МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Курсова робота

освітній ступінь – бакалавр

на тему: **«ОПТИМІЗАЦІЯ БУЛЕВИХ СХЕМ»**

Виконала
студентка 3-го року навчання
Освітньої програми
«Прикладна математика», 113
Черевко Крістіна
Керівник
Олійник Б.В.

Київ – 2023

Тема: Оптимізації булевих схем.

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	20.09.2022	
2.	Ознайомлення з темою курсової роботи.	30.09.2022	
3.	Розробка плану та структури роботи.	24.01.2023	
4.	Робота з науковою літературою.	11.02.2023	
5.	Побудова та програмування алгоритмів.	17.02.2023	
6.	Написання вступу.	29.03.2023	
7.	Написання теоретичної частини курсової роботи.	10.04.2023	
8.	Написання практичної частини курсової роботи.	15.04.2023	
9.	Попередній аналіз курсової роботи. Виправлення помилок та обробка зауважень.	06.05.2023	
10.	Створення презентації.	10.05.2023	
11.	Захист курсової роботи.	11.05.2023	

Зміст

1 Вступ

Цифрові інтегральні мікросхеми є ключовими компонентами усіх сучасних електронних пристроїв, починаючи від простих годинників і до складних комп'ютерних систем. Висока конкуренція в галузі електроніки змушує компанії постійно вдосконалювати технології виробництва та створює великий попит на все більш компактні, потужні та енергоефективні мікросхеми. Що, у свою чергу, перетворює задачу проектування ефективних цифрових мікросхем на ключове завдання у розробці сучасних електронних систем.

Розвиток технологій в області виробництва напівпровідників дозволив створювати мініатюрні транзистори, що, у свою чергу, дозволило розміщувати велику кількість транзисторів на компактних кремнієвих пластинах і, в результаті, проектувати набагато більш ефективні мікросхеми. У 1965 році співзасновник компанії Intel Гордон Мур сформулював статистичне припущення, так званий «Закон Мура», який полягає у тому, що кількість транзисторів, які можна розмістити на мікрочіпі, подвоюватиметься приблизно кожні два роки. Згодом Мур сформулював ще "Другий закон Мура", який говорить про те, що вартість фабрик з виробництва мікросхем експоненційно зростає з ускладненням мікрочіпів, що виробляються. Як бачимо подолання фізичних обмежень стає дедалі складнішим, а подальше зменшення розмірів компонентів, в свою чергу, призводить до значного збільшення вартості виробництва кремнієвих пластин [11].

Традиційно процес проектування мікрочіпів складається з декількох етапів: дизайн, верифікація дизайну, логічний синтез, фізичне проектування та розміщення, виробництво та тестування мікрочіпів. На етапі дизайну задається необхідна функціональність мікросхеми, визначаються основні компоненти майбутніх мікрочіпів та розробляється їх специфікація. Далі відбувається верифікація змодельованого мікрочіпу - перевірка на відповідність всім

вимогам та обмеженням. Наступний крок, логічний синтез, є одним з найважливіших етапів в проектуванні мікрочіпів. Це процес перетворення абстрактного опису бажаної поведінки цифрової схеми, у схему з логічних вентилів, яка задовольняє певним обмеженням щодо продуктивності, ціни, площі схеми та споживання електроенергії. Схему логічних вентилів легко перетворити у сукупність булевих функцій, які задають відповідні булеві схеми. І тому центральною задачею логічного синтезу є оптимізація булевих схем, а саме мінімізація їх площі або затримки. Наступний етап фізичного проектування передбачає ефективне розміщення транзисторів на кремнієвій пластині та їх з'єднання між собою. Останній етап - виробництво мікрочіпів та їх тестування, полягає у безпосередньому отриманні фізичного продукту та перевірці його на відповідність всім вимогам.

Ця курсова робота присвячена основній задачі логічного синтезу - оптимізації булевих схем, а саме мінімізації їх площі.

Метою цієї роботи є дослідження та розв'язок задачі оптимізації булевих схем.

В ході роботи були запрограмовано алгоритми оптимізації булевих схем на базі розкладу Шеннона з використанням перестановок змінних та без них, алгоритм із сумісним використанням розкладів Шеннона і Давіо, а також побудовано новий алгоритм, який одночасно використовує розклади Шеннона і Давіо для кожної перестановки змінних у булевій функції. В дослідженні також порівнюються результати роботи запропонованого нового алгоритму із іншими реалізованими алгоритмами та найкращими результатими представленими на всесвітньовідомій конференції International Workshop on Logic and Synthesis 2022 [15] (в якості вхідних даних для досліджуваних алгоритмів використовувались тестові приклади з вищезгаданої конференції).

Новизна цієї роботи полягає в розробці та дослідженні нового методу змен-

шення площі булевих схем, що мінімізує кількість їх двох-входових вершин.

Предметом дослідження в курсовій роботі виступають булеві схеми.

Об'єктом дослідження в цій роботі є оптимізація булевих схем.

Структурно курсова робота побудована наступним чином.

Другий розділ курсової роботи присвячений теоретичній частині: поняттям булевої функції, булевої мережі, орієнтованого ациклічного графа, And-Inverter та Xor-And-Inverter графів, структурного та функціонального хешування, розкладам Шеннона та Давіо.

В третьому розділі розглянуто приклад для демонстрації роботи алгоритмів оптимізації та наведене графічне представлення схем, отриманих в результаті роботи досліджуваних алгоритмів.

В четвертому розділі описано способи вводу/виводу булевих функцій, представлено псевдокоди чотирьох алгоритмів оптимізації булевих схем: алгоритм на основі розкладу Шеннона з фіксованим порядком змінних, алгоритм на основі розкладу Шеннона з перебором всіх перестановок змінних, алгоритм на основі розкладів Шеннона і Давіо з фіксованим порядком змінних та новий алгоритм, на основі розкладів Шеннона і Давіо з перебором всіх перестановок змінних. Також представлена реалізація структурного та функціонального хешування.

П'ятий розділ цієї курсової роботи присвячений експериментальним результатам, які були отримані при тестуванні алгоритмів на функціях зі змагань IWLS 2022 Programming Contest [15]. Було виконано порівняння результатів роботи запропонованого нового алгоритму та інших досліджуваних алгоритмів із найкращими результатими змагань, а також зроблено висновки за результатами даних тестів та порівнянь.

Шостий розділ завершує роботу і містить висновки.

2 Теоретична частина

Будь-яка електронна схема – це набір з'єднаних між собою окремих електронних компонент, таких як транзистори, резистори, конденсатори та діоди. Електронні схеми поділяються на два класи – аналогові та цифрові. В аналогових електронних схемах напруга і струм можуть змінюватися безперервно в часі. В цифрових схемах сигнал може приймати тільки декілька різних дискретних станів, які зазвичай кодують логічні або числові значення. У переважній більшості випадків у цифрових схемах використовується бінарна (двійкова) логіка, коли одному певному рівню напруги відповідає логічна одиниця, а іншому — нуль. Базовим елементом будь-якої цифрової схеми є логічний вентиль, зазвичай побудований на базі декількох транзисторів. *Логічний вентиль* перетворює один або декілька вхідних бінарних сигналів у вихідний бінарний сигнал, таким чином виконує деяку булеву операцію. Операції, які зазвичай виконують логічні вентилі, відомі нам ще з курсу дискретної математики. Це кон'юнкція (AND), диз'юнкція (OR), виключна диз'юнкція (XOR), заперечення (NOT) і т. д.

Для того, щоб далі в цій роботі ефективно працювати з алгоритмами оптимізації булевих схем, нагадаємо деякі базові поняття булевої алгебри.

2.1 Базові поняття булевої алгебри

Означення 2.1. *Булева змінна* – це змінна, що приймає значення з множини $\mathbb{B} = \{0, 1\}$, "0" зазвичай позначає "хибу", а "1" – "істину".

В булевій алгебрі визначені наступні базові операції: кон'юнкція (логічне і, $\wedge$), диз'юнкція (логічне або, $\vee$) та заперечення (логічне ні, $\neg$). Ми також будемо використовувати операцію виключної диз'юнкції (виключне або, $\oplus$).

