DOI: https://doi.org/10.36910/6775-2524-0560-2023-53-16
УДК 621.316.06
Громко Олександр Олександрович, бакалавр,
https://orcid.org/0009-0000-4563-8604
Боярінова Юлія Євгеніївна, с.н.с.-к.т.н., доцент,
https://orcid.org/0000-0002-8974-529X
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ, Україна

МЕТОДИ ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ БАГАТОРІВНЕВОЇ ОПТИМІЗАЦІЇ ТОПОЛОГІЙ ЕЛЕКТРИЧНИХ СХЕМ

Громко О.О., Боярінова Ю.Є. Методи застосування штучного інтелекту для багаторівневої оптимізації топології електричних схем. У даній статті проаналізовані методи та системи багаторівневої локальної оптимізації топології електричних схем, а також розглянуто метод оптимізації топології за допомогою генетичного алгоритму, що базується на основних засадах методології оптимізації та методології використання генетичного алгоритму для задач автоматизації.
Ключові слова: генетичні алгоритми, багаторівнева оптимізація, локальна оптимізація, топології електросхем, автоматизація оптимізації

Hromko O., Boyarinova Y. Methods of applying artificial intelligence for multi-level optimisation of electrical circuit topologies. This article analyses the methods and systems of multilevel local optimisation of electrical circuit topologies, and considers the method of topology optimisation using a genetic algorithm based on the basic principles of optimisation methodology and the methodology of using a genetic algorithm for automation tasks.
Keywords: genetic algorithms, multilevel optimisation, local optimisation, topological electrical circuits, optimisation automation

Постановка проблеми.

З роками потреба в ефективних інструментах оптимізації схем не змінилася, але причини для оптимізації цифрових схем змінилися. Оптимізація цифрових схем була активною сферою досліджень з початку 1950-х років. У ті часи цифрові системи, такі як революційний ENIAC (Електронний числовий інтегратор і калькулятор), важили кілька тон і містили тисячі резисторів, вакуумних ламп та інших дискретних компонентів (1). Через непомірну вартість логічних вентилів в ті часи, ранні зусилля з оптимізації були зосереджені в першу чергу на зменшенні кількості дискретних компонентів в заданій логічній схемі.

З появою транзисторів, а згодом і інтегральних схем, вартість логічних вентилів наприкінці 1960-х і на початку 1970-х років різко знизилася. Через це бракувало зацікавленості у витрачанні багато часу на складні методи оптимізації, коли ручне спрощення, здавалося, працювало адекватно для більшості конструкцій. Лише в середині та наприкінці 1970-х років оптимізація схем отримала друге дихання, і з тих пір продовжує розвиватися. Цьому відродженню сприяла низка факторів. Найвизначнішим з них є впровадження технології LSI (Large Scale Integration), а пізніше VLSI (Very Large Scale Integration). Це передбачало розміщення все більшої кількості логічних вентилів на все меншій площі поверхні. Оскільки різноманітність сфер застосувань цифрових схем зростала в геометричній прогресії, зростала і їхня складність. Якщо раніше схеми складалися з п'яти-десяти змінних і декількох сотень вентилів, то сьогодні не рідкість зустріти 30 і більше змінних, кілька виходів і тисячі, а в деяких випадках і мільйони вентилів. Як ви можете собі уявити, здатність людини-дизайнера синтезувати такі складні схеми накладає серйозні обмеження на досяжні цілі. Люди не тільки схильні до помилок, але їхні розробки часто є досить неефективними

На сьогоднішній день основними цілями оптимізації схем є (2):

1. Мінімізація загальної площі конструкції
2. Мінімізація часу затримки критичного шляху
3. Покращення тестованості та верифікованості синтезованої логіки.

