p1=7, p2=23, p3=29, p4=59, p5=61;
assign b1=A%p1;
assign b2=A%p2;
assign b3=A%p3;
assign b4=A%p4;
assign b5=A%p5;
endmodule
```

Симуляцію розробленого модуля виконаємо засобами Quartus Prime Lite Edition. Існують додаткові програмні інструменти сторонніх розробників, що підключаються до Qaurtus. Один з лідерів засобів тестування електронних проектів є компанія Mentor Graphics.

Симуляція – це програмне тестування проекту, яке завжди виконується до його запису в FPGA. Наприклад, якщо ми поставимо, послідовність і тривалість вхідних імпульсів, то система проектування зможе розрахувати результуючі сигнали на всіх своїх виходах. За допомогою симуляції можна заглянути всередину всіх модулів проекту і подивитися на процеси, що відбуваються

всередині мікросхеми. Quartus Prime Lite Edition надає наступні можливості для проведення моделювання за допомогою САПР сторонніх фірм:

– інтерфейс зв'язку Native Link з іншими САПР;

– генерація вихідного файлу зв’язків;

– бібліотеки для функціонального і часового моделювання;

– оцінка споживаної потужності утилітою Power Gauge;

– генерація тестового файлу та файлу ініціалізації пам'яті.

Результати компіляції модуля перетворення даних приведені на рисунку 3.4.

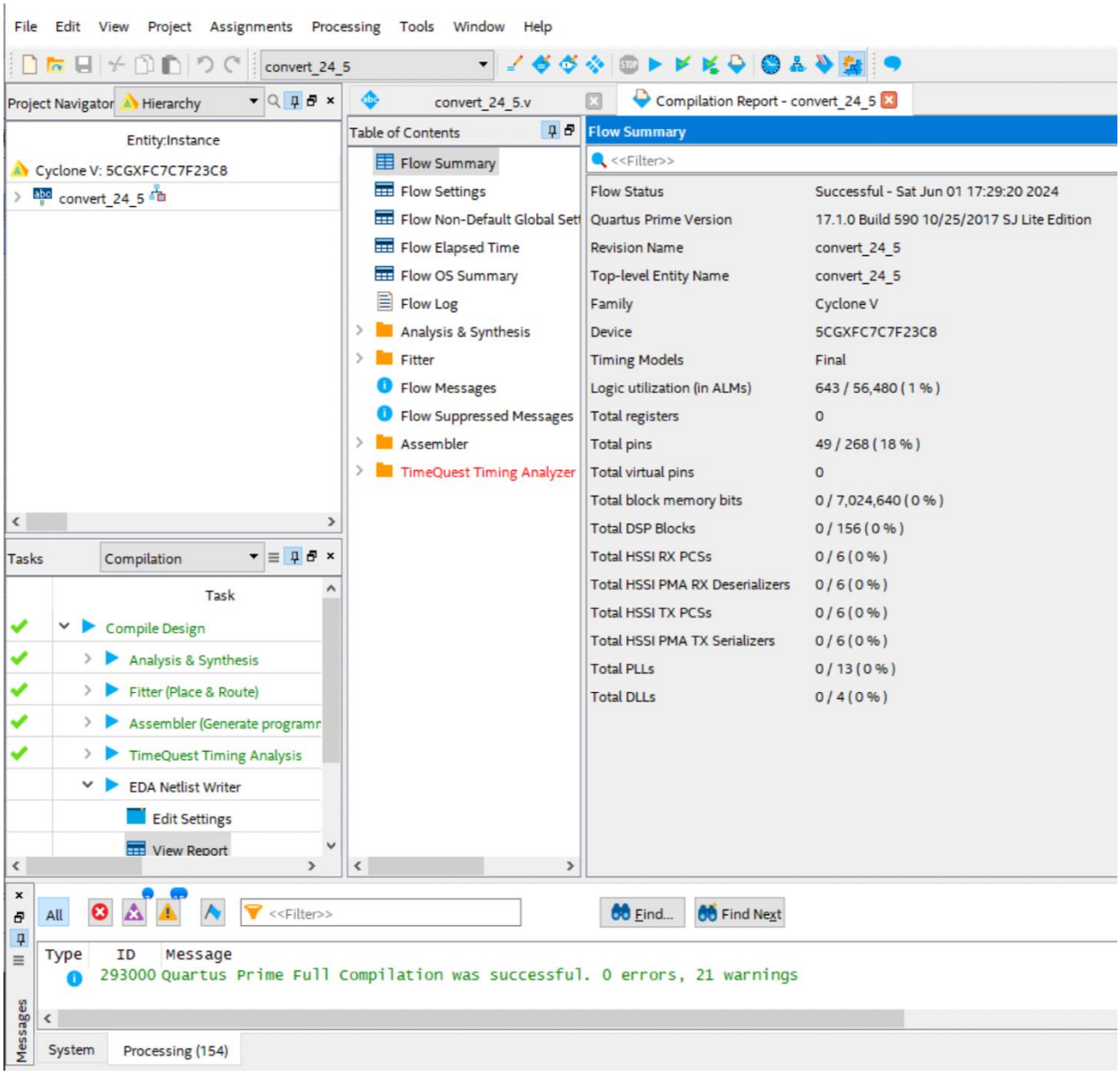


Рисунок 3.4 – Результати компіляції блоку

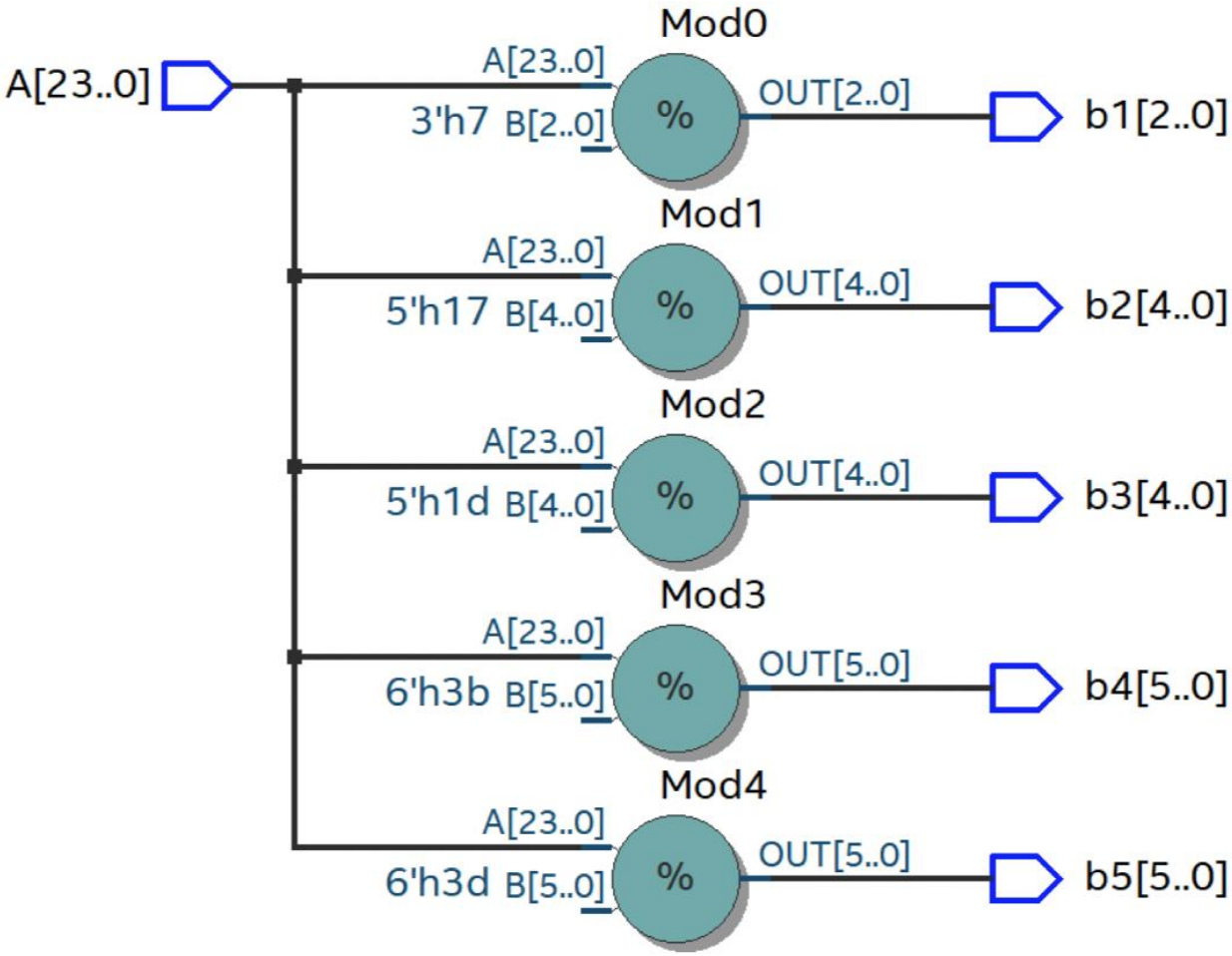


Рисунок 3.5 – Функціональна схема модуля перетворення

Інтерфейс модуля перетворення приведений на рисунку 3.6.

