Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп’ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»
В.о. зав. кафедрою БІСТ
Ольга МЕЛКОЗЬОРОВА
_____
« » 2023р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Розробка та дослідження архітектури матричного процесора, що функціонує у системі залишкових класів»

оцінка « » Керівник:
Голова ЕК
докт. техн. наук Сергій КОШМАН
Олександр ЛЕМЕШКО
Рецензент:
_____
докт. техн. наук Олена ТОЛСТОЛУЗЬКА
Виконавець : студент групи КБ-61

Євген ЛУНЬОВ

Харків – 2023

РЕФЕРАТ

Кваліфікаційна робота магістра Луньова Євгена Володимировича

Спеціальність 125 «Кібербезпека»

Назва кваліфікаційної роботи магістра: «Розробка та дослідження архітектури матричного процесора, що функціонує у системі залишкових класів»

Пояснювальна записка до дипломної роботи складається з вступу, чотирьох розділів, висновків, списку використаних джерел та містить 65 сторінок, 33 рисунки, 10 таблиць, 22 найменувань використаних джерел.

Метою роботи є підвищення продуктивності системи обробки інформації за рахунок застосування непозиційної системи числення у залишкових класах.

У роботі досліджені системи числення та їх використання, основи медулярної арифметики, структуру обчислювальних пристроїв, що працюють у системі залишкових класів, та способи їх реалізації. В результаті проведених досліджень змодельована система обробки інформації, яка побудована на основі системи залишкових класів. Результати моделювання показали ефективність використання системи залишкових класів для побудови системи обробки інформації.

Результат дослідження можливо використовувати при проєктуванні та побудові арифметичних розширювачів для підвищення швидкодії виконання арифметичних операцій.

Ключові слова: СИСТЕМА ОБРОБКИ ІНФОРМАЦІЇ, СИСТЕМА ЗАЛИШКОВИХ КЛАСІВ, МАТРИЧНИЙ ПРИНЦИП, ПРОГРАМОВАНА ЛОГІЧНА ІНТЕГРАЛЬНА СХЕМА, HDL – МОДЕЛЬ.

ABSTRACT

Master's qualification work – Yevhen Lunov

Speciality 125 «Cybersecurity»

The title of the master's qualification work: «Development and research of the matrix processor architecture functioning in the system of residual classes»

The explanatory note contains 65 pages, 33 figures, 10 tables, 22 annexes.

The aim the thesis is to increase the productivity of the information processing system due to the use of a non-positional counting system in residual classes.

The paper examines counting systems and their use, the basics of medullary arithmetic, the structure of computing devices operating in the system of residual classes, and methods of their implementation. As a result of the conducted research, an information processing system was modeled, which is built on the basis of a system of residual classes. The simulation results showed the effectiveness of using the system of residual classes to build an information processing sys.

The result of the research can be used in the design and construction of arithmetic expanders to increase the speed of performing arithmetic operations.

Keywords: CODE-BASED CRYPTOSYSTEM, NIST PQC CONTEST, PSEUDORANDOM NUMBER GENERATOR, POST-QUANTUM CRYPTOGRAPHY, SYNDROME DECODING.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА СИМВОЛІВ

АО	-	арифметична операція
АП	-	арифметичний пристрій
ЕОМ	-	електронна обчислювальна машина
ІМС	-	інтегральна мікросхема
НСЧ	-	непозиційна система числення
СОІ	-	система обробки інформації
СЗК	-	система залишкових класів
СЧ	-	система числення
ПЛІС	-	програмована логічна інтегральна схема
FPGA	-	Field Programmable Gate Array
VHDL	-	Very high speed integrated circuits Hardware Description Language
CLB	-	Configurable Logic Block
IOB	-	Input/Output Block
DCM	-	Digital Clock Manager

ЗМІСТ

ВСТУП

Актуальність роботи. Рішення широкого кругу складних обчислювальних науково-технічних задач вимагає значних об’ємів розрахунків. Дослідження пошуків ефективних методів та засобів підвищення продуктивності обробки інформації є перспективною темою для подальшого розвитку.

Створення надійних та швидкодіючих системи обробки інформації (СОІ) є важливою і актуальною задачею. Застосування таких систем досить широко розповсюджена у системах як критичного застосування, так і для систем сільськогосподарського виробництва.

Світовий ринок перебуває в бурному розвитку комп’ютерних технологій і процесів, які пов’язані з цим. Комп’ютерні технології проникають у всі сфери діяльності, залежність сучасної люди від них стає більшою а в деяких сферах незамінною і життєво необхідним. Кінцевими споживачами ставляться зовсім нові вимоги до швидкодії, надійності та ще ряду специфічних вимог до систем обробки інформації.

В основному, сучасний розвиток СОІ відбувається за сформульованим в 1975 році Гордоном Муром законом, який стверджує, що кількість транзисторів на кристалі мікросхеми збільшується кожні 24 місяці, тобто, потужність подвоюється кожні два роки але вчені і сам Гордон Мур заявляють, що цей закон скоро перестане діяти із за фізичних і природніх обмежень. Але на базі сучасного розвитку технологій можливе якісне і кількісне збільшення параметрів СОІ. Можливе використання нових ідей, методів, підходів або вдосконалення і розвиток старих, які із за деяких складнощів не були належно використанні.

Одним із напрямів підвищення продуктивності і надійності функціонування системи обробки інформації є розпаралелення алгоритму обробки інформації. Одним із методів розпаралелення алгоритмів є представлення чисел у коді системи залишкових класів (СЗК).

Першопрохідцями в застосуванні системи залишкових класів в системах обробки інформації можна вважати чехословацького інженера М. Валах і активно його підтримуючий математик А. Свобода, які вперше подали ідею застосування системи залишкових класів в обчислювальній техніці.

Розвиток обчислювальної техніки, яка функціонує на основі системи залишкових класів є актуальною і перспективною темою, так як СЗК дозволяє пришвидшити і збільшити надійність при обробці цифрової інформації.

Метою роботи є підвищення продуктивності системи обробки інформації за рахунок застосування непозиційної системи числення у залишкових класах.

Для реалізації поставленої мети необхідно розв'язати наступні задачі:

- дослідити особливості функціонування системи обробки інформації;
- дослідити існуючи методи підвищення продуктивності системи обробки інформації;
- синтезувати та дослідити матричну модель пристрою, що функціонує в системі залишкових класів;

Об'єкт дослідження – процеси обробки інформації у системах, що функціонують на основі СЗК.

Предмет дослідження – системи обробки інформації, що функціонують на основі СЗК.

Методи дослідження. В основу проведених в роботі досліджень були покладені принципи системного аналізу, метод теорії чисел при дослідженні і синтезі апаратних засобів реалізації арифметичних операцій у системі залишкових класів.

Практичне значення одержаних результатів. В результаті магістерської роботи досліджено принцип функціонування системи обробки інформації та розроблено однобайтову систему, яка функціонує на основі СЗК. Результати досліджень доцільно впроваджувати в системах, де ставляться високі вимоги до швидкості оброблення інформації.

1 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ФУНКЦІОНУВАННЯ ІСНУЮЧИХ СИСТЕМИ ОБРОБКИ ІНФОРМАЦІЇ

1.1 Опис «класичних» ахітектур СОІ

Принципи функціонування системи обробки інформації закладає розробник, під час проєктування архітектури СОІ. Архітектура визначає основні методи і принципи, в оброці інформації, і можливості самої СОІ. Головні функції СОІ полягають у зборі, накопиченні, збереженні, обробці та видачі інформації.

Поняття архітектура системи обробки інформації включає в себе формат даних і команд, методи адресації, систему команд, загальну організацію процесора, організацію основної пам’яті і пристроїв введення і виведення.

Дані це числа, які в системі обробки інформації, в основному, представлені в двійковій системі числення. В залежності від вибраної архітектури дані по різному проходять обробку, представлення. Від архітектури залежить споживчі характеристики: продуктивність, надійність, ємність пам’яті, ціна, тощо. Тому, виходячи з якими даними працює СОІ, які вимоги і особливості ставляться до функціонування СОІ, вибирають найбільш ефективну архітектуру. Розглянемо декілька найбільш розповсюджених архітектурних принципів.

Описана Джоном фон Нейманом архітектура універсального комп’ютера ENIAC, який був розроблений Преспером Екертом та Джоном Моучлі в Принстонському університеті. Інколи ще її називають принстонською архітектурою. Архітектурний принцип Джона фон Неймана зображено на рис. 1.1.

Головні особливості архітектури Джона фон Неймана [22]:

- в СОІ інформація ділиться на команди і дані;
- команди вказують які дії над якими операндами виконувати;
- данні і команди представлені в двійковому коді;
- дані і команди зберігаються в одній пам’яті;

- пам'ять має довільну адресацію, що дозволяє звертатись до будь якої комірки за необхідністю;

- послідовність виконання команд, за якою виконується алгоритм;

- пам'ять є лінійною.

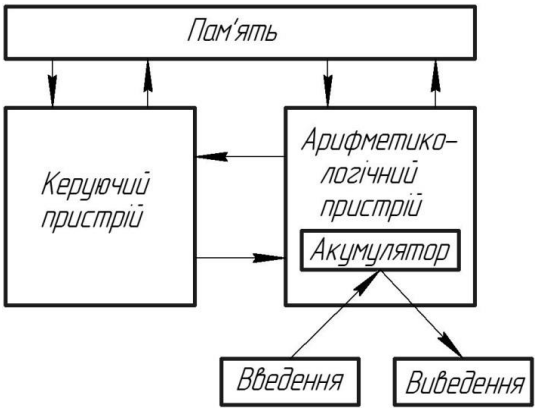

Рисунок 1.1 - Архітектурний принцип Джона фон Неймана

Джерело: розроблено автором

Функціонування, в такій архітектурі, здійснюється потактово за вказівкою команд програми. Команда вказує тип операції, адрес даних, над якими виконується операція і її запис у відповідний адрес. Для виконання команди необхідно виконати наступні дії:

- запис адреси команди до програмного лічильника;

- зчитування з основної пам’яті команди за вмістом програмного лічильника;

- розпізнавання команди (дешифровка);

- визначення адресу комірок даних та адресу запису результату;

- зчитування та подача даних в арифметико-логічний пристрій (АЛП);

- виконання операції над даними;

- запис результату в основну пам'ять.

Головним недоліком функціонування архітектури Джона фон Неймана являється те, що процесор та пам'ять загальмовують роботу один одному. Пам'ять працює набагато повільніше процесора.

Одним із засобів підвищення швидкості обробки інформації є збільшення розрядності інформаційного слова. Саме цю ідею Ховардом Айкеном було

реалізовано в Гарварді в комп’ютері Марк-1. Суть цієї ідеї в розділенні пам’яті на пам’ять даних і пам’ять команд [22]. Цим розділяється і шини передачі даних і команд. В такому випадку команди і дані можуть передаватись і оброблятись паралельно. На рисунок 1.2 зображено принцип гарвардської архітектури.

Головним недоліком такої архітектури є складність реалізації. Так як для кожного запам’ятовуючого пристрою потрібен контролер і шина.

Так, при збільшенні розрядності слова збільшується і кількість з’єднань, що ускладнює проєктування та розробку (Рис. 1.2).

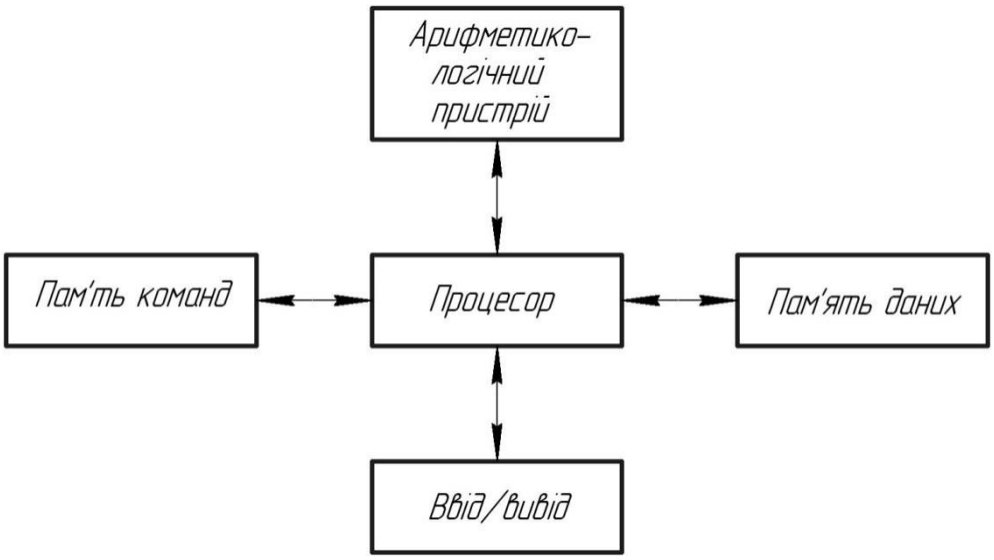

Рисунок 1.2 - Принцип гарвардської архітектури

Джерело: розроблено автором

Дуальна принстонсько-гарвардська архітектура являється поєднанням принципів гарвардської та принстонської архітектури, так як гарвардська архітектура складна при програмуванні а принстонська повільна.

Головною ідеєю є поділення кеш-пам’яті на кеш даних та кеш команд [12]. Принципи дуальна пристонсько-гарвадської архітектури зображені на рисунку 1.3.