Означення 2.2. *Булева функція* – це відображення $f : \mathbb{B}^n \to \mathbb{B}$, де $\mathbb{B} = \{0, 1\}$ – множина булевих значень.

Булева функція може бути повністю задана, тобто для $\forall x \in \mathbb{B}^n \ \exists f(x)$, або не повністю задана, в такому випадку функція f задана тільки для певної множини значень з $\mathbb{B}^n$. В даній курсовій роботі ми будемо працювати з повністю заданими функціями.

Означення 2.3. *Булевою алгеброю* називається множина B, яка містить елементи 0 та 1, і на якій задані бінарні операції $+$ та $\cdot$ й унарна операція $\neg$, і при цьому для $\forall x, y, z \in B$ виконуються наступні аксіоми:

$$x + 0 = x \text{ і } x \cdot 1 = x \qquad \text{(ідентичність)}$$
$$x + y = y + x \text{ і } x \cdot y = y \cdot x \qquad \text{(комутативність)}$$
$$(x + y) + z = x + (y + z) \text{ і } (x \cdot y) \cdot z = x \cdot (y \cdot z) \qquad \text{(асоціативність)}$$
$$x + (y \cdot z) = (x + y) \cdot (x + z) \text{ і } x \cdot (y + z) = (x \cdot y) + (x \cdot z) \qquad \text{(дистрибутивність)}$$
$$x + \bar{x} = 1 \text{ і } x \cdot \bar{x} = 0 \qquad \text{(доповнення)}$$

Ми будемо розглядати бінарну булеву алгебру, визначену на множині $\mathbb{B} = \{0, 1\}$, де операції $\mathbf{+}$ відповідатиме диз'юнкція, операції $\mathbf{\cdot}$ відповідатиме кон'юнкція, а операції $\mathbf{\neg}$ відповідатиме заперечення.

Означення 2.4. *Літерал* – булева змінна або її заперечення. Для змінної x маємо *позитивний* літерал x і *негативний* літерал $\bar{x}$.

Означення 2.5. Логічний добуток будь-якої кількості різних незалежних змінних (літералів), що входять із запереченням або без нього, називається *елементарною кон'юнкцією.*

Означення 2.6. Якщо функцію задано формулою у вигляді диз'юнкції елементарних кон'юнкцій, то її задано *диз'юнктивною нормальною формою (ДНФ).*

Наприклад: $f(x) = x_1x_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_2\bar{x}_3$.

Означення 2.7. *Кофактором* f_{x_i} функції f відносно літералу x_i називається значення функції $f(x_0, \dots x_{i-1}, 1, x_{i+1}, \dots x_n)$. Цей кофактор також називають *позитивним кофактором* змінної x_i і позначають f_i^1.

Означення 2.8. *Негативним кофактором* f_i^0 функції f відносно літералу $\bar{x}_i$ називається значення функції $f(x_0, \dots x_{i-1}, 0, x_{i+1}, \dots x_n)$.

Вище були наведені базові поняття булевої алгебри, що знадобляться нам в подальшому.

В наступних підрозділах сформульовано основну задачу логічного синтезу, розглянуто структури даних, які використовуються для представлення булевих схем у програмному забезпеченні, а також наведені теореми, що використані для реалізації алгоритмів оптимізації.

2.2 Основна задача логічного синтезу. Поняття про графи

Як нам уже відомо, основною задачею логічного синтезу є задача оптимізація булевих схем. У математичному вигляді її можна сформулювати наступним чином:

Дано булеву функцію $f : \mathbb{B}^n \to \mathbb{B}$, де $\mathbb{B} = \{0, 1\}$, сконструювати булеву схему, яка задає функцію f з мінімальною кількістю логічних вентилів.

У програмному забезпеченні булеві схеми зручно представляти у вигляді графів, тому згадаємо деякі поняття з теорії графів.

Означення 2.9. *Неорієнтований граф* – це пара $G = (V, E)$, де

$$V = V(G) \text{ – множина вершин,}$$
$$E = E(G) \subset V^{(2)} = \{\{a, b\} \mid a, b \in V\} \text{ – множина ребер.}$$

Означення 2.10. *Орієнтований граф* – це пара $G = (V, E)$, де

$V = V(G)$ – множина вершин,

$E = E(G)$ – множина впорядкованих пар елементів множини V (дуг, петель).

Зауваження 2.1. В даній роботі розглядаються графи без петель та кратних ребер.

Означення 2.11. *Орієнтованим ациклічним графом* (*Directed acyclic graph, DAG*) називається орієнтований граф, в якому відсутні орієнтовані цикли.

Означення 2.12. *Булева мережа* (*Boolean network*) – це орієнтований ациклічний граф, в якому вершинам відповідають логічні вентилі, а ребрам – з'єднання між вентилями.

В даній роботі терміни *булева мережа*, *список з'єднань* і *логічна схема*, є взаємозамінними.

Означення 2.13. *Комбінаційними булевими мережами* називають такі мережі, в яких вихідні сигнали залежать лише від поточних значень вхідних сигналів і не зберігають жодної інформації про попередні вхідні сигнали.

В цій роботі ми будемо розглядати тільки комбінаційні булеві мережі.

Означення 2.14. *Входами* (*fanins*) вершини n називаються вершини, з яких виходять стрілки, які входять до вершини n.

Означення 2.15. *Виходами* (*fanouts*) вершини n називаються вершини, в які входять стрілки, що виходять з вершини n.

Кожна вершина n може мати нуль або більше входів та виходів.

Означення 2.16. *Основні входи* (*primary inputs, PIs*) – вершини, у яких немає входів в даній булевій мережі.

Означення 2.17. *Основні виходи* (*primary outputs, POs*) – підмножина вершин мережі, які забезпечують функціональність мережі в її середовищі. Основні виходи не мають інших виходів в даній мережі.

Означення 2.18. *Розмір* (*площа*) схеми – кількість її вершин.

Означення 2.19. *Глибина* (*затримка*) – кількість вершин найдовшого шляху від основних входів до основних виходів.

Метою оптимізації з використанням локальних перетворень є зменшення площі та затримки мережі.

2.3 And-Inverter граф

Означення 2.20. *Логічний вентиль* – базовий елемент цифрової схеми, що перетворює один або декілька вхідних бінарних сигналів у вихідний бінарний сигнал, таким чином виконує деяку булеву операцію.

Означення 2.21. *Комбінаційний And-Inverter граф* (*And-Inverter Graph, AIG*) – це булева мережа, що складається з двох-входових *and*-вентилів та інверторів.

Для того, щоб отримати And-Inverter граф з ДНФ, необхідно факторизувати вершини, а потім *and*- та *or*-вентилі факторизованих форм перетворити у двох-входові *and*-вентилі та інвертори за допомогою закону Де Моргана. Після цього ці двох-входові *and*-вентилі додаються до менеджера And-Inverter графу у топологічному порядку.

Означення 2.22. *Локальна функція* вершини n мережі, що позначається як $f_n(x)$ – це булева функція логічного конусу з коренем у вершині n, яка виражена в термінах кінцевих вершин, x, розрізу n.

Означення 2.23. *Глобальна функція* вершини n – функція вершини n, яка виражена через всі входи мережі (PIs) і описує логічну операцію, яку виконує вершина n в контексті всього графа.

And-Inverter граф може представляти як локальні, так і глобальні функції. Завдяки малому використанню пам'яті, швидкості обробки та масштабованості, And-Inverter графи стали широко використовуваною структурою даних в області логічноно синтезу та верифікації.

Публікації щодо And-Inverter графів [16] та синтезу на основі And-Inverter графів [17][18] містять додаткову інформацію.

2.4 Xor-And-Inverter граф

Поняття And-Inverter графу може бути розширене, щоб включити двох-входові *xor*-вентилі, і отримати Xor-And-Inverter граф.

Означення 2.24. *Xor-And-Inverter граф* (*Xor-And-Inverter Graph, XAIG*) – булева мережа, що складається з двох-входових *and*-, *xor*-вентилів та інверторів.