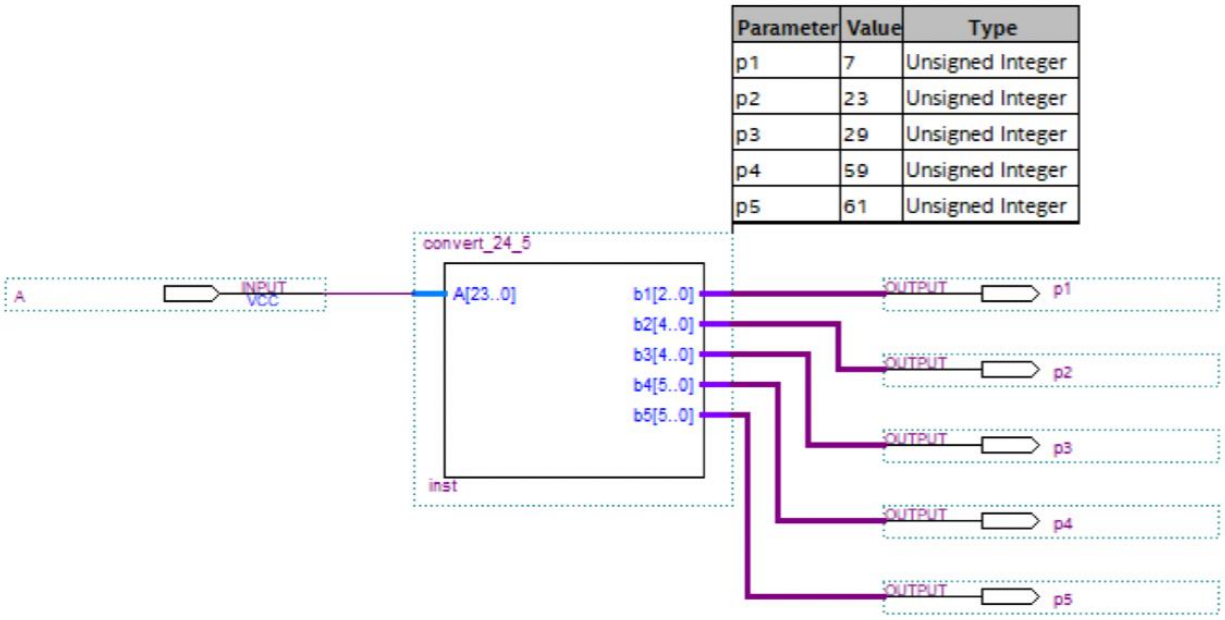


Рисунок 3.6 – Інтерфейс модуля перетворення

Пристрій реалізований на FPGA фірми Intel серії Cyclon V, мікросхема 5CGXFC7C7F23C8, час перетворення 24 - бітного двійкового коду в код системи залишкових класів становить 52,174 нс, загальна кількість логічних елементів – 634 та задіяні 49 виводи.

Перевірка роботи проекту складається з наступних кроків.

Крок 1. Налаштування програми моделювання.

Відкриваємо проект convert_24_5.qpf з робочої директорії.

В меню Assignments вибираємо команду Settings.

В категорії Simulator Settings встановлюємо режим Functional у випадаючому меню вікна Simulation mode.

Крок 2. Створення часової діаграми вхідних сигналів *.vwf файлу.

В меню File вибираємо команду New. У діалоговому вікні New переходимо до закладки Verification/Debugging Files і вибираємо University Program VWF (рисунок 3.7)

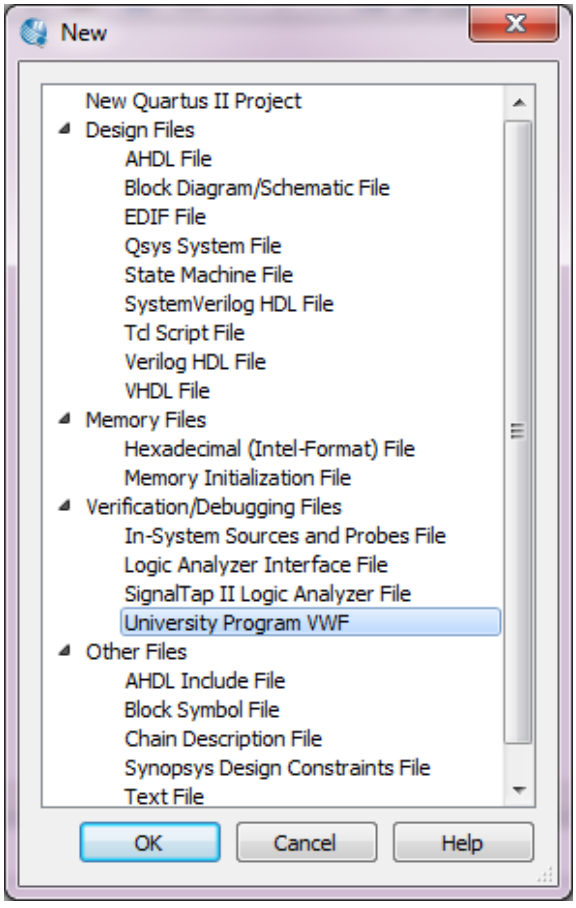


Рисунок 3.7 – Створення часової діаграми

Після чого відкриється вікно налаштування часової діаграми (рисунок 3.8):

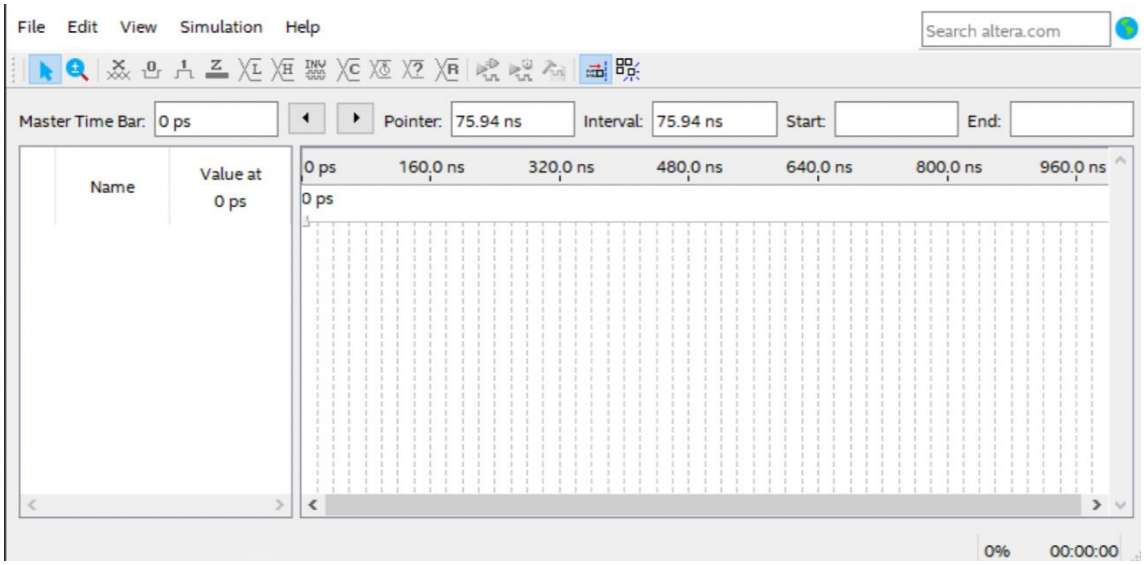


Рисунок 3.8 – Форма для створення часової діаграми

Крок 3. Створення входів та виходів модуля

В меню Edit вибираємо підміню Insert, а в ньому команду Insert Node or Bus. У діалоговому вікні Insert Node or Bus натискаємо кнопку Node Finder.