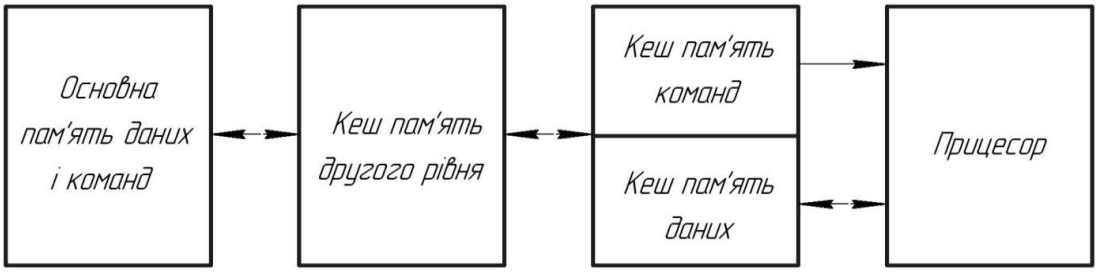

Рисунок 1.3 - Принципи дуальна пристонсько-гарвадської архітектури

Джерело: розроблено автором

Поєднання двох архітектур дало зручність пристонської архітектури а збоку гарвардської, збільшення швидкості обробки даних.

1.2 Дослідження систем числень для побудови СОІ

Вважають, що будь-яке число має значення і форму подання.

Значення числа визначає його відношення до значень інших чисел (більше, менше, рівно) і відповідно, порядок розташування чисел на числовій осі. Форма подання визначає порядок запису числа за допомогою визначених для цього знаків. При цьому значення числа не залежить від способу його представлення. Це означає, що число з одним і тим же значенням можна записати декількома способами, тобто відсутня взаємно однозначність між представленням числа і його значенням. В зв'язку з цим виникають питання про форму представлення числа і про можливості і методах переходу від однієї форми до іншої [1].

Спосіб представлення числа визначається системою числення (СЧ).

Система числення – це правило запису чисел за допомогою заданого набора спеціальних знаків – цифр.

Існує велика кількість систем числень. Від їх особливості залежить наочність відображення та виконання математичних операцій над числами. Всі системи числення можна розділити на декілька груп: непозиційні, позиційні та змішані.

Непозиційна система числення – система числень в якій позиція числа в цифрі не залежить її величина.

Із непозиційних найбільш відома, можна вважати, римську систему числення. В ній базові числа позначені великими латинськими літерами і відповідають 1 – I, 5 - V, 10 – X, 50 – L, 100 – C, 500 – D, 1000 – M. Всі інші числа будуються комбінацією базових в залежності правилам:

- якщо цифра молодшого значення стоїть зліва від цифри більшого значення то їх значення віднімається, якщо з права то додаються;
- цифри I, X, C, M можна записувати підряд не більше три рази кожну;
- цифри V, L, D для запису числа використовують тільки один раз.

Запис чисел в такій системі великий і не зручний так число 1648 матиме вигляд MDCXLVIII. Але виконання самих простих арифметичних операцій ще складніший, відсутній нуль і відсутність чисел більше M не дозволяє записати будь яку цифру. Тому римська система використовується тільки для нумерації.

Унарна – це система числення, в якій для запису числа використовують тільки один знак - | (вертикальна риска). Наступне число отримуємо із попереднього доданням ще одного знака. Унарна система важлива в теоретичному відношенні, оскільки число представлено самим простим способом і відповідно, прості операції з ним. Крім того, саме унарна система визначає значення цілого числа кількістю наявних в ній одиниць, яке не залежить від форми представлення.

Позиційна – це система числення, в якій значення кожної цифри в зображенні числа визначає її положення (позиція) в ряду інших цифр.

В загальному вигляді будь-яке число *D,* із позиційної системи числення, можна представити наступним чином [6]:

$$D = d_{p-1} \cdot b^{p-1} + d_{p-2} \cdot b^{p-2} + \ldots + d_1 \cdot b^1 + d_0 \cdot b^0 , + d_{-1} \cdot b^{-1} + \ldots$$

$$\ldots + d_{-n} \cdot b^{-n} = \sum_{i=-n}^{p-1} d_i \cdot b^i , \qquad (1.1)$$

де d_i - цифра (символ алфавіту) системи числення;

b - основа системи числення;

p - число цифр в цілій частині числа;

, - знак, який відокремлює цілу частину від дробової;

n - число цифр в дробовій частині числа;

b_i - вага цифри на відповідній і-й позиції числа.

Домінуючою в обчислювальній техніці являється саме позиційна система числення. Головною перевагою позиційного представлення числа являється те, що в ньому за допомогою кінцевого набора знаків можна записати необмежену кількість різних чисел. До того ж в позиційних системах числення набагато проще здійснювати арифметичні операція множення, ділення.

В комп'ютерній техніці широкого застосування, крім звичної нам десяткової системи числення знайшли двійкова, вісімкова та шістнадцяткова системи числення.

Найбільш звичною і розповсюдженою являється десяткова система числення. В якій для запису числа використовують десять цифр 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. Число представляє собою короткий запис многочленна, в який входе степінь деякого другого числа – основа системи числення:

$$373{,}33 = 3\cdot 10^2 + 7\cdot 10^1 + 3\cdot 10^0 + 3\cdot 10^{-1} + 3\cdot 10^{-2}.$$

В цьому випадку цифра 3 зустрічається чотири рази, але значення цих цифр різне і визначається їх нею позицією в числі.

1.2.1 Двійкова, вісімкова та шістнадцяткова система числення

Двійкова система числення з основою 2 і цифрами 0 і 1.

В обчислювальних системах виконуються такі дії як: збирання, зберігання, оброблення і представленні інформації, яка подається у вигляді потоку цифрових даних. Дані цифрового потоку можуть бути числа (як зі знаком так і без нього), масиви, адреса, символи алфавіту що використовується. Всі ці дані в комп'ютері представляються в двійковому вигляді [3-5]. Широкого застосування ця система числення здобула завдяки деяким особливостям обчислювальної техніки і легкій реалізації.

Вісімкова система числення має основу *8* і цифри *0, 1, …, 7*.

Шістнадцяткова системи числення має основу *16* і цифри *0,1 …,9, A, B, C, D, E, F*. При чому *A* цифра шістнадцятирічної системи числення, яка відповідає числу 10 в десятирічнній системі числення, $B_{(16)}=11_{(10)}$, $C_{(16)}=12_{(10)}$, $D_{(16)}=13_{(10)}$, $E_{(16)}=14_{(10)}$, $F_{(16)}=15_{(10)}$.

Оскільки двійковий запис громіздкий, так як містить багато цифр, і до того ж погано запам’ятовується і сприймається, тому в комп’ютерній техніці використовують шістнадцяткову і вісімкову систему числення. Вони використовується для нумерації ячейок пам’яті комп’ютера, запису кодів команд, нумерації пристроїв і регістрів, відображенні результату на дисплеї, як проміжні системи числення.

Використання цих систем числень також дозволяє зменшити обсяг програми при її написанні.

Вибір цих систем обумовлений їхньою легкістю переходу від двійкової до вісімкової та шістнадцяткової системи числення і навпаки.

Для переведення числа з двійкової системи числення в вісімкову/шістнадцятикову систему числення треба число розбити на тріади/тетроди відповідно вліво і вправо від коми і потім замінити кожну групу відповідно на значення в вісімковій/шістнадцятковій системі числення [3].

Приклади переведення з двійкової в вісімкову/шістнадцятикову СЧ:

$$101111{,}011_{(2)} = 101\ 111,\ 011_{(2)} = 57{,}3_{(8)};$$

$$11001,\ 01_{(2)} = 011\ 001,\ 010_{(2)} = 31{,}2_{(8)};$$

$$1101{,}1_{(2)} = 1101,\ 1000_{(2)} = D{,}8_{(16)};$$

$$101111{,}111_{(2)} = 0010\ \ 1111,\ 1110_{(2)} = 2F{,}E_{(16)}.$$

Для переведення числа із вісімкової/шістнадцятикової СЧ в двійкову систему необхідно замінити кожну цифру на тріади/тетради двійкових цифр відповідно.

Приклади переведення із вісімкової/шістнадцятикової системи числення в двійкову систему:

$$2541_{(8)} = 010\ 101\ 100\ 001_{(2)};$$

$$37{,}64_{(8)} = 011\ 111,\ 110\ 100_{(2)};$$

$$1377_{(16)} = 010\ 101\ 100\ 001_{(2)};$$

$$2F{,}E_{(16)} = 0010\ 1111,\ 1110_{(2)}.$$

Для переведення чисел з восьмирічної в шістнадцятирічну СЧ і навпаки використовується проміжну СЧ (двійкова або десяткова).

$$37_{(8)}=3\cdot 8^1+7\cdot 8^0=31_{(10)} \rightarrow 1F_{(16)}$$

1.2.2 Переведення чисел із системи числення з основою q у десяткову систему числення

Для переведення чисел із системи числень з основою q в десяткову СЧ необхідно число записати у кількісному еквіваленті, замінивши цифри системи числення з основою q та основу q їхніми десятковими еквівалентами, потім обчислити вираз за правилами арифметики десяткової системи числення [3].

$$11010010{,}01_{(2)}=1\cdot 2^7+1\cdot 2^6+0\cdot 2^5+1\cdot 2^4+0\cdot 2^3+0\cdot 2^2+1\cdot 2^1+0\cdot 2^0+0\cdot 2^{-1}+1\cdot 2^{-2}=$$

$$=128+64+0+16+0+0+2+0+0+0{,}25=210{,}25_{(10)};$$

$$735_{(8)}=7\cdot 10_{(8)}^2+3\cdot 10_{(8)}^1+5\cdot 10_{(8)}^0=7\cdot 8_{(10)}^2+3\cdot 8_{(10)}^1+5\cdot 8_{(10)}^0=477_{(10);}$$

$$92C8_{(16)}=9\cdot 10_{(16)}^3+2\cdot 10_{(16)}^2+C\cdot 10_{(16)}^1+8\cdot 10_{(16)}^0=9\cdot 16_{(10)}^3+2\cdot 16_{(10)}^2+12\cdot 16_{(10)}^1+$$

$$+8\cdot 16_{(10)}^0=37576_{(10)}.$$

1.2.3 Переведення чисел із десяткової системи числення в систему числення з довільною основою

Для переведення числа із десяткової СЧ в іншу позиційну СЧ, з основою j, необхідно послідовно поділити число на основу потрібної системи числення. Ділення продовжується доти, доки залишок від ділення не стане меншим від дільника.

Отримані остатки від ділення, взяті у зворотному порядку, будуть значеннями розрядів числа в новій системі числення.

Наприклад, переведемо числа *92* з десяткової системи числення в двійкову (рис. 1.4), вісімкову (рис. 1.5), шістнадцяткову (рис.1.6) СЧ.

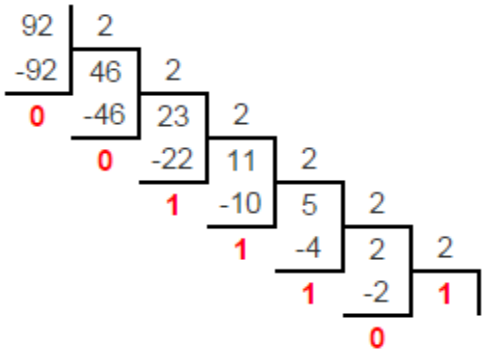

Рисунок 1.4 – Переведення числа 92 з десяткової системи числення в двійкову

Джерело: розроблено автором

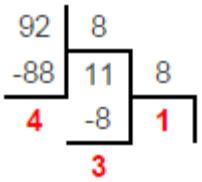

Рисунок 1.5 – Переведення числа 92 з десяткової системи числення в вісімкову СЧ

Джерело: розроблено автором

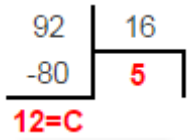

Рисунок 1.6 – Переведення числа 92 з десяткової системи числення в шістнадцятирічну СЧ

Джерело: розроблено автором

Так, $92_{(10)} = 1011100_{(2)}$; $92_{(10)} = 143_{(8)}$; $92_{(10)} = 5C_{(16)}$.

1.2.4 Двійково-десяткове представлення чисел

Для зменшення програмних та апаратних витрат при перетворенні двійкових чисел на десяткові, при виведенні або відображенні інформації використовують двійково-десяткову систему (BCD-система) або як прийнято називати такий код “8421”. У двійково-десятковій системі кожна цифри

десяткової системи кодуються двійковою тетрадою, що дозволяє ефективно і зручно. Так, число *369* в десятковій СЧ в двійково-десятковому представлені має вигляд:

$$396_{(10)}=0011\ 1001\ 0110_{(2\text{-}10)}.$$

Цей метод має надлишковість, оскільки використовує для кодування будь якої цифри 10 комбінацій з 16 можливих.

При використанні одного байта (8 біт) для кодування числа, зазвичай, 4 старші розряди відводять для кодування знаку числа, 4 молодші розряди для самої цифри. Кодування двійково-десяткових чисел наведено в таблиці 1.2.

Таблиця 1.1 – Кодування двійково-десяткових чисел

Цифра	Код BCD	Код "8421+3”	Код “5211”
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0011
3	0011	0110	0101
4	0100	0111	0111
5	0101	1000	1000
6	0110	1001	1001
7	0111	1010	1011
8	1000	1011	1101
9	1001	1100	1111
без знаку	1111		
додатні	1100		
від'ємне	1101		

Джерело: розроблено автором

1.2.5 Принципи виконання арифметичних операцій

Основні арифметичні операції додавання, віднімання, множення. Також до арифметичних відносяться наступні операцій [2]:

- ділення;

- взяття абсолютної величини від операнда;
- інверсія знака операнда;
- збільшення операнда на одиницю;
- зменшення операнда на одиницю.

Від вибраного представлення числа залежить особливості реалізації цієї операції. Також метод реалізації арифметичних операцій вибирають в залежності від поставлених цілей і, зачасту, від тенічних характеристик СОІ.

Додавання двійкових чисел, які представленні без знака, виконуються по аналогії з десятковою системою числення. Додавання починається з молодших розрядів, розряди які знаходяться на однакових позицій додаються, при виникненні переносу в цьому розряді значення цього переносу переноситься в наступний розряд і там додається. При виникненні в старшому розряді переносу виникає переповнення, внаслідок якого втрачається правильність результату, це потрібно враховувати при написанні програми. Приклад виконання операції додавання показано на рис.1.7.

