МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кваліфікаційна наукова
праця на правах рукопису

МОЛЧАНОВА АНАСТАСІЯ АНАТОЛІЇВНА

УДК 004.383 : 004.415.2

ДИСЕРТАЦІЯ

МЕТОДИ І ЗАСОБИ ПРОЄКТУВАННЯ СПЕЦІАЛІЗОВАНИХ КОНВЕЄРНИХ ОБЧИСЛЮВАЧІВ НА БАЗІ ПЛІС ДЛЯ ОБРОБКИ СИГНАЛІВ

123 — Комп’ютерна інженерія
12 — Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ А. А. Молчанова

Науковий керівник:
Романкевич Віталій Олексійович
доктор технічних наук, професор

Київ — 2023

АНОТАЦІЯ

Молчанова А. А. **Методи і засоби проєктування спеціалізованих конвеєрних обчислювачів на базі ПЛІС для обробки сигналів**. — Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 123 — Комп'ютерна інженерія. — Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2023.

Тема дисертації пов'язана з автоматизацією проєктування обчислювальних засобів на базі інтегральних схем надвисокої інтеграції (НВІС) та програмовних логічних інтегральних схем. В даний час складність проєктів обчислювальних систем для НВІС та програмовних логічних інтегральних схем досягла десятків мільйонів вентилів. Причому досі вирішальну роль грає технологія, яка основана на описі проєктів на рівні регістрових передач, продуктивність якої є обмеженою. Розвивається галузь високорівневого синтезу, який ґрунтується на трансляції опису алгоритму в опис на рівні регістрових передач і на порядок прискорює проєктування. Але в наявних засобах такого синтезу необхідно вручну задавати паралелізм алгоритму та особливості відображення в опис обчислювальної системи на рівні регістрових передач, яка часто одержує надмірні апаратні витрати чи продуктивність, що не відповідає заданій.

Невідповідність ефективності наявних засобів високорівневого синтезу складності та продуктивності обчислювальних систем, що проєктуються, представляє актуальну технічну проблему. Обчислювальні системи для цифрової обробки сигналів є такими системами, які вимагають для своєї реалізації проєктування НВІС та програмовних логічних інтегральних схем найбільшої складності і для них ця проблема стає актуальнішою.

Об’єктом дослідження є процес розроблення високопродуктивних паралельних обчислювальних засобів. Предметом дослідження є методи та засоби проєктування спеціалізованих обчислювальних систем конвеєрного типу для цифрової обробки сигналів на базі програмовних логічних інтегральних схем (ПЛІС).

Метою дисертації є підвищення ефективності проєктування конвеєрних обчислювальних систем на основі програмовних логічних інтегральних схем на базі запропонованого методу проєктування спеціалізованих конвеєрних структур на основі генетичного програмування, який дає змогу прискорити проєктування конвеєрних обчислювальних систем і підвищити відношення продуктивність — апаратні витрати завдяки формалізації проєктування і новим алгоритмам пошуку апаратних рішень із мінімізованими апаратними витратами при заданому періоді обчислень.

Для досягнення мети в дисертації виконуються завдання: проаналізувати алгоритми і пристрої цифрової обробки сигналів і сформулювати вимоги до елементної бази й засобів проєктування обчислювальних систем для цифрової обробки сигналів, проаналізувати алгоритмічні моделі та мови опису алгоритмів цифрової обробки сигналів, методи і засоби їхнього відображення в паралельні обчислювальні системи, вибрати найбільш придатні модель та метод відображення, теоретично обґрунтувати та розробити новий метод відображення алгоритму цифрової обробки сигналів у апаратні засоби, які конфігуруються в програмовну логічну інтегральну схему, на основі запропонованого методу розробити засоби автоматизації відображення алгоритмів цифрової обробки сигналів в обчислювальні системи на основі програмовних логічних інтегральних схем, перевірити ефективність розробленого методу під час проєктування низки спеціалізованих обчислювальних систем для вирішення завдань цифрової обробки сигналів.

Під час аналізу алгоритмічних моделей вибрана модель просторового графа синхронних потоків даних (ГСПД) та його відображення у структуру обчислювальної системи і розклад виконання алгоритму як найбільш придатні модель та метод відображення. Просторовий граф синхронних потоків даних представляє собою граф, операторні вершини якого розміщується у просторі з координатами місця, такту виконання і типу оператора таким чином, щоби шуканий розклад виконання алгоритму був коректним. Відображення просторового графа синхронних потоків даних є афінним відображенням у підпростір структур обчислювальних систем та підпростір розкладів.

Наукова новизна роботи. Вперше запропоновано метод проєктування спеціалізованих конвеєрних структур на основі генетичного програмування, який відрізняється тим, що алгоритм цифрової обробки сигналів, який відображається в структуру, задається просторовим ГСПД, задача мінімізації апаратних витрат вирішується із заданими часовими обмеженнями за допомогою еволюційного підходу, який ґрунтується на поданні хромосоми як закодованого ГСПД та відповідних функціях її зміни, а також двохетапному алгоритмі оптимізації. Запропонований метод дає змогу формалізовано вирішувати задачу синтезу обчислювальних систем для цифрової обробки сигналів і завдяки регулюванню ступеня розпаралелювання алгоритму та мінімізації апаратних витрат одержані структури мають високе співвідношення продуктивність — вартість.

Вперше запропоновано спосіб проєктування рекурсивних фільтрів на ПЛІС, який відрізняється тим, що завдяки застосуванню методу відображення просторового ГСПД, використання схем без блоків множення, а також пошуку коефіцієнтів фільтра методом модельованого відпалювання та застосування мови VHDL (VHSIC (very high speed integrated circuits) hardware description language), забезпечується одержання фільтрів з мінімізованими апаратними витратами та високою тактовою частотою.

Практична цінність результатів дисертаційної роботи полягає в тому, що використання нового методу проєктування обчислювальних систем забезпечує зниження трудомісткості і скорочення термінів одержання множини альтернативних оптимізованих структурних рішень, мінімізація використання обчислювальних ресурсів, зокрема пам'яті обчислювальних систем, розроблення високоефективних конвеєрних обчислювальних систем обробки сигналів із мінімізованими апаратними витратами за заданих пропускної спроможності й порядку подання даних у вхідному та вихідному потоках даних.

Розроблено програмний застосунок SDFCAD (synchronous data flow computer-aided design), у якому впроваджений запропонований метод. Цей застосунок дає змогу проєктувальнику описувати за допомогою графа синхронних потоків даних алгоритм цифрової обробки сигналів, моделювати алгоритм з різними степенями паралелізму, забезпечує автоматизований синтез обчислювальної системи із заданими властивостями, яка придатна для подальшої компіляції та конфігурування в програмовну логічну інтегральну схему довільної серії, а також впровадження в замовлену НВІС.

Розроблені з використанням нового методу проєкти обчислювальних систем, такі як процесор дискретного косинусного перетворення, процесор для швидкого перетворення Фур'є, рекурсивні фільтри, модулі обчислення синусоїдальних функцій у програмовних логічних інтегральних схемах, є налаштовуваними обчислювальними модулями, які описані на VHDL, мають високе відношення продуктивність — апаратні витрати та можуть бути впроваджені в нових розробках із мінімальними додатковими часовими і фінансовими витратами. Запропонований спосіб проєктування рекурсивних цифрових фільтрів впроваджено у вебзастосунку, який може бути вільно використаний у практиці проєктування пристроїв цифрової обробки сигналів.

Результати роботи впроваджені у двох НДР, що проводяться в Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» на кафедрі обчислювальної техніки, які присвячені проєктуванню високопродуктивних апаратних і програмних засобів.

Матеріали дисертації є корисними для викладачів і спеціалістів у галузях проєктування апаратних засобів обчислювальної техніки, систем телекомунікацій, зв'язку, вимірювання, радіолокації, штучного інтелекту, засобів мікроелектроніки.

Ключові слова: генетичне програмування, ПЛІС, цифрова обробка сигналів, структурний синтез, граф, граф потоків даних, system on a chip, speedup, graph encoding, scaling, artificial intelligence, performance, resource allocation.

SUMMARY

Molchanova A. A. **Methods and tools of designing the application specific pipelined processors for signal processing based on FPGA**. — Qualification scientific work on the rights of the manuscript.

Dissertation for the degree of Philosophy Doctor in speciality 123 — Computer Engineering. — National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2023.

The topic of the dissertation is related to the computer-aided design of very large scale integration (VLSI) application specific integral circuits (ASICs) and field programmable gate arrays (FPGAs). Currently, the complexity of the processing systems in ASIC and FPGA has reached tens of millions of gates. And still the design technology based on the register transfer level (RTL) description of projects plays the main role. But its productivity is limited. The field of high-level synthesis (HLS) is spreading. It is based on the translation of the algorithm description into the RTL description and speeds up the design by an order of magnitude. However, in the existing HLS tools, it is necessary to control manually the algorithm parallelism and the constraints of mapping the processing system structure to the RTL-description. Such a synthesis often leads to excessive hardware costs or performance that does not meet the limitations.

The discrepancy between the inefficiency of the existing HLS tools and needed hardware volume and performance of the designed system is an urgent technical problem. The digital signal processing (DSP) system is a system that requires the most complex implementation in ASIC and FPGA, and this technical problem becomes even more relevant now.

The object of the research is the design of the high-performance parallel processors. The subject of the research are the methods and tools for the design of the application-specific pipelined DSP processors based on FPGA.

The goal of the dissertation is to increase the efficiency of mapping DSP algorithms into parallel computational systems based on FPGA. The efficiency is increased by the proposed application-specific pipelined structures design method based on the genetic programming. This method helps to accelerate the design of a pipelined processing system and to improve its quality through formalization and automation of the design, and use of new hardware optimization algorithms.

To achieve the goal of the dissertation, the following tasks are performed. The dissertation analyzes the algorithms and devices for DSP, and formulates the requirements to element base and tools for designing DSP processing systems. It analyzes the algorithmic models and DSP algorithm description languages, methods and tools of mapping them to parallel processing systems, chooses the most suitable model and method of mapping. It substantiates theoretically and developes a new method of mapping DSP algorithms into hardware configured in FPGA as well. The development of tools for mapping the DSP algorithms into processing systems based on FPGA, and testing the effectiveness of the developed method when designing a number of application-specific computing systems for the DSP problem solving are performed.

During the analysis of algorithmic models, the model of the spatial synchronous data flow (SDF) and its mapping into the structure of a processing system and the schedule of the algorithm are chosen. It is proven as the most suitable model and method of mapping. Spatial SDF is a graph whose operator nodes are placed in 3D space. The coordinates of this space are the location, execution time and operator type, so that the resulting schedule of the algorithm execution is correct. The mapping of the spatial SDF is an affine mapping into the subspace of computing system structures and the subspace of schedules.

The scientific novelty of the work. A method for application-specific pipelined structures design based on the genetic programming is proposed. The method differs in that the DSP algorithm, which is mapped into a structure, is given

by a spatial SDF, the hardware minimization problem is solved with the given time constraints using an evolutionary approach based on the representation of a chromosome as an encoded SDF and the corresponding functions of its change, as well as a two-stage optimization algorithm.

A method of the IIR filter design based on FPGA is proposed. The method differs in that the use of the spatial SDF mapping method, and the multiplier-free circuits, as well as the filter coefficient search using the simulating annealing method, and the use of VHDL language provide the deriving of the filters with minimal hardware costs and high throughput.

The practical value of the dissertation results is that the use of a new method of the processing system design provides a reduction of both complexity and time of obtaining a variety of alternative optimized structural solutions. It improves the resource utilization effectiveness and processing system memory volume, provides the development of highly pipelined DSP processing systems with minimized hardware costs for a given bandwidth. Besides, it provides the natural order of data in input and output data streams.

The developed SDFCAD software framework implements the proposed method. This application allows the designer to describe the DSP algorithm using the SDF model providing the different degrees of parallelism, automated synthesis of processing systems with specified properties. The resulting processing system description is suitable for further compilation and configuration in FPGA of any series, as well as for implementation in ASIC.

Newly developed processing system projects, such as DCT processor, FFT processor, recursive filters, sine wave generating modules configured in FPGA, are customizable IP cores that are described in VHDL and have a high performance-hardware cost ratio. They can be built in the new projects with minimal additional time and financial costs. The proposed method of designing the recursive digital

filters is implemented in a Web application that can be freely used in the practice of designing the DSP devices.

The results of the work have been implemented in two scientific research works held at the National Technical University of Ukraine “Ihor Sikorsky Kyiv Polytechnic Institute” at the Computer Engineering Department, which are dedicated to the high-performance hardware and software design.

The materials of the dissertation are useful for the lecturers and specialists in the fields of computer hardware design, telecommunication systems, communication, measurement, radar, artificial intelligence, microelectronics.

Key words: genetic programming, FPGA, digital signal processing, structural synthesis, graph, data flow graph, SDF, system on a chip, speedup, graph encoding, scaling, artificial intelligence, performance, resource allocation.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

1. Сергієнко А. М., Романкевич В. О., Сергієнко А. А. Генетичне програмування спеціалізованих конвеєрних пристроїв. *Електронне моделювання*. 2020. Т. 42, №2. С. 25–40.
2. Serhienko A., Sergiyenko A. Modules for pipelined mixed radix FFT processors. *Int. J. of Reconfigurable Computing*. Hindawi Publishing Corp., 2016. V. 2016. P. 1–7.
3. Klymenko I., Tkachenko V., Serhienko A., Kulakov Y. Formalization of the concept of adaptive tasks mapping in the reconfigurable computers on FPGA. *Eastern European Journal of Enterprise Technologies*. 2018. V. 2, No. 9–92. P. 20–28.
4. Sergiyenko A., Serhienko, A. Complexity Reduced IIR Filter Design for FPGA. *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*. Kyiv, 2020. P. 1–4, doi: 10.1109/SAIC51296.2020.9239119.
5. Serhienko A., Sergiyenko A. Romankevich V. Genetic Programming of Pipelined Datapaths for FPGA. *IEEE 40-th Int. Conf. on Electronics and Nanotechnology, (ELNANO)*. Kyiv, 2020. P. 802–806.
6. Serhienko A., Sergiyenko A. VHDL Generation of Optimized IIR Filters. *IEEE 2-nd Ukraine Conference on Electrical and Computer Engineering, (UKRCON).* Lviv, Ukraine, July, 2019. P. 1171–1174.
7. Serhienko A., Sergiyenko A., Simonenko A. A method for synchronous dataflow retiming. *IEEE 1-st Ukraine Conf. on Electrical and Computer Engineering (UKRCON)*. Kyiv, 2017. P. 1015–1018.
8. Сергієнко А. М., Романкевич В. О., Сергієнко А. А. Генетичне програмування опису конвеєра даних мовою VHDL. *Прикладна математика та комп'ютинг* : тези десятої наук. конф. магістрантів та аспірантів ПМК’2018 (Київ, 21–23 бер. 2018). Київ, 2018. С. 153–157.

9. Сергієнко А. М., Хусейн К. С., Сергієнко А. А. Фільтри зі скінченною характеристикою з мінімізованими апаратними витратами. *Безпека, Відмовостійкість, Інтелект* : тези наук. конф. Київ, НТУУ «КПІ», 2018. С. 99–103.
10. Serhienko A., Sergiyenko A. Digital Filter Design using VHDL. *High Performance Computing* (HPC-UA 2018) : Proc. 5-th Int. Conf. Kyiv, 2018. P. 123–126.
11. Serhienko A., Sergiyenko A. Method of the Digital Filter Design using VHDL. *Winter InfoCom Advanced Solutions 2016* : Proc. Int. Conf. Kyiv, 2016. P. 68–69.
12. Сергієнко А. А. Реалізація конвеєрних процесорів швидкого перетворення Фур'є у ПЛІС. *Прикладна математика та комп'ютинг — 2016* : тези наукової конференції. Київ, 2016. С. 128–131.
13. Сергієнко А. А., Сергієнко А. М. Бібліотека модулів для швидкого перетворення Фур'є. *Інформатика та обчислювальна техніка – IOT-2016* : тези наук. конф. студентів, магістрантів та аспірантів. Київ, 2016. С. 114–117.
14. Сергієнко А. А. Сергієнко А. М. Набір модулів для швидкого перетворення Фур'є. *Infocom Advanced Solutions 2015* : тези міжнар. наук.-практ. конф. Київ, 2015. С. 52–53.
15. Сергієнко А. А., Клятченко Я. М. Процесор швидкого перетворення Фур'є за простою основою. *Сучасні методи, інформаційне та програмне забезпечення систем управління організаційно-технологічними комплексами* : тези всеукр. наук.-практ. інтернет-конф. Луцьк, 2015. С. 21–23.
16. Serhienko A., Sergiyenko A. Computing Pythagorean triples in FPGA. *High Performance Computing*, (HPC-UA'2013) : Proc. 3-d Int. Conf. Kyiv, 2013. P. 347–349.

17. Сергиенко А. М., Сергиенко А. А. Моделирование волновых процессов с помощью волновых фильтров. *Моделювання-2018* : тези міжн. наук. конф. Київ, 2018, С. 224–227.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ГА — генетичний алгоритм

ГП — генетичне програмування

ГПД — граф потоків даних

ГСПД — граф синхронних потоків даних

ДКП — дискретне косинусне перетворення

ДМП — дискретне математичне програмування

ДПФ — дискретне перетворення Фур’є

ЕА — еволюційний алгоритм

ЕМ — еволюційний метод

ІДКП — інверсне дискретне косинусне перетворення

ІС — інтегральна схема

КА — конфігурація алгоритму

КС — конфігурація структури

КП — конфігурація передування

ЛТ — логічна таблиця

НВІС — надвелика інтегральна схема

НІХ — нескінченна імпульсна характеристика

ОП — обчислювальний пристрій

ОС — обчислювальна система

ПЕ — процесорний елемент

ПЗП — постійний запам’ятовувальний пристрій

ПЛІС — програмовна логічна інтегральна схема

САПР — система автоматизованого проєктування

СІХ — скінченна імпульсна характеристика

СНК — система на кристалі

ЦЛП — цілочисельне лінійне програмування

ЦОС — цифрова обробка сигналів

ШПФ — швидке перетворення Фур’є

ALAP — as long as possible

ALDCT — Aldec discrete cosine transform

AOC — Altera OpenCL compiler

ARM — advanced RISC (reduced instruction set computer) machine

ASAP — as soon as possible

CAL — Cal Actor Language

CADDY — computer-aided design of non-standard databases

CDFG — control data flow graph

CDMA — code division multiple access

CFG — control flow graph

CLB — configurable logic block

CLBS — configurable logic block slice

CORDIC — coordinate rotation digital computer

DAG — directed acyclic graph

DFG — data flow graph

DOM — document object model

DSP — digital signal processing

ECGP — embedded cartesian genetic programming

ESL — electronic system level

FIFO — first in first out

FPGA — field-programmable gate array

FSM — finite-state machine

GMD-GP — genetic marker diversity algorithm for genetic programming

GPS — global positioning system

GSGP — geometric semantic genetic programming

HDL — hardware description language

HLS — high-level synthesis

IIR — infinite impulse response

IoT — Internet of things

JPEG — Joint Photographic Experts Group

LUT — look-up table

MPEG — Moving Picture Experts Group

NSGA — non-dominated sorting genetic algorithm

OFDM — orthogonal frequency-division multiplexing

OSCI — Open SystemC Initiative

PDGP — parallel distributed genetic programming

RAM — random access memory

RTL — register transfer level

RVC-CAL — reconfigurable video coding Cal Actor Language

SADF — scenario-aware data flow

SDF — synchronous data flow

SDFCAD — synchronous data flow computer-aided design

SFL — structured function description language

SPARCS — synthesis and partitioning for adaptive reconfigurable computing systems

SPL — search processing language

SPW — signal processing workstation

SSEA — steady-state evolutionary algorithm

SVG — scaled vector graphic

TLM — transaction-level modeling

UML — unified modeling language

VEGA — vector evaluated genetic algorithm

VHDL — VHSIC (very high speed integrated circuits) hardware description language

XML — extensible markup language

ВСТУП

Актуальність теми

У сучасних засобах обчислювальної техніки, таких, як персональні комп'ютери, засоби мобільного зв'язку та інтернету речей (IoT) реалізовані різноманітні алгоритми цифрової обробки сигналів. Системи цифрової обробки сигналів, керування та інші обчислювальні системи виконують алгоритми обробки потоків даних у реальному часі, тобто вони повинні реагувати на зовнішні дані зі швидкодією, яка є не меншою за швидкодію, що задається їхнім оточенням. Тому вони не допускають втрати вхідних даних через їхню обмежену швидкодію. Водночас від обчислювальних систем вимагається продуктивність від 1 до 1000 і більше млрд операцій за секунду [127, 160]. Крім того, до сучасних обчислювальних систем для цифрової обробки сигналів висуваються такі суперечливі вимоги, як висока продуктивність, низьке енергоспоживання, низька ціна і висока надійність. Причому, не зважаючи на зростання складності проєктів обчислювальних систем, необхідно прискорювати процес проєктування таких систем унаслідок ринкової конкуренції [43].

Програмовані логічні інтегральні схеми мають переваги як надвеликих інтегральних схем (НВІС), так і універсальних процесорів. Так само, як і у НВІС, у програмовних логічних інтегральних схемах обчислення реалізовані паралельно, а як у процесорах — алгоритми можуть програмуватись. Причому, на відміну від НВІС, у програмовних логічних інтегральних схемах виконується швидке й багаторазове програмування внутрішньої структури, а на відміну від універсальних процесорів алгоритми виконуються з на порядок меншим енергоспоживанням. Завдяки цьому, зараз програмовні логічні інтегральні схеми є обчислювальним середовищем для побудови спецпроцесорів цифрової обробки сигналів, а також прискорювачів універсальних процесорів, яке адаптується до особливостей задачі [107].

Програмування програмовних логічних інтегральних схем вимагає, щоби програміст одночасно залучив велику множину апаратних ресурсів, що паралельно виконують обчислення. Це вимагає від нього нового мислення в паралельних категоріях. У результаті розроблення проєктів для програмовних логічних інтегральних схем є дуже складною і відповідальною працею, для виконання якої треба бути досвідченим і високопрофесійним програмістом. Причому зі зростанням складності проєктів істотно зростає трудомісткість проєктування. Щоб ефективно розробляти проєкти обчислювальних систем на базі НВІС або програмовних логічних інтегральних схем зі складністю більш 200–500 тис. вентилів, необхідно розробляти нові методи й засоби відображення алгоритмів у обчислювальне середовище [111].

Згідно з загальноприйнятою методологією, відображення алгоритму в обчислювальну систему, тобто, планування обчислень, виконують у три етапи. На першому етапі вибирають множину апаратних ресурсів. На другому — складається розклад виконання операторів. І на третьому етапі оператори алгоритму розподіляються серед ресурсів [161, 230]. Таке відображення алгоритму часто призводить до рішень, які є далекими від оптимальних. Це пояснюється тим, що ці етапи мають суперечливі цілі: вибір ресурсів мінімізує апаратні витрати, а складання розкладу — мінімізує тривалість обчислень, збільшуючи апаратні витрати.

Наявна велика множина робіт у галузі теорії відображення паралельних алгоритмів в обчислювальні системи, які орієнтовані на реалізацію на основі програмовних логічних інтегральних схем і НВІС, як, наприклад, [3, 43, 58, 64]. Але лише деякі з цих методів набули широкого застосування в комерційних системах автоматизованого проєктування (САПР) для високорівневого синтезу, таких як Celoxica, Cadence C2S, Mathworks Simulink System Generator, Xilinx AccelDSP, Synplicity SynplifyDSP [107, 222]. Однак, вони далекі від досконалості. Багато з них ґрунтуються на одиничному

відображенні алгоритму. Водночас складність алгоритму, який може бути реалізований, залежить від обсягу обчислювальних ресурсів наявних програмовних логічних інтегральних схем. Дуже поширений підхід ґрунтується на тому, що шукана структура обчислювальної системи компілюється з обчислювальних макроблоків, які виконують стандартні процедури з поширених бібліотек, таких як OpenCE, OpenCV, TensorFlow. Для цього макроблоки мають бути заздалегідь розроблені й описані мовою Verilog, VHDL (VHSIC (very high speed integrated circuits) hardware description language) чи SystemC або згенеровані спеціальними програмами-генераторами макроблоків. В обох випадках підготовка макроблоків вимагає трудомісткої кваліфікованої праці розробників [77, 107].

Отже, завдання проєктування обчислювальних систем для програмовних логічних інтегральних схем і HBIC, які ставить сучасний етап розвитку цифрової обробки сигналів, не можуть бути виконані без розроблення нових методів і засобів відображення алгоритмів цифрової обробки сигналів. Таке розроблення можливе після аналізу наявних моделей і засобів завдання алгоритмів цифрової обробки сигналів, вибору такої обчислювальної моделі, яка забезпечувала б простоту свого аналізу і високу ефективність відображення алгоритму в апаратні засоби обчислювальної системи, а також розроблення методів оптимізації такого відображення.

Отже, дослідження й розроблення моделей, методів і засобів відображення алгоритмів цифрової обробки сигналів в апаратні обчислювальні системи, а також їхня практична реалізація й апробація є актуальними, мають теоретичний і практичний інтерес.

Метою дисертації є підвищення ефективності проєктування конвеєрних обчислювальних систем на основі програмовних логічних інтегральних схем на базі запропонованого методу проєктування спеціалізованих конвеєрних структур на основі генетичного програмування,

який дає змогу прискорити проєктування конвеєрних обчислювальних систем і підвищити відношення продуктивність — апаратні витрати завдяки формалізації проєктування і новим алгоритмам пошуку апаратних рішень із мінімізованими апаратними витратами при заданому періоді обчислень.

Об'єктом дослідження є процес розроблення високопродуктивних паралельних обчислювальних засобів.

Предметом дослідження є методи та засоби проєктування спеціалізованих обчислювальних систем конвеєрного типу для цифрової обробки сигналів на базі програмовних логічних інтегральних схем (ПЛІС).

Для досягнення поставленої мети **в дисертації вирішуються такі завдання**:

1. Проаналізувати завдання, алгоритми і пристрої цифрової обробки сигналів і сформулювати критерії оптимізації для структур, які синтезуються, та засобів проєктування обчислювальних систем для цифрової обробки сигналів.

2. Проаналізувати алгоритмічні моделі та мови опису алгоритмів цифрової обробки сигналів, методів і засобів їхнього відображення в паралельні обчислювальні системи відповідно до сформульованих вимог до них. Вибрати найбільш придатні модель та метод відображення для подальшого дослідження.

3. Теоретично обґрунтувати та розробити новий метод відображення алгоритму цифрової обробки сигналів у апаратні засоби, які конфігуруються в програмовні логічні інтегральні схеми, який відрізняється високим рівнем формалізації, ефективності.

4. На основі запропонованого методу розробити засоби автоматизації відображення алгоритмів цифрової обробки сигналів в обчислювальні системи на основі програмовних логічних інтегральних схем.

5. Перевірити ефективність розробленого методу під час проєктування низки спеціалізованих обчислювальних систем для вирішення широкого кола завдань цифрової обробки сигналів.

6. Розробити новий спосіб проєктування рекурсивних фільтрів на ПЛІС, який забезпечує одержання фільтрів з мінімізованими апаратними витратами та високою тактовою частотою.

На захист виноситься наступне.

1. Метод генетичного програмування просторового графа синхронних потоків даних, включно з представленням хромосоми, функціями репродукції особин, а також двоетапним процесом оптимізації.

2. Спосіб проєктування рекурсивних цифрових фільтрів на основі фазових фільтрів без блоків множення.

3. Структури спеціалізованих процесорів для цифрової обробки сигналів, таких, як процесор дискретного косинусного перетворення, процесор швидкого перетворення Фур'є, процесор для моделювання хвильових процесів у твердому тілі, процесор для генерування синусоїдальної функції, представленої раціональними числами.

Методи досліджень ґрунтуються на використанні теорії графів, теорії множин, теорії алгоритмів, теорії моделювання, матричної алгебри, методів комбінаторної оптимізації, а також теорем, тверджень та висновків, які доведені в дисертації.

Наукова новизна роботи. Вперше запропоновано метод проєктування спеціалізованих конвеєрних структур на основі генетичного програмування, який відрізняється тим, що алгоритм цифрової обробки сигналів, який відображається в структуру, задається просторовим графом синхронних потоків даних (ГСПД), задача мінімізації апаратних витрат вирішується із заданими часовими обмеженнями за допомогою еволюційного підходу, який ґрунтується на поданні хромосоми як закодованого ГСПД та відповідних

функціях її зміни, а також двохетапному алгоритмі оптимізації. Запропонований метод дає змогу формалізовано вирішувати задачу синтезу обчислювальних систем для цифрової обробки сигналів і завдяки регулюванню ступеня розпаралелювання алгоритму та мінімізації апаратних витрат одержані структури мають високе співвідношення продуктивність — вартість.

Вперше запропоновано спосіб проєктування рекурсивних фільтрів на ПЛІС, який відрізняється тим, що завдяки застосуванню методу відображення просторового ГСПД, використання схем без блоків множення, а також пошуку коефіцієнтів фільтра методом модельованого відпалювання та застосування мови VHDL, забезпечується одержання фільтрів з мінімізованими апаратними витратами та високою тактовою частотою.

Практична цінність результатів дисертаційної роботи полягає в тому, що використання запропонованого методу проєктування спеціалізованих конвеєрних структур забезпечує можливості: 1) зниження трудомісткості і скорочення термінів (приблизно вдвічі) одержання множини альтернативних оптимізованих структурних рішень; 2) покращення використання обчислювальних ресурсів і пам’яті обчислювальних систем; 3) розроблення високоефективних конвеєрних обчислювальних систем обробки сигналів із мінімізованими апаратними витратами за заданих пропускної спроможності й порядку подання даних у вхідному та вихідному потоках даних.

Розроблений програмний застосунок SDFCAD (synchronous data flow computer-aided design), у якому впроваджений запропонований метод. Цей застосунок, по-перше, дає змогу проєктувальнику описувати за допомогою графа синхронних потоків даних як алгоритм загалом, так і його складові підзадачі аж до оператора, використовуючи довільні типи даних та операції. По-друге, застосунок надає проєктувальнику можливість моделювання алгоритму як із максимальним ступенем паралелізму, так і з заданим. І по-

третє, він забезпечує автоматизований синтез обчислювальної системи із заданими властивостями, яка придатна для подальшої компіляції та конфігурування в програмовну логічну інтегральну схему довільної серії, а також впровадження в замовлену НВІС.

Проєкти обчислювальних систем для вирішення задач цифрової обробки сигналів, такі як процесор дискретного косинусного перетворення, процесор для швидкого перетворення Фур'є, рекурсивні фільтри, модулі обчислення синусоїдальних функцій у програмовних логічних інтегральних схемах, є налаштовуваними обчислювальними модулями, які описані на VHDL, мають високе відношення продуктивність — апаратні витрати та можуть бути впроваджені в різноманітних нових розробках із мінімальними додатковими часовими і фінансовими витратами. Запропонований метод проєктування рекурсивних цифрових фільтрів впроваджено у вебзастосунку [204], який може бути вільно використаний у практиці проєктування пристроїв цифрової обробки сигналів.

Зв'язок дисертації з науково-дослідними роботами. Результати роботи впроваджені у двох науково-дослідних роботах на кафедрі обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»: 1) НДР №2863-п «Створення засобів проектування та розробка на їх основі високопродуктивних процесорів систем технічного зору», номер державної реєстрації 0115U002326; 2) НДР ФІОТ 3ОТ/2017 «Методи і засоби відображення потокових алгоритмів у конфігуровні комп'ютери», номер державної реєстрації 0119U102212.

Правдивість теорем, тверджень та висновків підтверджується строгими теоретичними доведеннями та результатами випробувань моделей і зразків обчислювальних систем, які побудовані на їхній основі. Основні положення й теоретичні оцінки підтверджені результатами імітаційного моделювання на

комп’ютері та в програмному застосунку SDFCAD. Експериментальна перевірка наукових положень, пропозицій і одержаних результатів виконувалася через проєктування обчислювальних систем за запропонованими методами з використанням мови VHDL та розробленого застосунку SDFCAD з подальшим моделюванням у симуляторі, компілюванням у схему на рівні вентилів та конфігуруванням у програмовні логічні інтегральні схеми фірми Xilinx.

Особистий внесок. Дисертація є результатом самостійних наукових досліджень автора у сфері проєктування конвеєрних обчислювальних систем на основі ПЛІС. Наукові положення та основні результати, які містяться в дисертації, отримані автором самостійно у процесі науково-дослідницької роботи. У роботах [30, 210] автором запропоновано структуру хромосоми та функції розмноження алгоритму генетичного програмування. У роботах [31, 202] автором запропоновано метод генетичного програмування графа синхронних потоків даних. У роботах [206] автору належить покращення кроків оптимізації графа синхронних потоків даних. У роботах [204, 205, 208, 209, 211] автором запропоновано використати еволюційний підхід для синтезу цифрових фільтрів. У роботах [206, 207] автором запропоновані процедури ресинхронізації та згортання графа синхронних потоків даних. У роботі [130] автором уточнені принципи автоматизованого відображення алгоритму в програмовну логічну інтегральну схему. Прикладні аспекти застосування результатів розглянуті в роботах [20, 29, 31, 203].

Апробація роботи. Основні результати дисертаційної роботи були представлені й обговорювалися на 12 міжнародних та регіональних наукових конференціях, у тому числі: 1-, 2-й міжнародних конференціях “*IEEE Ukraine Conference on Electrical and Computer Engineering”* (*UKRCON*'17, Київ, 2017, *UKRCON*'19, Львів, 2019 р.), 40-й міжнародній конференції “IEEE International Conference on Electronics and Nanotechnology” (ELNANO’20, Київ, 2020 р.), 3-

, 5-й міжнародних конференціях “High Performance Computing*”* (HPC-UA’13, Київ, 2013 р., HPC-UA’18, Київ, 2018 р.), 6-й міжнародній конференції “Моделювання-2018” (Київ, 2018 р.), 1-, 3-й міжнародних конференціях “Infocom Advanced Solutions” (Infocom’15, Київ, 2015, Winter Infocom’16, Київ, 2016), 1-й, 3-й Міжнародних науково-практичних конференціях “Безпека. Відмовостійкість. Інтелект” (ICSFTI2018, Київ, 2018 р., ICSFTI2020, Київ, 2020 р.), всеукраїнській науково-практичній інтернет-конференції "Сучасні методи, інформаційне та програмне забезпечення систем управління організаційно-технологічними комплексами" (Луцьк, 2015 р.), 8-й, 10-й наукових конференціях магістрантів та аспірантів “Прикладна математика та комп’ютинг” (ПМК’2016, Київ, 2016 р., ПМК’2016, Київ, 2018 р.), науковій конференції студентів, магістрантів та аспірантів “Інформатика та обчислювальна техніка” (ІОТ-2017, Київ, 2016 р.).

Публікації. За матеріалами дисертації опубліковано 15 наукових робіт. Основні результати викладені в 15 роботах, у тому числі в 1 статті у виданні зі списку МОН [31], у 2 статтях у періодичних виданнях, які входять у наукометричні бази Scopus, WoS [130, 206], а також у 4 статтях у збірниках праць, які входять у наукометричні бази Scopus, WoS [202, 207, 209, 210].

Структура й обсяг роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, які викладені на 161 сторінці тексту, вміщує 29 рисунків і 7 таблиць, список літературних джерел із 240 найменувань.

У вступі обґрунтовується актуальність теми дисертаційної роботи, формулюється мета й завдання дослідження, основні положення, що виносяться на захист.

У першому розділі розглянуті основні завдання, алгоритми і пристрої цифрової обробки сигналів, вимоги до елементної бази й засобів проєктування обчислювальних систем для обробки сигналів. У результаті аналізу наявних алгоритмічних моделей обробки потоків даних був обраний граф синхронних

потоків даних як основна модель для задання алгоритмів цифрової обробки сигналів. Також були проаналізовані методи й засоби високорівневого синтезу обчислювальних систем. На основі зробленого аналізу проєктування обчислювальних систем вироблені вимоги до елементної бази, методів і засобів проєктування обчислювальних систем для реалізації алгоритмів цифрової обробки сигналів, розглянуто спосіб подання графа синхронних потоків даних на мові VHDL та зроблено висновок про перспективність методів еволюційної оптимізації під час синтезу систем цифрової обробки сигналів.

У другому розділі детально розглянуто метод синтезу конвеєрних обчислювальних систем для цифрової обробки сигналів на основі просторового графа синхронних потоків даних, проаналізовані еволюційні методи оптимізації і встановлено, що методи генетичного програмування можуть бути застосовані для оптимізації просторових графів синхронних потоків даних довільних розмірів. Як результат, розроблено новий метод генетичного програмування просторового графа синхронних потоків даних, включно з поданням хромосоми, функції ініціалізації особин, їхньої придатності, селекції та репродукції, а також із двоетапним алгоритмом оптимізації.

У третьому розділі описано розроблення програмного застосунку SDFCAD, для чого обґрунтовано вибір проміжної форми представлення проєкту та його частин, мови програмування, розроблена бібліотека типів та функцій, які необхідні для програмування як вводу-виводу проєкту, так і виконання оптимізації просторового графа синхронних потоків даних запропонованим методом генетичного програмування, дано опис структури матзабезпечення та ходу виконання алгоритму оптимізації.

У четвертому розділі виконано перевірку ефективності програмного застосунку SDFCAD під час проєктування конвеєрних обчислювальних

систем для цифрової обробки сигналів. Детально розглянуто проєктування спецпроцесора для дискретного косинусного перетворення, яке показало переваги нового методу. Також розглянуті питання проєктування таких обчислювальних систем, як конвеєрні процесори швидкого перетворення Фур'є, рекурсивних та нерекурсивних фільтрів, генератора синусоїдальних функцій.

У висновку наведено основні теоретичні та практичні результати роботи, рекомендації з їхнього використання.

РОЗДІЛ 1. МОДЕЛІ, МЕТОДИ ТА ЗАСОБИ ВИСОКОРІВНЕВОГО СИНТЕЗУ ПРОЦЕСОРІВ ДЛЯ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ

1.1 Задачі, алгоритми та пристрої цифрової обробки сигналів

Системи цифрової обробки сигналів (ЦОС), керування відносять до реактивних обчислювальних систем, тобто таких, які повинні реагувати на зовнішні дані зі швидкістю, не меншою за швидкість, що задає оточення, і які не допускають втрати даних через їхню недостатню швидкодію [48, 145]. Зараз важко назвати галузь науки й техніки, де не використовується ЦОС. Найбільш активно ЦОС використовується в транспорті, радіотехніці, зв'язку, вимірювальній, побутовій, медичній та робототехніці. Це передбачає масовий випуск та широку номенклатуру засобів ЦОС.

На кожному новому етапі розвитку ЦОС, який триває приблизно десятиріччя, складність алгоритмів ЦОС і продуктивність апаратури зростала більш ніж на десятковий порядок [15]. Початок останнього етапу припадає на середину минулого десятиріччя. Цей етап характеризується розвитком цифрового зв'язку покоління 5G, інтернету речей, обробки відеозображень надвисокої чіткості (4К, 8К), систем розпізнавання образів та штучного інтелекту. Для їхньої реалізації потрібне невпинне зростання продуктивності обчислювальних систем (ОС). Так, для кодування потоку даних за стандартом H.264 необхідна обчислювальна продуктивність близько $100 \cdot 10^9$ операцій за секунду [55].

Найбільший обсяг обчислень виконують нейронні мережі, які можна вважати крайнім випадком адаптивної фільтрації. Крім того, основною трудомісткою операцією згорткових нейронних мереж є фільтрація, тобто, один із видів ЦОС. Так, протягом 2012–2018 років обчислювальна потужність нейронних мереж зросла в 300 тис. разів, тобто, вона зростає вдвічі кожні 3.5

місяці [37]. Зараз обчислювачі для штучної нейронної мережі мають продуктивність від $0{,}5 \cdot 10^{12}$ до $100 \cdot 10^{12}$ і більше операцій за секунду [224].

Масове впровадження нових алгоритмів ЦОС завжди відставало від моменту їхнього розроблення приблизно на етап. Унікальні колись алгоритми кореляції та швидкого перетворення Фур'є (ШПФ) через десятиріччя знайшли масове впровадження в засобах GPS (global positioning system), CDMA (code division multiple access), OFDM (orthogonal frequency-division multiplexing). Слід очікувати аналогічних проривів стосовно інших алгоритмів ЦОС. Зараз, коли в багатьох випадках для розроблення і використання алгоритмів використовуються персональні комп'ютери, а для апаратної реалізації алгоритмів можна використати програмовні логічні інтегральні схеми (ПЛІС), нові алгоритми ЦОС можна впроваджувати негайно.

Нові алгоритми ЦОС відрізняються не лише збільшенням обсягів обчислень, але й тим, що зростає складність керування, кількість логічних операцій, довжина програм. Тому під час проєктування засобів ЦОС зростає роль програмістів за незменшуваної ролі проєктувальників апаратних засобів.

Отже, нові завдання, які ставить сучасний етап розвитку ЦОС, не можуть бути виконані без розроблення нових методів і засобів відображення алгоритмів у НВІС та ПЛІС. Таке розроблення можливе, насамперед, за умови створення нових засобів високорівневого синтезу. У наступних підрозділах будуть розглянуті основні проблеми й завдання, що стоять під час розроблення нових обчислювальних засобів ЦОС, та наявні моделі й засоби задання алгоритмів такої обробки.

1.2 Вимоги до апаратури ЦОС і засобів її проєктування

1.2.1 Вимоги до проєктування сучасних мікросхем

Теперішній етап розвитку мікроелектроніки характеризується тим, що призупинив дію закон Мура. Цей закон підтримувався принципом Деннарда,

який забезпечував незбільшення споживаної потужності в разі збільшення кількості транзисторів завдяки масштабуванню інтегральної технології. Але це масштабування припинилося близько десяти років тому. Через це зараз найдешевші інтегральні схеми (ІС) виготовляються за проєктними нормами більшими за 20 нм. Отже, не слід покладатися на масштабування інтегральних техпроцесів, а необхідно впроваджувати інновації на основі нових архітектур. Тому потрібне впровадження еквіваленту закону Мура для архітектурних інновацій. І це мають бути архітектури, які є специфічними для конкретної предметної області, тому що вони:

— забезпечують більш ефективний паралелізм для певного класу алгоритмів;

— більш ефективно використовують пропускну спроможність пам'яті;

— у них використовується не максимальна, а мінімально потрібна точність обчислень і, отже, мінімізуються апаратні й енерговитрати;

— у них мінімізовані накладні витрати на організацію обчислювального процесу [114].

Отже, алгоритми ЦОС являють собою певний клас алгоритмів, який вимагає особливої уваги до своєї архітектурної реалізації. Найбільшу ефективність для ЦОС може забезпечити використання спеціалізованих ІС, структура яких відповідає алгоритмам обробки. Зараз одна ІС замінює собою цілу обчислювальну систему з керуючим процесором, і тому її називають системою на кристалі (СНК) (System-on-the-Chip).

Задля швидкого виходу на ринок із новим виробом необхідно збільшувати продуктивність розробників і зменшувати цикл проєктування, намагаючись одержати кристали, що відповідають специфікації, за одну ітерацію їх проєктування [190]. Щільність кристалу, що збільшується, вимагає брати до уваги фізичні ефекти в ньому. Це взаємовплив сигналів у сусідніх провідниках, згасання сигналу в довгих лініях зв'язку (при нормах нижче 0,18

мкм), витрати енергії на розсіювання в провідниках і статичний струм (норми нижче 65 нм), дифракційні явища у фотолітографії та зменшення надійності інтегральних елементів (норми нижче 22 нм). Усе це призводить до збільшення частки витрат на проєктування й тестування нових мікросхем.

Починаючи із середини 90-х років, продуктивність розробників стала відставати від росту складності СНК. Наприклад, СНК із 5 млн вентилів, за умови продуктивності праці розробника 1 тис. вентилів за місяць, 10 розробників мусили б проєктувати впродовж 40 років. Для подолання цього відставання і зменшення вартості розроблення було вироблено три напрями розвитку нових САПР: повторне використання обчислювальних модулів, сумісне апаратно-програмне проєктування й системний синтез [14, 53].

1.2.2 Системний синтез

Під час розроблення пристрою на рівні регістрових передач (register transfer level — RTL) розробники спочатку описують функціональну модель проєкту з деталізацією до мікроархітектури кожного модуля. Опис цієї моделі виконується мовами VHDL і Verilog. RTL-опис є трудомістким, має велику кількість помилок, вимагає час на формування файлів обмежень компілятора-синтезатора й не гарантує одержання оптимального рішення. Причому тільки після одержання схеми на рівні вентилів можливо перевірити критерії оптимальності. Поправки до схеми пристрою призводять до суттєвих змін в RTL-описі, що супроводжується повтором ітерацій оптимізації. Тому RTL-розроблення є принципово ручним, неефективним і незахищеним від помилок [156].

Для прискорення розроблення обчислювальних пристроїв упродовж трьох останніх десятиріч розвивається напрям високорівневого синтезу. В останній час цей напрям одержав сталу назву electronic system level (ESL) design, тобто, розроблення на системному рівні [116]. Дійсно, продуктивність розробників зростає з підвищенням рівня абстракції, як це демонструє практика розроблення програмного забезпечення.

Під час такого синтезу початкову модель обчислювача описують на рівні, який вище від RTL-рівня і який відрізняється тим, що він не містить ні розкладу виконання алгоритму, ні призначення операцій на апаратні ресурси, ні керуючого автомату. Тому обсяг опису на порядок менший, ніж на рівні регістрових передач і моделювання такого опису проходить значно швидше.

Розрізняють поведінковий синтез, під час якого генерується опис пристрою на RTL-рівні для заданих апаратно-часових обмежень та структурний синтез, який іде далі RTL-рівня й закінчується схемою на рівні вентилів.

Під час високорівневого синтезу в циклі оптимізації автоматично виконуються складання розкладу, призначення операцій на ресурси, а також синтез пристрою керування. Значною перевагою високорівневого синтезу є можливість більш точного оцінювання параметрів швидкодії проєкту в комплексі, що включає як схеми обробки даних, так і керуючі автомати разом з керуючим мікропроцесором.

Попри наукові результати, які подавали надію розробникам систем ЦОС, починаючи з 1980-х років, високорівневий синтез усе ще не вважається звичною практикою в проєктуванні апаратних засобів ЦОС. Винятком є технологія графічного програмування така, що використовується в середовищі Matlab-Simulink. Повільні темпи впровадження цієї технології пояснюються бажанням проєктувальників керувати інтегруванням блоків у систему, змінюючи їхні параметри. Але таке бажання досі не є втіленим у засобах САПР у повному обсязі.

Нині потенціал високорівневого синтезу доволі високий, хоча спільнота розробників ще не зупинилася на єдиній мові опису проєкту, яка б знизила бар'єр для виводу інструментів на ринок. А поширені засоби високорівневого синтезу орієнтовані на обмежену цільову архітектуру. Найчастіше це архітектура, що включає вузол-майстер на універсальному процесорі та кілька вузлів-

виконавців у вигляді спецпроцесорів. Поступово такою мовою стає C/C++, у якій паралелізм задається гніздом циклів. Також має перспективи мова Python, програмування якою задач ЦОС полягає в програмуванні скрипту з викликів процедур зі стандартних бібліотек типу OpenCE, TensorFlow. Але в цьому випадку, власне, оптимізації проєкту не відбувається, бо проєкт компілюється зі заздалегідь спроєктованих блоків [107].

Отже, високорівневий синтез — це технологія, без якої неможливе розроблення сучасних СНК. Але вона страждає через труднощі її впровадження, які пов'язані з низькою якістю синтезованих рішень для багатьох алгоритмів, важким процесом її освоєння розробниками. Методи, які використовуються в цій технології, були винайдені 20–50 років тому і вимагають удосконалення.

1.2.3 Мінімізація енергоспоживання

Потужність, що споживається мікросхемою, прямо пропорційна кількості перемикань вентилів за одиницю часу, квадрату напруги живлення V_{cc} та ємності вентилів і провідників, які ведуть до них. З іншого боку, максимальна тактова частота f_{clk} пристрою пропорційна V_{cc} і обернено пропорційна еквівалентній ємності вентилів, що входять до критичного шляху.

Тому найкраща стратегія зменшення енергоспоживання — це зменшення напруги V_{cc}. Але в разі зменшення V_{cc} максимальна частота f_{clk} може зменшитися до рівня, яка є недостатньою для заданої продуктивності схеми, причому збільшується статичний струм через менш надійно закриті транзистори. Статичний струм не залежить від частоти f_{clk} і часом може бути більшим за струм перемикань. Причому частка енергоспоживання від постійного струму пропорційна апаратним витратам.

Для того, щоби продуктивність схеми залишалася високою, необхідно скорочувати довжину критичного шляху, застосовуючи метод конвеєризації, а також зменшуючи навантаження на виходи вентилів (fanout). Конвеєризовані

пристрої споживають менше енергії, позаяк у них за такої самої продуктивності може бути менше V_{cc}. [164].

Отже, найкраща стратегія оптимізації енергоспоживання — це розроблення пристрою, який забезпечує задану продуктивність і має мінімальне відношення: апаратні витрати — максимальна тактова частота f_{clk}. Крім того, такий пристрій займає меншу площу кристалу й забезпечує швидше розроблення проєкту.

1.2.4 Тенденції в розвитку архітектури СНК

Під час розроблення нових СНК необхідне опанування технології безітераційного розроблення кристалів (одержання працездатного кристалу з першого разу). Для виправлення логічних помилок передбачається один з острівців кристалу виконувати як ПЛІС. Зменшення швидкодії ПЛІС у порівнянні з замовленою схемою компенсується завдяки розпаралелюванню обчислень, що досягається в разі застосування високорівневого синтезу.

Впровадження ПЛІС, як частини СНК, забезпечує, по-перше, мініміза-цію енергоспоживання завдяки тому, що високопродуктивні обчислення виконуються апаратно й конфігурація ПЛІС може змінюватися динамічно, й, по-друге, дороблення апаратної реалізації алгоритмів може виконуватись і після виготовлення кристалу, що скорочує цикл розроблення [74, 165].

Передбачається в СНК впроваджувати обчислювальні модулі, що виконують функції інфраструктури: тестування, виправлення збоїв, заміщення непрацездатних ділянок кристалу [240].

1.2.5 Використання ПЛІС у проєктуванні СНК

Використання ПЛІС для реалізації СНК, на відміну від замовленої НВІС, має низку переваг. По-перше, усунення помилки в ПЛІС — майже безкоштовне і триває не більше за один день. По-друге, ПЛІС може бути повторно використана для реалізації різних функцій. По-третє, хоча швидкодія

вентилів у ПЛІС у декілька разів нижча, паралельна реалізація алгоритмів ЦОС у ПЛІС має більшу продуктивність і менше енергоспоживання, ніж у мікропроцесорах. По-четверте, ПЛІС використовуються для реалізації алгоритмів, які можуть мати зміни в майбутньому, причому через дистанційне програмування. До того ж, ПЛІС використовується для реалізації інфраструктури. Також розвиваються методи проєктування надійних вузлів на базі ПЛІС [74, 103].

На сьогодні найбільша частина проєктів на базі ПЛІС і НВІС мають складність від 50 до 500 тис. вентилів. Зростання складності проєктів стримується тим, що необхідно збільшувати штат розробників і використовувати дорожчі засоби проєктування. Одним зі способів розв'язання цієї проблеми є розроблення СНК на базі платформи. Зараз набувають поширення такі платформи, як Xilinx Zynq та Intel SoC FPGA, у складі яких є мікропроцесори ARM. Це дало змогу в кілька разів збільшити кількість розробників ПЛІС завдяки залученню програмістів мовою C++ та впровадженню технології високорівневого синтезу. За прогнозами, ПЛІС-платформи будуть поширеними під час розроблення спеціалізованих ОС принаймні наступні 15 років [107].

У найближчому майбутньому всі розробки СНК будуть виконуватися на ПЛІС, крім тих, що потребують найбільшої продуктивності, серійності виробництва й малої ціни [188]. На міжнародному форумі Next FPGA Platform були задекларовані наступні принципи та ідеї розвитку технології ПЛІС [192]. Історія ПЛІС пройшла три етапи: етап допоміжної логіки (1980–1995), етап інтерфейсів (1996–2017) і зараз почався етап апаратних прискорювачів. На останньому етапі ПЛІС набувають масового впровадження в датацентрах, тож обсяги використаних ПЛІС мають потроїтись. Проте для більш широкого впровадження ПЛІС галузь електроніки та екосистеми повинні запропонувати досконалі технології високорівневого проєктування.

З наведеного вище випливає наступне:

1. Проєктування сучасних НВІС характеризується, з одного боку, ускладненням впроваджуваних алгоритмів ЦОС, з іншого боку, необхідністю врахування фізичних явищ у кристалі, який виготовлено за новими проєктними нормами, що значно ускладнює розроблення СНК.

2. Для підвищення продуктивності розроблення СНК необхідно розвивати засоби САПР у напрямах повторного використання обчислювальних модулів сумісного апаратно-програмного проєктування й системного синтезу.

4. Жодне проєктування СНК не відбувається без розроблення прототипу на ПЛІС. Тепер і в майбутньому основна множина проєктів СНК буде виконуватись на ПЛІС. ПЛІС стають основою апаратних прискорювачів у датацентрах.

5. Щоб ефективно розробляти проєкти СНК на ПЛІС зі складністю понад 200–500 тис. вентилів, необхідне розроблення нових методів і засобів відображення алгоритмів в обчислювальне середовище.

6. З причин того, що швидкодія ПЛІС у 3–10 разів нижча за швидкодію відповідних НВІС і того, що підвищилися вимоги до проєктів, насамперед, необхідно розвивати методи й засоби системного синтезу, які завдяки регулюванню міри розпаралелювання алгоритму забезпечують автоматизований синтез низки оптимізованих пристроїв із різною пропускною спроможністю, зокрема, із необхідною.

7. Найкращою стратегією оптимізації, зокрема мінімізації енергоспоживання, є розроблення конвеєризованого пристрою, який забезпечує задану продуктивність і має мінімальне відношення: апаратні витрати — максимальна тактова частота.

1.3 Алгоритмічні моделі обробки потоків даних

Алгоритм ЦОС є основою специфікації для високорівневого синтезу пристроїв ЦОС. Крім того, інструменти проєктування на системному рівні найчастіше налаштовані на моделювання виконання алгоритмів, метою якого є правильно зафіксувати специфікацію системи, яка проєктується, що допомагає як у її впровадженні, так і для її перевірки та оцінювання параметрів [53].

За загальноприйнятим визначенням, алгоритм — це обчислювальний процес, який має початок, детерміновану послідовність дій і кінець, і який для різних початкових даних дає очікувані результати [7]. Це визначення стосується трансформаційних обчислювальних систем (ОС) без обмежень на швидкодію. У протилежних до них ОС реального часу періоди виконання алгоритмів не перевищують заданих часових обмежень. Такі ОС розділяють на інтерактивні, якщо вони реагують на вхідні дані з притаманною їм швидкістю, та реактивні, які не допускають втрати цих даних через свою недостатню швидкодію [48, 145]. Засоби ЦОС належать до ОС реального часу.

За визначенням Поста й Тюрінга, алгоритм — це обчислювальний процес, який виконується деякою моделлю обчислювача, що сконструйована в межах точних математичних понять [5]. Таке визначення алгоритму передбачає наступне.

1. Має бути певний суб’єкт або процесор, який вміє читати, розпізнавати конструктивні об'єкти (дані та команди) та правильно виконувати операції з ними відповідно до алгоритму.

2. Алгоритм призначений для ефективного обчислення деякої корисної функції для отримання правильного результату для певного ансамблю конструктивних об’єктів за скінченну кількість кроків за умови близькодії (доступ до операндів та операції з ними виконуються за прийнятний час).

3. Несуттєві деталі обчислювального процесу не враховуються [58].

На практиці, використовується набір різних обчислювальних моделей. Вони розрізняються як за сферою застосування, так і за рівнем абстракції.