Xor-And-Inverter граф конструюється аналогічно до And-Inverter графу.

2.5 Структурне та функціональне хешування

Структурне хешування And-Inverter графу гарантує, що у всіх вершинах мережі наявні всі константи, а для кожної пари вершин існує не більше одного двох-входового *and*-вентиля, для якого дані вершини є входами (з точністю до перестановки). Структурне хешування здійснюється за допомогою таблиць хешування при створенні *and*-вершин та їх додаванні до менеджера And-Inverter графу. Структурне хешування може бути застосовано під час

побудови And-Inverter графу на льоту, що дозволяє зменшити розмір And-Inverter графу.

Функціональне хешування And-Inverter графу забезпечує унікальність булевої функції кожної внутрішньої вершини з точністю до доповнення. Функціональне хешування дає змогу привести And-Inverter граф до "напівканонічного" вигляду (FRAIG [19]), оскільки в даному випадку функція кожної вершини є унікальною, проте ця ж сама булева функція може бути представлена різними FRAIG.

2.6 Розклад Шеннона. Розклад Давіо

Розглянемо основні теореми, які будемо використовувати при побудові алгоритмів в розділі 4.

Однією з найвідоміших теорем в булевій алгебрі є розклад Шеннона [23]. Ця теорема є дуже зручною для використання у рекурсивних процедурах та алгоритмах. Нижче наведено формулювання теореми.

Булеву функцію $f(x_1, \ldots, x_n)$ від n змінних, будемо позначати як f.

Теорема 2.1. *Розклад Шеннона.* Нехай $f : \mathbb{B}^n \to \mathbb{B}$ – булева функція, тоді для $\forall x_i \in \mathbb{B}^n$ виконується наступна тотожність:

$$f = x_i \cdot f_i^1 + \bar{x}_i \cdot f_i^0 \tag{1}$$

Наступна теорема має назву розклад Давіо або розклад Ріда-Мюллера [12] і дає змогу розкласти булеву функцію на кофактори, використовуючи операцію "виключне або". Наведемо формулювання даної теореми.

Теорема 2.2. Нехай $f : \mathbb{B}^n \to \mathbb{B}$ – булева функція, тоді для $\forall x_i \in \mathbb{B}^n$ має місце позитивний та негативний розклад Давіо:

позитивний $$f = f_i^0 \oplus x_i \cdot f_i^2 \tag{2}$$

негативний $$f = f_i^1 \oplus \bar{x}_i \cdot f_i^2 \tag{3}$$

де $f_i^2 = f_i^0 \oplus f_i^1$.

3 Мотивація

В цьому розділі розглядається приклад, що демонструє результати роботи чотирьох алгоритмів, реалізація та побудова яких наведені в розділі 4.

3.1 Приклад

Цілочисельне множення відіграє центральну роль у багатьох обчисленнях. Апаратна реалізація схем для множення є ключовим елементом багатьох конструкцій, в тому числі тих, що використовуються для прискорення обчислень у машинному навчанні. Часто використовується конструкція схеми для множення на основі алгоритму Бута [6].

Саме тому для ілюстрації розглянемо частковий добуток в алгоритмі Бута з основою 4. Частковий добуток – це булева функція з 5-ма входами, таблиця істинності якої, у шістнадцятковій системі числення виглядає як 0xF335ACC0.

Вершини графів на рисунках нижче мають наступну структуру: перший рядок – номер вершини, наступні рядки - функціональне представлення вершини. У функціональному представлення один рядок позначає *and*-вентиль, два рядки – *xor*-вентиль.

На рисунку 3.1 показана схема, що отримана в результаті застосування алгоритму лише на основі розкладу Шеннона з фіксованим порядком змінних. По суті дана схема є бінарною діаграмою рішень з фіксованим порядком змінних [9].

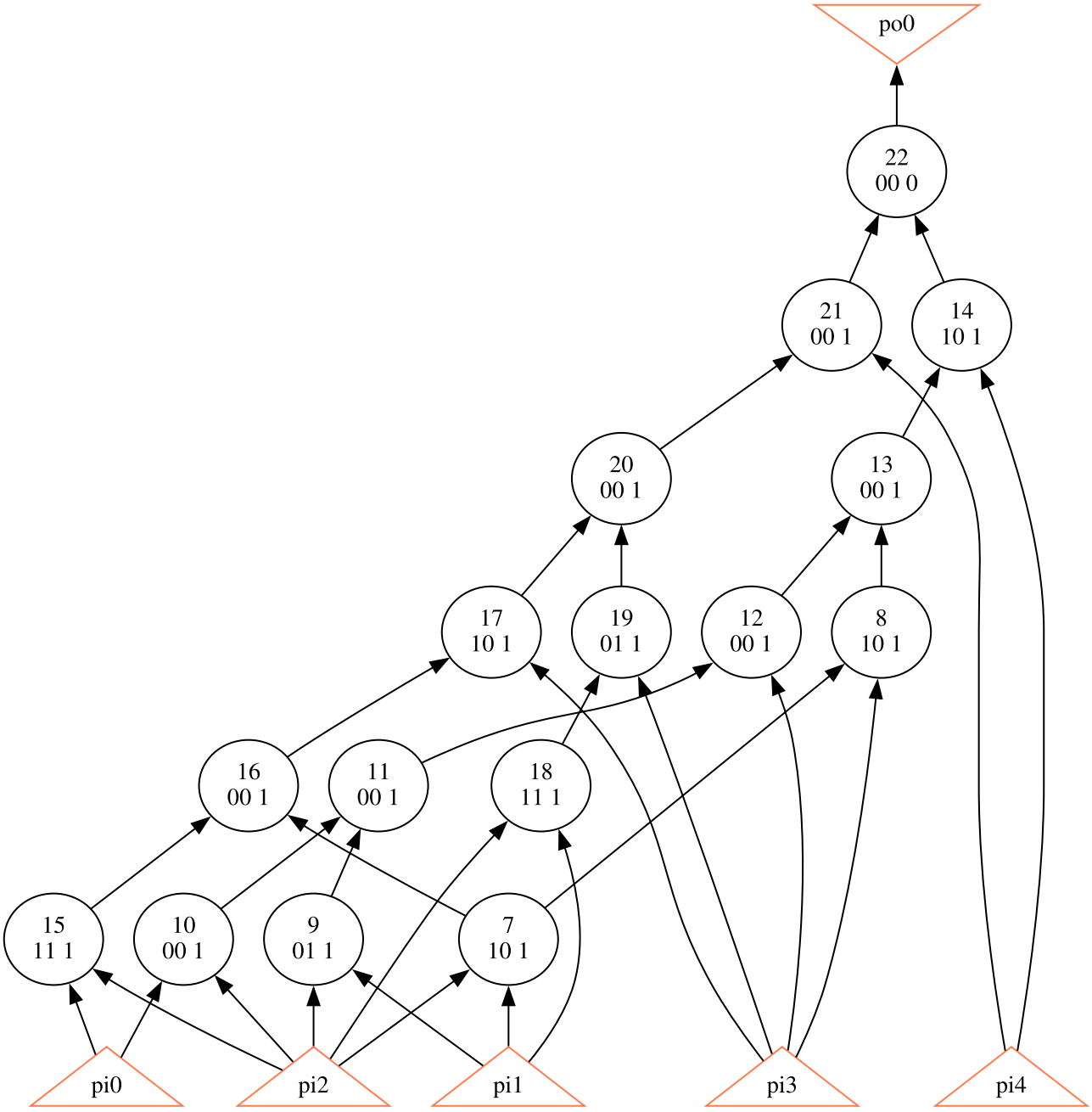

Рисунок 3.1: AI граф, побудований на основі розкладу Шеннона з фіксованим порядком змінних

Якщо виконати перебір всіх порядків синтезу зміних (фактично перебрати

всі перестановки зміних, вперше запропоновано у [22]) і для кожної перестановки застосувати алгоритм на основі розкладу Шеннона, а в кінці вибрати схему із найменшою кількістю вершин, то отримаємо схему із 14 вершинами наведену на рисунку 3.2.