У вікні Node Finder встановлюємо режим Pins: all у випадаючому меню віконця Filter. Натискаємо кнопку List.

Виділяємо сигнали b01, b02, b03, b04, b05 і s. Натисніть кнопку “>” для перенесення виділених сигналів у вікно Selected Nodes. Натискаємо OK. Натискаємо ще раз OK в діалоговому вікні Insert Node or Bus.

У файлі *.vwf виділяємо сигнали b1, b2, b3, b4, b5 і A. Натискаємо праву кнопку миші і вибираємо команду Properties. Після цього змінюємо формат представлення сигналів з Binary на Unsigned Decimal. Натискаємо OK.

У меню Edit вибираємо команду End Time. У діалоговому вікні End Time встановите значення параметра Time рівне 10 us. Натискаємо OK.

Крок 4. Формування вхідного сигналу

У файлі *.vwf повинні відображатися всі обрані сигнали. Задаємо значення вхідного сигналу так, як показано на рисунку 3.9.

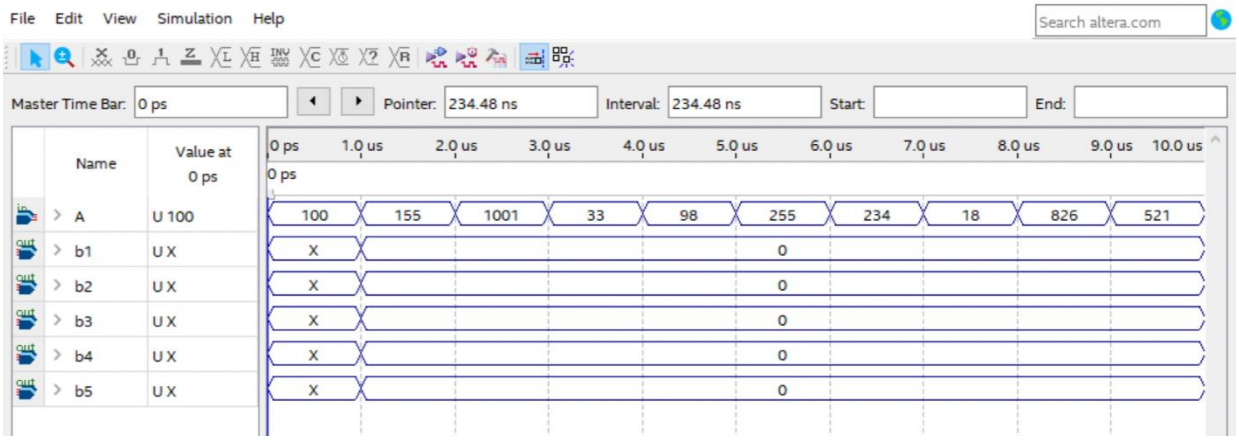


Рисунок 3.9 – Формування вхідного сигналу

Крок 5. Запуск програми моделювання

Зберігаємо тестовий файл під ім'ям conver_24_5.vwf (vector waveform file).

В меню Processing вибираємо команду Start Simulation. Після закінчення моделювання з'являється повідомлення "Simulation was Successful". Натискаємо OK.

Крок 6. Аналіз результатів.

Вікно Simulation Report відкриється автоматично після запуску програми моделювання (рисунок 3.4). В результаті моделювання отримали вихідні сигнали на виходах b1, b2, b3, b4, b5, які відповідають залишкам від ділення A на задані модулі (рисунок 3.10) .

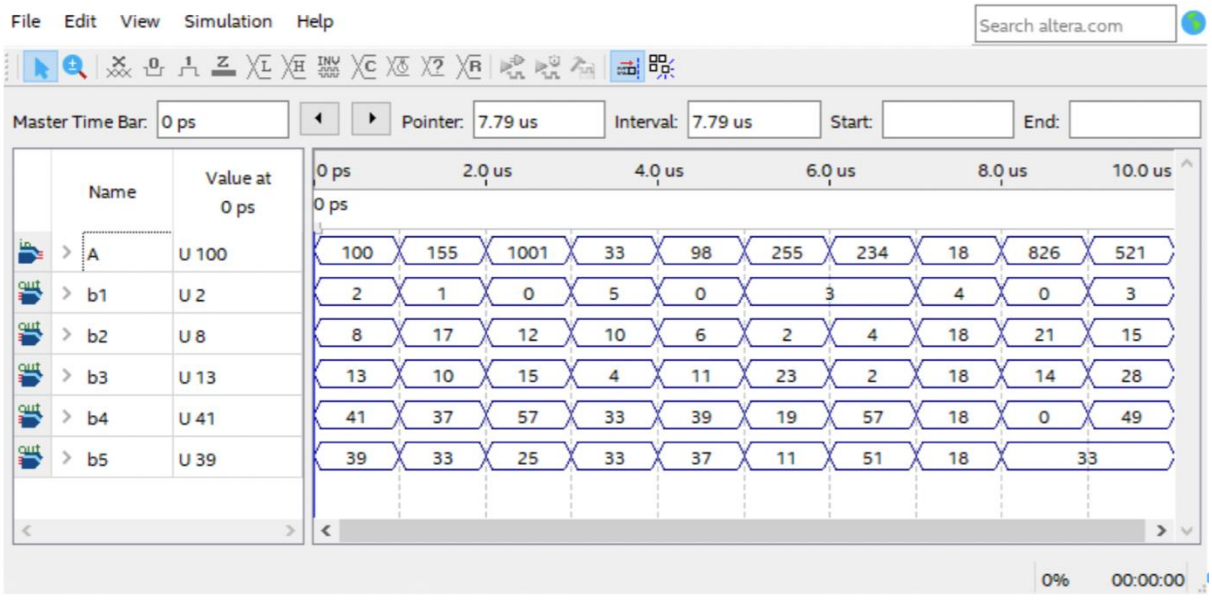


Рисунок 3.10 – Результати моделювання

Результати моделювання засвідчують коректну роботу модуля перетворення. Перевірити правильність роботи модуля можна обчисливши залишки від вхідного сигналу A по відповідних модулях.

Наприклад. Нехай A=155, тоді:

$b1 = 155\ (mod\ 7) = 1;$

$b2 = 155\ (mod\ 23) = 17;$

$b3 = 155\ (mod\ 29) = 10;$

$b4 = 155\ (mod\ 59) = 37;$

$b5 = 155\ (mod\ 61) =$33.

Отримані значення відповідають результату отриманому при верифікації схеми (рисунок 3.10).

3.3 Порівняння двох методів реалізації модуля перетворення

Для реалізації перетворювача з двійкової системи числення в систему залишкових класів вибрано модулі: $m1 = 5, m2 = 7, m3 = 8,$ які забезпечують робочий діапазон $M = m1 \times m2 \times m3 = 5 \times 7 \times 8 = 280.$ Даний набір модулів дозволяє опрацьовувати числа в діапазоні від 0 до 279, що забезпечує роботу з байтами.

Реалізація перетворення двійкового коду в код СЗК на мові Verilog має вигляд [19, 20]:

```
module convert_8 (input[7:0]A, output [2:0]b1,b2,b3);
parameter  m1=5, m2=7, m3=8;
assign b1=A%p1;
assign b2=A%p2;
assign b3 = A%p3;
endmodule
```

Результати компіляції модуля перетворення даних приведені на рисунку 3.11.