У функціональному алгоритмі немає обмежень на порядок обчислення функцій-операторів, крім порядку, який задано залежностями за даними. Водночас результати функцій обчислюються тільки тоді, коли для них є відповідні вхідні дані. Модель обчислювача для функціонального алгоритму прийнято представляти графом. У такій моделі вершини графа представляють функції, а дуги — залежності за даними й керуванням. Реалізація алгоритму на такій моделі являє собою передачу даних за напрямком дуг і запуски функцій-операторів вершин відповідно до їхньої інцидентності.

Граф потоків даних (ГПД) — це напрямлений граф, вершини якого — актори — представляють оператори, а дуги — канали передачі даних. Актори споживають дані, що називаються мітками або токенами, і видають мітки на свої виходи. ГПД, запропонований Деннісом, є моделлю, у якій обчислення визначаються наявністю даних на інформаційних і керуючих дугах [1].

У моделі мережі процесів Кана процеси у вершинах взаємодіють через канали у вигляді потоків даних типу FIFO (first in first out) [124]. Граф Карпа й Міллера (граф обчислень, ГПД) також має в дугах FIFO. Але його актори запускаються, якщо кількість міток у вхідному потоці дорівнює заданому числу [4].

Коректність задання алгоритму на моделі обчислювача перевіряють або аналітично (якщо можливо), або за допомогою складання розкладу обчислювального процесу на моделі. Розклад також складають при безпосередньому виконанні алгоритму в певній ОС. Якщо структура ОС відповідає графу моделі, то складання розкладу полягає в обчисленні конкретного моменту виконання кожного оператора. Якщо структура ОС довільна, то додатково виконують етап призначення операторів на процесори цієї ОС.

Розрізняють повністю статичний, частково статичний (self-timed), динамічний розклади, а також розклад зі статичним призначенням [144]. У разі синтезу обчислювачів на базі НВІС, СНК і ПЛІС відповідні САПР дозволяють тільки повністю статичний розклад, тобто, усі етапи складання розкладу проходять до виконання алгоритму в ОС. Це пов'язано з тим, що результівна ОС на рівні логічних схем повинна бути детермінована й не допускати динамічного переналаштування під час виконання алгоритму [65, 122].

Актор або процес, або просто оператор, який позначений вершиною ГПД, виконує деяку функцію під час свого запуску. Якщо актор споживає і виробляє деяку кількість міток, ця кількість стабільна під час виконання алгоритму й може бути визначена під час компіляції, то такий актор є регулярним**.** Якщо ця кількість може змінюватися залежно від даних у потоках, то це динамічний актор.

ГПД, у якому всі актори регулярні, дістав назву графа синхронних потоків даних (ГСПД, synchronous data flow, SDF) [8]. Слово "синхронний" означає те, що алгоритм виконується циклічно, причому кількість спожитих і згенерованих міток кожною вершиною протягом циклу є постійною. У ГСПД до міток у потоках додають теги (індекси змінних), які строго відповідають номерам циклів, що є ознакою синхронності потоків.

Якщо кількість спожитих із кожного входу міток і згенерованих кожною вершиною міток в одному циклі є тією самою, наприклад, дорівнює одиниці, то такий граф називають однорідним ГСПД (homogeneous SDF). Однорідний ГСПД взаємно однозначно відповідає сигнальному графу або обчислювальній схемі, які використовуються в ЦОС [11, 14].

Якщо кількість спожитих на вході та згенерованих міток хоча б однією вершиною є різною, то ГСПД називається неоднорідним ГСПД (багатократним, багатошвидкісним ГСПД, multirate SDF) [2, 12, 32, 143]. Неоднорідний ГСПД складніше аналізувати, ніж однорідний, і він може мати блокування.

Тому його перетворюють в еквівалентний однорідний ГСПД [142]. Ця модель набула поширення й застосовується в пакеті Matlab-Simulink [146].

У параметричному ГСПД актори можуть мати деякі набори функцій, а дуги — відповідні набори розміток, які можна динамічно змінювати незалежно від потоків даних на період не менше одного циклу. Для такого ГСПД можливо скласти квазістатичний розклад [51]. До таких ГСПД також належать булевий ГПД та його узагальнення — цілочисельний ГПД [61], але за їхнього застосування значно ускладнюється аналіз ГСПД на блокування.

У моделі ієрархічного ГПД вершина актора представлена моделлю того чи іншого типу. Якщо модель нижчого рівня не вироджена, то вона представляється автоматом із пам'яттю. Тому моделі класифікують як такі, що мають актори з пам'яттю або без неї.

Параметричний ГСПД часто одержують склеюванням вершин однорідного ГСПД. Тоді нова вершина актора містить кілька вершин початкового графа й тому є ієрархічною. Нова вершина має складну функцію, що покриває функції вершин, які склеєні, і за один цикл алгоритму послідовно обчислює оператори цих вершин. Процедура такого перетворення ГСПД набула назви згортання ГСПД (folding), а одержаний граф — згорнутого ГСПД (folded SDF). Згорнутий ГСПД є ізоморфним графу структури конвеєрного обчислювача, а згортання ГСПД є методом синтезу таких обчислювачів [178].

В останні роки було запропоновано декілька видів моделей ГПД, які мають збільшену експресивну силу. Усі вони пов'язані з моделями скінченних автоматів (FSM, finite-state machines) та моделями, які ґрунтуються на ускладненні поведінки акторів. До них належать метод El Greco [62], який був впроваджений у системі Synopsys System Studio, *charts-діаграми з асинхронними потоками даних [95], система DF* [73], модель ГПД із дозволом вмикання акторів [184], модель ГПД за сценарієм (SADF — scenario-aware data flow) [227] та модель TensorFlow [105].

Отже, ГПД використовується для задання різних алгоритмів обробки потоків даних, включно з ЦОС, і має багато різновидів, створених для спрощення його аналізу чи збільшення виразності. Алгоритм, заданий як ГПД, показує, як одержується потік результатів і не задає точний порядок обчислень. Його відображення в ОС полягає в знаходженні розкладу і відображенні вершин графа в процесорні елементи (ПЕ) ОС. Вибір різновиду ГПД залежить від сфери застосування алгоритму (моделювання або обробка сигналів, характер сигналів) і способу його виконання (апаратний чи програмний) .

З метою використання для системного синтезу ОС для ЦОС найбільш спрощеною та придатною є модель ГСПД. Ієрархічні моделі також можуть застосовуватись, позаяк вони розгортаються до однорівневої моделі ГСПД. Однорідний ГСПД переважає завдяки простоті його аналізу й можливості еквівалентних перетворень у нього інших типів ГСПД [17].

1.4 Методи високорівневого синтезу ОС обробки потоків даних

У цьому підрозділі розглядаються основні відомі й застосовані методи високорівневого синтезу ОС обробки сигналів із метою визначення їхніх особливостей і можливостей удосконалення.

1.4.1 Етапи високорівневого синтезу

Класичний підхід до розв’язання задач високорівневого синтезу структур графічно можна зобразити, як показано на рисунку 1.1 [79, 226]. Початковими даними для високорівневого синтезу є алгоритм обробки потоків даних і критерій оптимізації. Результатами синтезу є опис структури конвеєра обробки даних (datapath) на рівні регістрових передач, а також відповідного керуючого автомата. Структура конвеєра і його компоненти мають відповідати елементному базису з урахуванням затримок елементів, тактової частоти, площі кристала й енергоспоживання.

Основними етапами високорівневого синтезу є аналіз поведінки ОС, що синтезується, вибір способу проєктування, планування обчислень, синтез керуючого автомата і з'єднання модулів в ОС. На етапі опису алгоритму вхідна модель та обмеження проєкту перетворюються у внутрішнє представлення. Для представлення цієї моделі, як правило, використовують графічні структури [147, 225]. Причому найпоширенішою є модель графів потоків керування й даних (CDFG), яка складається з графів потоків керування (CFG, ГПК) та потоків даних (DFG, ГПД), і яка фактично стала стандартом [83].

Рисунок 1.1 — Схема класичного підходу до вирішення задач високорівневого синтезу структур

На етапі поведінкового перетворення використовуються різноманітні методи оптимізації, такі як усунення неробочого коду, оптимізація виразів,

діапазонів розрядності операндів, заміна ділення на множення на обернену величину, множення на константу — на додавання тощо.

На етапі визначення множини ресурсів оцінюються мінімальні ресурси та обмеження латентної затримки. Часто виконується відображення один до одного і множина ресурсів збігається зі множиною операторів у ГПД. Але в решті випадків виникає складна оптимізаційна проблема, коли кілька операторів можуть виконуватися на одному ресурсі, яка є NP-повною [228]. Причому складності додає те, що операції можуть бути спорідненими (додавання, віднімання), а розрядність операндів — різною. Також слід враховувати різну тривалість виконання операцій. До того ж, точне рішення часто неможливе, бо завантаженість ресурсів можна визначити лише на етапі складання розкладу.

До складання розкладу висуваються вимоги масштабованості, точності оцінювання часу виконання, керованості ходом синтезу [167]. Масштабованість означає те, що складання розкладу має виконуватися за прийнятний час для ГПД будь-якого розміру, включно з такими, кількість вершин яких більша за 10 тисяч. Точність оцінювання часу виконання означає те, що алгоритм має виконувати ефективну оптимізацію критичного шляху. Тобто, для складання розкладу для ПЛІС слід ретельно враховувати затримки мультиплексорів, зв’язків між елементами, які варіюються в широких межах залежно від складності схеми, рівня конвеєризації, ступеня близькодії груп елементів, інтелектуальних властивостей програми розміщення та трасування та ін. Причому частка затримки в лініях зв’язку непередбачувано змінюється від 10% до 70% від загальної затримки критичного шляху.

Під час призначення на регістри їхня кількість мінімізується. Дві змінні можуть мати спільний регістр, якщо вони належать до взаємовиключних гілок обчислень або їхні тривалості існування (living periods) не перетинаються. При цьому для мінімізації регістрів часто використовується алгоритм лівого краю [135]. Таке розділене використання регістрів, як правило, потребує приєднан-

ня до їхніх входів додаткових мультиплексорів та відповідної корекції критичного шляху. Крім того, мінімальна кількість регістрів не є ефективною, бо призводить до надлишкової складності мультиплексорів на їхніх входах та підвищення споживаної потужності від нераціональних пересилок даних між регістрами [161].

На етапі побудови схемних зв'язків та керуючих підсистем слід дотримуватися раніше вибраних ресурсів та складеного розкладу. При цьому часто необхідне втручання розробника. Тому інструмент САПР має забезпечити гнучкість для визначення особливостей розробки для досягнення оптимальних результатів.

На відміну від компіляторів програм, компілятори високорівневого синтезу дають величезні розміри простору для оптимізації. З одного боку, це відкриває можливості для досягнення кращої реалізації, а з іншого — це є причиною попадання ходу оптимізації в один із численних локальних мінімумів. Тому розробник повинен мати важелі для керування ходом синтезу. Для цього розробник має бути спроможним накладати певні обмеження на складання розкладу для критичних частин ГПД.

Щоби допомогти розробникам зробити кращий вибір щодо мікроархітектури ОС, САПР має забезпечити набір можливостей: наприклад, допомогти вибрати віртуальний модуль із бібліотеки, організувати розділену буферну пам'ять. Однак простір рішень вибору мікроархітектури дуже великий, і рішення дуже взаємозалежні. Тому, навіть за дотримання значень за замовчуванням, уточнення мікроархітектури вимагає взаємодії людини із САПР, і це є важливою частиною процесу розроблення проєкту. У цьому полягає керованість ходом синтезу. Саме тому проєктування за допомогою високорівневого синтезу має давати змогу оцінювати різні архітектури за лічені хвилини чи години, що неможливо за умов розробки лише на рівні RTL [43].

1.4.2 Методи високорівневого синтезу

Синтез ОС означає формування множини альтернативних структурних рішень і вибір найбільш переважного рішення як оптимального за заданим критерієм оптимальності. Під час цього відбирається таке рішення, яке задовольняє низку обмежень, таких, як тривалість тактового інтервалу, площа кристалу, границя енергоспоживання, і яке мінімізує функцію вартості. Така оптимізація виконується на кожному з етапів проєктування ОС.

Складність процесу синтезу полягає в тому, що різні аспекти й етапи розроблення ОС суттєво залежать один від одного. Кожен з етапів проєктування (рис. 1.1) не може бути виконаний незалежно, щоби не зменшити можливості глобальної оптимізації ОС. Також складність синтезу ґрунтується на великій різниці між початковою специфікацією й описом результівної ОС. Нарешті, складність синтезу виражається в паралелізмі, притаманному сучасним ОС для ЦОС. Через це переважна більшість задач синтезу є NP-повною [93, 230].

Складність синтезу зменшують шляхом декомпозиції. Виконують декомпозицію складного проєкту в просторі, тобто до множини проєктних модулів, які синтезують окремо, або в часі, коли розділяють процес синтезу на окремі незалежні, простіші стадії.

При часовій декомпозиції виділяють еволюційний підхід, а також покрокове конструювання. Якщо специфікація, що перетворюється, представляє весь проєкт, то це еволюційний підхід (refinement, reformulation approach). Якщо на кожному кроці додаються частини проєкту, які залишаються потім незмінними, то це підхід покрокового конструювання (incremental approach). Водночас перетворювання не змінюють рівень абстракції опису проєкту, а призводять до його уточнення й наближення до результату [140, 168, 221].

1.4.3 Методи пошуку структурних рішень

Вибір методу пошуку рішення — це друга проблема після формалізації завдання на проєктування. Якщо при формалізації всі керовані параметри вдалося представити в числовому вигляді, то часто застосовують методи дискретного математичного програмування (ДМП). Можливості розв’язання задач ДМП вивчаються в теорії складності завдань вибору, де показано, що завдання, що належать до класу NP-повних задач, у загальному випадку вдається вирішувати, лише якщо вони обмеженого розміру. Для синтезу структур на основі ГСПД — обмеження оцінюється як 15–30 вершин. Тому більшість практичних завдань структурного синтезу вирішують за допомогою наближених (евристичних) методів.

Основу великої групи математичних методів ДМП, що виражають прагнення до скорочення перебору варіантів, складають операції поділу множини варіантів на підмножини і відсікання неперспективних підмножин. Ці методи об'єднуються під назвою методу гілок і меж.

Серед інших наближених методів розв’язання задачі ДМП виділяється метод локальної оптимізації. Позаяк для оптимізації простір рішень має метрику, то використовують поняття околу поточної точки пошуку. Замість перебору точок у всьому просторі рішень перебираються точки тільки в околі, де очікується рішення. Недоліком методу є його «застрявання» в околах локальних екстремумів.

Для підвищення ефективності пошуку при локальній оптимізації застосовують метод оптимізації із заборонами (tabu search). Для цього в критерій оптимізації вводять заборони на потрапляння в неефективні точки.

Експертна система є типовою системою штучного інтелекту, у якій база знань містить відомості, отримані від людей-експертів у конкретній предметній області. Труднощі формалізації процедур структурного синтезу призвели до популярності застосування експертних систем у САПР, оскільки

в них здійснюється синтез на основі досвіду й неформальних рекомендацій [10]. Синтез схем нерекурсивних фільтрів, процесорів ШПФ у сучасних САПР вважаються прикладами роботи експертних систем.

Еволюційні методи (ЕМ) ґрунтуються на статистичному підході до дослідження простору рішень та ітераційному наближенні до шуканого рішення. На відміну від точних методів ДМП, ЕМ дають змогу знаходити рішення для складних задач, які є близькими до оптимальних, за прийнятний час, а на відміну від експертних систем та евристичних методів оптимізації характеризуються істотно меншою залежністю від особливостей предметної галузі й у більшості випадків забезпечують кращий ступінь наближення до оптимального рішення.

Найважливішим окремим випадком ЕМ є генетичні методи й алгоритми. Генетичні алгоритми (ГА) ґрунтуються на пошуку кращих рішень за допомогою спадкування й посилення корисних властивостей множини об'єктів — моделей рішень у процесі імітації їхньої еволюції. Властивості об'єктів представлені значеннями параметрів, що об'єднуються в запис, який називається хромосомою.

Серед ЕМ знаходять застосування також методи, які, на відміну від ГА, оперують не множиною хромосом, а лише єдиною хромосомою. Так, метод дискретного локального пошуку (hill climbing) ґрунтується на випадковій зміні параметрів (тобто, генів у хромосомі). Такі зміни називають мутаціями. Після чергової мутації оцінюють значення функції корисності F (fitness function) і результат мутації зберігається в хромосомі, тільки якщо F покращилася. В іншому ЕМ під назвою модельоване відпалювання (Simulated Annealing) результат мутації зберігається з певною ймовірністю, що залежить від значення F та від тривалості оптимізації, яка вважається температурою [235].

Отже, задачу структурного синтезу системи ЦОС слід вирішувати за допомогою ЕМ через те, що її розмірність може бути практично необмеженою.

Серед великої множини цих методів генетичні алгоритми є перспективними, бо вони здатні знаходити абсолютний оптимум серед множини локальних [133].

1.4.4 Методи планування обчислень

1.4.4.1 Базові поняття та визначення

У більшості методів задача планування декомпонується на етапи визначення множини ресурсів (allocation), складання розкладу (scheduling) і призначення на ресурси (assignment) (див. рис. 1.1). Серед них етап складання розкладу є найскладнішим і найвідповідальнішим.

Задача планування — оптимальне розподілення множини операцій на обмеженому наборі ресурсів.

Мета планування — мінімізація конкретної об'єктивної характеристики процесу обробки даних, яка задана відповідно до початкової задачі: загального часу обчислення, вартості ресурсів тощо.

Під час високорівневого синтезу ОС має значення планування в умовах порядку передування операцій. Ці умови виражаються графом залежності за даними (directed acyclic graph, DAG) або ГПД. У загальному випадку, ГПД є графом циклічного алгоритму. А більшість алгоритмів планування працюють з ациклічним DAG. Це не є суттєвим обмеженням, позаяк при інтерпретації ГПД часто будується ациклічний граф залежностей, який дорівнює DAG.

Ациклічний граф залежностей за даними формально представляється як $DAG(V\,E)$, де V — множина операцій (без урахування часу виконання), E — відношення попередності та зв'язки по даних.

Істотними для планування є типи операцій (f_q — операція типу q) і типи обчислювальних блоків або ПЕ (p_k — ПЕ типу k). P позначає множину усіх ПЕ, P_k — множину ПЕ типу k. ПЕ можуть бути трьох типів:

— універсальні — ПЕ виконує будь-яку операцію з наявних у задачі;

— різнорідні, коли кожний ПЕ виконує свої функції;

— коли ПЕ різних типів виконують частково однакові операції, а частково різні.

У випадку, коли операції виконуються за різні інтервали часу, для спрощення планування тривалість виконання операцій можна прийняти як тривалість найдовшої операції. У межах переліченого вище ставиться спрощена задача складання розкладу.

Нехай задано:

— ациклічний граф $DAG(V, E)$;

— F — множина операцій $f_i \in F$ тривалістю в один такт;

— K — кількість окремих типів функціональних блоків;

— $|P_k|$ — кількість функціональних блоків типу k;

— усі обчислювальні блоки різнорідного типу;

— загальний час виконання алгоритму T в тактах.

Необхідно знайти такий план $\sigma : F \to \{0, 1, ..., T\}$ (відображення σ операцій з F у часові відліки $t \in T$), за якого виконувалось би наступне:

$$|\{f_q \in F \wedge f_q \in p_k : \sigma(f_q) = t\}| \le |P_k| \; \forall \; t \in \{0, 1, ..., T\}, k \in \{1, ..., K\},$$

та

$$\forall \; (f_i, f_j) \in E \Rightarrow \sigma(f_i) < \sigma(f_j).$$

За умови необмежених ресурсів мінімальним часом виконання задачі є час, який відповідає критичному шляху в DAG. Водночас проміжні вершини графу мають свободу переміщуватись уздовж осі часу. Тому для кожної проміжної вершини істотні два параметри: ранній час виконання $\sigma_{ASAP}(f_i)$ (as soon as possible, ASAP) та пізній час виконання $\sigma_{ALAP}(f_i)$ (as long as possible, ALAP).

Задача планування на формальному рівні набуває двох форм:

— задача планування з обмеженням на ресурси;

— задача планування з обмеженням за часом.

Обидві задачі описуються в термінах цілочисельного лінійного програмування (ЦЛП), яке є точним методом ДМП. Задача ЦЛП формулюється так: є деяка функція вартості $c(x)$ та система лінійних рівнянь виду $A*\boldsymbol{x} = \boldsymbol{b}$ (a_{ij}, b_j — цілі). Необхідно знайти таке ціле невід'ємне значення параметрів вектора $\boldsymbol{x}$, що задовольняє всі обмежувальні умови, за якого значення функції вартості було б мінімальним.

Задача планування з обмеженням на ресурси виглядає таким чином [148]:

$$\min T, \tag{1.2}$$

$$\Sigma\, x_{i,t} - |P_k| \le 0, \text{ для } 1 \le t \le T,\ 1 \le k \le K \quad f_i \in p_k, \tag{1.3}$$

$$\Sigma\, x_{i,t} = 1 \text{ для } 1 \le i \le |V| \quad t \in [\sigma_{ASAP}(f_i),\ \sigma_{ALAP}(f_i)], \tag{1.4}$$

$$\Sigma\, (t \cdot x_{i,t}) - \sigma(f_i) = 0,\ \forall f_i \quad t \in [\sigma_{ASAP}(f_i),\ \sigma_{ALAP}(f_i)], \tag{1.5}$$

$$\sigma(f_i) - \sigma(f_j) \le -1,\ \forall (f_i, f_i) \in E, \tag{1.6}$$

$$\sigma(f_i) - T \le 0,\ \forall f_i \in V_{end}. \tag{1.7}$$

Задана множина P ПЕ, яка складається з $|K|$ підмножин $\{p_k\}$ ПЕ різних типів. У кінцевому підсумку необхідно мінімізувати тривалість плану T (1.2). Факт того, чи була операція f_i розподілена на пристрій на кроці виконання t, визначається значенням змінної $x_{i,t} \in (0,1)$. Формула (1.3) задає обмеження на ресурси. Формула (1.4) задає умову обов’язковості призначення операції на будь-який ресурс. Момент часу призначення $\sigma(f_i)$ операції f_i визначається формулою (1.5). Умова дотримання попередності задається формулою (1.6). А формула (1.7) перевіряє, що тривалість плану дійсно вміщується в значення T.

Задача планування з обмеженням на кінцевий час виглядає так:

$$\min \Sigma c_k \cdot |p_k| \ (k = 1 \ldots K), \tag{1.8}$$

$$\Sigma x_{i,t} - |p_k| \le 0, \text{ при } f_i \in p_k \text{ і } 1 \le t \le T, \ 1 \le k \le |K| \tag{1.9}$$

$$\begin{aligned} &\Sigma x_{i,t} = 1 \text{ для } 1 \le i \le |V| \\ &t \in [\sigma_{ASAP}(f_i), \ \sigma_{ALAP}(f_i)], \end{aligned} \tag{1.10}$$

$$\begin{aligned} &\Sigma (t \cdot x_{i,t}) - \Sigma (t \cdot x_{j,t}) \le -1 \quad \forall (f_i, f_i) \in E \\ &t \in [\sigma_{ASAP}(f_i), \ \sigma_{ALAP}(f_i)]. \end{aligned} \tag{1.11}$$

Основною метою цієї задачі є оптимальним чином завантажити ресурси роботою. Функція (1.8), що мінімізується, це функція вартості ресурсів, причому c_k є оцінкою вартості ресурсу p_k.

1.4.4.2 Методи планування з обмеженням на ресурси

Далі розглядаються відомі методи, які є наближеними методами ДМП. Найбільш відомим підходом для планування з обмеженням на ресурси є складання розкладу за списком (list scheduling). Згідно з ним, усі операції сортуються за пріоритетами, заносяться в список і далі призначаються на пристрої в міру готовності до виконання й за пріоритетом [6]. Якість планування залежить від вибору функції пріоритету. Основною залежністю, яка впливає на пріоритет розміщення операції, є функція мобільності, яка дорівнює:

$$M(f) = \sigma_{ALAP}(f) - \sigma_{ASAP}(f). \tag{1.12}$$

Отже, для операцій, розташованих на критичному шляху, $M(f) = 0$ і чим вище значення $M(f)$, тим менший пріоритет у призначенні вершини f на пристрій саме в цей момент часу [176].

На відміну від функції мобільності, яка залишається «локальною» щодо конкретної операції, функція CADDY дає змогу оцінити всю множину вершин перед розподілом, що дає більш ефективний розклад [104].

Складання розкладу спрямованим зусиллям (force-directed scheduling) є одним із найбільш поширених методів планування при точному значенні часового обмеження T_{exact} [181]. Відповідність між вершинами й часовими кроками встановлюється на основі функції сила (force). Виконується ASAP і ALAP-планування з урахуванням часового обмеження $T_{exact} \geq T_{ASAP}$. Розраховується середнє значення $M(f)$(1.12) для всіх часових кроків. Множина величин $M(f)$ називається графом розподілення (distribution graph). Основна ідея планування є балансування графу розподілення в такий спосіб, щоби його ширина, яка відповідає множині ресурсів, була мінімальною. При цьому чим менше значення сили в операції в цей момент, тим більш ефективне розподілення вийде при її призначенні операції на ресурс саме в цей момент.

1.4.4.3 Планування обчислень у конвеєрних ОС

У сучасних ОС використовуються конвеєризовані ПЕ. Основні поняття, пов’язані з конвеєризацією, наступні.

Час обчислення T — це час, необхідний для обчислення даної функції для однієї групи аргументів в обчислювальному пристрої без урахування витрат, пов’язаних із конвеєризацією.

Затримка конвеєра L — це мінімально можливий час між двома послідовними завантаженнями в конвеєр груп даних для обробки.

Ступінь конвеєра — апаратно-структурна одиниця, яка забезпечує незалежне виконання однієї стадії алгоритму обчислення заданої функції.

У теорії конвеєрних ОС найчастіше розглядаються такі структури, у яких у будь-який момент часу кожен ступінь конвеєра над однією групою даних виконує одну визначену операцію, яка не змінюється, тобто, ступені конвеєра мають незмінну функціональність. Але логіка багатьох алгоритмів

припускає зміну конфігурації апаратних ресурсів та їх виконуваних функцій від кроку до кроку. Такий вид конвеєризації називається функціональною конвеєризацією. Планування при функціональній конвеєризації полягає у відображенні саме ГСПД у множину апаратних ресурсів і множину часових інтервалів. У простому випадку задача зводиться до відображення DAG, що відповідає ациклічному підграфу однорідного ГСПД. Але в такому разі не береться до уваги перекриття операцій у часі на початку й наприкінці виконання DAG.

У роботах [121, 179] запропоновано методи планування для конвеєрних ОС, у яких після спискового складання розкладу операції призначаються на конвеєрні ресурси, з огляду на циклічний характер обчислень. Також використовують метод відображення гнізда циклів у неконвеєрну ОС із подальшим перетворенням її в конвеєрну ОС [67]. У [232] вдосконалено метод силового планування для розроблення конвеєрних ОС.

У роботах [65, 138] запропоновано метод складання розкладу на основі комбінування розкладів шляхів (path-based scheduling), згідно з яким фінальний розклад для жмутка гілок є перекриттям розкладів для гілок, які входять у цей жмуток.

1.4.4.4 Задача призначення ресурсів

Підходи, що використовуються при призначенні на ресурси (assignment) використовують різноманітні ітеративні алгоритми, метод гілок та меж, а також методи лінійного програмування. Усі вони подібні в тому, що вони виконуються в покроковому режимі, коли розклад алгоритму вже складений.

Методи призначення ресурсів набули розвитку при розв’язанні задачі призначення на регістри. Якщо алгоритм представлений DAG, припускається, що значення змінної *r* має бути збережене в регістрі, якщо в даному часовому кроці була виявлена дуга графу, яка відповідає *r*. Своєю чергою, зі значенням *r*, створеним як результат операції *f*, пов’язано множину дуг графу, які об’єд-

нують вершину операції f з її наступниками $succ(f)$. Тривалість існування r визначається парою моментів часу виникнення змінної $\sigma(f)+\Delta(f) = \sigma(f)+1$ та її зникнення $\max(\sigma(succ(f))$. Задача призначення регістрів передбачає визначеня мінімальної кількості регістрів і мінімізує кількість пересилок між ними.

Евристика Тсенга полягає в побудові й декомпозиції графа з кліками, кожна з яких відповідає множині змінних, які можуть бути призначені на один регістр і проходить за принципом покрокового конструювання [228].

У графі конфліктів вершини v і w, які відповідають змінним, з'єднуються ребром $e = (v, w) \in E$ тоді, коли вони не сумісні в часі. Задача розподілення регістрів зводиться до мінімального розфарбування ребер такого графу, де кількість кольорів дорівнює кількості регістрів [212].

Граф $G = (V, E)$ називається інтервальним графом, якщо ребра E відповідають лінійній множині інтервалів I, причому дві вершини графу з'єднуються ребром, коли відповідні цим вершинам інтервали зі множини I перетинаються. Цей граф використовує метод лівої границі (left-edge scheduling), який є найбільш ефективним з алгоритмів призначення регістрів. Його суть полягає в тому, щоби так сортувати інтервали в таблиці зайнятості регістрів, щоби колонки лівого краю таблиці виявилися б найбільш заповненими [135].

Недолік методу лівого краю полягає в тому, що він не забезпечує пошук оптимального рішення для циклічного алгоритму, і його результат не є ефективним за міжрегістровими пересилками. Але цей метод можна пристосувати для мінімізації ресурсів ПЕ, включно з конвеєрними ПЕ [3].

Ітераційний характер ГСПД можна описати інтервальним графом із циклічними дугами (circular-arc graph) [229]. Одне зі спрощень планування полягає в тому, що змінна циклу розбивається на дві змінні, які призначаються на різні регістри, між якими організують пересилки [218]. Інший метод

полягає в призначенні першими змінних, які поводяться циклічно, а решти змінних — за допомогою покрокового конструювання [113].

1.4.4.5 Методи одночасного складання розкладу і призначення на ресурси

Найпоширенішим методом одночасного планування і призначення на ресурси є одиничне відображення ГСПД у структуру ОС. У такому разі, як правило, використовується метод ресинхронізації (retiming) [150], який полягає в перепризначенні глибини буферів у дугах ГСПД. У роботі [186] ГСПД оптимізується за допомогою ресинхронізації з використанням характеристики мобільності операцій і ймовірності розділення ресурсів.

У роботі [16] запропоновано метод просторового ГСПД, який полягає в представленні ГСПД у багатовимірному просторі, його оптимізації та відображенні в підпростір структур та підпростір розкладів. Основними перевагами методу є виконання оптимізації як розкладу, так і вибору та призначенню на ресурси з прийняттям до уваги періодичності алгоритму та функціонально конвеєрної реалізації синтезованого пристрою. Причому метод дає змогу синтезувати ОС, яка має заданий рівень паралелізму, тобто, він поєднує в собі такі методи, як одиничне відображення, ресинхронізація, алгоритм лівого краю, згортання ГСПД та інші [207].

Отже, сучасні методи відображення алгоритмів характеризуються наступним.

Наявні методи відображення алгоритмів в обчислювальне середовище мають етапи вибору множини ПЕ, складання розкладу роботи ПЕ, призначення операторів на ПЕ та змінних на регістри, вибору множини шин і призначення міжпроцесорних обмінів на ці шини. Загальним для цих методів є те, що всі ці етапи виконуються цілком окремо один від одного, хоча в різних

методах — у різному порядку. Водночас кожен з етапів часто має протилежну мету, і це ускладнює оптимізацію відображення.

Більшість із цих методів належить до множини методів покрокового конструювання за допомогою певної стратегії, яка забезпечує кінцеве рішення досить наближене до оптимального. Слушно, що такі стратегії орієнтовані на відображення деякого обмеженого класу алгоритмів у спеціалізовану структуру ОС. Здебільшого такою структурою є набір ПЕ, блоків пам'яті, з'єднаних за допомогою загальних шин або мультиплексорів. Отож, ці методи в багатьох випадках не можуть дати прийнятне рішення.

Метод просторового ГСПД має безумовні переваги над іншими методами планування для конвеєрних ОС і тому його слід узяти за основу для подальших досліджень.

1.4.5 Засоби високорівневого синтезу

До засобів проєктування ОС належать мови опису алгоритмів і апаратури, засоби моделювання і відображення алгоритмів у структури ОС.

Алгоритмічна мова має, з одного боку, бути виразною, з іншого боку, точно відображати відповідну множину алгоритмів і підлягати компіляції та трансформації в інші засоби задання алгоритму, у тому числі в апаратні засоби ОС. Така мова має також ефективно виражати апаратні особливості ОС, бути зручною, набути широкого використання і статусу стандарту.

Засіб моделювання алгоритмів є невід'ємною частиною САПР ОС і слугує для верифікації алгоритмів, описаних відповідною мовою, шляхом їх виконання в часі на визначеній моделі обчислювача. Ефективний засіб моделювання забезпечує дослідження алгоритму в інтерактивному режимі, яке ґрунтується на швидкій компіляції програми, високопродуктивному моделюванні алгоритму та виразному відображенні результатів моделювання.

Засоби відображення алгоритмів виконують перетворення алгоритму, заданого мовою опису, у проєкт ОС, що виконує цей алгоритм, користуючись тими чи іншими методами відображення й низкою критеріїв оптимізації.

1.4.5.1 Мови високорівневого синтезу

Мови Lustre [105] і Signal [141] — мови текстового задання ГПД. Програма на Lustre має більшу виразність, ніж на Signal, завдяки можливості програмування булевих, цілочисельних та інших потоків даних.

Мова програмування ГСПД Esterel [49] забезпечує імперативний стиль моделювання ГСПД. На її основі фірмою Seawright створено інструмент синтезу під назвою Protocol Compiler [200].

У мові SpecC [92] було впроваджено моделювання рівня транзакцій (transaction-level modeling, TLM). Рівень TLM є більш високим у порівнянні з RTL і реалізується завдяки функції каналу, яка абстрагує деталі обміну даними та синхронізації при захисті від блокування. Тепер принцип TLM також застосовано в мові SystemVerilog [220]. Також він залучений у мові SystemC.

Мова CAL (CAL Actor Language) розроблена у 2001 р. як частина симулятора Ptolemy II в університеті Берклі. Вона застосовується для програмування ГПД при синтезі апаратного та програмного забезпечення ЦОС. У мові CAL актори ГПД мають специфікацію функцій, їхніх пріоритетів, керування запуском, схеми портів. CAL є гнучкою мовою, але відсутність блокувань перевіряється лише моделюванням [85]. На основі CAL були розроблені засоби OpenDF [52, 59], RVC-CAL [239] та ін. для моделювання ГПД та відображення його в паралельні програми чи структури.

Мови Verilog і VHDL є поширеними мовами проєктування ОС на рівні регістрових передач. Verilog і VHDL використовують модель паралельних процесів, які обмінюються сигналами [13]. На відміну від Verilog, мова VHDL — строго типізована, підтримує типи користувача.

Мова VHDL бралася за основу в багатьох дослідницьких проєктах високорівневого синтезу [82]. У роботах [65, 151] показано, що системний опис на VHDL важко транслюється до регістрового рівня через те, що в цю мову закладено низький рівень синхронізації й пересилок даних. У теперішній час стиль системного опису на VHDL містить опис алгоритму функціонування ОС і протокол вводу-виводу даних [196].

Оскільки Verilog є найбільш поширеною мовою для опису мікросхем у промисловості, виконано низку спроб її модернізації. SystemVerilog найбільш вдала спроба модернізації Verilog, що набуває поширення [34].

Мова SystemC була розроблена для проєктування мікросхем і прикладного матзабезпечення, а також для стандартизації опису модулів комітетом Open SystemC Initiative (OSCI) [94]. Обчислювальною моделлю SystemC є така сама модель, що у VHDL і Verilog, а конструкції SystemC мають взаємно однозначні еквіваленти в мовах VHDL і Verilog [56]. Ця мова використовує мову C++ для створення середовища моделювання. SystemC підтримує моделювання рівня RTL і більш високих рівнів, таких як модель TLM. За таких умов СНК описується як мережа ПЕ, які комунікують між собою.

SysteMoc — це бібліотека на основі SystemC для розроблення потокових програм та синтезу систем обробки сигналів. SysteMoc ґрунтується на моделі ГПД і була перевірена при програмуванні ЦОС для ПЛІС [111].

Система проєктування Signal Processing Workstation (SPW) і мова SPW використовують модель обчислення, яка схожа на ГСПД, але синхронізація запусків виконується за сигналом утримування [44].

1.4.5.2 Графічне програмування ГПД

Мова Statecharts (діаграми станів) [108] призначена для опису ієрархічних скінченних автоматів за допомогою графічного синтаксису. Вона використовує таку саму модель, як Esterel. Остаточним результатом компіляції

є програма мовою С або опис на рівні регістрових передач. Мова Statecharts була прийнята як частина стандарту UML (unified modeling language) [193].

Проєкти Ptolemy та Ptolemy II в університеті Берклі справили значний вплив на загальну тенденцію до перегляду розроблення вбудованих систем із погляду моделей обчислення. Це інструмент графічного вводу ГПД, який, крім моделювання, забезпечує інфраструктуру програмного забезпечення для експериментів із новими методами високорівневого синтезу. Він дав змогу використовувати ієрархічні моделі на скінченних автоматах та ГСПД [84]. В урізаному вигляді став частиною системи Studio фірми Synopsys [103].

Програма Simulink фірми MathWorks використовується в складі системи Matlab. У ній застосована модель неоднорідного ГСПД. Алгоритм задається графічно, причому для опису вершин використовуються такі мови, як Fortran, Ada [146]. Аналогічні властивості має система Scade фірми Esterel [50].

1.4.5.3 Високорівневі компілятори

У роботі [66] зроблено огляд понад двадцяти експериментальних і комерційних засобів високорівневого синтезу, які були розроблені до 2007 р. З-поміж них найбільшу частину складають компілятори, які орієнтовані на реалізацію ОС у ПЛІС і декілька компіляторів для ОС зі спеціальною архітектурою. У більшості з них опис алгоритму виконується мовою програмування такою як С. При цьому паралелізм алгоритму має виражатися в циклах або за допомогою прагм задання паралельних гілок, а також у поданні даних масивами чи векторами. У меншості засобів алгоритм задається спеціалізованою мовою, такою як Stream-C, Handel-C, де паралелізм задається явно на моделі ГПД. Результатом майже всіх компіляторів є RTL-опис ОС, який необхідно обробляти засобами САПР для тієї чи іншої технології ПЛІС чи НВІС.

За останні півтора десятиріччя більше десятка експериментальних високорівневих компіляторів дістали роль комерційних. З-поміж них найбільшого поширення набули наступні [107].

OpenCL Compiler (AOC) фірми Intel використовує мову програмування OpenCL. Опис OpenCL ґрунтується на мові C і фактично є множиною викликів функцій з однойменної стандартної бібліотеки. Для кожної функції заздалегідь розроблено відповідний генератор RTL-описів. Тому за таким викликом автоматично генерується опис конвеєрної схеми з заданим рівнем паралелізму. Тобто, AOC є експертною системою проєктування.

Компілятор SDAccel фірми Xilinx призначений для того, щоб інженери програмного забезпечення розробляли та оптимізували застосунки, розроблені мовами OpenCL, C та C++ для платформ на ПЛІС, наприклад, як прискорювачі в центрах обробки даних.

Компілятор Vivado HLS від Xilinx — це допрацьований інструмент HLS AutoPilot, розроблений в університеті UCLA. Алгоритм задається мовами C, C++ та SystemC. На виході одержується опис модулів ПЛІС на рівні RTL мовами VHDL, Verilog. Можливе підключення деяких функцій бібліотеки ЦОС OpenCV, а також IP-ядер Xilinx. Користувач мусить явно задати низку обмежень конкретної платформи, тактової частоти, призначення інтерфейсу, способів обробки циклів та ресурсів для певних інструкцій. Паралелізм алгоритму, зокрема конвеєрна обробка, задається в операторі циклу як ідіома [198].

Компілятор Catapult фірми Calypto транслює опис рівнів TLM і ESL мовою C++ чи SystemC в опис RTL мовою VHDL, Verilog або SystemC. Оптимізація та розпаралелювання мають здійснюватися, забезпечуючи велику низку обмежень під час синтезу й, отже, вимагати глибоких знань особливостей апаратури для досягнення гідної якості результату синтезу.

Компілятор C-to-Silicon від Cadence використовує вхідні мови C, C++, SystemC та OSCI TLM і генерує код на Verilog. У його бібліотеці є кілька генераторів модулів різних функцій. Керування відображенням виконується за допомогою скриптів Tcl.

Фреймворк Cynthesizer від Cadence є аналогом компілятора Catapult, але є більш багатофункціональним і охоплює моделювання та інтеграцію готових модулів. Але методи оптимізації, такі як розгортання циклів або конвеєризація, мають бути визначені вручну спеціальними директивами [158].

HDL Coder — це генератор коду компанії MathWorks, який генерує опис мовою VHDL або Verilog на рівні RTL з функцій Matlab та моделей Simulink для ПЛІС. Такий генератор генерує специфічний код на основі модулів Xilinx LogiCORE або Altera DSP Builder. При цьому кожній вбудованій функції відповідає своя програма-генератор модулів.

Компілятор SPIRAL, який розроблений в університеті Карнегі — Меллона, призначений для генерації модулів фільтрів зі скінченною імпульсною характеристикою (CIX) та з нескінченною імпульсною характеристикою (HIX), процесорів дискретних ортогональних перетворень та ін. Алгоритм задається мовою обробки сигналів SPL (search processing language). Під час синтезу алгоритм представляється у матричному вигляді, й використовуються еволюційні методи факторизації матриці, динамічне програмування та методи штучного інтелекту типу експертної системи з навчанням [91]. Недоліком розглянутого підходу є обмежена множина алгоритмів, які представляються в матричному вигляді.

Аналіз сучасних систем для високорівневого синтезу показує наступне.

1. Наявні системи для високорівневого синтезу відрізняються тим, що кожна орієнтована на деякий клас алгоритмів: алгоритми ЦОС, зображень, керування та інші. Обмеженням також є цільова архітектура: конвеєрна схема, мікропроцесорна система, спеціалізований пристрій. Іншим обмеженням є рівень розпаралелювання ОС: повне розпаралелювання або розпаралелювання в декілька разів, розпаралелювання з побітною обробкою.

2. Переважна більшість систем орієнтовані на вхідні мови C, C++, SystemC, а на виході мають опис на рівні RTL мовами VHDL або Verilog. Вони

вимагають явного задання паралелізму, найчастіше, у вигляді операторів циклу, а також прагм і обмежень, які безпосередньо не притаманні вказаним мовам. Високоефективний синтез вимагає від користувача знань особливостей апаратної реалізації алгоритму.

3. Тенденцією є синтез як об'єднання кількох модулів, які беруться з бібліотек, таких як OpenCL, OpenCV, LogiCORE, DSP Builder або генеруються спеціалізованими програмами чи компіляторами.

4. Багато систем є інтерактивними системами для напівручного відображення алгоритму в ОС із графічним інтерфейсом і дають ефективні рішення. Причому інструментальною мовою цих систем найчастіше є Java.

5. Деякі види модулів, наприклад, цифрові фільтри, синтезуються генераторами модулів, які побудовані за принципом експертної системи.

1.5 Висновки до першого розділу

1. Для розроблення проєктів на ПЛІС необхідні нові методи й засоби відображення алгоритмів в обчислювальне середовище, які завдяки регулюванню міри розпаралелювання алгоритму забезпечують автоматизований синтез низки оптимізованих пристроїв із різною пропускною спроможністю.

2. Вимоги до автоматизованих засобів синтезу систем ЦОС такі: опис алгоритму має явним чином задавати паралелізм обчислень, відображення алгоритму повинне мати масштабування рівня його розпаралелювання, синтез повинен бути масштабованим відносно складності алгоритму й бути керованим за пропускною спроможністю, апаратними витратами, енергоспоживанням системи, особливостями елементної бази.

3. Представлення алгоритму у вигляді просторового ГСПД забезпечує опис алгоритму з явним заданням паралелізму обчислень, масштабоване відображення алгоритму, а також більш ефективну оптимізацію синтезу. Крім того, ГСПД може служити основою графічного програмування ОС.

4. Найкращою стратегією оптимізації в засобах відображення алгоритмів в обчислювальне середовище є розроблення конвеєризованого пристрою, який забезпечує задану продуктивність і має максимальне відношення тактова частота — апаратні витрати. Перспективними є методи еволюційної оптимізації під час синтезу систем ЦОС, зокрема, генетичні алгоритми. На відміну від ДМП, такі алгоритми дають змогу проєктувати структури великих розмірів з високими показниками якості.

На основі цих висновків у наступному розділі розглядається розроблення методу проєктування паралельних ОС для ЦОС.

РОЗДІЛ 2. МЕТОД ПРОЄКТУВАННЯ КОНВЕЄРНИХ ПРИСТРОЇВ ДЛЯ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ

У першому розділі були розглянуті наявні методи та засоби високорівневого синтезу систем для СНК, включно із системами ЦОС, і зроблені висновки про наступне.

Потрібні нові засоби високорівневого синтезу, які забезпечують керування процесом оптимізації, зокрема масштабуванням рівня розпаралелювання. З-поміж методів такого синтезу заслуговує уваги метод просторового ГСПД, який забезпечує оптимізацію одночасно на етапах вибору ресурсів, складання розкладу і призначення на ресурси. Але для його впровадження необхідно розробити та налаштувати метод цілочисельної оптимізації, який дасть змогу виконувати автоматизоване відображення ГСПД довільних розмірів за прийнятний час. Методи еволюційної оптимізації мають великі перспективи для вирішення цього завдання.

Тому в цьому розділі детально розглядаються метод просторового ГСПД, методи еволюційної оптимізації та на основі них розробляється метод проєктування конвеєрних пристроїв для ЦОС.

2.1 Метод просторового ГСПД

2.1.1 Графи синхронних потоків даних

Як вказувалось у розділі 1, ГСПД задає алгоритм безвідносно результівного розкладу. Цей розклад залежить від конкретної архітектури ОС, де реалізується алгоритм. Суть високорівневого синтезу полягає в представленні ГСПД, верифікації його шляхом моделювання і трансформації такого опису в опис ОС із необхідними параметрами на рівні регістрових передач. Отже, ГСПД можна розглядати як початкові дані для синтезу апаратних засобів ЦОС.

До ГСПД у цій роботі визначено таку низку вимог:

— ГСПД є однорідним;

— актори виконують свої функції миттєво, функція запуску актора — це наявність усіх потрібних даних на його входах;

— розрізняються потоки з буферами FIFO і без них, потік без буфера передає дані (мітки) миттєво в напрямку, вказаному ребром, усі буфери FIFO синхронізуються з єдиного джерела синхросерії та спрацьовують за фронтом синхросерії.

Остання вимога накладає обмеження на ймовірні розклади виконання алгоритму, але і є необхідною умовою для можливості синтезу ОС на основі VHDL-опису [14].

Розглянемо приклад алгоритму обчислення диференційного рівняння

$$y'' + bxy + cy = 0,$$

що використовується для тестування методів високорівневого синтезу [69, 182]. Воно розв’язується за алгоритмом:

$$\begin{aligned}
&i = 0;\\
&\text{while } (x_i < eps) \text{ do}\\
&\quad x_{i+1} = x_i + dx;\\
&\quad u_{i+1} = u_i - b \cdot x_i \cdot u_i \cdot dx - c \cdot y_i \cdot dx;\\
&\quad y_{i+1} = y_i + u_i \cdot dx;\\
&\quad i = i + 1;\\
&\text{end;}
\end{aligned} \qquad (2.1)$$

де i — номер ітерації, eps, b, c, dx — константи. Алгоритм зупиняється, тобто, переривається запис у регістри змінних x_i, u_i, y_i, при досягненні межі $x_i \geq a$.

Цей алгоритм зображено як ГСПД на рис. 2.1. На ньому знаками “+”, “×”, пустим кружечком і товстою крапкою позначені вершини множення на

коефіцієнт, додавання, представлення i+1-х та i-х змінних, відповідно, числами позначені номери вершин та дуг. Пунктиром позначено дуги міжітераційної залежності, які навантажені затримками, позначеними товстими відрізками. Змінні в ГСПД позначаються мітками.

Виконання алгоритму в ГСПД полягає в переміщенні міток (даних) уздовж дуг, спрацьовуванні вершин та видаванні міток на виходи вершин. Мітка в дузі, яка навантажена k затримками, затримується в ній на k ітерацій. Чергова ітерація закінчується в момент, коли мітки повертаються в початкові вершини, наприклад, позначені на рис. 2.1 кружечком.

При одиничному відображенні ГСПД у структуру пристрою вершини додавання, множення, затримки відображаються у відповідні помножувачі, суматори, регістри, які сполучені лініями зв'язку узгоджено з ГСПД.

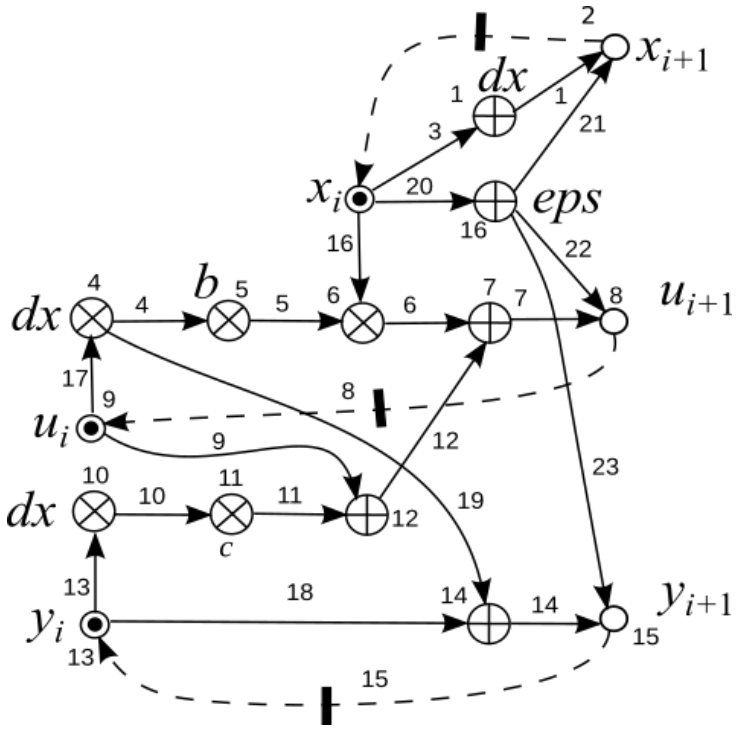

Рисунок 2.1 — ГСПД розв'язання диференційного рівняння

Водночас розклад алгоритму визначається поведінкою ГСПД та затримками логічних схем, у які відображаються вершини графу, цикл виконання

однієї ітерації дорівнює $L = 1$ такт. У цьому прикладі довжина критичного шляху дорівнює $T_C = 3t_M + t_A$, де t_M, t_A — затримки помножувача та суматора, відповідно.

2.1.2 Просторовий ГСПД

Довільний граф алгоритму G_A, зокрема ГСПД, задається матрицею інцидентності

$$A = ||a_{i,j}||_{N*M},$$

де $a_{i,j} = 1$, якщо i-та вершина інцидентна дузі j графа G_A та є її кінцем, або $a_{i,j} = 1$, коли i-та вершина є початком дуги j, та $a_{i,j} = 0$ — якщо i-та вершина не інцидентна дузі j, N і M кількість вершин та дуг графа G_A.

У загальному випадку, ГСПД можна представити у n-вимірному просторі $\mathbb{Z}^n$. Вектори $\boldsymbol{K}_i \in \mathbb{Z}^n$ ототожнюються з вершинами графа G_A. Координати векторів $\boldsymbol{K}_i$ можуть мати різну інтерпретацію. Частина координат — це координати $\boldsymbol{K}_{Si}$ ПЕ, у якому виконується i-й оператор, а інша частина $\boldsymbol{K}_{Ti}$ — кодує момент часу виконання оператора $\boldsymbol{K}_i = \left(\boldsymbol{K}_{Si}^T, \boldsymbol{K}_{Ti}^T\right)^T$.

Тобто, вектор $\boldsymbol{K}_i$ грає роль тега для i-ї вершини. Деякі координати $\boldsymbol{K}_i$ можуть кодувати відлік, наприклад, рівень ієрархії алгоритму і структури, тип оператора й ПЕ, затримку виконання оператора, особливості операторів, такі, як необхідність конвеєрної реалізації, заданий момент вводу даного тощо.

При розміщенні ГСПД алгоритму в $n = 3$-вимірному цілочисельному просторі $\mathbb{Z}^3$ i-й оператор ототожнюється з вектором $\boldsymbol{K}_i = (s, p, t)^T$, ($i = 1 \ldots N$), де p — тип операції, s — просторова координата, t — часова координата. Далі буде розглядатися ГСПД саме у 3-вимірному цілочисельному просторі.

Вважається, що вершини ГСПД генерують мітки, що означають окремі змінні x_j. Якщо оператор генерує кілька результатів, які прямують у різні оператори, то цей оператор необхідно розщепити так, щоби він виконувався в кількох вершинах ГСПД, які генерують окремі змінні.

Аналогічно, кожна змінна x_j з M змінних алгоритму ототожнюється з n-вимірним вектором $\boldsymbol{D}_j$; (j = 1 ... M) ∈ $\mathbb{Z}^3$, причому

$$\boldsymbol{D}_j = \boldsymbol{K}_l - \boldsymbol{K}_i, \tag{2.2}$$

де $\boldsymbol{K}_i$ відповідає оператору з результатом x_j, а $\boldsymbol{K}_l$ — оператор, для якого x_j — це операнд, тобто, $\boldsymbol{K}_i$ безпосередньо передує $\boldsymbol{K}_l$.

Вважається, що багатовимірний простір має якісь метрики. У цьому випадку є просторова метрика $S(\boldsymbol{K}_{Si})$, яка виражає абсолютні координати в просторі та часова метрика $T(\boldsymbol{K}_{Ti})$, яка визначає конкретний момент виконання i-го оператора. Відповідно, $S(\boldsymbol{D}_{Sj})$ дорівнює відстані між двома ПЕ, а $T(\boldsymbol{D}_{Tj})$ — різниці моментів виконання суміжних операторів і є більшою або дорівнює тривалості виконання оператора, вершина якого інцидентна ребру $\boldsymbol{D}_j$ та є його початком.

Для того, щоби задати ГСПД через ці множини векторів $\boldsymbol{K}_i$, $\boldsymbol{D}_j$,, введені наступні визначення.

Конфігурацією алгоритму (КА) є трійка $C_A = (K, D, A)$, де $K = (\boldsymbol{K}_1, ..., \boldsymbol{K}_N)$ — матриця координат векторів-вершин $\boldsymbol{K}_i = (s_i, p_i, t_i)^{\mathrm{T}} \in \mathbb{Z}^3$; $D = (\boldsymbol{D}_1, ..., \boldsymbol{D}_{M+1})$ — матриця координат векторів-дуг $\boldsymbol{D}_j = (s_j, p_j, t_j)^{\mathrm{T}} \in \mathbb{Z}^3$; $A = || a_{i,j} ||_{N*(M+1)}$ — матриця інцидентності графа G_A, яка доповнена M+1-м стовпцем з елементом $a_{p,M+1} = 1$. Цей елемент відповідає базисному вектору $\boldsymbol{D}_{M+1} = \boldsymbol{D}_B$, який з'єднує початок системи координат простору $\mathbb{Z}^3$ з довільним вектором $\boldsymbol{D}_B = \boldsymbol{K}_p \in K$ [18].

2.1.3 Основи синтезу конвеєрних ОС на базі просторового ГСПД

Якщо ГСПД представлений у просторі $\mathbb{Z}^3$, то відповідно до умови ін'єктивності відображення алгоритму, граф структури ОС і моменти виконання операторів мають бути представлені в підпросторах $\mathbb{Z}^2$ і $\mathbb{Z}$. Конфігурацією структури (КС) є трійка $C_S = (K_S, D_S, A)$, де $K_S = (K_{S1}, ..., K_{SN})$, $D_S = (D_{S1}, ..., D_{S(M+1)})$ — матриці координат векторів-вершин $K_{Si} = (s_i, p_i)^{\mathrm{T}} \in \mathbb{Z}^2$ і

векторів-дуг $D_{Sj} = (s_j, p_j)^{\mathrm{T}} \in \mathbb{Z}^2$, A — матриця інцидентності ГСПД. K_{Si} — це координати ПЕ, у якому виконується i-й оператор, а D_{Sj} — відносні координати лінії зв'язку, якою передається j-а змінна.