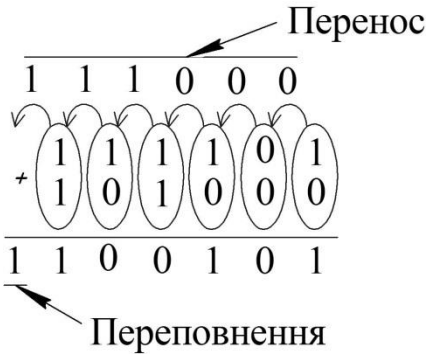

Рисунок 1.7 - Приклад виконання операції додавання чисел без знаку

Джерело: розроблено автором

Двійкові числа зі знаком можуть бути представленні в таких видах: прямий, обернений, доповняльний та зміщений код [18].

В прямому коді додавання вимагає аналізу знаків числа. Якщо знаки одинакові то числа додаються і результату присвоюється їх знак . Якщо знаки різні то числа віднімаються а результату віднімання присвоюється знак

найбільшого за модулем числа. У випадку якщо модулі одинакові отримуємо знак плюс.

В оберненому і доповняльному коді на додавання не впливає знак чисел і відбувається за тим принципом як і прямому коді додатні числа тільки з деякими відмінностями. В оберненому коді при переносу із найстаршого розряду, одиницю потрібно подавати на молодший розряд. В доповняльному коді перенос не потребується. На рис. 1.8 зображено додавання чисел в доповняльному та оберненому коді.

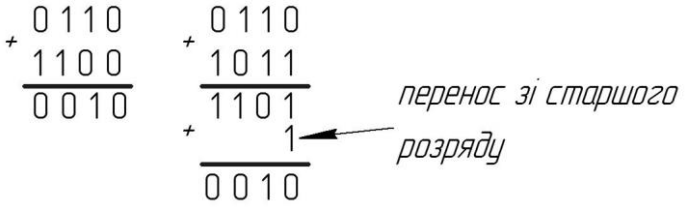

Рисунок 1.8 - Додавання чисел 6 та – 4 в доповняльному (зліва)та оберненому (справа) коді

Джерело: розроблено автором

У випадках коли результат операції виходить за межі точність і достовірність обрахунків може падати. Тому програмістам під час розробки повинні враховувати цю проблему переповнення. Для фіксації переповнення використовують модифіковані коди з двома розрядами для знака, так плюс представляється як 00, мінус – 11. Так як переповнення може виникнути якщо знаки доданків одинакові, тому порівнюється результат додавання знаків. Тобто якщо в результаті додавання знак відрізняється від знаку доданків то виникає переповнення [19].

Віднімання можна проводити двома способами:

- віднімання розрядів чисел
- додавання двійкового коду (від’ємник записується з протилежним знаком)

Перший метод використовують, зазвичай, для віднімання чисел які представленні в прямому коді. Віднімання в такому випадку проходить по

аналогії як і в десятковій системі числення. Віднімаються розряди відповідних позицій, при виникненні запозичення одиниця вираховується зі старшого розряду.

В оберненому або доповняльному коду можливе використання іншого методу. Для цього потрібно замінити знак від'ємника, інвертувати всі розряди а в випадку доповняльного коду додати до від'ємника одиницю в молодший розряд. Потім отримані коди додати. Множення проводиться в числах, які представленні в прямому, оберненому та доповняльному коді. Для визначення знака користуються таблицею 1.2.

Таблиця 1.2 – Знак результату при множенні чисел X на Y

Знак, a_i	Знак, b_i	Знак результату, z_i
0	0	0
0	1	1
1	0	1
0	0	0

Джерело: розроблено автором

При множенні в прямому коді, модулі перемножуються як двійкові числа без знаку, так як процедура одна і та ж сама. При множенні чисел представлених в оберненому коді, потрібно всі від'ємні числа інвертувати і проводити множення по аналогії з прямим кодом.

В загальному випадку множення двох чисел, для прикладу візьмемо 4-х розрядні числа $A=(a_3a_2a_1a_0)_2$ і $B=(b_3b_2b_1b_0)_2$, представлено на рисунку 1.9.

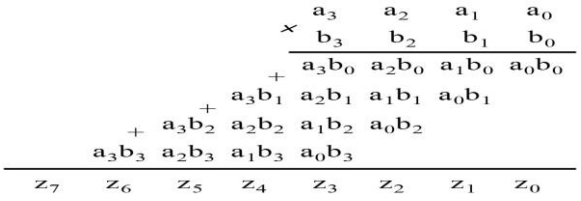

Рисунок 1.9 – Узагальнений випадок множення двох 4-х розрядних чисел

Джерело: розроблено автором

Де z_i результат додавання добутків відповідних розрядів $a_i b_i$. Значення результату множення двох розрядів a_i і b_i визначається згідно таблиці 1.3.

Таблиця 1.3 – Значення *i*-го розряду при множенні двох чисел

a_i	b_i	результату z_i
0	0	0
0	1	0
1	0	0
0	0	0
1	1	1

Джерело: розроблено автором

В першому розділі було проаналізовано принципи представлення і кодування цифрової інформації в СОІ. Принцип кодування в СОІ вибирається відповідно поставленим задачам і областю застосування. З вищесказаного можна сформулювати задачі які потрібно вирішити, для досягнення поставленої мети:

- вибір ефективного методу кодування;
- вибір ефективних принципів виконання арифметичних операцій;
- реалізації структури обробки інформації;
- апаратний синтез розробленої СОІ.

2 ДОСЛІДЖЕННЯ ТЕОРЕТИЧНИХ ОСНОВ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ

2.1 Представлення чисел у системі залишкових класів

Недоліком будь-якої позиційної системи числення є наявність міжрозрядних зв'язків. Дійсно, результат додавання в *i*-м розряді залежить не тільки від *i*-х розрядів, а і від переносу із *i-1* розряду, в кінцевому підсумку і від всіх значень молодших розрядів *i-1, i-2, …, 1, 0.* Тому розрахунок розрядів може проходити тільки послідовно (з урахуванням формування переносу із попереднього, молодшого, розряду) [20,21]. Ця обставина перешкоджає розпаралелювання процесу обчислення і, природно, знижує швидкість оброблення даних.

В рамках позиційних систем числення відомо декілька способів логічного і схемотехнічного прискорення арифметичних операцій – паралельний перенос, матрична (таблична) арифметика і інші, але всі вони вимагають досить великих апаратних затрат.

Пошук нових шляхів побудови арифметичних пристроїв СОІ, яка дозволяє виключити залежність між розрядами при виконанні арифметичних операцій, призвів до використання в машинній арифметиці теорії залишків, в рамках якої була розроблена непозиційна система числення – система числення в залишкових класах (СЗК) [17].

Про застосування СЗК в обчислювальній техніці в 1955 р в короткій статті висловив чехословацький інженер М. Валах, його активно підтримав математик А. Свобода. Вони і стали першопрохідцями застосування СЗК в машинній арифметиці. Їх роботами зацікавилися американські вчені. Через кілька років Свобода і Валах переїхали до США, де роботи над модульної арифметикою (заснованої на СЗК) були продовжені.

Якщо фіксований ряд цілих додатних чисел $p_1, p_2, ..., p_n$ названих основами СЗК, то під системою залишкових класів розуміється така непозиційна система числення, в яких любе додатне ціле число представлене в виді набора залишків, які утворені від ділення цього числа на вибрані основи.

В системі залишкових класів число представляється у вигляді незалежних залишків [7-9]. Всі залишки чисел представляються згідно виразу

$$a_n = A - \left(\frac{A}{p_n} \right) \times p_n, \qquad (2.1)$$

де A – число в позиційній системі числення;

p_n – основа системи залишкових класів;

$\left(\frac{A}{p_n} \right)$ - ціла частина від A/p_n, при чому a_n - найменший цілий залишок від ділення числа A на p_n.

Також формулу 2.1 можна подати у такому вигляді:

$$A \equiv a_n (mod p_n). \qquad (2.2)$$

Наприклад визначимо залишок від числа $A=31$ по основі $p = 7$.

$$a = 31 - \left(\frac{31}{7} \right) \times 7 = 31 - 4 \times 7 = 3;\ 31 \equiv 3(mod 7).$$

Для представлення чисел в СЗК необхідно вибрати систему основ – множина цілих чисел $p_1, p_2, ..., p_n$. Тоді будь-яке число A може бути представлено в СЗК наступним чином:

$$A = (a_1, a_2, ..., a_n), \qquad (2.3)$$

де $a_i \equiv A(mod p_i)$.

Однозначне представлення чисел в СЗК залежить від вибраних основ і діапазону, які вони охоплюють. Якщо основи взаємно прості, діапазон чисел із ПСЧ знаходиться в межах $[0,(P-1)]$:

$$P = p_1 \cdot p_2 \cdot \ldots \cdot p_n . \qquad (2.4)$$

Наприклад представимо число $A=31$ по основи $p_1=3$, $p_2=5$, $p_3=7$ в СЗК. Для представлення числа в СЗК необхідно знайти залишок від ділення на кожну основу.

$$a_1 = 31 - (31/3) \times 3 = 31 - 10 \times 3 = 1 \to 1(mod3) ;$$

$$a_2 = 31 - (31/5) \times 5 = 31 - 6 \times 5 = 1 \to 1(mod5) ;$$

$$a_3 = 31 - (31/7) \times 7 = 31 - 7 \times 4 = 4 \to 4(mod7) .$$

Отже, число $A=31$по основам $p_1=3$, $p_2=5$, $p_3=7$ в СЗК матиме вигляд *(1, 1, 4).*

Представимо в СЗК декілька чисел:

Діапазон однозначного представлення для вибраних основ $P=3 \cdot 5 \cdot 7=105$

1 = (1, 1, 1); 106 = (1, 1, 1); 37 = (1, 2, 2); 142 = (1, 2, 2).

Як видно, при виході із діапазону *[0,104]* втрачається відповідність числа між позиційною системою числення і СЗК. Отже для збільшення діапазону представлення чисел в СЗК необхідно збільшити значення основи або ввести додаткову основу.

В таблиці 2.1 зображено перші 20 чисел від 0 до 19 в системі залишкових класів по основам (3,5,7).

Таблиця 2.1 – Переведення чисел із десяткової СЧ в СЗК

Десяткове число	Код в СЗК по основам (3,5,7)	Десяткове число	Код в СЗК по основам (3,5,7)
0	(0,0,0)	10	(1,0,3)
1	(1,1,1)	11	(2,1,4)
2	(2,2,2)	12	(0,2,5)
3	(0,3,3)	13	(1,3,6)
4	(1,4,4)	14	(2,4,0)
5	(2,0,5)	15	(0,0,1)

6	(0,1,6)	16	(1,1,2)
7	(1,2,0)	17	(2,2,3)
8	(2,3,1)	18	(0,3,4)
9	(0,4,2)	19	(1,4,5)

Джерело: розроблено автором

2.2 Переведення чисел з системи залишкових класів в позиційну систему числення

Переведення чисел з системи залишкових класів в позиційну систему числення виконується декількома способами. Розглянемо декілька з них, а саме:

- основою перетворення є Китайська теорема про залишки або системи ортогональних базисів;
- на базі поліадичного коду.

Спосіб заснований на основі Китайської теоремі про залишки базується на наступній ідеї [10-12]:

$$A=(a_1,a_2,...,a_n)=(a_1,0,...,0)+(0,a_2,...0)+...+(0,0,...a_n)=$$
$$=a_1\cdot(1,0,...,0)+a_2(0,1,...0)+...+a_n(0,0,...1). \quad (2.5)$$

Тобто, нам потрібно знайти систему ортогональних базисів. Ортогональним по основі p_i вважається число, якщо всі цифри в ньому нульові, крім числа по основі p_i, тобто $B_1 = (1, 0, ..., 0)$, $B_2 = (0, 1, ..., 0)$, ..., $B_n = (0, 0, ..., 1)$. Ці вектора шукаються для кожного базису, для цього потрібно вирішити наступне рівняння:

$$(P_i\cdot b_i)(mod p_i)=1; \quad (2.6)$$

$$P_i = P / p_i, \quad (2.7)$$

де P_i - основа числа;

p_i – основа СЗК;

b_i – ваговий коефіцієнт.

В такому випадку позиційне представлення матиме вигляд

$$A = (a_1 \cdot (P_1 \cdot b_1) + a_2 \cdot (P_2 \cdot b_2) + ... + a_n \cdot (P_n \cdot b_n))(modP). \quad (2.8)$$

Наприклад переведемо число *A=(1.2.2)*, яке представлено в системі залишкових класів по основам $p_1=3$, $p_2=5$, $p_3=7$ в позиційну систему числення. Згідно формули (2.4) знаходимо величину P.

$$P = 3 \cdot 5 \cdot 7 = 105.$$

Знаходимо числові основи формула (2.7)

$$P_1 = 105/3 = 35;$$

$$P_2 = 105/5 = 21;$$

$$P_3 = 105/7 = 15.$$

Знаходимо вагові коефіцієнти формула (2.6)

$$(35 \cdot b_1)(mod3) = 1 \rightarrow b_1 = 2;$$

$$(21 \cdot b_2)(mod5) = 1 \rightarrow b_2 = 1;$$

$$(15 \cdot b_3)(mod7) = 1 \rightarrow b_3 = 1.$$

Тепер за формулою 2.8 обчислюємо значення числа *A* в десятковій СЧ

$$A = (1.2.2) = (1 \cdot 35 \cdot 2 + 2 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1)(mod105) = (70+42+30)(mod105) =$$
$$= 142(mod\ 105) = 37$$

Недоліком даного методу є в тому, що він потребує множення, додавання, взяття по модулю великих чисел.