Зменшення загальної площі схеми є важливим з кількох причин. Очевидно, що менші цифрові схеми можна розмістити на менших площах (годинники, радіоприймачі, ноутбуки тощо). Ніде це не має більшого значення, ніж в сучасних аерокосмічних апаратах, де простір на літальному апараті є критично важливим активом. Ще однією причиною зменшення площі поверхні є те, що "вартість виготовлення схеми фактично є експоненціальною функцією від площі, принаймні, якщо схема

велика (3)". Зменшення часу затримки критичного шляху, очевидно, прискорить роботу схеми. Найшвидшою схемою буде та, в якій сигнали проходять через найменшу кількість вентилів. Оскільки кожна система булевих функцій може бути виражена в еквівалентній SOP-формі, це представлення перетворюється на схему з глибиною у два вентилі. Хоча це, як правило, найшвидша схема

найшвидшу з можливих схем, її реалізація часто вимагає великої площі поверхні і може призвести до неприйнятного вентилювання деяких вентилів. Таким чином, зазвичай потрібно знайти компроміс між зменшенням площі поверхні та зменшенням затримки поширення. Останнім часом у ряді систем з'явилися засоби для оптимізації схеми на основі такого компромісу.

З ускладненням схем все більшого значення набуває ідея розробки цифрових систем, які можна легко протестувати, щоб переконатися, що вони функціонують правильно. На щастя, створення оптимізованих схем, які піддаються тестуванню і перевірці, часто є побічним продуктом автоматизованих методів оптимізації (2). Майбутні системи проектування генеруватимуть повний набір тестових векторів, що є результатом процесу синтезу та оптимізації схеми. Інша мета систем оптимізації - перепроектувати задану схему, переводячи її з однієї технології на іншу, використовуючи при цьому всі переваги цільової технології. Багато систем мінімізації можуть звести схему до системи, що складається зі стандартних вентилів І та АБО. Однак вони можуть бути реалізовані з використанням вентилів NAND і NOR або більш сучасної технології. Коли оптимізована схема І-АБО трансформується в нову технологію, отримана схема, як правило, не є оптимальною і може піддаватися подальшому скороченню. Якщо цільова технологія відома з самого початку, можна спроектувати і оптимізувати цифрову схему з урахуванням цільової технології. Хорошим показником якості системи оптимізації є її здатність включати нові технології в міру їх розвитку і здійснювати перетворення між технологіями.

Щоб бути конкурентоспроможними в майбутньому, системи оптимізації схем повинні мати можливість швидко створювати перевірені схеми на основі поведінкових специфікацій. "Ця здатність буде ставатиме все більш важливою, оскільки ринок спеціалізованих інтегральних схем (ASIC) продовжує відповідати своїм прогнозам швидкого зростання (2)

Аналіз останніх досліджень і публікацій.

У той час як область дворівневої оптимізації стає все більш зрозумілою, багаторівнева оптимізація все ще перебуває в стані активного дослідження. Багаторівнева реалізація булевої функції (див. Рисунок 3.4) дозволяє необмежено використовувати проміжні сигнали. Можливість повторного використання проміжних сигналів може призвести до необмеженої кількості способів представлення багаторівневих схем. Хоча це значно розширює ступінь свободи, що надається проектувальнику, це також супроводжується значним збільшенням складності. Саме ця складність перешкоджає створенню ефективних автоматизованих систем багаторівневої оптимізації.

Принципи локальної багаторівневої оптимізації. У локальній оптимізації початкова схема синтезується з системи рівнянь, що задають схему. Система, що базується на правилах, виконує повторне сканування, застосовуючи локальні правила перетворення для модифікації схеми. Ці правила спрямовані на зменшення площі схеми або затримки поширення. 3-16 Важливо точно розуміти, коли застосовувати методи локальної оптимізації. Слово "оптимізація" в нашому контексті означає, що існує початкова схема, яку потрібно оптимізувати. Таким чином, система локальної оптимізації, як правило, виконує своє завдання у наступні три окремі кроки (4):

1. Мінімізація булевих рівнянь, що задають схему,
2. Синтез початкової схеми,
3. Оптимізація схеми для заданої технології.

Лише третій крок передбачає застосування локальних перетворень. На кроці 1 система булевих рівнянь, що описує схему, скорочується за допомогою математичних методів, використовуючи всі переваги будь-яких умов "don't-care". На кроці 2 застосовуються різноманітні методи для синтезу початкової схеми. Часто використовується обмежений набір типів затворів наприклад, вентилі NAND/NOR та мультиплексори. Деякі системи включають багаторівневі методи синтезу, такі як факторизація, щоб скористатися перевагами будь-яких загальних проміжних доданків. Нарешті, на кроці 3 застосовуються принципи локальної оптимізації. Ключем до успіху локальної оптимізації є використання ефективної системи, заснованої на правилах. Кожне правило передбачає заміну існуючої конфігурації одного або декількох елементів схеми на еквівалентну, але більш бажану конфігурацію, як показано на Рисунку 1 (4). Оскільки ця заміна стосується лише

кількох елементів схеми, вона вважається локальною оптимізацією на відміну від глобального підходу, який охоплює всю логіку схеми. Правила слідують тим, яких дотримується розробник при ручній оптимізації логічної схеми. Нові правила можуть бути включені до системи локальної оптимізації, просто додавши їх до бібліотеки.