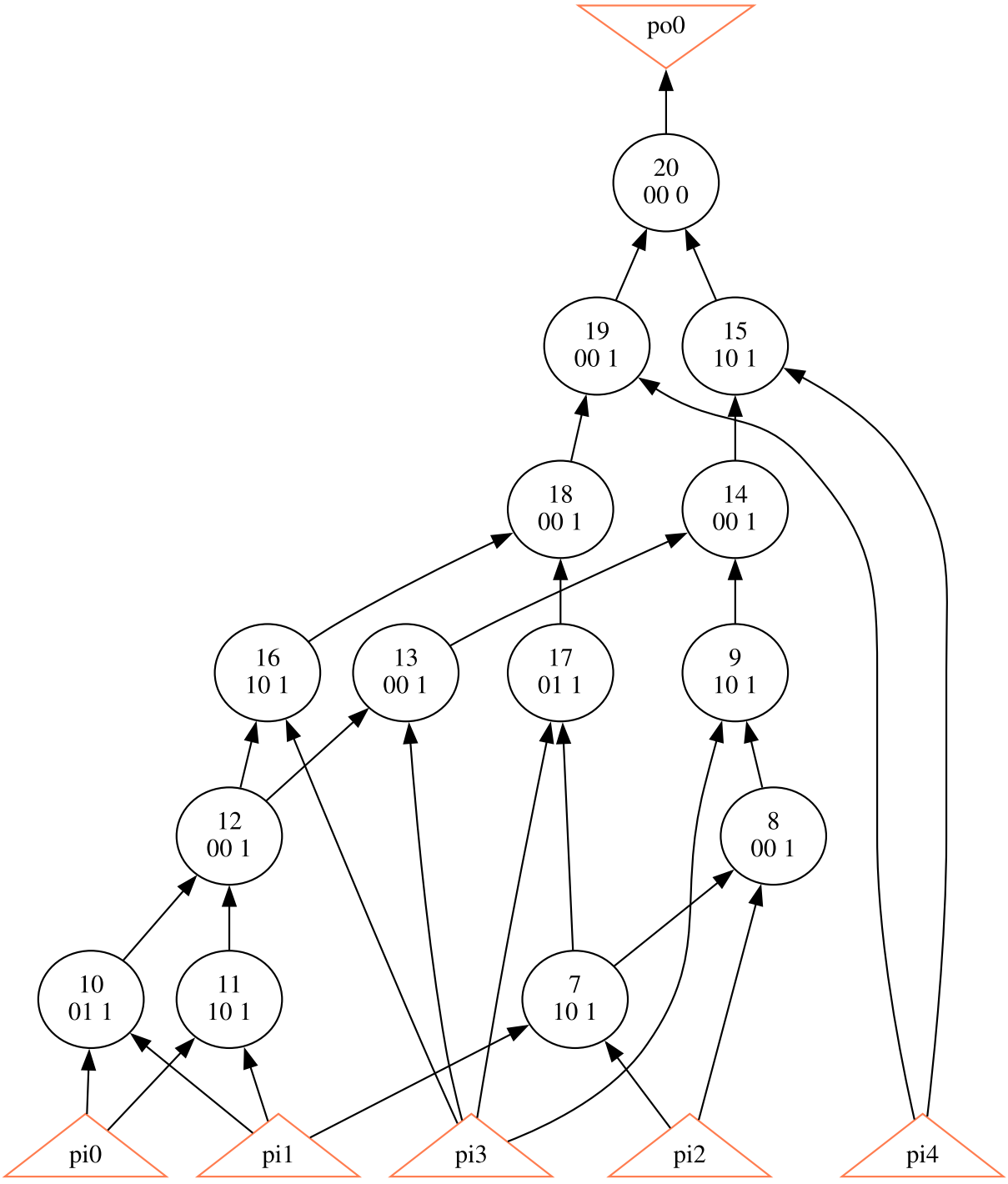

Рисунок 3.2: AI граф, побудований на основі розкладу Шеннона з перебором всіх перестановок

Після використання алгоритму, що базується на трьох канонічних розкладах для фіксованого порядку змінних (вперше описаний у [21]), отримаємо булеву схему, що є Xor-And-Inverter графом з 9-ма вершинами (рисунок 3.3).

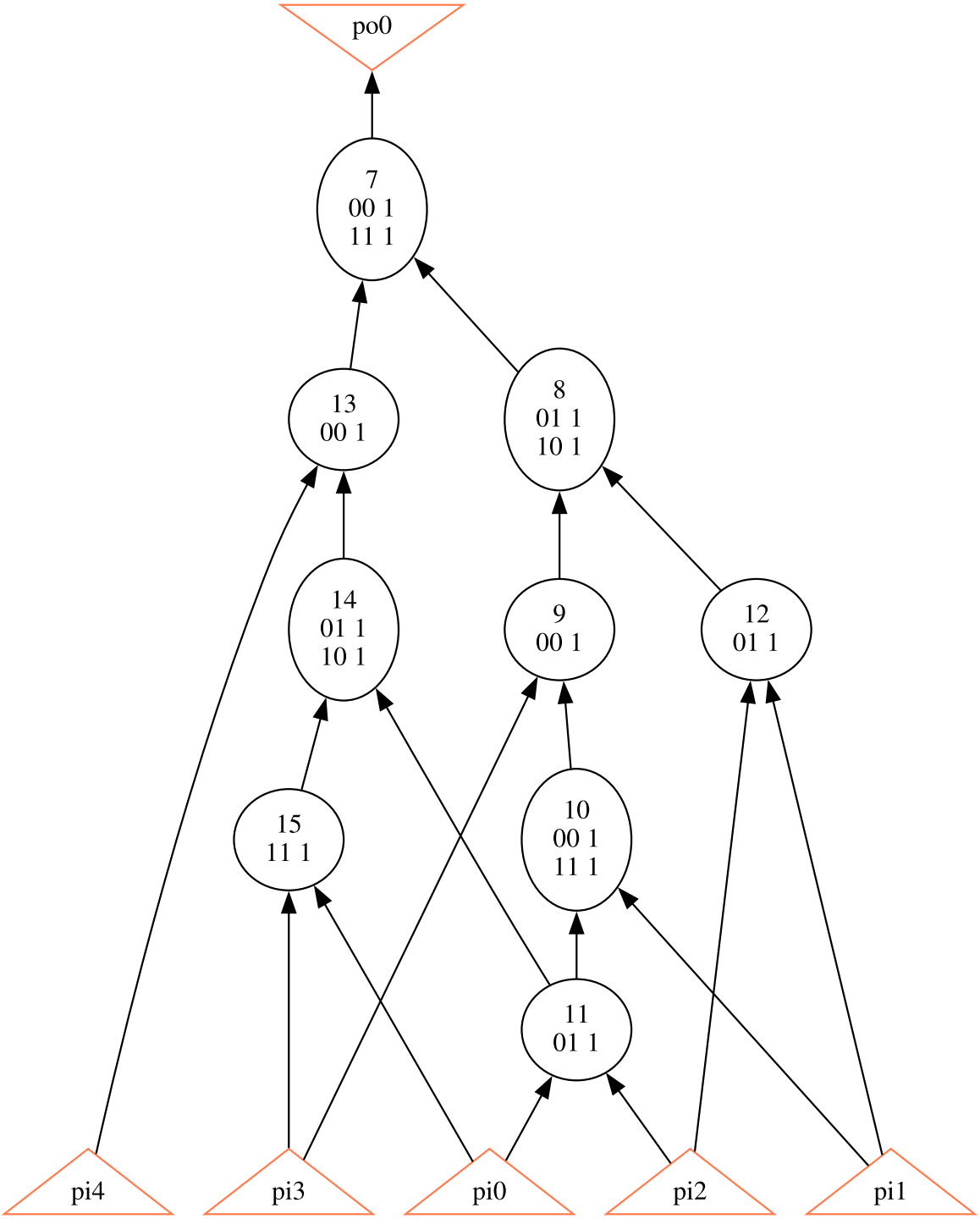

Рисунок 3.3: Мінімізований XAI граф, побудований на основі трьох канонічних розкладів з фіксованим порядком змінних

Алгоритм, що ми побудували, використавши три канонічні розклади та повний перебір порядків синтезу змінних, дає змогу отримати схему з 8 двохвходових вершин (рисунок 3.4).