Спосіб який базується на поліадичном коді має ідею, що будь-яке число *A* може бути представлено взаємно простими числами $p_1, p_2, ..., p_n$, [10-12] як:

$$A = a_1 + a_2 \cdot p_1 + a_3 \cdot p_1 \cdot p_2 + ... + a_{n-1} \cdot p_1 \cdot p_2 \cdot ... \cdot p_{n-2} + a_n \cdot p_1 \cdot p_2 \cdot ... \cdot p_{n-1}, \quad (2.9)$$

$$\text{де } 0 < a_i < p_i.\ A(modp_1) = x_1 = a_1; \quad (2.10)$$

$$(A - a_1)(mod p_2) = (x_2 - a_1)(mod p_2) = (a_2 \cdot p_1)(mod p_2) \rightarrow a_2 =$$
$$= ((p_1 - 1)(mod p_2) \cdot (x_2 - a_1))(mod p_2); \quad (2.11)$$

$$(A - a_1 - a_2 p_1)(mod p_3) = (a_3 \cdot p_1 \cdot p_2)(mod p_3) \rightarrow a_3 =$$
$$= ((p_2 - 1)(mod p_3) \cdot ((p_1 - 1)(mod p_3) \cdot (x_3 - a_1) - a_2))(mod p_3); \quad (2.12)$$

Можна замітити, що для використання цього методу потрібні константи типу $(p_i^{-1})(mod p_k)$. А виконання розрахунку a_3 можна проводити з появою a_1.

Приклади перетворення числа *A=(1.2.2),* яке подано у системі залишкових класів по основам $p_1=3$, $p_2=5$, $p_3=7$ в позиційну систему числення.

$$a_1 = x_1 = 1;$$

$$a_2 = ((p_1^{-1})(mod p_2)) \cdot ((x_2 - a_1)(mod p_2)) = ((3^{-1})(mod\ 5) \cdot (1 - 2))(mod\ 5) = 2 \cdot 1 = 2;$$

$$a_3 = ((p_2^{-1})(mod p_3) \cdot ((p_1^{-1})(mod p_3) \cdot (x_3 - a_1) - a_1))(mod p_3) = ((5^{-1})(mod\ 7) \cdot$$
$$\cdot((3^{-1})(mod\ 7) \cdot (2 - 1) - 1))7 = = (3 \cdot (5 \cdot (1) - 1))\ (mod\ 7) = (3 \cdot (4))\ (mod\ 7) = 5;$$

$$A = a_1 + a_2\ p_1 + a_3 \cdot p_1 \cdot p_2 = a_1 + 3 \cdot a_2 + 3 \cdot 5 \cdot a_3 = 1 + 2 \cdot 3 + 3 \cdot 5 \cdot 5 = 37.$$

Щоб знайти значення $(3^{-1})(mod 5)$ потрібно вирішити рівняння $(3 \cdot x)(mod 5) = 1$, де $0 \leq x < 5$.

2.3 Арифметичні операції з додатними числами у СЗК

Додавання і множення в системі залишкових класів можна проводити, якщо числа і результат над числами не виходить за межі діапазону *[0,P)* формула 2.4

Представимо число *A* і *B* у наступному вигляді:

$$A=(\alpha_1, \alpha_2, \ldots, \alpha_n);$$
$$B=(\beta_1, \beta_2, \ldots, \beta_n).$$

Результат додавання і віднімання матиме вигляд:

$$A+B=(\gamma_1, \gamma_2, \ldots, \gamma_n);$$
$$A \cdot B=(\delta_1, \delta_2, \ldots, \delta_n),$$

де

$\gamma_i=(a_i+\beta_i)\ (mod\ p_i);$

$\delta_i=(ai \cdot \beta i)\ (mod\ p_i).$

В якості результату приймається найменший залишок для додавання - формула (2.13), множення - формула (2.14).

$$\gamma_i = a_i + \beta_i - \frac{a_i + \beta_i}{p_i} \cdot p_i \ ; \qquad (2.13)$$

$$\delta_i = a_i \cdot \beta_i - \frac{a_i \cdot \beta_i}{p_i} \cdot p_i \, . \qquad (2.14)$$

Приклади виконання додавання (рис. 2.1) і множення (рис. 2.2) чисел у системі залишкових класів по основам $p_1=3$, $p_2=5$, $p_3=7$.

17 = (2, 2, 3)	27 = (0, 2, 6)
+ 13 = (1, 3, 6)	+ 10 = (1, 0, 3)
30 = (0, 0, 2)	37 = (1, 2, 2)

Рисунок 2.1 – Операція додавання

Джерело: розроблено автором

10 = (1, 0, 3)	3 = (0, 3, 3)
* 5 = (2, 0, 5)	* 4 = (1, 4, 4)
50 = (2, 0, 1)	12 = (0, 2, 5)

Рисунок 2.2 – Операція множення

Джерело: розроблено автором

Операція віднімання в системі залишкових класів не визначена, так як відсутні від'ємні числа. Але якщо виконується умова *(A-B)* лежить в межах *[0,P)*, можна записати [15, 16]

$$\lambda_i = a_i - \beta_i - \left[\frac{a_i + \beta_i}{p_i} \right] \cdot p. \qquad (2.15)$$

Операція віднімання можлива тоді, коли результат цієї операціє буде додатнім. Тоді виконується розрахунок відповідного значення залишку по відповідній основі.

Приклади виконання віднімання (рис. 2.3) чисел у системі залишкових класів по основам $p_1=3$, $p_2=5$, $p_3=7$.

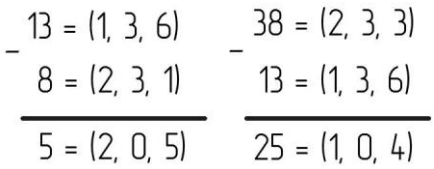

Рисунок 2.3 - Оперція віднімання

Джерело: розроблено автором

2.4 Дослідження властивостей системи залишкових класів

Виходячи з вищесказаного можна сформулювати наступні властивості системи залишкових класів [13]:

Незалежність залишків. Звідси випливає незалежність розрядів числа один від одного і можливість їх незалежної паралельної обробки, тобто ми можемо побудувати систему, яка буде проводити розрахунки над кожною основою окремо і паралельно. Також при виникненні в одному з трактів помилки вона не розповсюджується на сусідні тракти або навіть самоусувається (наприклад множення числа з помилкою на нуль) це дозволяє значно підвищити точність розрахунків.

Рівноправність залишків. Залишок a_i числа $A=(a_1,a_2,...,a_n)$, несе інформацію про все вихідне число. Використовуючи цю властивість з першою дозволяє синтезувати надійність ЕОМ в системі залишкових класів, відповідну модель в позиційній системі числення.

Малорозрядність залишків. В зв’язку з тим, що число представлене у вигляді залишку від ділення на вибрану основу, то отриманий залишок має меншу розрядність, це дозволяє зменшити час на виконання операції. Час виконання операції над числом рахується по найбільшому залишку, так як

виконання обчислення проводиться над кожним із залишків паралельно. Також ця властивість дозволяє використання матричної арифметики.

Система залишкових класів є фундаментальною для теорії і практики машинної арифметики і дозволяє вирішувати задачі недоступні для позиційної СЧ [7]. Перерахуємо основні переваги:

- можливість повного розпаралелення реалізації арифметичних операцій;
- малорозрядність залишків дозволяє ефективно використовувати матричні схеми;
- арифметичні операції додавання, віднімання, множення є модульними і можуть виконуватись за один такт;
- паралельна структура суттєво підвищує надійність розрахунків і забезпечує високу здатність до арифметичної самокорекції.

Також ця система має певні недоліки:

- неможливість візуального порівняння чисел;
- трудність виконання не модульних операцій (перетворення чисел між різними СЧ, визначення знака, знаходження переповнення числа);
- відсутність зручного методу округлення;
- трудність в роботі з числами в від'ємних діапазонах.

2.5 Принципи обробки цифрової інформації в системі залишкових класів

Існує декілька принципів обробки цифрової інформації (рис. 2.4). Розглянемо ці методи детальніше.

Рисунок 2.4 – Принципи обробки цифрової інформації

Джерело: розроблено автором

Пошук шляхів підвищення продуктивності СОІ реального часу привів до необхідності розробки методу табличної (матричної) реалізації модульних операцій. У загальному випадку табличний операційний пристрій (ОП) СОІ для реалізації арифметичних операцій (які реалізуються в унітарному коді) являє собою двухвходовий ПЗП. Для кожного з входів ПЗП кількість вхідних шин для l-байтової ($8l$ двійкових розрядів) СОІ дорівнює 2^{8l}. При цьому загальна кількість логічних схем збігу И у вузлах ПЗП (яке в основному і визначає загальну кількість обладнання табличного операційного пристрою КС) дорівнює $N_{1\ ПСЧ} = 2^{8l} \times 2^{8l} = 2^{16l}$. Очевидно, що таблична реалізація цілочисельних модульних арифметичних операцій в ПСЧ доцільна тільки для значення l=1. Дійсно, в цьому випадку $N_1 = 2^{16} = 65536$, що є прийнятною кількістю обладнання для сучасного розвитку елементної бази. Однак, як зазначалося вище, тенденція розвитку засобів обробки цифрової інформації спрямована на збільшення довжини розрядної сітки КС. Так для випадків l = 4 і l = 8 маємо, що $N_{4ПСЧ}=2^{32}\times2^{32}=2^{64}$ та $N_{8ПСЧ}=2^{64}\times2^{64}=2^{128}$. Якщо врахувати, наприклад, що 2^{32}=4294967296, 2^{64}=18446744073709551616 та $2^{128}\approx3{,}4\times10^{38}$, то очевидно, що табличний метод реалізації арифметичних операцій в ПСЧ практично не застосовний.

Інші результати можна отримати, якщо розглянути СЗК. Дійсно, в загальному випадку, при реалізації алгоритмів модульної обробки інформації для табличного ОП КС необхідно $N_{1СЗК} = \sum_{i=1}^{n} m_i^2$ схем збігу І. Тоді для КС в СЗК з l = 4 і l = 8 відповідно маємо $N_{4СЗК}$=2397 та $N_{8СЗК}$=13275, що цілком прийнятно при реалізації арифметичних операцій додавання, віднімання і множення в СЗК використовуючи таку елементну мікроелектронну базу як БІС, НВІС, ПЛМ або ПЛІС.

Основні переваги табличного варіанту побудови ОП СОІ в СЗК:

- табличні схеми мають високу надійність, так як реалізуються у вигляді компактних ПЗП; в цьому випадку весь тракт ОП СОІ будується за блоковим принципом, що покращує ремонтопридатність СОІ (зменшується час відновлення $Т_в$ СОІ);

- простота побудови табличних схем і дешифраторів, що мають кількість виходів, відповідних основи СЗК;

- висока швидкодія; результат операції може бути отриманий в момент надходження вхідних операндів, тобто в один такт; час виконання арифметичних операцій в СЗК можна порівняти з тактовою частотою обчислювача, що принципово неможливо для позиційних обчислювальних машин при існуючій елементній базі.

3 РОЗРОБКА ТА ДОСЛІДЖЕННЯ СТРУКТУРИ СИСТЕМИ ОБРОБКИ ІНФОРМАЦІЇ ПОБУДОВАНОЇ НА ОСНОВІ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ

3.1 Вибір засобів проєктування

Розробку проєкту будемо проводити на програмованій логічній інтегральній схемі (ПЛІС) з використанням спеціалізованих, для цього, систем автоматичного проєктування (САПР) Active-HDL.

Програмована логічна інтегральна схема, ПЛІС (PLD – Programmable Logic Device) – це електронний компонент, використовується для створення цифрової інтегральної схеми. Головним елементом ПЛІС є кристал. На кристалі розміщуються прості логічні вентилі різних типів і об'єднуючих їх постійні шини провідників. Подібна мікросхема представляє собою універсальну заготовку, в якій розробник може видалити непотрібні зв'язки між вентилями, міняючи тим самим логічну функцію ПЛІС, тим самим створювати унікальний, необхідний логічний пристрій [9]. Розрізняють декілька різновидів ПЛІС: FPGA , CPLD, GAL, PAL. Так як FPGA має більш гнучку архітектуру, більшу функціональність, більше логічних елементів в порівняні з іншими ПЛІС , розробку будемо проводити з урахуванням особливостей цієї системи.

Field Programmable Gate Array (FPGA) – логічна матриця, програмована в умовах експлуатації. Всі ПЛІС програмуються користувачем але відмінність FPGA від інших ПЛІС в загрузці і зберіганні їх структури в швидкодіючих оперативних запам'ятовуючих пристроїв, що дозволяє оперативне перепрограмування безпосередньо в процесі функціонування системи. Опис проєкту для ПЛІС FPGA здійснюється на язику опису апаратури інтегральних схем VHDL (Very high speed integrated circuits Hardware Description Language) або Verilog HDL які не мають особливо принципових відмінностей [8,10].

Язик VHDL є фактично міжнародним стандартом в області автоматизації проєктування цифрових систем. VHDL призначений для специфікації - точного опису проєктованих систем і їх моделювання на початкових етапах

проєктування – алгоритмічному і логічному. На VHDL опис проєкту можна здійснювати на структурному (сукупність елементів і зв'язків між ними) і поведінковому (алгоритм) рівні. За часту використовують обидва варіанти. На низькому рівні описують поведінкову характеристику окремих елементів, на високому рівні показують зв'язки між ними.

Основні етапи розробки і проєктування ПЛІС показано на рисунку 3.1.