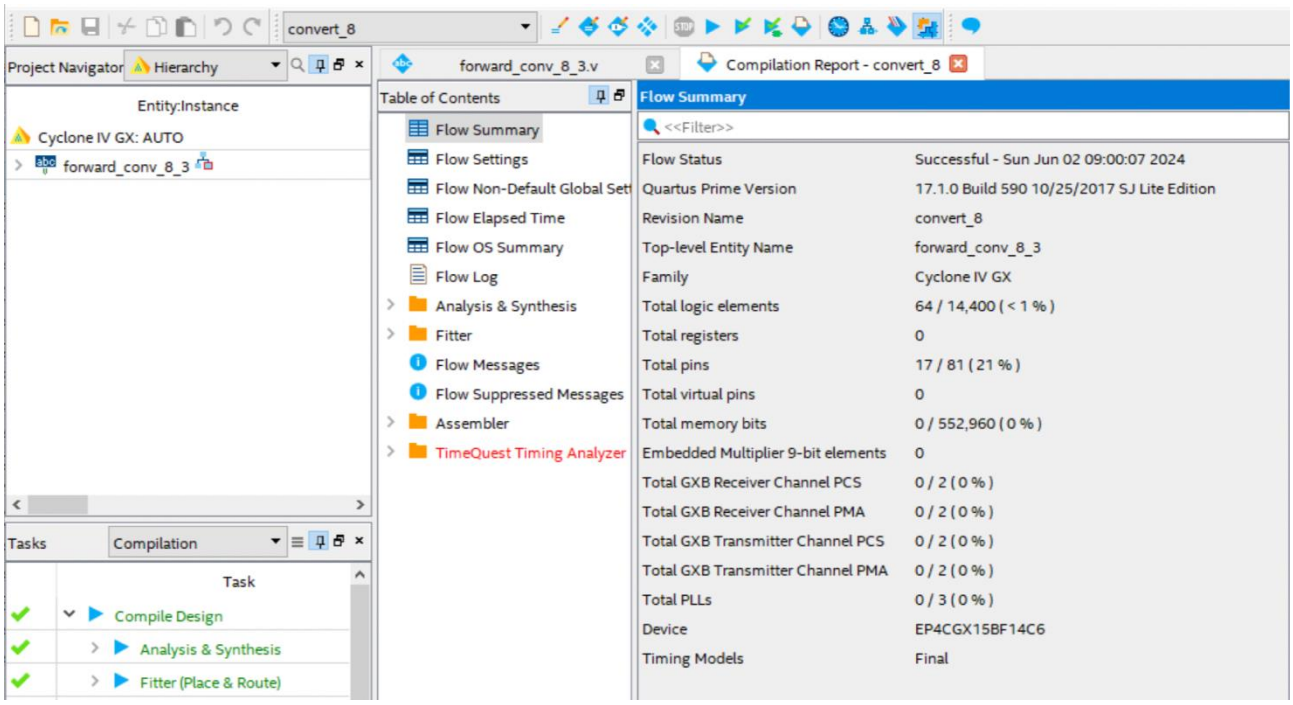


Рисунок 3.11 – Результати компіляції модуля

Результати моделювання блоку перетворення, які підтверджують його коректну роботу приведені на рисунку 3.12.

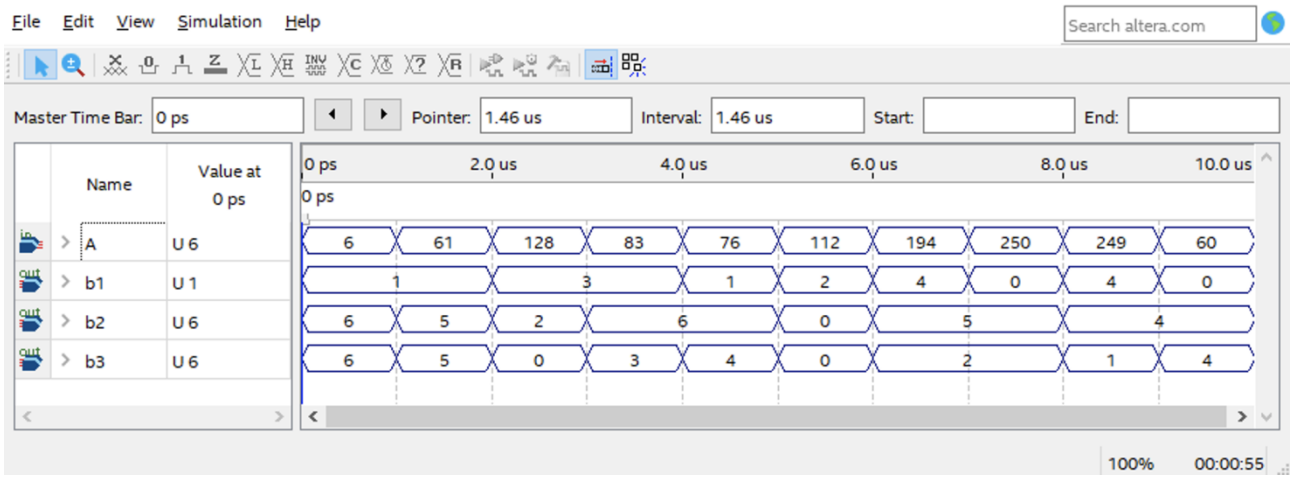


Рисунок 3.12 – Результати моделювання роботи блоку перетворення

Використання модуля типу 2^n спрощує побудову пееретворювача, так як залишок по даному модулю є n молодших розрядів двійкового числа (рисунок 3.13) [2, 23]. Блок перетворення двійкового коду в код СЗК описаний на мові Verilog і реалізований на ПЛІС фірми Altera Family Cyclone IV GX, Device

EP4CGX15BF14C6 (рисунок 3.14). Апаратні затрати для перетворення 8-бітного двійкового коду становлять 64 Total logic elements, час перетворення становить 2 нс.

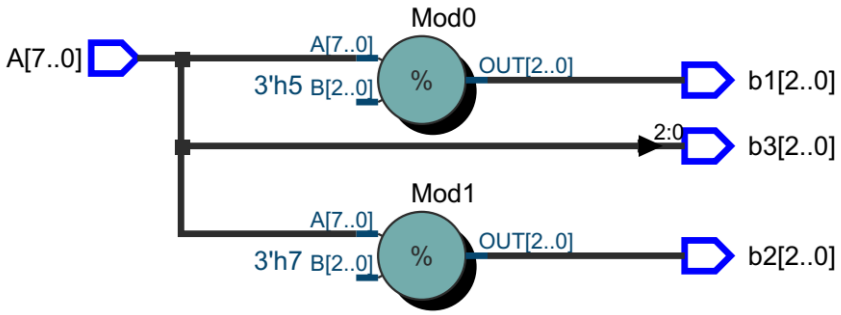


Рисунок 3.13 – Функціональна схема

Можливі набори модулів для різної розрядності вхідних даних приведені в таблиці 3.1.

Таблиця 3.1 – Набори модулів для перетворенняя даних в СЗК

Розрядність вхідних даних	Набір модулів	Значення модулів	Робочий діапазон
8	1	3, 8, 11	$P_n = 263$
	2	5, 7, 8	$P_n = 280$
16	1	29, 43, 53	$P_n = 66091$
	2	3, 8, 11, 13, 23	$P_n = 78936$
24	1	233, 239, 307	$P_n = 17095909$
	2	7, 23, 29, 59, 67	$P_n = 18456557$
32	1	241, 257, 263, 271	$P_n = 4414435801$
	2	19, 29, 37, 43, 59, 89	P_n =4603241891

Проведені експериментальні дослідження апаратної складності блоку перетворення з двійкового коду в СЗК при різних наборах модулів, приведених в таблиці 3.1 (рисунок 3.14, рисунок 3.15).