Конфігурацією передування (КП) є трійка $C_T = (K_T, D_T, A)$, де $K_T = (t_1, ..., t_N)$, і $D_T = (d_1, ..., d_{(M+1)})$. Однорядковій матриці K_T відповідає вектор часової розгортки $T = (t_1, ..., t_N)^{\mathrm{T}}$, що вказує, у які моменти часу виконуються операторні вершини.

На рис. 2.2 показані приклад КА (а) та відповідні КС (б) і КП (в).

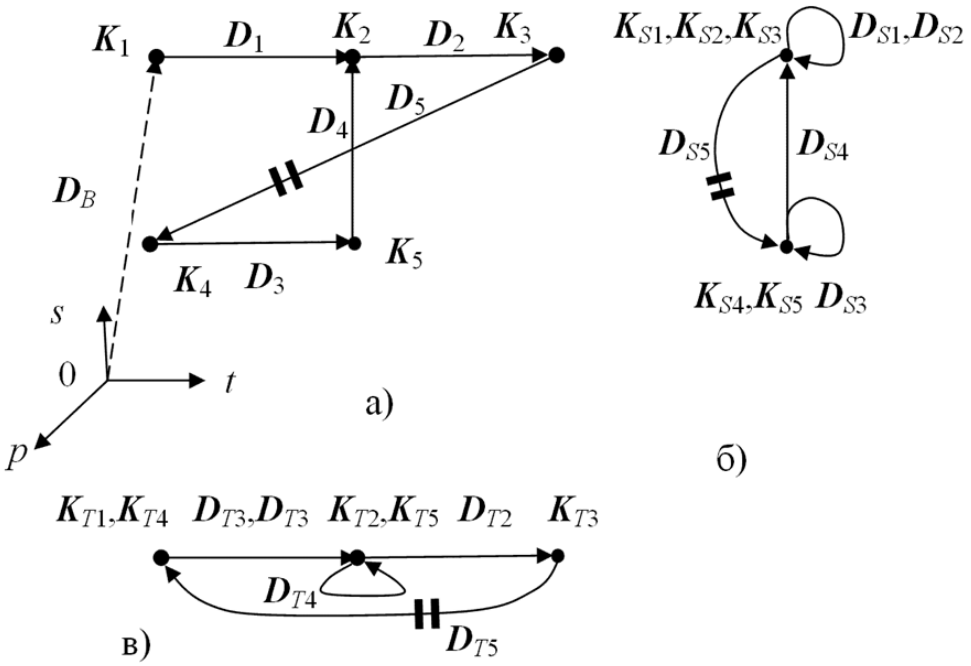

Рисунок 2.2 — Приклад КА та відповідні КС і КП

КА є коректною, якщо в матриці K відсутні пари однакових векторів, тобто

$$\forall\, \boldsymbol{K}_{\mathrm{i}}, \boldsymbol{K}_{\mathrm{j}}\ (\boldsymbol{K}_{\mathrm{i}} \neq \boldsymbol{K}_{\mathrm{j}},\ i \neq j). \qquad (2.3)$$

Якщо ГСПД має цикли, то повинна дорівнювати нулю сума векторів-дуг $\boldsymbol{D}_{\mathrm{j}}$, що входять у будь-який цикл графа, тобто, для i-го циклу

$$\sum b_{\mathrm{i,j}}\, \boldsymbol{D}_{\mathrm{j}} = (0,0,0)^{\mathrm{T}}, \tag{2.4}$$

де $b_{\mathrm{i,j}}$ — елемент i-го рядка цикломатичної матриці ГСПД C_T для ГСПД є (строго) коректною відносно часової функції R, якщо

$$\begin{aligned}
&\boldsymbol{K}_{Tr} - \boldsymbol{K}_{Ti} = \boldsymbol{D}_{Tj} \in D_T \Rightarrow \\
&\begin{cases} R\left(\boldsymbol{K}_{Tr}\right) \geq R\left(\boldsymbol{K}_{Ti}\right), \text{ — для ненавантажених дуг} \\ R\left(\boldsymbol{K}_{Tr}\right) < R\left(\boldsymbol{K}_{Ti}\right). \text{ — для дуг } D_{Dj} \text{, що навантажені затримками} \end{cases} \\
&i, r = 1, ..., N; \quad j = 1, ..., M + 1;
\end{aligned} \tag{2.5}$$

(для строгої коректності — нерівність строга), тобто, якщо вершини $\boldsymbol{K}_{Tr}$, $\boldsymbol{K}_{Ti}$ сполучені дугою, то момент виконання оператора попередньої вершини $\boldsymbol{K}_{Ti}$ менший за момент виконання наступної $\boldsymbol{K}_{Tr}$. На рис. 2.2 дуга $\boldsymbol{D}_5$ навантажена двома затримками, що означає, що в ній операнд затримується на 2 ітерації.

У простому випадку кожен оператор алгоритму виконується за один такт, тобто, $R(s_i,\ p_i,\ t_i)^{\mathrm{T}} = t_i$. Такий випадок є натуральним у проєктуванні конвеєрних обчислювальних пристроїв (ОП) на RTL-рівні. Складні оператори можуть виконуватися за кілька тактів. Але, працюючи в конвеєрному режимі, відповідні їм ПЕ мають ланцюжок конвеєрних ступенів, кожний із яких має однотактову затримку.

Тоді $R(\boldsymbol{D}_{\mathrm{j}}) = 0$, якщо відповідний ПЕ не має регістра результату й передає результат негайно в наступний ПЕ, $R(\boldsymbol{D}_{\mathrm{j}}) = p$, якщо на виході ПЕ стоїть буфер FIFO з p регістрами. На рис. 2.3 показана модель ПЕ, у який відображаються операторні вершини. Вона має арифметико-логічний пристрій ALU, вихід якого підключений до буфера FIFO. Входи ALU підключені до виходів мультиплексорів, на входи яких поступають дані з інших ПЕ або з даного ПЕ.

Дуги міжітераційної залежності, що навантажені числом w,

$$\boldsymbol{D}_{Dj} = (k_i,\ s_i,\ -wL)^T \tag{2.6}$$

означають затримку на w циклів (ітерацій). Отже, просторовий ГСПД є об'єднанням ациклічного графа, який виконує обчислення однієї ітерації, та множини дуг $\boldsymbol{D}_{Dj}$, які зображують міжітераційні затримки змінних на wL тактів.

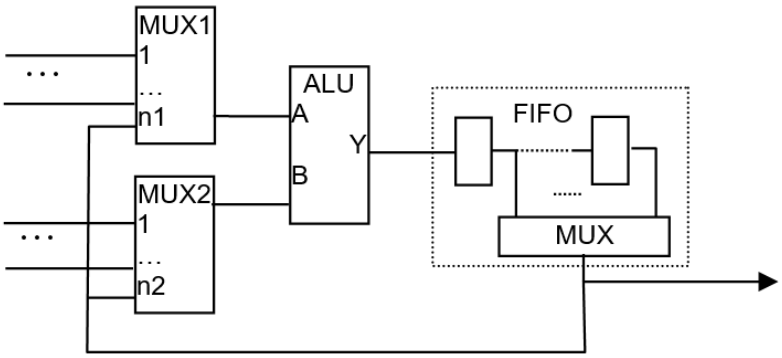

Рисунок 2.3 — Модель ПЕ з пам'яттю

Розклад для КА C_A, з періодом L тактів є коректним, якщо оператори, які відображаються в один і той самий ПЕ, виконуються в різних тактах, тобто:

$$\forall\, \boldsymbol{K}_i\,,\, \boldsymbol{K}_j \left(k_i = k_j\,,\, s_i = s_j\right) \Rightarrow$$
$$\begin{cases} t_i \equiv t_j \bmod L \text{ — } \left(\boldsymbol{K}_i, \boldsymbol{K}_j\right) \text{ — дуга, що навантажена затримками} \\ t_i \not\equiv t_j \bmod L \text{ — у решті випадків} \end{cases} \tag{2.7}$$

Однотипні оператори треба відображати в ПЕ того самого типу:

$$\boldsymbol{K}_i, \boldsymbol{K}_j \in K_{p,\,q} \left(k_i = k_j = p,\, s_i = s_j = q\right),\ \left|K_{p,\,q}\right| \le L, \tag{2.8}$$

де $K_{p,\,q}$ — множина векторів-вершин операторів p-го типу, що відображаються в q-й ПЕ p-го типу ($q = 1, 2, \ldots, q^p_{max}$).

КА може бути перетворена в КС і КП так:

$$K = \begin{pmatrix} K_S \\ K_T \end{pmatrix}; \quad D = \begin{pmatrix} D_S \\ D_T \end{pmatrix}, \tag{2.9}$$

якщо КА, КП і розклад є коректними, тобто, вони задовольняють умови (2.3) – (2.8), причому одержана модель ОС буде правильно виконувати заданий алгоритм у конвеєрному режимі з періодом L тактів.

Ефективна КА шукається за два етапи. На першому етапі ГСПД розміщується в тривимірному просторі як множини векторів $\boldsymbol{K}_{\mathrm{i}}$ і $\boldsymbol{D}_{\mathrm{j}}$ з урахуванням умов (2.3) – (2.8), тобто, формується початкова КА. Кількість ПЕ в структурі мінімізується через виконання умови $|K_{\mathrm{p,q}}| \rightarrow L$, тобто, коли кількість вершин, що відображаються в один ПЕ, прямує до L. Також виконується перестановка вершин відносно осі часу *ot*, яка відповідає ресинхронізації ГСПД [207].

На другому етапі КА урівноважується. При цьому розглядається ациклічний підграф ГСПД без дуг $\boldsymbol{D}_{\mathrm{D}j}$, які навантажені затримками. У всі його дуги включаються проміжні вершини операторів затримки (регістрів). В урівноваженій КА всі вектори-дуги, крім навантажених дуг, дорівнюють $\boldsymbol{D}_{\mathrm{j}} = (a_{\mathrm{j}}, b_{\mathrm{j}}, 1)^{\mathrm{T}}$ або $\boldsymbol{D}_{\mathrm{j}} = (a_{\mathrm{j}}, b_{\mathrm{j}}, 0)^{\mathrm{T}}$. Водночас вершини-оператори формують яруси, відстань між якими за координатою *ot* дорівнює 1 такт. Урівноважена КА оптимізується через взаємні перестановки векторів-вершин з одного ярусу, які призводять до мінімізації кількості регістрів та входів мультиплексорів.

2.1.4 Пошук ОС за допомогою розв'язання задачі цілочисельної лінійної оптимізації

З рівності (2.2) та визначення КА випливає таке рівняння:

$$D = KA. \tag{2.10}$$

Відповідно, існує зворотне рівняння

$$K = D_O \mathrm{A}_O^{-1}, \tag{2.11}$$

де D_O — матриця векторів-дуг; A_O — матриця інцидентності кістяка ГСПД.

Тоді пошук оптимального рішення полягає в знаходженні такої матриці K, яка мінімізує критерій якості. Для цього спочатку задають вектори матриці D_O, що забезпечують мінімальну тривалість виконання ітерації алгоритму, а вектори $\boldsymbol{K}_i$ знаходять зі співвідношення (2.11). При пошуку ефективних структурних рішень необхідно також керуватися закономірностями (2.3) – (2.8).

За цих умов пошук розкладу полягає в наступному. Необхідно призначити всім векторам-дугам $\boldsymbol{D}_i \in D_O$ координату $t_i = 1$, тобто встановити затримку в один такт і знайти K_T з відношення (2.11). Решту елементів матриці D_T знаходять із рівнянь (2.10) та (2.8). Якщо для деяких із них не справджується нерівність (2.7), то збільшують координату t_i у певних векторів $\boldsymbol{D}_i \in D_\mathrm{O}$ й повторюють пошук. Решту координат векторів $\boldsymbol{K}_i$ знаходять з умов (2.7), (2.11).

Отож, будується розклад типу *ASAP*, позаяк кожен оператор виконується за один такт і немає зайвих затримок. Якщо одержана матриця K не задовольняє критерій оптимальності, то змінюють деякі вектори $\boldsymbol{D}_i$, які пов’язані з цими ресурсами, або змінюють L і повторюють пошук розкладу.

При формуванні розкладу ГСПД під час пошуку решти векторів D_{Tj} з рівняння (2.10) імовірна поява некоректного розкладу за умовою (2.7). Кістяк ГСПД, підібраний для знаходження розкладу, називається досконалим. Досконалий кістяк — це кістяк ГСПД, у якому: а) немає стягувальних дуг, б) немає дуг міжітераційних залежностей, в) дуги входять у найдовші маршрути, г) відмічені вершини, у які заходять критичні поперечні дуги. У роботі [25] доведено, що до досконалого кістяка належить критичний шлях алгоритму й що розклад, побудований на його основі, є розкладом типу *ASAP*.

Розглянемо приклад синтезу ОС, що реалізує алгоритм (2.1). Частково цей приклад опубліковано в [28]. На першому етапі формується початкова КА. У досконалому кістяку ГСПД цього алгоритму (рис. 2.4) вектор-дуга $\boldsymbol{D}_{\text{В}}$ з'єднує початок системи координат із довільною вершиною.

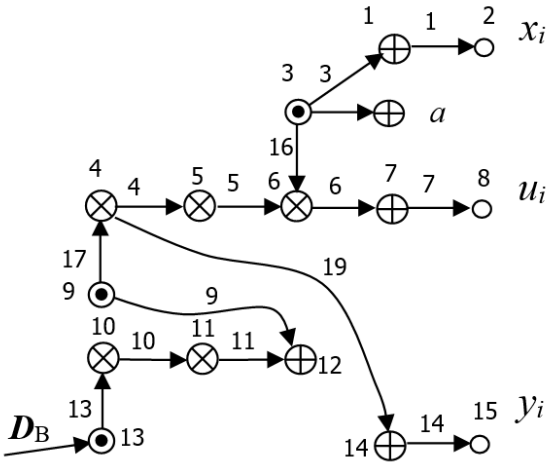

Рисунок 2.4 — Досконалий кістяк для ГСПД на рис. 2.1

У цьому прикладі використовуються суматори, що мають затримку в один такт та конвеєрні блоки множення з затримкою у два такти. З урахуванням цього, складається матриця затримок дуг кістяка:

$$D_{OT} = \begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 9 & 10 & 11 & 13 & 14 & 16 & 17 & 18 & B \\ 1 & x_1 & 2 & 2 & 2 & 1 & x_2 & 2 & 2 & x_3 & 1 & x_4 & x_5 & x_6 & 0 \end{pmatrix}$$

де $x_1, \ldots, x_5$ — невідомі величини, які залежать від векторів із рівнянь циклів (2.4), кількість яких дорівнює кількості дуг, які доповнюють кістяк до ГСПД. У цьому ГСПД п'ять циклів і, відповідно, складається система з п'яти рівнянь (2.4):

$$\begin{cases} \boldsymbol{D}_1 + \boldsymbol{D}_2 + \boldsymbol{D}_3 = 0, \\ \boldsymbol{D}_4 + \boldsymbol{D}_5 + \boldsymbol{D}_6 + \boldsymbol{D}_7 + \boldsymbol{D}_8 + \boldsymbol{D}_{17} = 0, \\ \boldsymbol{D}_{14} + \boldsymbol{D}_{15} + \boldsymbol{D}_{18} = 0, \\ \boldsymbol{D}_4 + \boldsymbol{D}_5 + \boldsymbol{D}_6 - \boldsymbol{D}_{12} - \boldsymbol{D}_9 + \boldsymbol{D}_{17} = 0, \\ \boldsymbol{D}_{13} + \boldsymbol{D}_{10} + \boldsymbol{D}_{11} - \boldsymbol{D}_9 + \boldsymbol{D}_{17} + \boldsymbol{D}_{19} - \boldsymbol{D}_{18} = 0. \end{cases}$$

Розв’язавши цю систему рівнянь, знаходять $x_1 = 6$; $x_2 = 5$; $x_3 = 5$; $x_5 = 0$; $x_6 = 6$. Змінна $x_4 = 6$ вибирається довільно. Період алгоритму $L = 7$ вибирається як довжина критичного шляху:

$$\boldsymbol{D}_8 = (\boldsymbol{D}_4 + \boldsymbol{D}_5 + \boldsymbol{D}_6 + \boldsymbol{D}_7 + \boldsymbol{D}_{17});$$

За (2.11) знаходимо матрицю K_T:

$$K_T = A_O \begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 9 & 10 & 11 & 13 & 14 & 16 & 17 & 18 & B \\ 1 & 6 & 2 & 2 & 2 & 1 & 5 & 2 & 2 & 5 & 1 & 6 & 0 & 6 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 8 & 9 & 2 & 4 & 6 & 8 & 10 & 11 & 4 & 5 & 7 & 9 & 0 & 6 & 7 \end{pmatrix}.$$

Перевіряється, що справджується вимога (2,7). Решту координат матриці K одержують згідно з (2.3), зважаючи на те, що кількість вершин, що стоять в одному рядку, паралельному осі ox, має прямувати до L:

$$K = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 2 & 3 & 4 & 1 & 1 & 1 & 2 & 3 & 4 & 1 & 1 & 2 & 4 & 2 & 3 \\ 1 & 4 & 3 & 2 & 2 & 2 & 1 & 4 & 3 & 2 & 2 & 1 & 3 & 1 & 4 \\ 8 & 9 & 2 & 4 & 6 & 8 & 10 & 11 & 4 & 5 & 7 & 9 & 0 & 6 & 7 \end{pmatrix} \qquad (2.12)$$

На рис. 2.5 показаний побудований просторовий ГСПД.

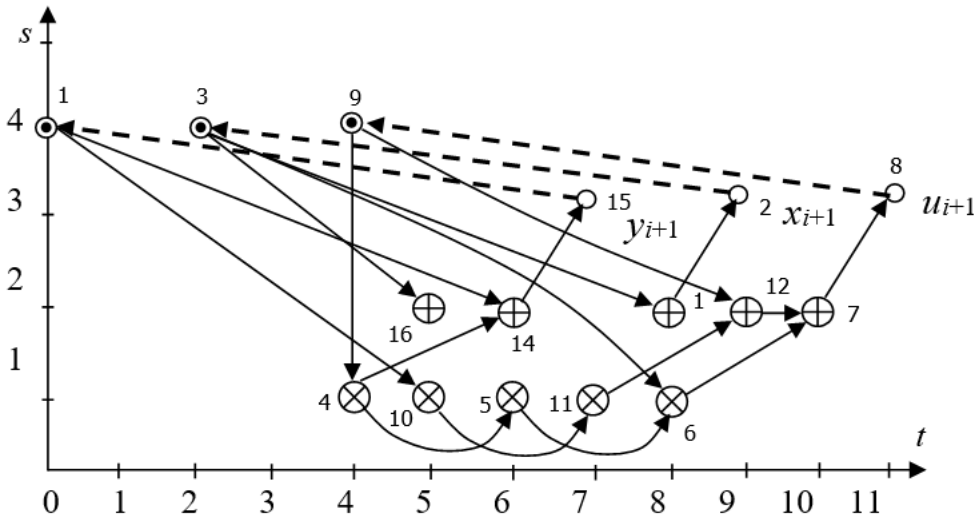

Рисунок 2.5 — Просторовий ГСПД, який обчислює рівняння (2.1)

На другому етапі КА урівноважується. Урівноважена КА показана на рис. 2.6. На ньому кружечком показано вершину затримки.

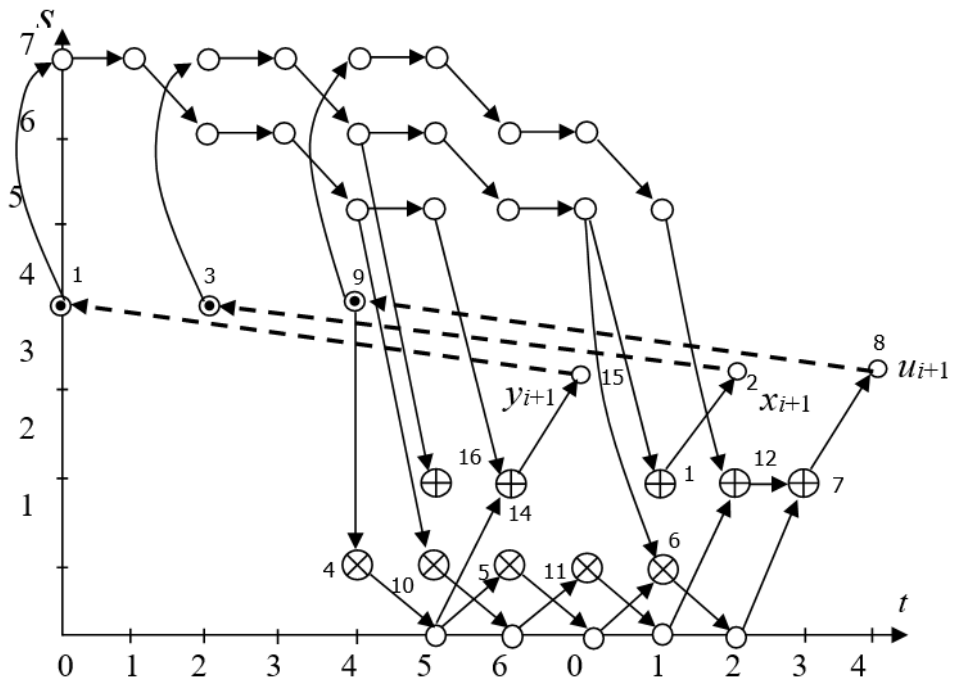

Рисунок 2.6 — Урівноважений просторовий ГСПД, який обчислює рівняння (2.1)

Відлік тактів виконується за модулем $L = 7$, щоби показати справдження умови (2.7) та спростити подальший опис мовою VHDL. Опис синтезованого пристрою мовою VHDL, який побудований за просторовим ГСПД на рис. 2.6, наведений нижче.

```
entity ODE1 is port(CLK,RST,START:in bit;
          XO,YO:out integer);
end entity ODE1;
architecture synt of ODE1 is
      constant x0:integer:=100;
      constant u0:integer:=0;
      constant y0:integer:=200;
      constant eps:integer:=4000;
      constant dx: integer:=3;
      constant b: integer:= 5;
      constant c: integer:= 6;
      signal ct7: integer range 0 to 7;
      signal r0,r5,r6,r7,sm,mpu: integer;
      signal rdy:bit;
begin
      CNTRL: process(CLK,RST)
      Begin
            if RST='1' then ct7<=0;rdy<='1';
```

```
            elsif CLK='1' and CLK'event then
                  if ct7=8 then ct7<=0; -- clock counter
                  else ct7<=ct7+1;
                  end if;
                  if START='1' and ct7=6 then
                        rdy<='0';
                  elsif ct7=6 and sm>0 then
                        rdy<='1';
                  end if;
            end if;
      end process;

      DATAPATH: process(CLK,RST) -- datapath
      begin
            if CLK='1' and CLK'event then
                  if START='1' and ct7=6 then
                        r6<=u0; r5<=x0;sm<=y0 + u0*dx;
                        mpu<=u0*dx*b;r0<=y0*dx;
                  elsif rdy='0' then
                         case ct7 is
                              when 0 => r0<=mpu;mpu<=r0*c;r7<=sm;YO<=sm;
                              when 1 => r0<=mpu; mpu<= r0*c; sm<= r5+dx;
                              when 2 => r0<=mpu; sm<=r5-r0;
                                        r7<=x0; r6<=r7;XO<=sm;
                              when 3 => sm<=sm-r0;
                              when 4 => mpu<=sm*dx; r7<=sm; r6<=r7; r5<= r6;
                              when 5 => r0<=mpu;mpu<= r5*dx; sm<=r6-eps;
                              when others=>r0<=mpu;mpu<=r0*b;
                                           sm<=r0+r5;r6<=r7;r5<= r6;
                         end case;
                  end if;
            end if;
      end process;
end synt;
```

Опис має два оператори процесу. Процес CNTRL описує керуючий автомат, який генерує сигнал закінчення виконання алгоритму rdy та 7 тактів виконання алгоритму за лічильником ct7. Процес DATAPATH описує поведінку конвеєра пристрою в кожному із семи тактів алгоритму. Спочатку регістри конвеєра встановлюються в початковий стан і по закінченню сигналу START конвеєр починає виконання алгоритму. При цьому в кожному рядку оператора case описані всі дії, які відбуваються в тому чи іншому такті алгоритму відповідно до ГСПД на рис. 2.6.

Цей опис можна скомпілювати будь-яким компілятором-синтезатором в ОС під відповідну технологію. Критичний шлях у цій конвеєрній ОС проходить через блок множення з мультиплексором. Отже, тут досягається максимально можлива тактова частота. При цьому, власне, структура ОС не будується. На рис. 2.7 наведена схема синтезованої ОС для довідки.

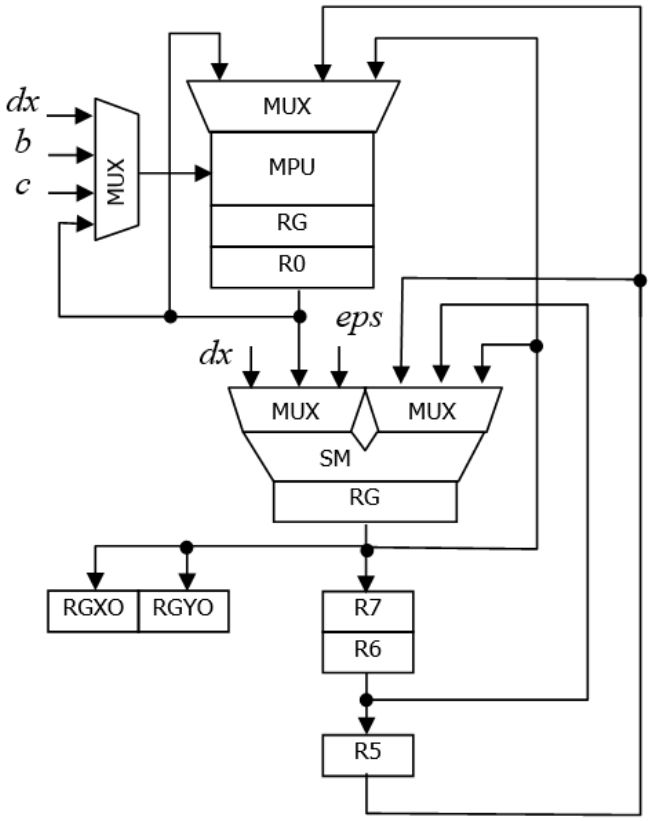

Рисунок 2.7 — Структура конвеєрної ОС, що виконує алгоритм (2.1)

2.1.5 Оцінювання методів синтезу ОС за допомогою розв'язання задачі цілочисельної лінійної оптимізації

Цілочисельна оптимізація розкладу полягає в побудові низки оптимізованих рішень та виборі найбільш переважного рішення за критерієм оптимальності. Кожне з таких рішень однозначно кодується матрицею K. Початкове оптимізоване рішення обчислюється як розклад, одержаний за правилами, які описані в підрозділі 2.1.4. Решта рішень одержується завдяки перетворенням просторового ГСПД за допомогою перестановки векторів $\boldsymbol{K}_i$ у просторі та взаємної перестановки таких векторів з урахуванням умов (2.3) – (2.8). Так відбувається сканування простору ефективних рішень.

При розв’язанні задачі ЦЛП оптимальне рішення знаходиться лише для ГСПД, кількість вершин яких обмежена двома-трьома десятками. Умови (2.3) – (2.8) є межами сканування простору ефективних рішень. Тож удосконалений метод на основі гілок та меж навряд суттєво відсуне границю розмірів ГСПД, які можуть бути оброблені за допомогою такої оптимізації. У наступному підрозділі розглядаються методи еволюційної оптимізації, які придатні для синтезу ОС на основі ГСПД довільних розмірів.

2.2 Еволюційні методи оптимізації структур

У першому розділі було наголошено про перспективність напряму генетичних алгоритмів для оптимізації структур ОС. У цьому підрозділі ці алгоритми розглядаються з метою подальшого формування методу структурного синтезу пристроїв для ЦОС.

2.2.1 Принципи еволюційних методів оптимізації

Метагевристика — це назва широкого класу методів розв’язання комбінаторних задач оптимізації, які, на відміну від аналітичних методів, знаходять рішення за кілька ітерацій. Метагевристика зазвичай застосовується до задач, для яких відомі алгоритми мають асимптотичну часову складність. Хоча такий метод не дає гарантії, що буде знайдене оптимальне рішення, метагевристика пропонує хороші шанси знайти “достатньо вдале” рішення.

Метагевристики працюють ітеративно, покращуючи набір кандидатських, ефективних рішень. Властивістю алгоритмів такої оптимізації є те, що вони, як правило, можуть бути припинені після будь-якої ітерації, оскільки в будь-який момент часу вони мають, принаймні, якихось оптимізованих кандидатів на рішення.

Клас метагевристик застосовує фізичні аналогії, як, наприклад, моделюване відпалювання множини рішень [128] або алгоритм затоплення цієї множини [80]. Водночас, ще більший клас метагевристик спирається на

принципи біологічної еволюції та процеси в організмах у різних формах. Це оптимізація рою [126], яка імітує поведінку, наприклад, птахів, які шукають їжу. Оптимізація колонії мурашок [78] імітує поведінку мурах. Поведінка сім'ї бджіл є основою методу [166]. Імунна система хребетних моделюється в [86].

Еволюційні алгоритми — це узагальнені метагевристичні алгоритми оптимізації, які використовують такі механізми, як мутація, схрещування, природний добір та тести виживання, щоб ітераційно удосконалювати набір кандидатів на рішення [133]. Вони мають аналогію теорії еволюції Дарвіна. Ця теорія ґрунтується на наступних спостережених закономірностях.

Особини виду мають велику родючість, але чисельність популяції виду приблизно залишається постійною. Оскільки особини змагаються за обмежені харчові ресурси, триває боротьба за виживання. Варіації здатностей особин впливають на здатність до виживання. Частина варіацій є спадковою. Найкращі особини виживають і дають потомство, якому ймовірніше передадуть свої риси. Вид повільно пристосовується до певного середовища під час еволюції, що може призвести до появи нових видів. Благо еволюції — це суб'єктивна категорія й розглядається як встановлення балансу видів на певній території.

Теорія розширеного еволюційного синтезу додає уточнення до теорії Дарвіна. Це такі наступні феномени й поняття. Селекція доповнюється розвитком задля адаптації до зміни зовнішнього середовища. Причому фенотипи формують ніші особин із подібними ознаками. Трапляється невипадкова мутація, яка спричинена зміною оточення фенотипу. Зміни генів можуть бути також великими. Під еволюцією розуміється не просто зміна в генах, а адаптивне поширення успадкованих ознак. Під макроеволюцією розуміється еволюційний процес із накопиченням ознак, формуванням шаблонних генів, трендів, які стають помітними через кілька поколінь [139].

Еволюційні алгоритми (ЕА) абстрагуються від біологічних процесів, а також вносять зміну до семантики блага еволюції, яке стає метою задачі

синтезу [139]. Простір пошуку $\mathbb{G}$ в еволюційних алгоритмах абстрагований до множини всіх можливих хромосом, тобто, рядків $g \in \mathbb{G}$, що кодують конкретне технічне рішення. Отже, g кодує генотип певної особини x, яка є кандидатом на рішення з простору рішень $\mathbb{X}$. Особина x моделюється як фенотип, який одержується за відображенням генотипу $x = \text{Phenotype}(g)$. Ступінь відповідності оптимальному рішенню визначається за критеріальною функцією оптимальності $Q(x)$, яка рухає процес еволюції в необхідному напрямі, тобто функція $Q(x)$ означає формальне благо еволюції.

2.2.2 Еволюційний алгоритм загального виду

У принципі, усі ЕА є варіаціями та розширеннями базового підходу, визначеного наступним алгоритмом, який спирається на функції та змінні, визначені нижче.

Алгоритм 1.

Вхідні дані: TCriterion() — критерій зупинки алгоритму;

Cmpf — відношення порядку;

ps — розмір популяції.

Змінні: *t* — лічильник поколінь;

Pop — популяція, яка складається з генів *g*;

Popm — частина популяції для розмноження;

v — величина придатності.

Результат — набір найкращих рішень.

begin

t = 0;

Pop = Initialize(*ps*);

while not TCriterion() {

t = *t* + 1;

v = Fitness(*Pop*(*t* – 1), *Cmpf*),

Popm = Select(*Pop*(*t* – 1), *v, ps*);

Pop(*t*) = Reproduce(*Popm*);

}

return Phenotype(Optimal(*Pop*));

end;

Функція Initialize(*ps*) генерує початкову випадкову популяцію з номером $t = 0$, яка складається з *ps* осіб.

Функція TCriterion() перевіряє, чи справджується критерій переривання алгоритму.

Значення придатності *v* визначається для генотипів кожної особини через обчислення критерію ефективності $Q(x)$ та порівняння згідно з ним з іншими особинами зі множини *Pop*(*t*). При цьому за генотипом *g* будується спрощена модель фенотипу *g.x*, для якої дається оцінка ефективності Fitness.

Функція Select(*Pop*($t - 1$), *v*, *ps*) вибирає множину *ps* особин *Popm*, які придатні для розмноження.

Функція Reproduce(*Popm*) генерує нову популяцію з використанням мутацій, схрещування тощо.

Функції Optimal(*Pop*) та Phenotype використовуються для виділення генотипів найліпших особин та побудови їхніх фенотипів. Рис. 2.8 ілюструє хід виконання еволюційного алгоритму загального виду.

2.2.3 Види еволюційних алгоритмів

Розрізняють п’ять видів еволюційних алгоритмів:

— Генетичні алгоритми (ГА), які оперують бітовими векторами як генотипами $g \in \mathbb{G}$.

— Еволюційні стратегії, які виконують операції з векторами *g* реальних чисел.

— Генетичне програмування (ГП), яке має подвійне визначення. З одного боку — це еволюційні методи створення нових алгоритмів і програм, з

іншого боку — це ЕА, які оперують з особинами, які мають структуру алгоритму, найчастіше — дерева.

— Навчальні класифікаційні системи, які для виведення класифікаційних правил використовують ГА.

— Еволюційне програмування, яке ґрунтується на тому, що вся популяція є шуканим рішенням і тому операція схрещування не має сенсу.

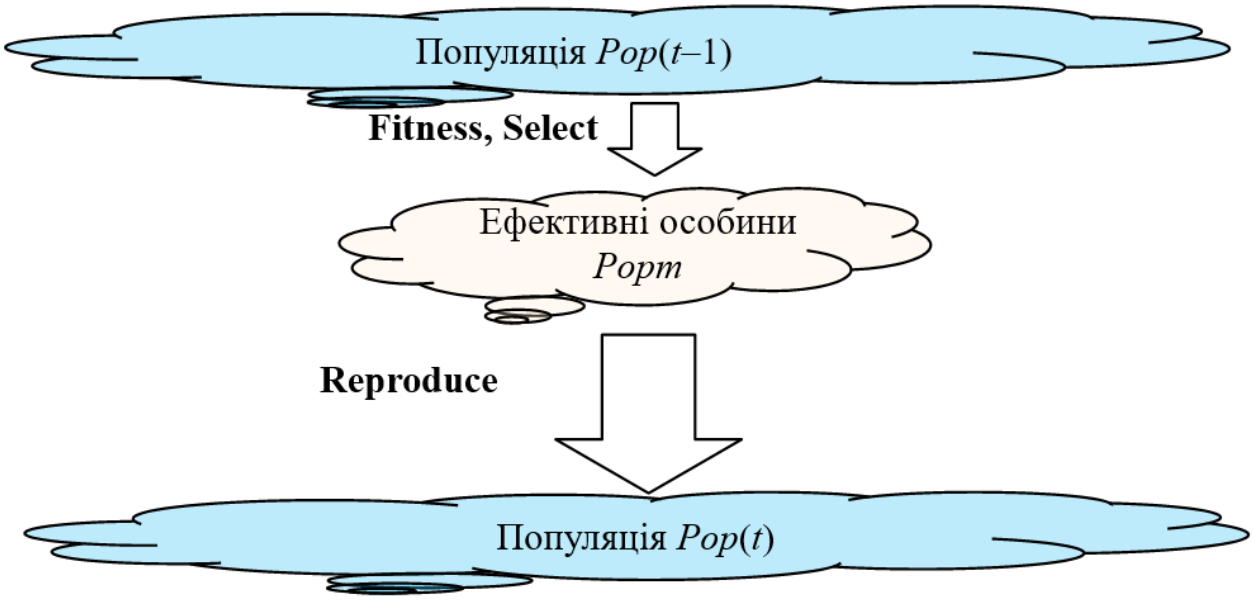

Рисунок 2.8 — Хід виконання еволюційного алгоритму

Усі ці ЕА виконуються за наведеним вище алгоритмом 1. Основні відмінності еволюційних алгоритмів полягають у різних:

— кодуваннях хромосоми *g*;

— чисельностях популяції *ps* чи кількостях використовуваних популяцій;

— способах добору особин для розмноження;

— способах добору народжених особин до складу популяції.

Згідно з цією класифікацією, алгоритми, що розробляються в цій роботі, належать до алгоритмів генетичного програмування, бо їхнім результатом є оптимізовані алгоритми, представлені як ГСПД.

2.2.4 Параметри еволюційного алгоритму

Продуктивність та успішність ЕА, застосованого до задачі, заданої цільовою функцією $Q(x)$ та простору рішень $\mathbb{X}$, визначається особливостями:

— основні параметри алгоритму, такі, як розмір покоління *ps* або коефіцієнти частки покоління для схрещування та мутації;

— те, чи використовує алгоритм множину *Arc* найкращих знайдених осіб, і якщо так, то яка стратегія обмеження використовується для запобігання її надмірному зростанню;

— особливості виконання функцій Fitness та Select;

— простір пошуку $\mathbb{G}$ та функції оптимальності генотипу;

— функція перетворення генотипу у фенотип $x = \text{Phenotype}(g)$, що відображає простір пошуку в простір рішень.

Отже, різні ГА відрізняються між собою саме цим переліком параметрів.

2.2.5 Будова хромосоми в еволюційних алгоритмах

Основним конструктивним об’єктом ЕА є хромосома.

Хромосома — це генотип g, закодований бінарними кодами як вектор або запис кількох векторних полів, тобто, генів.

Гени бувають фіксованої та змінної довжини. Структура генів задається статично або може змінюватися динамічно, наприклад, від мутацій.

Інтрон — це частина хромосоми, яка не впливає на фенотип x.

Від будови та складності хромосоми суттєво залежить складність та ефективність ГА. Так, важливо, щоби за хромосомою g, з одного боку, можна було точно відтворити фенотип x особини, тобто об’єкт, який оптимізується. З іншого боку, хромосома g має дати змогу нескладно побудувати модель

особини $g.x$ таку, щоби за нею можна було б визначити придатність $v(g.x)$, яка якомога ближче б наближалася до критерію оптимальності $Q(x)$.

2.2.6 Селекція в еволюційних алгоритмах

ЕА суттєво розрізняються за принципом реалізації функцій Select, Reproduce та ін. У зв'язку з цим, введені наступні визначення.

Покоління — це популяція, яка містить у собі лише особини, які одержані після розмноження і яка не зберігає батьків [46].

Лівий добір — добір, який не дозволяє відтворюватися кращим особам, щоби запобігти передчасній конвергенції процесу оптимізації.

Правий добір — добір, який не дозволяє розведення найгіршим особам, щоби корисні ознаки не розсіювалися занадто сильно.

Зберігаючий добір — добір, який комбінує особини з минулих поколінь та новонароджених.

Добір можна характеризувати через параметри λ — кількість створених дітей і μ — кількість батьківських особин.

Стратегія добору з вимиранням батьків позначається як (μ, λ) — стратегія, і створює $\lambda \geq \mu$ дитячих особин з μ доступних генотипів, з яких зберігаються лише μ найкращих рішень.

У стратегії зберігаючого добору $(\mu + \lambda)$ також λ дітей створюються від μ батьків. Потім множини батьків та дітей об'єднуються в популяцію розміром $\lambda + \mu$ і в ній «виживають» лише найкращі особини. Більшість із таких стратегій ґрунтується на рандомізації, яка не виключає вибір гірших особин [170].

Обережний еволюційний алгоритм (Steady-state evolutionary algorithm, SSEA) — алгоритм зберігаючого добору, у якому значення λ є малими в порівнянні з μ. Недоліком такої стратегії є ризик передчасного виродження ефективних особин у популяції [68].

Елітний еволюційний алгоритм забезпечує те, що принаймні одна копія найкращих особин поточного покоління передається наступному поколінню. Основна перевага такого алгоритму полягає в тому, що його конвергенція гарантована, але є ризик сходження до локального оптимуму [40].

Доповненням до алгоритму 1 є застосування множини-архіву *Arc*, у яку зберігаються генотипи найкращих особин. Спочатку ця множина пуста, потім вона заповнюється, а гірші генотипи з неї викидаються. Якщо множина *Arc* стає занадто великою, вона скорочується за алгоритмом, який кластеризує елітні особини в ніші й викидає з них гірші, зберігаючи різноманіття осіб. Відповідно, множина *Arc* стає додатковим параметром до функцій Fitness, Select.

Отже, функція Fitness має виділяти найкращі особини за умови збереження різноманіття особин, щоби конвергенція алгоритму не призвела до одного локального оптимуму. Водночас функція загальності дає змогу виділяти ніші та зосереджувати в множині-архіві *Arc* видатні особини з кількох ніш.

2.2.7 Функція оцінки ефективності

Оптимізація технічних рішень ґрунтується на тому, що в просторі рішень $\mathbb{X}$ існує частковий порядок *Cmpf* серед рішень $g.x$, завдяки якому можна вибрати найкращий варіант. Було б простіше, коли частковий порядок був би накладений на простір пошуку $\mathbb{G}$, тобто, порядок серед хромосом g. Але це складно. Частковий порядок *Cmpf* виконується присвоєнням єдиного числа $v(g.x)$ кожному кандидату на рішення $g.x$ — його придатності.

Придатність, яка присвоєна окремій особині, відображає її ранг у популяції, а також може містити інформацію про належність її до певної підмножини особин з аналогічними властивостями — ніші. Отож, враховується не тільки якість кандидата на рішення, але і якість різноманітності популяції. Це може покращити шанс визначити глобальний оптимум, а також

збільшити ефективність оптимізації. Причому створення ніш відповідає теорії розширеного еволюційного синтезу [139].

Таким чином, придатність $v(g.x)$ залежить не тільки від кандидата на рішення $g.x$, але і від усієї популяції *Pop* та від множини-архіву *Arc*. Отже, вона визначається як функція $v{:}\mathbb{X} \longmapsto \mathbb{R}^+$ і записується як

$$v = \text{Fitness}(Pop, Cmpf) \Rightarrow v(g.x) \in V \subseteq \mathbb{R}^+, \forall g \in Pop;$$

за браком множини-архіву *Arc* та

$$v = \text{Fitness}(Pop, Arc, Cmpf) \Rightarrow v(g.x) \in V \subseteq \mathbb{R}^+, \forall g \in Pop \cup Arc;$$

за наявності множини-архіву *Arc*.

У більшості алгоритмів функція Fitness повертає менше значення для кращих особин, тобто,

$$\forall g_1, g_2 \in Pop \cup Arc \; (g_1.x \succ g_2.x \Rightarrow v(g_1.x) < v(g_2.x)).$$

У визначенні функції Fitness найскладнішим є перетворення генотипу g у приблизну модель фенотипу $g.x$. Для спрощення функції Fitness розроблені різні підходи. Підхід Парето-оптимізації полягає в тому, що за генотипом g вдається розрахувати якісь окремі параметри — часткові критерії оптимальності й за ними відкидати такі особини g, які є заздалегідь неоптимальними [98]. Недоліком упорядкування за Парето є те, що неможливо визначити, що два довільні рішення $g_1.x$, $g_2.x$ розташовані близько в просторі рішень $\mathbb{X}$.

Цей недолік мінімізується в разі використання функції загальності (sharing function). Вона обчислюється на основі функції близькості

$$d = \text{dist}(g_1, g_2),$$

яка оцінює ступінь близькості двох рішень за ефективністю. Вона може мати різні метрики, наприклад, евклідову. Тоді функція загальності показує, якою мірою дві особини g_1, g_2 поділяють (share) одне й те ж саме місце у 𝕏:

$$\mathrm{Sh}\,(d, \sigma) = \begin{cases} 1 - (\mathrm{d}/\,\sigma)^s, & \text{при } 0 \le d < \sigma, \\ 0, & \text{при } \sigma \le d, \end{cases}$$

де σ — константа, яка задає радіус певної ділянки, що визначає кластер споріднених осіб, так звану нішу, $s \ge 1$ — коефіцієнт форми кластера. Отже, для особини g можна обчислити ступінь кластеризації (niche count):

$$m\,(g, Pop) = \sum_{i=1}^{|P|} \mathrm{Sh}\left(\mathrm{dist}\left(g, g_i \in Pop\right), \sigma\right).$$

Функція загальності слугує для розподілу особин по низці пагорбів у ландшафті функції Fitness, причому пагорб отримує частку популяції, яка є пропорційною його висоті [118]. Результат розподілення особин за нішами залежить від різниць висот пагорбів і тому залежить від складності функції оптимальності. Для зменшення цього впливу придатність корегується:

$$v(g.x) = m(g,\, Pop)\cdot \mathrm{Fitness}(g,\, Cmpf).$$

На рис. 2.9 показана функція загальності залежно від векторного критерію (f_1, f_2), де f_1 — апаратні витрати, а f_2 — затримка критичного шляху. Товстою лінією на рис. 2.9 показано границю, до якої дотикаються точки особин, які є оптимальними за Парето. Еліпсами позначені множини — ніші, які виділяються функцією загальності $\mathrm{Sh}(d, \sigma)$.

Отже, функція Fitness має виділяти найкращі особини за умови збереження різноманіття особин, щоби конвергенція алгоритму не призвела до локального оптимуму. Водночас функція загальності дає змогу виділяти ніші й зосереджувати в архіві *Arc* видатні особини з кількох ніш.

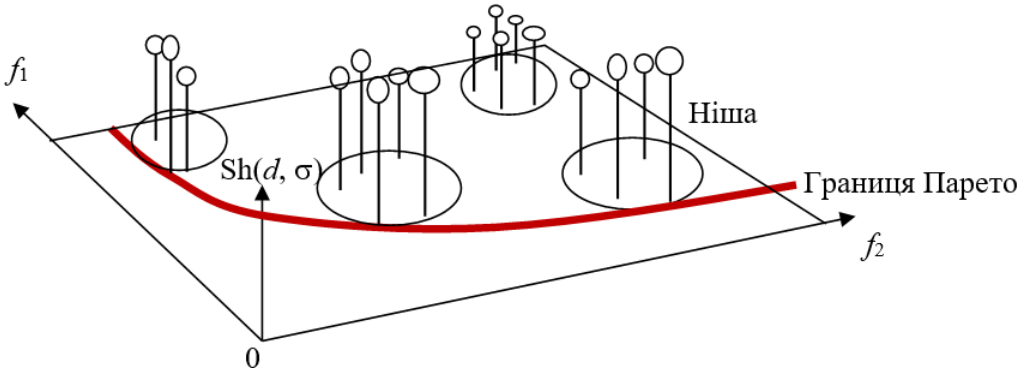

Рисунок 2.9 — Функція загальності залежно від векторного критерію (f_1, f_2)

Крім того, якщо в ніші збирається велика кількість особин із приблизно однаковою ефективністю, то це свідчить про конвергенцію до локального оптимуму. Щоби її зменшити, треба з певною ймовірністю викидати такі особини з покоління на етапі селекції.

Як показує теорія розширеного еволюційного синтезу [139], має сенс проводити макроеволюцію поколінь, коли добираються особини з генами, які представляють шаблони, що відповідають корисним ознакам фенотипу. Наприклад, якщо якийсь підграф ГСПД відображається в оптимальний апаратний модуль, то гени, що кодують цей підграф, представляють такий шаблон. Отже, треба більше добирати особин із такими шаблонами в елітну множину або суттєво зменшити ймовірність мутацій у генах такого шаблону.

Ця особливість досі не була використана у відомих генетичних алгоритмах, оскільки, як правило, у них не прослідковується прямий зв’язок між цифровим кодом генів та загальною їхньою користю для фенотипу.

2.2.8 Операція селекції

Алгоритми селекції мають великий вплив на ефективність еволюційних алгоритмів. Селекція вибирає найбільш ефективних осіб для розмноження

$$Popm = \text{Select}(Pop(t-1), v, ps).$$

Селекція може виконуватися детерміновано за ефективністю особини, що задається параметром *v* і/або з залученням випадковості. Крім того, у селекції бере участь також множина-архів *Arc*. Існують два види алгоритмів селекції: з підстановкою й без підстановки [194].

В алгоритмі добору без підстановки кожна особина з популяції *Pop* вибирається для розмноження лише один раз і тому вона трапляється в множині *Popm* не більш як один раз. Множина *Popm* за алгоритмом ***із*** підстановкою, може містити одну й ту ж саму особину кілька разів.

Зазвичай алгоритми вибору використовуються у варіанті із підстановкою. Якщо $|Pop| < ps$, то в множина *Popm* формується за алгоритмом без підстановки й вона має менше, ніж *ps* осіб.

Перед селекцією обчислюється функція ефективності особин. А алгоритми добору базують свої рішення виключно на значенні *v* особини. Багато алгоритмів добору працюють лише зі скалярною ефективністю *v*. При багатокритеріальній оптимізації алгоритми вибору можуть виконуватися послідовно — масив особин, які вибрані за першим критерієм, є вхідним масивом для формування множини ефективних особин за наступним критерієм і т. д. У деяких алгоритмах спочатку відкидаються явно найгірші особини, а потім виконується вибір за більш точними критеріями з-поміж решти особин.

Алгоритм із відсіканням (детермінована селекція або селекція з порогом) повертає $k < ps$ найкращих особин зі списку *Pop*. Потім ці елементи копіюються стільки разів, скільки потрібно, доки розмір *Pop* не досягне *ps*. Зазвичай використовуються значення $k = |Pop|/2$ або $k = |Pop|/3$.

Алгоритм із відсіканням використовується в еволюційних алгоритмах з $(\mu + \lambda)$ та (μ, λ)-стратегіями. Загалом, еволюційні алгоритми повинні мати критерій ефективності, який включає інформацію про різноманітність, щоби запобігти передчасній конвергенції до локального оптимуму. У роботі [137]

доведено, що усікання є оптимальною стратегією вибору для схрещування за умови використання правильного значення k.

Алгоритм із пропорційним добором є однією з перших схем добору [117]. За ним імовірність $P(g_1)$ того, що окремий індивід $g_1 \in Pop$ попадає в множину $Popm$, пропорційна його параметру $v(g_1.x)$. Ця величина визначається як

$$P\left(g_1\right) = \frac{v\left(g_1.x\right)}{\sum_{\forall g \in Pop} v(g.x)}. \tag{2.13}$$

З-поміж різновидів цього алгоритму найпоширеніший — метод Монте-Карло або метод рулетки, згідно з яким особини g_i розміщуються на колесі рулетки й ширина сектора, який займає особина, пропорційний її корисності $v(g_i.x)$. Де колесо випадково зупиняється — там перебуває особина, що попадає в множину $Popm$. Недоліком цього алгоритму є те, що результат сильно залежить від форми функції ефективності Fitness і він є найбільш ефективним, якщо ця функція є лінійною.

Алгоритм із турнірним добором є простою, але й однією з популярних та ефективних схем добору. При цьому k елементів вибирають у популяції Pop та порівнюють один з одним на турнірі за $v(g_i.x)$. Після цього переможець цього змагання входить до множини $Popm$. У турнір найчастіше входять $k = 2$ особин.

Для турніру учасники вибираються випадковим чином. Якщо припустити, що в множині $Popm$ міститься ps особин, то кожна особина бере участь у двох турнірах. При цьому не має значення форма й нормалізованість функції $v(g.x)$. Турнірний добір створює велику множину кращих особин популяції та майже не добирає жодної поганої особини [36, 149].

Алгоритм впорядкованого добору ґрунтується на тому, що ймовірність обрання особини пропорційна її позиції (рангу) у відсортованому списку осо-

бин. Чим менший номер k отримує особина в списку, тим вище буде ймовірність того, що особини, які переважають її за $v(g.x)$, попадуть у множину $Popm$. При цьому номер k перераховується в параметр q послаблення ймовірності

$$q = \frac{1}{1 - \frac{\log k}{\log ps}}.$$

Впорядкований добір зосереджується на малій групі вибраних особин, але навіть найгірші кандидати на рішення все ще мають ймовірність виживання, іншими словами, він призначає підвищену плодючість для небагатьох особин, але зберігає також і неефективні (в даному поколінні) генотипи.

Алгоритм рейтингового добору [42] ґрунтується на тому, що ймовірність вибору особини пропорційна позиції (рейтингу) у відсортованому списку популяції. Це робить плавною різницю між значеннями v.

Алгоритм VEGA (Vector Evaluated Genetic Algorithm) [197] застосовує вибір, який не використовує скалярну функцію v, але для кожного з параметрів $f_i \in F$ він формує підмножину множини $Popm$ розміром $ps/|F|$. Тому він застосовує алгоритм пропорційного вибору для кожного складника векторного критерію. Доведено, що такий алгоритм дає такі ж самі результати, як і алгоритм, у якому критерій $v(g.x)$ сформований як зважена сума критеріїв f_i [90].

2.2.9 Операція розмноження

Алгоритм оптимізації використовує інформацію, зібрану до даної ітерації t, для створення кандидатів рішення, що оцінюються на ітерації $t + 1$. В ЕА існують чотири основні операції розмноження:

Створення — створює новий генотип без усіляких батьків.

Дублювання — нагадує поділ клітин, даючи дві однакові особини з однієї.

Мутація — полягає в невеликій випадковій зміні в генотипі.

Схрещування, рекомбінація — комбінує два споріднені генотипи в нові генотипи.

Під час створення генотипу, наприклад, у першому поколінні, його параметри задаються випадковим чином так, аби особина належала простору рішень $g \in \mathbb{G}$.

Дублювання використовується у випадку ефективного рішення, властивості якого треба використати в наступних поколіннях:

$$g_n = \text{duplicate}(g),\ g \in \mathbb{G}.$$

Мутація виконується для створення нового генотипу через модифікацію наявного. Вона трапляється або випадковим чином, або детерміновано:

$$g_n = \text{mutate}(g),\ g \in \mathbb{G},\ g_n \in \mathbb{G}.$$

Рекомбінація виконує відображення $\mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}$. Вона може траплятись як випадково, так і детерміновано:

$$g_n = \text{recombine}(g_a, g_b);\ g_a, g_b \in \mathbb{G} \Rightarrow g_n \in \mathbb{G}.$$

Схрещування є вужчим терміном за рекомбінацію, бо виконується лише тоді, коли простір $\mathbb{G}$ є лінійним, як, наприклад, у просторового ГСПД.

Отже, розмноження — це створення нової популяції операціями створення, дублювання, мутації й рекомбінації. Усі чотири операції виконуються довільним чином, наприклад, мутація після рекомбінації.

2.2.10 Операції розмноження в генетичних алгоритмах

ГА є різновидом ЕА й тому він спадкує їхні методи, алгоритми та властивості. При створенні хромосоми g її гени задаються випадковими числами так, аби особина належала простору рішень $g \in \mathbb{G}$.

Мутація виконується через зміну частини (алеля) одиночного гена або алелів у кількох генах. Гени вибираються випадково. Зміна алеля полягає у

випадковому інвертуванні одиночних бітів. Якщо ген кодує певну лінійну величину, то він модифікується додаванням випадкового інкременту.

Перестановка (permutation) — це вид мутації, коли в генах однієї хромосоми виділяються два одноманітні алелі, які міняються місцями.