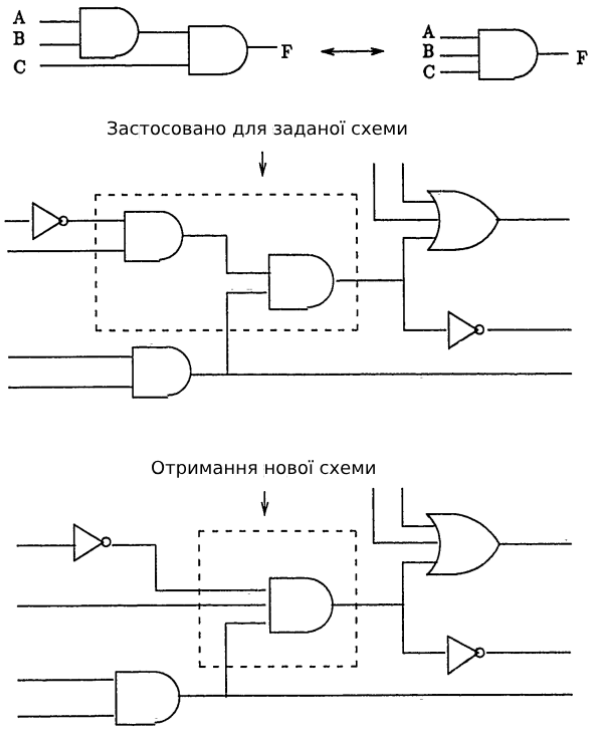

Рис.1 - Приклад правила локального перетворення

Під час процесу оптимізації правила застосовуються впорядковано. Порядок, в якому вони застосовуються, може кардинально вплинути на кінцевий результат. Порядок, в якому вони застосовуються, може мати значний вплив на кінцевий результат. Загальні правила, які зменшують як площу, так і швидкість, застосовуються в першу чергу. Вони мають пріоритет за їх відносною важливістю. Ці правила застосовуються багаторазово, доки не залишиться більше загальних правил.

На цьому етапі деякі більш просунуті системи, такі як SOCRATES, виконують часовий аналіз критичного шляху. Якщо час затримки критичного шляху не відповідає встановленим інструкцій або попередньо визначених меж, то до вентилів на критичному шляху будуть застосовані деякі правила економії часу. Коли бажаний час буде досягнуто, будуть застосовані правила економії простору до вентилів, що не знаходяться на критичному шляху. Кінцевим результатом цих систем локальної оптимізації часто є значно покращений конструкція. Мало того, що проекти стають швидшими і меншими, так ще й час, необхідний для їх синтезу за допомогою системи локальної оптимізації їх за допомогою системи локальної оптимізації значно скорочується. Деякі системи довели свою комерційну життєздатність і здатнимі постійно перевершувати дизайнерів-людей.

Системи локальної оптимізації. Історія розвитку систем локальної оптимізації є відносно недавньою. Цьому сприяв нещодавній інтерес до застосування систем, заснованих на правилах, до практичних проблем, які було важко вирішити, використовуючи інші підходи. IBM була однією з перших компаній, яка проявила інтерес, розробивши систему логічного синтезу (LSS) на початку 1980-х. Пізніше з'явилися LORES, DAS/Logic, SOCRATES та інші. LSS. LSS (5) була розроблена для перетворення високорівневої специфікації схеми в реалізацію виробничої якості за допомогою серії локальних перетворень. Першочерговою метою була система, яка виробляє реалістичні, не обов'язково оптимальні, схеми, які задовольняють низці обмежень. Вони повинні відповідати вимогам цільової технології, використовувати всі переваги цієї технології та створювати логіку з прийнятною кількістю вентилів і довжиною шляху. Система працює поетапно. Вона починається з перекладу специфікації системи з мови регістрового перенесення на мережу, що складається з І, АБО, НЕ, дешифраторів, суматорів тощо. Далі, через серію локальних перетворень на декількох рівнях, мережа замінюється більш примітивними реалізаціями, такими як NAND і NOR, і врешті-

