КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня магістр
за спеціальністю 122 Комп'ютерні науки
на тему:

СИНТЕЗ КОМБІНАЦІЙНИХ СХЕМ ТА КІНЦЕВИХ АВТОМАТІВ. МЕРЕЖА ПЕТРІ

Виконав студент 2 курсу
магістратури
Микола ПУЧКО-КОЛЕСНИК _____
(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Олексій ФЕДОРУС _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри математичної інформатики
«____» _____ 202_ р.,
протокол № ___
Завідувач кафедри
В. М. Терещенко _____
(підпис)

Київ – 2023

РЕФЕРАТ

Мережі Петрі – математичний апарат для моделювання динамічних дискретних систем. Вперше описані Карлом Петрі у 1962 році .

Мережа Петрі є дводольним орієнтованим графом , що складається з вершин двох типів - позицій і переходів, з'єднаних між собою дугами, вершини одного типу не можуть бути з'єднані безпосередньо. У позиціях можуть розміщуватись мітки (маркери), здатні переміщуватися по мережі.

Мережа Петрі – інструмент для моделювання динамічних систем. Теорія мереж Петрі робить можливим моделювання системи математичним уявленням її у вигляді мережі Петрі, аналіз якої допомагає отримати важливу інформацію про структуру та динамічну поведінку системи, що моделюється.

Можливо кілька шляхів практичного застосування мереж Петра при проектуванні та аналізі систем. У одному з підходів мережі Петрі розглядаються як допоміжний інструмент аналізу. Тут для побудови системи використовуються загальноприйняті методи проектування, потім побудована система моделюється мережею Петрі, і побудована модель аналізується.

В іншому підході весь процес проектування та визначення характеристик проводиться у термінах мереж Петрі. І тут завдання полягає у перетворенні представлення мережі Петрі на реальну інформаційну систему.

Безперечною перевагою мереж Петрі є математично суворий опис моделі. Це дозволяє проводити їхній аналіз за допомогою сучасної обчислювальної техніки (у тому числі з масово-паралельною архітектурою).

Мережі Петрі сутнісно є однією з форм імітації дискретних процесів. Вони були у великій моді років 20 тому, коли за їхньою допомогою сподівалися розраховувати згадані процеси (без імітації). У переважній більшості застосувань від звичайних імітаційних моделей вони відрізняються лише великою наукоподібністю та специфічною термінологією.

Мережі Петрі – інструмент дослідження систем. Нині мережі Петрі застосовуються переважно у моделюванні. У багатьох областях досліджень явище вивчається безпосередньо, а побічно, через модель.

Модель - це уявлення, зазвичай, у математичних термінах те, що вважається найбільш характерним у досліджуваному об'єкті чи системі. Маніпулюючи моделлю системи, можна отримати нові знання про неї, уникаючи небезпеки, дорожнечі або незручності аналізу самої реальної системи. Зазвичай моделі мають математичну основу.

Розвиток теорії мереж Петрі проводилося за двома напрямками. Формальна теорія мереж Петрі займається розробкою основних засобів, методів і понять, необхідні застосування мереж Петри. Прикладна теорія мереж Петрі пов'язана головним чином із застосуванням мереж Петрі до моделювання систем, їх аналізу та глибоким проникненням, що виходить в результаті цього в моделювані системи.

Моделювання в мережах Петрі проводиться на подійному рівні. Визначаються, які дії відбуваються у системі, які стан передували цим діям та які стану прийме система після виконання дії. Виконання подієвої моделі у мережах Петрі визначає поведінка системи. Аналіз результатів виконання може сказати у тому, у яких станах перебувала чи перебувала система, які стану у принципі недосяжні. Однак такий аналіз не дає числових характеристик, що визначають стан системи. Розвиток теорії мереж Петрі призвело до появи, про, “кольорових” мереж Петрі. Поняття кольоровості тісно пов'язане з поняттями змінних, типів даних, умов та інших конструкцій, більш наближених до мов програмування. Незважаючи на деякі подібності між кольоровими мережами Петрі та програмами, вони ще не застосовувалися як мова програмування.