Як видно з рисунку 3.14 комбінаційний метод (Method 2) має меншу апаратну складність при меншій розрядності модулів а використання операції отримання залишку (%) мови Verilog (Method 1) показує кращі результати при більшій розрядності модулів (рисунок 3.14). Однак при розрядності модулів 7 біт апаратні затрати при використанні комбінаційного методу (Method 2) будуть більші порівняно з Method 1 (рисунок 3.15).

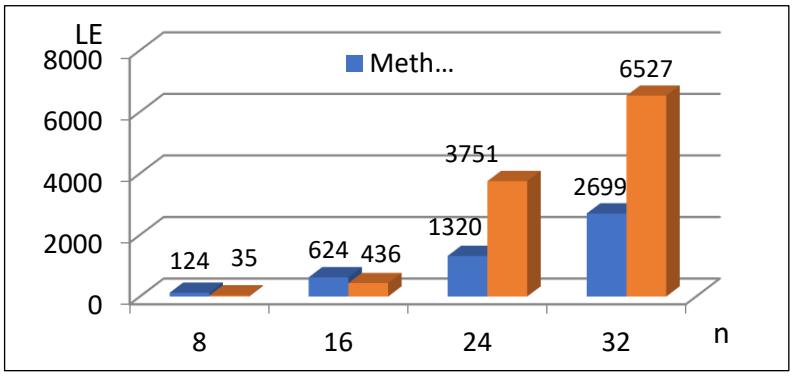


Рисунок 3.14 – Апаратна складність перетворювача з двійкового коду в

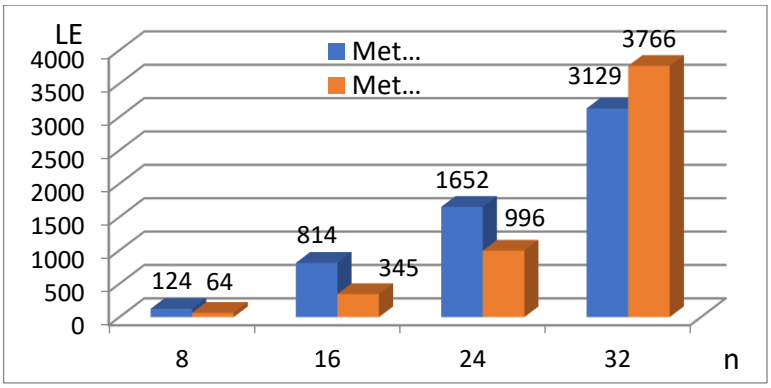


Рисунок 3.15 – Апаратна складність перетворювача з двійкового коду в СЗК при використанні набору модулів 2 (таблиця 3.1).

3.4 Висновки

Реалізовано перетворення двійкового коду в код СЗК на мові Verilog. Проведені експериментальні дослідження апаратної складності блоку перетворення з двійкового коду в СЗК при різних наборах модулів

ВИСНОВКИ

В роботі розв’язано актуальну задачу розробки модуля перетворення даних з двійкової системи числення в систему залишкових класів на програмованих логічних інтегральних схемах. При цьому отримано наступні результати.

1. Проведено аналіз та дослідження існуючих методів переведення даних з двійкової системи числення в систему залишкових класів. Проведені дослідження дають можливість вибрати оптимальний метод переведення чисел з двійкової системи числення в систему залишкових класів, в залежності від діапазону вхідних чисел, вимог до часу переведення та апаратних затрат.

2. Проаналізовано існуючі методи виконання арифметичних операцій в системі залишкових класів та показано переваги використання табличних методів, які забезпечують більшу швидкодію.

3. Обґрунтовано вибір програмованих логічних інтегральних схем в якості елементної бази для реалізації модуля перетворення даних з двійкової системи числення в систему залишкових класів.

4. Розроблено перетворювач 24- розрядного двійкового коду в код системи залишкових класів, який забезпечує високу швидкодію за рахунок використання таблиць. Пристрій реалізований на FPGA фірми ALTERA серії Cyclone, мікросхема 5CGXFC7C7F23C8, час перетворення 24 - бітного двійкового коду в код системи залишкових класів становить 50,268 нс, загальна кількість логічних елементів – 634 та задіяні 49 виводи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Piestrak S. J. A high-speed realization of residue to binary system converter. IEEE Trans. Circuits Syst. II Analog. Digit. Signal Process, 1995. Vol. 42, no. 10, Pp. 661–663.

2. Dhurkadas A. Comments on ‘A high-speed realization of a residue to binary number system converter. IEEE Trans. Circuits. Syst. II Analog. Digit. Signal Process. 1998. Vol. 45, no. 3. Pp. 446–447.

3. Wang Z., Jullien G. A., and Miller W. C. An improved residue to binary converter. *IEEE Trans. Circuits Syst*. I, 2000. Vol. 47, Pp. 1437–1440.

4. Wang Y., Song X., Aboulhamid M. and Shen H.. Adder- based residue to binary number converters for $\{2^n–1, 2^n, 2^n+1\}$. *IEEE Trans. Signal Process*. 2002. Vol. 50, no. 7. Pp. 1772–1779.

5. Hiasat A. A. and Abdel-Aty-Zohdy H. S. Residue to binary arithmetic converter for the moduli set $\{2^k, 2^k+1, 2^{k-1}-1\}$. *IEEE Trans. Circuits Syst. II Analog. Digit. Signal Process*. 1998. Vol. 45, no. 2. Pp. 204–209.

6. Wang W., Swamy M. N. S., Ahmad M.O., and Wang Y., A high-speed residue-to-binary converter for three-mod- uli $\{2^k, 2^k ¡ 1, 2^{k-1} ¡ 1\}$ RNS and a scheme for its VLSI implementation,” *IEEE Trans. Circuits Syst. II Analog. Digit. Signal Process*. 2000. Vol. 47, no. 12. Pp. 1576–1581.

7. Ananda Mohan P. V. New residue to binary converters for the moduli set $\{2^k, 2^k-1, 2^{k-1}-1\}$ *in IEEE Region 10 Con- ference,* TENCON, 2008. Pp. 1–6.

8. Ananda Mohan P. V. RNS to binary converter for the new three moduli set $\{2^{n+1}+1, 2^n, 2^n-1\}$. *IEEE Trans. Circuits Syst. II Express Briefs*. 2007. Vol. 54, no. 9. Pp. 775–779.

9. Hiasat A. and Sweidan A. Residue-binary-decoder for an enhanced moduli set. *IEE Proc. Comput. Digit. Tech*, 2004. Vol. 151. P. 127–130.

10. Molahosseini A.S., et al. Efficient MRC based residue to binary converters for the new moduli sets $\{2^{2n}, {}^{2n-1}, 2^{2n}–1\}$ and $\{2^{2n}, 2^{n}\text{-}1, 2^{n-1}\text{-}1\}$. *IWICE Trans. Inf. Syst.* 2009. Vol. E92-D. Pp. 1628–1638.

11. Ming-Hwa Sheu, Siang-Mia Siao, Yin-Tsung Hwang, Chi- Chia Sun, and You-Ping Lin, "New adaptable three mod- uli set $\{2^{n+k},2^{n}¡1,2^{n-1}¡1\}$ for residue number system – based on finite impulse response implementation," *IEICE Electron. Express*, 2016. Vol. 13, no. 11. Pp.1– 9.

12. Ananda Mohan P. V. Reverse converters for a new moduli set $\{2^{2n}\text{-}1, 2^{n}, 2^{2n}+1\}$", *CSSP*, 2007. Vol. 26. Pp. 215–228.

13. Chaves R. and L. Sousa. $\{2^{n} + 1, 2^{n+k}, 2^{n} – 1\}$ a new RNS moduli set extension. In Euromicro Symposium on Digital System Design: Architectures, Methods and Tools, 2004, *DSD* 2004. Pp. 210–217.

14. Tyagi A. A reduced-area scheme for carry-select adders, *IEEE Trans. Comput*. 1993. Vol. 42. Pp. 1163–1170.

15. Bakalos D., Vergos H. T., and Spyrou A. Efficient mod- ulo 2^{n}-11 squarers. Integr. *VLSI J*, 2011. Vol. 44. Pp. 163–174.