решт, технологічно специфічними пристроями. Весь процес розвивається від початкових етапів, які відносно не залежать від цільової технології, до кінцевих етапів, які є специфічними для конкретної технології. Використання декількох етапів синтезу та оптимізації логіки дозволяє системі скористатися численними методами спрощення, які можна застосувати на кожному рівні.

Система LSS продемонструвала значний успіх, особливо при її використанні на проектах IBM. "Зокрема, в одному проекті вона була використана на 90 відсотках з більш ніж 100 дизайнів мікросхем. Це дозволило скоротити початковий час розробки вдвічі (5)". Незважаючи на успіх, LSS також мала свої недоліки. Він був обмежений у своїй здатності вирішувати проблеми синхронізації та кількості вентилів. Вона не використовувала повною мірою всю інформацію, яка була доступна в умовах "мені байдуже". Вона також часто перевершувала системи, які підходили до проблеми з більш глобальної точки зору. Незважаючи на це, вона забезпечила основу, з якої випливали майбутні системи.

LORES LORES (Система логічної реорганізації) - це система, яка автоматично розділяє і реструктурує логічну схему, що складається зі стандартних SSI і MSI, таким чином, щоб типи вентилів і кількість вхідних/вихідних клем були сумісні з вимогами LSI (6). Переваги БІС численні: менший розмір, краща продуктивність, знижене енергоспоживання, а часто і зниження загальної вартості. Ці переваги не даються задарма, і ціна, яку нам часто доводиться платити, - це збільшення часу на розробку, більша складність і складніший процес тестування та верифікації. LORES зосереджує свої зусилля на досягненні деяких переваг LSI шляхом подолання деяких труднощів.

LORES - це система автоматичної логічної оптимізації, основні функції якої описані нижче (6):

1. Витягнути одну з розділених секцій з початкового проекту.
2. Усунути будь-яку невикористану або надлишкову логіку.
3. Перетворити або реструктурувати логіку на ті логічні елементи, які дозволені в LSL.
4. Розділіть логічну схему так, щоб кількість вентилів і вхідних/вихідних клем були в межах заданих лімітів.

Одним з основних недоліків LORES є його нездатність працювати з асинхронними схемами. Таким чином, для схем, що містять асинхронний компонент, може знадобитися певний обсяг моделювання та ручної модифікації (6). З іншого боку, "LORES безпосередньо сприяв скороченню часу розробки і трудових ресурсів за рахунок автоматизації процесу перетворення схем і зменшення необхідності перевірки проекту (7)".

Нещодавно з'явилася нова версія LORES під назвою LORES/EX (8). Вона призначена для

трансформувати існуючу логічну схему з однієї технології на іншу. Вона не тільки гнучко реагує на зміни в технології, але й має ряд інших особливостей. Вона вводить правила стандартизації, які спрямовані на зменшення та спрощення розміру бази правил. Вона має функції, які використовують вирішення конфліктів для покращення загальної якості схем. Він також має можливість розділяти великі схеми на менші, що дозволяє LORES обробляти схеми з кількістю вентилів до 10 000 за практичну кількість часу.

DAS/Logic. DAS/Logic (Design Assistant Series) - це інструмент, що розробляється в Carnegie Group Inc. для допомоги в проектуванні IC. Він описаний Бірмінгемом (8) як заснована на правилах система, написана на мові OPS5, яка перетворює поведінковий опис системи в принципову електричну схему. На вхід подається високорівневий опис бажаної поведінки схеми, а на виході - набір стандартних комірок і список з'єднань.

На кожному рівні проектування виконуються різні кроки. Ці кроки включають (8):

1. Створення структури для поточного рівня проектування.
2. Пошук можливостей оптимізації або порушень обмежень.
3. Виправлення порушень або застосування оптимізуючих перетворень.
4. Генерування обмежень/можливостей для структури наступного рівня.