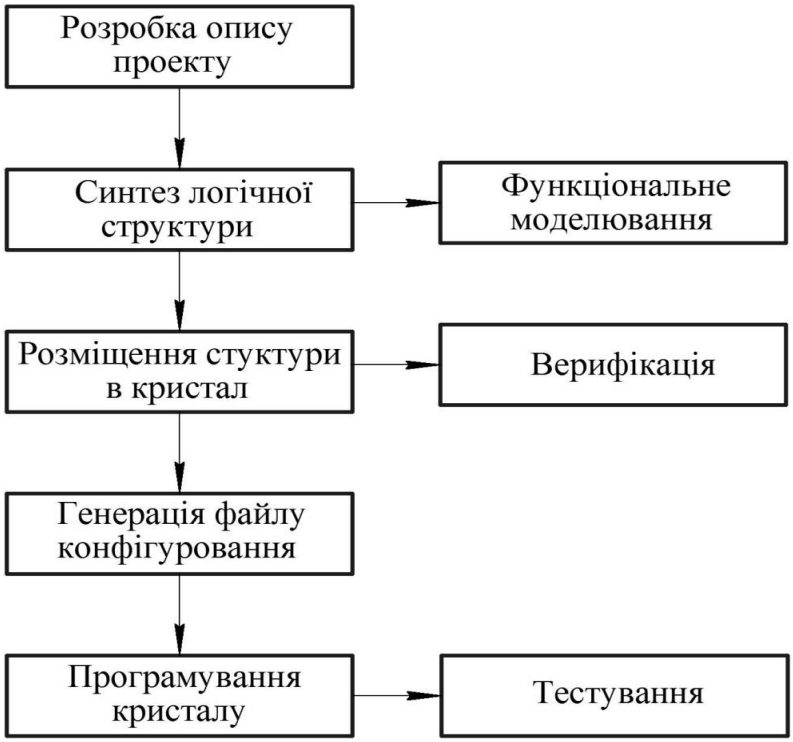

Рисунок 3.1 – Основні етапи проєктування ПЛІС

Джерело: розроблено автором

Опираючись на головні властивості СЗК, а саме незалежність і малорозрядність залишків, використання матричної арифметики є ефективним і передбачуваним. Матрична арифметика дозволяє виконувати арифметичні операції практично в один машинний такт, і зводиться до вибору із пам'яті програми необхідної величини.

Для представлення чисел в СЗК необхідно вибрати систему основ – множина цілих чисел $p_1, p_2, \ldots, p_n$. Треба враховувати що основи повинні бути взаємопрості, і при виході із діапазону $[0,(P-1)]$ втрачається відповідність числа між позиційною системою числення і СЗК. Функціонування

розроблюваної СОІ знаходиться в межах від *0* до 2^8*-1*, тобто в діапазоні [0,255]. Тому вибираємо основи СЗК $p_1=3$, $p_2=4$, $p_3=5$, $p_4=7$. Межі функціонування за цими основами знаходиться як $P=p_1 \cdot p_2 \cdot p_3 \cdot p_4=3 \cdot 4 \cdot 5 \cdot 7=420>255$, що задовольняє вимогам які ставляться. На рисунку 3.2 зображено структуру СОІ.

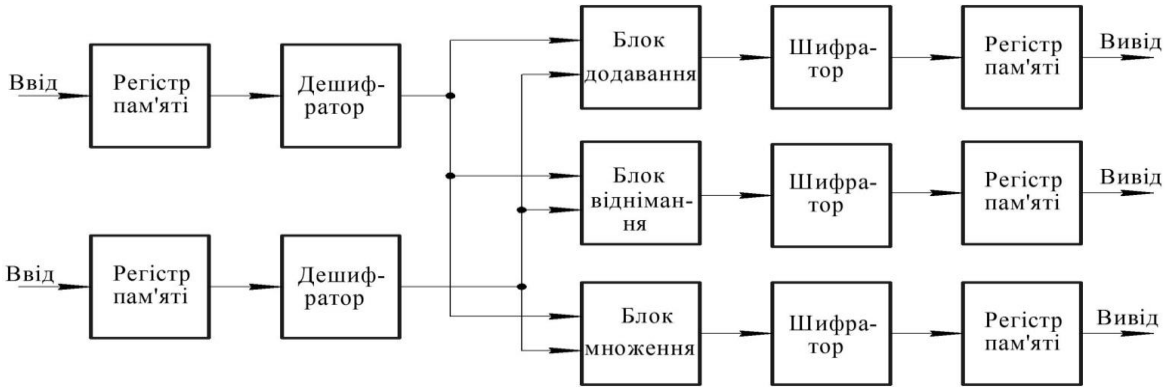

Рисунок 3.2 – Структура системи обробки інформації

Джерело: розроблено автором

Відповідно до кожної основи складаються таблиці відповідностей, в яких зображено результат виконання арифметичних операцій додавання, віднімання, множення, в таблиці 3.1, 3.2, 3.3 наведено результат виконання відповідних арифметичних операцій по основі $p_4=7$.

Таблиця 3.1 - Результат додавання чисел A і B

A\B	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Джерело: розроблено автором

Таблиця 3.2 - Результат віднімання чисел A і B

A\B	0	1	2	3	4	5	6
0	0	6	5	4	3	2	1
1	1	0	6	5	4	3	2
2	2	1	0	6	5	4	3
3	3	2	1	0	6	5	4
4	4	3	2	1	0	6	5
5	5	4	3	2	1	0	6
6	6	5	4	3	2	1	0

Джерело: розроблено автором

Таблиця 3.3 - Результат множення числа A на B

A\B	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Джерело: розроблено автором

3.2 Опис проєкту

Відповідно до розробленої структури системи обробки інформації (рис. 3.2) було розроблено HDL модель (рис. 3.3).

Опишемо структуру розробленої системи по основі $p_4=7$. Так в СОІ входе: регістри пам’яті, шифратори, дешифратори, блоки виконання арифметичних операцій: додавання, віднімання, множення. Розглянемо кожен із них детальніше.

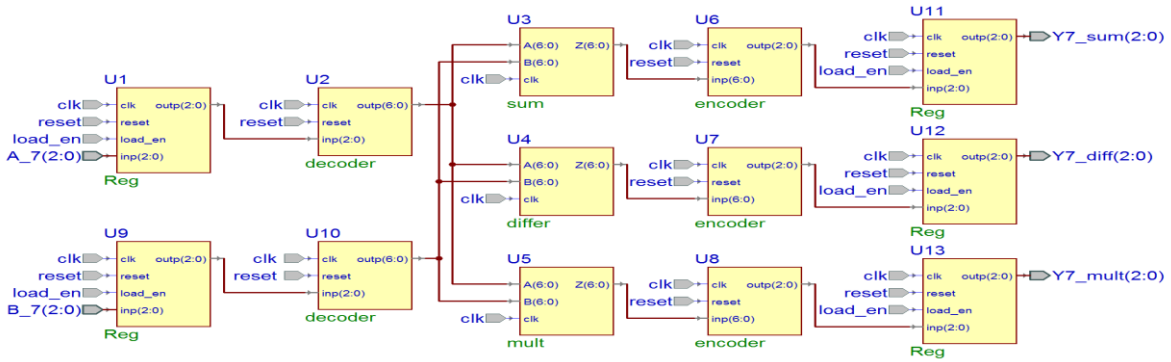

Рисунок 3.3 – HDL-модель COI по основі p_4=7

Джерело: розроблено автором

Регістр пам’яті (рис.3.4) – заносить вхідні данні і результат операції над вхідними даними у пам’ять. Де *inp* – вхід з якого зчитується вхідна інформація. *Load_en* – команда дозволу на запис вхідної інформації з входу *inp*. *Reset* – команда обнулення пам’яті регістра. *Clk* – сигнал, який імітує тактову частоту процесора. *Outp* – вихід, з якого за необхідністю можна зчитати записану інформацію в пам’яті.

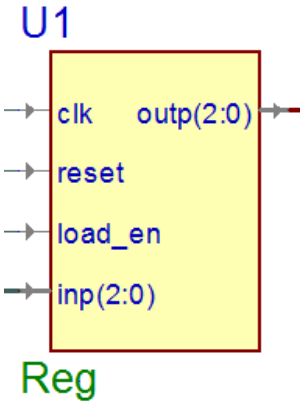

Рисунок 3.4 - Регістр пам’яті

Джерело: розроблено автором

Повний алгоритм регістру пам’яті матиме наступний вигляд:

library ieee;

use ieee.std_logic_1164.all;

```
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity reg is
 port(  clk : in std_logic;
        reset : in std_logic;
        load_en :in std_logic;
        inp : in std_logic_vector(2 downto 0);
        outp : out std_logic_vector(2 downto 0) );
end reg;
architecture reg of reg is
 signal tmp_reg : std_logic_vector(2 downto 0);
begin
 process  (clk)       is
 begin
        if clk'event and clk='1' then
                if reset = '1' then
                        tmp_reg <= (others => '0');
                elsif load_en = '1' then
                        tmp_reg <= inp;
                end if;
        end if;
 end process;
 outp <= tmp_reg;
end reg;
```

Дешифратор (рис. 3.5) – виконує кодування вхідних даних, які надходять з регістру пам’яті, в унарний код. Так дешифратор при отриманні даних на вхід *inp*, видає на вихід outp значення, які відповідають унарному коду вхідного сигналу. *Reset* – команда скидання. *Clk* – сигнал, який імітує тактову частоту процесора.

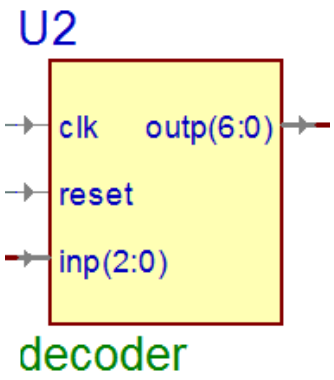

Рисунок 3.5 – Дешифратор

Джерело: розроблено автором

Повний код дешифратора матиме наступний вигляд:

```
library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;
        entity decoder is
          port(  clk : in std_logic;
          reset : in std_logic;
          inp: in std_logic_vector(2 downto 0);
          outp : out std_logic_vector(6 downto 0) );
        end decoder;
        architecture decoder of decoder is
        begin
          process(clk)
        begin
          if clk'event and clk ='1' then
                if reset = '1' then
                        outp <= (others => '0');
                else
                        case inp is
                                when "000" => outp <= "0000001";
                                when "001" => outp <= "0000010";
```

```
                    when "010" => outp <= "0000100";
                    when "011" => outp <= "0001000";
                    when "100" => outp <= "0010000";
                    when "101" => outp <= "0100000";
                    when "110" => outp <= "1000000";
                    when others => outp <= "0000000";
                end case;
            end if;
        end if;
    end process;
    end decoder;
```

Блок виконання операції додавання (рис. 3.6) – виконує додавання двох вхідних величин які поступають на вхід *A* і *B*. *Load_en* – дозвіл на виконання операції. *Clk* – сигнал, який імітує тактову частоту. Згідно таблиці 3.1 складається правила функціонування.

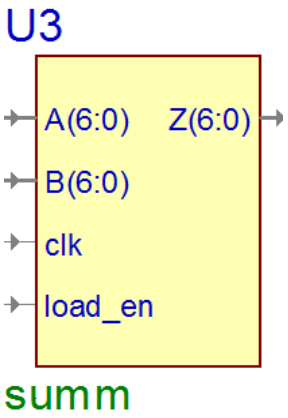

Рисунок 3.6 – Блок додавання

Джерело: розроблено автором

Код блоку додавання матиме наступний вигляд:

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
    use ieee.std_logic_unsigned.all;
```

```
entity sum is
  port(    A: in std_logic_vector(6 downto 0);
    B: in std_logic_vector(6 downto 0);
    clk: in std_logic;
    load_en: in std_logic;
    Z: out std_logic_vector(6 downto 0);
    reset: in std_logic );
end sum;
architecture sum of sum is
begin
  process (clk)
begin
        if clk'event and clk='1' then
            if reset = '1' then
            elsif load_en = '1' then
Z(0)<=(A(0) and B(0)) or (A(1) and B(6)) or (A(2) and B(5)) or (A(3) and
B(4)) or (A(4) and B(3)) or (A(5) and B(2)) or (A(6) and B(1)) ;
Z(1)<=(A(0) and B(1)) or (A(1) and B(0)) or (A(2) and B(6)) or (A(3) and
B(5)) or (A(4) and B(4)) or (A(5) and B(3)) or (A(6) and B(2)) ;
Z(2)<=(A(0) and B(2)) or (A(1) and B(1)) or (A(2) and B(0)) or (A(3) and
B(6)) or (A(4) and B(5)) or (A(5) and B(4)) or (A(6) and B(3)) ;
Z(3)<=(A(0) and B(3)) or (A(1) and B(2)) or (A(2) and B(1)) or (A(3) and
B(0)) or (A(4) and B(6)) or (A(5) and B(5)) or (A(6) and B(4)) ;
Z(4)<=(A(0) and B(4)) or (A(1) and B(3)) or (A(2) and B(2)) or (A(3) and
B(1)) or (A(4) and B(0)) or (A(5) and B(6)) or (A(6) and B(5)) ;
Z(5)<=(A(0) and B(5)) or (A(1) and B(4)) or (A(2) and B(3)) or (A(3) and
B(2)) or (A(4) and B(1)) or (A(5) and B(0)) or (A(6) and B(6)) ;
Z(6)<=(A(0) and B(6)) or (A(1) and B(5)) or (A(2) and B(4)) or (A(3) and
B(3)) or (A(4) and B(2)) or (A(5) and B(1)) or (A(6) and B(0)) ;
            end if;
        end if;
```

end process;

end sum;

Блок віднімання (рис. 3.7) – виконує віднімання чисел які надходять на вхід *A* і *B*. *Clk* – сигнал, який імітує тактову частоту. *Load_en* – дозвіл на виконання операції. Згідно таблиці 3.2 складається правила функціонування.

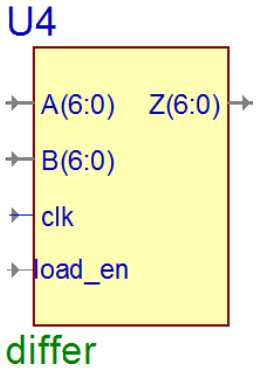

Рисунок 3.7 – Блок віднімання

Джерело: розроблено автором

Код блоку віднімання матиме наступний вигляд:

library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;

entity differ is

port(A: in std_logic_vector(6 downto 0);

B: in std_logic_vector(6 downto 0);

clk: in std_logic;

load_en: in std_logic;

Z: out std_logic_vector(6 downto 0));

end differ;

architecture differ of differ is

begin

process (clk)

begin