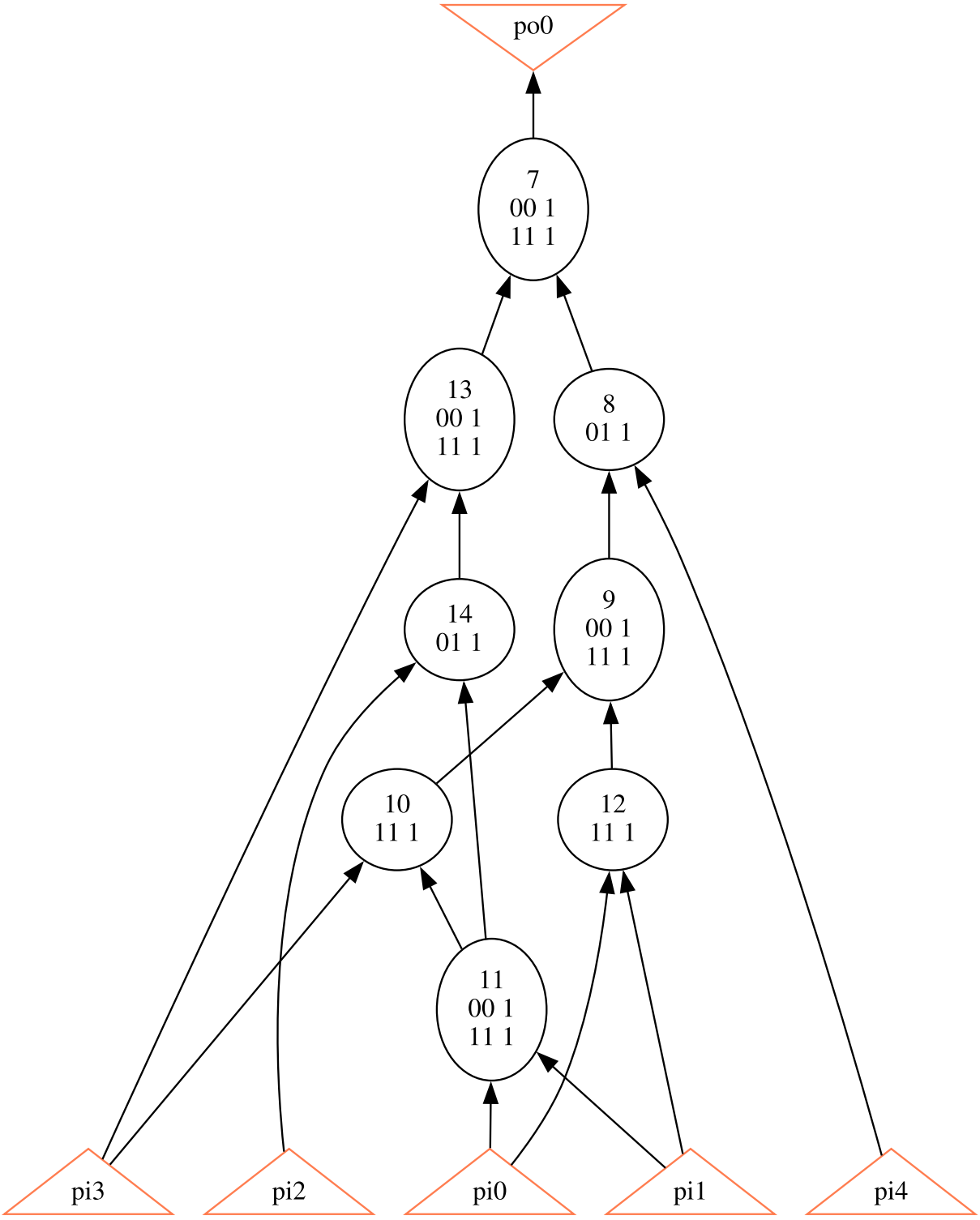

Рисунок 3.4: Мінімізований XAI граф, побудований на основі трьох канонічних розкладів з перебором всіх порядків змінних

Найменша доведена схема для цієї функції складається з 7 вершин (рисунок 3.5), вона була обчислена з використанням точного синтезу на основі SAT [14], реалізованого в команді *twoexact* в системі ABC [1].

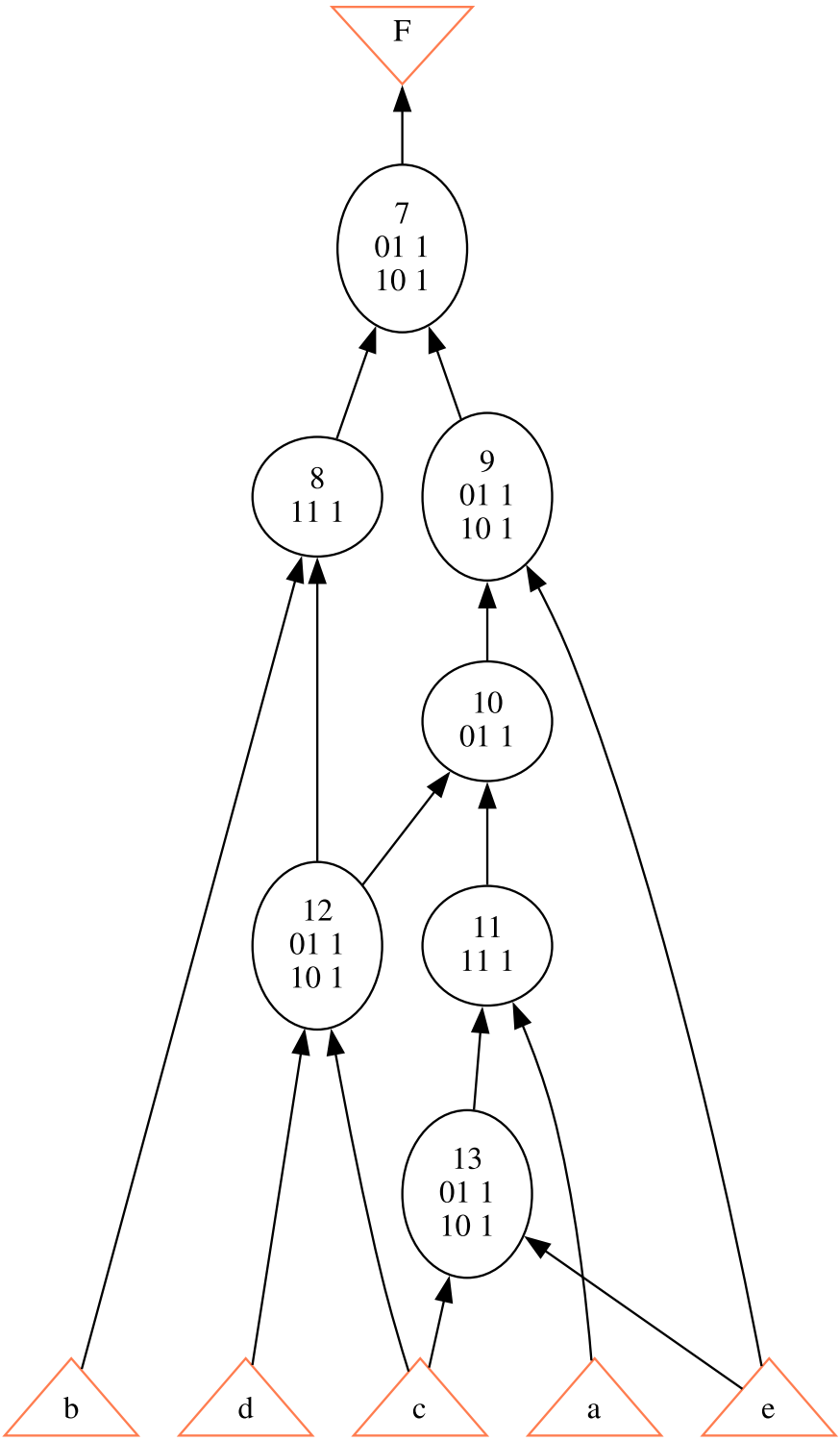

Рисунок 3.5: Доведений мінімальний XAI граф, отриманий за допомогою точного синтезу на основі SAT

Рисунки 3.1 - 3.5 згенеровані за допомогою команди *show* з ABC [1].