DAS/логічне проектування слідує підходу "зверху вниз" у поєднанні з деякими стратегіями "знизу вгору". стратегіями. Низхідний підхід дозволяє системі працювати з більш глобальної точки зору; однак, знаючи особливості технології реалізації, бажано також використовувати висхідні стратегії проектування. Наприклад, не варто створювати систему з чотирма входами АБО-транзисторів, якщо вони не існують у цільовій технології. Розробники DAS/Logic зрозуміли, що ідеальна система логічної оптимізації повинна включати комбінацію стратегій "зверху вниз" і "знизу вгору".

На момент останнього звіту (8) система все ще перебувала на стадії розробки, але демонструвала значний прогрес. Вся робота над системою була спрямована на використання "розумних" методів ШІ для вирішення складних завдань. "Завдяки використанню знань про предметну область, отриманих від дизайнерів, DAS/Logic здатна розумно застосовувати свої оптимізаційні та конструктивні знання, тим самим різко скорочуючи обсяг пошуку, необхідний для досягнення високої якості дизайну (8)".

SOCRATES. SOCRATES (Synthesis and Optimization of Combinatorics using a Rule-based And Technology-independent Expert System) наразі є однією з найбільш успішних систем логічної оптимізації. Вона була розроблена з метою об'єднання найпривабливіші риси сучасних систем оптимізації. Отримана система поєднує використання методів дворівневої та багаторівневої оптимізації з локальними алгоритмами оптимізації на основі правил. Отриману систему можна узагальнено представити як таку, що виконує наступні кроки (9):

1. Рівняння або мережеві списки, що використовуються для визначення бажаної поведінки системи

трансформуються у формат, сумісний з мінімізатором ЗЛП EXPRESSO.

2. EXPRESSO зводить кожну функцію до її мінімальної СОП-форми.

3. Слабкий поділ2 використовується для перетворення дворівневої реалізації в скорочену, але еквівалентну багаторівневу схему.

4. Експертна система на основі правил використовується для застосування локальних перетворень до схеми, щоб оптимізувати її для конкретної технології.

SOCRATES також має можливість вводити існуючі схеми у визначеному форматі для подальшої оптимізації або перетворення їх на нову технологію. Це, мабуть, одна з найбільш перевірених систем у своїй здатності працювати з обмеженнями за площею та часом. Користувач може вказати оптимальний компроміс між збільшенням швидкості та зменшенням розміру. Після цього схема може бути оптимізована для досягнення бажаних цілей. База правил побудована таким чином, що користувач може легко вводити додаткові правила, які автоматично перевіряються і класифікуються в базі знань (4).

SOCRATES вживає кілька заходів для забезпечення ефективної роботи. Вона використовує метарубрики для того, щоб простір пошуку був якомога меншим. Це, в поєднанні з тим фактом, що вона написана мовою С, а не типовою мовою експертних систем, забезпечує порівняно короткий час виконання. Завдяки цим вдосконаленням "комбінаційні схеми, на синтез і оптимізацію яких пішло б кілька днів, тепер можна згенерувати за лічені хвилини" (4). У підсумку, для великих схем (від 75 до 100 вентилів) SOCRATES у більшості випадків досягає зменшення площі від 25 до 60 відсотків; ці результати надзвичайно добре порівнюються з тими, що досягаються експертами вручну.

Запропонований метод оптимізації.

Генетичний алгоритм(ГА) - це метод стохастичної комбінаторної оптимізації. Первинні рамки генетичних алгоритмів були вперше представлені доктором Д. Голландом у 1975 році. На початку генетичний алгоритм необхідно розробити уявлення про можливі рішення, які часто називають хромосомами або особинами . Хромосома складається з набору генів, які можуть бути представлені двійковими кодами. Різні комбінації генів утворюють різні хромосоми. Кожна хромосома є можливим розв'язком задачі. Набір хромосом називається популяцією покоління. Хромосоми в поколінні змушені еволюціонувати до кращих у наступному поколінні за допомогою трьох основних операторів генетичного алгоритму: розмноження, кросинговеру та мутації, а також визначеної задачею функції пристосованості . При розмноженні частина відібраних точних копій хромосом поточної популяції стає частиною нащадків . При кросинговері випадково вибрані ділянки двох індивідуальних хромосом міняються місцями для отримання нащадків (11). При мутації випадково вибрані гени в хромосомах змінюються з імовірністю, що дорівнює заданій частоті мутацій (10). Для гена, що кодує двійковий код, це означає, що цифра 1 стає цифрою 0 і навпаки. Функція пристосованості, яку також називають цільовою функцією, є оціночною функцією, яка відіграє роль середовища для розрізнення хороших і поганих хромосом [11]. Лише певна кількість хромосом, перевірених і відібраних фітнес-функцією, може вижити і передати свої гени наступному поколінню. Еволюція зупиниться, коли певний критерій буде виконано.