```
        if clk'event and clk='1' then
            if reset = '1' then
                Z<= (others => '0');
            elsif load_en = '1' then
Z(0)<=(A(0) and B(0)) or (A(1) and B(1)) or (A(2) and B(2)) or (A(3) and B(3)) or (A(4) and B(4)) or (A(5) and B(5)) or (A(6) and B(6)) ;
Z(1)<=(A(0) and B(6)) or (A(1) and B(0)) or (A(2) and B(1)) or (A(3) and B(2)) or (A(4) and B(3)) or (A(5) and B(4)) or (A(6) and B(5)) ;
Z(2)<=(A(0) and B(5)) or (A(1) and B(6)) or (A(2) and B(0)) or (A(3) and B(1)) or (A(4) and B(2)) or (A(5) and B(3)) or (A(6) and B(4)) ;
Z(3)<=(A(0) and B(4)) or (A(1) and B(5)) or (A(2) and B(6)) or (A(3) and B(0)) or (A(4) and B(1)) or (A(5) and B(2)) or (A(6) and B(3)) ;
Z(4)<=(A(0) and B(3)) or (A(1) and B(4)) or (A(2) and B(5)) or (A(3) and B(6)) or (A(4) and B(0)) or (A(5) and B(1)) or (A(6) and B(2)) ;
Z(5)<=(A(0) and B(2)) or (A(1) and B(3)) or (A(2) and B(4)) or (A(3) and B(5)) or (A(4) and B(5)) or (A(5) and B(0)) or (A(6) and B(1)) ;
Z(6)<=(A(0) and B(1)) or (A(1) and B(2)) or (A(2) and B(3)) or (A(3) and B(4)) or (A(4) and B(6)) or (A(5) and B(6)) or (A(6) and B(0)) ;
            end if;
        end if;
    end process;
end differ;
```

Блок множення (рис 3.8) – виконує множення чисел які надходять на вхід *A* і *B. Clk* – сигнал, який імітує тактову частоту. Згідно таблиці 3.3 складається правила функціонування. Код блоку множення матиме наступний вигляд:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity mult is
  port(   A: in std_logic_vector(6 downto 0);
```

```
    B: in std_logic_vector(6 downto 0);
    clk: in std_logic;
    load_en: in std_logic;
    Z: out std_logic_vector(6 downto 0));
end mult;
architecture mult of mult is
begin
  process (clk)
begin
        if clk'event and clk='1' then
            if reset = '1' then
            Z<= (others => '0');
                elsif load_en = '1' then
Z(0)<=(A(0) and B(0)) or (A(0) and B(1)) or (A(0) and B(2)) or (A(0) and B(3)) or (A(0) and B(4)) or (A(0) and B(5)) or (A(0) and B(6)) or (A(1) and B(0)) or (A(2) and B(0)) or (A(3) and B(0)) or (A(4) and B(0)) or (A(5) and B(0)) or (A(6) and B(0)) ;
Z(1)<=(A(1) and B(1)) or (A(2) and B(4)) or (A(3) and B(5)) or (A(4) and B(2)) or (A(5) and B(3)) or (A(6) and B(6)) ;
Z(2)<=(A(1) and B(2)) or (A(2) and B(1)) or (A(3) and B(3)) or (A(4) and B(4)) or (A(5) and B(6)) or (A(6) and B(5)) ;
Z(3)<=(A(1) and B(3)) or (A(2) and B(5)) or (A(3) and B(1)) or (A(4) and B(6)) or (A(5) and B(2)) or (A(6) and B(4)) ;
Z(4)<=(A(1) and B(4)) or (A(2) and B(2)) or (A(3) and B(6)) or (A(4) and B(1)) or (A(5) and B(5)) or (A(6) and B(3)) ;
Z(5)<=(A(1) and B(5)) or (A(2) and B(6)) or (A(3) and B(4)) or (A(4) and B(3)) or (A(5) and B(1)) or (A(6) and B(2)) ;
Z(6)<=(A(1) and B(6)) or (A(2) and B(3)) or (A(3) and B(2)) or (A(4) and B(5)) or (A(5) and B(4)) or (A(6) and B(1)) ;
            end if;
        end if;
```

end precess;

end mult;

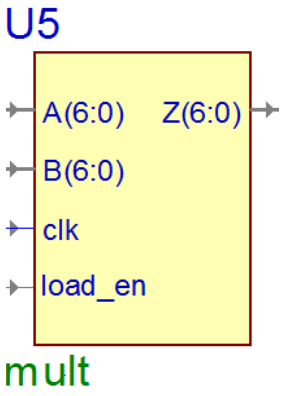

Рисунок 3.8 - Блок множення

Джерело: розроблено автором

Функціонують блоки: додавання, віднімання, множення наступним чином, якщо в якомусь розряді числа *A* і *B* присутня одиниця то на виході *Z*, згідно певної комбінації формується код, який відповідає результату арифметичної операції над даними числами.

Отриманий від арифметичної операції код поступає на вхід *inp* шифратора (рис 3.9), який кодує отриманий код від арифметичної операції у двійковий. Отриманий результат записується в регістр пам'яті, звідки може бути поданий на вихід або використаний для проведення потрібних операцій.

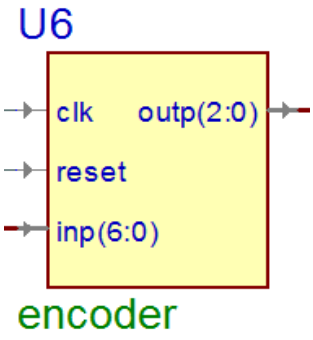

Рисунок 3.9 – Шифратор

Джерело: розроблено автором

Код блоку шифратора матиме наступний вигляд:

```
        library ieee;
        use ieee.std_logic_1164.all;
        use ieee.std_logic_arith.all;
entity encoder is
port( clk : in std_logic;
        reset : in std_logic;
        inp: in std_logic_vector(4 downto 0);
        outp : out std_logic_vector(2 downto 0));
end encoder ;
architecture encoder of encoder is
begin
process(clk)
begin
  if clk'event and clk ='1' then
          if reset = '1' then
                  out_2 <= (othyrs => '0');
          else
                  case inp is
                          when "0000001" => outp <= "000";
                          when "0000010" => outp <= "001";
                          when "0000100" => outp <= "010";
                          when "0001000" => outp <= "011";
                          when "0010000" => outp <= "100";
                          when "0100000" => outp <= "101";
                          when "1000000" => outp <= "110";
                          when others => outp <= "000";
                  end case;
          end if;
  end if;
end process;
end encoder;
```

Підготовлений опис проєкту на HDL використовують для синтезу логічної структури пристрою – тут визначаються логічні елементи і зв’язки між ними, створюється список (Netlist).

В процесі синтезу відбувається контроль коректності опису, відсутність синтаксичних і інших помилок, мінімізація логічних структур з урахуванням використовуємої в ПЛІС елементної бази [25].

Функціональне моделювання спроєктованого пристрою (Simulation), ціллю якого є перевірка відповідності функції. Для цього потрібно підготовки послідовних тестових векторів, для чого можна скористатись вбудованим в симулятор редактором тестових сигналів (Waveform Editor) [25, 26].

На рисунку 3.10 зображено фрагмент моделювання розробленого проєкту При моделюванні проєкту в якості вхідних даних задаються числа які представлені в СЗК.

Так число А яке, представлене в СЗК по основам $p_1=3$, $p_2=4$, $p_3=5$, $p_4=7$ надходить на входи: *A_3, A_4, A_5, A_7* відповідно. Числом В, яке представлене в СЗК по тим же основам $p_1=3$, $p_2=4$, $p_3=5$, $p_4=7$ надходить на входи: *B_3, B_4, B_5, B_7* відповідно.

В результаті виконання процесу обробки на вихід *Y3_sum, Y4_sum, Y5_sum, Y7_sum* надходить результат арифметичного додавання відповідно по основам $p_1=3$, $p_2=4$, $p_3=5$, $p_4=7$.

По аналогії на вихід *Y3_diff, Y4_diff, Y5_diff, Y7_diff* надходить результат віднімання, а на вихід *Y3_mult, Y4_mult, Y5_ mult, Y7_mult* результат множення по тим же основам.

Таблиця модульного додавання по модулю п’ять (табл. 3.4) дає наочне уявлення про алгоритм виконання цієї операції.

Аналогічно виконуються останні арифметичні операції – віднімання (табл. 3.5) та множення (табл. 3.6).

Таблиця 3.4 Модульне додавання по модулю п’ять

b_i	a_i				
	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Джерело: розроблено автором

Таблиця 3.5 Модульне віднімання по модулю п’ять

b_i	a_i				
	0	1	2	3	4
0	0	1	2	3	4
1	4	0	1	2	3
2	3	4	0	1	2
3	2	3	4	0	1
4	1	2	3	4	0

Джерело: розроблено автором

Таблиця 3.6 Модульне множення по модулю п’ять

b_i	a_i				
	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Джерело: розроблено автором

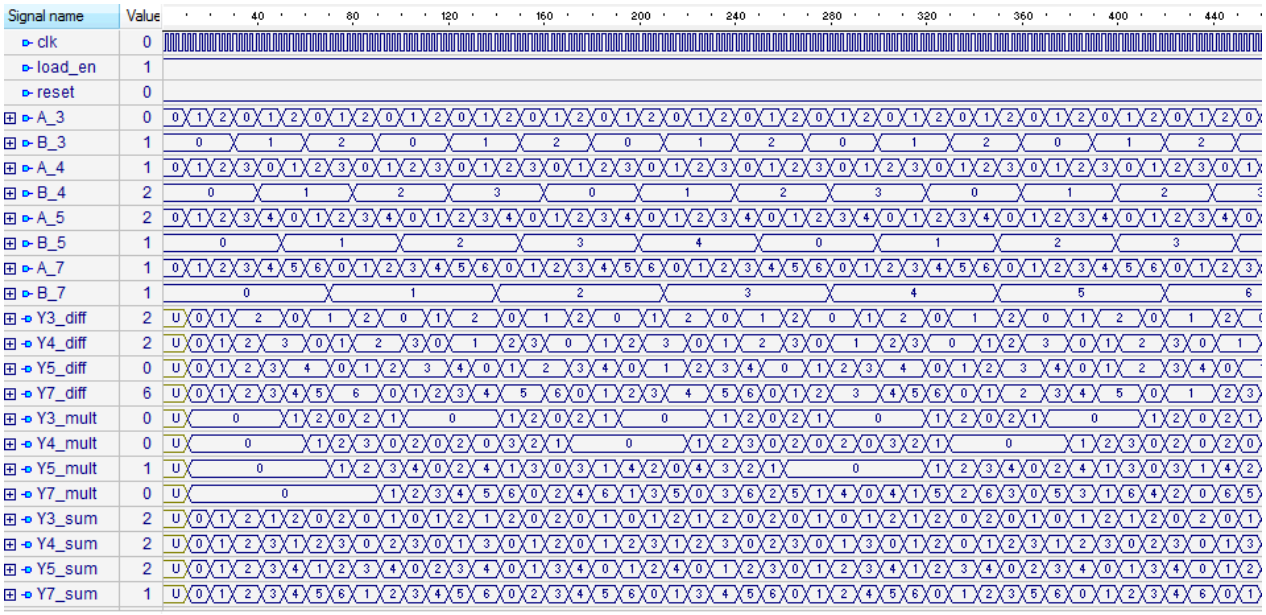

Рисунок 3.10 – Фрагмент імітації HDL – моделі по заданим основам

Джерело: розроблено автором

Розглянемо більш детально результат моделювання. Для цього задамо числа *A=17* і *B=6*. В СЗК число *17=(2,1,2,4), 6=(0,2,1,6).* Результати основних арифметичних операцій додавання, віднімання, множення зображено на рисунку 3.11, 3.12, 3.13 відповідно.

$$\begin{array}{r} 17 = (2,\ 1,\ 2,\ 3) \\ +\ \ 6 = (0,\ 2,\ 1,\ 6) \\ \hline 23 = (2,\ 3,\ 3,\ 2) \end{array}$$

Рисунок 3.11 - Результат додавання чисел 17 і 6

Джерело: розроблено автором

$$\begin{array}{r} 17 = (2,\ 1,\ 2,\ 3) \\ -\ \ 6 = (0,\ 2,\ 1,\ 6) \\ \hline 11 = (2,\ 3,\ 1,\ 4) \end{array}$$

Рисунок 3.12 - Результат віднімання чисел 17 і 6

Джерело: розроблено автором

$$\begin{array}{r} 17 = (2, 1, 2, 3) \\ \times \quad 6 = (0, 2, 1, 6) \\ \hline 42 = (0, 2, 1, 4) \end{array}$$

Рисунок 3.13 - Результат множення чисел 17 і 6

Джерело: розроблено автором

Порівнявши отримані результати при моделюванні (рис. 3.14) і розраховані (рис. 3.11, 3.12, 3.13), видно вірність розрахунків і спроєктованої системи. Так результат додавання в СЗК *(2,3,3,2)*, віднімання – *(2,3,1,4)*, множення – *(0,2,1,4)*.