16. Schneiderman R. DSPs evolving in consumer electronics applica- tions. *IEEE Signal Process. Mag.*, 2010. Vol. 27, no. 3. Pp. 6–10.

17. Lin S., Kim Y. B., and Lombardi F. CNTFET-based design of ternary logic gates and arithmetic circuits," *IEEE Trans. Nanotechnol.*, 2011, vol. 10, no. 2. P. 217–225.

18. Sugahara S. and Nitta J. Spin-transistor electronics: An overview and outlook. *Proc. IEEE*. 2010. Vol. 98, no. 12. Pp. 2124–2154.

19. Swartzlander E. E., Cho H., Kong I. and S. W. Kim. Computer arith- metic implemented with QCA: A progress report. In *Proc. Asilomar Conf. Signals, Systems and Computers*, Pacific Grove, CA. 2010. Pp. 1392–1398.

20. Cong J., Sarkar V., Reinman G. and A. Bui. Customizable domain-specific computing. *IEEE Des. Test Comput*. 2011. Vol. 28, no. 2. Pp. 6–15.

21. Ouyang J., Sun G., Chen Y. and L. Duan. Arithmetic unit design us- ing 180nm TSV-based 3D stacking technology. In *Proc. IEEE Int. Conf. 3-D System Integration,* San Francisco, CA. Sept. 2009. Pp. 1–4.

22. Agbedemnab, P. A., Yellakuor, E., & Daabo, M. I. *Single and Multiple Error Detection and Correction using Redundant Residue Number System for Cryptographic and Stenographic Schemes*. 2019. *4*(4). Pp.1–14.

23. Ahmadi, K., & Salari, E. *An Image Hiding Algorithm Using Chinese Remainder Theorem with Reduced Distortion*. 2014. Pp. 240–245.

24. Alhassan, I. Z., & Ansong, E. D. *Enhancing Image Security during Transmission using Residue Number System and k-shuffle*. 2020. *4*(2) Pp. 399–424

25. Alhassan, S., & Gbolagade, K. A. *Enhancement of the Security of a Digital Image using the Moduli Set*. 2013. *2*(7). Pp. 2223–2229.

26. Ammar, A., Ai, A. W. B. A. N. Y., Youssef, M., & Emam, A. *A Secure Image Coding Scheme using Residue Number System*. 2001. Pp.399–405.

27. Aremu, I. A., Gbolagade, K. A. (). *An overview of Residue Number System.* 2017. *6*(10). Pp. 1618–1623.

28. Rajalakshmi, K., Nivedita, R. VLSI implementation of Smith–Waterman algorithm for biological sequence scanning. *Lecture Notes in Electrical Engineering*, 2018. *453*. Pp.231–245.

29. Azizifard, A., Qermezkon, M., Postizadeh, T., & Barati, H. *Data Steganography on VoIP through Combination of Residue Number System and DNA Sequences*. 2014. *5*(2). Pp.7–22.

30. Babatunde, Akinbowale N, Jimoh, R. G., & Gbolagade, K. A. (). An algorithm for a residue number system based video encryption system. *Annals. Computer Science Series. 14th Tome 2nd Fasc. – 2016. XIV*. Pp.137–145.

31. Babatunde, Akinbowale Nathaniel. Methodology for Image Cryptosystem Based on a Gray Code Number System. *School of Computing, Engineering & Physical Sciences Computing and Information Systems Journal* 2019. Vol 23, No 2. Pp.23-28.

32. Babatunde, Akinbowale Nathaniel, Jimoh, E. R., Oshodi, O., & Alabi, O. A. Performance analysis of gray code number system in image security. *Jurnal Teknologi Dan Sistem Komputer*, 2019. *7*(4), Pp. 141–146.

33. Bajard, J. C., Duquesne, S., Ercegovac, M., & Meloni, N. *Residue systems efficiency for modular products summation : Application to Elliptic Curves Cryptography*. 2006. 6313. Pp.1–11.

34. Bhangale, P. P., Raje, R. S., Maurya, J., & Gawad, A. *Image Security using AES and RNS with Reversible Watermarking*. 2017. *4*(5). Pp.345–349.

35. Campobello, G., Leonardi, A., Palazzo, S., & Member, S. *Improving Energy Saving and Reliability in Wireless Sensor Networks Using a Simple CRT-Based Packet-Forwarding Solution*. 2012. *20*(1). Pp. 191–205.

36. Eseyin, J. B. *An Overview of Public Key Cryptosysems and Application of Residue Number System.* 2019. *4*(2). Pp.37–44.

37. Eseyin, J. B., Gbolagade, K. A. *A Residue Number System Based Data Hiding Using Steganography and Cryptography*. 2019. *5*(2). Pp. 345–351.

38. Gomes, R. R., Liberato, A. B., Dominicini, C. K., Ribeiro, M. R. N., & Martinello, M. KAR: Key-for-Any-Route, a Resilient Routing System. *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W* 2016, 2016. Pp. 120–127.

39. Hiasat, A. A Reverse Converter for Three-Moduli Set. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 2019. Pp. 548–553.

40. Junior, J. A., Fernando, L., Nascimento, L., Carlos, L., & Albini, P. *Using the Redundant Residue Number System to increase Routing Dependability on Mobile Ad Hoc Networks*. 2011. Pp.67–73.

41. Kehinde, H., & Alagbe, K. Residue Number System: An Important Application in Bioinformatics. *International Journal of Computer Applications*, 2018. *179*(10). Pp.28–33.

42. Liberato, A., Martinello, M., Gomes, R. L., Beldachi, A. F., Salas, E., Villaca, R., Simeonidou, D. *Residue Defined Networking Architecture*. 2018. *15*(4), P. 1473–1487

43. Madhukumar, A. S., Chin, F., & Premkumar, A. B. Residue number system based multicarrier CDMA for broadband mobile communication systems. *Midwest Symposium on Circuits and Systems*. 2000. *2*. P.536–539.

44. Пристрій для перетворення паралельного двійкового коду в код системи залишкових класів. Патент №104912 України: МПК G06F 7/72 (2006.01), G06F 7/38 (2006.01), H03M 7/18 (2006.01) / Винахідники: Яцків В. В., Саченко А. О., Су Цзюнь / Власник: Тернопільський національний економічний університет – № a201204834; заявл.17.04.2012 р.; опубл. 25.03.2014 р., Бюл. №6.

45. Molahosseini A. S., De Sousa, L. S., Chang, C. H. Embedded systems design with special arithmetic and number systems. Springer. 2017. 390 p.

46. Omondi A. Residue Number System: Theory and Implementation / A.Omond, B.Premkumar. Imperial College Press, 2007. – Vol. 2. – 296 p.