Як показує теорія розширеного еволюційного синтезу [139], найчастіше відбуваються незначні зміни в генах при мутації. Але можливі й великі, стрибкоподібні зміни. Причому ці зміни представляють певні шаблони, патерни, які характеризують корисні зміни в характеристиках фенотипу. Ця особливість досі не була використана у відомих генетичних алгоритмах.

Рекомбінація виконується як випадковий вибір однойменних алелів у хромосомах двох особин і їхній обмін. Можливий вибір кількох алелів для обміну. При класичному схрещуванні у двох хромосомах вибирається спільна точка схрещування на межі двох генів і частини обох хромосом до цієї точки міняються місцями. При багатоточковому схрещуванні вибирається кілька точок, які обмежують ділянки алелів, що міняються місцями.

2.2.11 Генетичне програмування

ГП за своїм другим визначенням є сукупністю ЕА, які продукують програми, алгоритми та подібні конструкції. Мета ГП полягає в тому, щоби знайти програму чи функціональну схему, яка дає такі ж самі результати або виявляє таку саму поведінку, що й заданий алгоритм відповідно до визначених ситуацій. Оскільки ГП безпосередньо мають справу із синтезом структур і програм, зокрема для ЦОС, далі методи ГП розглядаються більш детально.

На відміну від ГА, у методах ГП хромосома представляється не як лінійний список, а як структура у вигляді дерева або іншого графа. Причому в більшості методів це дерево відповідає структурі шуканого алгоритму. Це призводить до інших способів створення, мутації та рекомбінації, таких як вирощування (grow), обгортування (wrapping), інкапсуляція, підтягування

(lifting) дерев, заміна, тасування, редагування піддерев. Ген часто виглядає як набір кодонів, які представлені символами цільової мови програмування [35].

Прикладом особливого ГП є метод паралельного розподіленого генетичного програмування (Parallel Distributed Genetic Programming — PDGP) [185]. У ньому граф алгоритму представлений в n-вимірній решітці. Вершини графу позначені операціями. Еволюції підлягають мітки вершин та зв'язки між вершинами, за винятком обмежень. Крім того, на зв'язки накладаються ваги, які також еволюціонують. У такий спосіб "вирощуються" штучні нейронні мережі.

Для пошуку алгоритмів, що виконуються в системі з обмеженим паралелізмом, в [163] запропоновано метод прикладного декартового генетичного програмування (Embedded Cartesian Genetic Programming, ECGP). Програмування назване декартовим, бо вершини графа, як і в PDGP-методі, представлені у n=2-вимірній решітці, вершини, що стоять лівіше, передують тим, що стоять правіше. Генотип представлений рядком чисел, які кодують вершини решітки, тобто, решітка розміром nm має до nm генів у хромосомі. Числа показують з'єднання вершин і їхні функції. Додано три нові операції розмноження:

— компресія, яка випадково вибирає кілька вершин і замінює вершиною-модулем, яка виконує відповідні операції з цих вершин;

— розкриття, яке вибирає випадково вершину-модуль і замінює її на відповідну множину вершин із їхніми зв'язками;

— мутація модулів.

Цей метод нагадує метод згортання ГСПД, але він не стосується складання розкладу, яке повинне виконуватись окремо.

У [120] досліджено алгоритм ECGP типу (1+λ), який формує популяцію з елітних батьків та дітей у пропорції 1:λ, де найчастіше λ = 4, причому виконується лише мутація в окремих генах без схрещування. І якщо якийсь із

батьків не брав участі в мутації в даному поколінні, він буде мутований у наступному поколінні. У результаті показано, що такий алгоритм дає кращі рішення, ніж алгоритм зі схрещуваннями.

У фундаментальному дослідженні [234] доведено, що при відповідній стратегії генетичного програмування досягається знаходження оптимального рішення з певною ймовірністю. При цьому така стратегія повинна запобігати конвергенції еволюційного процесу до локального оптимуму.

2.2.12 Особливості запобігання конвергенції

У ГП для забезпечення генетичного різноманіття використовують особливі методи. Деякі з цих підходів ґрунтуються на відомих методиках, таких, як селекція з нішами або обмін функції придатності (fitness sharing) [97].

Парето-оптимізація з урахуванням віку (Age-Fitness, A/F) [199] використовує тривалість існування генів. Генотипний вік — це міра того, як довго в популяції еволюціонує найстаріший ген генотипу особини. Метод A/F використовує багатоцільовий підхід, щоб одночасно оптимізувати ефективність при мінімізації генотипного віку. Це дає змогу новому генетичному матеріалу поширюватися та потрапляти в різні регіони пошукового простору.

Алгоритм різноманітності генетичних маркерів (genetic marker diversity algorithm, GMD-GP) запобігає конвергенції дерев-хромосом [63]. Тут генетичний маркер — це верхній фрагмент дерева. Щільність генетичного маркера — це частка особин у популяції, яка його містить і грає роль міри різноманіття.

2.2.13 Генетичне програмування в САПР

Методи ГП використовуються в багатьох експериментальних САПР мікросхем та ПЛІС. У системі SPARCS (Synthesis and Partitioning for Adaptive Reconfigurable Computing Systems) розклад алгоритму шукається за допомогою лінійного програмування, а розподіл операцій на ресурси ПЛІС — за допомогою ГП [172]. У роботі [100] ГП використовується для підвищення

ефективності спискового планування, а в роботі [180] — для розпаралелювання циклів.

У роботі [57] через ГП шукається розклад виконання періодичного алгоритму на заданій множині ресурсів. При цьому за жадібним алгоритмом шукається початкове рішення, а остаточне — за допомогою ГА без мутацій.

У роботі [153] ГА використано для складання розкладу з мінімізацією площі та затримки, а в [154] — для розміщення блоків та їхнього з'єднання. У [102] описано ієрархічний ГА, у якому спочатку виконується вибір ресурсів, а потім — складання розкладу. Лише деякі алгоритми, наприклад, [70], враховують витрати на взаємозв'язки та мультиплексори в шуканій за ГА функціональній схемі, причому лише наприкінці оптимізації.

У роботі [183] описаний метод NSGA-II (non-dominated sorting genetic algorithm), у якому на основі графа алгоритму за допомогою ГА послідовно виконується побудова структури, складання розкладу, призначення на регістри. Тут ген представляє відношення оператор — ресурс. Причому оператору відповідають кілька ресурсів, куди він може потенційно бути призначений. Також у хромосомі закодована важливість ресурсів, яка використовується в списковому плануванні та інформація про зв'язки для побудови структури.

За методом, мутація виконується з імовірністю 10% і полягає в перепризначенні оператора ресурсу і випадковому вилученні ресурсу з гена. Схрещування має ймовірність 90% і полягає в обміні генами, які стосуються тих самих операцій. Елітна множина формується зі збереженням представників із кількох ніш за стратегією, що ймовірність вибору особини обернено пропорційна густині особин у регіоні (звідки назва NSGA). Функція придатності розраховується на основі регресійної моделі, побудованої за даними попередніх проєктів. Складання розкладу та побудова структури виконуються окремо.

Основною відмінністю NSGA-II є функція відстані купкування (crowding distance), яка характеризує унікальність рішення, так що до

наступного покоління потрапляють особи не тільки Парето-оптимальні, але й унікальні. При популяції 1000 осіб і 200 поколіннях оптимізація пристрою, що виконує дискретне косинусне перетворення, триває 485 хв. Тривалість оптимізації пропорційна довжині хромосоми: приблизно 28 хв. на ген.

У роботі [110] наведено удосконалений метод NSGA-II, критерій якості якого бере до уваги не тільки тривалість алгоритму в тактах і апаратні витрати (регістрів, суматорів, помножувачів), але й енергоспоживання, а кількість регістрів мінімізується алгоритмом лівого краю.

У роботі [131] розглядається метод ГП із геометричною семантикою (Geometric Semantic Genetic Programming, GSGP), який застосовано до декартового ГА, суть якого в тому, що додано геометричний семантичний оператор, який спрощує визначення функції придатності.

У [154] ГА використовуються для призначення вершин ГСПД на ресурси та побудови структури пристрою. У [102] реалізований ієрархічний ГА, у якому після ГА вибору ресурсів виконується ГА розкладу. У [39] застосовано підхід ГП, у якому рішення представлені у вигляді граматичного дерева, за яким на мові SFL (structured function description language) відтворюється опис пристрою. Але за таким деревом складно розрахувати функцію придатності.

У роботі [132] запропоновано кодувати в хромосомі рішення як список пріоритетів, що визначає, у якому порядку операції мають вибиратися алгоритмом спискового планування. Однак це призводить до оптимізації за одним параметром при неточній апроксимації інших параметрів.

У кількох роботах [88, 174] було запроваджено кодування хромосом на основі пар: операція — ресурс, де ця операція виконується. У деяких із цих підходів ГА може генерувати нездійсненні рішення, які треба виправити чи відкинути, витрачаючи час і ресурси для обчислень.

У літературі [101] було введено кілька методик спрощення функції придатності для прискорення ГА. Спрощена функція може бути або ендоген-

ною [216], або екзогенною [45, 123]. Успадкування функції придатності [216] — це один із найбільш перспективних ендогенних підходів, коли властивість певної частки осіб наступного покоління успадковується від батьків. У роботі [195] використана модель, яка ґрунтується на квадратичному наближенні функції придатності. У роботі [71] досліджена спадкована функція придатності як осереднена функція придатності батьків.

Пошук екзогенної наближеної функції придатності полягає в побудові спрощеної моделі реального пристрою та оцінюванні його параметрів [45]. У [159] пропонуються прості функції придатності, які не беруть до уваги помилки в схемі чи непрацездатність рішення. У [60] апаратні витрати оцінюються за допомогою лінійного регресійного підходу, який також може брати до уваги ефект логічної оптимізації. Але більшість запропонованих моделей не враховують взаємозв'язки між ресурсами, за винятком роботи [70], у якій розглянуто також оптимізацію мультиплексорів.

Метод ГА для синтезу конвеєрних пристроїв на основі ГПД, який використовує критерій зваженої суми об’єму апаратури й затримки, описаний в [132]. У хромосомі кодується пріоритет вершини для спискового планування. Недолік методу в тому, що такий критерій не показує нелінійність функції, і відкидає деякі Парето-оптимальні рішення. Для прискорення конвергенції використовують алгоритм диференційної еволюції [219]. При цьому мутація полягає в перетворенні трьох особин $\boldsymbol{v}_i$ (векторів) за формулою:

$$\boldsymbol{v} = \boldsymbol{v}_1 + F(\boldsymbol{v}_2 - \boldsymbol{v}_3),$$

де $F \in (0, 2)$. У [214] цей метод використовується для проєктування конвеєрних процесорів, причому конвергенція триває лише до 25 поколінь.

Розгляд еволюційних алгоритмів дає змогу зробити наступні висновки.

1. Еволюційні алгоритми є основою багатьох ефективних методів оптимізації складних систем. Серед них генетичне програмування знаходить застосування в багатьох САПР систем обчислювальної техніки.

2. Основні складнощі, що виникають при застосуванні генетичного програмування та чинники, що суттєво впливають на його ефективність — це трудомісткість функції придатності, вибір кодування генів та хромосоми, алгоритму селекції та операцій розмноження.

3. Для синтезу обчислювачів для ЦОС найбільш придатним методом генетичного програмування слід вважати метод прикладного декартового генетичного програмування, який має певні властивості методу просторового ГСПД, а також метод NSGA-II, який показав високу ефективність при синтезі конвеєрних пристроїв для ПЛІС.

2.3 Метод генетичного програмування просторового ГСПД

Вище було встановлено, що метод просторового ГСПД вимагає модернізації в напрямі покращення оптимізації за умови великих розмірів графу. Також встановлено, що таке покращення можливе й буде ефективним, якщо оптимізація виконується за допомогою генетичного програмування. Отже, у цьому підрозділі описується хід такої модернізації та формулюється метод генетичного програмування просторового ГСПД. Для такої модернізації треба, найперше, розробити спосіб кодування хромосоми, функцію придатності, алгоритми селекції та операцій розмноження.

2.3.1 Вибір способу кодування хромосоми

Метод проєктування конвеєрних обчислювачів, який описано в підрозділі 2.1, полягає в поданні ГСПД алгоритму в тривимірному просторі у вигляді конфігурації алгоритму $K_G = (K, D, A)$ та в розщепленні її на просторову конфігурацію $K_{GS} = (K_S, D_S, A)$ і конфігурацію подій

$K_{GT} = (K_T, D_T, A)$. За просторовою конфігурацією знаходять шукану структуру пристрою, а за конфігурацією подій — розклад виконання операторів.

Просторовий ГСПД характерний тим, що матриця K кодує певне допустиме рішення. Тому при синтезі конвеєрного пристрою за допомогою просторового ГСПД цілочисельна оптимізація розкладу полягає в побудові низки оптимізованих рішень та виборі найбільш переважного рішення K за заданим критерієм оптимальності. Причому цей критерій нескладно одержати за матрицями K, D, A. Оптимізовані рішення одержуються завдяки еквівалентним перетворенням просторового ГСПД, наприклад, за допомогою локальної перестановки векторів $\boldsymbol{K}_i$ у просторі та взаємної перестановки таких векторів з урахуванням умов коректності конфігурацій.

Метод просторового ГСПД є гарним претендентом для ефективної реалізації в ГП, бо за своїми властивостями він нагадує елементи методу прикладного декартового ГП.

Хромосома — це генотип, закодований бінарними кодами як вектор або запис кількох векторних полів, тобто, генів. У разі ГСПД, матриця K має всю інформацію, яка має міститись у хромосомі. Для традиційного представлення хромосоми треба стовпчики матриці K розгорнути в один рядок g.

Отже, хромосомою є матриця K, яка розгорнута в рядок. Кожен її стовпчик відповідає окремому гену. Крім того, вектори $\boldsymbol{K}_i$ можуть мати додаткові координати, які слугують для керування синтезом. Наприклад, окрема координата може вказувати, що даний ген кодує оператор вводу, часова і просторові координати якого є незмінними, тобто, він є інтроном і тому він не підлягає модифікації.

Наприклад, матриця K (2.12) кодується наступною хромосомою

g={2,1,8;3,4,9;4,3,2;1,2,4;1,2,6;1,2,8;2,1,10;3,4,11;4,3,4;1,2,5;1,2,7;2,1,9;4,3,0;2,1,6;3,4,7}.

Тут кожна трійка представляє окремий ген (k_i, s_i, t_i), причому k_i = 1, 2, 3, 4 означає вершини множення, додавання, вводу-виводу.

2.3.3 Операції відтворення

Генетичний алгоритм оптимізації використовує інформацію, зібрану до даної ітерації t, для створення кандидатів рішення, що оцінюються на ітерації $t + 1$, тобто, для відтворення генотипів. В еволюційних алгоритмах існують чотири основні операції відтворення: створення, яке створює новий генотип, дублювання, яке дає кілька копій однієї особини, мутація, що полягає в невеликій випадковій зміні в генотипі та рекомбінація (схрещування), яка комбінує два споріднені генотипи в нові генотипи.

При створенні генотипу просторового ГСПД, наприклад, у першій популяції, його параметри s_i, t_i генів задаються випадковим чином так, аби особина належала простору рішень $g \in \mathbb{G}$. При цьому особина є життєздатною, якщо виконуються умови (2.3) – (2.8). Також можливо одержати особини як результат виконання алгоритму цілочисельної оптимізації, який наведено в підрозділі 2.1.4. Принаймні, досить побудувати просторовий ГСПД із коректним досконалим кістяком. При цьому прискорюється конвергенція алгоритму. Але ймовірне попадання в локальний оптимум.

Дублювання використовується у випадку ефективного рішення, властивості якого треба використати в наступних популяціях. При цьому нова особина має таку ж саму хромосому $g \in \mathbb{G}$.

Мутація виконується для створення нового життєздатного генотипу через модифікацію наявного. При цьому в довільно вибраному гені (k_i, s_i, t_i) випадково змінюються параметри s_i, t_i у такий спосіб, щоби генотип залишався життєздатним (умови (2.3) – (2.8)). Під час мутації оператор, що відповідає вектору $\boldsymbol{K}_i$, одержує призначення на інший ПЕ s_i, або виконується в іншому такті t_i, або виконується в іншому ПЕ та іншому такті. Аналогічно

відбувається мутація групи генів (алелів). Якщо ГСПД є врівноваженим, то при мутації змінюється лише параметр s_i.

Перестановкою є така мутація, коли в однієї особини вибираються два гени одного типу k_i: (k_i, s_p, t_p) та (k_i, s_q, t_q) й обмінюються своїми параметрами, даючи гени (k_i, s_q, t_q) та (k_i, s_p, t_p). При цьому два однойменні оператори або обмінюються процесорними елементами, де вони виконуються, або міняються тактами виконання, якщо вони не залежать один від одного. В останньому випадку перевіряється лише умова (2.5) для дуг $\boldsymbol{D}_j$, які є суміжними до вершин цих операторів.

Рекомбінація виконує, власне, схрещування двох особин. Для рекомбінації вибираються два i-ті гени (k_i, s_{ai}, t_{ai}) та (k_i, s_{bi}, t_{bi}) з двох вибраних випадково генотипів g_a, g_b і параметри першого гена замінюються параметрами другого, даючи ген (k_i, s_{bi}, t_{bi}) нового генотипу g_n. Після цього перевіряються умови (2.3) – (2.8) і якщо вони не справджуються, то рекомбінація вважається не дійсною. Аналогічно виконується рекомбінація алелів.

Операції відтворення проілюстровані рис. 2.10. Отже, розмноження — це формування нової сукупності особин шляхом виконання операцій створення, дублювання, мутації та рекомбінації.

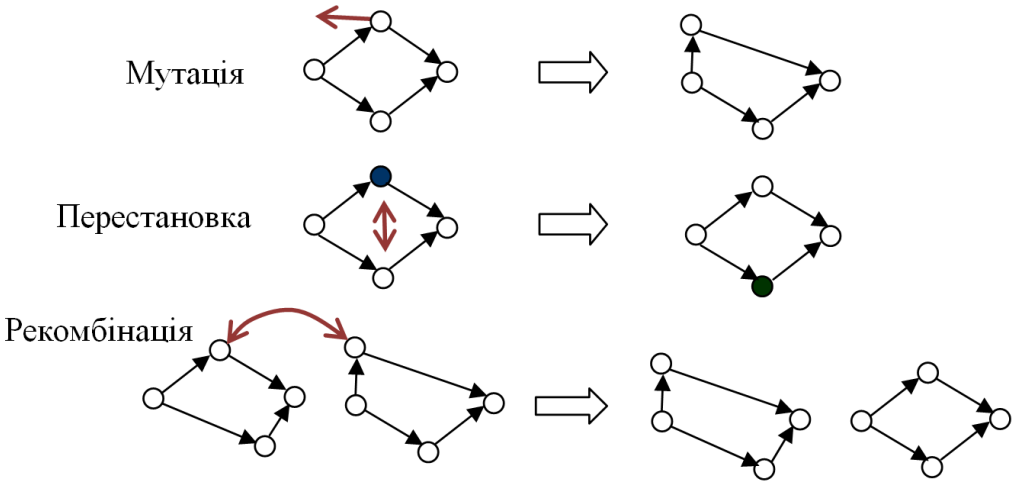

Рисунок 2.10 — Операції відтворення для просторового ГСПД

Усі чотири операції виконуються довільно, наприклад, мутація проводиться після рекомбінації. Але при відтворенні окремі гени, що утворюють інтрони, не піддаються змінам, оскільки вони не впливають на характеристики фенотипу. Це, наприклад, гени вершин введення-виведення.

2.3.4 Функція придатності

Найскладнішим завданням алгоритму ГП є реалізація функції придатності Fitness() в алгоритмі 1. При використанні методу просторового ГСПД ця функція стає набагато простішою, ніж в інших алгоритмах ГП, оскільки цей метод не потребує безпосередньої побудови структурної моделі шуканої ОС. Наприклад, апаратна складність оцінюється як зважена кількість ПЕ, яка для ПЕ k_i-го типу обчислюється як кількість множин вершин з однаковими координатами k_i та s_i.

Більш точна функція придатності враховує кількість входів мультиплексорів вершин ПЕ. У роботі [17] показано, що для урівноваженого ГСПД кількість входів N_{Mi} мультиплексорі i-ї вершини ПЕ є не більшою за кількість неоднакових векторів $\boldsymbol{D}_{i,j} = (s_j,\ p_j,\ t_{\mathrm{D}i,j})^{\mathrm{T}}$, які від'ємно інцидентні вершинам операторів, що відображаються в цю вершину ПЕ, без тих векторів, для яких $s_j = 0$, $p_j = 0$, тобто, векторів, які відповідають збереженню результату в цьому ПЕ.

Кількість входів мультиплексорів залежить від нумерації входів операторних вершин. Ці входи відображаються в однойменні входи ПЕ, на яких можуть стояти мультиплексори (рис. 2.3). На рис. 2.11 показаний приклад підграфа ГСПД, який відображається в ПЕ.

В одному випадку нумерації входів цього прикладу одержується ПЕ з двома двовходовими мультиплексорами, які позначені товстими рисочками, а в іншому випадку — лише один мультиплексор. Отже, нумерація входів операторної вершини має значення для оптимізації ОС. Для виконання такої

оптимізації треба додати додаткову координату до векторів $\boldsymbol{D}_j$, яка дорівнює номеру входу операторної вершини, у яку входить цей вектор.

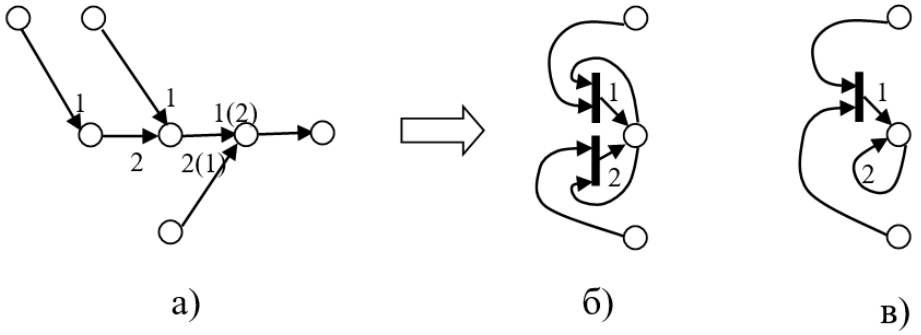

Рисунок 2.11 — Приклад відображення КА (а) у КС при одній нумерації входів вершини (б) та іншій нумерації входів (в)

Але, якщо входи оператора є нерівноправними, то оптимізація за рахунок тасування входів операторів стає неможливою. Це, наприклад, такі оператори, як множення на константу, операції з операндами з різною розрядністю.

2.3.5 Функція придатності при реалізації алгоритмів ЦОС у ПЛІС

Отже, для визначення складності особини, тобто, ОС, треба за кодом хромосоми визначити n_A — кількість суматорів, n_M — кількість блоків множення, n_R — кількість регістрів, n_X — кількість входів мультиплексорів. На основі них розраховується інтегральний критерій апаратної складності

$$Q_S = C_R n_R + C_A n_A + C_M n_M + C_X n_X , \qquad (2.14)$$

де C_R, C_A, C_M, C_X — коефіцієнти складності регістра, суматора, помножувача, входу мультиплексора, відповідно. У роботі [16] з урахуванням складності сучасних ПЛІС фірми Xilinx [217, 233] показано, що при виконанні алгоритмів ЦОС у ПЛІС можна вважати, що складність двох-тривходового суматора відповідає складності двох регістрів, складність одного входу мультиплексора відповідає 0,57 складності регістра і складність блоку множення удвадцятеро

більша за складність регістра. З урахуванням цього, формула (2.14) виглядає як наступна

$$Q_S = n_R + 2n_A + 20n_M + 0{,}57n_X, \qquad (2.15)$$

Ланцюгом затримки є маршрут від однієї вершини ГСПД до іншої, у який входять кілька векторів-дуг $\boldsymbol{D}_i = (k_i, s_i, 0)^T$ і один вектор $\boldsymbol{D}_i = (k_i, s_i, t_i)^T$, який проходить через кілька операторних вершин, де $t_i > 0$, причому в маршрут входять лише ті вершини, у які входять дуги $\boldsymbol{D}_i$ (дотикаються кінцем). У функціональній схемі ОС цей маршрут проходитиме від виходу одного регістра до входу іншого й може бути критичним шляхом.

Критерій швидкодії пристрою буде такий:

$$Q_T = n'_A + c_{TM} n'_M + c_{TX} n'_X, \qquad (2.16)$$

де n'_A — кількість суматорів; n'_M — кількість блоків множення; n'_X — кількість мультиплексорів, що стоять у ланцюгу максимальної довжини, який сполучає вихід одного регістра і вхід іншого регістра функціональної схеми ОС, c_{TM}, c_{TX} — відношення затримки блоку множення й мультиплексора до затримки суматора, відповідно. Тут одинична затримка оцінюється як затримка суматора з урахуванням середніх затримок у лініях зв'язку. Для ПЛІС фірми Xilinx $c_{TM} = 2{,}1$; $c_{TX} = 0{,}55$.

Відповідно, ланцюг максимальної довжини шукається серед усіх ланцюгів затримки, для яких обчислюється функція Q_T (2.16), яка є максимальною для цього ланцюга. Найчастіше в такі ланцюги входить максимальна кількість послідовно з'єднаних операторних вершин, особливо вершин множення.

Якщо розглядається алгоритм ЦОС, то його період найчастіше дорівнює періоду дискретизації вхідних даних. Так що за один такий період L пристрій, наприклад, фільтр, розраховує алгоритм, що стосується одного нового даного

й одного результату. У пристроях із максимальним ступенем розпаралелювання цей період дорівнює періоду синхросигналу, тобто одному такту [51].

У реальних проєктах не обов'язково, щоби період дискретизації дорівнював періоду синхросигналу. Наприклад, фільтри на ПЛІС можуть працювати з тактовими частотами до 300–1000 МГц. Водночас, у багатьох застосуваннях, скажімо, у системах обробки відеосигналів, достатньо частоти дискретизації 1–50 МГц. Таким чином, виконавши фільтр із високою частотою синхросигналу й таким, що виконує період обчислень за $L > 1$ тактів, можна майже в L разів зменшити апаратні витрати.

Хоча два блоки, що обробляють сигнал із тією самою частотою дискретизації, але з різним періодом L, виконують одну й ту саму кількість операцій, блок із більшим L має не тільки менші апаратні витрати, але й менше енергоспоживання. Це пояснюється тим, що пристрій, що займає меншу площу на кристалі, має меншу середню довжину ліній зв'язку і, отже, менше розсіювання енергії на них.

Отож, з урахуванням періоду виконання алгоритму L, треба формулу (2.16) помножити на значення L:

$$Q_T = L\left(n'_A + c_{TM}\, n'_M + c_{TX}\, n'_X\right). \qquad (2.17)$$

Але при пошуку оптимального рішення можна використати й (2.16). Річ у тім, що оптимізація пристроїв найчастіше виконується для фіксованого значення L. У системах обробки даних зі зворотними зв'язками не завжди можна виконати умову $L = 1$, через те, що згідно з (2.16) період синхросигналу буде занадто довгим. Тому оптимальне рішення шукається послідовно для $L = 2, 3, \ldots$, поки не будуть одержані мінімальні апаратні витрати, а період виконання алгоритму не перевищуватиме межу заданого періоду дискретизації. При цьому на кожному кроці значення L фіксується й тому критерій (2.16) справджується.

Якщо маємо урівноважений ГСПД, у якому всі вектори-дуги, крім векторів затримки $\boldsymbol{D}_{Dj}$, дорівнюють $\boldsymbol{D}_i = (k_i, s_i, 1)^T$, тобто, у ланцюг затримки входить лише одна вершина, то часовий критерій значно спрощується:

$$Q_T = L\left(\alpha + \mu c_{TM} + \xi c_{TX}\right), \tag{2.18}$$

де $\alpha, \mu, \xi \in \{0,1\}$, $\alpha \oplus \mu = 1$, бо в критичному шляху стоятиме або помножувач (якщо в алгоритмі є множення), або суматор, або один із цих елементів із мультиплексорами на входах.

Функції Q_S і Q_T разом представляють векторну функцію придатності й тому можуть слугувати для пошуку Парето-оптимального рішення (див. рис. 2.9). Остаточна — інтегральна функція придатності — розраховується за формулою

$$v = Q_S \cdot Q_T . \tag{2.19}$$

2.3.6 Операція селекції

Операція селекції має мету, яка є протирічною. З одного боку, вона повинна вибирати найкращі особини для розмноження. З іншого боку, вона має забезпечити генне різноманіття популяції задля запобігання деградації ходу еволюції в бік несуттєвого локального оптимуму. Також ця операція не має бути занадто складною, щоби не переважувати й так складний еволюційний алгоритм.

Елітний еволюційний алгоритм забезпечує те, що принаймні одна копія найкращих особин поточного покоління передається наступному поколінню. Перевага такого алгоритму полягає в тому, що його конвергенція гарантована, але є ризик сходження до локального оптимуму. Цей ризик регулюється часткою C_E елітних особин у множині особин популяції. Ця частка вибирається з діапазону (0,01–0,2).

Алгоритм із пропорційним добором обчислює ймовірність $P(g_1)$ того, що окремий індивід $g_1 \in Pop$ потрапить у множину *Popm,* і ця ймовірність пропорційна параметру придатності $v(g_1.x)$, який обчислюється за формулою (2.19). Сама ймовірність може бути обчислена за формулою (2.13).

Серед різновидів заслуговує на увагу метод Монте-Карло (метод рулетки). Ефективність цього алгоритму забезпечується тим, що функція придатності (2.19) є лінійною відносно апаратних витрат.

Пропонується застосувати в методі синтезу ОС функцію селекції Qvalue. Згідно з нею, ймовірність того, що *i*-а особина буде вибрана в нове покоління, розраховується за формулою

$$P_i = \frac{Q_{max} - Q_i}{\sum_i (Q_{max} - Q_i)}, \qquad (2.20)$$

де Q_i — значення складності *i*-ї особини, $Q_{\max}$ — максимальне значення складності в поколінні, тобто, найгірша придатність. Значення складності може бути розраховане за формулою (2.19).

Алгоритм селекції є найбільш відповідальним у методі ГП, тому що від нього найбільше залежить те, чи знайдеться найбільш оптимізоване рішення в потенційно нескінченному просторі рішень. Тому він потребує додаткових досліджень. Зокрема, є слушним дослідити алгоритми селекції за напрямами:

— селекція ніш (рис. 2.9), у яких зберігається генетичне різноманіття особин, які можуть дати в разі схрещування більш оптимізовані рішення;

— селекція для макроеволюції, коли зберігаються під час мутації та схрещування або підставляються під час мутації сформовані шаблонні гени.

Шаблонні гени — це множина генів, яка відображається в оптимальну підструктуру ОС. Шаблонні гени з’являються як результат вдалої еволюції або при ручному проєктуванні певних частин проєкту. Наприклад, це можуть бути модулі буферної пам’яті, фільтри з оптимальною структурою, стандартні

блоки, як, наприклад, помножувач з акумулятором, блок множення комплексних чисел. Шаблонним геном кодується також ланцюжки вершин затримки, які відображаються в регістрову пам'ять або буфер типу FIFO.

2.3.7 Реалізація методу генетичного програмування просторового ГСПД

Метод генетичного програмування просторового ГСПД полягає в представленні ГСПД у багатовимірному просторі з урахуванням умов коректності відповідних КА і КС, а також умов конвеєрного виконання алгоритму й описі одержаного просторового ГСПД мовою опису апаратури, такою, як VHDL. Від відомого методу просторового ГСПД метод відрізняється тим, що над просторовим ГСПД виконується оптимізація методом ГП. Причому як хромосома g використовується список координат, одержаний із матриці координат K векторів-вершин ГСПД, у якому координати вершини графа представляють ген. Еволюційний процес виконується у два етапи. На першому етапі береться покоління особин із випадковим розміщенням операторних вершин у просторі й над ним виконується ГП, результатом якого є покоління оптимізованих особин, які кодують неурівноважені ГСПД. На другому етапі ГСПД особин покоління урівноважуються і знову виконується ГП. З-поміж особин останнього покоління вибирається найкраща особина і вона описується мовою VHDL.

Метод застосовується з наступним алгоритмом.

Алгоритм 2.

Вхідні дані: ГСПД $C_A = (K, D, A)$;

Q_S, Q_T — критерії якості рішення;

ni — кількість поколінь;

ps — розмір популяції;

ce — частка елітної множини.

Змінні: t — лічильник поколінь;

Pop — популяція, яка складається з *g*;

Popm — частина популяції для розмноження;

v — величина придатності.

Результат — набір найкращих рішень.

```
begin
    Pop(0) = Initialize(ps);
    for (t = 1; t < ni; t++){
        v = Fitness(Pop(t – 1), Q_S, Q_T);
        Popm = Select(Pop(t – 1), v, ps,ce);
        Pop(t) = Reproduce1(Popm,ps,ce);
    }
    Pop(0) = Balancing(Pop(t));
    for (t = 1; t < ni; t++){
        t = t + 1;
        v = Fitness(Pop(t – 1), Q_S, Q_T);
        Popm = Select(Pop(t – 1), v, ps,ce);
        Pop(t) = Reproduce2(Popm,ps,ce);
    }
    return Phenotype(Optimal(Pop(t));
end;
```

Функція Initialize(*ps*) генерує початкову популяцію з номером $t = 0$, яка складається з *ps* особин, причому генотипи *g* формуються з векторів $\boldsymbol{K}_i = (k_i, s_i, t_i)$, у яких координати s_i, t_i є випадковими й задовольняють умови (2.3) – (2.8).

Функція Fitness(*Pop*(*t* – 1), Q_S, Q_T) кожній особині в популяції надає число *v*, яке пропорційне якості цієї особині й обчислюється за формулою (2.19).

Функція Select(*Pop*(*t* – 1), *v, ps,ce*) вибирає множину *ps* осіб *Popm*, які придатні для розмноження. Серед них *ps·ce* особин формують елітну множину *Popm.e*, а серед решти *ps*(1 – *ce*) особин шукаються особини для розмноження за алгоритмом рулетки або за функцію селекції Qvalue.

Функція Reproduce1(*Popm*) генерує нову популяцію. Спочатку виконується схрещування особин, які вибрані за алгоритмом рулетки. При кожному схрещуванні одержуються дві життєздатні особини, з-поміж яких вибирається краща за функцією Fitness. Схрещування повторюється до *ps* разів.

Потім у випадково вибраних особин виконується перестановка та мутація. При мутації в гені (k_i, s_i, t_i) випадковим чином змінюються параметри s_i, t_i, причому параметр t_i змінюється в певному діапазоні — діапазоні можливих тактів виконання відповідного оператора ГСПД. Якщо особина після перестановки чи мутації є нежиттєздатною за умовами (2.3) – (2.8), то вона відкидається й ці операції повторюються ще певну кількість разів. Якщо після перестановок та мутацій кількість особин є меншою за *ps*(1 – *ce*), то решта особин вибирається з-поміж найкращих зі множини *Popm.* Насамкінець, формується елітна множина *ps·ce* особин, у яку входить частка особин з елітної множини *Popm.e* та кілька найкращих особин після розмноження.

Після еволюції *ni* поколінь перший етап оптимізації закінчується. Тоді за функцією Balancing(*Pop*(*t*)) виконується балансування ГСПД особин за правилами, що описані в підрозділі 2.1.3.

Другий етап оптимізації виконується над поколінням збалансованих особин. Він відрізняється від першого етапу лише функцією Reproduce2(*Popm,ps,ce*), у якій при мутації змінюється лише параметр s_i, а після перестановки завжди одержується життєздатна особина.

Функції Optimal(*Pop*) та Phenotype використовуються для виділення генотипу найліпшої особини (або кількох особин) та побудови її фенотипу. При цьому за хромосомою *g* відтворюється КА, яка описується мовою VHDL

так, як вказано в підрозділі 2.1.4. Синтезована конвеєрна ОС компілюється відповідним компілятором-синтезатором, результатом чого є остаточні параметри апаратних витрат та максимальної тактової частоти.

2.4 Висновки до другого розділу

1. Метод синтезу конвеєрних ОС для ЦОС на основі просторового ГСПД є ефективним і може бути застосований із традиційними методами цілочисельної оптимізації для ГСПД невеликих розмірів.

2. Проаналізовані еволюційні методи оптимізації і встановлено, що методи генетичного програмування можуть бути застосовані для оптимізації просторових ГСПД довільних розмірів.

3. Вперше розроблено метод генетичного програмування просторового ГСПД, який оснований на тому, що над просторовим ГСПД виконується оптимізація методом ГП. Водночас запропоновані представлення хромосоми, функції ініціалізації особин, їхньої придатності, селекції та репродукції, а також двоетапний алгоритм оптимізації. Метод відрізняється від відомих тим, що оптимізоване рішення одержується формально й гарантовано, і завдяки цьому метод може бути впроваджений у САПР спеціалізованих комп'ютерних засобів.

4. Сформульовано критерії оптимізації для елементної бази, які можуть бути використані як функція придатності при оптимізації за методом ГП.

У наступному розділі розглядатиметься опис експериментальної системи, яка реалізує запропонований метод.

РОЗДІЛ 3. ЗАСІБ ПРОЄКТУВАННЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ ДЛЯ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ

3.1 Початкові дані для розроблення засобів проєктування ОС для ЦОС

У попередньому розділі було запропоновано новий метод проєктування ОС для ЦОС, який може бути ефективнішим за інші подібні методи. Для доведення цього треба розробити програмний комплекс, який, з одного боку, ефективно виконує алгоритм цього методу, з іншого боку, є зручним у користуванні та має перспективи впровадження в широку практику. Це протирічна мета, бо для її досягнення матзабезпечення має виконувати завдання комбінаторної оптимізації якнайшвидше і водночас мають бути реалізовані зручний інтерфейс, можливість стикування з наявними засобами САПР ПЛІС, НВІС.

Останнім часом багато компаній, що поширюють САПР мікросхем, пропонують їхній сервіс, який розміщено в хмарному середовищі [231]. З-поміж них Aldec, Mentor, Synopsys (логічні симулятори), Tabula (синтез ПЛІС), Cadence (керування проєктом) уже мають успіх. Розроблення модулів для ЦОС у хмарному середовищі має низку переваг: зменшення вартості, наявність усіх необхідних інструментів, захищеність даних тощо. Але вказані та інші засоби проєктування в хмарах виконують лише збереження бібліотеки готових модулів, верифікацію, розміщення, трасування та супроводження проєкту, а не власне структурний синтез. Отже, є слушним розробляти такий засіб проєктування ОС для ЦОС, який би знайшов впровадження в просторі вебзастосунків та хмарному середовищі.

Основними чинниками затримок у хмарних обчисленнях та вебзастосунках є затримки транзакцій у комунікаційних каналах та тривалість виконання складних завдань. Для мінімізації цих затримок слушно розрізняти

два види таких завдань. Завдання першого виду забезпечують ввід-вивід початкових даних та результатів проєктування і ґрунтуються на інтерактивній взаємодії з розробником-користувачем. Тому ці завдання треба виконувати на клієнтському боці системи проєктування. Завдання іншого виду виконують синтез модулів ОС, зокрема, за запропонованим методом за складним багато-ітераційним еволюційним алгоритмом і потребують високої обчислювальної потужності. Обміни даними між цими завданнями мають затримку, яка залежить від роботи комунікаційних каналів і яка є значно більшою за швидкість відгуку системи вводу-виводу на клієнтському боці розробника.

У засобах проєктування модулів ОС для ЦОС слід використати розроблений метод генетичного програмування ГСПД. Для організації проєктування модулів ОС у хмарному середовищі початкові, проміжні та результівні дані пропонується представляти у форматі XML (extensible markup language). Цей формат забезпечує стандартизоване представлення як алгоритмів, так і результівних структур, а також є натуральним для обробки в хмарному середовищі. Інтернет-браузери на клієнтському боці та процесори сервера, а також бібліотеки мов програмування забезпечують ефективний ввід-вивід XML-файлів. Багато САПР мікросхем використовують формат XML для зберігання та поширення даних, що обробляються [47].

Формат даних масштабованої векторної графіки (scaled vector graphic, SVG), який ґрунтується на форматі XML, слід вибрати для кодування графічних об’єктів у системі, що розробляється, зокрема, для графічного представлення ГСПД. Такий формат є зручним як для вебзастосувань загалом, так і для результівних структур зокрема. Існує досить широкий набір вільних засобів та бібліотек для обробки даних у цьому форматі. У роботі [76] показана ефективність застосування формату SVG в системах проєктування засобів електроніки, які використовуються в хмарному середовищі.

Мову Java також слід вибрати як робочу мову програмування функцій САПР. Ця мова є натуральною як для клієнтської, так і для серверної частини. Вона забезпечує також паралельні обчислення в багатопроцесорній системі хмарного середовища, яка прискорює виконання синтезу. Також існують вільні бібліотеки Java-функцій, які обробляють дані як в XML, так і в SVG-форматі.

Нова САПР, яка призначена для відображення ГСПД у структуру ОС для ЦОС, повинна мати змогу працювати на клієнтському боці з використанням мови Java. Зокрема, вона повинна мати програму для ручного графічного вводу ГСПД. При цьому, як алгоритм, так і результівна структура будуть зберігатися в XML-файлах. Також у цьому форматі зберігатимуться бібліотечні компоненти. Ці компоненти мають розроблятися в редакторі компонентів, який забезпечує їхній опис мовою VHDL з різними форматами даних та заданням налаштувальних констант. При цьому допускається представлення сигналів ЦОС у таких форматах даних, як цілі, числа з фіксованою та рухомою комою.

Нова система має транслювати XML-опис як алгоритму, так і результівної структури в опис мовою VHDL. Такий опис може моделюватись у VHDL-симуляторі, а також компілюватись у схему на рівні вентилів одним із поширених компіляторів-синтезаторів. Також система має забезпечувати ручну оптимізацію ГСПД, багатопараметричне керування ходом оптимізації. При цьому автоматично застосовується низка обмежень, що накладаються на ГСПД, які забезпечують синтез працездатного конвеєрного модуля, що виконує алгоритм із заданим періодом L.

3.2 Склад і алгоритми роботи засобу проєктування обчислювальних систем для ЦОС

3.2.1 Елементи системи SDFCAD

Система SDFCAD є комплексом програм для вводу ГСПД, його редагування, оптимізації та відображення в конвеєрний обчислювальний пристрій. Система складається з таких частин:

— візуальний інтерфейс користувача;

— модуль, що реалізує еволюційний алгоритм (ГП);

— бібліотека функцій для роботи з текстами на XML;

— бібліотека компонентів (моделей вершин).

3.2.2 Бібліотека функцій для роботи з текстами на XML

Проєкт в SDFCAD та його частини, бібліотека компонентів, файл налаштувань зберігаються у XML-форматі для універсального подання інформації при передаванні даних між компонентами системи та подальшого можливого стикування з поширеними засобами САПР. Функції для роботи з текстами на XML призначені для граматичного розбору XML-файлів та перетворення зчитаної інформації у внутрішнє представлення (DOM-модель, document object model).

Наприклад, при відкритті файлу з проєктом створюється новий зчитувач графу XMLFCGraphReader, з нього отримується перелік компонентів у вигляді DOM-елементів (елементів XML-документа) і так само перелік з’єднань між компонентами. Кожен компонент з DOM-елемента перетворюється у формат внутрішнього програмного подання бібліотечних компонентів LibraryFComponent. Уже в цьому форматі компоненти-вершини та їхні ребра-зв’язки перетворюються в графічне представлення, яке вимальовується у відповідній вкладці у вікні візуального інтерфейсу, розташовуючись відповідно до їхніх координат.

Так само, при збереженні проєкту для поточної вкладки викликається процедура запису графа в XML-файл. Ця процедура створює XML-документ і потім записує його у файл засобами стандартної бібліотеки. А для того, щоби записати в документ інформацію про компоненти та з'єднання між ними, для кожного компонента викликається його власна функція запису в XML-документ buildXMLDocument, і для лінкера, відповідно, теж викликається його власна процедура buildXMLDocument.

Окремі функції читають опис елементів у SVG-форматі та вимальовують згідно з ним вершини ГСПД на екрані комп'ютера.

3.2.3 Бібліотека компонентів

Як роз'яснювалось у першому розділі, ГСПД являє собою певну модель паралельного обчислювача, архітектуру, яка виконує алгоритм, що розглядається. Вершини ГСПД моделюють виконання елементарних операцій, а дуги — потоки даних між вершинами. Згідно з методом просторового ГСПД, модель зберігає також інформацію (просторове положення вершин), яка дає змогу однозначно побудувати структуру та розклад конкретного конвеєрного обчислювача. Отже, опис поведінки вершин ГСПД дає змогу перетворити абстрактний ГСПД з акторами та токенами в конкретну модель обчислювача з реальними операціями та даними, на основі якої описується конкретний варіант реального пристрою.

У бібліотеці компонентів зберігаються описи моделей вершин ГСПД, які розглядаються. Така модель може бути підготовлена заздалегідь або складена користувачем. Наприклад, модель вершини додавання вміщує в себе такі дані, як координати вершини в просторі, тобто, вектор $\boldsymbol{K}_i$, опис поведінки додавання мовою VHDL та опис графічного зображення мовою SVG.

VHDL-модель вершини представлена своїми об'єктом та архітектурою. Об'єкт вміщує в собі вхідні та вихідні порти вершини та налаштувальні константи, такі, як розрядності портів, затримки логічних схем. Крім портів,

які відповідають входам-виходам вершини, вказуються такі керуючі порти, як вхід синхросигналу та початкового встановлення. Основний тип портів — це std_logic та std_logic_vector, можливі типи signed, fixed, real.

Архітектура вміщує в собі опис поведінки вершини, причому наявність внутрішніх регістрів не обов'язкова. Загалом, архітектура описує певну функціональну схему, таку, як на рис. 2.3, у якій мультиплексорів, як правило, немає.

Для вводу та редагування бібліотечного компонента використовується окрема підпрограма візуального інтерфейсу користувача. Як правило, розрядності портів є налаштовуваними під час встановлення компонента-вершини в ГСПД, який проєктується.

Якщо ГСПД уже сформований, то підпрограма візуального інтерфейсу користувача може скомпілювати моделі всіх вершин ГСПД у структурну VHDL-модель, яка може бути змодельована в довільному VHDL-симуляторі для періоду $L = 1$ такт, щоби перевірити правильність введеного алгоритму. Також після того, як процес оптимізації буде закінчений, на основі моделей вершин складається остаточна модель синтезованого пристрою, яка функціонує з заданим періодом L.

3.2.4 Візуальний інтерфейс користувача

Візуальний інтерфейс користувача призначений для взаємодії користувача з програмним комплексом SDFCAD, тобто, для вводу початкових ГСПД, їхнього графічного редагування, введення й корекції критеріїв оптимізації та інших параметрів проєкту, запуску оптимізації за допомогою ГП, відображення і виводу результатів проєктування.

У лівій частині головного вікна інтерфейсу є панель, де відображається перелік бібліотечних компонентів. Основну частину вікна займає поле, де можна редагувати графи. Згори панель із кнопками перемикання режимів

курсора, масштабування зображення графа та виклику оптимізації, а також меню (рис. 3.1).

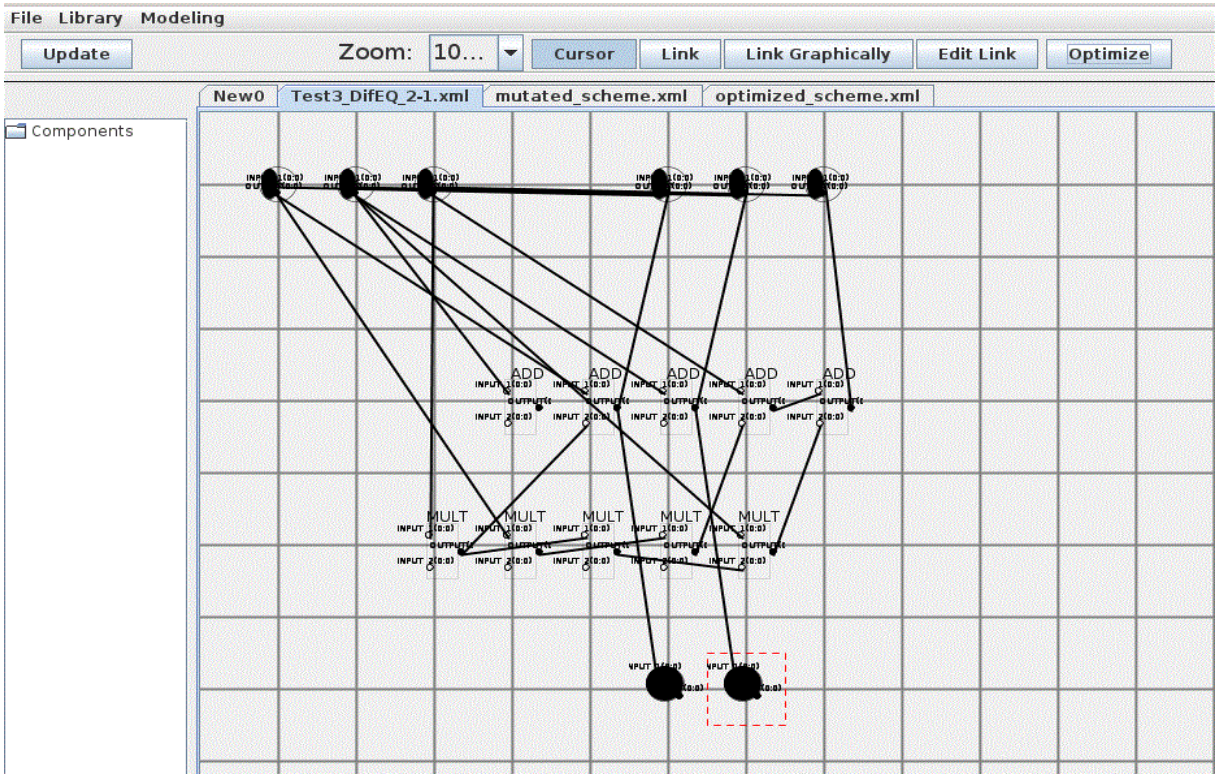

Рисунок 3.1 — Головне вікно інтерфейсу користувача з відображенням ГСПД, як на рис. 2.5

Кожен граф відкривається в окремій вкладці, кілька вкладок можуть між собою перемикатися. Новий граф створюється в порожній вкладці. З панелі з бібліотечними компонентами необхідні компоненти перетягуються на поле вкладки й розташовуються на потрібних позиціях за допомогою перетягування.

У меню є опції створення нової порожньої вкладки, відкриття для редагування графа вже створеного XML-файлу, збереження графа з поточної вкладки в XML-файл, закриття поточної вкладки, виходу з програми, додавання нового компонента до бібліотеки, редагування вже наявного бібліотечного компонента.

При натисненні кнопки “Optimization” на верхній панелі, відкривається діалогове вікно, де користувач задає параметри оптимізації, після чого запускається процес оптимізації. У вікні задання параметрів можна задати:

— чи треба автоматично вставляти регістри, чи ні;

— чи треба виконувати еволюційний алгоритм;

— чи треба перед еволюцією зробити мутацію заданого графа (випадково розташувати його вершини);

— максимальну кількість поколінь (ціле число, у діапазоні 0–1000);

— розмір популяції, тобто, кількість особин у кожному поколінні (ціле число, у діапазоні 2–1000);

— спосіб вибору батьків у еволюційному алгоритмі: спосіб QValue; спосіб Roulette;

— чи застосовувати ніші в способі Roulette, чи ні;

— розмір еліти (ціле невід’ємне число);

— інші параметри.

Водночас задано правило, що розмір еліти не може перевищувати половину популяції. В експериментах здебільшого розмір еліти брався як 10% від розміру популяції.

Спочатку для графа з поточної вкладки створюється його подання для еволюційного алгоритму — особина.

Якщо прапорець “чи мутуватиме заданий граф” = true, то особина мутує. Це означає, що вершини графа будуть розташовані випадковим чином, і навіть якщо користувач задав вдале розташування вершин графа, ця інформація не буде збережена. Під час цього для особини викликається процедура мутації з імовірністю 100%, після чого створюється XML-записувач XMLWriter, який робить з особини XML-документ за допомогою функції buildGraphDocument.

Після мутації особина перетворюється в XML-файл і цей файл відкривається в новій вкладці в графічному інтерфейсі. Після цього виконується

власне оптимізація тієї самої мутованої особини. Якщо ж прапорець “чи мутуватиме заданий граф” = false, то просто виконується оптимізація особини, без мутації. Це означає, що задане рішення, якщо воно вдале, у процесі еволюції, швидше за все, збережеться.

Сама оптимізація складається з кількох етапів.

1. Якщо в параметрах вказано, що має виконуватися еволюційний алгоритм, на основі даної особини створюється нова популяція. У цій популяції виконується еволюційний алгоритм. Її результатом є одна-єдина особина. Для цього обирається найкраща особина з останнього покоління — тобто, така особина, у якої значення критерію найменше.

2. Якщо в параметрах вказано, що автоматичне вставлення регістрів відбувається, то ГСПД урівноважується за допомогою функції addRegisters, а потім мінімізуються регістри функцією mergeRegisters.

3. Повторюється п. 1.

4. Оновлюються дані XML за допомогою процедур addNewComponents, updateComponentCoordinates. Створюється XML-записувач, який формує XML-документ.

У результаті виконання оптимізації одержується XML-файл просторового ГСПД рішення. Цей файл автоматично відкривається в новій вкладці після завершення оптимізації.

3.3 Модуль еволюційного алгоритму

Модуль еволюційного алгоритму фактично реалізує розроблений метод ГП просторового ГСПД, який описано в другому розділі. Тому далі він викладений детально.

3.3.1 Об’єкти, які застосовані в еволюційному алгоритмі

Тип порта — наприклад, вхід1, вхід2, вихід (для 2-входових елементів).

Порт — тип порта разом із переліком з’єднань, пов’язаних із цим портом.

Вершина графа — це тип елемента, який вона позначає, разом зі списком його портів та номером цієї вершини в графі.

З’єднання — позначає ребро графа і вміщує номер вершини, номер порта, звідки виходить, номер вершини, номер порта, куди воно заходить.

Ген — це вершина графа, до якої додано координати простору та часу.

Хромосома — масив генів.

3.3.2 Створення популяції

Популяція створюється на основі таких параметрів: коренева особина (заданий ГСПД), розмір популяції (кількість особин у кожному поколінні), прапорець, що вказує, чи має мутувати перше покоління.

Одержується перелік з’єднань кореневої особини.

Обчислюється значення критерію якості для кореневої особини.

Створюється стільки нових особин, скільки розмір популяції, і ці всі особини додаються до першого покоління. Кожна нова особина створюється на основі таких даних:

— перелік з’єднань кореневої особини;

— клон хромосоми кореневої особини;

— період L кореневої особини;

— випадкове ціле число — початкове значення для генератора випадкових чисел, який у кожної особини свій.

Клон хромосоми кореневої особини — це нова хромосома, що складається з копій генів кореневої особини. Водночас копія гена в собі містить ту ж саму вершину, що й оригінальний ген (себто, передається не новий об’єкт, а

лише посилання на старий). Але оскільки вершина не містить інформації про своє розташування, а її координати записані у її гені, то таким чином, клон хромосоми виявляється цілком самостійним від кореневої хромосоми. Як результат, різні особини мають незалежні гени, з якими виконуються вільно всі операції ГП і жодних впливів однієї хромосоми на іншу не виникає.

Отож, перелік з'єднань, який відповідає матриці інцидентності *A* конфігурації алгоритму використовується один на всі особини. Водночас при створенні особини їй передається не копія переліку з'єднань, а посилання на нього.

Оскільки всі особини першого покоління поки що ідентичні, то наступним кроком є їхня мутація. Мутація виконується з імовірністю 100% задля задання більшої різноманітності.

3.3.3 Мутація особини

Мутація може виконуватися або з якоюсь заданою ймовірністю, або з тією, що за замовчуванням (80%). Мутації підлягають гени в хромосомі по черзі. Для входів і виходів мутація не виконується. Якщо ж ген не є входом чи виходом, генерується випадкове число з рухомою комою в діапазоні від 0 до 1. Якщо це число менше або дорівнює ймовірності мутації, то виконується наступне.