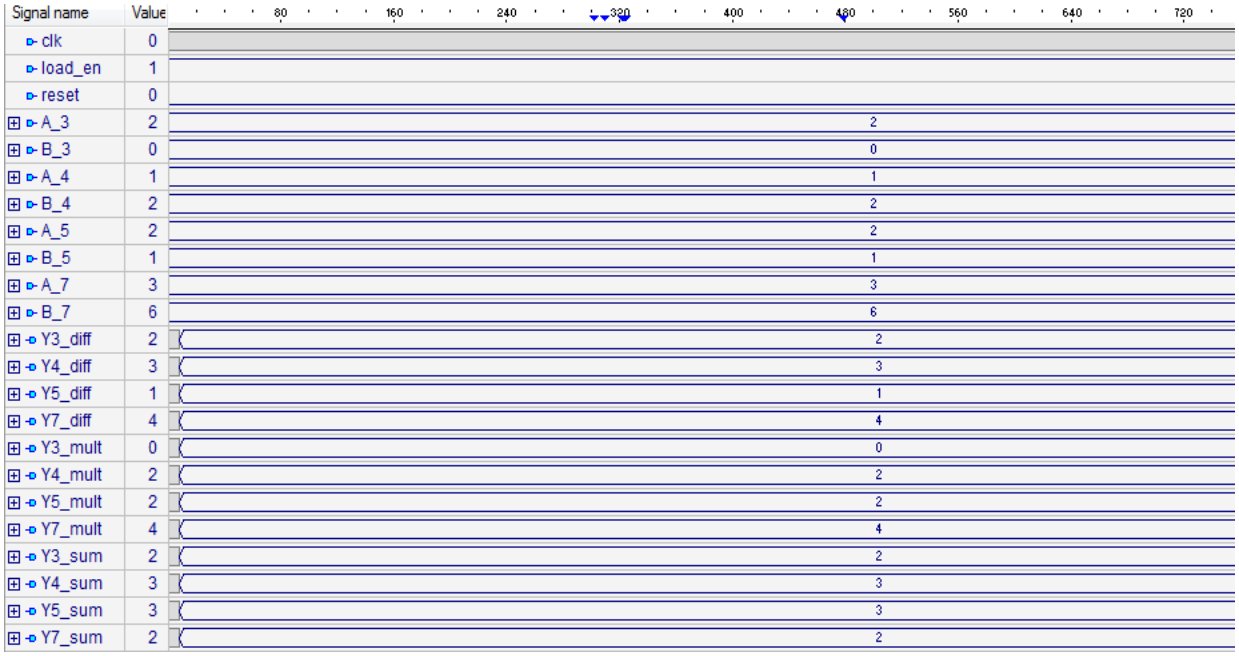

Рисунок 3.14 – Моделювання HDL – моделі по заданим основам.

Джерело: розроблено автором

В третьому розділі було розроблено HDL-модель системи обробки інформації. Моделювання показало високі результати в використанні СЗК під час обробки цифрової інформації. Сформульовано основні вимоги, щодо подальшої апаратної реалізації проєкту.

4 ВИБІР АПАРАТНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ МАТРИЧНОЇ СИСТЕМИ ОБРОБКИ ІНФОРМАЦІЇ

4.1 Основні характеристики ПЛІС

Для реалізації розробленої системи обробки інформації, яка функціонує на основі системи залишкових класів, було вибрано виробника ПЛІС Xilinx, сімейства Virtex-II, серія XC4VSX35-10FF668C, яка має наступні характеристики :

- Високопродуктивні, програмовані користувачем логічні інтегральні схеми з архітектурою FPGA:

- логічна ємність 80 тис. системних вентилів на кристалі;
- внутрішня тактова частота до 420 МГц;
- швидкість обміну даними понад 840 Мб/с по одному контакту;
- введення/виведення.

- Ієрархічна система елементів пам'яті:

- 16 Кбіт розподіленої пам'яті на кристал;
- 144 Кбіт вбудованої пам'яті на кристал, реалізованої на блоках двухпортової ОЗП по 18 Кбіт.

- Інтерфейси до модулів зовнішньої пам'яті:

- інтерфейс до DDR-SDRAM;
- інтерфейс до FSRAM;
- інтерфейс до QDRTM-SRAM;
- інтерфейс до Sigma RAM.

- Спеціалізовані вбудовані модулі для реалізації арифметичних функцій:

- блоки перемножувачів 18x18 біт;
- вбудована логіка прискореного перенесення для реалізації високошвидкісних арифметичних операцій.

- Гнучка архітектура з балансом швидкодії і щільності упаковки логіки:

- 1 024 регістрів/засувок з дозволом тактування і синхронним/асинхронним скиданням і установкою;
- 1024 чотирьох вхідних функціональних генераторів (4-LUT);

- каскадуємі ланцюжки для функцій з великою кількістю входів;

- внутрішні шини з трьома станами.

- Швидкодіючі вбудовані цифрові модулі управління синхронізацією (DCM):

- 8 модулів;

- точне підстроювання фронтів тактуючих сигналів;

- множення, ділення частоти;

- зсув фази з високою роздільною здатністю;

- захист від електромагнітних перешкод;

- 16 мультиплексорів глобальних тактових сигналів.

- Технологія міжз'єднання Active Interconnect:

- сегментована структура трасування четвертого покоління;

- прогнозовані затримки, не залежні від коефіцієнта розгалуження по виходу.

- Програмовані блоки введення/виведення, що підтримують більшість цифрових сигнальних стандартів.

- 120 програмованих користувачем блоків введення/виведення;

- 19 однополюсних і 6 диференціальних стандартів введення/виведення;

- програмована навантажувальна здатність по струму (від 2 мА до 24 мА) на кожен вихід.

- Програмований імпеданс для однополюсних стандартів у кожному блоці введення-виведення.

- Сумісність з шинами PCI-133 МГц, PCI-66 МГц і PCI-33 МГц.

- Диференціальна передача сигналів:

- підтримка передачі сигналів зі швидкістю 840 Мбіт / сек за стандартом LVDS (Low-Voltage Differential Signaling);

- підтримка стандарту BLVDS (Bus LVDS);

- підтримка стандарту LDT (Lightning Data Trans-port);

- підтримка стандарту LVPECL (Low-Voltage Posi-tive Emitter-Coupled Logic);

- вбудовані вхідні і вихідні регістри з подвоєною швидкістю передачі даних.

- Конфігурація кристала зберігається в зовнішньому ПЗП, і завантажується в кристал після включення живлення автоматично або примусово:

- необмежене число циклів завантаження;
- п'ять режимів завантаження;
- шифрування конфігураційної послідовності за стандартом TRIPLE DES;
- підтримка конфігурації за стандартом IEEE 1532;
- можливість часткового ре конфігурування.

- Підтримка команд периферійного сканування, наведених у специфікації Стандарт IEEE 1149.1.
- КМОП - технологія виготовлення з проєктними нормами 0,15 мкм з 8 шарами металізації і швидкодіючими транзисторами з довжиною каналу 0,12 мкм.
- Напруга живлення ядра кристала 1.5 В, блоків введення-виведення від 1.5 В до 3.3 В, залежно від запрограмованого сигнального стандарту.

4.2 Дослідження архітектури Virtex-II

В FPGA Virtex-II використовуються багаторівневі конфігуруємі, різні по складності логічні компоненти, основним з яких є Configurable Logic Block (CLB). Ці блоки під’єднані до зовнішніх контактам мікросхеми через Input/Output Block (IOB), які розміщені по периметру [14]. Архітектуру ПЛІС Virtex-II зображено на рисунку 4.1.

Крім універсальних логічних компонентів, FPGA Virtex-II містить декілька спеціальних функціональних компонентів:

- Block SelectRAM – окремий блок двухпортової оперативної пам’яті загального значення, об’ємом 18 кбіт;
- Multiplier – блоки 18-розрядних перемножувачів;
- Global Clock Mux - мультиплексом глобальних ланцюгів синхронізації.

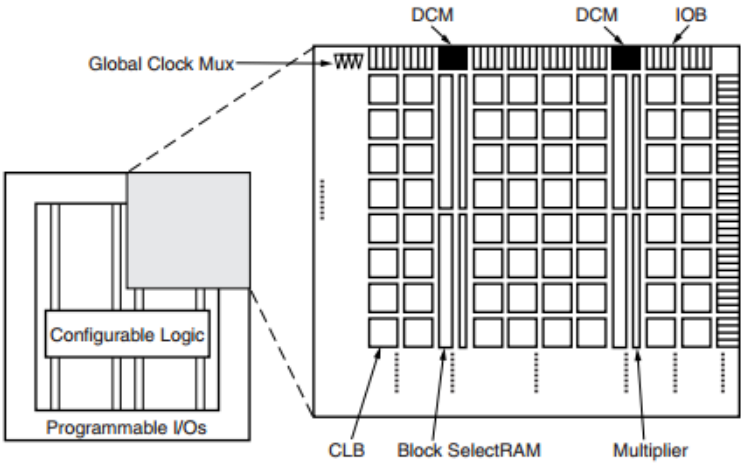

Рисунок 4.1 Архітектура ПЛІС Virtex-II

Джерело: розроблено автором

Для конфігурування використовуються ієрархічні технології активних міжз'єднань з буферизацією – Active Interconnect Technology.

На рисунку 4.2 зображена узагальнена схема з'єднання компонентів FPGA VirtexI.

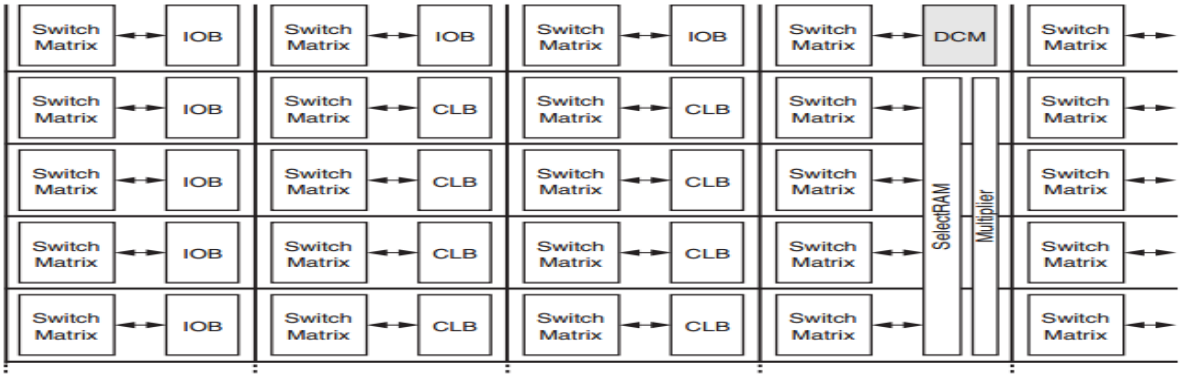

Рисунок 4.2 – Узагальнена схема з'єднання компонентів FPGA Virtex-II

Джерело: розроблено автором

Вертикальні і горизонтальні лінії показують шини матриці глобальних міжз'єднань . Конфігуровані логічні блоки, блоки ввода/вивода і інші компоненти можуть підключатись до шини глобальної синхронізації (по 8 в кожному квадраті). При цьому створюються з'єднання другого рівня.

З’єднання першого рівня можуть бути між любими блоками FPGA і глобальними шинами через матриці комутації (Switch Matrix).

Можливість реалізації логічних функцій визначається структурою конфігурованого логічного блока – CLB (рис. 4.3).

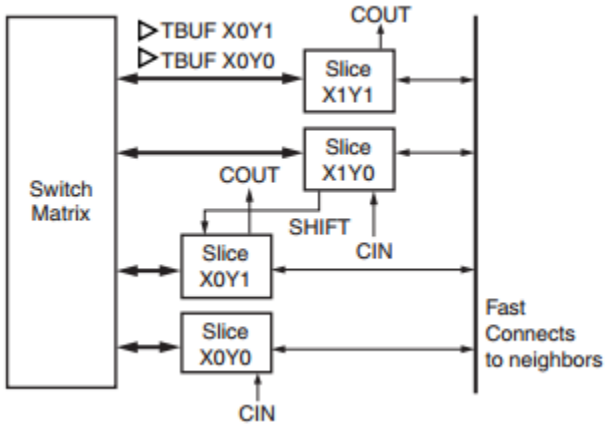

Рисунок 4.3 – Конфігуруємий логічний блок

Джерело: розроблено автором

Кожний блок складається із чотирьох секцій (Slice), зв’язаних з комутуючою матрицею (Switch Matrix), з можливістю організації локальних зворотних зв’язків, також з ланцюгами швидких з’єднань до сусідніх блоків (Fast Connects to neighbors). Логічні схеми, також, мають входи CIN і виходи COUT, для отримання ланцюгів пришвидшеного переносу.

В складі конфігуруємого блока є два драйвера лінії с трьома станами (TBUF), яка дозволяє створювати двохнаправлені магістральні зв’язки між блоками.

Кожна секція Slice (рис 4.4) складається із 4-вхідних генератора логічних функцій, схему арифметичної логіки с ланцюгами переносу для організації багаторозрядних обчислювачів, набір мультиплексорів і два елементи пам’яті.

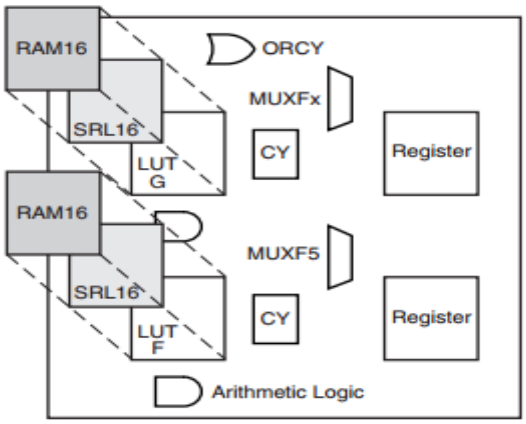

Рисунок 4.4 - Структура секції Slice

Джерело: розроблено автором

В свою чергу, генератор логічних функцій може бути сконфігуровано як 4-входову таблицю перетворення – Look-Up-Table (LUT), 16-бітову розподілену пам’ять (RAM16) або як 16-розрядний зсувний регістр (SRL16). Сигнали на основні виходи секцій можуть надходити з виходу генераторів функцій або через елементи пам’яті (Register). Таблиці перетворення (LUT F і LUT G) реалізують випадкові логічні комбінаційні функції від 4 змінних, а за допомогою мультиплексорів MUXFx і MUXF5 кількість змінних генерованих функцій може збільшено до восьми.