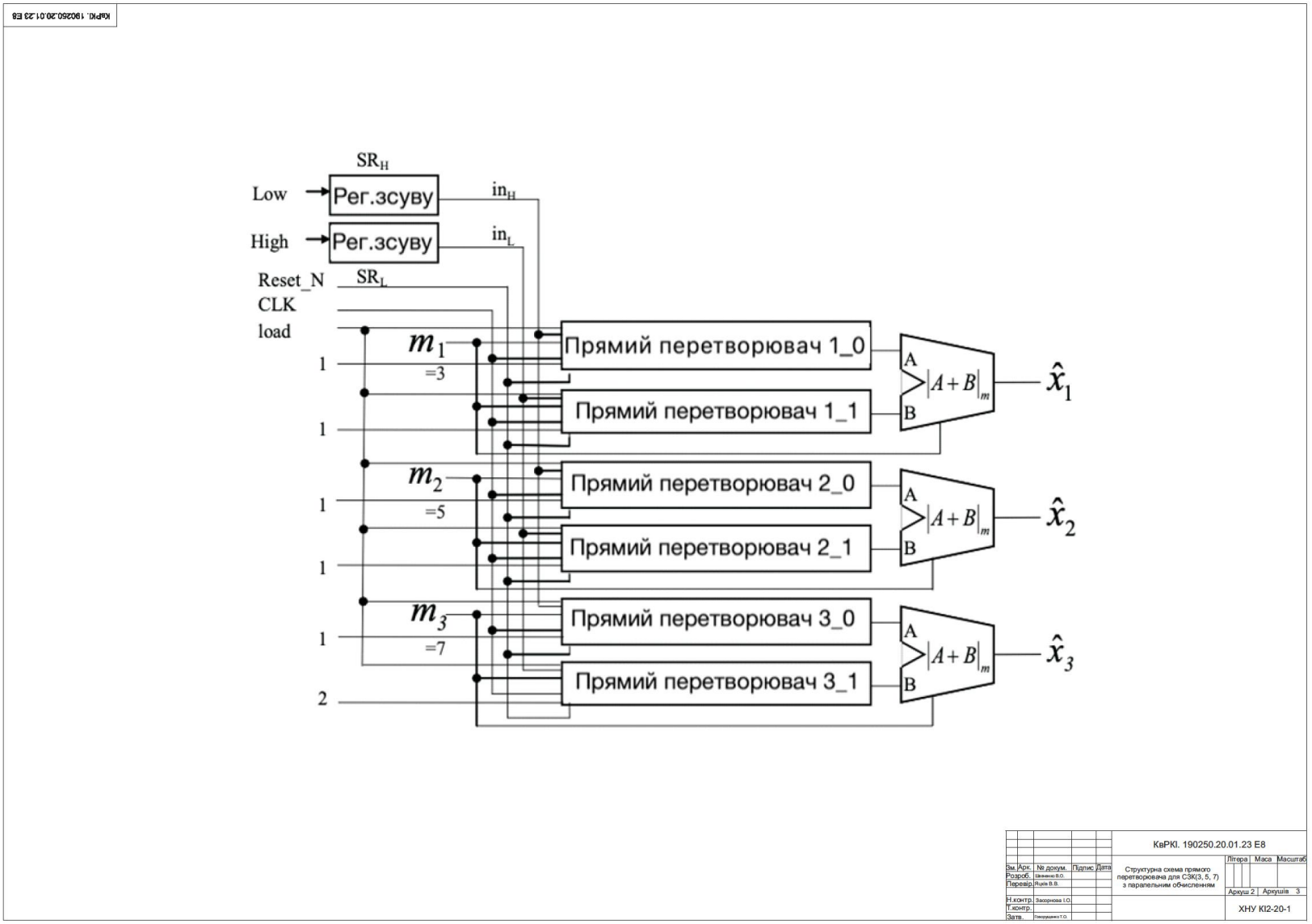
47. Яцків В. В. Методи виконання модулярних операцій та їх реалізація на ПЛІС / В. В. Яцків // *Вісник Хмельницького національного університету. Технічні науки* . 2014. № 6. С. 218-224.

48. Кондратенко Ю.П., Сидоренко С.А., Підопригора Д.М. VHDL-моделі для проектування цифрових пристроїв. Миколаїв: УДМТУ, 2002. 60 с.

49. Кондратенко Ю.П., Поведінковий синтез цифрових пристроїв у середовищі Active-HDL. Миколаїв: Вид-во МФ НаУКМА, 2002. 116 с.

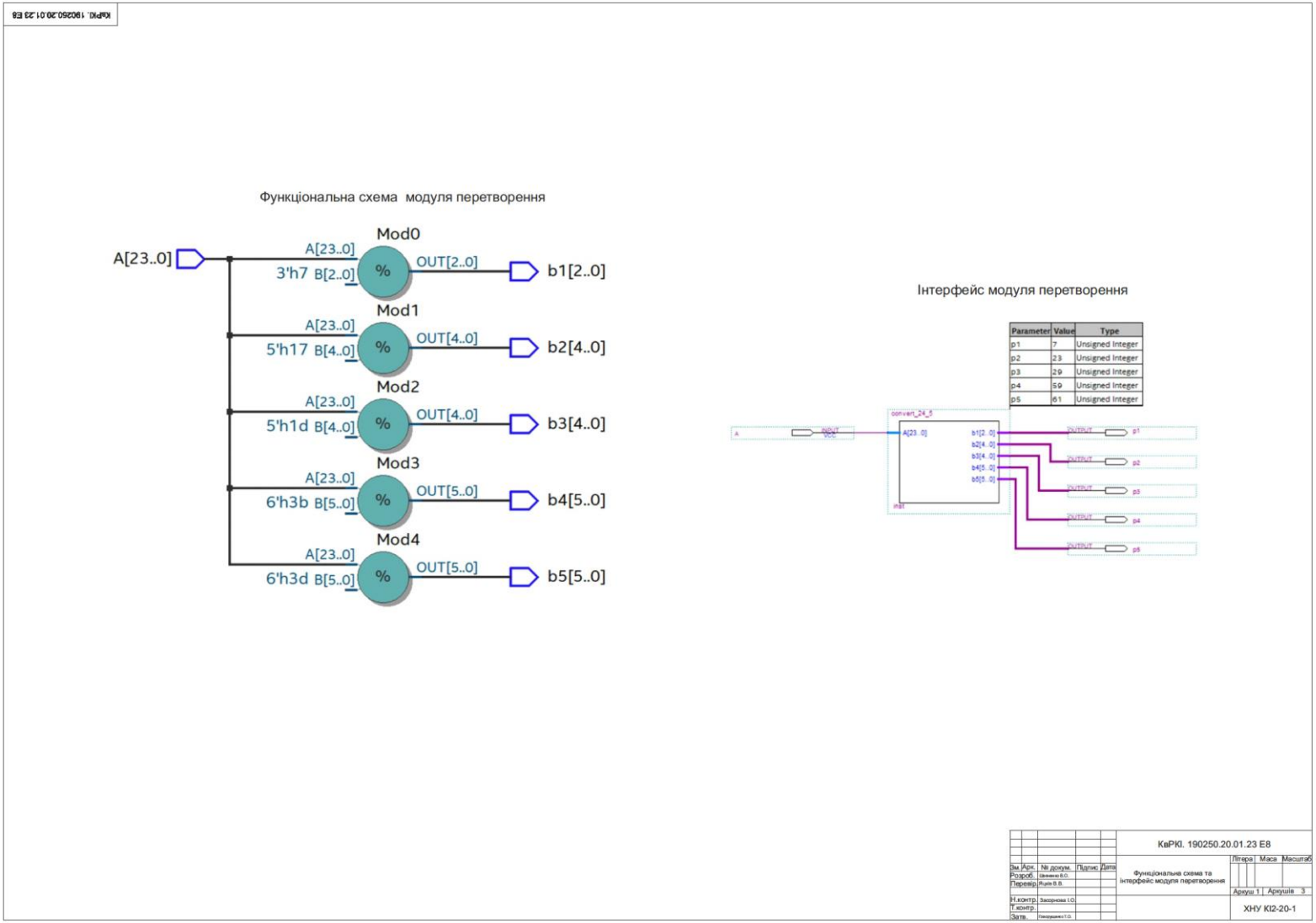
ДОДАТОК А

Копія креслення «Структурна схема прямого перетворювача для СЗК(3, 5, 7) з паралельним обчисленням»



ДОДАТОК Б

Копія креслення «Функціональна схема та інтерфейс модуля перетворення»

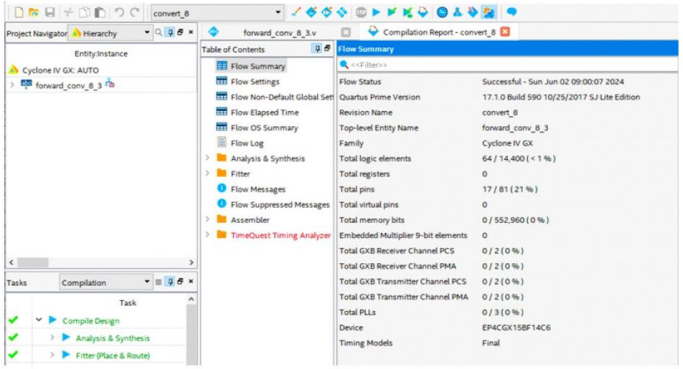


ДОДАТОК В

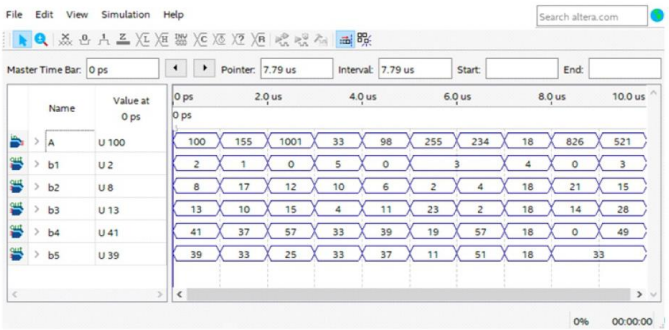
Копія креслення «Результати моделювання»

КвРКІ. 190250.20.01.23 Е8

Результати компіляції модуля



Результати моделювання



						КвРКІ. 190250.20.01.23 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Шевченко В.О.			Результати моделювання		Літера	Маса	Масштаб
Перевір.		Ruslo B.B.							