1. Знаходиться множина часово-просторових позицій, на які можна поставити вершину ГСПД, яка відповідає цьому гену, так, щоб особина лишалася життєздатною, тобто, ГСПД — коректним.

2. З цієї множини, якщо вона не порожня, випадковим чином обирається нова позиція. При цьому генерується випадкове число в діапазоні від 0 до 1.0 і множиться на кількість позицій у множині, округлюється до найближчого цілого; після чого зі множини береться позиція з відповідним номером.

3. Ген одержує нове значення, тобто, вершина переставляється на нову позицію.

4. Гени в хромосомі сортуються за часом і простором: спочатку гени сортуються за координатою простору за незменшенням, а потім гени з однаковим значенням координати простору сортуються за координатою часу за незменшенням. Це спрощує надалі пошук потрібних генів.

3.3.4 Перестановка вершини на нову позицію

При перестановці вершини, тобто, зміні параметрів гена g мають виконуватися такі правила:

1) переставити вершину можна лише на порожню позицію, тобто, за умовою коректності ГСПД не може такого бути, що на цій позиції вже стоїть інша вершина;

2) горизонталь, тобто, лінія, що паралельна осі часу, на яку переставляється вершина, має бути або порожньою, або на ній мають бути елементи того самого типу, що й дана вершина;

3) якщо горизонталь не порожня, то кількість вершин на ній не має бути більшою за $L - 1$, причому координата t нової позиції не повинна бути порівняною з іншими вершинами на горизонталі за модулем L, тобто, має виконуватися рівняння (2.7).

Якщо ці правила виконуються, то перестановка вершини з цим геном зі старої позиції (t_1, s_1) на нову позицію (t_2, s_2) виглядає так:

— для гена встановлюються нові координати часу і простору t_2 й s_2 відповідно;

— стара позиція (t_1, s_1) позначається як вільна;

— нова позиція (t_2, s_2) позначається як зайнята геном g;

— якщо на горизонталі s_2 нічого не було, то позначається, що тепер вона відведена під елементи того самого типу, що й даний ген;

— виконується перевірка, чи ще є якісь вершини на горизонталі s_1 і якщо ні, то вона позначається порожньою.

3.3.5 Обчислення критерію якості для особини

Значення критерію показує якість особини. Значення обчислюється як добуток апаратних витрат і швидкодії (2.19). При цьому чим менше це значення, тим краща особина.

Значення апаратних витрат обчислюється за формулою (2.14), причому коефіцієнти складності регістра, суматора, помножувача, входу мультиплексора задані такими, як у формулі (2.15). Також ці коефіцієнти можуть бути задані користувачем. Оскільки заздалегідь не відома кількість вершин того чи іншого типу, їхнє визначення виконується наступним чином.

Створюється об’єкт, у якому є такі поля:

— посилання на особину, апаратуру якої рахуємо;

— кількість суматорів;

— кількість помножувачів;

— кількість входів мультиплексорів;

— кількість регістрів;

— кількість входів;

— кількість виходів.

Одержується відображення $g.x$ хромосоми g в апаратний простір, і за цим відображенням рахуються кількості компонентів кожного з типів, перелічених вище. Відображення $g.x$ є списком апаратних компонентів, які будуть у пристрої фактично, за винятком мультиплексорів, для яких визначається лише кількість їхніх входів.

Критерій швидкодії ($1/Q_T$ у формулі (2.19)) визначається наступним чином.

Виконується прохід по тактах (вертикалях) через множину вершин-генів. При цьому часова координата береться за модулем періоду L.

На кожній з L ітерацій:

1) одержується множина G_i генів, що розташовані в такті з номером n, причому $i \equiv n \bmod L$;

2) одержується ланцюг (K_k, D_k), який складається зі з'єднань $\boldsymbol{D}_{j,k} = \boldsymbol{K}_{j+1,k} - \boldsymbol{K}_{j,k}$, у яких вершини й початку $\boldsymbol{K}_{j,k}$, і кінця $\boldsymbol{K}_{j+1,k}$ входять у множину G_i, причому $T(\boldsymbol{D}_{j,k}) = 0$ для всіх з'єднань у ланцюгу, крім останнього;

3) на основі цих даних обчислюється затримка ланцюга в i-му такті періоду як сума очікуваних затримок вершин $\boldsymbol{K}_{j,k}$, які входять у ланцюг. Причому, якщо вершина $\boldsymbol{K}_{j,k}$ відображається в ПЕ, на вході якого стоїть мультиплексор, то до її затримки додається затримка мультиплексора. Затримка вершини затримки дорівнює нулю або затримці мультиплексора, що стоїть на вході регістра, у який вона відображається.

Найдовша з таких затримок для всіх i є затримкою критичного шляху, тобто, шуканим значенням критерію Q_T.

Затримки вершин $\boldsymbol{K}_{j,k}$, за замовчування дорівнюють: для суматора — 1.0, для помножувача — 2,1, для мультиплексора — 0,55. Ці затримки можуть встановлюватися користувачем.

3.3.6 Відображення хромосоми в структуру процесора

1. Гени хромосоми сортуються за часом і простором для зручності сканування відповідного просторового ГСПД. Шукане відображення = ∅. Верхня горизонталь s_0 — координата простору першого гена в хромосомі, нижня горизонталь $s_{\max}$ — координата простору останнього гена в хромосомі.

2. Виконується обхід, тобто, сканування хромосоми, починаючи з верхньої горизонталі ($n = s_0$).

3. Поки $n <= s_{\max}$, вибираються гени на n-й горизонталі, які зберігаються в множині G_n.

4. Обчислюється кількість входів мультиплексорів для множини G_n. Ця кількість додається до відображення. Також до відображення додається компонент типу type, який є спільним для G_n.

5. Обчислюється кількість регістрів для множини G_n як максимальна довжина проєкції вектора $\boldsymbol{K}_{j,k}$, який виходить із вершини з геном, що належить G_n. Така сама кількість регістрів додається до відображення.

6. $n = n + 1$ й перехід до п. 3.

3.3.7 Обчислення кількості входів мультиплексорів

Процедура обчислення кількості входів мультиплексорів виконується для множини G_n генів, розташованих на горизонталі з номером n. Причому гени вже відсортовані за координатою часу.

Якщо на цій горизонталі розташовані вершини, які є кінцями дуги міжітераційної залежності (тип Transfer_Control), то процедура повертає нуль — бо така вершина не потребує мультиплексора.

Мультиплексор потрібен у тому випадку, якщо після відображення графа в апаратуру в один і той же порт ведуть ребра з різних компонентів. Це означає, що в один і той же порт компонентів, що розташовані на одній горизонталі, ведуть ребра з різних горизонталей. Отже, для того, щоби з'ясувати, чи є входи мультиплексорів, і скільки саме, треба знайти координати, у яких розташовані вершини, з яких ребра ведуть у даний порт вершини, що розглядається. Ці координати зберігаються в множині spaces.

Кількість мультиплексорів для даного ПЕ дорівнює сумі кількостей входів мультиплексорів усіх портів вершин на горизонталі, що розглядається.

3.3.9 Пошук множини вільних позицій

При мутації вершина переставляється на вільну позицію. У результаті виконання процедури одержується множина часово-просторових позицій

(t_i, s_i), на які можна переставити вершину з даним геном так, що при цьому особина лишиться коректною (життєздатною).

З огляду на те, що при перестановці вершин не мають утворюватися зв'язки, які є зворотними в часі, шукається ліва та права межі часового інтервалу, у якому можуть бути доступні позиції для перестановки вершини. Вершину можна переставити лише на ті горизонталі, на яких розташовані елементи того самого типу, або на порожні горизонталі, на яких ще нічого немає. Отже, якщо горизонталь порожня, то до множини доступних позицій додаються всі позиції від лівої межі включно до правої межі не включно. Якщо ж на горизонталі елементи того самого типу, то з-поміж позицій, що лежать на цій горизонталі, шукаються доступні позиції в інтервалі між лівою та правою межами й додаються до множини доступних для перестановки.

3.3.10 Процедура еволюції в популяції

Ця процедура має такі вхідні параметри:

— максимальна кількість поколінь;

— розмір популяції (кількість особин у популяції);

— спосіб вибору батьків;

— розмір еліти.

Загалом, процес еволюції поділяється на обробку окремих поколінь, й обробка кожного покоління складається з таких кроків.

1. Обчислення кількості нових особин, які треба створити для нового покоління.

2. Оцінювання якості особин у поточному поколінні.

3. Створення нового покоління, що, своєю чергою, поділяється на підзадачі:

1) вибір пари батьківських особин;

2) операція кросоверу;

3) перевірка одержаних нащадків на життєздатність;

4) селекція нащадків.

4. Заміна поточного покоління на нове.

Еволюція припиняється через задану кількість поколінь. Хоча в деяких випадках еволюція завершується до того, як були пройдені всі покоління, коли процес оптимізації попадає в стійкий локальний оптимум.

Реалізація вищеназваних кроків має відмінності залежно від способу вибору батьків. Усього способів вибору батьків розглядається три: QValue, Roulette, Roulette з використанням ніш.

3.3.11 Еволюція з вибором батьків за методом QValue

На першому етапі обчислюється кількість особин, які треба створити для нового покоління. У разі використанні способу QValue не застосовується множина елітних особин. Тому при цьому способі вибору батьків для нового покоління треба одержати рівно стільки особин, скільки заданий розмір популяції. Хоча не обов'язково всі з цих особин виявляться новими.

На другому етапі оцінюється якість особин у поточному поколінні. Для кожної особини обчислюється значення критерію. Чим воно нижче, тим кращою вважається особина.

На третьому етапі виконується вибір пар батьків із використанням власне способу QValue, розмноження з використанням операцій кросовера, мутації та перевірки на життєздатність і селекції нащадків.

На четвертому етапі виконується заміна поточного покоління на нове.

3.3.12 Вибір пари батьківських особин

Обидві батьківські особини для розмноження обираються за одним і тим же принципом, єдина відмінність — під час вибору другої батьківської особини враховується, що друга особина не може бути тією самою, що й перша, тобто, батьки однозначно мають бути різними.

Процедура вибору батьків за методом QValue виконується таким чином. Для кожної особини $g_i \in Pop$ в поточному поколінні є обчислене значення якості, тобто, критерію Q_i. Серед множини якостей покоління знаходиться найбільше з них $Q_{\max}$.

Для кожної особини розраховується ймовірність вибору для батьківства:

$$P_i = \frac{Q_{max} - Q_i}{\sum_i (Q_{max} - Q_i)}. \quad (3.1)$$

Генерується випадкове число rand в діапазоні від 0 до 1, і якщо rand < P_i, то g_i обирається батьківською особиною й наступні особини вже не розглядаються. Ця особина видаляється зі множини *Pop* та відповідне їй значення Q_i зі множини якостей, щоби ці дані не впливали на процес вибору інших батьків.

Якщо ж дійшли до кінця множини *Pop* і так і не пощастило вибрати батьківську особину, виконується перехід на початок множини *Pop* і знову так само шукається батьківська особина.

У разі виродження еволюції всі значення Q_i виявляються однаковими. Це означає, що всі особини досягли однакових значень критерію, а, отже, знаменник у формулі (3.1) дорівнюватиме нулю. Тому робиться висновок, що більше немає сенсу продовжувати еволюцію і виконується вихід із процедури без вибору батьківської особини.

3.3.13 Операція кросоверу

При виконанні операції кросоверу спочатку одержуються клони хромосом обох вибраних батьківських особин. На основі цих клонів виконується багатоточковий взаємний обмін генами. При цьому одержуються дві хромосоми для нащадків за наступним алгоритмом.

Вхідні параметри цієї процедури — дві батьківські хромосоми $g_{Б1}$ та $g_{Б2}$. А шуканих нащадків позначимо як $g_{Н1}$ та $g_{Н2}$. Оскільки обидві хромосоми є різними просторовими поданнями одного й того самого ГСПД, то гени та їхня кількість у них збігаються. Відрізняється лише розташування генів у часово-просторових координатах.

У самій хромосомі гени відсортовані за координатами, тому їхній порядок у різних хромосомах різний. Але, оскільки кожна вершина графа має власний ідентифікаційний номер, то за ним знаходять відповідні гени.

Кросовер виконується наступним чином. Для кожної вершини графа беруться відповідні гени $\boldsymbol{K}_1 \in g_{Б1}$ та $\boldsymbol{K}_2 \in g_{Б2}$. Генерується випадкове число від 0 до 1. Якщо це число менше чи дорівнює ймовірності перестановки генів (за замовчуванням, 0,5), то $\boldsymbol{K}_1$ потрапляє в $g_{Н2}$, а $\boldsymbol{K}_2$ — в $g_{Н1}$. Якщо ж випадкове число більше, то $\boldsymbol{K}_1$ потрапляє в $g_{Н1}$, а $\boldsymbol{K}_2$ — в $g_{Н2}$.

Виконується перевірка нащадків на життєздатність. Може бути кілька випадків:

— життєздатні обидва нащадки — тоді виконуються такі кроки:

1) мутація обох нащадків;

2) для кожного нащадка обчислюється значення критерію;

3) селекція: у наступне покоління пройде той із двох нащадків, у якого значення критерію менше;

— життєздатний лише один із них — тоді виконується його мутація й саме він відмічається як обраний для нового покоління;

— жоден не життєздатний — тоді обраною в наступне покоління є краща з батьківських особин до операції кросовера, причому, без мутації.

Якщо обраний нащадок виявляється гіршим за найкращу батьківську особину, то обраною стає ця батьківська особина. Щоби визначити, хто кращий, хто гірший, порівнюються їхні значення критерію.

Далі обрана особина додається до нового покоління.

3.3.14 Еволюція з вибором батьків за методом Roulette

На першому етапі обчислюється кількість особин, які треба створити для нового покоління ps_n. Оскільки при використанні цього методу застосовується елітна множина, то кількість нових особин дорівнює розміру популяції без розміру еліти: $ps_n = ps - ps_e$.

На другому етапі оцінюється якість особин у поточному поколінні. Воно виконується так само, як у випадку методу QValue: для всіх особин обчислюється значення критерію.

На третьому етапі створюється нове покоління.

1. Вибір пари батьківських особин виконується за правилом рулетки.
2. Кросовер відбувається так само, як у методі QValue.
3. Перевірка одержаних нащадків на життєздатність.
4. Формування нового покоління й селекція нащадків.

Правило рулетки реалізується наступним чином. Створюється лінійка, яка імітує рулетку. Лінійка починається з нуля і складається з числових інтервалів, кожен із яких відповідає певній особині. Ширина інтервалу для певної особини g_i дорівнює

$$\Delta_i = \frac{1}{v_i},$$

де v_i — значення критерію для особини g_i. Тобто, чим більше значення критерію, тим менший інтервал. Одержані інтервали Δ_i відкладаються на числовій прямій від нуля (не включно). Припустимо, що маємо інтервали завширшки 1.5, 0.5, 1.3. Тоді на лінійці будуть інтервали (0, 1.5], (1.5, 2], (2, 3.3].

Береться випадкове число в діапазоні від 0 до кінця лінійки. У який інтервал потрапило випадкове число, та відповідна особина й обирається першою батьківською особиною. Далі інтервал, що відповідає першій

особині, викидається з лінійки й шукається друга батьківська особина так само, як шукалася перша особина.

При розгляді кожної пари батьків обидва їхні нащадки мутують і, якщо обидва життєздатні, то вони додаються до нового покоління; якщо життєздатний лише один, то він мутує й додається до нового покоління; якщо ж обидва не життєздатні, тоді ніхто з них не додається взагалі.

На четвертому на етапі з одержаної множини нащадків обираються найкращі. Отриманих нащадків може бути, як максимум, удвічі більше за розмір популяції, коли життєздатними виявилися всі, а може бути надто мало, якщо нежиттєздатних виявилося дуже багато. Але якщо після розгляду всіх батьківських пар нове покоління виявилося порожнім, тобто, усі нащадки виявилися нежиттєздатними, то роль нового покоління виконує поточне.

Селекція полягає в тому, що серед усіх нащадків шукаємо множину S найкращих особин. Це такі особини, у яких значення критерію найменше. Якщо виявиться, що всього нащадків менше, ніж потрібна кількість нових особин ps_n, то, фактично, відібрані будуть усі нащадки. Їхня недостача компенсується наступним чином.

Обчислюється, скільки ще треба добрати особин:

$$\delta = ps_n - |S|.$$

Беруться δ нащадків зі множини S, для кожного з них створюються клони, які додаються до множини S. Якщо $\delta > |S|$, то клонування виконується кілька разів, допоки $|S| = ps_n$.

З-поміж особин у поточному поколінні шукається ps_e. найкращих особин, тобто з найменшим значенням критерію, які представляють еліту. Множина S разом з обраною елітою складають нове покоління. І, нарешті, поточне покоління заміняється на нове, тобто, тепер популяція складається з особин нового покоління.

3.3.15 Визначення ніш

Якщо в алгоритмі використовуються ніші, то перед тим, як створювати особини для нового покоління, обчислюються нішеві коефіцієнти k_H. Оскільки алгоритм із нішами виконується доти, доки лишається хоча б одна ніша, то якщо в цей момент ніш уже немає, батьки обиратися не будуть.

Константи за замовчуванням:

— радіус ніші $r = 10$;

— поріг належності до ніші $sh_L = 5$.

Для кожної особини $g_i \in Pop$ з поточного покоління обчислюється кількість апаратури кожного виду: суматорів n_{Ai}, помножувачів n_{Mi}, входів мультиплексорів n_{Xi}, регістрів n_{Ri}. Далі для кожної особини g_i обчислюється значення критерію і знаходиться найкраща особина (та, у якої значення критерію найменше). Значення апаратури найкращої особини запам'ятовуються як n_{AB}, n_{MB}, n_{XB}, n_{RB}.

Далі для кожної особини g_i перевіряється, чи вона в принципі належить до якоїсь ніші, чи ні. Це робиться наступним чином.

1. Для всіх $g_i \neq g_j$ обчислюється відстань між g_i та g_j:

$$d_k = \sqrt{\left(n_{A_i} - n_{A_j}\right)^2 k_A^2 + \left(n_{M_i} - n_{M_j}\right)^2 k_M^2 + \left(n_{X_i} - n_{X_j}\right)^2 k_X^2 + \left(n_{R_i} - n_{R_j}\right)^2 k_R^2}, \quad (3.2)$$

де $k_A = 1$, $k_M = 10$, $k_X = 0.3$, $k_R = 1$ — відповідні коефіцієнти складності. Також знаходиться значення функції подібності (sharing function, sh-функції):

$$sh_k = \begin{cases} \dfrac{r}{d_k}, & \text{якщо } (0 < d_k < r), \\ 0, & \text{інакше.} \end{cases}$$

2. Одержується сума всіх значень sh-функції:

$$sh = \sum_k sh_k .$$

3. Якщо $sh \geq sh_L$, то особина g_i належить до якоїсь ніші, інакше — не належить і є випадковою особиною. Така особина не братиме участі в операціях кросовера.

Якщо ж встановлено, що g_i належить до якоїсь ніші, то починається пошук таких ніш, які є Парето-оптимальними. Особини, що входять у такі ніші є ефективними, вони "випирають" серед інших по якійсь одній з осей у багатовимірному параметричному просторі (див. рис. 2.9). У даному випадку маємо 4-вимірний простір, де по осях ведеться відлік кількості суматорів, помножувачів, входів мультиплексорів, регістрів.

Вважається, що ефективні особини це не ті, що мають малу відстань d_k від найкращої особини, а ті, що наближені до якоїсь однієї осі. Вони мають бути на відстані від неї не далі, ніж як у найкращої особини плюс певна величина. Така величина обчислюється відносно параметрів кращої особини:

$$\delta_A = 0.3n_{AB}; \quad \delta_M = 0.3n_{MB}; \quad \delta_X = 0.3n_{XB}; \quad \delta_R = 0.3n_{RB}.$$

Тоді вважається, що якщо $n_{Ai} \leq n_{AB} + \delta_A$, то суматорів особині доволі мало, щоби вважати, що ця особина належить до ніші ефективних особин. Отже, якщо особини g_i "випирає" по якійсь одній осі, а по всіх інших значення велике, то відмічаємо, що g_i належить до якоїсь ніші й буде розглядатися далі. В інакшому ж випадку ця особина далі не розглядатиметься і в кросовері не братиме участі. Множина осіб, які належать ефективним нішам і які лишилися для подальшого розгляду, позначається як B.

Якщо немає жодної особини $g_i \in B$, то це означає, що немає жодної ефективної ніші, тому робиться висновок, що більше немає сенсу

продовжувати еволюцію. Виконується вихід із процедури, без обчислення коефіцієнтів.

При формуванні ефективних ніш по черзі попарно розглядаються особини g_i та g_j, такі, що $g_i, g_j \in B$. Обчислюється відстань d_k (3.2) між g_i та g_j. Якщо $d_k < r$, то g_i та g_j опиняються в одній і тій самій ніші. При цьому, якщо ніші накладаються чи перетинаються, то вони також об'єднуються в одну нішу.

Якщо ніша лише одна, то нішевий коефіцієнт дорівнює $k_H = 1$. Якщо ж ніш більше, ніж одна, то нішеві коефіцієнти обчислюються наступним чином.

1. Шукається центр ніші. Спочатку для особини g_i обчислюється кількість сусідів. Сусіди — це такі особини g_j з даної ніші, що відстань d_k між g_i та g_j така, що $d_k < r$. Тоді особина, у якої сусідів найбільше і є центром ніші.

2. Якщо центр ніші є Парето-оптимальним серед інших центрів ніш, тобто, "випирає" по якійсь осі (рис. 2.9), то вважається, що ніша є ефективною.

3. Нішеві коефіцієнти обчислюються таким чином. Якщо центр ніші є Парето-оптимальним по осі регістрів (а суматорів, помножувачів та входів мультиплексорів "багато"), то

$$k_H = \frac{n_{R_Ц}}{n_{R_B}} \geq 1,$$

де $n_{RЦ}$ — кількість регістрів у центрі ніші. Аналогічно розраховуються нішеві коефіцієнти по нішах, які Парето-оптимальні за іншими характеристиками.

3.3.16 Еволюція з вибором батьків за методом Roulette з нішами

Цей метод ГП відрізняється від просто методу Roulette у наступному. Формується множина K обчислених нішевих коефіцієнтів, у якій кожній особині g_i у відповідність поставлений нішевий коефіцієнт k_{Hi}. Якщо ж

застосовується метод Roulette без ніш, то просто всі нішеві коефіцієнти дорівнюють $k_{Hi} = 1$.

Створюється лінійка, яка імітує рулетку. Ширина інтервалу для певної особини e дорівнює

$$\Delta_i = \frac{k_{H_i}}{v_i}.$$

Отже, нішевий коефіцієнт показує, наскільки гіршими є особини в ніші у порівнянні з найкращою особиною в поколінні. Множення на коефіцієнт k_{Hi} ширини інтервалу Δ_i підвищує шанси уціліти особинам цієї ніші задля збереження генетичного різноманіття в наступному поколінні. Якщо для даного ГСПД виявляється, що ніші швидко деградують і їхня кількість зменшується до нуля, то треба збільшити нішеві коефіцієнти.

3.4 Синтез процесора для розв'язання диференційного рівняння

У підрозділі 2.1 вже було показано приклад синтезу процесора для розв'язання диференційного рівняння методом просторового ГСПД. У цьому підрозділі показується синтез такого ж самого процесора, але методом генетичного програмування за допомогою системи SDFCAD з метою ілюстрації ходу виконання методу та показу його ефективності.

Спочатку вводиться ГСПД алгоритму. Цей ГСПД виглядає так, як показано на рис. 3.1. Приймається, що блок множення виконує свою операцію за один такт. Тоді період алгоритму складає $L = 5$ тактів. Процесор був синтезований трьома методами: Qvalue, Roulette та Roulette з нішами (Roulette+N). Число особин у поколінні $ps = 300$. Максимальна кількість поколінь — 50. Виконуються два етапи Алгоритму 2 (розд. 2.3) — до й після балансування ГСПД. Для методів Roulette та Roulette+N використовувалась елітна множина розміром 10% від ps.

ГСПД однієї з особин після мутації в першому поколінні показаний на рис. 3.2. За цим ГСПД можна визначити, що $n_A = 4$, $n_M = 2$, $n_X = 9$, $n_R = 12$.

Оцінка ходу генетичного програмування зазвичай ілюструється графіком залежності функції придатності v від номера покоління. На рис. 3.3 показана залежність якості проєкту від покоління на першому етапі синтезу до врівноваження ГСПД, а на рис. 3.4 — та сама залежність на другому етапі після врівноваження ГСПД при виконанні оптимізації трьома методами.

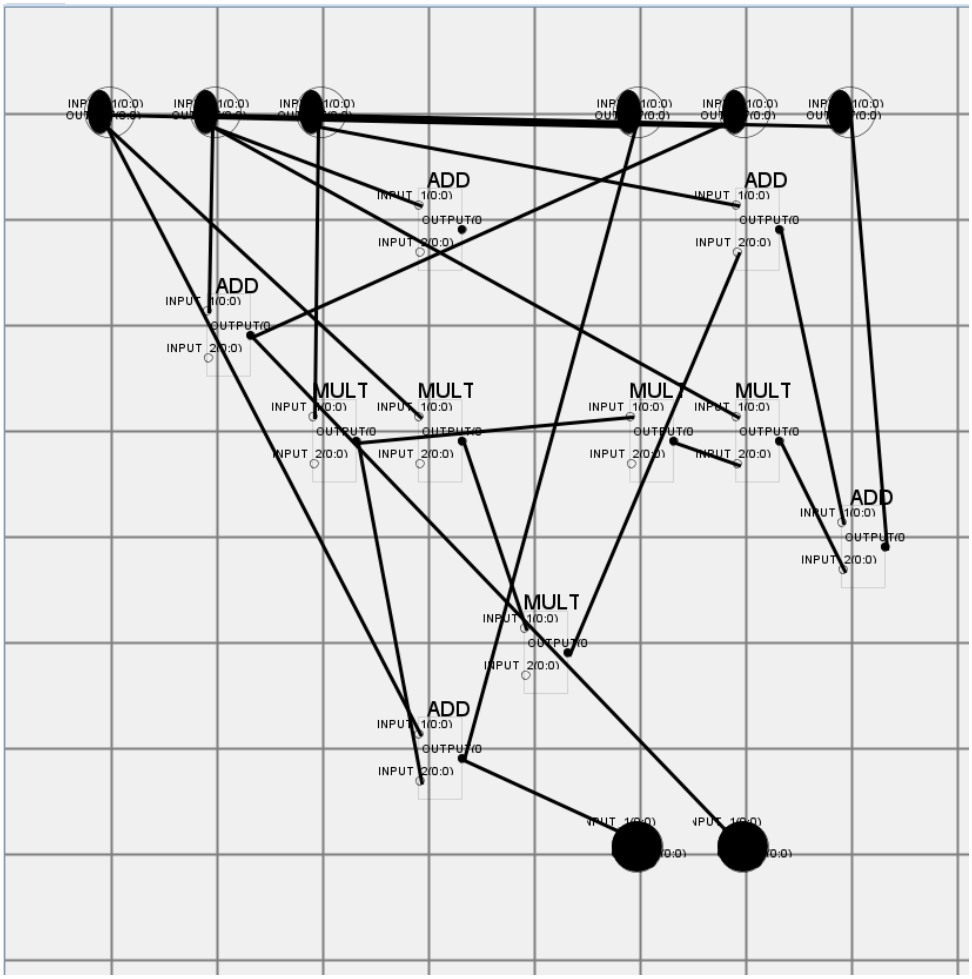

Рисунок 3.2 — ГСПД після мутації

Аналіз графіків на рис. 3.3, 3.4 показує, що алгоритм Qvalue має нестабільні властивості конвергенції. Алгоритми Roulette та Roulette+N мають швидку конвергенцію до оптимального рішення, яка закінчується за 3–30

поколінь завдяки наявності елітної множини. Алгоритм Roulette+N дає кращу оптимізацію за інші алгоритми, принаймні, для даного ГСПД завдяки ширшому скануванню простору рішень.

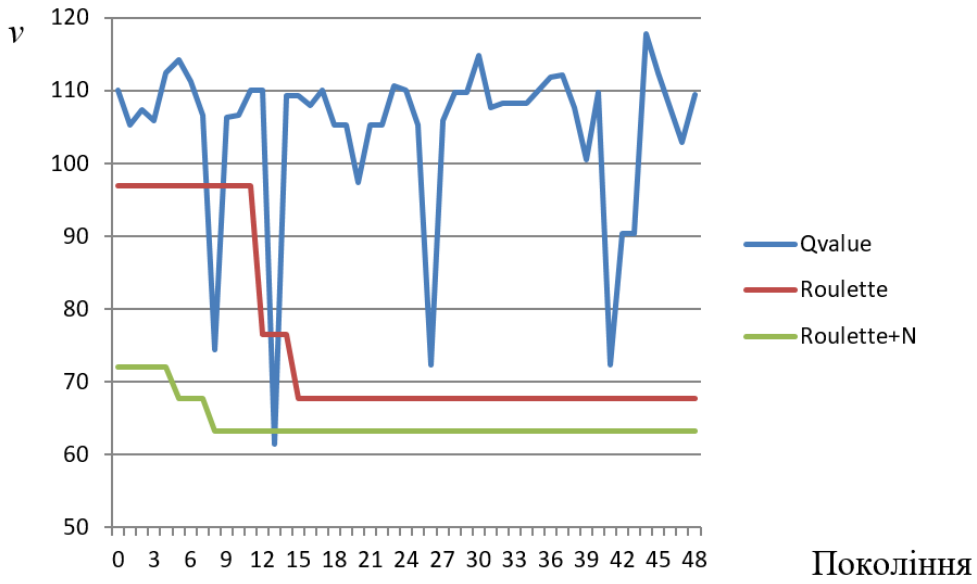

Рисунок 3.3 — Залежність функції придатності від покоління до урівноваження ГСПД

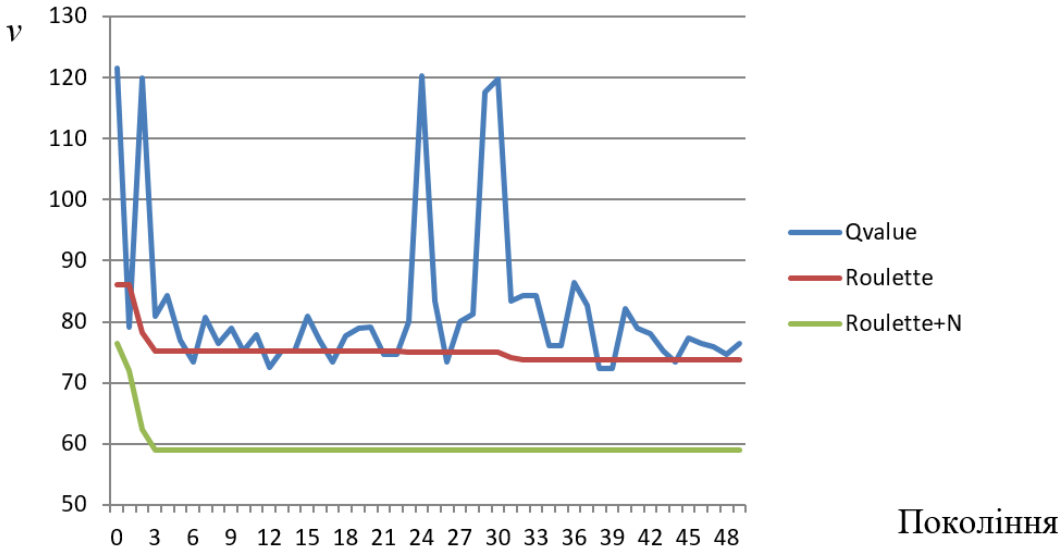

Рисунок 3.4 — Залежність функції придатності від покоління після урівноваження ГСПД

Оптимальний варіант ГСПД, який одержаний за методом Roulette+N показаний на рис. 3.5. Для нього $n_A = 1$, $n_M = 1$, $n_X = 9$, $n_R = 2$. Протягом оптимізації цим методом значення критерію оптимальності зменшилося на 29%. Опис відповідного ОП мовою VHDL подано у Додатку А.

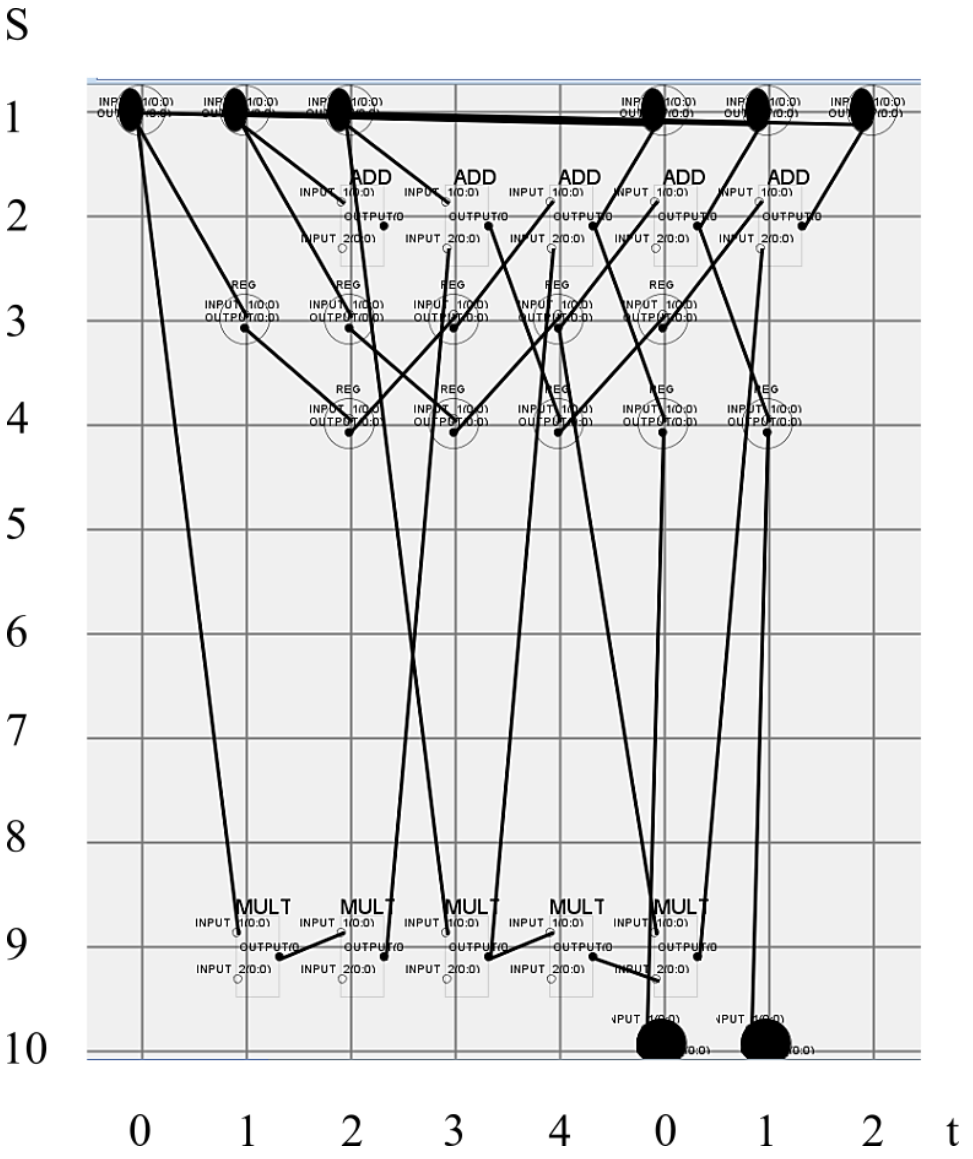

Рисунок 3.5 — Результат оптимізації ГСПД за методом Roulette+N

Дане рішення суттєво переважає за якістю рішення, яке одержане в розділі 2, у якого $n_A = 1$, $n_M = 1$, $n_X = 11$, $n_R = 6$. Але при його синтезі приймалися припущення, що множення виконується за один такт, а не за два,

і те, що результати видаються через два виходи постійно, а не через один і в різних тактах.

При оптимізації ГСПД має значення вибір розміру популяції *ps*. Наприклад, у разі проєктування даного процесора при зменшенні *ps* з 300 до 100 якість рішення погіршилася з $v = 74.06$ до $v = 74.97$, тобто, у знайденому рішенні кількість входів мультиплексорів збільшилася на два. Зате тривалість оптимізації зменшилася більш як утричі.

Порівнюючи результати методів Roulette та Roulette+N, можна висновити, що додавання ніш дає кращі результати. На рис. 3.6 показаний графік зміни кількості ніш від покоління до покоління.

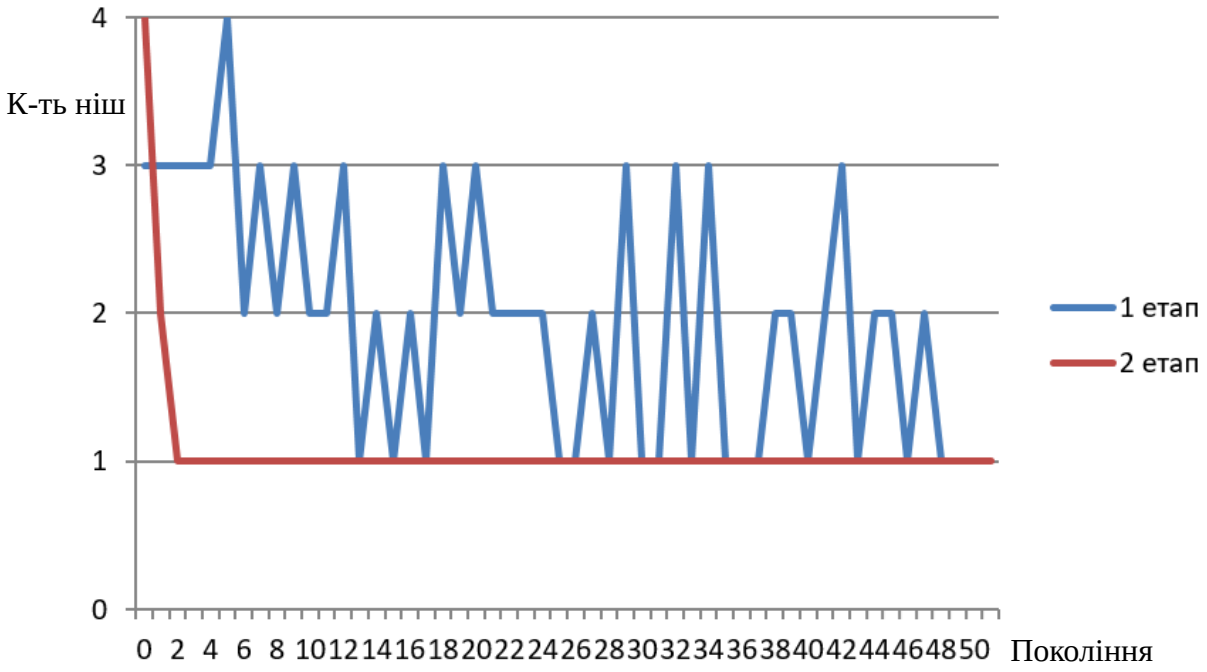

Рисунок 3.6 — Зміна кількості ніш під час оптимізації

На рис. 3.6 видно, що ніші на другому етапі швидко деградують через те, що знайдене оптимальне рішення, яке суттєво переважає рішення, що належать нішам.

3.5 Висновки до третього розділу

Отже, метод генетичного програмування просторового ГСПД, розроблений у другому розділі, знайшов втілення в програмному комплексі SDFCAD. Розроблення та попереднє тестування цього комплексу свідчить про наступне.

1. Система SDFCAD здатна виконувати ввід алгоритмів, заданих ГСПД, й автоматично виконувати синтез конвеєрних процесорів, які реалізують цей алгоритм за допомогою генетичного програмування.

2. Реалізовано три алгоритми генетичного програмування, які відрізняються процедурами селекції — алгоритм із пропорційним добором Qvalue, алгоритм Монте-Карло Roulette та алгоритм Монте-Карло з нішами Roulette+N. Результати експериментів із синтезом процесора для розв’язання диференційного рівняння показав найбільшу ефективність останнього алгоритму, якому вдалося знайти оптимальне рішення, яке не гірше за точне рішення, знайдене за алгоритмом цілочисельної оптимізації.

Наступний розділ присвячений розгляду синтезу конвеєрних процесорів для виконання алгоритмів ЦОС за допомогою розробленого методу.

РОЗДІЛ 4. ПРОЄКТУВАННЯ СИСТЕМ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ

4.1 Процесор дискретного косинусного перетворення

4.1.1 Алгоритм дискретного косинусного перетворення

Двовимірне дискретне косинусне перетворення (ДКП) відображає масив дійсних чисел розмірами $n \cdot n$ у масив косинусного спектра. Пара двовимірних ДКП та інверсного ДКП (ІДКП) використовується для ущільнення зображень за стандартами JPEG і MPEG, для яких $n = 8$. За звичайним алгоритмом двовимірне ДКП обчислюється як n-точкове ДКП над стовпчиками матриці розміром $n \cdot n$, а потім — n-точкове ДКП над рядками матриці проміжних результатів. Так, восьмиточкове ДКП обчислюється за формулами:

$$Y(0) = \frac{1}{\sqrt{8}} \sum_{m=0}^{7} X(m), \quad Y(k) = \frac{1}{2} \sum_{m=0}^{7} X(m) cos \frac{(2m + 1)k\pi}{16}, \qquad (4.1)$$

де $X(m)$ — вхідні дані, $Y(k)$ — k-й коефіцієнт косинусного спектра, $k = 1, 2, \ldots, 7$.

Розроблення спецпроцесорів для ДКП і ІДКП є типовим тестом для систем високорівневого синтезу. Структури процесорів складаються з вхідної буферної пам’яті, двох блоків конвеєрної обробки, що виконують ДКП або ІДКП розміром 8, і буферної пам’яті між ними для транспозиції матриці проміжних результатів. Загальний блок керування призначений для генерації необхідних адресних послідовностей і керуючих сигналів.

Швидкі алгоритми обчислення одновимірних ДКП та ІДКП ґрунтуються на факторизації формул (1) та (2) і в більшості випадків є різновидами алгоритму Чена [169]. Ці алгоритми характеризуються мінімальною кількістю

операцій (рекорд — 11 операцій множення). Але через неоднорідність графа алгоритму утруднюється побудова конвеєрного обчислювача ДКП. Як тестовий алгоритм ДКП у багатьох роботах приймають 8-точкове ДКП [119]. Його ГСПД показаний на рис. 4.1. У ньому 13 операцій множення.

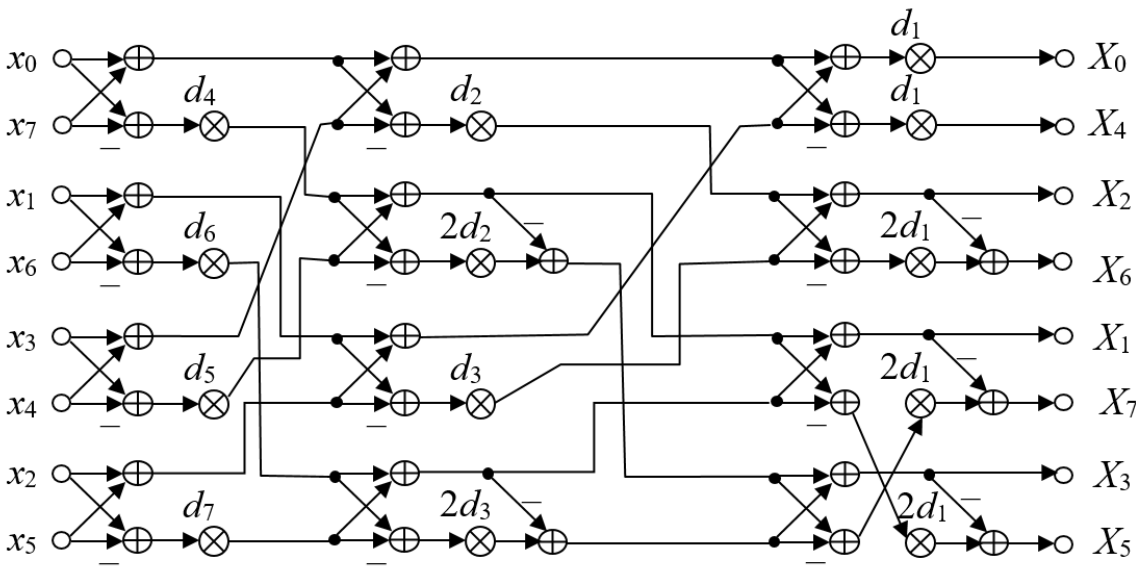

Рисунок 4.1 — ГСПД стандартного 8-точкового ДКП

У цьому алгоритмі $d_1 = \sqrt{0.5} = 0.707$, $d_2 = \sqrt{0.5(1 + d_1)} = 0.924$, $d_3 = \sqrt{0.5(1 - d_1)} = 0.383$, $d_4 = \sqrt{0.5(1 + d_2)} = 0.990$, $d_5 = \sqrt{0.5(1 - d_2)} = 0.139$, $d_6 = \sqrt{0.1(1 + d_3)} = 0.900$, $d_7 = \sqrt{0.5(1 - d_3)} = 0.437$.

Оскільки обробка в процесорі ДКП ведеться в конвеєрному режимі, доцільно вибрати період виконання алгоритму $L = 8$ тактів. Тоді дані й результати поступають по одному в кожному такті. Як видно з рис. 4.1, порядок даних на вході та виході ГСПД відрізняється від натурального. Для вирівнювання порядку даних встановлюють вхідний та вихідний буфери. Але метод просторового ГСПД для встановлення гідного порядку даних дає змогу ставити автоматично регістрові схеми з мінімізованою кількістю регістрів.

4.1.2 Проєктування процесора ДКП за допомогою системи SDFCAD

ГСПД, введений у систему SDFCAD, показано на рис. 4.2. Алгоритм містить 13 операцій множення та 29 додавання. За період $L = 8$ тактів він споживає 8 даних і виводить 8 результатів через один порт вводу й порт виводу в натуральному порядку.

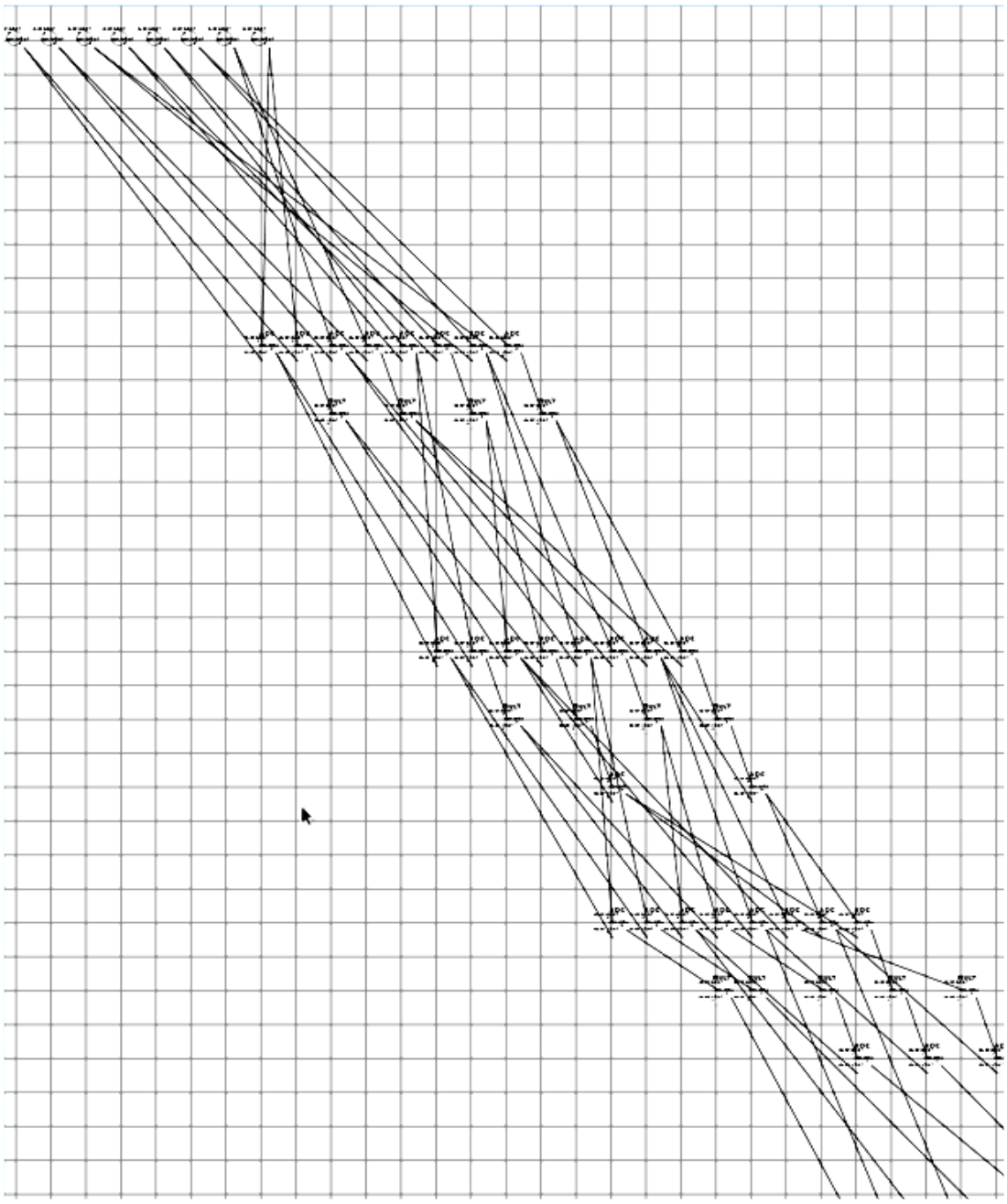

Рисунок 4.2 — Просторовий ГСПД ДКП у вікні SDFCAD

Синтез схеми процесора виконується за методом, розробленим у другому розділі з використанням засобу автоматичного синтезу SDFCAD. Спочатку на основі ГСПД, представленого на рис. 4.1, було згенероване перше покоління особин, які мали хромосоми g_i з випадковим заданням генів.

Вибрана кількість поколінь дорівнює 300. Хоча хромосоми мають випадкові координати векторів-вершин, результівні КА є коректними, тобто, кожна особина відображається в коректні, але неоптимальні структуру та розклад.

Потім виконується алгоритм еволюційної оптимізації — генетичного програмування ГСПД. Для аналізу роботи еволюційного підходу алгоритм виконувався з двома видами функцій селекції: Qvalue (2.20) та Roulette, тобто, метод Монте-Карло.

Останній алгоритм має дві модифікації: без використання ніш, з використанням ніш (Roulette+N). Оцінка ГП зазвичай ілюструється графіком залежності функції придатності (критерію якості, такого як (2.19)) від номера покоління. На рис. 4.3 показана залежність якості проєкту від покоління на першому етапі синтезу до врівноваження ГСПД, а на рис. 4.4 — та ж сама залежність на другому етапі після врівноваження ГСПД.

Аналіз графіків на рис. 4.3, 4.4 показує, що алгоритми на цих двох етапах сходяться до стабільного локального мінімуму протягом 40 поколінь.

ГСПД був перетворений в опис процесора ДКП, який подано в Додатку Б.

У таблиці 4.1 показані апаратні витрати до (найкраща особина у 0-му поколінні) та після оптимізації процесора ДКП цими трьома методами, а також процесора, ГСПД якого оптимізований вручну.

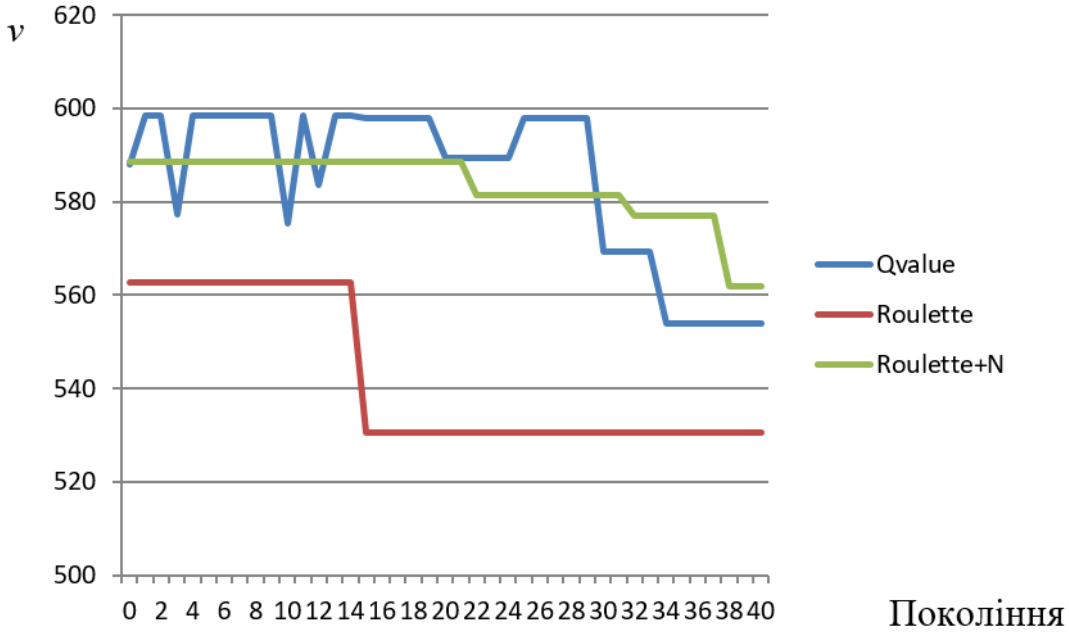

Рисунок 4.3 — Залежність функції придатності від покоління до урівноваження ГСПД

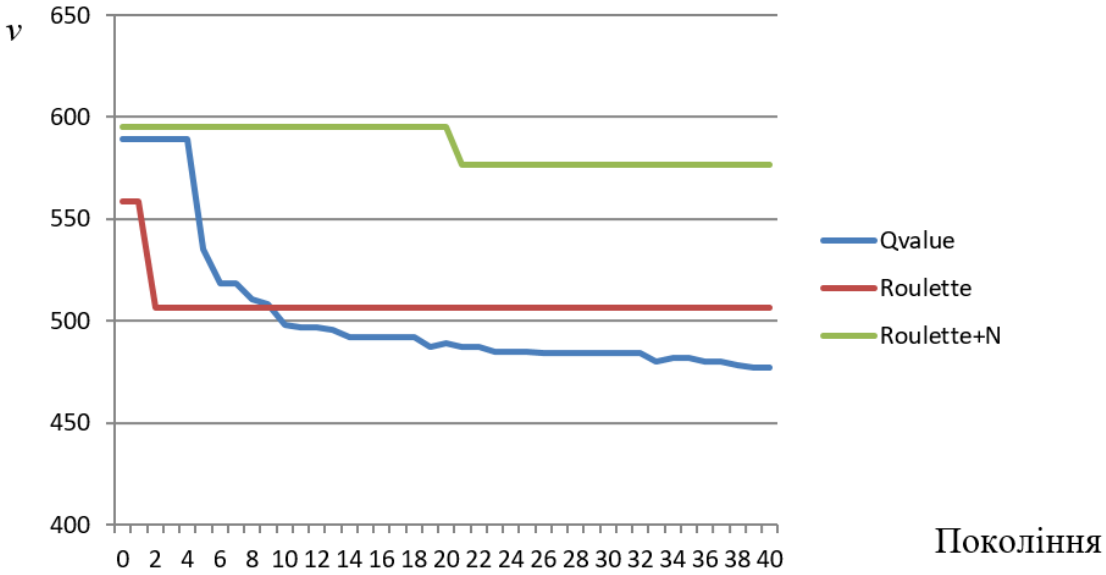

Рисунок 4.4 — Залежність функції придатності від покоління після урівноваження ГСПД

Таблиця 4.1 — Апаратні витрати до та після оптимізації процесора різними методами

Метод оптимізації	Блоків множення, n_M		Суматорів, n_A		Регістрів, n_R		Входів мультиплексорів, n_X		v	
	До	Після	До	Після	До	Після	До	Після	До	Після
Qvalue	7	3	18	9	96	28	43	231	584	480
Roulette	7	3	19	6	107	63	42	230	608	546
Roulette+N	6	3	19	11	101	38	49	232	562	506
Вручну	—	3	—	6	—	58	—	117	—	275

Аналіз таблиці 4.1, а також графіків на рис. 4.2, 4.3 показує, що:

— методи ГП спроможні автоматично оптимізувати конвеєрні процесори з покращенням критерію якості на 11–22%;

— рівень якості результату синтезу суттєво залежить від випадкового задання початкового покоління особин (діапазон відхилення 10%);

— значення критерію оптимальності для рішення, знайденого автоматично, може бути на 80% гіршим за значення критерію для рішення, знайденого вручну за методом просторового ГСПД;

— усі методи зменшують кількість блоків множення, регістрів, суматорів до мінімальних значень за рахунок збільшення кількості входів мультиплексорів;

— найкращі результати показав метод Roulette;

— слушно впливати на хід синтезу керуванням вагових коефіцієнтів у формулі критерію (2.14), наприклад, варто зменшити коефіцієнт складності регістрів, яких у ПЛІС завжди достатньо й кількість яких приблизно пропорційна кількості входів мультиплексорів. Крім того, це сприяє зменшенню енергоспоживання через зменшення міжрегістрових пересилок.