При конфігурації генератора логічних функцій в якості пристрою пам'яті (RAM16) на базі декількох логічних блоків можна створити блок розподіленої оперативної пам’яті з різною організацією від 16*8 біт до 128*1 біт для однопортового доступу і від 16*4 до 64*1 – для двохпортового доступу.

Блок вводу/виводу Input/Output Block (IOB) компонуються в банки (рис. 4.5) і розміщуються по периметру кристала. Кожен блок може бути сконфігурований як вхідний, вихідний або ж двохнаправлений контакт FPGA для несиметричної лінії зв’язку. Два блока IOB використовуються в якості драйвера диференціальної лінії зв’язку. Підтримується більше 30 стандартів обміну інформації по несиметричним або диференціальним лініям зв’язку, забезпечуючи сумісність із стандартними шинами PCI, PCI-X, AGP, CardBus і т.д.

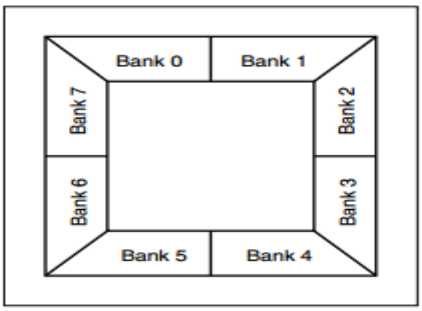

Рисунок 4.5 – Розміщення банок на кристалі

Джерело: розроблено автором

На рисунку 4.6 показано спрощену схему блоку ввода/вивод. Кожний зовнішній контакт мікросхеми з'єднаний з трьома буферами вузла ввода (Input), вивода (Output) і двохнаправленого вводу/виводу (3-State) блока (IOB). Ці вузли призначені для узгодження зовнішніх пристроїв, підключених до FPGA, по електричним рівням сигналів і по імпедансам джерела і приймача сигналу. До того ж ця схема містить два елементи пам'яті (Reg ICK або Reg OCK), необхідних для буферизації даних. Також є підтримка механізмів обміну даним з подвійною швидкістю (Double date rate - DDR).

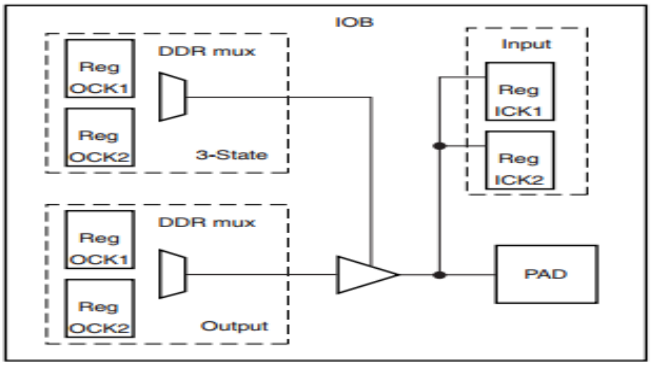

Рисунок 4.6 - Спрощена схема блоку вводу/виводу

Джерело: розроблено автором

При проєктуванні каскадуємих цифрових пристроїв важно забезпечити точність і стабільність синхронної роботи всіх каскадів. Для цього служить

модуль цифрового керування синхронізацією Digital Clock Manager (DCM), який дозволяє синтезувати широку сітку частот синхроімпульсів, синхронізувати їх з зовнішнім або внутрішнім задаючим генератором, формувати необхідні здвиги фаз, компенсувати затримки.

Block SelectRAM блок оперативної пам'яті загального призначення, ємністю 18 кбіт, які можуть бути використанні як одно- або двох- портова пам'ять з організацією 16К*1, 8К*2, 4К*4, 2К*8, 1К*16 або 512*36 біт.

Фірма Xilinx вперше застосувала для зберігання конфігурації спроєктованої на ПЛІС системи статичну оперативну пам'ять, це дозволило швидко змінювати структуру зв'язків і знімає обмеження на кількість циклів реконфігурації. Тобто зміни можуть вноситись оперативно в процесі експлуатації кінцевим користувачем.

4.3 Основні аспекти розміщення проєкту на кристалі

Після моделювання синтезована логічна структура реалізується як фізичний пристрій в кристалі. Вихідними даними для цього служить список ланцюгів і введенні попередньо накази, які сигнали до яких виводів мікросхеми ПЛІС повинні бути підключенні. Реалізація моделі в кристалі здійснюється за допомогою програмного забезпечення Foundation Series.

В процесі «виготовлення» структури кристала послідовно виконуються [28-30]:

- компіляція списку ланцюгів (Translate) в тимчасовий формат службової бази даних;

- створення схеми (Map) на основі доступних ресурсів ПЛІС – виконання конфігурованих логічних блоків, макроячейок, блоків вводу/виводу;

- розміщення схеми в визначені вузли ПЛІС і створення необхідних з'єднань (Place&Route) з використанням матриць глобальних і локальних комутацій трасировкі кристалу;

- розрахунок часових затримок сигналів в реалізованій схемі (Timing) – часове моделювання;

- створення двійкового файлу конфігурації ПЛІС (Configure bitstream), який в кінцевому результаті и являється основним результатом проєктування, цей файл завантажуватиметься в ПЛІС.

Крім створення двійкового файлу конфігурації (Configure bitstream), створюються звіти, які містять повідомлення про помилки і попередження, таблиці відповідностей сигналів і виводів мікросхеми, статичні данні про ресурси які використовуються та інше. По цим звітам проводиться верифікація проєкту – порівняння отриманих результатів з вихідними даними. Додатково надається можливість коректування компоновки проєкту в ПЛІС – реалізовану структуру можна проконтролювати по геометричному зображенні топології кристалу, із цього зображення визвати схему окремого логічного блока і т.д..

На етапі програмування кристалу двійковий файл конфігурації ПЛІС (Configure bitstream) завантажують через JTAG – інтерфейс в ПЛІС, після чого проводиться заключне тестування спроєктованої системи. Тестування може виконуватись методом пограничного сканування або за допомогою комплекту контрольно-вимірювальних приладів. Також, може створюватись оціночні модулі, якщо макет створюваної системи не виготовлений [3,5].

В четвертому розділі описано архітектуру і основні характеристики ПЛІС Virtex-II, на якому можлива реалізація розробленої СОІ. Розкрито головні аспекти розміщення проєкту на кристалі. Передбачувані галузі використання результатів як в системах критичної дії, так і в системах сільськогосподарського виробництва.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1) Результати дослідження існуючих систем обробки інформації показали, що застосування звичайного кодування цифрової інформації не дозволяє суттєво, без нарощення апаратних характеристик, збільшити швидкість і надійність обробки інформації.

2) Результати дослідження кодування цифрової інформації на основі системи залишкових класів показали її перевагу та ефективність під час обробки інформації, сформульовано основні властивості при представленні чисел в СЗК: малорозрядність, незалежність, рівноправність залишків.

3) На основі властивостей СЗК досліджено основні принципи обробки цифрової інформації. Як показали дослідження, час виконання основних арифметичних операцій, при використанні матричного принципу, дорівнює одому машинному такту, тобто вибору із пам’яті програми необхідного значення.

4) Розроблена HDL-модель, яка описує процес і особливості функціонування СОІ, дозволяє вибрати необхідні засобі апаратної реалізації на ПЛІС.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кошман С. О. Алгоритм оптимізації обладнання систем обробки інформації АСКОЕ, що функціонують у модулярній арифметиці // Вісник Харківського державного технічного університету сільського господарства. Проблеми енергозабезпечення та енергозбереження в АПК України. 2005. Вип. 37. С. 240-244.
2. Кошман С. О. Дослідження методів оптимізації структур систем обробки формації // Вісник Харківського національного технічного університету сільського господарства імені Петра Василенка. Проблеми енергозабезпечення та енергозбереження в АПК України. 2007. Вип. 57. С. 111-116.
3. Кошман С. О. Концепція підвищення продуктивності обробки інформації у реальному часі // Вісник Харківського національного технічного університету сільського господарства імені Петра Василенка. Проблеми енергозабезпечення та енергозбереження в АПК України. 2011. Вип. 117. С. 63-65.
4. Кошман С. А., Фурман И. А., Краснобаев В. А. Вариант синтеза процессора в системе остаточных классов // Радиотехника и Информатика. 2003. №2. С. 94-96.
5. Кошман С. А. Особенности матричного способа реализации процессора в СОК // Інтегровані комп’ютерні технології в машинобудуванні ІКТМ-2005: тези доп. Міжнародної науково-технічної конференції. Харків, 2005. С. 314.
6. Краснобаєв В. А., Кошман С. О. Застосування системи залишкових класів у машинній арифметиці // Вісник Харківського державного технічного університету сільського господарства. Проблеми енергозабезпечення та енергозбереження в АПК України. 2003. Вип. 19. С. 134-136.

7. Орлов А.О. Архітектура комп’ютера [Текст]: наукове видання / А.О. Мельник. – Луцьк: Волинська обласна друкарня, 2008. – 470 с.
8. Фурман І. О., Краснобаєв В. А., Кошман С. О. Аналіз табличних алгоритмів реалізації модульних операцій у автоматизованих системах обробки цифрової інформації // Вісник Харківського державного технічного університету сільського господарства. Проблеми енергозабезпечення та енергозбереження в АПК України. 2004. Вип. 27. С. 174-178.
9. Amir Sabbagh Molahosseini, Leonel Seabra de Sousa, Chip-Hong Chang. Embedded Systems Design with Special Arithmetic and Number Systems. Springer International Publishing, 2017. 389p.
10. Hsu J. Y. Computer Architecture: Software Aspects, Coding, and Hardware. CRC Press, 2017. 456 p.
11. Karpinski, M., Kovalchuk, L., Kochan, R., Oliynykov, R., Rodinko, M., Wieclaw, L., 2021. BLOCKCHAIN TECHNOLOGIES: PROBABILITY OF DOUBLE-SPEND ATTACK ON A PROOF-OF-STAKE CONSENSUS. Sensors, 21(19), 6408. https://doi.org/10.3390/s21196408. (дата звернення 12.07.2023)
12. Koshman S. A., Barsov V. I., Krasnobayev V. A., Yaskova K. V., Derenko N. S. Method of bit-by-bit tabular realization of arithmetic operations in the system of residual classes // Радіоелектронні і комп'ютерні системи. 2009. № 5 (39). С. 44-48.
13. Koshman S., Yanko A., Krasnobayev V. Algorithms of data processing in the residual classes system // Problems of Infocommunications Science and Technology PIC S&T 2017: abstr. 4th International Scientific-Practical Conference. Kharkiv, 2017. P. 117-121.
14. Krasnobayev, V.A., Koshman, S.A., 2019. METHOD FOR IMPLEMENTING THE ARITHMETIC OPERATION OF ADDITION IN RESIDUE NUMBER SYSTEM BASED ON THE USE OF THE PRINCIPLE OF CIRCULAR SHIFT. Cybernetics and Systems Analysis

pp. 692-698. https://doi.org/10.1007/s10559-019-00179-8. (дата звернення 24.08.2023)

15. Krasnobayev V., Koshman S., Moroz S., Kalashnikov V. and Kalashnikov V. 2020. DATA ERRORS CONTROL IN THE MODULAR NUMBER SYSTEM BASED ON THE NULLIFICATION PROCEDURE. International Journal of Computing 19(2), pp. 237-246. https://computingonline.net/computing/article/view/1767.(дата звернення: 20.01.2023) Krasnobayev, V. A., Kuznetsov, A. A., Koshman, S. A. and Kuznetsova, K. O., 2020. A METHOD FOR IMPLEMENTING THE OPERATION OF MODULO ADDITION OF THE RESIDUES OF TWO NUMBERS IN THE RESIDUE NUMBER SYSTEM. Cybernetics and Systems Analysis, Vol. 56, No. 6, November, pp. 1029-1038. https://doi.org/10.1007/s10559-020-00323-9. (дата звернення: 17.04.2023)
16. Krasnobayev V. A., Yanko A. S., Kurchanov V. N., Koshman S. A. The analysis of the tasks and algorithms of data integer processing in the residual classes system // Радіоелектронні і комп'ютерні системи. 2016. № 1 (75). С. 19-28.
17. Kuznetsov, A., Oleshko, O., Kuznetsova, K., 2020. ENERGY GAIN FROM ERROR-CORRECTING CODING IN CHANNELS WITH GROUPING ERRORS. Acta Polytech 60, 65–72. https://doi.org/10.14311/AP.2020.60.0065.
18. Kuznetsov, A.A., Stefanovych, O.O., Prokopovych-Tkachenko, D.I., Kuznetsova, K.O., 2019. 3D STEGANOGRAPHY INFORMATION HIDING. Telecommunications and Radio Engineering, vol. 78. https://doi.org/10.1615/TelecomRadEng.v78.i12.30.(дата звернення 19.04.2023)
19. Patterson D. A., Hennessy J. L. Computer organization and design, fifth edition: the hardware/software interface (The Morgan Kaufmann series in computer architecture and design) 5th edition. Morgan Kaufmann, 2013. 800 p.

20. Shekhanin, K., Kuznetsov, A., Krasnobayev, V., Smirnov, O., 2020. Detecting Hidden Information in FAT. International Journal of Computer Network and Information Security(IJCNIS) 12, 33–43. https://doi.org/10.5815/ijcnis.2020.03.04. (дата звернення 20.11.2022)
21. Shiva S. G. Computer Organization, Design, and Architecture: fourth edition. Taylor & Francis, 2000. 736 p.