Хоча для заданого прикладу часткового добутку в алгоритмі Бута з основою 4 доведену мінімальну схему, що складається з 7-ми вершин (рисунок 3.5), не згенерував жоден із використаних при дослідженні чотирьох алгоритмів, однак запропонований новий алгоритм згенерував найменшу схему, що складається з 8-ми вершин (3.4), серед усіх реалізованих в роботі алгоритмів. При цьому схема з нового алгоритму відрізняється від доведеної мінімальної всього на одну вершину..

В той же час, час генерації схеми за допомогою вказаного нового алгоритму склав менше ніж 0.01 секунди, що більше ніж в 100 разів краще в порівнянні з однією секундою для алгоритму точного синтезу.

Окрім цього, ймовірно, за допомогою "ослаблення" порядку синтезу змінних у кофакторах та додавання евристик для цілеспрямованого пошуку, модифікована версія запропонованого алгоритму зможе згенерувати схему з мінімальною кількістю вершин, витративши на це значно менше часу, ніж алгоритм точного синтезу.

4 Алгоритми оптимізації

Цей розділ присвячений алгоритмам оптимізації. Спочатку розглянуто способи вводу булевих функцій та виводу їх схем. Далі розглянуто алгоритм на основі розкладу Шеннона, запропоновано псевдокод побудованого алгоритму, що базується на трьох канонічних розкладах. Наведено процедуру для генерування перестановок змінних. Також представлена реалізація структурного та функціонального хешування для додаткової оптимізації схем.

4.1 Правила вводу та виводу булевих функцій

Для початку розглянемо способи задання функції в програмній реалізації.

Булева функція може задаватися таблицею істинності у двійковій чи шістнадцятковій системах числення, при цьому таблиця істинності буде записана як стрічка двійкових або шістнадцяткових символів відповідно.

Приклад 4.1. На вхід системі подається шістнадцяткова стрічка “80”. Знайти таблицю істинності в такому випадку.

Двійковий запис вхідної стрічки “80” – 1000 0000. Кількість змінних даної функції можна знайти як $nVars = log_2 a$, де a – кількість цифр у записі стрічки в двійковій системі. Маємо: $log_2 8 = 3$ – кількість змінних.
Порядок розрядів в початковій стрічці є зворотним, тобто перша цифра відповідає варіанту, коли всі змінні набувають значення 1. Таким чином початковій стрічці відповідає таблиця істинності наведена в таблиці 4.1.

a	b	c	$f(a,b,c)$
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Таблиця 4.1: Значення функції $f = 80$

На виході ми будемо отримувати синтезовану схему від вхідної функції, представлену у вигляді And-Inverter графу або Xor-And-Inverter графу, в залежності від використаного алгоритму.

У програмній реалізації вентилі *xor*- та *and*- відображаються парами цілих чисел. Оскільки обидві операції комутативні, то входи вершини можна поміняти місцями без зміни функції цієї вершини. Таким чином порядок входів може бути використаний для того, щоб відрізнити *and*- та *xor*- вентилі: якщо перший вхід менший за другий, то це *and*-вентиль, а, якщо навпаки, то *xor*-вентиль.

4.2 Алгоритм оптимізації на основі розкладу Шеннона

Перш за все розглянемо найпростіший алгоритм оптимізації на основі розкладу Шеннона [9]. Псевдокод рекурсивної процедури обходу графу наведений нижче в цьому підрозділі. На вхід процедура приймає граф, номер найвищої вершини графу та номер поточної змінної, відносно якої відбувається

розклад. Алгоритм знаходить кофактори для поточної змінної і за розкладом Шеннона будує відповідні вершини та додає їх літерали у масив графу. Метод *mux* використовує закон Де Моргана і дозволяє представити операцію *or* через операцію *and*.

```
int synthesis_and_rec(gg *gateGraph, int truthTableId, int varId)
{
    int iLit;
    if ((iLit = hash_function(gateGraph, truthTableId)) >= 0)
        return iLit;
    if (!has_var(gateGraph.funcs, truthTableId, varId))
        return synthesis_and_rec(gateGraph, truthTableId, varId - 1);
    int f0 = cof0(gateGraph.funcs, truthTableId, varId);
    int f1 = cof1(gateGraph.funcs, truthTableId, varId);
    int lit0 = synthesis_and_rec(gateGraph, f0, varId - 1);
    int lit1 = synthesis_and_rec(gateGraph, f1, varId - 1);
    kc_vt_shrink(gateGraph.funcs, 2);
    return mux(gateGraph, kc_varToLiteral(1 + varId, 0), lit1, lit0);
}
```

4.3 Алгоритм оптимізації на основі трьох канонічних розкладів

Псевдокод рекурсивної процедури обходу схеми наведений нижче в цьому підрозділі. На відміну від попереднього алгоритму, тут застосовані три канонічні розклади. В даному випадку, замість побудови *діаграми рішень* [13] і подальшого її обходу, наш алгоритм будує Xor-And-Inverter граф (XAIG) "на ходу". Це досягається тим, що спочатку обраховується розмір підграфу для кожної функції-наступника f_i^0, f_i^1 та f_i^2, а потім окремо для кожного канонічного розкладу розраховується площа всього Xor-And-Inverter графу. Далі алгоритм вирішує, який канонічний розклад дасть мінімальну площу та будує тільки необхідні двох-входові вершини для цього розкладу.

```
int Synthesis_XAIG_Rec(gg *gateGraph, int truthTableId, int varId) {
int res0, res1, res2, res, n01, n02, n12;
        /* a hash table lookup to avoid synthesizing the same function twice */
       /* this step also handles the trivial cases of the constant functions */
        int iLit;
        if ((iLit = hash_function(gateGraph, truthTableId)) >= 0)
                return iLit;
        int f0 = cofactor0(gateGraph.funcs, truthTableId, varId);
        int f1 = cofactor1(gateGraph.funcs, truthTableId, varId);
        int f2 = xor(gateGraph.funcs, f0, f1);
        /* simplification for equal cofactors */
        if (f0 == f1)
        return Synthesis_XAIG_Rec(gg, truthTableId, varId - 1);

        /* recursive traversal */
            int lit0 = Synthesis_XAIG_Rec(gg, f0, varId - 1);
            int lit1 = Synthesis_XAIG_Rec(gg, f1, varId - 1);
            int lit2 = Synthesis_XAIG_Rec(gg, f2, varId - 1);
            kc_vt_shrink(gateGraph.funcs, 3);

        /* compute subgraph size */
        /* Shannon */
        n01 = nodeCount(lit0, lit1) + shannonNodeCount(lit0, lit1);
        /* positive Davio */
        n02 = nodeCount(lit0, lit2) + davioNodeCount(lit0, lit2);
        /* negative Davio */
        n12 = nodeCount(lit1, lit2) + davioNodeCount(lit1, lit2);

        /* construct nodes for minimal expansion */
        int minVal = min(n01, n02, n12);
        if (minVal == n01) {
                return mux(gateGraph, litPos(varId + 1), lit0, lit1);
        } else if (minVal == n02) {
                return and_xor(gateGraph, litPos(varId + 1), lit2, lit0);
        } else {
                return and_xor(gateGraph, litNeg(varId + 1), lit2, lit1);
        }
        return -1;
```

```
}
```