4.1.3 Апаратні параметри синтезованого процесора

Апаратні параметри синтезованих конвеєрних процесорів показані в таблиці 4.2. Там ці результати порівнюються з характеристиками 1D ДКП-процесорів, які отримані іншими відомими методами синтезу.

Таблиця 4.2 — Апаратні витрати синтезованих процесорів ДКП

Метод	Кількість суматорів	Кількість блоків множення	Кількість регістрів
Згортання ГСПД [169]	6	3	29
Ресинхронізація та складання розкладу [119]	7	3	34
Запропонований, Roulette	6	3	63
Запропонований, QValue	9	3	28

Отже, процесор, який синтезовано автоматично за методом генетичного програмування просторового ГСПД, за своїми основними показниками є не гіршим, а подекуди навіть кращим за відомі взірці таких процесорів.

Остаточний ГСПД був описаний мовою VHDL і компілятором-синтезатором перетворений у прошивки ПЛІС різних серій. При цьому розрядність вхідних даних дорівнює 8, а результатів та коефіцієнтів — 12. 1D ДКП-процесор має кілька реалізацій у ПЛІС, які розроблені вручну й описані в літературі. Параметри деяких із них разом із новим процесором для серії ПЛІС Xilinx Vivado-7 показані в таблиці 4.3.

Варто зауважити, що у процесорі A1DDCT кількість блоків множення зменшена до нуля завдяки тому, що множення виконується через додавання зсунутого множеного й тому кількість логічних таблиць (LUT, look-up tables) набагато більша, ніж в інших процесорів. Слід зважати на витрати на помножувачі, які оцінюються як $1.05(n-1)^2$, де n — розрядність множників. Для 8-розрядних процесорів ДКП це складає 100 LUT з урахуванням 10-

розрядних коефіцієнтів та 12-розрядних множених. Якщо синтезується менше DSP48, ніж розраховано, наприклад, 4 замість 6, то це означає, що блоки множення малої розрядності реалізовані на LUT, а не на DSP48. Це враховано при розрахунку відношення пропускна здатність — апаратні витрати у МГц/ LUT (більше — краще).

Таблиця 4.3 — Параметри процесорів 1D ДКП

Процесор	LUT	CLB	Тригерів	Блоків DSP48	Макс. тактова частота, МГц	Відношення МГц/ LUT
Синтезований 8 біт	245	140	525	3	389	0,72
ALDCT [26] 8 біт	322	121	387	3	347	0,56
A1DDCT [72] 8 біт	950	286	1133	0	450	0,47
Синтезований 16 біт	319	156	541	3	370	0,40
Shazeeda [213] 16 біт	2412	—	1901	20	230	0,1

Отож, автоматично синтезований процесор ДКП має не гірші, а подекуди кращі характеристики, ніж у процесорів, які були розроблені кваліфікованими інженерами.

Сам по собі 1D ДКП-процесор не має практичного значення. Але він є складовою частиною 2D ДКП-процесора, який є основою процесора компресії зображень. У літературі є багато описів взірців таких процесорів і в таблиці 4.4 наведено порівняння багатьох із них. Для порівняння з ними був розроблений 16-розрядний 2D ДКП-процесор на основі синтезованого 1D ДКП-процесора, який розглянуто вище.

Заслуговує уваги порівняння з процесором Hannig, який був синтезований за допомогою фреймворку PARO. Цей фреймворк залучає розпаралелювання алгоритму, представленого гніздом циклів через розгортання та згортання циклів і розв’язання задачі планування класичним способом. Отже, за такої

ж самої розрядності новий процесор має на третину менше блоків множення, на 11% менші апаратні витрати в LUT і на 19% більшу тактову частоту за рахунок збільшення кількості тригерів у півтора рази. Це свідчить про переваги методу ГП просторового ГСПД над іншими методами синтезу.

Таблиця 4.4 — Параметри процесорів 2D ДКП

Процесор	Серія ПЛІС	Роз-ряд-ність	LUT	Триге-рів	Блоків DSP48	Блоків RAM	Макс. тактова частота, МГц	Від-ношення МГц/LUT
Синтезований	Virtex-2	8	1031	1073	6	0	195	0,13
ALDCT [26]	Virtex-2	8	599	—	6	0	230	0,07
Синтезований	Virtex-2	16	1575	1747	6	0	155	0,04
Hannig [106]	Virtex-2	16	1754	1152	8	1	130	0,03
Xilinx [33]	Virtex-2	16	6832	7964	0	2	96	0,01
Синтезований	Spartan-3e	8	1168	1108	4	0	124	0,08
Kusuma [136]	Spartan-3e	8	1750	1696	11	1	85	0,01
Pradeepthi [187]	Spartan-3e	8	1239	1551	8	1	101	0,02
Синтезований	Virtex-5	8	548	1059	6	0	215	0,19
Parate [177]	Virtex-5	8	1152	578	4	1	—	—
Синтезований	Virtex-7	8	657	1098	4	0	323	0,23
2D_DCT_2ROM [129]	Virtex-7	8	1786	—	0	2	338,5	0,19
2D_DCT_4ROM [129]	Virtex-7	8	2021	—	0	4	256	0,13

Також порівняння синтезованого процесора зі взірцями процесорів, які оптимізувалися вручну, показує, що цей процесор є не гіршим за них, а за

деякими показниками, як, наприклад, максимальна тактова частота — навіть кращим, причому за умови його збільшеної розрядності. Це пояснюється тим, що синтезований процесор має високий рівень конвеєризації, що підтверджується надмірною кількістю тригерів. Крім того, досліди з налаштуваннями компілятора-синтезатора довели, що часові показники не покращуються при встановленні режиму синтезу з додатковою ресинхронізацією.

Отож, розроблення спецпроцесорів ДКП показало високу ефективність методу ГП просторового ГСПД. Ці процесори можна використовувати в усіх нових пристроях для обробки фото- і відеозображень, включно з такими, для яких потрібна обробка з високою пропускною спроможністю.

4.2 Синтез процесорів для швидкого перетворення Фур'є

4.2.1 Структури конвеєрних процесорів ШПФ

Алгоритм швидкого перетворення Фур'є (ШПФ) широко застосовується в системах ЦОС. Через високі вимоги до продуктивності, процесор ШПФ має великі апаратні витрати та споживає велику потужність. Алгоритми ШПФ ґрунтуються на факторизації дискретного перетворення Фур'є, яка виконується за підходом: поділяй та володарюй. В алгоритмі Кулі — Тьюки за основою 2 N-точкове дискретне перетворення Фур’є (ДПФ) декомпонується на два $N/2$-точкових ДПФ, а потім ці $N/2$-точкові ДПФ рекурсивно поділяються на менші ДПФ, закінчуючи двоточковими ДПФ. Процедура двоточкового ДПФ, яка додатково має операцію повороту комплексного вектора, названа базовою операцією ШПФ за основою 2 і має 10 операцій додавання та 4 операції множення на комплексні коефіцієнти повороту.

Для зменшення кількості операцій множення використовуються алгоритми за більшою основою, такою як 4, 8, але при цьому структура базової операції стає складнішою. У цьому випадку застосовують алгоритм із розщепленою основою, який має переваги алгоритмів за основою 2 і 4 [81].

Алгоритми базових операцій зі взаємно-простими основами використовують в алгоритмах Томаса-Гуда. Вони ґрунтуються на китайській теоремі про залишки при розкладанні N-точкового ДПФ на менші p-точкове і r-точкове ДПФ, причому $N = p \cdot r$; p, r є взаємно простими [171]. За допомогою такої факторизації можна уникнути множення на комплексні коефіцієнти повороту між ступенями алгоритму ШПФ завдяки збільшенню кількості додавань та неоднорідності графу алгоритму. Модифікацією алгоритму зі взаємно-простими основами є алгоритм ШПФ Віноградa. У цьому алгоритмі досягається мінімальна кількість множень, але кількість додавань збільшується.

Конвеєрні процесори ШПФ — це високопродуктивні паралельні структури, які обробляють потоки даних [189]. З-поміж них найпопулярнішими є архітектура зі зворотними зв'язками в ступенях конвеєра, які виконують затримки частини потоків даних, та архітектура з комутаторами на входах ступенів із багатопотоковою затримкою.

В архітектурах першого виду кожен зі ступенів конвеєра містить блок множення на поворотний коефіцієнт, схему r-точкового ДПФ ($r = 2, 4$) та буфер даних, який виконує зворотний зв'язок. Цей буфер заповнюється перед черговим обчисленням малоточкового ДПФ. Тому така структура не може бути повністю завантажена [112, 189].

У другому типі архітектур ступені конвеєра містять буфер даних, r-точковий блок ДПФ і блок множення на поворотний коефіцієнт, які з'єднані послідовно. Буфер даних базується на багатовходовому комутаторі затримки. Він видає набори r комплексних даних, які паралельно подаються в блок ДПФ. Така структура забезпечує максимальну пропускну спроможність завдяки великому обсягу обладнання [54, 157, 189]. Крім того, він має реалізувати ШПФ за малою і єдиною основою r, тому що для ШПФ зі змішаною основою буфери даних стають занадто складними.

Блоки ДПФ, які використовуються в конвеєрних процесорах ШПФ, як правило, проєктуються за відображенням один до одного відповідного ГСПД цього малоточкового ДПФ. Як результат, дані в них мають поступати через r входів. Нехай маємо конвеєрний процесор, у якому два ступені виконані так, щоби число N розкладалося на множники p, r; $p \neq r$. Тоді буфер даних між цими ступенями має складатися з r FIFO завглибшки більше ніж p, які заповнюються з p входів у непрямому порядку. Отже, щоби забезпечити правильний порядок слідування даних для цих ступенів, до їхніх входів треба підключати доволі складні буферні схеми.

Нехай у такі блоки ДПФ r даних поступають послідовно протягом r тактів. Тоді як буфери даних, так і блоки множення на поворотні коефіцієнти суттєво спрощуються. Через таку послідовну дію алгоритм виконання ДПФ у цих блоках має виконуватися з уповільненням в $L = r$ разів. Крім того, у таких блоках апаратні витрати можуть бути зменшені до r разів. Недолік структури таких процесорів ШПФ полягає в зменшенні потенційної пропускної спроможності до r разів. Але в цій ситуації зручніше забезпечити гідну підвищену пропускну спроможність через збільшення кількості процесорів ШПФ, що працюють паралельно. Принаймні, у переважній більшості високопродуктивних систем ЦОС обробка потоків сигналів виконується з тактовою частотою, яка дорівнює частоті дискретизації, що цілком задовольняється в процесорах із такою структурою.

4.2.2 Спосіб проєктування блоків *r*-точкового ДПФ

У цьому підрозділі пропонується спосіб проєктування блоків r-точкового ДПФ за допомогою методу просторового ГСПД, які застосовуються для побудови конвеєрних процесорів ШПФ, у яких дані слідують по одному в кожному такті. Такий спосіб забезпечує проєктування процесорів ШПФ, з мінімізованими апаратними витратами при оптимізованій тактовій частоті [21, 22, 206].

Крім використання просторового ГСПД, спосіб ґрунтується на використанні алгоритму Віноградa обчислення малоточкового ШПФ [171] та спеціалізованих блоків множення на коефіцієнт. Такий блок множення використовує дерево суматорів, яке виконує додавання часткових добутків на коефіцієнт, поданий у канонічній знаковій двійковій системі числення [127].

Спосіб пояснюється на прикладі проєктування блока 3-точкового ДПФ. Цей блок виконує алгоритм Віноградa, який описано у [112]:

$$
\begin{aligned}
&t = x_1 + x_2, \quad p = x_2 - x_1,\\
&m_0 = x_0 + t, \ m_1 = (\cos 2\pi/3 - 1)\cdot t, \ m_2 = j\cdot \sin 2\pi/3\cdot p,\\
&s = m_0 + m_1,\\
&X_0 = m_0, \ \ X_1 = s + m_2, \ X_2 = s - m_2,
\end{aligned}
$$

де $\{x_0, x_1, x_2\}$ — вхідні комплексні дані, $\{X_0, X_1, X_2\}$ — множина комплексних результатів, $j = \sqrt{-1}$. У цьому алгоритмі коефіцієнт $\cos 2\pi/3 = 0.5$, тому $m_1 = 1{,}5 \cdot t$. Алгоритм виконує 12 додавань і 4 множення дійсних чисел.

Для мінімізації множень і збільшення тактової частоти розробляється спеціалізований блок множення з урахуванням мінімізації кількості операцій [134]. Тоді коефіцієнт $k = \sin 2\pi/3 = 0{,}866025 = = 0{,}1101110110110100_2$. Для мінімізації операцій додавання він подається в знаковій канонічній системі, тобто, з розрядами 1, 0, –1. І тоді $k = 1.00\bar{1}0\ 00\bar{1}0\ 0\bar{1}0\bar{1}\ 01$.

Звідси, множення $k \cdot p = \sin 2\pi/3 \cdot p$ виконується як

$$k \cdot p = p - (p + p2^{-4})2^{-3} + ((p2^{-2} - p)2^{-2} - p)2^{-10}.$$

ГСПД цього алгоритму показаний на рис. 4.5. На ньому вершина зі знаком «+» виконує комплексне додавання, символ «*>>l*» означає зсув вправо на *l* розрядів, ребро, яке навантажене *j*, означає множення на уявну одиницю *j,* що означає інверсію уявної частини числа й перестановку дійсної та уявної частин комплексного числа. Вважається, що кожна вершина виконує затримку на

один такт, тобто, їй відповідає АЛП із регістром на виході. Отже, цей ГСПД відповідає структурі блока, який виконує 3-точкове ДПФ із періодом $L = 1$ такт.

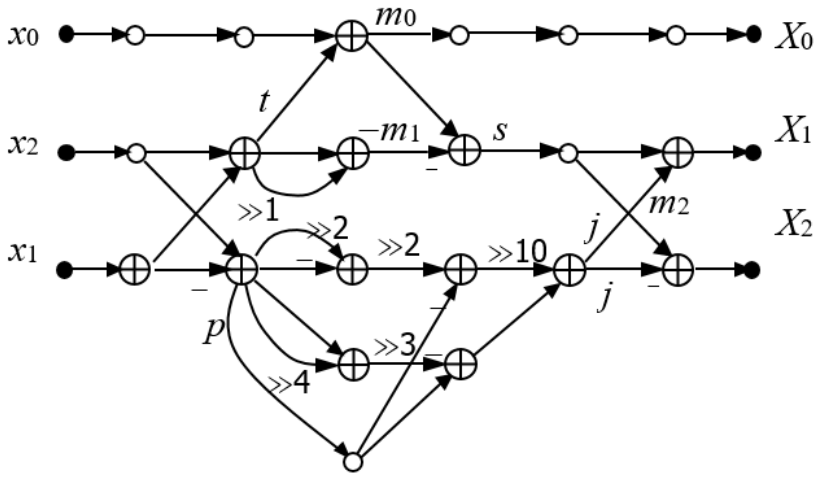

Рисунок 4.5 — ГСПД 3-точкового ДПФ

Згідно з методом просторового ГСПД, вершини графа на рис. 4.5 розміщуються в тривимірному просторі для $L = 3$ у вигляді просторового ГСПД, який показано на SDF, що проілюстровано на рис. 4.6.

Можемо бачити, що просторовий ДПФ кодує як граф алгоритму, так і структуру модуля, де він виконується. Цей ГСПД за методикою, описаною в підрозділі 2.1.4, переводиться в опис VHDL у стилі для синтезу, який подано в Додатку В.

У ньому використовуються сигнали та порти, які представляють виходи відповідних операторних вершин ГСПД на рис. 4.6. Комплексні змінні представлені парою сигналів із суфіксами r та i у їхніх іменах. Вхідний імпульс START синхронізує генератор фаз, який описаний в операторі процесу CNTRL. Він генерує сигнал фази CYC з періодом $L = 3$ такти.

Розрахунки ДПФ проводяться в операторі процесу CALC. Оператор CASE в ньому складається з трьох альтернатив залежно від сигналу CYC. В i-й альтернативі розміщуються оператори, що виконуються в циклі $i = t \bmod 3$,

який вказано на осі *t* на рис. 4.6. Кожне призначення сигналу в цьому процесі відображається у відповідному регістрі конвеєрної схеми. Альтернативи оператора CASE відображаються у відповідні входи мультиплексорів на входах суматорів [41]. Отримана структура блока ДПФ показана на рис. 4.7. Він показаний лише як ілюстрація, оскільки його формування не є необхідним.

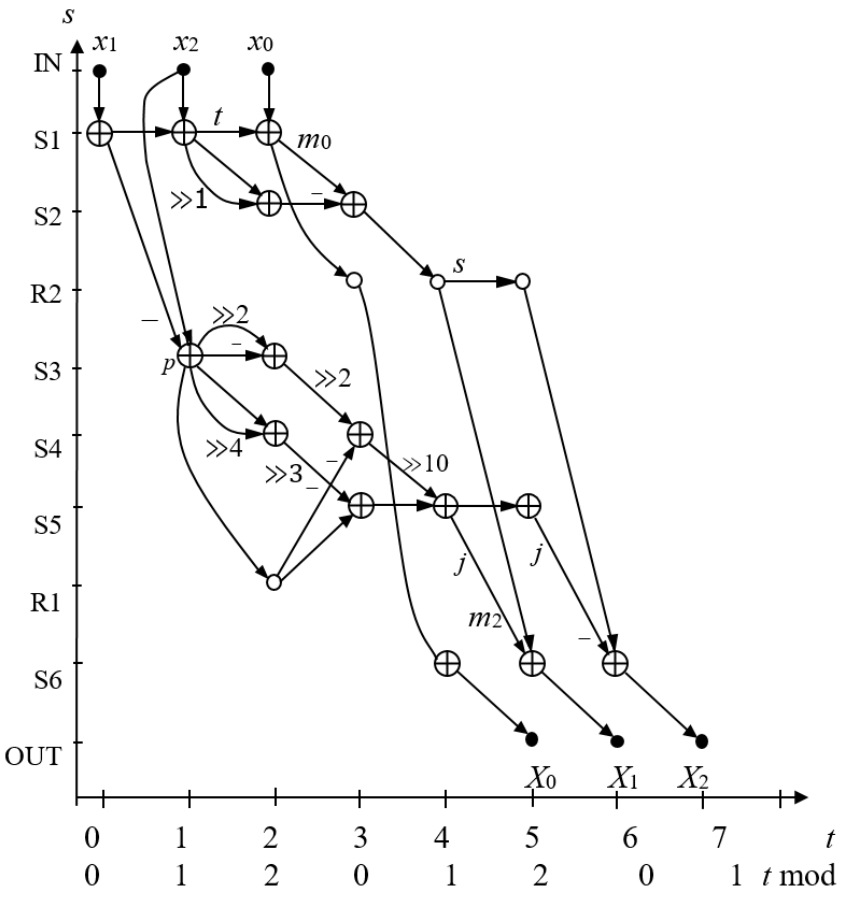

Рисунок 4.6 — Просторовий ГСПД 3-точкового ДПФ

Як видно на рис. 4.7, суматори з двома вхідними мультиплексорами розміщуються між двома сусідніми регістрами цього блока. Один розряд такого суматора з мультиплексорами відображається в єдину 6-вхідну логічну таблицю (ЛТ), яка використовується в сучасних ПЛІС. Тому цей блок має найкоротший критичний шлях і максимальну тактову частоту.

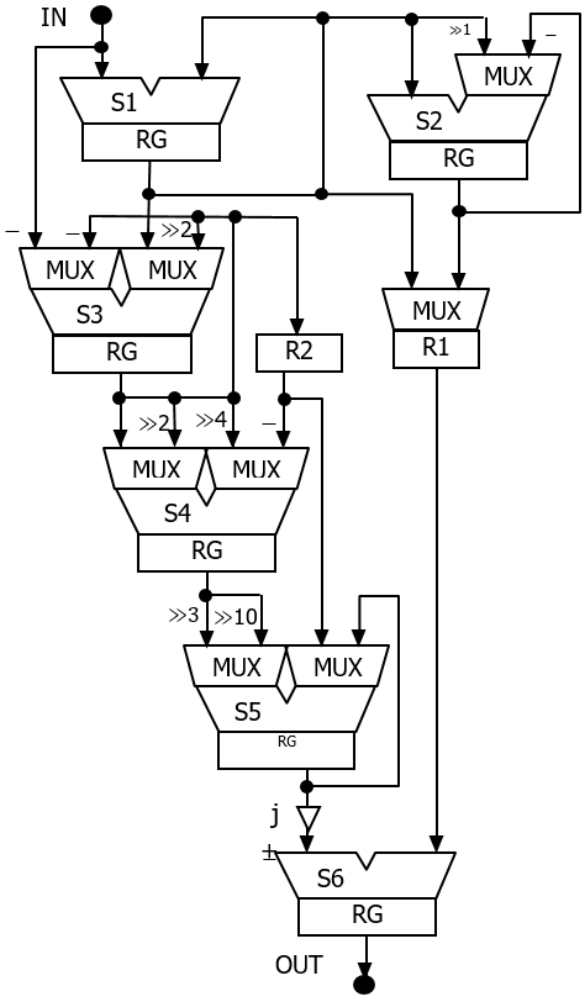

Рисунок 4.7 — Структура модуля ДПФ

4.2.3 Результати синтезу модулів *r*-точкового ДПФ

Набір модулів ДПФ був розроблений із використанням описаного вище способу. Кожен із них виконує ввід по одній вибірці за такт у натуральному порядку, що забезпечує простий спосіб їхнього підключення до системи та синхронізації. Опис мовою VHDL синтезованого процесора для 5-точкового ДПФ подано в Додатку Г.

Крім того, були також синтезовані відповідні буфери перевпорядкування на основі регістрів послідовного зсуву SRL16 у ПЛІС Xilinx. Це допомагає сконструювати модулі ДПФ вищого порядку на основі алгоритму Томаса-Гуда, наприклад, для $r = 15 = 3 \cdot 5$, які мають мінімізований обсяг обладнання.

Результати конфігурування блоків у ПЛІС Xilinx Kintex-7 для 16-розрядних вхідних даних наведені в таблиці 4.5.

Для порівняння ефекту від використання спеціалізованих блоків множення на прикладі синтезу модуля 3-точкового ДПФ у таблиці 4.5 також показано результати проєктування блоку з помножувачами DSP48. Порівняння блоків ДПФ показує, що тактову частоту блока без помножувачів можна збільшити до 1,5 разів. Аналіз таблиці 4.5 також показує, що тактова частота блока зменшується зі збільшенням довжини перетворення *r*. Це пояснюється тим, що відношення затримок маршрутів до критичної затримки шляху в ПЛІС досягає 80% і вище. Тому інструмент розміщення та трасування САПР ПЛІС не здатний ефективно оптимізувати проєкти з великою кількістю взаємозв'язків.

Таблиця 4.5 — Результати конфігурування блоків ДПФ у ПЛІС

Довжина ДПФ, *r*	**Апаратні витрати, ЛТ + DSP48**	**Максимальна тактова частота, МГц**
3	245	640
3	201 + 2	433
4	215	548
5	945	435
8	1187	424
15 = 3·5	2131	346
16	3616	368
64 = 8·8	1985 + 4	338
128 = 8·16	5277 + 4	324

У таблиці 4.6 показано порівняння синтезованого 64-точкового процесора ШПФ з аналогічними процесорами. Його перевагами є невеликі апаратні витрати, мала кількість блоків DSP48, висока тактова частота.

Таблиця 4.6 — 64-точкові процесори ШПФ, сконфігуровані в ПЛІС Xilinx

Серія ПЛІС	**Апаратні витрати, ЛТ + DSP48**	**Максимальна тактова частота, МГц**	**Джерело**
Spartan-3E	758 + 8	170	[173]
	1063 + 12	116	[115]
	1984 + 4	127	Запропонований
Virtex-5	695 + 24	384	[99]
	628 + 4	325	Запропонований

Отже, реалізація малоточкових блоків ДПФ у ПЛІС забезпечує проєктування високопродуктивних конвеєрних процесорів ШПФ з оптимізованим обсягом обладнання. Блок ДПФ, у який дані поступають по одному в кожному такті, синтезований за запропонованим способом, має високу тактову частоту та малий обсяг обладнання за рахунок конвеєрних обчислень, властивостей 6-вхідних LUT та спеціалізованих блоків множення на коефіцієнт. Встановлено, що при $r < 8$ можливий синтез блоку ДПФ у напівавтоматичному режимі, але за більшої складності ДПФ метод ГП є найбільш ефективним [23, 24].

4.3 Проєктування рекурсивних фільтрів

4.3.1 Спосіб розроблення цифрових фільтрів

Традиційна методика розроблення цифрових фільтрів для реалізації в ПЛІС полягає у виконанні наступних кроків. Шукається набір коефіцієнтів

фільтру, який задовольняє специфікацію фільтра, за допомогою відповідного інструмента САПР, наприклад, Matlab. Потім коефіцієнти квантуються, і розраховується передатна характеристика з округленими коефіцієнтами для підтвердження узгодженості зі специфікацією фільтра. Округлені коефіцієнти вбудовуються в налаштовуваний модуль фільтра, який надається постачальником ПЛІС або в модель фільтра, описану мовою VHDL або Verilog. Нарешті, модель фільтра тестується з використанням належного тестового стенда до та після синтезу, розміщення та трасування [75, 127].

Для впровадження цієї методики потрібна спеціальна й небезкоштовна програмна система, така як Matlab. Інший недолік такої методики полягає в необхідності перевірки частотної характеристики після кожного варіанта квантування коефіцієнтів. Крім того, і структура фільтра, і схема округлення результатів визначають цю частотну характеристику, що значно ускладнює перевірку фільтра.

4.3.2 Використання методу генетичного програмування ГСПД для розроблення рекурсивних фільтрів

Насамперед, метод генетичного програмування ГСПД може показати свою ефективність саме в проєктуванні рекурсивних фільтрів. ГСПД рекурсивних фільтрів відрізняються від ГСПД інших цифрових фільтрів чи алгоритмів ЦОС тим, що їхня оптимізація через конвеєризацію, ресинхронізацію, згортання є дуже обмеженою. Це пояснюється тим, що ГСПД рекурсивних фільтрів мають цикли зворотних зв'язків, кількість затримок в яких має бути незмінною при будь-яких кроках оптимізації. У такому випадку, метод оптимізації просторового ГСПД і відповідно, метод його генетичного програмування має низку переваг серед інших методів структурного синтезу рекурсивних фільтрів [16, 207].

По-перше, у переважній більшості методів структуру рекурсивного фільтра одержують відображенням ГСПД один до одного [160]. При цьому фільтр має надлишкові апаратні витрати при помірній тактовій частоті, яка визначається через довжину критичного шляху, який проходить через цикл зворотного зв’язку [16].

По-друге, інші способи відображення виконуються або за допомогою складання розкладу і призначення на ресурси, або методом згортання ГСПД, які були розглянуті в першому розділі.

У цьому випадку, генетичне програмування ГСПД має такі переваги, як те, що цей метод формально виконує таку саму оптимізацію, як ресинхронізація, згортання ГСПД, причому вибір ресурсів, складання розкладу та призначення на ресурси виконуються одночасно, що дає змогу одержувати більш оптимізовані рішення [207]. Загалом, відображення ГСПД алгоритмів зі зворотними зв’язками потребує більш детального вивчення. Далі будуть розглянуті деякі аспекти проєктування рекурсивних фільтрів, які також суттєво впливають на якість остаточного структурного рішення фільтра.

4.3.3 Використання мови VHDL для розроблення цифрових фільтрів

Мова VHDL зазвичай використовується для опису конкретної програми цифрової структури для її конфігурування в ПЛІС. Але її спроможність математичної обробки даних недооцінена. Бібліотека IEEE містить пакети MATH_REAL та MATH_COMPLEX. Вони складаються з типів, констант і функцій із рухомою комою, які є зручними для складної обробки дійсних чисел. Ці пакети зазвичай використовуються для проєктування тестового стенда проєкту VHDL. Але вони мають багато можливостей для вивчення алгоритмів ЦОС. Також на цих пакетах базуються ефективні підпрограми для розв'язання системи лінійних рівнянь та ДПФ. Таким чином, можливості мови

VHDL для математичної обробки та моделювання наближаються до можливостей пакету Matlab [41].

Для аналізу та синтезу цифрового фільтра зазвичай використовується комплексна змінна $Z = e^{j\varphi}$. Для отримання цієї змінної у VHDL можна використовувати таку функцію кута φ:

```
function Z(fi: real) return COMPLEX_POLAR is begin
        return exp(COMPLEX_TO_POLAR(MATH_CBASE_J)*fi);
end Z;
```

Задавши цю функцію, можна описати складну функцію передачі фільтра. Наприклад, функція передачі рекурсивних фільтрів низьких частот на основі фазового фільтра, яка подана в математичній формі:

$$H(z) = z^{-1} + \frac{a + a(1 + b)z^{-1} + z^{-2}}{1 + a(1 + b)z^{-1} + az^{-2}}; \qquad (4.2)$$

описується на VHDL як функція:

```
function LPF(a, b, fi: real) return COMPLEX_POLAR is
        begin
        return Z(-fi) +
                (COMPLEX_TO_POLAR(COMPLEX'(a,0.0)) +
                a(1+b)*Z(-1.0*fi) + Z(-2.0*fi))/
                (COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0)) +
                a(1+b)*Z(-1.0*fi) + a*Z(-2.0*fi)) ;
end LPF;
```

Цей фільтр характерний тим, що його параметри регулюються зміною незалежних змінних [89, 191]:

$$a = \frac{1 - \operatorname{tg} d_f}{1 + \operatorname{tg} d_f};\ b = -\cos f_C \cdot (1 + a), \qquad (4.3)$$

де d_f — ширина перехідної смуги, f_C — ширина смуги пропускання.

Обчислити графік функції передачі (4.1) у частотному просторі в симуляторі VHDL можна, використовуючи такий оператор процесу:

```
process(CLK)
      variable p, phas: real :=0.0;
      variable Hz: COMPLEX_POLAR;
begin
       a <= 1.5;  b <= 0.64;       -- коефіцієнти фільтра
      if CLK='1' and CLK'event then
           phas := phas + 0.001; -- лічильник фази (частоти)
           p := phas * MATH_PI * 2.0;  -- нормалізована фаза
           ph <= phas;                 -- сигнал частоти
       end if;
       Hz := LPF(a, b, p);                    --  H(z)
       Mag <= abs(Hz);                  -- амплітуда H(z)
       Phase <= Hz.ARG;                 -- фаза  H(z)
       Logm <= 20.0*log10(abs(Hz));     -- H(z) у децибелах
end process;
```

Цей процес генерує графіки амплітудно-частотних, логарифмічних та фазових характеристик у симуляторі VHDL протягом кількох тисяч тактів.

Перевірені коефіцієнти фільтра вводяться в модель фільтра, описану на VHDL. Цю модель можна протестувати до та після синтезу за допомогою тестового стенда, такого, як описаний у [205]. За допомогою цього тестування згенеровані дійсна та уявна компоненти аналітичного сигналу подаються на входи двох примірників одного фільтра. Частота аналітичного сигналу змінюється лінійно й таким чином, через встановлену затримку вимірюється модуль і фаза сигналу на виході фільтрів.

4.3.4 Рекурсивні фільтри без блоків множення

Рекурсивні фільтри забезпечують меншу складність та більшу ефективність фільтрації в порівнянні з фільтрами зі скінченною імпульсною характеристикою (СІХ). Але вони менше використовуються в системах ЦОС на ПЛІС через збільшення розрядності даних та обмежену пропускну спроможність. Максимальна частота рекурсивних фільтрів обмежена довжиною критичного шляху, яка оцінюється затримкою зворотного зв'язку

фільтра. Цю затримку не можна звести до мінімуму за допомогою конвеєризації, яка зазвичай використовується для СІХ-фільтрів [127].

Одним з ефективних методів прискорення рекурсивного фільтра в ПЛІС є спрощення множення за допомогою заміни апаратного блока множення на набір суматорів, які додають зсунуті множені [127, 162] або завдяки збереженню кратних коефіцієнтів в ЛТ [201]. Такі фільтри називають фільтрами без блоків множення (multiplier-less). Сучасні ПЛІС містять 6-вхідні ЛТ, які забезпечують реалізацію одноступеневої мережі тривхідних суматорів [77]. У цій ситуації бажано представляти коефіцієнти фільтра як раціональні числа в канонічній знаковій системі числення:

$$c = k2^p + l2^q + m2^r, \tag{4.4}$$

де p, q, r — цілі числа, $k, l, m \in \{0, 1, -1\}$. Отже, такі фільтри, у яких коефіцієнти представляються у формі (4.4), мають найкраще відношення пропускна спроможність — апаратні витрати.

4.3.5 Спосіб пошуку коефіцієнтів рекурсивного фільтра

Вказані фільтри без блоків множення шукаються за наступним способом. За допомогою традиційних методів, наприклад, використовуючи Matlab, знаходяться реальні значення коефіцієнта C. Потім їхні наближені значення, які представлені у формі (4.4) шукають біля точки рішення C, що забезпечує оптимальну реалізацію множення на коефіцієнт. Цей пошук можна виконувати методом сканування простору рішень [9] або методом еволюційної оптимізації [162, 175]. Але для звичайних рекурсивних фільтрів не завжди вдається одержати коефіцієнти за цим способом за умови високоякісної передатної функції. По-перше, високоякісні рекурсивні фільтри дуже чутливі до точності представлення коефіцієнтів, по-друге, такі фільтри можуть бути нестійкими, якщо округлити знайдені реальні коефіцієнти.

Рекурсивні фільтри на основі фазових фільтрів, на відміну від інших фільтрів, можуть мати коефіцієнти у формі (4.4) [215]. Такий фільтр має передатну характеристику

$$H(z) = (A_1(z) + A_2(z))/2, \tag{4.5}$$

де $A_1(z)$, $A_2(z)$ — передатні функції фазових фільтрів. Своєю чергою, фазові фільтри другого порядку описуються такою передатною функцією:

$$H_i(z) = \frac{b_i + c_i z^{-1} + z^{-2}}{1 + c_i z^{-1} + b_i z^{-2}}. \tag{4.6}$$

У фільтрі з передатною функцією (4.3) $A_1(z) = z^{-1}$, $A_2(z)$ — дріб, який дорівнює (4.6) при відповідному переобчисленні коефіцієнтів. Такі фільтри є стабільними за будь-яких коефіцієнтів та є нечутливими до варіацій коефіцієнтів. Остання властивість забезпечує успішний пошук коефіцієнтів у формі (4.4). Навіть більше, ця нечутливість збільшується в спеціально підібраних різновидах ГСПД такого фільтра[125]. Так, у роботі [238] з-поміж 24 різних ГСПД фільтрів (4.6) вибрано граф Стоянова-Кавамата, який забезпечує мінімальну кількість розрядів у коефіцієнтах при високих параметрах передатної характеристики.

Ступінь фільтра, який обчислює функцію (4.6), який має максимальну продуктивність, описується ГСПД, що показано на рис. 4.8. Аналіз цього ГСПД показує, що відповідна структура фільтра має лише два блоки множення, повністю конвеєризована, критичний шлях проходить через один суматор і помножувач на коефіцієнт *c*. Отже, найбільша продуктивність досягається тоді, коли коефіцієнт *c* має мінімальну кількість одиниць у поданні (4.4) або дорівнює нулю [19].

Коефіцієнти у формі (4.4) треба шукати за допомогою програми VHDL. Ітерація такої програми полягає в здійсненні наступних кроків. По-перше,

набір коефіцієнтів *b*, *c* вибирається з таблиці коефіцієнтів, рівних (4.4). Потім ці коефіцієнти заносяться у формулу (4.5), яка оцінюється за допомогою оператора процесу, показаного в підрозділі 4.3.2. Нарешті, результати порівнюються зі специфікацією фільтра і зберігається набір ефективних коефіцієнтів. Після декількох ітерацій цього алгоритму вибирається оптимальний набір коефіцієнтів фільтра. Отриманий набір коефіцієнтів заноситься у VHDL-опис фільтра.

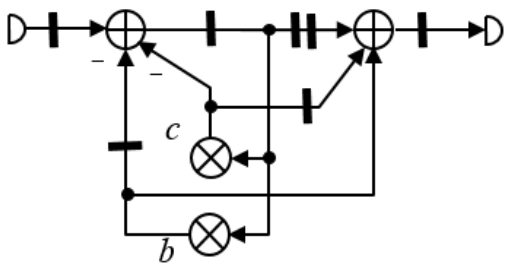

Рисунок 4.8 — ГСПД фазового фільтра з максимальною швидкодією

Заміна блоку множення на схему множення на коефіцієнт є ефективною не тільки в рекурсивних фільтрах, але й в інших спеціалізованих пристроях ЦОС. Так, у роботі [20] пропонується суттєво зменшити апаратні витрати також у нерекурсивних фільтрах завдяки поданню коефіцієнтів у формі (4.4). Для цього набір їхніх коефіцієнтів розділяється на дві групи: ті, що помножуються на апаратних блоках множення, та ті, що подаються у формі (4.4) і на які виконується множення, використовуючи тривходові суматори. При цьому приблизно вдвічі зменшується кількість блоків множення.

4.3.6 Еволюційний синтез коефіцієнтів рекурсивного фільтра

Коефіцієнти фільтра у формі (4.4) утворюють складений вектор $\boldsymbol{S} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_i, \ldots, \boldsymbol{s}_n)$, який належить до багатовимірного простору рішень, де $\boldsymbol{s}_i = (k_i, l_i, m_i, p_i, q_i, r_i)$ — кількість коефіцієнтів. Оптимізація фільтра полягає в

знаходженні деякого глобального екстремуму функції якості Ф(***S***). Ця функція має обчислювати рівні пульсації в смугах пропускання та непропускання передатної функції $H(z)$. Він також має враховувати кількість нульових параметрів k_i, l_i, m_i, оскільки це мінімізує апаратну складність фільтра.

Якщо $n > 3$, то пошук коефіцієнтів у формі (4.4) за методом сканування простору рішень стає нераціональним. Тоді можна застосувати метод модельованого відпалювання, який є різновидом еволюційного програмування, коли вся популяція, яка складається з однієї особини, є шуканим рішенням. Згідно з [128], він моделює процес дифузії молекул, координати яких представлені вектором ***S***. Під час оптимізації модель фізичного тіла з цими молекулами спочатку нагрівається до температури відпалу, а потім охолоджується. Ефект цього методу не залежить від фізичної природи оптимізованого об'єкта. Для пошуку коефіцієнтів приймаються такі аналогії [236]: стан тіла — вектор рішень ***S***, функція якості Ф(***S***) означає його енергію; зміна стану тіла — це заміна ***S*** на наступне рішення $\boldsymbol{S}_n$; температура t — окремий параметр, що знижується при оптимізації, кінцева температура $t_{\min}$ — точка зупинки оптимізації.

Узагальнений алгоритм модельованого відпалу такий:

```
Вибираються початкове рішення Sn,
температура t
коефіцієнт остигання a;
repeat{
    вибирається випадкове рішення S,
                розташоване в околі Sn;
 ΔФ=Ф(S) – Ф(Sn);//декремент енергії
    if ΔФ<0 then
          Sn = S;
    else {
       генерується випадкове число p;
         if p<exp(ΔФ/t)  then
           Sn = S;
        }
      t=t*a;
 }
 until t<tmin;
```

Цей алгоритм був реалізований у програмі мовою VHDL з використанням методу розроблення рекурсивних фільтрів, описаних вище. Для спрощення процесу пошуку вектор $\boldsymbol{s}_i$ кодується за його початковим зображенням c_i, обчисленим за формулою (4.3). Усі можливі значення c_i відбираються в постійному запам'ятовувальному пристрої (ПЗП), тобто $c_i = f(a_i)$, і рішення кодується як $\boldsymbol{S}' = (a_1, \ldots, a_i, \ldots, a_n)$. Наприклад, для $p < q < r < 8$ ПЗП кодується як

```
constant ints1:Tarr2:=
      (0,1,2,3,4,5,6,7,8,9,10,12,14,15,16,17,18,20,24,28,30,
      31,32,33,34,36,40,48,56,60,62,63,64,
      65,66,68,72,80,96,112,120,124,126,127);
```

Початкове рішення $\boldsymbol{S}'_n$ задається для набору коефіцієнтів, які є близькими до деякого точного рішення, одержаного, скажімо, у програмі Matlab. Вони також можуть бути обрані довільно. Випадковий розв'язок $\boldsymbol{S}'$ одержується з попереднього значення за допомогою додавання деякого випадкового вектора $(\delta_1, \ldots, \delta_i, \ldots, \delta_n)$. Його позитивні та негативні елементи змінюються в деякому діапазоні, який звужується зі зменшенням t. Функція $\Phi(\boldsymbol{S}')$ обчислюється як сума пульсацій функції $|H(\boldsymbol{S}', z)|$ плюс параметр, який пропорційний кількості одиниць у поданні коефіцієнтів (4.4) [211]. Програма для генерації коефіцієнтів фільтрів за цим способом подана в Додатку Е.

На рис. 4.9 показана позиція елементів вектора $\boldsymbol{S}$, який представляє коефіцієнти *b, c* фільтра другого порядку до й після оптимізації. Параметри *p, q, r* розкладання цих коефіцієнтів представлені точками у двовимірному просторі рішень. Спочатку ці точки хаотично переміщуються в просторі з векторами переміщення і наприкінці займають якесь випадкове стабільне місце, яке відповідає оптимальному значенню коефіцієнтів.

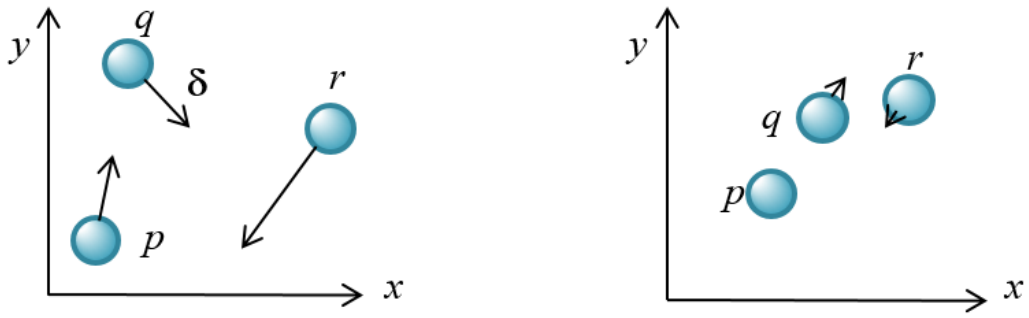

Рисунок 4.9 — Представлення коефіцієнтів *b*, *c* до й після оптимізації

4.3.7. Експериментальні результати

Описаний вище метод був використаний для побудови набору рекурсивних фільтрів порядку від 5 до 9. Ці фільтри були поміщені в базу даних вебзастосунку IIR Filter Generator [204, 209]. Ця програма генерує синтезовану модель VHDL фільтра із заданою розрядністю входу та виходу, та шириною смуги пропускання. У Додатку Д подано приклад фільтра, згенерованого цією програмою.

На рис. 4.10 показані частотні відгуки H_1, H_2 двох напівсмугових фільтрів та їхнього послідовного з'єднання $H_P = H_1 \cdot H_2$. Для напівсмугового фільтра частота зрізу дорівнює $0{,}25f_S$, і тому коефіцієнти $c_i = 0$, де f_S — частота дискретизації. Тому цей фільтр має лише 2 ненульових коефіцієнти, які не складно знайти простим скануванням простору рішень.

Параметри синтезованого фільтра наведені в таблиці 4.7. Тут апаратні витрати наведено в кількості вирізок конфігурованих логічних блоків (CLBS). У таблиці 4.7 наведено результати синтезу іншого напівсмугового фільтра без блоків множення з подібними властивостями, структура якого взята з роботи [237]. Він має значно гірші характеристики через те, що його ГСПД має довший критичний шлях.

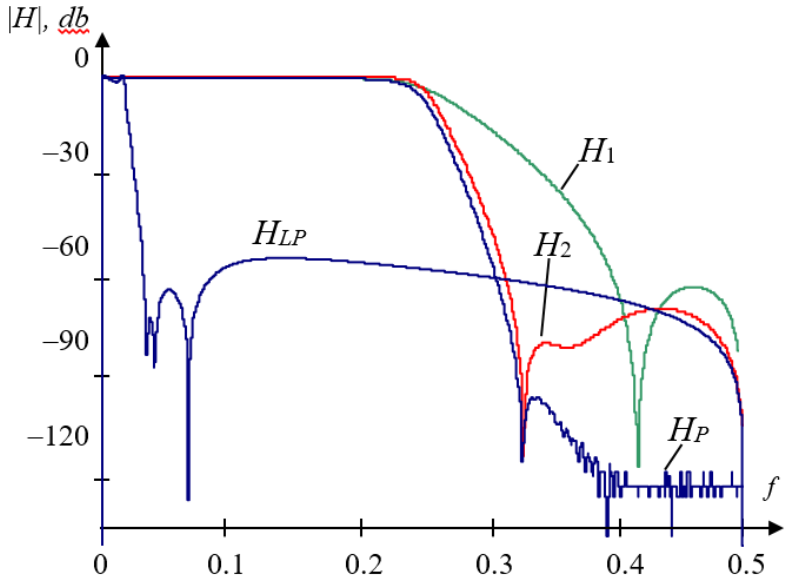

Рисунок 4.10 — Передатні характеристики синтезованих напівсмугового фільтра (H_P) та фільтра нижніх частот (H_{LP})

Іншим прикладом є синтез фільтра нижніх частот із частотою зрізу 0,025 f_S. Структура фільтра відповідає (4.5), де $A_1(z)$, $A_2(z)$ — передатні функції фазових фільтрів 3-го та 4-го порядку відповідно.

Таблиця 4.7 — Параметри синтезованих фільтрів, реалізованих у ПЛІС Kintex та їхнє порівняння з аналогами

Тип фільтра	Апаратні витрати, CLBS	Максимальна тактова частота, МГц	Рівень придушення, дБ	Посилання
Напівсмуговий	203	690	120	—
Напівсмуговий	441	107	106	[237]
Нижніх частот	179	310	54	—
Нижніх частот	203	189	57	[238]

Коефіцієнти, які знайдені методом модельованого відпалювання, реалізованого у VHDL-програмі, дорівнюють

$$c_0 = -1.00\bar{1}01;\ b_1 = 1.00\bar{1}01;\ c_1 = -10.0000\bar{1};$$
$$b_2 = 1.00\bar{1}0\bar{1};\ c_2 = -10.00\bar{1};\ b_3 = 1.0000\bar{1}01;\ c_3 = -10.0000\bar{1}00\bar{1}.$$

Отримана в результаті функція передачі фільтра нижніх частот показана на рис. 4.10, а його характеристики наведені в таблиці 4.7. Цей фільтр має менший обсяг обладнання, значно більшу частоту та приблизно рівний рівень придушення, у порівнянні з аналогічним фільтром, показаним у [238]. Це пояснюється тим, що ГСПД аналогічного фільтра має набагато довший критичний шлях через складність схеми Стоянова-Кавамата, яка в ньому реалізована.

Отже, у цьому підрозділі показано, що мова VHDL дає змогу інженерам розробляти цифрові фільтри, не виходячи за межі редактора та VHDL-симулятора. Таким чином, можливість швидкого моделювання складного процесу оптимізації забезпечує ефективний пошук оптимальних структурних рішень рекурсивних фільтрів. На відміну від використання таких поширених інструментів проєктування, як Matlab, VHDL-симулятор забезпечує керування схемою фільтра, враховуючи як його коефіцієнти, так і розрядність даних та спосіб здійснення арифметичної операції, а також особливості його конфігурування в ПЛІС.

Було доведено, що якщо коефіцієнти фільтра, що не містить блоків множення, мають у своєму поданні не більш як три ненульові розряди, то його конвеєрна реалізація в ПЛІС має найвищу тактову частоту. На прикладах конструкції рекурсивних фільтрів без блоків множення показано ефективність використання мови VHDL.

Спосіб розроблення рекурсивних цифрових фільтрів із застосуванням властивості фазових фільтрів бути динамічно переналаштованими був

використаний для моделювання хвилевих процесів поширення ультразвуку у твердому тілі [31].

4.4 Обчислення синусоїдальних функцій у ПЛІС

В алгоритмах ЦОС дуже поширене обчислення синусоїдальних функцій, які необхідні, наприклад, в алгоритмах ДПФ, змішувачах сигналів. Існує багато методів обчислення тригонометричних функцій у FPGA. Один із найбільш використовуваних методів ґрунтується на табличних функціях [96]. Метод обчислення інтерполяційного полінома потребує багатьох додавань та множень даних із великою пропускною спроможністю [96, 109]. Три десятиліття метод CORDIC (coordinate rotation digital computer) широко застосовується в FPGA для обчислення різних тригонометричних функцій [38, 96, 152]. Єдиний недолік цього способу полягає в тому, що для n точних розрядів даних алгоритм виконує n послідовних ітерацій. Як результат, алгоритм CORDIC має тривалу приховану затримку обчислень. У цьому підрозділі пропонується новий метод обчислення тригонометричних функцій в FPGA, який базується на властивостях піфагорових трійок.

4.4.1 Піфагорові трійки та їхні властивості

Розв’язання задачі Піфагора полягає в знаходженні всіх прямокутних трикутників зі сторонами, що виражаються цілими числами a, b, c, тобто, у розв’язанні діофантового рівняння [223]:

$$a^2 + b^2 = c^2. \tag{4.7}$$

Воно являє собою рівняння кола з радіусом c, для якого виконується таке співвідношення:

$$\sin^2 \varphi + \cos^2 \varphi = 1, \tag{4.8}$$

де φ = atan(a/b). З рівнянь (4.7), (4.8) видно, що для кутів φ значення тригонометричних функцій дорівнюють раціональним дробам:

$$\sin\varphi = a/c; \quad \cos\varphi = b/c; \quad \tan\varphi = a/b. \tag{4.9}$$

Тому можна отримати точні значення тригонометричних функцій (4.9), які виражаються відповідною піфагоровою трійкою (a, b, c) для заданого кута φ’. Проблема полягає в тому, як знайти трійку (a, b, c), яка забезпечує задовільну похибку ε подання кута. Задачу Піфагора можна розв’язати одним із методів [223]. Усі вони є комбінаторними. Так, Евклід запропонував формулу для знаходження піфагорової трійки для будь-яких двох натуральних чисел m і n, $m > n$, а саме:

$$a = m^2 - n^2; \qquad b = 2mn; \qquad c = m^2 + n^2. \tag{4.10}$$

Виконуючи перебір m і n, можна знайти таку пару m, n, яка задовольняє обмеження $\varepsilon \le |\varphi' - \varphi|$, де φ = atan($a/b$). Область пошуку m, n мінімізується за допомогою формули

$$\frac{n}{m} \approx \frac{\sqrt{r^2+1}-1}{r}, \tag{4.11}$$

де r = tan φ’. Через співвідношення (4.10), (4.11) обчислення функцій синуса та косинуса за допомогою піфагорової трійки є складнішим, ніж звичайні методи. Але має сенс сформувати таблицю з трійок, які відповідають різним кутам. Для малих кутів $\varphi \approx \tan\varphi \approx 1/n$ відомі формули одержання трійок, які залежать лише від одного аргументу:

$$a = 2n + 1; \quad b = 2n^2 + 2n; \quad c = 2n^2 + 2n + 1; \tag{4.12}$$

$$a = 4n; \qquad b = 4n^2 - 1; \qquad c = 4n^2 + 1. \tag{4.13}$$

Якщо розглянути цілі значення a, b зі знаком, то отримаємо узагальнені піфагорійські трійки [87]. Для таких трійок визначена операція додавання

кута, яка імітує множення комплексних чисел. Розглянемо дві трійки (a_1,b_1,c_1), (a_2,b_2,c_2) з кутами φ_1, φ_2, відповідно. Тоді нова трійка має кут $\varphi = \varphi_1 + \varphi_2$:

$$(a, b, c) = (a_1,b_1,c_1) \oplus (a_2,b_2,c_2) = (a_1b_2 + b_1a_2 , b_1b_2 – a_1a_2, c_1c_2). \quad (4.14)$$

Узагальнені піфагорові трійки разом з операцією додавання кута утворюють абелеву групу [223].

4.4.2 Використання піфагорових трійок в обчисленнях

Рівняння (4.9) є точними значеннями тригонометричних функцій. Отже, ці значення можна використовувати в комп'ютерах без помилок подання даних. Ця особливість піфагорових трійок використовується в деяких комп’ютерних графічних інструментах [87]. У зв'язку з тим, що точні значення синуса та косинуса (4.9) задаються трійками коротких цілих чисел, таблиці цих функцій мають мінімізований обсяг.

Деякі спецпроцесори, які сконфігуровані в ПЛІС, використовують подання даних у вигляді раціональних дробів. Раціональний дріб *a/b* — це числовий об'єкт із цілими числами в чисельнику та знаменнику. Раціональні дроби забезпечують простий набір арифметичних операцій. Множення й ділення *a/b* на *c/d* дорівнює *ac/*(*bd*), та *bd/*(*ac*), відповідно. Додавання їх дорівнює (*ad*+*bc*)/(*bd*). За точністю подання даних, раціональні дроби займають місце між цілими числами та числами з рухомою комою.

Розроблено набір процесорів для розв’язування задач лінійної алгебри, що показало високу ефективність подання даних раціональними дробами [155]. ДПФ ґрунтується на множенні на поворотні коефіцієнти. Коли ці коефіцієнти базуються на піфагорових трійках, помилки обчислення ДПФ суттєво мінімізуються [27].

Недолік представлення тригонометричних функцій рівностями (4.9) полягає в складній процедурі пошуку правильного піфагорового трикутника

для заданого кута φ’. Нижче показаний метод, який спрощує пошук цього трикутника і який реалізований в FPGA.

4.4.3 Обчислення піфагорових трійок в FPGA

Алгоритм, реалізований в FPGA, не може залежати від вхідних даних. Якщо алгоритм має низку етапів, то їхня кількість має бути заздалегідь визначена. Тому алгоритми, які ґрунтуються на комбінаторному пошуку, зазвичай не відповідають реалізації в FPGA. Тому, алгоритм пошуку піфагорових трійок не може бути комбінаторним у разі його реалізації в ПЛІС.

Тому пропонується триетапний алгоритм пошуку піфагорових трійок [203]. Він ґрунтується на виконанні додавання кутів (4.14). Заданий кут φ’ представляється сумою кутів $\varphi' = \varphi'_1 + \varphi'_2$. На першому етапі шукають трійку (a_1,b_1,c_1) для якої кут $\varphi_1 = \varphi'_1 + \delta_{\varphi 1}$ відрізняється від заданого значення φ'_1 на різницю $\delta_{\varphi 1}$. На другому етапі шукають трійку (a_2,b_2,c_2) для кута $\varphi_2 = \varphi'_2 - \delta_{\varphi 1}$. Шукана піфагорова трійка обчислюється на третьому етапі за формулою (4.14). Вона являє собою кут $\varphi = \varphi_1 + \varphi_2 = \varphi'_1 + \varphi'_2 + \delta_{\varphi 2}$, де $\delta_{\varphi 2}$ — помилка подання кута φ’.