Алгоритм починається зі створення випадкової початкової популяції. Після цього алгоритм створює послідовність нових популяцій. Таким чином, першим кроком для створення нової

популяції для наступного покоління є оцінка кожного члена поточної популяції шляхом обчислення його фітнес-значення. Після обчислення фітнес-значення алгоритм зупиняється, коли він знайде розв'язок або один з критеріїв зупинки буде виконано, і алгоритм поверне найкращий розв'язок у поточній популяції. На противагу цьому, поточна популяція буде змінена поточними нащадками, щоб сформувати наступне покоління, якщо розв'язок не буде задовільним. Таким чином, нова популяція створюється за допомогою генетичних операцій, таких як селекція, мутація і кросинговер. Для оператора селекції з популяції вибираються два батьки з кращими параметрами відповідно до їх пристосованості. Діти виробляються від батьків або шляхом внесення випадкових змін до одного з батьків, що називається мутацією, в той час як процес об'єднання елементів вектора пари батьків називається операцією кросинговеру. Після цього створюється нове "потомство" або особина відповідно до результату генетичного оператора. Нарешті, нова згенерована популяція використовується, де особина передається назад до фітнес-функції для подальшого прогону алгоритмів.

Висновки

Запропонований метод проектування електричних схем з використанням генетичного алгоритм у дозволяє автоматично визначати необхідні параметри елементів схеми для забезпечення бажаних характеристик схеми для різних типів навантаження та різних топологій схем. Таким чином, цей інструмент оптимізації для проектування електричних схем може бути застосований для вирішення традиційного методу пошуку параметрів електричного кола. Для подальшого розвитку, продуктивність схеми може бути покращена шляхом введення додаткових обмежень у функції пристосованості. Крім того, за допомогою генетичних алгоритмів можна розробляти інші аналогові схеми, такі як операційні підсилювачі на BJT і CMOS, операційний транзистор на CMOS і узгоджувальну мережу.

Хоча локальні системи оптимізації є досить успішними і застосовуються для вирішення дедалі складніших завдань, самі по собі вони не є вирішенням усіх проблем оптимізації. Вони виявилися надзвичайно ефективними, коли мова йде про перетворення заданої схеми в нову технологію, але все ще дуже слабкі, коли мова йде про повне використання всієї інформації, що міститься в специфікації, в тому числі умов "неважливо". SOCRATES зробила кроки для використання такої інформації, включивши в свою систему прості алгоритми глобальної оптимізації. Багато локальних систем оптимізації обмежені ефективністю своїх систем, заснованих на правилах. Хоча швидкість цих систем покращується, ефективність все ще залишається головною перешкодою. Майбутнє систем локального редизайну дуже світле, і в майбутньому ймовірно, з'явиться більше гібридних систем на основі штучного інтелекту