4.4 Генерація перестановок

Для мінімізації отриманого And-Inverter графу з алгоритму на основі розкладу Шеннона та Xor-And-Inverter графу з алгоритму оптимізації на основі трьох канонічних розкладів, використовується рекурсивний алгоритм для перебору всіх можливих перестановок змінних. Алгоритм генерування перестановок змінних [5] отримує наступну перестановку на основі поточної, як показано у псевдокоді нижче:

```
void get_Next_Permutation(int *currentPerm, int nVars) {
        int i = nVars - 1;
        while (i ⩾ 0 && currentPerm[i - 1] ⩾ currentPerm[i])
                i = i - 1;
        if (i ⩾ 0) {
                int j = nVars;
                while (j > 0 && currentPerm[j - 1] ⩽ currentPerm[i - 1])
                        j = j - 1;
                swapElements(j - 1, i - 1);
                i = i + 1;
                j = nVars;
                while (i < j) {
                        swapElements(i - 1, j - 1);
                        i = i + 1;
                        j = j - 1;
                }
        }
}
```

4.5 Реалізація структурного та функціонального хешування

Структурне та функціональне хешування реалізовано в операціях *and* та *xor* і використовується в усіх чотирьох алгоритмах. Псевдокод операції *and* наведений нижче в цьому підрозділі, для операції *xor* процедура виглядає аналогічно.

```
static inline int gateGraph_and(kc_gg *gateGraph, int lit1, int lit2)
{
    // structural hashing
    if (lit1 == 0)
        return 0;
    if (lit2 == 0)
        return 0;
    if (lit1 == 1)
        return lit2;
    if (lit2 == 1)
        return lit1;
    if (lit1 == lit2)
        return lit1;
    if ((lit1 ⊕ lit2) == 1)
        return 0;
    if (lit1 > lit2)
        swap(int, lit1, lit2)
    int truthTableId = truthTable_and(&gateGraph ->truthTables, lit1, lit2);
    // functional hashing
    int lit = hash_node(gateGraph, lit1, lit2, truthTableId);
    if (lit == -1)
        return append_node(gateGraph, lit1, lit2, truthTableId);
    kc_vt_resize(&gateGraph→truthTables, truthTableId);
    return lit;
}
```

Структурне хешування реалізується за рахунок спрощень для елементарних випадків (x та y – літерали вершин, у програмній реалізації представлені

булевими таблицями істинності):

$$x = 0, \forall y: \ x \cdot y = 0 \qquad y = 0, \forall x: \ x \cdot y = 0$$
$$x = 1, \forall y: \ x \cdot y = y \qquad y = 1, \forall x: \ x \cdot y = x$$
$$x = y \Rightarrow x \cdot y = x \cdot x = x \qquad x \oplus y = 1, \Rightarrow x \cdot y = x \cdot \bar{x} = 0$$

Функціональне хешування забезпечується за допомогою перевірки унікальності булевої функції у кожній вершині графу, якщо метод *hash_node* повертає -1, то це означає, що у графі немає вершини з такою булевою функцією і тому ми додаємо нову вершину.

5 Експериментальні результати

Для реалізації алгоритмів було створено програму на мові C++, що використовує таблиці істинності для ефективного представлення булевих функцій. На вхід програма приймає функцію або список функцій, що задані у двійковому або шістнадцятковому вигляді. При заданні функції також задається алгоритм, яким необхідно мінімізувати схему. Результат роботи алгоритмів виводиться в консоль у вигляді списку вершин із двома входами і записується у файл із розширення .aig, який можна візуалізувати з використанням системи ABC [1].

Запропоновані в даній роботі алгоритми реалізовані та перевірені на 33 тестових прикладах з 9 або менше змінними з IWLS 2022 Programming Contest [15]. Коректність роботи алгоритмів перевіряється шляхом побудови таблиць істинності схем і порівняння їх із початковими булевими функціями.

Таблиця 5.1 містить результати роботи алгоритмів синтезу. Стовпчик "*Function*" містить назви файлів із таблицями істинності. Стовпчик "*Ins*" ("*Outs*") вказує кількість основних входів (виходів). Стовпчики "Shannon", "Shannon and variable orders", "Three expansions", "Three expansions and variable orders" представляють площі схем, отриманих за допомогою чотирьох алгоритмів: алгоритм на базі розкладу Шеннона з фіксованим порядком змінних, алгоритм на основі розкладу Шеннона з повним перебором всіх перестановок, алгоритм на базі трьох канонічних розкладів з фіксованим порядком змінних та новий алгоритм на основі трьох канонічних розкладів з повним перебором всіх перестановок відповідно.

Час роботи варіюється від 0.01 секунди до 80 годин (приклад ex31), в залежності від тестового прикладу. Повільність роботи пояснюється необхідністю перебирати всі можливі перестановки змінних, це також обґрунтовує вибір тестових прикладів, з кількістю входів не більше 9, оскільки для 10

змінних кількість перестановок буде становити 3628800, що призводить до дуже великих часових затрат на деяких схемах.

Як бачимо алгоритми, що будують Xor-And-Inverter граф зазвичай дають менший результат, аніж алгоритми на базі розкладу Шеннона, проте експерименти з ex16 – ex20 показали, що схеми, побудовані за допомогою алгоритмів на основі розкладу Шеннона, були менші. Це пояснюється тим, що алгоритми з трьома канонічними розкладами вибирають відповідний розклад на основі розмірів підграфів поточної вершини, що не завжди призводить до мінімального результату загалом. У цих тестових прикладах виявляється, що розклад Шеннона дає меншу результуючу схему. Тим не менш, загалом новий алгоритм з трьома розкладами і повним перебором перестановок залишається більш потужним за три інші досліджувані алгоритми.

Особливістю булевих схем, що були отримані згідно з алгоритмами з трьома розкладами, є те, що вони не можуть бути безпосередньо порівняні зі схемами, що були згенеровані учасниками IWLS 2022 Programming Contest [15], оскільки схеми, представлені учасниками змагань використовують лише *and*-вершини, в той час як схеми в цій курсовій роботі використовують *and*- та *xor*-вершини. Отже, для порівняння результуючих схем, що були згенеровані за допомогою досліджуваних алгоритмів, в цій роботі необхідно спочатку перетворити Xor-And-Inverter графи на And-Inverter графи. Це було зроблено за допомогою команди *&deepsyn –I 10 –J 1000* в системі ABC [1].

В останніх двох стовпчиках таблиці 5.1 наведено результати порівнянь розмірів схем, отриманих новим алгоритмом після перетворення їх на And-Inverter графи із розмірами найкращих схем зі змагань. Висновок полягає в тому, що після обробки, 14 з 33 схем мають такий самий розмір (ex10, ex11, ex12, ex16, ex17, ex28, ex34, ex37, ex41, ex42, ex43, ex49, ex50, ex52), що і найкращі схеми на конкурсі, а в чотирьох прикладах (ex20, ex29, ex51, ex55)

результуюча схема є меншою.

Function	Ins	Outs	Shannon (AIG)	Shannon and variable orders (AIG)	Three expansions (XAIG)	Three expansions and variable orders (XAIG)	Our result after post-processing (AIG)	IWLS'22 contest best result (AIG)
ex00	6	1	41	36	29	23	25	23
ex01	6	1	42	41	28	24	31	27
ex02	8	1	136	126	96	83	99	88
ex03	8	1	41	37	21	19	27	24
ex08	8	8	787	776	588	554	676	544
ex09	8	8	795	766	579	552	662	555
ex10	5	1	18	18	14	14	10	10
ex11	7	1	36	36	24	24	20	20
ex12	9	1	60	60	45	45	30	30
ex16	5	5	30	30	33	33	18	18
ex17	6	6	45	45	53	53	24	24
ex18	7	7	63	63	76	76	34	32
ex19	8	8	84	84	108	108	44	38
ex20	9	9	108	108	140	140	55	56
ex28	7	10	134	77	65	50	39	39
ex29	9	1	61	61	51	51	37	39
ex31	9	18	2188	1924	1751	1571	1671	1351
ex33	5	28	178	171	154	131	84	77
ex34	9	5	432	160	187	107	46	46
ex35	7	2	34	25	22	19	17	15
ex37	8	63	618	432	533	336	147	147
ex38	8	7	98	56	51	47	31	28
ex41	5	3	37	37	17	17	17	17
ex42	7	3	75	75	31	31	28	28
ex43	8	4	100	100	42	42	37	37
ex46	5	8	74	56	39	34	34	32
ex49	7	10	189	77	106	52	39	39
ex50	8	2	62	24	22	12	18	18
ex51	8	2	91	50	21	19	28	29
ex52	8	2	36	27	27	22	19	19
ex53	8	2	117	66	71	51	42	40
ex54	8	2	25	19	13	13	13	12
ex55	8	7	292	253	189	157	148	156
Total			**7127**	**5916**	**5226**	**4510**	**4250**	**3658**