Кут φ'_1 представляється старшими розрядами коду кута φ’, а кут φ'_2 — молодшими розрядами. Тоді перший етап алгоритму може бути реалізований у ПЗП, для якого код φ'_1 є кодом адреси. Нехай молодший розряд коду φ'_2 дорівнює $\pi/2^{15}$, тобто, $\delta_{\varphi 2} < \pi/2^{16}$ і ми враховуємо, що $n \approx 1/\tan \varphi_2$. Тоді другий етап алгоритму можна обчислити за формулою (4.12), коли $\varphi_2 < \pi/512$ і за формулою (4.13), коли $\varphi_2 < \pi/1024$.

Структура модуля, який здійснює пошук піфагорової трійки для кутів $0 < \varphi' < \pi/4$, показаний на рис. 4.11. Код вхідної фази зберігається в регістрі RGP. Старші розряди коду вибирають піфагорову трійку (a_1,b_1,c_1) та значення похибки кута $\delta_{\varphi 1}$ в ПЗП ROM1. Скориговане значення φ_2 кута φ'_2, яке задається молодшими розрядами регістра RGP, формується суматором SM1 і є адресою

вибірки другої піфагорової трійки (a_2,b_2,c_2) у ПЗП ROM2. Блоки множення MPU і суматори SM1, SM2 обчислюють формулу (4.15).

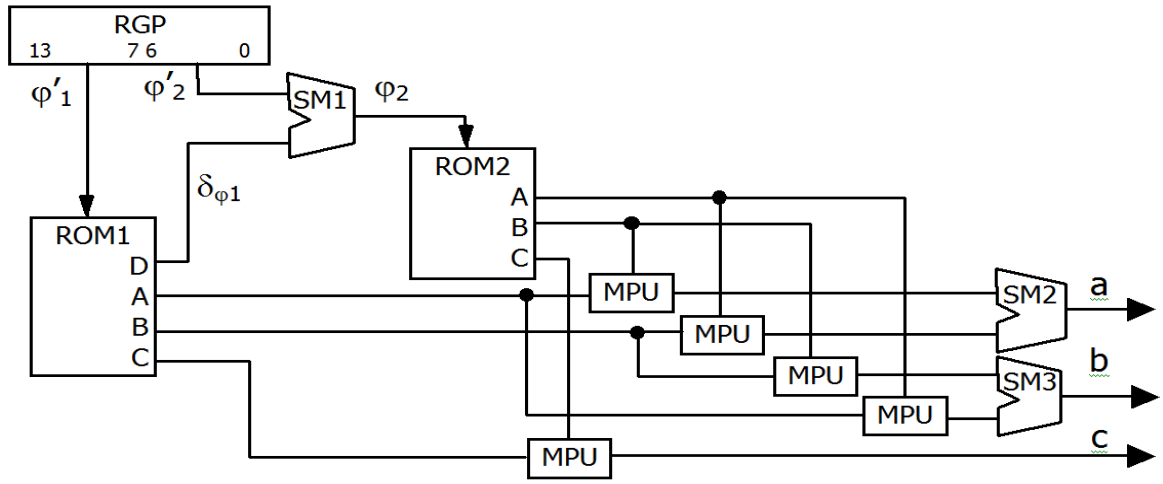

Рисунок 4.11 — Структура обчислювача піфагорових трійок

Як приклад, розглянемо кут φ’ = π/6. Він представляється кодом $85 \cdot 2^7+43$ в RGP. За адресою 85 з ROM1 зчитується трійка (120,209,241) і код помилки — 7. Код для вибору другої трійки обчислюється як $\varphi_2 = \varphi'_2 - \delta_{\varphi 1} = 43+7 = 50$. Цей код представляє кут 0,0024. Тоді коефіцієнт n дорівнює n =]1/tan(0.0024)[= 417. Згідно з формулою (4.13), друга піфагорова трійка дорівнює (835, 348612, 348613), і вона читається з ROM2 за адресою $\varphi_2 = 50$.

Результівна піфагорова трійка, яка обчислена за формулою (4.14), дорівнює (42007955, 72759708, 84015733). Вона представляє кут π/6 з похибкою 1.47610^{-5} радіанів, що дорівнює 0,31 від молодшого розряду вхідного числа. За необхідності, реальне значення синуса й косинуса цього кута обчислюється через ділення за формулою (4.9).

Структура обчислювача на рис. 4.11 реалізована в ПЛІС Xilinx Artix XC7A20SL і займає 235 ЛТ і 8 блоків множення DSP48. Максимальна тактова частота досягає 135 МГц. Ця частота може бути набагато вищою, коли застосовується конвеєризація. Якщо потрібні результати з фіксованою комою,

необхідно приєднати до виходу блок ділення, обсяг апаратного забезпечення якого становить чотири сотні ЛТ для 16-розрядного результату. Для порівняння, модуль CORDIC з еквівалентною точністю має вчетверо більший обсяг апаратного забезпечення та латентну затримку, що на порядок перевищує затримку цього пристрою [152].

Отже, підхід із використанням піфагорових трійок забезпечує простий метод отримання точних значень тригонометричних функцій. Запропоновано новий алгоритм обчислення піфагорових трійок, який базується на триетапному алгоритмі. Модуль, який реалізує цей алгоритм і сконфігурований у FPGA, має порівняно невеликі апаратні витрати й може обчислити піфагорову трійку для заданого кута за один такт. Цей модуль може використовуватися в спецпроцесорах як для ЦОС, так і розв'язання задач лінійної алгебри та в інших галузях.

4.5 Висновки до четвертого розділу

1. Встановлено, що метод ГП просторового ГСПД дає змогу автоматизовано знаходити оптимізовані рішення при проєктуванні процесорів для обробки цифрових сигналів з високою пропускною спроможністю.

2. Реалізація малоточкових блоків ДПФ у ПЛІС забезпечує проєктування високопродуктивних конвеєрних процесорів ШПФ з оптимізованим обсягом обладнання (до 28–70% менший). Блок ДКП, синтезований за запропонованим методом, має до 2–50% більшу тактову частоту та до 18–40% менший обсяг обладнання у порівнянні з аналогами. Він має відношення продуктивності до апаратних витрат до 9–160% більше, ніж у аналогів. Встановлено, що при невеликому ГСПД синтез блоку дискретних перетворень можливий у напівавтоматичному режимі, але за більшої складності ГСПД метод ГП є найбільш ефективним.

3. Запропоновано новий спосіб проєктування рекурсивних фільтрів за рахунок запропонованого способу множення на коефіцієнти, завдяки чому вони мають обсяг обладнання на 12–50% менший, частоту в 0,5–5,5 разів більшу та приблизно рівний рівень придушення, у порівнянні з аналогічними фільтрами.

4. Показано, що підхід із використанням піфагорових трійок забезпечує простий метод обчислення точних значень тригонометричних функцій у ПЛІС, який використовується в ЦОС, і може бути реалізований з фіксованою комою, що значно простіше, ніж при використанні плаваючої коми подвійної точності.

ВИСНОВКИ

У цій дисертаційній роботі вирішується задача підвищення ефективності проєктування конвеєрних обчислювальних систем на основі ПЛІС.

1. Проаналізовано завдання, алгоритми і пристрої ЦОС. Сформульовано критерії оптимізації для структур, які синтезуються, та засобів проєктування обчислювальних систем для ЦОС.

2. Проаналізовано алгоритмічні моделі та мови опису алгоритмів ЦОС, методів і засобів їхнього відображення в паралельні обчислювальні системи відповідно до сформульованих вимог до них. Вибрано просторовий ГСПД та метод його відображення як найбільш придатну модель та метод відображення для подальшого дослідження.

3. Вперше запропоновано метод проєктування спеціалізованих конвеєрних структур на основі генетичного програмування, який відрізняється тим, що алгоритм цифрової обробки сигналів, який відображається в структуру, задається просторовим ГСПД, задача мінімізації апаратних витрат вирішується із заданими часовими обмеженнями за допомогою еволюційного підходу, який ґрунтується на поданні хромосоми як закодованого ГСПД та відповідних функціях її зміни, а також двохетапному алгоритмі оптимізації. Запропонований метод дає змогу формалізовано вирішувати задачу синтезу обчислювальних систем для цифрової обробки сигналів і завдяки регулюванню ступеня розпаралелювання алгоритму та мінімізації апаратних витрат одержані структури мають високе співвідношення продуктивність — вартість. Розроблено три алгоритми генетичного програмування, які відрізняються процедурами селекції — алгоритм із пропорційним добором Qvalue, алгоритм Монте-Карло Roulette та алгоритм Монте-Карло з нішами Roulette+N.

4. На основі запропонованого методу розроблено засоби автоматизації відображення алгоритмів ЦОС в обчислювальні системи на основі ПЛІС. Розроблена система SDFCAD, яка реалізує новий метод, здатна виконувати ввід алгоритмів, заданих ГСПД, й автоматично виконувати синтез конвеєрних процесорів, які реалізують цей алгоритм за допомогою генетичного програмування.

5. Перевірено ефективність розробленого методу під час проєктування низки спеціалізованих обчислювальних систем для вирішення широкого кола завдань ЦОС. Для цього виконано низку дослідів із синтезу конвеєрних процесорів для різних алгоритмів ЦОС. Реалізація малоточкових блоків ДПФ у ПЛІС забезпечує проєктування високопродуктивних конвеєрних процесорів ШПФ з оптимізованим обсягом обладнання (до 28–70% менший). Блок ДКП має до 2–50% більшу тактову частоту та до 18–40% менший обсяг обладнання, а також відношення продуктивності до апаратних витрат до 9–160% більше, ніж у аналогів. Розроблений генератор функцій синуса та косинуса на основі піфагорових трійок відрізняється високою точністю генерованих значень.

6. Вперше запропоновано спосіб проєктування рекурсивних фільтрів на ПЛІС, який відрізняється тим, що завдяки застосуванню методу відображення просторового ГСПД, використання схем без блоків множення, а також пошуку коефіцієнтів фільтра методом модельованого відпалювання та застосування мови VHDL, забезпечується одержання фільтрів з обсягом обладнання на 12–50% меншим та тактовою частотою в 0,5–5,5 разів більшою в порівнянні з аналогічними фільтрами.

7. Результати роботи впроваджені у двох науково-дослідних роботах: 1) НДР №2863-п «Створення засобів проектування та розробка на їх основі високопродуктивних процесорів систем технічного зору», номер державної реєстрації 0115U002326; 2) НДР ФІОТ 3ОТ/2017 «Методи і засоби

відображення потокових алгоритмів у конфігуровні комп’ютери», номер державної реєстрації 0119U102212.

8. Результати досліджень можливо використовувати при проєктуванні високопродуктивних апаратних засобів ЦОС, зокрема, апаратних прискорювачів обчислень, у тому числі для штучного інтелекту, у нових САПР, а також у навчальному процесі.

9. Подальші наукові дослідження слід спрямувати на вдосконалення алгоритму генетичного програмування, адаптації методу до реалізації мов високого рівня, таких як TensorFlow.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем / Вальковский В. А., Котов В. Е., Миклошко Й. и др. М.: Наука, 1982. 340 с.
2. Вайдьнатхан П. П. Цифровые фильтры, блоки фильтров и полифазные цепи с многочастотной дискретизацией. Методический обзор. *ТИИЭР*. 1990. Т. 78, № 3. С. 77–120.
3. Каневский Ю. С., Сергиенко А. М. Формализованное проектирование систолических структур и их процессорных элементов. *Автоматика и вычислительная техника*. 1990. № 3. С. 72–78.
4. Карп Р. М., Миллер Р. Е. Параллельные схемы программ / перев. Кагр R. M., Miller R. E., Parallel Program Schemata. *Journal of Computer and System Sciences*. 1969. No 3. P. 147–195 / *Кибернетический сборник*. Вып. 13. / ред. Лупанов О. Б. С. 5–30.
5. Котов В. Е. Введение в теорию схем программ. Новосибирск: Наука, 1978. 258 с.
6. Кофман И. Г. ЭВМ и теория расписаний. М.: Мир, 1979. 460 с.
7. Криницкий Н. А. Алгоритм. *Энциклопедия кибернетики*. Киев: УСЭ, 1974. С. 94–95.
8. Ли Е. А., Мессершмитт Д. Г. Вычисления с синхронными потоками данных. *Труды ТИИЭР*. 1987. Т. 75, № 9. С. 107–119. / перев. E. A. Lee and D. G. Messerschmitt. Synchronous Data Flow. *IEEE Proceedings.* Sept. 1987.
9. Мингазин А. Т. Синтез цифровых фильтров для высокоскоростных систем на кристалле. *Цифровая Обработка Сигналов.* 2004. № 2. С. 14–23.
10. Норенков И. П. Основы автоматизированного проектирования: Учеб. для вузов. 4-е изд. М.: Изд-во МГТУ им. Н. Э. Баумана, 2009. 432 с.

11. Оппенгейм А. В., Шафер Р. В. Цифровая обработка сигналов. М.: Связь, 1979. 416 с.
12. Сергиенко А. Б. Цифровая обработка сигналов : второе изд. СПб: Питер, 2006. 751 с.
13. Сергиенко А. М. Методика проектирования цифровых фильтров с помощью VHDL. *Моделювання та інформаційні технології*: зб. наук. праць. Ін-т проблем моделювання в енергетиці ім. Г. Е. Пухова, НАНУ, 2002. Вип. 12. С. 99–107.
14. Сергиенко А. М. VHDL для проектирования вычислительных устройств. Киев: Диасофт, 2004. 205 с.
15. Сергиенко А. М. 5,5 десятилетий цифровой обработки сигналов. *Argc&Argv*. 2006. № 1. С. 19–25.
16. Сергиенко А. М., Симоненко, В. П. Отображение периодических алгоритмов в программируемые логические интегральные схемы. *Электронное моделирование*. 2007. Т. 29, № 2. С. 49–61.
17. Сергиенко А. М., Симоненко В. П. Алгоритмические модели обработки потоков данных. *Электронное моделирование*. 2008. Т. 30, № 6. С. 4960.
18. Сергиенко А. М. Пространственный граф синхронных потоков данных. *Вісник НТУУ «КПІ»*: *Інформатика, управління та обчислювальна техніка*: зб. наук. праць. 2010. №. 51. С. 40–46.
19. Сергиенко А. М., Лесик Т. М. Перестраиваемые цифровые фильтры на ПЛИС. *Электронное моделирование*. 2010. Т. 32, № 6. С. 47–56.
20. Сергиенко А. М., Сергиенко А. А. Моделирование волновых процессов с помощью волновых фильтров. *Моделювання-2018* : тези міжн. наук. конф. Київ, 2018, С. 224–227.
21. Сергієнко А. А., Клятченко Я. М. Процесор швидкого перетворення Фур'є за простою основою. *Сучасні методи, інформаційне та програмне забезпечення систем управління організаційно-*

технологічними комплексами : тези всеукраїнської науково-практичної інтернет-конф. Луцьк, 2015. С. 21–23.

22. Сергієнко А. А., Сергієнко А. М. Набір модулів для швидкого перетворення Фур'є. *Infocom Advanced Solutions 2015* : тези міжнар. наук.-практ. конф. Київ, 2015. С. 52–53.
23. Сергієнко А. А. Реалізація конвеєрних процесорів швидкого перетворення Фур'є у ПЛІС. *Прикладна математика та комп'ютинг — 2016* : тези наукової конференції. Київ, 2016. С. 128–131.
24. Сергієнко А. А., Сергієнко А. М. Бібліотека модулів для швидкого перетворення Фур'є. *Інформатика та обчислювальна техніка – ІОТ-2016* : тези наук. конф. студентів, магістрантів та аспірантів. Київ, 2016. С. 114–117.
25. Сергієнко А. М. Досконалий кістяк графа алгоритму. *Вісник НТУУ "КПІ": Інформатика і обчислювальна техніка*: зб. наук. праць. 2007. №46. С. 62–67.
26. Сергієнко А. М., Лепеха В. Л., Лесик Т. М. Спецпроцесори для двовимірного дискретного косинусного перетворення. *Вісник НТУУ «КПІ», Сер.: Інформатика і обчислювальна техніка*: зб. наук. праць. Т. 47. 2007. С. 230–233.
27. Сергієнко А. М., Мєлковська В. М., Стіренко С. Г. Спосіб демодуляції сигналів з багаточастотною модуляцією. *Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка*: збірник наукових праць. Т. 48. 2008. С. 82–84.
28. Сергієнко А. М., Сімоненко В. П. Складання розкладу для графів синхронних потоків даних. *Системні дослідження та інформаційні технології*. 2016. №1. С. 51–62.
29. Сергієнко А. М., Романкевич В. О., Сергієнко А. А. Генетичне програмування опису конвеєра даних мовою VHDL. *Прикладна*

математика та комп'ютинг : тези десятої наук. конф. магістрантів та аспірантів ПМК'2018 (Київ, 21–23 бер. 2018). Київ, 2018. С. 153–157.

30. Сергієнко А. М., Хусейн К. С., Сергієнко А. А. Фільтри зі скінченною характеристикою з мінімізованими апаратними витратами. *Безпека, Відмовостійкість, Інтелект* : тези наук. конф. Київ, НТУУ «КПІ», 2018. С. 99–103.
31. Сергієнко А. М., Романкевич В. О., Сергієнко А. А. Генетичне програмування спеціалізованих конвеєрних пристроїв. *Електронне моделювання.* 2020. Т. 42, №2. С. 25–40.
32. Цифровая обработка сигналов: справочник / Гольденберг Л. М. и др. М.: Радио и связь, 1985. 312 с.
33. 2-D Discrete Cosine Transform (DCT). V2.0. *Xilinx Inc Product Specification*. 2002. March 14. 11 p. URL: http://www.xilinx.com.
34. Accellera. SystemVerilog 3.0. Accellera's Extensions to Verilog. URL: http://www.accellera.org.
35. Affenzeller M., Winkler S. Genetic Algorithms and Genetic Programming. Modern Concepts and Practical Applications. Chapman & Hall, CRC: 2009. 358 p.
36. Ahn C. W., Ramakrishna R. S. Augmented compact genetic algorithm. *Proc. 5th Int. Conf. on Parallel Processing and Applied Mathematics* (PPAM 2003) / *Lecture Notes in Computer Science* (LNCS). Berlin, Springer, 2004. V. 3019. P. 560–565.
37. AI and Compute. May 16, 2018. URL: https://openai.com/blog/ai-and-compute/
38. Andraka R. A survey of Cordic algorithms for FPGA based computers. *ACM/SIGDA 6-th International Symposium on FPGAs*: 1981–2000, 1998.

39. Aráujo S. G., Mesquita A. C., Pedroza A. Optimized Datapath Design by Evolutionary Computation. *IWSOC: International Workshop on System-on-Chip for Real-Time Applications*. 2003. P. 6–9.
40. Artemio C. Coello C. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*. 1999. V. 1, No 3. P. 269–308.
41. Ashenden P. J., Lewis J. The Designer's Guide to VHDL. Morgan Kaufmann, 2008. 936 p.
42. Baker J. E. Adaptive Selection Methods for Genetic Algorithms. *Proc. 1st International Conference on Genetic Algorithms*. 1985. P. 101–111.
43. Balarin F., Kondratyev A., Watanabe Y. High Level Synthesis. *Electronic Design Automation for IC System Design, Verification, and Testing* / Ed-s: Lavagno L. et al. CRC Press, 2016. P. 229–274.
44. Barrera B., Lee E. A. Multirate signal processing in Comdisco's SPW. *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing* (ICASSP 91), Toronto, Ontario, Canada. 1991. V. 2. P. 1113–1116.
45. Barthelemy J. F. M., Haftka R. T. Approximation concepts for optimum structural design – a review. *Structural Optimization.* 1993. No 5. P. 129–144.
46. Prugel-Bennett A. and Rogers A. Modelling GA Dynamics / *Theoretical Aspects of Evolutionary Computing*. Springer, 2001. P. 59–86. URL: http://eprints.ecs.soton.ac.uk/13205/
47. Berman V. Standards: The P1685 IP-XACT IP Metadata Standard. *IEEE Design & Test of Computers.* 2006. V. 23, No 4. P. 316–317.
48. Berry G. Real Time programming: Special purpose or general purpose languages. *Information Processing* / G. Ritter, Ed. Elsevier Science Publishers B. V. North Holland. 1989. V. 89. P. 11–17.

49. Berry G., Gontier G. The Esterel synchronous programming language: Implementation. *Science of Computer Programming*. 1992. V. 19, No. 2. P. 87–152.
50. Berry, G. The effectiveness of synchronous languages for the development of safety-critical systems. *White paper*. Esterel Technologies, 2003.
51. Bhattacharya B. and Bhattacharyya S. S. Parameterized Dataflow Modeling for DSP Systems. *IEEE Trans. on Signal Processing*. 2001. V. 49, No. 10. P. 2408–2421.
52. Bhattacharyya, S. S., Brebner G., Janneck J. W. et al. OpenDF — A dataflow toolset for reconĩgurable hardware and multicore systems. *Proc. of the Swedish Workshop on Multi-Core Computing.* Ronneby, Sweden, 2008.
53. Bhattacharyya S., Wolf M. Tools and Methodologies for System-Level Design. *Electronic Design Automation for IC System Design, Verification, and Testing*. CRC Press, 2016. P. 39–57.
54. Bi G., Jones E. A pipelined FFT processor for word-sequential data. *IEEE Transactions on Acoustics, Speech and Signal Processing*. 1989. V. 37, No. 12. P. 1982–1985.
55. Bienik J., Uhrina M., Kuba M., Vaculik M. Performance of H.264, H.265, VP8 and VP9 Compression Standards for High Resolutions. *19-th Int. Conf. on Network-Based Information Systems.* 2016. P. 246–252.
56. Black, D. C., Donovan J. SystemC: From the Ground Up. 2nd edn. Springer, 2010. 244 p.
57. Bonsma, E., Gerez, S. A genetic approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays. *ProRISC Workshop on Circuits, Systems and Signal Processing.* 1997. P. 67–75.

58. Börger E. Abstract State Machines: A Method for High-Level System Design and Analysis. *Formal Methods: State of the Art and New Directions* / P. Boca, J. P. Bowen, J. Siddiqi, Ed-s. Springer, 2009. P. 79–116.
59. Boutellier, J., Sadhanala V., Lucarz C., Mattavelli M., Brisk P. Scheduling of dataflow models within the reconfigurable video coding framework. *Proc. of the IEEE Workshop on Signal Processing Systems*. 2008. P. 182–187.
60. Brandolese C, Fornaciari W, Salice F. An area estimation methodology for FPGA based designs at SystemC-level. *Design Automation Conference,* (DAC). ACM, New York, USA, 2004. P. 129–132.
61. Buck J. T., Ha S., Lee E. A., Messerschmitt D. G. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. J. Computer Simul.* 1994. No. 4. P. 155–182.
62. Buck J., Vaidyanathan R. Heterogeneous modeling and simulation of embedded systems in El Greco. *Proc. Int. Workshop on Hardware/Software Co-Design*. 2000. P. 142–146.
63. Burks, A. R., Punch, W. F. An efficient structural diversity technique for genetic programming. *Proc. 2015 Ann. Conf. on Genetic and Evolutionary Computation,* (GECCO'15). ACM, Madrid, 2015. P. 991–998.
64. Camposano R. Path-based scheduling for synthesis. *IEEE Trans. on Computer-Aided Design*. 1991. № 10. P. 85–93.
65. Camposano R., Saunders L. F., Tabet R. M. VHDL as Input for High-Level Synthesis. *IEEE Design & Test of Computers*. 1991. March. P. 43–49.
66. Cardoso J. M. P., Diniz P. C. Compilation Techniques for Reconfigurable Architectures. Springer, 2009. 223 p.
67. Catthoor F., De Man H. Application-specific architectural methodologies for high-throughput digital signal and image processing. *IEEE Trans. on Acoustics, Speech and Signal Processing*. 1990. V. 38, No. 2. P. 339–349.

68. Chafekar D., Xuan J., Rasheed K. Constrained Multi-objective Optimization Using Steady State Genetic Algorithms. *Proc. of Genetic and Evolutionary Computation* (GECCO), 2003. P. 813–824.
69. Chao L., La Paugh A., Sha E. Rotation scheduling: A loop pipelining algorithm. *Proc 30-th Design Automation Conf.* (DAC'93). 1993. P. 566–572.
70. Chen D., Cong J. Register binding and port assignment for multiplexer optimization. *Asia South Pacific Design Automation Conference* (ASP-DAC) : proc. int. conf. 2004. P. 68–73.
71. Chen J. H., Goldberg D. E., Ho S. Y., Sastry K. Fitness inheritance in multi-objective optimization. *Genetic and Evolutionary Computation Conference*, (GECCO) 2002. 2002. P. 319–326.
72. Coelho D. F. G. DCT_1D_VHDL. GitHub repository. Last version: 4 Feb 2018. URL: https://github.com/diegofgcoelho/dct_1d_vhdl (last accessed: 20.10.2022).
73. Cossement, N., Lauwereins R., Catthoor F. DF*: An extension of synchronous dataflow with data dependency and non-determinism. *Proc. of the Forum on Specification and Design Languages,* (FDL00/SSDL Workshop), Tübingen, Germany. 2000.
74. Culbertson B., Amerson R., Carter R. J., Kuekes P., Snider G. Defect Tolerance on the Teramac Custom Computer. *Proc. 5th IEEE Symp. on Field-Programmable Custom Computing Machines*. 1997. P. 140–147.
75. Daitx F. F, Rosa V. S., Costa E., Flores P., Bampi S. VHDL Generation of Optimized FIR Filters. *2-nd Int. Conf. on Signals, Circuits and Systems*, 7-9 Nov. 2008. P. 1–5.
76. Dawson S., Sezer P. S. SVG for Remote EDA Schematic Representation. *Proc. 3rd Conf. on Scalable Vector Graphics* (SVG Open 2004). 2004. URL: http://www.svgopen.org/ 2004/papers/SVGforEDASchematics/

77. Designing with Xilinx FPGAs: Using Vivado. / S., Churiwala, Ed. Switzerland : Springer, 2017. 270 p. DOI: https://doi.org/10.1007/978-3-319-42438-5.
78. Dorigo M., Stuetzle T. Ant Colony Optimization. Bradford/MIT Press, Cambridge, MA, USA, 2004. 319 p.
79. Dossis M. High-Level Synthesis for Embedded Systems. *Embedded Systems – Theory and Design Methodology.* / K. Tanaka, Ed. Rijeka, Croatia, Intech WEB. ORG. 2012. P. 341–366.
80. Dueck, G. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *J. Comput. Phys.* Elsevier, Amsterdam, Netherlands, 1993. V. 104, No. 1. P. 86–92.
81. Duhamel P., Hollmann H. Split radix FFT algorithm. *Electron. Lett*. 1984. V. 20, No. 1. P. 14–16.
82. Dutt N., Ramachandran C. Benchmarks for the 1992 High Level Synthesis Workshop. *UCI Technical Report*. October, 1992. No. 92−108.
83. Edwards S., Lavagno L., Lee E. A., and Sangiovanni-Vincentelli A. Design of embedded systems: Formal models, validation and synthesis, *Proc. of the IEEE*. 1997. V. 85, No. 3. P. 366–390.
84. Eker J., Janneck J., Lee E. A., Liu J., Liu X., Ludvig J., Sachs S., Xiong Y. Taming heterogeneity − the Ptolemy approach. *Proc. of the IEEE*. 2003. V. 91, No. 1. P. 127−144.
85. Eker J., Janneck J. W. Dataflow programming in CAL — Balancing expressiveness, analyzability, and implementability. *Proc. of the IEEE Asilomar Conference on Signals, Systems, and Computers*. 2012. P. 1120–1124.
86. Farmer J. D., Packard N., Perelson A. The Immune System, Adaptation and Machine Learning. *Physica D: Nonlinear Phenomena 2*. Elsevier, Amsterdam, Netherlands, 1986. P. 187–204.

87. Farouki R. T. Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable. *Series: Geometry and Computing*. Springer, 2008.
88. Ferrandi F., Lanzi P. L., Palermo G., Pilato C., Sciuto D., Tumeo A. An evolutionary approach to area-time optimization of FPGA designs. *Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation*, (ICSAMOS). 2007. P. 145–152.
89. Fettweis A. Wave digital filters: Theory and practice. *Proc. IEEE*. 1986. V. 74, № 2. P. 270–327.
90. Fonseca C. M., Fleming P. J. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. *Proc. 5-th Int. Conf. on Genetic Algorithms.* 1993. P. 416–423.
91. Franchetti F., Voronenko Y., Milder P., Chellappa S., Telgarsky M., Shen H., D'Alberto P., de Mesmay F., Hoe J., Moura J., Puschel M. Domain-specific library generation for parallel software and hardware platforms. *IEEE Int. Symp. on Parallel and Distributed Processing* (IPDPS). April 2008. P. 1–5.
92. Gajski D. D., Zhu J., Dömer R., Zhao S. SPECC: Specification Language and Methodology. US: Springer, 2000. 313 p.
93. Garey M. J., Johnson D. S. Complexity results for Multiprocessor Scheduling under Resource Constraints. *SIAM J. Computing*. 1975. V. 4. P. 397–411.
94. Gerlach J., Rosenstiel W. System Level Design Using the SystemC Modeling Platform. URL: http://www.systemc.org.
95. Girault A., Lee B., Lee E. A. Hierarchical finite state machines with multiple concurrency models. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*. 1999. V. 18, No. 6. P. 742–760.
96. Goldberg B.-G. Digital Frequency Synthesis Demystified. USA: LLH Technology Publ., 1999. 352 p.

97. Goldberg D. E., Richardson J. Genetic algorithms with sharing for multimodal function optimization. Genetic Algorithms and Their Applications. *Proc. of 2-nd Int. Conf. on Genetic Algorithms*, Lawrence Erlbaum, Hillsdale, 1987. P. 41–49.
98. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Boston, MA, USA: Addison-Wesley Longman Publ., January, 1989. 412 p.
99. Grajal J., Sanchez M. A., Garrido M., Gustafsson O. Pipelined Radix-2^k Feedforward FFT Architectures. *IEEE Trans. on VLSI Systems*. 2013. V. 21, No. 1. P. 23–32.
100. Grajcar M. Conditional Scheduling for Embedded Systems using Genetic List Scheduling. *Proc. 13-th Int. Symp. on System Synthesis* (ISSS), Madrid, Spain, 2000. P. 123–128.
101. Grefenstette J. J., Fitzpatrick J. M. Genetic search with approximate function evaluation. *Int. Conf. on Genetic Algorithms*. Mahwah, NJ, USA: Lawrence Erlbaum Assoc. Inc., 1985. P. 112–120.
102. Grewal G., Cleirigh M. O., Wineberg M. An evolutionary approach to behavioural-level synthesis. *The 2003 Congress on Evolutionary Computation,* (CEC'03), 2003. V. 1, Dec., 2003. P. 264–272.
103. Gulizia S., Do K. Synopsys System Studio Speeds DSP Algorithm Development With New Matrix Data-Type Support. 3 March 2020. URL: https://news.synopsys.com/index.php?s=20295&item=123136.
104. Gutberlet P., Rosenstiel W. Scheduling Between Basic Blocks in the CADDY Synthesis System. *EDAC 92*, Brüssel, Belgien, 16–19 März 92, 1992. P. 496–500.
105. Halbwachs N., Caspi P., Raymond P., Pilaud D. The synchronous data flow programming language LUSTRE. *Proc. IEEE*. 1991. V. 79, No. 9. P. 1305–1320.

106. Hannig F. Scheduling Techniques for High-Throughput Loop Accelerators. Erlangen, 2009. 295 p.

107. Hannig F. A Quick Tour of High-Level Synthesis Solutions for FPGAs. *FPGAs for Software Programmers.* / D. Koch, F. Hannig, D. Ziener, Ed-s. Springer, 2016. P. 49–59.

108. Harel D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*. 1987. V. 8, No. 3. P. 231–274.

109. Hariharan K., Hubert E. B., Divyalakshmi K. V. O., Shamalla K., Kumar A. V. Coherent Sinusoid Generation using Novel DDFS Architecture. *Int. J. of Smart Home*. 2012. V. 6, No. 1. P. 17–28.

110. Harish Ram D. S., Bhuvaneswari M. C., Prabhu1 S. S. A Novel Framework for Applying Multiobjective GA and PSO Based Approaches for Simultaneous Area, Delay, and Power Optimization in High Level Synthesis of Datapaths. *VLSI Design*. V. 2012, Hindawi Publ. Article ID 273276, 12 p.

111. Haubelt C. Falk J., Keinert J., Schlichter T., et al. A SystemC-based design methodology for digital signal processing systems. *EURASIP Journal on Embedded Systems*. Hindawi Pub., 2007. Article ID 47580, 22 p.

112. He S., Torkelson M. A new approach to pipeline FFT processor. *Proc. 10-th Intern. Parallel Processing Symposium* (IPPS ’96), Honolulu, Hawaii, USA, April, 1996. P. 766–770.

113. Hendren L. J., Gao G. R., Altman E. R., Mukerji C. A register allocation framework based on hierarchical cyclic interval graphs. *The Journal of Programming Languages*. 1993. V. 1, No. 3. P. 155–185.

114. Hennesy J., Patterson D. A New Golden Age of Computer Architecture. *Communications of the ACM*. 2019. V. 62, № 2. P. 48–60.

115. High perfomance 64-point Complex FFT/IFFT, Xilinx Product Specification. V.7.0, June, 2009. URL: http://www.xilinx.com/ipcenter.

116. High-Level Synthesis. From Algorithm to Digital Circuit. / Coussy P., Morawiec A., Ed-s. Springer, 2008. 297 p.

117. Holland H. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, 1975.

118. Horn J., Nafpliotis N., Goldberg D. E. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. *Proc. 1-st IEEE Conf. on Evolutionary Computation*. 1994. V. 1. P. 82–87.

119. Hsiao S.-F., Shiue W.-R., Tseng J.-M. A cost efficient fully-pipelinable architecture for DCT/IDCT. *IEEE Trans. On Communications*. 1991. V. 39, No. 5. P. 640–643.

120. Husa J., Kalkreuth R. A Comparative Study on Crossover in Cartesian Genetic Programming. *Proc. 21-st European Conf. Genetic Programming*, (EuroGP), Parma, Italy, April 4–6, 2018. Springer: LNCS, 2018. V. 10781. P. 203–219.

121. Hwang K. S., Casavant A. E., Chang C. T., d'Abreu M. A. Scheduling and hardware sharing in pipelined data paths. *Proc. Int'l Conf. on Computer Aided Design*. 1989. P. 24–27.

122. IEEE Standard For Verilog Register Transfer Level Synthesis (IEEE Std 1364.1 –2002). IEEE, NY, 2002.

123. Jin Y. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput*. 2005. V. 9, No. 1. P. 3–12.

124. Kahn G. The semantics of a simple language for parallel programming. *Proc. IFIP Congress'74*. North-Holland Pub. Comp., 1974. P. 471–475.

125. Karjalainen M., Erkut C. Digital Waveguides versus Finite Difference Structures: Equivalence and Mixed Modeling. *EURASIP Journal on Applied Signal Processing.* 2004. №7. P. 978–989.

126. Kennedy J., Eberhart R. Particle swarm optimization. *Proc. IEEE Int. Conf. Neural Netw*. IEEE Press, Piscataway, NJ, USA, 1995. V. 4. P. 1942–1948.

127. Khan S. A. Digital Design of Signal Processing Systems. A Practical Approach. UK: Wiley, 2011. 506 p.

128. Kirkpatrick S., Gelatt C. D., Vercchi M. P. Optimization by simulated annealing. *Science.* High Wire Press, Stanford, CA, USA, 1983. V. 220. P. 671–680.

129. Kitsos P., Voros N. S., Dagiuklas T., Skodras A. N. A High Speed FPGA Implementation of the 2D DCT for Ultra High Definition Video Coding. *DSP 2013* : proceedings of the 18th International Conference on Digital Signal Processing, Fira, Greece, 01–03 July 2013. IEEE, 2013. DOI: http://dx.doi.org/10.1109/ICDSP.2013.6622742.

130. Klymenko I., Tkachenko V., Serhienko A., Kulakov Y. Formalization of the concept of adaptive tasks mapping in the reconfigurable computers on FPGA. *Eastern European Journal of Enterprise Technologies*. 2018. V. 2, No. 9‒92. P. 20–28.

131. Koncal O., Sekanina L. Cartesian Genetic Programming as an Optimizer of Programs Evolved with Geometric Semantic Genetic Programming. *Proc. 22-nd European Conf. Genetic Programming*, (EuroGP), Leipzig, Germany, April 24–26, 2019. P. 98–113.

132. Krishnan V, Katkoori S. A genetic algorithm for the design space exploration of datapaths during high-level synthesis. *IEEE Trans. on Evolutionary Computation*. 2006. V. 10, No. 3. P. 213–229.

133. Kruse R., Borgelt C., Braune C., Mostaghim S., Steinbrecher M. Computational Intelligence. A Methodological Introduction : 2-nd Ed. Springer, 2016. 564 p.

134. Kumm M. Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays. Springer, 2016. 206 p.

135. Kurdahi F., Parker A. REAL: A program for register allocation. *Proc. of the Design Autom. Conf.* (DAC). Miami Beach, Florida, USA, June, 1987. P. 210–215.

136. Kusuma E. D., Widodo T. S. FPGA implementation of pipelined 2D-DCT and quantization architecture for JPEG image compression. *2010 International Symposium on Information Technology*, Kuala Lumpur, 2010. P. 1–6.

137. Laessig J., Hoffmann K. H., and Enachescu M. Threshold Selecting: Best Possible Probability Distribution for Crossover Selection in Genetic Algorithms. *Genetic and Evolutionary Computation Conf.* 2008. P. 2181–2185.

138. Lakshminarayana G., Khouri G., Jha N. K. Wavesched: A novel Scheduling Technique for Control-Flow Intensive Behavioral Descriptions. *Proc. ICCAD*, Nov. 1997. P. 245–251.

139. Laland K. N., Uller T., Feldman M. W., Sterelny K., Muller G. B., Moczek A., Jablonka E. Odling-Smee J. The extended evolutionary synthesis: its structure, assumptions and predictions. *Proc. R. Soc. B,* 282: 2015. 1019. URL: http://dx.doi.org/10.1098/ rspb.2015.1019

140. Larsson. M. J. P. A Transformational Approach to Formal Digital System Design. *Licentiate Thesis 378*, Dept. Computer and Information Sci., Linkoping Univ., May 1993. 133p.

141. Le Guernic P., Gautier T., Le Borgne M., Le Maire C. Programming real-time applications with SIGNAL. *Proc. IEEE*. 1991. V. 79, No. 9. P. 1321–1336.

142. Lee E. A., Messerschmitt D. G. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Trans. on Computers*. 1987. V. C–36, № 1. P. 24–35.

143. Lee E. A. Recurrences, Iteration, and Conditionals in Statically Scheduled Block Diagram Languages. *VLSI Signal Processing III* / R.W. Brodersen, H. S. Moscovitz, Ed-s. IEEE Press, New York, 1988. P. 330–340.

144. Lee E. A., Ha S. Scheduling Strategies for Multiprocessor Real-Time DSP. *Proc. of IEEE GLOBECOM.* Nov. 1989. P. 1279–1284.

145. Lee E. A. Embedded Software. *Advances in Computers*, / M. Zelkowitz. Ed. Academic Press, London, 2002. V. 56.

146. Lee, E. A. and Neuendorffer S. Concurrent models of computation for embedded software. *IEE Proc.-Comput. Digit. Tech*. 2005. V. 152, № 2. P. 239–250.

147. Lee, E. A., Varaiya, P. Structure and Interpretation of Signals and Systems, 2nd edn., LeeVaraiya org., 2011.

148. Lee J. H., Hsy Y. C., Lin Y. L. A New Integer Linear Programming Formulation for the Sheduling Problem in Data Path Synthesis. *Proc. Int. Conf. Computer Aided Design* (ICCAD). 1989. P. 20–23.

149. Lee S., Soak S., Kim K., Park H., Jeon M. Statistical properties analysis of real world tournament selection in genetic algorithms. *Applied Intelligence*. 2008. V. 28, No. 2. P. 195–205.

150. Leiserson C. E., Saxe J. B. Retiming Synchronous Circuitry. *Algorithmica*. 1991. No. 6. P. 5–35.

151. Lis J. S., Gajski, D. VHDL Synthesis using Structured Modeling, *Proc. IEEE/ACM Design Automation Conference*. 1989. P. 606–609.

152. LogiCORE IP CORDIC v5.0 DS858. Product Specification. Xilinx Inc. October 19, 2011. URL: http://www.xilinx.com/

153. Mandal C., Chakrabarti P., Ghose S. Design space exploration for data path synthesis. *Proc. 10-th Int. Conf. on VLSI Design*. 1996. P. 166–170.

154. Mandal C., Chakrabarti P. P., Ghose S. GABIND: a GA approach to allocation and binding for the high-level synthesis of data paths. *IEEE Trans. on VLSI Systems*. 2000. V. 8, No. 6. P. 747–750.

155. Maslennikow O., Maslennikowa N., Ratushnyak P., Sergiyenko A., Wozniak M. Applicaton specific processors for the autorgressive signal analysis. *8-th Int. Conf. Parallel Processing and Applied Mathematic*, (PPAM'2009), Part I, LNCS, Springer, 2010. V. 6068. P. 80–86.

156. Maxfield C. The Design Warrior's Guide to FPGAs. Newnes, Elsevier, 2004. 542 p.

157. Meher P. K., Patra J. C., Vinod A. P. Efficient systolic designs for 1-and 2-D DFT of general transform-lengths for high-speed wireless communication applications. *J. Sig. Process. Sys.* 2010. V. 60. P. 1–14.

158. Meredith M. High-Level SystemC Synthesis with Forte's Cynthesizer. *High-Level Synthesis* / Coussy P., Morawiec A. Ed-s. Springer, Dordrecht, 2008. P. 75–97.

159. Meribout M., Motomura M. Efficient metrics and high-level synthesis for dynamically reconfigurable logic. *IEEE Trans. Very Large Scale Integr. Syst.* 2004. V. 12, No. 6. P. 603–621.

160. Meyer-Baese. U. Digital Signal Processing with Field Programmable Gate Arrays. Springer, 4-th Ed. 2014. 930 p.

161. Micheli G. D. Synthesis and Optimization of Digital Circuits, McGraw-Hill, Inc. 1994.

162. Milic D., Lutovac M. D. Design of multiplierless elliptic IIR filters with a small quantization error. *IEEE Trans. on signal processing*. 1999. V. 47, No. 2. P. 469–479.

163. Miller J. F. Cartesian Genetic Programming. Berlin: Springer. 2011.

164. Mohanty S. P., Ranganathan N., Kougianos E. Patra P. Low-Power High-Level Synthesis for Nanoscale CMOS Circuits. Springer. 2008. 302 p.

165. Motomura M., Hariyama M., Watanabe M. Advanced Devices and Architectures. *Principles and Structures of FPGAs*. / H. Amano Ed. Singapore: Springer, 2018. P. 179–206.

166. Nakrani S., Tovey S. On honey bees and dynamic server allocation in internet hosting centers. Adaptive Behavior. *SAGE Publications*, New York, NY, USA, 2004. V. 12. P. 223–240.

167. Natale M. D. Models and Tools for Complex Embedded Hardware and Systems. *Electronic Design Automation for System Design, Verification and Testing*. / L. Lavagno et al. Ed-s. CRC Press, 2016. P. 141–197.

168. Nestor J., Krishnamoorthy G. SALSA: A New Approach to Scheduling with Timing Constraints. *Proc. ICCAD-90*. Nov. 1990. P. 262–265.

169. Nikara J., Takala J., Akopian D., Saarinen J. Pipeline Architecture for DCT/IDCT. *IEEE Int. Symp. on Circuits and Systems*, (ISCAS 2001), May 6-9, Sydney, Australia, 2001. P. 902–905.

170. Nimwegen E., Crutchfield J. P. Optimizing Epochal Evolutionary Search: Population-Size Dependent Theory. *Machine Learning*. 2001. V. 45, No. 1. P. 77–114.

171. Nussbaumer H. J. Fast Fourier Transform and Convolution Algorithms, Springer, 1982.

172. Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul, M., Vemuri, R. An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. *Proc. of the Reconfigurable Architectures Workshop* (RAW'98), Berlin/Heidelberg: Springer, 1998, V. 1388. P. 31–36.

173. Ouerhani Y., Jridi M. Implementation Techniques of High-Order FFT into Low-Cost FPGA. *Int. IEEE Midwest Symp. on Circuits and Systems* (MWSCAS), 2011. P. 1–4.

174. Palesi M, Givargis T. Multi-objective design space exploration using genetic algorithms. *Int. Symposium on Hardware/software Codesign,* (CODES), New York, NY, USA: ACM, 2002. P. 67–72.

175. Pan S.-T. CSD-Coded Genetic Algorithm on Robustly Stable Multiplierless IIR Filter Design. *Mathematical Problems in Engineering*, Hindawi Publ. 2012. V. 2012, Article ID560650, 15 p.

176. Pangrle B. M., Gajski D. D. Design Tools for Intellingent Silicon Compilation. *IEEE Trans. CAD*. 1987. V. 6, No. 6. P. 1098–1112.

177. Parate P. P., Mohota N. A. FPGA Implementation of 2D-DCT for Image Compression. *International Journal of Science and Research*. 2013. Vol. 4, No 7. P. 142–145.

178. Parhi K. K., Wang C. Y., Brown A. P. Synthesis of control circuits in folded pipelined DSP architectures. *IEEE J. Solid-St. Circ.* 1992. V. 27. P. 29–43.

179. Park N., Kurdahi F. J. Module Assignment and Interconnect Sharing in Register-Transfer Synthesis of Pipelined Data Paths. *Proc. IEEE Int. Conf. on Computer Aided Design*. Santa Clara, Calif., Nov, 1989. P. 16–19.

180. Parsa S., Lotfi S. A New Genetic Algorithm for Loop Tiling. *The Journal of Supercomputing*, 2006. V. 37. No 3. P. 249–269.

181. Paulin P. G., Knight J. P. Force – Directed Sheduling for the Behavioral Synthesis of ASICs. *IEEE Trans. CAD*. 1988. V. 7, No. 3. P. 356–370.

182. Paulin P. G., Knight, J. P. Girczyc E. F. HAL: A multi-paradigm approach to automatic data path synthesis. *Proc. 23-rd IEEE Design Automation Conf.*, Las Vegas, NV, July 1986. 1986. P. 263–270.

183. Pilato C., Tumeo A., Palermo G., Ferrandi F., Lanzi P.L., Sciuto D. Improving evolutionary exploration to area-time optimization of FPGA designs. *J. of Systems Architecture*, 2008. V. 54. P. 1046–1057.

184. Plishker W., Sane N., Kiemb M., Anand K., Bhattacharyya S. S. Functional DIF for rapid prototyping. *Proc. Int. Symp. on Rapid System Prototyping*, Monterey, CA, 2008. P. 17–23.

185. Poli R. Evolution of Graph-like Programs with Parallel Distributed Genetic Programming. *Genetic Algorithms* : Proc. 7-th Int. Conf. 1997. P. 346–353.

186. Potkonjak M., Rabaey J. Optimizing Resource Utilization using Transformations. *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*. 1991. P. 88–91.

187. Pradeepthi T., Ramesh A. P. Pipelined Architecture of 2D-DCT, Quantization and Zigzag Process for JPEG Image Compression using VHDL. *International Journal of VLSI design & Communication Systems*. 2011. Vol. 2, No 3. P. 99–110.

188. Rabaey J. System-on-Chip-Challenges in the Deep-Sub-Micron Era. A case for the network-on-a-Chip. *Interconnect-Centric Design for Advanced SoC and NoC*. Springer. 2004. P. 3–24.

189. Rabiner L. R., Gold B. Theory and Application of Digital Signal Processing. Upper Saddle River, NJ, USA: Prentice-Hall, 1975.

190. Raghunathan V., Srivastava M. B., Gupta R. K. A Survey of Techniques for Energy Efficient On-Chip Communication. *Design Automation Conference,* (DAC-2003). ACM, 2003. P. 900–905.

191. Regalia P. A., Mitra S. K., Vaidyanathan P. P. The Digital All-Pass Filter: A Versatile Signal Processing Building Block. *Proc. IEEE*. 1988. V. 76, № 1. P. 19–37.

192. Roje M. Insights from the Next FPGA Platform Event. Acronix, 04.02.2020. URL: https://www.achronix.com/blog/insights-from-the-next-fpga-platform-event/

193. Rumbaugh J., Jacobson I., and Booch G. The Unified Modeling Language Reference Manual, 2nd edn. Addison-Wesley, Boston, MA, 2004.

194. Sastry K., Goldberg D. E. Genetic Algorithms Laboratory (IlliGAL), Modeling Tournament Selection With Replacement Using Apparent Added Noise. *Proc. of the Genetic and Evolutionary Computation Conf.* (GECCO-2001). 2001. P. 781.

195. Sastry K., Lima C. F., Goldberg D. E. Evaluation relaxation using substructural information and linear estimation. *Proc. of the Genetic and Evolutionary Computation Conf.* (GECCO-2006). ACM, Seattle, USA, 2006. P. 419–426.

196. Savaton B., Casseau E., Martin E. Behavioral VHDL Styles and High Level Synthesis for IPs. Forum on Design Languages, 2000. P. 4–8.

197. Schaffer J. D. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. *Proc. 1-st Int. Conf. on Genetic Algorithms and Their Applications*. 1985. P. 93–100.

198. Schmid M., Schmitt C., Hannig F., Malazgirt G. A., Sonmez N., Yurdakul A., Cristal A. Big Data and HPC Acceleration with Vivado HLS. *FPGAs for Software Programmers.* / D. Koch, F. Hannig, D. Ziener, Ed-s. Springer. 2016. P. 115–136.

199. Schmidt M., Lipson H. Age-fitness Pareto optimization. *Genetic Programming Theory and Practice VIII. Genetic and Evolutionary Computation* / Riolo, R., McConaghy, T., Vladislavleva, E., Ed-s. Springer, Ann Arbor, 2010. V. 8. P. 129–146.

200. Seawright A., Buck J., Holtmann U., Meyer W., Pangrle B., Verbrugghe R. A system for compiling and debugging structured data processing

controllers. *Proc. European Design Automation Conf.* Switzerland, Geneva, Sept. 1996. P. 86–91.

201. Sergiyenko A., Vasylienko V., Maslennikow O. FIR filter soft core generator. *Prace IV Konferencji Krajowej Reprogramowalne uklady cyfrowe*, (RUC'2001). Szczecin, Poland. 2001. P. 167–172.

202. Sergiyenko A., Serhienko A., Romankevich V. Genetic Programming of Pipelined Datapaths for FPGA. *Proc. IEEE 40-th Intern. Conf. on Electronics and Nanotechnology* (ELNANO'2020). 2020. P. 802–806.

203. Serhienko A., Sergiyenko A. Computing Pythagorean triples in FPGA. *High Performance Computing*, (HPC-UA'2013) : Proc. 3-d Int. Conf. Kyiv, 2013. P. 347–349.

204. Serhienko A. VHDL design of multiplier-free IIR filters, Kiev: Sikorsky's KPI, 2016. URL: http://kanyevsky.kpi.ua/GEN_MODUL/APgen/APMF_help.php

205. Serhienko A., Sergiyenko A. Method of the Digital Filter Design using VHDL. *Winter InfoCom Advanced Solutions 2016* : Proc. Int. Conf. 2016. P. 68–69.

206. Serhienko A., Sergiyenko A. Modules for pipelined mixed radix FFT processors. *Int. J. of Reconfigurable Computing*. Hindawi Publishing Corp. 2016. V. 2016. P. 1–7.

207. Serhienko A., Sergiyenko A., Simonenko A. A method for synchronous dataflow retiming. *IEEE 1-st Ukraine Conf. on Electrical and Computer Engineering (UKRCON).* Kyiv, 2017. P. 1015–1018.

208. Serhienko A., Sergiyenko A. Digital Filter Design using VHDL. *High Performance Computing,* (HPC-UA 2018) : Proc. 5-th Int. Conf. 2018. P. 123–126.

209. Serhienko A., Sergiyenko A. VHDL Generation of Optimized IIR Filters. *IEEE 2-nd Ukraine Conference on Electrical and Computer Engineering, (UKRCON)*, Lviv, Ukraine, July 2–6, 2019. P. 1171–1174.

210. Serhienko A., Sergiyenko A. Romankevich V. Genetic Programming of Pipelined Datapaths for FPGA. *IEEE 40-th Int. Conf. on Electronics and Nanotechnology,* (ELNANO). Kyiv, 2020. P. 802–806.

211. Serhienko A., Sergiyenko A., Ukpu M. Multiplierless IIR Filter Design for FPGA. *3-d Int. Conf. on Security, Fault Tolerance, Intelligence (ICFTI2020)*. Kyiv, 2020. P. 1–9.

212. Sethi R. Complete register allocation problems. *SIAM Journal on Computing*. 1975. V. 4, No. 3. P. 226–248.

213. Shazeeda, Shashidhar T. M., Anupama B. G., Chandini A. C. FPGA Based Design of 1D DCT/IDCT for MIMO OFDM Channel Estimation. European Scientific Journal. 2014. Vol. 10, No 30. P. 88–109. URL: https://eujournal.org/index.php/esj/article/view/4443/4252.

214. Shilpa K. C., Lakshminarayan C., Manoj K. S. Adaptive Differential Evolution for Optimal Schedule in Behavioral Level Synthesis. *IOSR Journal of VLSI and Signal Processing* (IOSR-JVSP). 2016. V. 6, No. 4. P. 55–59.

215. Smith III J. O. Physical Modeling Using Digital Waveguides. *Computer Music Journal,* 1993. V. 16, № 4. P. 74–91.

216. Smith R. E., Dike B. A., Stegmann S. A. Fitness inheritance in genetic algorithms. *Symposium on Applied computing* (SAC), New York, NY, USA: ACM Press. 1995. P. 345–350.

217. Spartan-6 FPGA DSP48A1 Slice User Guide. UG389 (v1.2). *Xilinx Inc*. May 29, 2014. 46 p.

218. Springer D. L., Thomas D. E. Exploiting the Special Structure of Conflict and Compatibility Graphs in High-Level Synthesis. *Proc. Int. Conf. Computer Aided Design* (ICCAD). 1990. P. 254–257.

219. Storn R., Price K. Differential evolution: A simple and efficient adaptive scheme of global optimization over continuous spaces. *Journal of Global Optimization*. 1997. P. 341–359.

220. Sutherland S., Davidmann S., Flake P. SystemVerilog For Design: A Guide to Using SystemVerilog for Hardware Design and Modeling. Springer Science & Business Media, 2013. 374 p.

221. Swamy G. Incremental Methods for Formal Verification and Logic Synthesis. *PhD thesis*, University of California at Berkeley, 1996. 182 p.

222. System Generator for DSP. Getting Started Guide. August, 2007. 85p. URL: http://www.xilinx.com.

223. Tan L. The Group of Rational Points on the Unit Circle. *Mathematics Magazine*. 1996. V. 69, No. 3. P. 163–171.

224. Tensilica DNA Processor IP for AI Inference Industry-leading Performance and Power Efficiency for On-device AI Applications. 2019. 2 p. URL: www.cadence.com.

225. The OMG Consortium, Uniled Modeling Language™ (UML®) Resource Page. OMG Adopted Specification, September, 9, 2013. URL: http://www.uml.org.

226. The Synthesis Approach to Digital System Design / P. Michel, U. Lauther, P. Duzy, Ed-s. Kluwer Academic Pub. 1992. 415 p.

227. Theelen B. D., Geilen M. C., Basten T., Voeten J. P., Gheorghita S. V., Stuijk S. A scenario-aware dataflow model for combined long-run average and worst-case performance analysis. *Proc. Int. Conf. on Formal Methods and Models for Codesign*. Washington DC, 2006. P. 185 – 194.