Список бібліографічного опису

1. Hayes, John P. *Computer Architecture and Organization*. [Електронний ресурс] / Hayes, John P. -1988. – Режим доступу до ресурсу: https://archive.org/details/computerarchitec0003edhaye/page/n5/mode/2up
2. Brayton, R.K., G.D. Hachtel and A.L. Sangiovanni-Vincentelli. *Multilevel Logic Synthesis*, [Електронний ресурс] / Brayton, R.K., G.D. Hachtel, A.L. Sangiovanni-Vincentelli - 1990 – Режим доступу до ресурсу: https://people.eecs.berkeley.edu/~alanmi/publications/other/multi_level.pdf
3. Ullman, Jeffrey D. *Computational Aspects of VLSI,*. [Електронний ресурс] / Ullman, Jeffrey D. 1984. – Режим доступу до ресурсу: https://dl.acm.org/doi/10.5555/1095588
4. de Geus, Aart J. and William W. Cohen. *A Rule-Based System for Optimizing Combinational Logic,*, [Електронний ресурс] / de Geus, Aart J. and William W. Cohen – 1985. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/4069623
5. Darringer, John A. *LSS: A System for Production Logic Synthesis* [Електронний ресурс] / Darringer, John A – 1984. – Режим доступу до ресурсу:https://ieeexplore.ieee.org/document/5390295
6. Nakamura, Shunichiro, Shinichi Mural and Chiyoji Tanaka. "LORES: Logic Reorganization System [Електронний ресурс] / Nakamura, Shunichiro, Shinichi Mural and Chiyoji Tanaka. – 1978. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/1585181
7. Enomoto, Kiyoshi. *LORES-2: A Logic Reorginization System*, [Електронний ресурс] / Enomoto, Kiyoshi – 1985. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/4069658
8. Ishikawa, J. *A Rule Based Logic Reorganization System LORES/EX,*, [Електронний ресурс] / Ishikawa, J – 1988. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/25703
9. Gregory, David. *SOCRATES: A System For Automatically Synthesizing and Optimizing Combinational Logic*, [Електронний ресурс] / Gregory, David - 1986. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/1586072

10. D.E Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, [Електронний ресурс] / D.E Goldberg – 1989. – Режим доступу до ресурсу: https://www.academia.edu/7459663/Genetic_algorithms_in_search_optimization_and_machine_learning
11. C. S. Moo, H. L. Cheng, and S. J. Guo, *Designing passive LC filters with contour maps*, [Електронний ресурс] / C. S. Moo, H. L. Cheng, and S. J. Guo - 1997.– Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/627503

References

1. Hayes, John P. *Computer Architecture and Organization*, 1988. – Available at: https://archive.org/details/computerarchitec0003edhaye/page/n5/mode/2up [Accessed 4 Dec. 2023].
2. Brayton, R.K., G.D. Hachtel and A.L. Sangiovanni-Vincentelli. *Multilevel Logic Synthesis*, 1990 – Available at: https://people.eecs.berkeley.edu/~alanmi/publications/other/multi_level.pdf [Accessed 4 Dec. 2023].
3. Ullman, Jeffrey D. *Computational Aspects of VLSI*, 1984. – Available at: https://dl.acm.org/doi/10.5555/1095588 [Accessed 4 Dec. 2023].
4. de Geus, Aart J. and William W. Cohen. *A Rule-Based System for Optimizing Combinational Logic*,, 1985 – Available at:
https://ieeexplore.ieee.org/document/4069623 [Accessed 4 Dec. 2023].
5. Darringer, John A. and others. *LSS: A System for Production Logic Synthesis*, 1984 – Available at: https://ieeexplore.ieee.org/document/5390295 [Accessed 4 Dec. 2023].
6. Nakamura, Shunichiro, Shinichi Mural and Chiyoji Tanaka. "LORES: Logic Reorganization System, 1978 – Available at:
https://ieeexplore.ieee.org/document/1585181 [Accessed 4 Dec. 2023].
7. Enomoto, Kiyoshi and others. *LORES-2: A Logic Reorginization System*, 1985 – Available at: https://ieeexplore.ieee.org/document/4069658 [Accessed 4 Dec. 2023].
8. Ishikawa, J. and others. *A Rule Based Logic Reorganization System LORES/EX*,, 1988 – Available at: https://ieeexplore.ieee.org/document/25703 [Accessed 4 Dec. 2023].
9. Gregory, David and others. *SOCRATES: A System For Automatically Synthesizing and Optimizing Combinational Logic*, 1986 – Available at: https://ieeexplore.ieee.org/document/1586072 [Accessed 4 Dec. 2023].
10. D.E Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989 – Available at: https://www.academia.edu/7459663/Genetic_algorithms_in_search_optimization_and_machine_learning [Accessed 4 Dec. 2023].
11. C. S. Moo, H. L. Cheng, and S. J. Guo, *Designing passive LC filters with contour maps*, 1997 – Available at: https://ieeexplore.ieee.org/document/627503 [Accessed 4 Dec. 2023].