Таблиця 5.1: Кількість двох-входових вершин в мінімізованих схемах, що згенеровані різними алгоритмами

6 Висновки

В даній курсовій роботі було розглянуто один з центральних етапів проектування мікрочіпів – логічний синтез, і його основну задачу – оптимізацію булевих схем. Ми ознайомились з основними поняттями булевої алгебри, необхідними для роботи зі схемами. Було розроблено програму [10] для оптимізації булевих схем із 1-16 входами і довільною кількість виходів, що використовує таблиці істинності для представлення булевих функцій. Використання таблиць істинності дозволило спростити код та прискорити обчислення.

У програмі реалізовано наступні алгоритми оптимізації: алгоритм на основі розкладу Шеннона (4.2), алгоритм на основі розкладу Шеннона (4.2) з використанням всіх перестановок змінних (4.4), алгоритм із використанням розкладів Шеннона і Давіо (4.3) та фіксованим порядком змінних і алгоритм, що використовує три канонічні розклади (4.3) для кожної перестановки змінних (4.4).

Новизною цієї роботи стала розробка алгоритму, що використовує три канонічні розклади і повний перебір перестановок змінних для синтезу.

Подальша робота може включати дослідження евристичних компромісів між якістю та часом виконання. Наприклад, можливо обмежити вичерпне перебір порядків синтезу змінних, зберігаючи мінімальність отриманих схем, або дозволити довільний порядок змінних при кофакторизації, як у вільних *бінарних діаграмах рішень* (*BDD*) [2].

Результати досліджень, виконаних в рамках цієї курсової роботи, були також викладені у статті «Area Minimization Using Binary Decision Diagrams Without Constructing Them», яка була подана й прийнята на конференцію RM2023.

Література

[1] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification.* URL: http://www-cad.eecs.berkeley.edu/~alanmi/abc.

[2] J. Bernet, M. Görgen, C. Meinel, and A. Slobodova. "Boolean manipulation with free BDD's. First experimental results". In: *Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC* (1994), pp. 200–207. DOI: 10.1109/EDTC.1994.326915.

[3] V. Bertacco and M. Damiani. "The disjunctive decomposition of logic functions". In: *1997 Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)* (1997), pp. 78–82. DOI: 10.1109/ICCAD.1997.643371.

[4] P. Bjesse and A. Borälv. "DAG-aware circuit compression for formal verification". In: *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004* (2004), pp. 42–49. DOI: 10.1109/ICCAD.2004.1382541.

[5] E. Blurock. *Generate All Permutations of an Array.* URL: https://www.baeldung.com/cs/array-generate-all-permutations.

[6] A. D. Booth. "A signed binary multiplication technique". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 4 (2 1951), pp. 236–240. DOI: 10.1093/qjmam/4.2.236. URL: https://uma.ac.ir/files/site1/g_zare_3fd00a8/booth51.pdf.

[7] R. K. Brayton and C. T. McMullen. "The decomposition and factorization of Boolean expressions". In: *Proceedings of the 15th International Symposium on Circuits and Systems.* 1982, pp. 29–54.

[8] R. K. Brayton and A. Mishchenko. "ABC An Academic Industrial-Strength Verification Tool". In: *Proceedings of the 22nd Computer Aided Verification International Conference*. Vol. 6174. Lecture Notes in Computer Science. 2010, pp. 24–40. DOI: 10.1007/978-3-642-14295-6_5. URL: https://link.springer.com/content/pdf/10.1007/978-3-642-14295-6_5.pdf.

[9] R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35 (1986), pp. 677–691. DOI: 10.1109/TC.1986.1676819. URL: https://www.cs.cmu.edu/~bryant/pubdir/ieeetc86.pdf.

[10] K. Cherevko. *Area Minimization algorithms*. URL: https://github.com/criistiiina/logic-synthesis-reed-muller.

[11] D. I. Cutress. *TSMC Financial Year 2022*. 2022. URL: https://morethanmoore.substack.com/p/tsmc-financial-year-2022.

[12] M. Davio, A. Thayse, and J.-P. Deschamps. *Discrete and switching functions*. eng. Advanced book program. St. Saphorin Switzerland: Georgi Publishing Co., 1978. ISBN: 0070155097.

[13] R. Drechsler. "Pseudo Kronecker Expressions for Symmetric Functions". In: *IEEE Transactions on Computers* 48 (9 1999), pp. 987–990. DOI: 10.1109/12.795226. URL: https://people.eecs.berkeley.edu/~alanmi/publications/other/tc99_drechsler.pdf.

[14] W. Haaswijk, M. Soeken, A. Mishchenko, and G. D. Micheli. "SAT-based exact synthesis: Encodings, topology families, and parallelism". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.4 (2020), pp. 871–884. DOI: 10.1109/TCAD.2019.2897703. URL:

https://si2.epfl.ch/~demichel/publications/archive/2020/winston-exact.pdf.

[15] *IWLS 2022 Programming Contest*. 2022. URL: https://github.com/alanminko/iwls2022-ls-contest.

[16] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21.12 (2002), pp. 1377–1394. DOI: 10.1109/TCAD.2002.804386. URL: https://people.eecs.berkeley.edu/~alanmi/courses/2005_290A/papers/01097859.pdf.

[17] A. Mishchenko and R. Brayton. "Scalable Logic Synthesis using a Simple Circuit Structure". In: *Proceedings of the 15th International Workshop on Logic & Synthesis*. 2006, pp. 15–22. URL: https://people.eecs.berkeley.edu/~alanmi/publications/2006/iwls06_sls.pdf.

[18] A. Mishchenko, S. Chatterjee, and R. Brayton. "DAG-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis". In: *Proceedings of the 43rd annual Design Automation Conference*. 2006, pp. 532–535. DOI: 10.1145/1146909.1147048. URL: https://people.eecs.berkeley.edu/~brayton/publications/2006/dac06_rwr.pdf.

[19] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. "FRAIGs: A Unifying Representation for Logic Synthesis and Verification". In: *ERL Technical Report*. 2005. URL: https://people.eecs.berkeley.edu/~alanmi/publications/2005/tech05_fraigs.pdf.

[20] Y. Miyasaka, A. Mishchenko, J. Wawrzynek, and N. J. Fraser. "Synthesizing practical Boolean functions using truth tables". In: *Proceedings of the 31st International Workshop on Logic & Synthesis*. 2022. URL: https://

people.eecs.berkeley.edu/~alanmi/publications/2022/iwls22_reo.pdf.

[21] M. A. Perkowski, M. Chrzanowska-Jeske, A. Sarabi, and I. Schäfer. "Multi-Level Logic Synthesis Based on Kronecker Decision Diagrams and Boolean Ternary Decision Diagrams for Incompletely Specified Functions". In: *VLSI Design*. Vol. 3. 1995, pp. 301–313. DOI: 10.1155/1995/24594. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3c19e969d720f02b774e28db2f7c6265c57e840e.

[22] R. Rudell. "Dynamic variable ordering for ordered binary decision diagrams". In: *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. 1993, pp. 42–47. DOI: 10.1109/ICCAD.1993.580029.

[23] C. E. Shannon. "The synthesis of two-terminal switching circuits". In: *The Bell System Technical Journal* 28.1 (1949), pp. 59–98. DOI: 10.1002/j.1538-7305.1949.tb03624.x.