228. Tseng C., Siewiorek D. Automated Synthesis of DataPaths in Digital Systems. *IEEE Trans. on Computer-Aided Design*. 1986. V. 5, No. 3. P. 274–277.

229. Tucker A. Coloring a family of circular arcs. *SIAM J. Appl. Math*. 1975. V. 29, No. 3. P. 493–502.

230. Ullman J. NP-Complete Scheduling Problems. *J. Comput. Syst. Sci*. 1975. V. 10, No. 3. P. 384–393.

231. Vakali A., Jain L. C. New Directions in Web Data Management. Berlin, Heidelberg: Springer. 2011. 347 p.

232. Verhaegh W. F. J., Lippens P. E. R., Aarts E. H. L., Korst J. H. M., van der Werf A., Meerbergen J. L. Efficiency improvements for force-directed scheduling. *IEEE/ACM international conference proceedings on Computer-aided design*, (ICCAD'92). 1992. P. 286–291.

233. Virtex-6 Family Overview. Advance Product Specification. DS150 (v1.2). June 24, 2009. 9 p. URL: http://www.xilinx.com

234. Vitányi P. A discipline of evolutionary programming. *Theoretical Computer Science*. 2000. Vol. 241, Issues 1–2. P. 3–23. DOI: https://doi.org/10.1016/S0304-3975(99)00263-7.

235. Weise T. Global Optimization Algorithms. Theory and Application. Version: 2009-06-26. 820 p. URL: http://www.it-weise.de/

236. Yang X.-S., Nature-Inspired Optimization Algorithms. London: Elsevier. 2014.

237. Yeung K. S., Chan S. C. The Design and Multiplier-Less Realization of Software Radio Receivers With Reduced System Delay. *IEEE Trans. On Circuits and Systems Regular Papers.* 2004. V. 51. P. 2444–2449.

238. Yli-Kaakinen J., Anzova V. I., Saramaeki T. An Algorithm for the Design of Multiplierless IIR Filters as a Parallel Connection of Two All-Pass Filters. IEEE Asia Pacific Conf. on Circuits and Systems, (APCCAS). 2006. P. 744-747.

239. Yviquel, H. Lorence A., Jerbi K., Cocherel G. Orcc: Multimedia development made easy. *Proc. ACM Int. Conf. on Multimedia*. 2013. P. 233–242.

240. Zorian Y. Embedded memory test and repair: infrastructure IP for SOC yield. *Proc. Int. Test Conf.* Baltimore, MD, USA, 2002. P. 340–349.

ДОДАТОК А. Лістинг опису мовою VHDL процесора для розв'язання диференційних рівнянь

```
library   IEEE;
use IEEE.Std_logic_1164.all;
entity ODE_1 is port(CLK,RST,START:in std_logic;
                XO:out integer);
end entity ODE_1;

architecture synt of ODE_1 is
        constant x0:integer:=100;
        constant u0:integer:=0;
        constant y0:integer:=200;
        constant eps:integer:=4000;
        constant dx: integer:=10;
        constant b: integer:= 5;
        constant c: integer:= 6;
        signal ct5: integer range 0 to 4;
        signal r3,r4,sm2,mpu12: integer;
        signal rdy:std_logic;
begin
        CNTRL: process(CLK,RST)
        Begin
                if RST='1' then ct5<=0;rdy<='1';
                elsif CLK='1' and CLK'event then
                        if ct5=4 then ct5<=0; -- clock counter
                        else ct5<=ct5+1;
                        end if;
                        if START='1' and ct5=4 then
                                rdy<='0';
                        elsif ct5=4 and sm2>0 then
                                rdy<='1';
                        end if;
                end if;
        end process;

        DATAPATH: process(CLK,RST)-- datapath
        begin
                if CLK='1' and CLK'event then
                        if START='1' and ct5=4 then
                                  r3<=u0; r4<=x0;sm2<=y0 + u0*dx; mpu12<=u0*dx*b;
                        elsif rdy='0' then
                                case ct5 is
                                        when 0 => r4<=mpu12;mpu12<=r4*c;r3<=sm2;r3<=sm2;
                                        when 1 => r0<=mpu; mpu<= r0*c; sm<= r5+dx;
                                        when 2 => r0<=mpu; sm<=r5-r0;r7<=x0; r6<=r7;XO<=sm;
                                        when 3 => sm<=sm-r0;
                                        when
others=>r0<=mpu;mpu<=r0*b;sm<=r0+r5;r6<=r7;r5<= r6;
                                end case;
                        end if;
                end if;
        end process;
end synt;
```

ДОДАТОК Б. Лістинг опису мовою VHDL процесора ДКП

```
--------------------------------------------------------------------------------
--
-- Title       : DCT8_1
-- Design      : DCT8
-- Author      : Anastasia Serhienko
-- Company     : KPI
--
--------------------------------------------------------------------------------
--
-- File        : DCT8_1.vhd
-- Generated   : Sat Jul 18 08:57:43 2020
-- From        : interface description file
-- By          : SDFLAB ver. 1.2
--
--------------------------------------------------------------------------------
--
-- Description :   DCT processor derived from the automatic synthesis
--                                              16-bit data
--------------------------------------------------------------------------------
-- Kintex7 Vivado
-- 319 LUT  541 RG  156  CLB  3 DSP 2.700 ns area opt   DSP assigned
-- in ISE
-- 537 LUT  518 RG  241  CLB  4 DSP 3.370 ns    DSP strongly constrained
-- 275 LUT  503 RG  220  CLB  3 DSP 3.670 ns area opt   DSP assigned
-- 263 LUT  503 RG  227  CLB  3 DSP 3.74 ns speed opt retiming  DSP assigned

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_signed.all;
use IEEE.STD_LOGIC_arith.all;

entity DCT8_1 is
        port(
                CLK : in STD_LOGIC;
                RST : in STD_LOGIC;
                START : in STD_LOGIC;
                DATA_IN : in STD_LOGIC_VECTOR(15 downto 0);
                RDY : out STD_LOGIC;
                DATA_OUT : out STD_LOGIC_VECTOR(15 downto 0)
                );
end DCT8_1;


architecture synt of DCT8_1 is
        signal  rg2,rg3,rg5,rg6,rg9,rg10,rg13,  rg14,rg17,rg19,rg20,rg23,rg25,rg26:  std_logic_vector(15
downto 0);
        signal rg35,rg36,rg38,rg40,rg41: std_logic_vector(15 downto 0);
        signal rg44: std_logic_vector(15 downto 0);
        signal rg46,rg48, rg49: std_logic_vector(15 downto 0);
        signal rg57,rg59,rg61,rg62: std_logic_vector(15 downto 0);
        signal rg66,rg67,rg68: std_logic_vector(15 downto 0);
        signal rg73,rg71: std_logic_vector(15 downto 0);
        signal rg78,rg79,rg80: std_logic_vector(15 downto 0);
        signal rg85,rg86: std_logic_vector(15 downto 0);
```

```
	signal rg84,rg87,rg89,rg90,rg92,rg93,rg94: std_logic_vector(15 downto 0);
	signal sm35: std_logic_vector(15 downto 0);
	signal sm56: std_logic_vector(15 downto 0);
	signal sm70: std_logic_vector(15 downto 0);
	signal sm77: std_logic_vector(15 downto 0);
	signal sm91: std_logic_vector(15 downto 0);
	signal mpu43i: std_logic_vector(15+12 downto 0);
	signal mpu66i: std_logic_vector(15+12 downto 0);
	signal mpu88i: std_logic_vector(15+12 downto 0);
	signal mpu43: std_logic_vector(15 downto 0);
	signal mpu66: std_logic_vector(15 downto 0);
	signal mpu88: std_logic_vector(15 downto 0);


	constant d1: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.7071*2**10),12);
	constant d12: std_logic_vector(11 downto 0):=
conv_std_logic_vector(integer(0.7071*2**11),12);
	constant d2: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.924*2**10),12);
	constant d3: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.3827*2**10),12);
	constant d4: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.990*2**10),12);
	constant d5: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.139*2**10),12);
	constant d6: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.900*2**10),12);
	constant d7: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.437*2**10),12);
	constant d22: std_logic_vector(11 downto 0):= conv_std_logic_vector(integer(0.924*2**11),12);
	constant d32: std_logic_vector(11 downto 0):=
conv_std_logic_vector(integer(0.3827*2**11),12);

	--signal mpu37,rg20,rg20: std_logic_vector(16 downto 0);
	signal st: natural range 0 to 7;
	signal ct2: natural range 0 to 31;

begin

	FSM: process(CLK,RST) begin
		if CLK='1' and CLK'event then
			if RST='1' then
				st<=0;
				ct2<=0;
				RDY<='0';
			elsif START ='1' then
				st<=0;
				RDY<='0';
				ct2<=0;
			else
				if st=7 then
					st<= 0;
				else
					st<= st + 1;
				end if;
				if ct2 = 24 then
					RDY<='1';
				elsif ct2<= 25 then
					ct2<= ct2+1;
				end if;
			end if;
		end if;
```

```
end process;


DATAPATH:
        process(CLK,RST,mpu43,mpu66,mpu88)
        variable m1,m2,m3:std_logic_vector(11 downto 0);
        variable n1:std_logic_vector(15 downto 0);
        variable n2:std_logic_vector(15 downto 0);
        variable n3:std_logic_vector(15 downto 0);

begin
        mpu43<=mpu43i(25 downto 10);
        mpu66<=mpu66i(25 downto 10);
        mpu88<=mpu88i(25 downto 10);


        if CLK='1' and CLK'event then
                if RST='1' or START ='1' then
                        rg25<=x"0000";rg2<=x"0000";rg3<=x"0000";rg5<=x"0000";
                        rg6<=x"0000";rg9<=x"0000";rg10<=x"0000";rg13<=x"0000";
                        rg14<=x"0000";rg17<=x"0000";rg19<=x"0000";rg20<=x"0000";
                        rg23<=x"0000";rg26<=x"0000";


                else
                        m1:=d4; n1:=sm35;     --for logic
                        m2:= d2; n2:=sm56;
                        m3:= d1; n3:= rg85;
                        case st is
                                when 0 =>
                                        rg25 <= DATA_IN;
                                        rg26 <=rg25;
                                        sm35 <= SXT(rg23,16) + rg25;
                                        --        mpu43i<= d4*sm35;
                                        m1:=d4; n1:=sm35;
                                        sm56 <= rg49 + rg48;
                                        rg59 <= sm56 ;
                                        rg68 <= mpu66;
                                        sm77 <= SXT(rg68,16) + rg71;
                                        rg84 <=  sm77;
                                        --        mpu88i<= d1*rg85;
                                        m3:= d1; n3:= rg85;
                                        DATA_OUT<=rg93;
                                when 1 =>
                                        rg19 <= DATA_IN;
                                        rg20 <=rg19;
                                        rg14 <=rg13;
                                        sm35 <= SXT(rg23,9) - rg26;
                                        rg41 <= sm35;
                                        rg44 <= mpu43;
                                        sm56 <= SXT(rg38,16) + rg36;
                                        --        mpu66i<= d2*sm56;
                                        m2:= d2; n2:=sm56;
                                        sm77 <= SXT(rg71,16) + rg73;
                                        rg78 <= sm77;
                                        sm91 <= mpu88 - rg80;
```

```
        DATA_OUT<=rg90;
when 2 =>
        rg13 <= DATA_IN;
        rg14 <=rg13;
        sm35<= SXT(rg20,16) + rg17 ;
        --mpu43i<= d6*sm35;
        m1:=d6 ; n1:=sm35;
        sm56 <= SXT(rg36,16) - rg38 ;
        sm77 <= SXT(rg62,16) + sm56  ;
        rg61 <= sm56;
        sm70 <= mpu66 - rg59 ;
        --        mpu88i<= d1*sm77;
        m3:= d1; n3:=sm77;
        rg94 <= sm91;
        DATA_OUT<=rg79;
when 3 =>
        rg5 <= DATA_IN;
        rg6 <=rg5;
        sm35 <= SXT(rg20,16) - rg17;
        rg38 <= sm35;
        rg49 <= mpu43;
        sm56 <= rg44 + rg46;
        --mpu66i<= d3*sm56;
        m2:=d3; n2:=sm56;
        rg73 <= sm70;
        sm77 <= SXT(rg61,16) + rg62;
        rg87 <= sm77;
        sm91 <= mpu88 - rg79;
        DATA_OUT<=rg80;

when 4 =>
        rg2 <= DATA_IN;
        rg3 <=rg2;
        sm35 <= SXT(rg6,16) + rg2;
        --mpu43i<= d5*sm35;
        m1:=d5; n1:=sm35;
        sm56 <= rg44 - rg46;
        rg57 <= sm56 ;
        sm77 <= SXT(mpu66,16) + rg68;
        rg67 <= mpu66;
        rg86 <= sm77;
        --mpu88i<= d1*rg84;
        m3:=d1; n3:=rg84;
        rg93 <= sm91;
        DATA_OUT<=rg78;

when 5 =>
        rg9 <= DATA_IN;
        rg10 <=rg9;
        sm35 <= SXT(rg6,16) - rg3;
        sm56 <= SXT(sm35,16) - rg41   ;
        rg40 <= sm35;
        rg46 <= mpu43;
        --mpu66i<= d3*sm56;
        m2:=d32; n2:=sm56;
        sm77 <= SXT(rg68,16) + rg67;
```

```
                                        rg80 <= sm77;
                                        --      mpu88i<= d12*rg87;
                                        m3:=d12; n3:=rg87;
                                        sm91 <= mpu88 - rg78;
                                        DATA_OUT<=rg89;


                                    when 6 =>
                                        rg17 <= DATA_IN;
                                        sm35 <= SXT(rg14,16) + rg10;
                                        --      mpu43i<= d7*sm35;
                                        m1:=d7; n1:=sm35;
                                        sm56 <= sxt(rg41,16)  + rg40;
                                        rg62 <= sm56;
                                        sm77 <= SXT(rg59,16) + rg57;
                                        sm70 <= mpu66 - rg57 ;
                                        rg85 <= sm77;
                                        --mpu88i<= d1*rg86;
                                        m3:=d12; n3:=rg86;
                                        rg90 <= mpu88;
                                        rg92 <= sm91;
                                        DATA_OUT<=rg92;

                                    when others =>
                                    rg23 <= DATA_IN;
                                    sm35 <= SXT(rg14,16) - rg10;
                                    rg36 <= sm35;
                                    sm56 <= mpu43 - rg49;
                                    rg48 <= mpu43;
                                    --mpu66i<= d2*sm56;
                                    m2:=d22; n2:=sm56;
                                    rg71 <= sm70;
                                    sm77 <= SXT(rg59,16) + rg57;
                                    rg79 <= sm77;
                                    rg89 <= mpu88;
                                    DATA_OUT<=rg94;
                            end case;
                        end if;
                        mpu43i<= m1*n1;
                        mpu66i<= m2*n2;    --d2*sm56;
                        mpu88i<= m3*n3;
                end if;
        end process;

end synt;
```

ДОДАТОК В. Лістинг опису мовою VHDL 3-точкового ДПФ

```
--------------------------------------------------------------------------------
--
-- Title       : DFT3
-- Design      : FFT12
-- Author      : Anastasia Serhienko
-- Company     : KPI
--
--------------------------------------------------------------------------------
--
-- File        : DFT3.vhd
-- Generated   : Sun May  3 17:07:15 2016
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.22
--
--------------------------------------------------------------------------------
--
-- Description :              3-point FFT
--    START    /\_____
--    f1       __/\___
--    RDY                   _/\___
--    F0                    ___/\__
--Virtex5
-- 307 LUT  84 CLBs 3.711ns
-- 332 LUT 105 CLBs 3.336 ns + 1 stage+compress
-- 351 LUT 117 CLBs 3.309 ns + 1 stage+retim+compress
-- 409 LUT 186 CLBs 2.975 ns + 1 stage+retim-compress
-- 403 LUT 130 CLBs 3.030 ns + 1 stage-retim-compress
-- 228 LUT  72 CLBs 3.606 ns + 1 DSP
-- 210 LUT  64 CLBs 3.999 ns + 1 DSP +compress
-- 228 LUT  72 CLBs 3.606 ns + 1 DSP


--Spartan3DSP
-- 462 LUT 277 CLBs 6.578 ns + 1 stage
-- 246 LUT 159 CLBs 6.339 ns + 1 DSP
--Kintex7
-- 281 LUT 104 CLBs 2.305 ns -retim-compress
-- 250 LUT  68 CLBs 2.363 ns -retim+compress
-- 238 LUT  68 CLBs 1.408 ns + 1 stage-retim+compress
-- 239 LUT  69 CLBs 1.408 ns + 1 stage-retim-compress
-- 201 LUT  82 CLBs 2.308 ns + 1 DSP
-- 194 LUT  55 CLBs 2.308 ns + 1 DSP
--Spartan-6
-- 278 LUT  93 CLBs 3.367 ns -retim-compress
-- 264 LUT  72 CLBs 3.957 ns -retim+compress
-- 238 LUT  68 CLBs 3.463 ns + 1 stage-retim+compress
-- 231 LUT  79 CLBs 3.354 ns + 1 stage-retim-compress
-- 257 LUT  82 CLBs 3.771 ns + 1 DSP
-- 214 LUT  59 CLBs 3.960 ns + 1 DSP+compress


--------------------------------------------------------------------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_logic_arith.all;
--use IEEE.STD_logic_signed.all;

entity DFT3 is
	port(
		CLK : in STD_LOGIC;
		RST : in STD_LOGIC;
		START : in STD_LOGIC;
		DRI : in STD_LOGIC_VECTOR(15 downto 0);
		DII : in STD_LOGIC_VECTOR(15 downto 0);
		RDY : out STD_LOGIC;
		DRO : out STD_LOGIC_VECTOR(17 downto 0);
		DIO : out STD_LOGIC_VECTOR(17 downto 0)
		);
end DFT3;

architecture synt of DFT3 is
	signal	s1r,s2r,s3r,s5r,s6r,r1r,r2r: signed(17 downto 0);
	signal	s1i,s2i,s3i,s5i,s6i,r1i,r2i,rm,rr: signed(17 downto 0);
	signal s4r,s4i :signed(17 downto 0);
	signal p,pd:signed(35 downto 0);

	signal cyc: natural range 0 to 3;
	signal del: natural range 0 to 7;

begin
	CNTRL:process(CLK) begin
		if rising_edge(CLK) then
			if RST='1' then
				cyc<=0;
				del<=0;
			elsif START ='1' then
				cyc<=0;
				del<=0;
			else
				if cyc=2 then
					cyc<=0;
				else
					cyc<=cyc +1;
				end if;
				if del /= 4 then
					del<=del+1;
				end if;
			end if;
		end if;
	end process;
	RDY<='1' when del = 3 else '0';

   CALC: process(CLK) begin
     if rising_edge(CLK) then
  case cyc is
    when 0 =>
      S1r<=  signed(SXT(DRI,s1r'length));
      S1i<=  signed(SXT(DII,s1I'length));
```

```
        S2r<=S1r - S2r;
        S2i<=S1i - S2i;
        R2r<= S1r;
        R2i<= S1i;
        S4r<= SHR(S3r,"010") - R1r ;
        S4i<= SHR(S3i,"010") - R1i ;
        S5r<=R1r - SHR(S4r,"011");
        S5i<=R1i - SHR(S4i,"011");
        S6r<= R2r - S5i;
        S6i<= R2i + S5r;
      when 1 =>
        S1r<=  S1r + signed(DRI);
        S1i<=  S1i + signed(DII);
        R2r<= S2r;
        R2i<= S2i;
        S3r<= S1r - signed(DRI);
        S3i<= S1i - signed(DII);
        S5r<=S5r + SHR(S4r,"1010");
        S5i<=S5i + SHR(S4i,"1010");
        S6r<=R2r;
        S6i<=R2i;
      when others=>
        S1r<=  S1r + signed(DRI);
        S1i<=  S1i + signed(DII);
        S2r<= S1r + SHR(S1r,"001");
        S2i<= S1i + SHR(S1i,"001");
        S3r<= SHR(S3r,"010") - S3r;
        S3i<= SHR(S3i,"010") - S3i ;
        S4r<= S3r + SHR(S3r,"0100") ;
        S4i<= S3i + SHR(S3i,"0100") ;
        R1r<=S3r;
        R1i<=S3i;
        S6r<= R2r + S5i;
        S6i<= R2i - S5r;
       end case;
      end if;
    end process;

                                    DRO<=std_logic_vector(S6r(17 downto 0));
                                    DIO<=std_logic_vector(S6i(17 downto 0));
end synt;
```

ДОДАТОК Г. Лістинг опису мовою VHDL 5-точкового ДПФ

```
--------------------------------------------------------------------------------
--
-- Title       : DFT5
-- Design      : FFT12
-- Author      : CAM
-- Company     : KPI
--
--------------------------------------------------------------------------------
--
-- File        : DFT5.vhd
-- Generated   : Sun july  3 17:07:13 2015
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.22
--
--------------------------------------------------------------------------------
--
-- Description :              5-point FFT
--      START /\_____
--      f1       __/\___
--     RDY                   _/\___
--     F0                    ___/\__
-- SNR   <2e-5

--Spartan3DSP
--1869 LUT 1028 CLBs 8.83 ns

--Kintex7
-- 824 LUT 305 CLBs 2.122 ns -retim-compress
-- 916 LUT 239 CLBs 2.373 ns -retim+compress
--Spartan-6
--1228 LUT 471 CLBs 4.582 ns -retim-compress
--1165 LUT 326 CLBs 4.607 ns -retim+logmin
-- 994 LUT 258 CLBs 5.713 ns -retim+compress
--1082 LUT 279 CLBs 5.315 ns +retim+compress
--1022 LUT 374 CLBs 5.544 ns +retim+logmin

--------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_logic_arith.all;
use IEEE.STD_logic_signed.all;

entity DFT5 is
        generic(nr: natural :=16);
        port(
                CLK : in STD_LOGIC;
                RST : in STD_LOGIC;
                START : in STD_LOGIC;
                DRI : in STD_LOGIC_VECTOR(nr-1 downto 0);
                DII : in STD_LOGIC_VECTOR(nr-1 downto 0);
                RDY : out STD_LOGIC;
                DRO : out STD_LOGIC_VECTOR(nr+2 downto 0);
```

```
                DIO : out STD_LOGIC_VECTOR(nr+2 downto 0)
                );
end DFT5;

architecture synt of DFT5 is
        signal r1r,r2r,r3r,r1i,r2i,r3i: STD_LOGIC_VECTOR(nr-1 downto 0);
        signal s1r,r4r,r5r,s3r: STD_LOGIC_VECTOR(nr+1 downto 0);
        signal s1i,s3i,r4i,r5i: STD_LOGIC_VECTOR(nr+1 downto 0);
        signal s2r,s4r,s5r,r6r,r7r,r8r,r9r,r10r: STD_LOGIC_VECTOR(nr+2 downto 0);
        signal s2i,s4i,s5i,r6i,r7i,r8i,r9i,r10i,s15r,s15i: STD_LOGIC_VECTOR(nr+2 downto 0);
        signal s7r,s7i: STD_LOGIC_VECTOR(nr+1 downto 0);
        signal s6r,s6i,s8r,s8i,s9r,s9i,s10r,s10i: STD_LOGIC_VECTOR(nr+2 downto 0);
        signal s11r,s11i,s12r,s12i :STD_LOGIC_VECTOR(nr+3 downto 0);

        signal cyc: natural range 0 to 5;
        signal del: natural range 0 to 15;

begin
        CNTRL:process(CLK) begin
                if rising_edge(CLK) then
                        if RST='1' then
                                cyc<=0;
                                del<=0;
                        elsif START ='1' then
                                cyc<=0;
                                del<=0;
                        else
                                if cyc=4 then
                                        cyc<=0;
                                else
                                        cyc<=cyc +1;
                                end if;
                                if del /= 10 then
                                        del<=del+1;
                                end if;
                        end if;
                end if;
        end process;
        RDY<='1' when del = 9 else '0';


        CALC: process(CLK) begin
                if rising_edge(CLK) then
                        if RST='1' then
                                S1r<=(others=>'0');     S2r<=(others=>'0');
                                S3r<=(others=>'0');     S4r<=(others=>'0');
                                S5r<=(others=>'0');     S6r<=(others=>'0');
                                S7r<=(others=>'0');     S8r<=(others=>'0');
                                S9r<=(others=>'0');     S10r<=(others=>'0');
                                S11r<=(others=>'0');    S12r<=(others=>'0');
                                S1i<=(others=>'0');     S2i<=(others=>'0');
                                S3i<=(others=>'0');     S4i<=(others=>'0');
                                S5i<=(others=>'0');     S6i<=(others=>'0');
                                S7i<=(others=>'0');     S8i<=(others=>'0');
                                S9i<=(others=>'0');     S10i<=(others=>'0');
                                S11i<=(others=>'0');    S12i<=(others=>'0');
```

```
            R1r<=(others=>'0');       R2r<=(others=>'0');
            R1i<=(others=>'0');       R2i<=(others=>'0');
            R3r<=(others=>'0');       R4r<=(others=>'0');
            R3i<=(others=>'0');       R4i<=(others=>'0');
            R5r<=(others=>'0');
            R5i<=(others=>'0');
else
        R7r<= S4r;
                        R7i<= S4i;
        R8r<= R7r;
                        R8i<= R7i;
        R9r<= R8r;
                    R9i<= R8i;
                        R3r<=DRI;
                        R3i<=DII;
                        R4r<= S1r;
                        R4i<= S1i;
                         R5r<= S3r;
                        R5i<= S3i;

            case cyc is
                    when 0 =>
                        R1r<=   DRI;
                        R1i<=   DII;
                        S1r<=  S1r - R4r;
                        S1i<=  S1i - R4i;
                        S2r<=  S1r(nr+1)&S1r + R4r;
                        S2i<=  S1i(nr+1)&S1i + R4i;
                        S3r<=  S3r + R5r;
                        S3i<=  S3i + R5i;
                        S4r<=  SXT(R1r,S4r'length);
                        S4i<=  SXT(R1i,S4r'length);
                        S8r<=  SXT(S3r,S8r'length-1)&'0' + S3r;
                        S8i<=  SXT(S3i,S8r'length-1)&'0' + S3i;
                        S10r<= SXT(R5r,S10r'length-1)&'0' + SHR(R5r,"100");
                        S10i<= SXT(R5i,S10r'length-1)&'0' + SHR(R5i,"100");

                        R6r<= S5r;
                        R6i<= S5i;

                        S12r<= S11i + SHL(S4r,"01") ;
                        S12i<= SHL(S4i,"01")  - S11r ;


                    when 1 =>
                        R2r<=   DRI;
                        R2i<=   DII;
                        S2r<=  S2r + SHR(S2r,"010");
                        S2i<=  S2i + SHR(S2i,"010");
                        S4r<=  S4r + S2r;
                        S4i<=  S4i + S2i;
                        S5r<=SXT(S1r,S5r'length) +SHR(S1r,"011");
                        S5i<=SXT(S1i,S5r'length) +SHR(S1i,"011");

                        S6r<= S3r&'0' - SHR(S3r,"011");
                        S6i<= S3i&'0' - SHR(S3i,"011");
```

```
                S7r<= SHR(S3r,"001")  - SHR(S3r,"100");
                S7i<= SHR(S3i,"001")  - SHR(S3i,"100");
                S8r<= S8r - SHR(S8r,"101");
                S8i<= S8i - SHR(S8i,"101");
                S10r<= SHR(S10r,"01") - SHR(S10r,"0111") ;
                S10i<= SHR(S10i,"01") - SHR(S10i,"0111") ;

                R10r<= S10r;
                R10i<= S10i;
                S12r<=  SXT(S9i,S12r'length) + SHL(S15r,"01") ;
                S12i<=  SHL(S15i,"01")  - SXT(S9r,S12r'length) ;


                when 2 =>
                S1r<=  SXT(DRI,S1r'length);
                S1i<=  SXT(DII,S1I'length);
                S4r<=  S4r - S2r;
                S4i<=  S4i - S2i;
                S5r<=S5r - SHR(S1r,"110");
                S5i<=S5i - SHR(S1i,"110");

                S7r<= S7r - SHR(S3r,"1000");
                S7i<= S7i - SHR(S3i,"1000");
                S10r<= S10r + R10r;
                S10i<=          S10i + R10i;
                S12r<=  SHL(S15r,"01") - SXT(S9i,S12r'length);
                S12i<=  SHL(S15i,"01")  + SXT(S9r,S12r'length) ;



                when 3 =>
                S1r<= S1r + DRI;
                S1i<= S1i + DII;
                S3r<= DRI - S1r;
                S3i<= DII - S1i;

                S5r<= SHR(S5r,"001") + SHR(S5r,"1000");
                S5i<= SHR(S5i,"001") + SHR(S5i,"1000");
                S6r<= S6r + SHR(S7r,"0100");
                S6i<= S6i + SHR(S7i,"0100");

                S12r<=  SHL(R8r,"01")  - S11i;
                S12i<=  SHL(R8i,"01") + S11r;


        when others=>
                S1r<= SXT(R2r,S1R'length) + DRI;
                S1i<=  SXT(R2i,S1R'length)  + DII;
                --R4r<= S1r;
--              R4i<= S1i;
                S3r<= SXT(R2r,S1R'length) - DRI;
                S3i<= SXT(R2i,S1R'length) - DII;
                S4r<=  S4r + S5r;
                S4i<=  S4i + S5i;
                S15r<=  S4r - S5r;
                S15i<=  S4i - S5i;
```

```
                                        R5i<= S3i;
                                        S9r<=S6r - SHR(S8r,"010");
                                        S9i<=S6i - SHR(S8i,"010");
                                        S11r<=  S6r - SXT(S10r,S11r'length);
                                        S11i<=  S6i - SXT(S10i,S11r'length);

                                        S12r<=SXT(R8r,S12r'length-1)&'0'        ;
                                        S12i<=SXT(R8i,S12r'length-1)&'0'        ;


                                end case;
                        end if;
                end if;
        end process;



        DRO<=S12r(nr+3 downto 1);
        DIO<=S12i(nr+3 downto 1);
end synt;
```

ДОДАТОК Д. Лістинг опису мовою VHDL згенерованого фільтру без помножувачів

```
--------------------------------------------------------------------------------
--
-- Title   : APHBF
-- Design  : AllpassLib
--------------------------------------------------------------------------------
--
-- File    : APHBF_318_52.vhd
-- Generated: 13th January 2017 19 11
-- Разработано лабораторией Каневского      : kanyevsky.kpi.ua site
--
-- Все права защищены . Этот файл используется как есть.
-- Файл в свободном доступе. Не убирать этот заголовок.
--------------------------------------------------------------------------------
--
-- Description : Тип ФИЛЬТРА - Полуполосный
--     DI = 24 bits, DO = 24 bits - low pass output
--     Cutoff frequency = 0.318 of the sampling frequency
--     Supression level > 52.6 db
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;
entity APHBF_318_52 is
port(
   CLK : in STD_LOGIC;
   DI : in STD_LOGIC_VECTOR(23 downto 0);
   DO : out STD_LOGIC_VECTOR(23 downto 0)   );
end APHBF_318_52;
architecture synt of APHBF_318_52 is
signal s1d,s1db, s1dbd,s1d2,s1d3,s1d4:Signed(27 downto 0):= (others=>'0');
signal x1,x1d,x1d2,x1d3,x1d4,x1d5: Signed(26 downto 0):= (others=>'0');
signal y1d,y1dh,y1t:      Signed(27 downto 0):=(others=>'0');
signal s2d,s2dc, s2dcd,s2d2,s2d3,s2d4:Signed(29 downto 0):= (others=>'0');
signal x2,x2d,x2d2,x2d3: Signed(26 downto 0):= (others=>'0');
signal y2d,y2dh,y2t:      Signed(29 downto 0):=(others=>'0');
signal s3d,s3da,s3dad,s3d2,s3d3,s3d4:Signed(27 downto 0):= (others=>'0');
signal x3: Signed(26 downto 0):= (others=>'0');
signal y3d,y3dh,y3dt:    Signed(27 downto 0):=(others=>'0');

begin
  s1db <=   SHIFT_RIGHT(s1d,2) + SHIFT_RIGHT(s1d,3);

 process(CLK) begin
  if rising_edge(CLK) then
    x1<=signed(DI&"000");
    x1d<=x1; x1d2<=x1d; x1d3<=x1d2;
    x1d4<=x1d3; x1d5<=x1d4;
    s1d2<=s1d; s1d3<=s1d2; s1d4<=s1d3; s1dbd<= s1db;
    s1d<=x1d5 - s1dbd; y1d<=s1dbd + s1d4;
  end if;
 end process;

  s2dc <=   SHIFT_RIGHT(s2d,1) + SHIFT_RIGHT(s2d,2);
```

```
 process(CLK) begin
  if rising_edge(CLK) then
    x2<=signed(DI&"000");
    x2d<=x2; x2d2<=x2d; x2d3<=x2d2;
    s2d2<=s2d; s2d3<=s2d2; s2d4<=s2d3; s2dcd<= s2dc;
    s2d<=x2 - s2dcd; y2d<=s2dcd + s2d4;
  end if;
 end process;

  s3da <=    SHIFT_RIGHT(s3d,3) - SHIFT_RIGHT(s3d,6);

 process(CLK) begin
  if rising_edge(CLK) then
    x3<= RESIZE(y2d,x3'length);
    s3d2<=s3d; s3d3<=s3d2; s3d4<=s3d3; s3dad<= s3da;
    s3d<=x3 - s3dad;              y3dt<=RESIZE(s3dad,y3dt'length) + s3d4;
    y3d<= y1d + y3dt;
  end if;
 end process;

 DO<=std_logic_vector(y3d(27 downto 4));
end synt;
```

ДОДАТОК Е. Лістинг опису мовою VHDL програми синтезу коефіцієнтів фільтрів без помножувачів

```
-------------------------------------------------------------------------------------------------
--
-- Title       : Filter_Estim
-- Design      : Wave_filter
-- Author      : Anastasia Serhienko
-- Company     : KPI
--
-------------------------------------------------------------------------------------------------
--
-- File        : Filter_Estim.vhd
-- Generated   : Sat Jan 22 11:15:17 2019
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.20
--
-------------------------------------------------------------------------------------------------
--
-- Description :  searching for coefficients
--
-------------------------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.all,IEEE.MATH_REAL.all,IEEE.MATH_Complex.all;

entity Filter_Synt is
end Filter_Synt;


architecture Filter_Estim of Filter_Synt is
type     arrr is array(natural range<>) of real;
  -- tested frequencies
  constant freqs:arrr(0 to 5):=(0.01,0.016,     0.04,0.06,0.08,0.1);
  --tested magnitudes
  constant predel:arrr(0 to 5):=(1.0,1.0,      0.0,0.0,0.0,0.0);
  -- weight of magnitude
        constant waznost:arrr(0 to 5):=(4.0,1.0,    100.0,100.0,100.0,100.0);

        function Z(fi:real)  -- complex number Z  in power of angle -Pi +Pi
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
        begin
                return exp(COMPLEX_TO_POLAR(MATH_CBASE_J)*fi);
        end Z;

        function zweno1(a:real; --          (a+Z)/(1+aZ)
                fi:real)  -- angle -Pi +Pi
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
                variable tn,td:   COMPLEX_POLAR;
        begin
                tn:=COMPLEX_TO_POLAR(COMPLEX'(A,0.0))+Z(-fi);
                td:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))+a*Z(-fi);
                return tn/td;
        end zweno1;
```

```
        function zweno2(b,c:real; -- (b+cZ+Z^2)/(1+cZ+bZ^2)
                fi:real)  --
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
                variable tn,td:   COMPLEX_POLAR;
        begin
                tn:=COMPLEX_TO_POLAR(COMPLEX'(b,0.0))
                +c*Z(-fi) + Z(-2.0*fi);
                td:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))
                +c*Z(-fi) + b*Z(-2.0*fi);
                return tn/td ;
        end zweno2;

        function zweno2M(b,c:real; -- (b+c(b+1)Z+Z^2)/(1+c(b+1)Z+bZ^2)
                fi:real)  -- angle -Pi +Pi
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
                variable tn,td:   COMPLEX_POLAR;
        begin
                tn:=COMPLEX_TO_POLAR(COMPLEX'(b,0.0))
                +c*(b+1.0)*Z(-fi) + Z(-2.0*fi);
                td:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))
                +c*(b+1.0)*Z(-fi) + b*Z(-2.0*fi);
                return tn/td ;
        end zweno2m;

        function zweno1x2(a:real; --      (a+Z)/(1+aZ)
                fi:real)  --- angle -Pi +Pi
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
                variable tn,td:   COMPLEX_POLAR;
        begin
                tn:=COMPLEX_TO_POLAR(COMPLEX'(A,0.0))+Z(-2.0*fi);
                td:=
COMPLEX_TO_POLAR(MATH_CBASE_1)+COMPLEX_TO_POLAR(COMPLEX'(A,0.0))*Z(-2.0*fi);
                return tn/td;
        end zweno1x2;

        function zweno2x2(b,c:real; -- (b+c(b+1)Z^2+Z^4)/(1+c(b+1)Z^2+bZ^4)
                fi:real)  -- angle -Pi +Pi
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
                variable tn,td:   COMPLEX_POLAR;
        begin
                tn:=COMPLEX_TO_POLAR(COMPLEX'(b,0.0))
                +c*(b+1.0)*Z(-2.0*fi) + Z(-4.0*fi);
                td:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))
                +c*(b+1.0)*Z(-2.0*fi) + b*Z(-4.0*fi);
                return tn/td ;
        end zweno2x2;

        function zweno2G(b,c:real; --  (-b-c-1 +(c-b)Z+Z^2)/(1+(c-b)Z+(-b-c-1)Z^2)
                fi:real)  -- angle -Pi +Pi ; b=g1, c=g2
                return COMPLEX_POLAR is -- ( POSITIVE_REAL,PRINCIPAL_VALUE) - Magnitude-phase
                variable tn,td:   COMPLEX_POLAR;
        begin
                tn:=COMPLEX_TO_POLAR(COMPLEX'(-b-c-1.0,0.0))
                +(c-b)*Z(-fi) + Z(-2.0*fi);
                td:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))
```

```
        +(c-b)*Z(-fi) +(-b-c-1.0)*Z(-2.0*fi);
        return tn/td ;
end zweno2G;

function MAXZ(x1,x2,x3,x4,x5,x6:real)return real is
        variable t:real:=x1;
begin
        if x2>t then t:=x2;end if;
        if x3>t then t:=x3;end if;
        if x4>t then t:=x4;end if;
        if x5>t then t:=x5;end if;
        if x6>t then t:=x6;end if;
        return t;
end MAXZ;

function DB(mi:COMPLEX_POLAR)return real is begin
        return 20.0*LOG10(ABS(mi));
end DB;
function MAG(mi:COMPLEX_POLAR)return real is begin
        return (ABS(mi));
end MAG;


type Tarr is array (0 to 516) of integer;
constant ints3:Tarr:=(0,1,2,3,4,5,6,7,8,
9, 10, 11,12, 13,14,15,16,
17,18, 19,20, 21,22,23,24,
25,26, 27,28, 29,30,31,32,
33,34, 35,36, 37,38,39,40,
41,42, 44,46, 47,48,49,50,
52,54, 55,56, 57,58,59,60,
61,62, 63,64, 65,66,67,68,
69,70, 71,72, 73,74,76,78,
79,80, 81,82, 84,88,92,94,
95,96, 97,98, 100,104,108,
110,111,112,113,114,116,118,119,
120,121,122,124,125,126,127,128,
129,130,131,132,133,134,135,136,
137,138,140,143,144,145,146,148,
152,156,158,159,160,161,162,164,
168,176,184,188,190,191,192,193,
194,196,200,208,216,220,222,223,
224,225,226,228,232,236,238,239,
240,241,242,244,246,247,248,249,
250,251,252,253,254,255,256,257,
258,259,260,261,262,263,264,265,
266,268,270,271,272,273,274,276,
280,284,286,287,288,289,290,292,
296,304,312,316,318,319,320,321,
322,324,328,336,352,368,376,380,
382,383,384,385,386,388,392,400,
416,432,440,444,446,447,448,449,
450,452,456,464,472,476,478,479,
480,481,482,484,488,492,494,495,
496,497,498,500,502,503,504,505,
506,507,508,509,510,511,512,513,
```

```
514,515,516,517,518,519,520,521,
522,524,526,527,528,529,530,532,
536,540,542,543,544,545,546,548,
552,560,568,572,574,575,576,577,
578,580,584,592,608,624,632,636,
638,639,640,641,642,644,648,656,
672,704,736,752,760,764,766,767,
768,769,770,772,776,784,800,832,
864,880,888,892,894,895,896,897,
898,900,904,912,928,944,952,956,
958,959,960,961,962,964,968,976,
984,988,990,991,992,993,994,996,
1000,1004,1006,1007,1008,1009,1010,1012,
1014,1015,1016,1017,1018,1019,1020,1021,
1022,1023,1025,1026,1027,1028,1029,1030,
1031,1032,1033,1034,1036,1038,1039,1040,
1041,1042,1044,1048,1052,1054,1055,1056,
1057,1058,1060,1064,1072,1080,1084,1086,
1087,1088,1089,1090,1092,1096,1104,1120,
1136,1144,1148,1150,1151,1152,1153,1154,
1156,1160,1168,1184,1216,1248,1264,1272,
1276,1278,1279,1280,1281,1282,1284,1288,
1296,1312,1344,1408,1472,1504,1520,1528,
1532,1534,1535,1536,1537,1538,1540,1544,  --447
1552,1568,1600,1664,1728,1760,1776,1784,
1788,1790,1791,1792,1793,1794,1796,1800,  --463
1808,1824,1856,1888,1904,1912,1916,1918,  --471
1919,1920,1921,1922,1924,1928,1936,1952,  --479
1968,1976,1980,1982,1983,1984,1985,1986,  --487
1988,1992,2000,2008,2012,2014,2015,2016,
2017,2018,2020,2024,2028,2030,2031,2032,
2033,2034,2036,2038,2039,2040,2041,2042, --511
2043,2044,2045,2046,2047);-- 516,
--        others=>0);

type Tarr2 is array (0 to 111) of integer;
constant ints2:Tarr2:=(0,1,2,3,4,5,6,7,8,
9, 10,12,14,15,16,17,18,
20,24,28,30,31,32,33,34,
36,40,48,56,60,62,63,64,
65,66,68,72,80,96,112,120,
124,126,127,128,129,130,132,136,
144,160,192,224,240,248,252,254,
255,256,257,258,260,264,272,288,
320,384,448,480,496,504,508,510,
511,512,513,514,516,520,528,544,
576,640,768,896,960,992,1008,1016,
1020,1022,1023,1024,1025,1026,1028,1032,-- 96
1040,1056,1088,1152,1280,1536,1792,1920, --104
1984,2016,2032,2040,2044,2046,2047);  --111

function CO(i:integer) return real is begin -- real coefficient
        return   real(ints3(i))/2048.0;
end CO;
function CO2(i:integer) return real is begin-- real coefficient
        return   real(ints2(i))/2048.0;
```

```
end CO2;

signal doc: COMPLEX_POLAR;
signal do:real;
constant sl:real:=0.03125;
signal   ph:real:=0.0;
signal fi:real;

signal CLK,rst : STD_LOGIC:='0';
--signal C1:real:= -0.25 - 0.015625;
signal zatuch,sigma,zmin,magn: real;
signal c0,c1,c2,c3,b1,b2,b3:real:=0.5;
signal c0o,c1o,c2o,c3o,b1o,b2o,b3o:real;
signal i0,i1,i2,i3,i4,i5,i6:natural:=0;
signal io0,io1,io2,io3,io4,io5,io6:natural:=0;
signal z0,z1,z2,z3,z4,z5,z6:real:=0.5;
-- low border of coefficients
constant l0:natural:=460;
constant l1:natural:= 104;--478;
constant l2:natural:= 103;--466;
constant l3:natural:= 105; --488;
constant l4:natural:=486;--467
constant l5:natural:=452;
constant l6:natural:=490;
-- high border of coefficients
constant h0:natural:=467; --478;
constant h1:natural:= 106;--489; --490;
constant h2:natural:= 105; --482;
constant h3:natural:= 107;--495;
constant h4:natural:=489;
constant h5:natural:=473;
constant h6:natural:=505;

type Tarrz is array (0 to 5) of real;


begin
        clk<= not clk after 5 ns;
        rst<='1', '0' after 33 ns;
        process(CLK)  -- traversing the coefficient space
                variable p, phase:real:=0.0;
                variable err,ei,z1,z2,z3:real;
                variable mi,mm: COMPLEX_POLAR;
                variable Zi:tarrz:=(others=>0.0);
        begin
                if clk='1' and clk'event then
                        phase:= phase+0.002;
                        ph<=phase;
                        p:=phase*MATH_PI*2.0;

                        if rst='1' then
                                i0<=l0;
                                i1<=l1;
                                i2<=l2;
                                i3<=l3;
                                i4<=l4;--470; --
```

```
        i5<=l5;
        i6<=l6;
        zmin<=1.0;
        zatuch<=1.0;
else
        if i0<h0 then
                i0<=(i0 + 1);
        else
                i0<=l0;
                if i1<h1 then
                        i1<=(i1 + 1);
                else
                        i1<=l1;
                        if i2<h2 then
                                i2<=(i2 + 1);
                        else
                                i2<=l2;
                                if i3<h3 then
                                        i3<=(i3 + 1);
                                else
                                        i3<=l3;
                                        if i4<h4 then
                                                i4<=(i4 + 1);
                                        else
                                                i4<=l4;
                                                if i5<h5 then
                                                        i5<=(i5 + 1);
                                                else
                                                        i5<=l5;
                                                        if i6<h6 then
                                                                i6<=(i6 + 1);
                                                        else
                                                                i6<=l6;
                                                        end if;
                                                end if;
                                        end if;
                                end if;
                        end if;
                end if;
        end if;
        c0<= - CO(i0);
        c1<= -2.0* CO2(i1);
        c2<=-2.0* CO2(i2);
        c3<=-2.0*  CO2(i3);
        b1<=CO(i4);
        b2<=CO(i5);
        b3<=CO(i6);

end if;

for i in 0 to 5 loop
        p:=freqs(i)*MATH_2_PI;

        mi:=(zweno1(c0,p)*zweno2(b1,c1,p)
        + zweno2(b2,c2,p)*zweno2(b3,c3,p))/2.0; --      synthesized filter
```

```
                    if (i=0) or (i=1) then
                        zi(i):=0.02*ABS(mag(mi) - 1.0);
                    else
                        zi(i):= mag(mi);
                    end if;

                end loop;
                magn<=DB(mi);

                z0<=zi(0);
                z1<=zi(1);
                z2<=zi(2);
                z3<=zi(3);
                z4<=zi(4);
                z5<=zi(5);

                zatuch<= MAXZ(zi(0),zi(1),zi(2),zi(3),zi(4),zi(5));
                if zmin > zatuch*1.1 then
                    zmin<=zatuch;
                    io0<=i0;
                    io1<=i1;
                    io2<=i2;
                    io3<=i3;
                    io4<=i4;
                    io5<=i5;
                    io6<=i6;
                    c0o<=c0;
                    c1o<=c1;
                    c2o<=c2;
                    c3o<=c3;
                    b1o<=b1;
                    b2o<=b2;
                    b3o<=b3;
                end if;

            end if;

        end process;


end Filter_Estim;
```

ДОДАТОК Ж. Публікації за темою дисертації

1. Сергієнко А. М., Романкевич В. О., Сергієнко А. А. Генетичне програмування спеціалізованих конвеєрних пристроїв. *Електронне моделювання*. 2020. Т. 42, №2. С. 25–40.
2. Serhienko A., Sergiyenko A. Modules for pipelined mixed radix FFT processors. *Int. J. of Reconfigurable Computing*. Hindawi Publishing Corp., 2016. V. 2016. P. 1–7.
3. Klymenko I., Tkachenko V., Serhienko A., Kulakov Y. Formalization of the concept of adaptive tasks mapping in the reconfigurable computers on FPGA. *Eastern European Journal of Enterprise Technologies*. 2018. V. 2, No. 9–92. P. 20–28.
4. Sergiyenko A., Serhienko, A. Complexity Reduced IIR Filter Design for FPGA. *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*. Kyiv, 2020. P. 1–4, doi: 10.1109/SAIC51296.2020.9239119.
5. Serhienko A., Sergiyenko A. Romankevich V. Genetic Programming of Pipelined Datapaths for FPGA. *IEEE 40-th Int. Conf. on Electronics and Nanotechnology, (ELNANO)*. Kyiv. 2020. P. 802–806.
6. Serhienko A., Sergiyenko A. VHDL Generation of Optimized IIR Filters. *IEEE 2-nd Ukraine Conference on Electrical and Computer Engineering, (UKRCON).* Lviv, Ukraine, July, 2019. P. 1171–1174.
7. Serhienko A., Sergiyenko A., Simonenko A. A method for synchronous dataflow retiming. *IEEE 1-st Ukraine Conf. on Electrical and Computer Engineering (UKRCON)*. Kyiv, 2017. P. 1015–1018.
8. Сергієнко А. М., Романкевич В. О., Сергієнко А. А. Генетичне програмування опису конвеєра даних мовою VHDL. *Прикладна математика та комп'ютинг* : тези десятої наук. конф. магістрантів та аспірантів ПМК’2018 (Київ, 21–23 бер. 2018). Київ, 2018. С. 153–157.

9. Сергієнко А. М., Хусейн К. С., Сергієнко А. А. Фільтри зі скінченною характеристикою з мінімізованими апаратними витратами. *Безпека, Відмовостійкість, Інтелект* : тези наук. конф. Київ, НТУУ «КПІ», 2018. С. 99–103.
10. Serhienko A., Sergiyenko A. Digital Filter Design using VHDL. *High Performance Computing* (HPC-UA 2018) : Proc. 5-th Int. Conf. Kyiv, 2018. P. 123–126.
11. Serhienko A., Sergiyenko A. Method of the Digital Filter Design using VHDL. *Winter InfoCom Advanced Solutions 2016* : Proc. Int. Conf. Kyiv, 2016. P. 68–69.
12. Сергієнко А. А. Реалізація конвеєрних процесорів швидкого перетворення Фур'є у ПЛІС. *Прикладна математика та комп'ютинг — 2016* : тези наукової конференції. Київ, 2016. С. 128–131.
13. Сергієнко А. А., Сергієнко А. М. Бібліотека модулів для швидкого перетворення Фур'є. *Інформатика та обчислювальна техніка – IOT-2016* : тези наук. конф. студентів, магістрантів та аспірантів. Київ, 2016. С. 114–117.
14. Сергієнко А. А. Сергієнко А. М. Набір модулів для швидкого перетворення Фур'є. *Infocom Advanced Solutions 2015* : тези міжнар. наук.-практ. конф. Київ, 2015. С. 52–53.
15. Сергієнко А. А., Клятченко Я. М. Процесор швидкого перетворення Фур'є за простою основою. *Сучасні методи, інформаційне та програмне забезпечення систем управління організаційно-технологічними комплексами* : тези всеукр. наук.-практ. інтернет-конф. Луцьк, 2015. С. 21–23.
16. Serhienko A., Sergiyenko A. Computing Pythagorean triples in FPGA. *High Performance Computing*, (HPC-UA'2013) : Proc. 3-d Int. Conf. Kyiv, 2013. P. 347–349.

17. Сергиенко А. М., Сергиенко А. А. Моделирование волновых процессов с помощью волновых фильтров. *Моделювання-2018* : тези міжн. наук. конф. Київ, 2018, С. 224–227.

ДОДАТОК К. Акти про впровадження результатів дисертаційного дослідження

ЗАТВЕРДЖУЮ
Проректор з навчальної роботи
Національного
технічного університету України
«Київський політехнічний інститут
імені Ігоря Сікорського»

_____ Анатолій МЕЛЬНИЧЕНКО
«____» _____ 2023 р.

АКТ

впровадження результатів дисертаційного дослідження асистентки кафедри Обчислювальної техніки факультету інформатики і обчислювальної техніки Національного технічного університету України «КПІ ім. Ігоря Сікорського» Молчанової Анастасії Анатоліївни на тему «Методи і засоби проєктування спеціалізованих конвеєрних обчислювачів на базі ПЛІС для обробки сигналів» на здобуття ступеня доктора філософії..

Комісія у складі: голова – завідувач кафедри ОТ КПІ ім. Ігоря Сікорського, д.т.н., проф. Стіренко С.Г.; члени комісії – професор кафедри ОТ КПІ ім. Ігоря Сікорського, д.т.н., доц., Клименко І. А., професор кафедри ОТ КПІ ім. Ігоря Сікорського, д.т.н., с.н.с. Сергієнко А. М. цим Актом засвідчує, що результати дисертаційного дослідження Молчанової Анастасії використані співробітниками кафедри ОТ КПІ ім. Ігоря Сікорського при підготовці та викладанні курсів лекцій «САПР комп'ютерних систем» та «Системи цифрової обробки сигналів і зображень». Зокрема впроваджено огляд алгоритмів оптимізації синтезу комп'ютерних систем, що включає новий метод проєктування спеціалізованих конвеєрних структур на основі генетичного програмування, розроблена лабораторна робота по вивченню проєктування цифрових фільтрів на ПЛІС з застосуванням нового способу множення на коефіцієнти.

Голова комісії
д.т.н., проф. Сергій СТІРЕНКО

Члени комісії
д.т.н., доц. Ірина КЛИМЕНКО

д.т.н., с.н.с. Анатолій СЕРГІЄНКО

ЗАТВЕРДЖУЮ
Проректор з навчальної роботи
Національного
технічного університету України
«Київський політехнічний інститут
імені Ігоря Сікорського»

_____ Анатолій МЕЛЬНИЧЕНКО
«____» _____ 2023 р.

АКТ

про використання результатів дисертаційного дослідження асистентки кафедри Обчислювальної техніки факультету інформатики і обчислювальної техніки Національного технічного університету України «КПІ ім. Ігоря Сікорського» Молчанової Анастасії Анатоліївни на тему «Методи і засоби проєктування спеціалізованих конвеєрних обчислювачів на базі ПЛІС для обробки сигналів» на здобуття ступеня доктора філософії.

Комісія у складі: голова – завідувач кафедри ОТ КПІ ім. Ігоря Сікорського, д.т.н., проф. Стіренко С.Г.; члени комісії – професор кафедри ОТ КПІ ім. Ігоря Сікорського, д. т. н., доц., Клименко І. А., професор кафедри ОТ КПІ ім. Ігоря Сікорського, д. т. н., с. н. с. Сергієнко А. М. цим Актом засвідчує, що результати дисертаційної роботи Молчанової Анастасії Анатоліївни отримані нею особисто та використані у:

– НДР №2863-п «Створення засобів проектування та розробка на їх основі високопродуктивних процесорів систем технічного зору», номер державної реєстрації 0115U002326;

– НДР ЗОТ/2017 «Методи і засоби відображення потокових алгоритмів у конфігуровані комп'ютери», що виконується за ініціативою авторів, № держреєстрації 0117U005087, закінчення у 2023 р.,

– Веб-застосунку «Генератор рекурсивних фільтрів без блоків множення», що розміщений на сайті, що належить кафедрі ОТ, URL: https://kanyevsky.kpi.ua/GEN_MODUL/APgen/FiltergenAP.php.

В темах та Веб-застосунку використані наступні результати дисертаційної роботи Молчанової А.А.:

– метод проєктування спеціалізованих конвеєрних структур на основі генетичного програмування, який забезпечує проєктування конвеєрних ОС для розв'язання задач ЦОС, у яких досягається високе відношення продуктивність — вартість, зокрема, зпроєктований за методом процесор дискретного

косинусного перетворення має на третину менше блоків множення, на 11% менші апаратні витрати логічних схем і на 19% більшу тактову частоту;

– спосіб множення на коефіцієнти, а також множина структур цифрових фільтрів, які мають оптимізовані апаратні витрати і швидкодію, в яких застосовано цей спосіб і які згенеровані за новим методом.

Голова комісії
д.т.н., проф. Сергій СТІРЕНКО

Члени комісії
д.т.н., доц. Ірина КЛИМЕНКО

д.т.н., с.н.с. Анатолій СЕРГІЄНКО