Н.контр.		Захорнова І.О.					Аркуш 3	Аркушів 3	
Т.контр.									
Затв.		Токарушкіна О.О.				ХНУ КІ2-20-1			

ДОДАТОК Г

Програмний код

```
module forward_conv_8_3 (out0, out1, out2, in);
	output [1:0] out0; // Module: 3
	output [2:0] out1; // Module: 7
	output [3:0] out2; // Module: 13
	input [7:0] in; // Max value: 255

	forward_conv_module_3_1 inst3(in, out0);
	forward_conv_module_7_1 inst7(in, out1);
	forward_conv_module_13_1 inst13(in, out2);
endmodule

module forward_conv_module_3_1 (in, out);
	input [7:0] in; // Max value: 255
	output [1:0] out; // Max value: 2
	// Optimal block size: 2
	wire [3:0] inter1; // Max value: 12
	assign inter1 = in[1:0] + in[3:2] + in[5:4] + in[7:6];
	mod_3_12 inst(inter1, out);
endmodule

module mod_3_12 (in, out);
	input [3:0] in;
	output reg [1:0] out;
	always @ (in)
	begin
	// we have small max value so we can use table here
	case (in)
		0: out = 0;
```

```
                1: out = 1;
                2: out = 2;
                3: out = 0;
                4: out = 1;
                5: out = 2;
                6: out = 0;
                7: out = 1;
                8: out = 2;
                9: out = 0;
                10: out = 1;
                11: out = 2;
                12: out = 0;
                default: out = 0;
        endcase
        end
endmodule

module forward_conv_module_7_1 (in, out);
        input [7:0] in; // Max value: 255
        output [2:0] out; // Max value: 6
        // Optimal block size: 3
        wire [4:0] inter1; // Max value: 17
        assign inter1 = in[2:0] + in[5:3] + in[7:6];
        mod_7_17 inst(inter1, out);
endmodule

module mod_7_17 (in, out);
        input [4:0] in;
        output reg [2:0] out;
        always @ (in)
```

```
        begin
        // we have small max value so we can use table here
        case (in)
                0: out = 0;
                1: out = 1;
                2: out = 2;
                3: out = 3;
                4: out = 4;
                5: out = 5;
                6: out = 6;
                7: out = 0;
                8: out = 1;
                9: out = 2;
                10: out = 3;
                11: out = 4;
                12: out = 5;
                13: out = 6;
                14: out = 0;
                15: out = 1;
                16: out = 2;
                17: out = 3;
                default: out = 0;
        endcase
        end
endmodule

module forward_conv_module_13_1 (in, out);
        input [7:0] in; // Max value: 255
        output [3:0] out; // Max value: 12
        // Optimal block size: 1
```

```
        wire [5:0] inter1; // Max value: 47
        assign inter1 = in[0] + 2*in[1] + 4*in[2] + 8*in[3] + 3*in[4] + 6*in[5] +
12*in[6] + 11*in[7];
        mod_13_47 inst(inter1, out);
    endmodule

    module mod_13_47 (in, out);
        input [5:0] in;
        output reg [3:0] out;
        always @ (in)
        begin
        // we have small max value so we can use table here
        case (in)
            0: out = 0;
            1: out = 1;
            2: out = 2;
            3: out = 3;
            4: out = 4;
            5: out = 5;
            6: out = 6;
            7: out = 7;
            8: out = 8;
            9: out = 9;
            10: out = 10;
            11: out = 11;
            12: out = 12;
            13: out = 0;
            14: out = 1;
            15: out = 2;
            16: out = 3;
```

17: out = 4;

18: out = 5;

19: out = 6;

20: out = 7;

21: out = 8;

22: out = 9;

23: out = 10;

24: out = 11;

25: out = 12;

26: out = 0;

27: out = 1;

28: out = 2;

29: out = 3;

30: out = 4;

31: out = 5;

32: out = 6;

33: out = 7;

34: out = 8;

35: out = 9;

36: out = 10;

37: out = 11;

38: out = 12;

39: out = 0;

40: out = 1;

41: out = 2;

42: out = 3;

43: out = 4;

44: out = 5;

45: out = 6;

46: out = 7;

```
            47: out = 8;
            default: out = 0;
        endcase
        end
endmodule
```

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016337485

Дата перевірки:
09.06.2024 11:44:36 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 15:42:15 EEST

ID користувача:
100005591

Назва документа: Шевченко_Програмно-технічний модуль перетворення даних в систему залишкових класі...

Кількість сторінок: 65 **Кількість слів:** 12436 **Кількість символів:** 87854 **Розмір файлу:** 8.10 MB **ID файлу:** 1016138471

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.93%
Схожість

Найбільша схожість: 3.07% з джерелом з Бібліотеки (ID файлу: 1011131030)

6.13% Джерела з Інтернету	403	Сторінка 67
4.45% Джерела з Бібліотеки	103	Сторінка 72

3.93% Цитат

Цитати	1	Сторінка 73
Посилання	1	Сторінка 73

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	24
Підозріле форматування	12 сторінок

Sun Jun 09 15:01:20 EEST 2024, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

ID: 129190 Назва: БКР Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA Додано в БД: 2024-06-09 Автора: В.О. Шевченко Керівники: В.В. Яцків Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	66989	619	941 (1%)	12 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Шевченко Владислав Олегович

Тема: Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг дипломної роботи:

Кількість листів креслень 3; кількість сторінок записки 58

1. Короткий зміст роботи та прийнятих рішень В роботі розв'язано актуальну задачу розробки модуля перетворення даних з двійкової системи числення в систему залишкових класів на програмованих логічних інтегральних схемах.

2. Висновок про відповідність роботи дипломному завданню Дипломний проект відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено дослідження предметної області та виконано постановку завдання. У другому розділі розглянуто методи перетворення даних в системи залишкових класів. У третьому розділі реалізовано програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA.

4. Позитивні сторони роботи: Розроблено модуль перетворення даних з двійкової системи числення в систему залишкових класів на програмованих логічних інтегральних схемах

5. Негативні сторони роботи: Доцільно було б навести схему електричну принципову для розробленого модуля перетворення даних

6. Оцінка графічного оформлення та пояснювальної записки роботи: пояснювальна записка та листи креслення виконані згідно діючих вимог

7. Відгук про роботу в цілому: В загальному робота виконана на достатньому рівні.

8. Інші зауваження: –

9. Оцінка дипломної роботи:

Розглянувши позитивні та негативні сторони представленої дипломної роботи вважаю, що робота заслуговує оцінки «добре» 3,75 (С)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Вагуїв Руслан Олександрович, доц. КІІ

"12" 06 2024р.

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA

Автор: Шевченко Владислав Олегович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Яцків Василь Васильович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділі аналізу існуючих аналогів та відомих рішень, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 6.9% і адресується до 558 першоджерела; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В.В. Яцків

Гарант ОП _____ С. М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Шевченка Владислава Олеговича
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-20-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

18.06.2024
дата

підпис