Незважаючи на описані вище переваги мереж Петрі, незручності застосування мереж Петрі як мову програмування укладені у процесі виконання у обчислювальної системі. У мережах Петрі немає суворо поняття процесу, який можна було б виконувати на зазначеному процесорі. Немає також однозначної послідовності виконання мережі Петрі, оскільки вихідна теорія представляє нам мову для опису паралельних процесів.

Перша частина курсової роботи присвячена мінімізації булевих функцій двома різними способами, а також побудові комбінаційних схем у базисах, що складаються лише з однієї функції.

Друга частина містить основні поняття та визначення з теорії кінцевих автоматів, а також приклад їх використання для конкретного автомата. Сюди входить мінімізація кінцевих автоматів за кількістю станів, мінімізація булевих функцій, що описують комбінаційну частину з подальшою реалізацією отриманого автомата на логічних елементах із певного базису та елементах пам'яті – тригерах та затримках.

У третій частині розглянуто питання аналізу функціонування та програмного моделювання мереж Петрі. Різними методами вивчені поведінкові характеристики заданої мережі Петрі. Складено найпростішу програму, що моделює всі ситуації, що виникають в мережі.

ОЗНАЧЕННЯ

Мінімізація булевих функцій є процесом спрощення логічних виразів або таблиць істинності з метою зменшення кількості логічних елементів, використаних для їх реалізації. Це важливий крок у синтезі комбінаційних схем, оскільки дозволяє скоротити використання ресурсів і підвищити ефективність системи.

Комбінаційна схема - це цифрова схема, в якій виходи залежать від поточних значень входів. Вона складається з комбінації логічних вентилів (AND, OR, NOT тощо) та зв'язків між ними. Комбінаційні схеми використовуються для виконання логічних операцій і обчислень, зазвичай без використання пам'яті.

Мінімізація кінцевих автоматів є процесом спрощення структури автомата з метою зменшення кількості станів, переходів та логічних елементів, не змінюючи його функціональності. Мінімізація допомагає покращити продуктивність автомата, зменшити витрати пам'яті та поліпшити його реалізацію.

ЗМІСТ

ВСТУП

Робота присвячена синтезу дискретних пристроїв з “пам'яттю” (кінцевих автоматів) та “без пам'яті” (комбінаційних схем), а також аналізу реальних процесів за допомогою мереж Петрі.

У першій частині розглянута мінімізація булевих функцій, заданих у вигляді СДНФ, за допомогою двох різних способів: карт Карно та методу склеювання Квайна – МакКласкі. Отримані у вигляді мінімізованих ДНФ функції були приведені до базисів, що складаються лише з однієї функції: І – НЕ та АБО – НЕ, а потім реалізовані у вигляді комбінаційних схем на відповідних логічних елементах.

У другій частині заданий за умовою у функціональному вигляді кінцевий автомат було мінімізовано за кількістю станів. Для отриманого автомата було збудовано граф станів. Потім, перейшовши до двійкового подання вхідних, вихідних сигналів і сигналів стану, в автоматі були виділені елементи пам'яті і комбінаційна частина, яка була мінімізована за кількістю змінних. Автомат був реалізований у базисі І – АБО – НЕ з використанням D – тригера та затримки.

У третій частині була проаналізована задана мережа Петрі за допомогою двох способів: матричного та заснованого на побудові дерева покриваності, а також написана програма для її моделювання.

1 СИНТЕЗ КОМБІНАЦІЙНИХ СХЕМ

1.1 Постановка задачі

Для двох булевих функцій, побудованих за варіантом завдання у вигляді

$$F_1 = F_1(x_1, x_2, x_3, x_4) = F_1[g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8](x_1 x_2 x_3 x_4) \qquad (1.1.1)$$

$$F_2 = F_2(x_1, x_2, x_3, x_4) = F_1[z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8](x_1 x_2 x_3 x_4), \qquad (1.1.2)$$

де g_i, z_i – десяткові числа з діапазону від 0 до 15 у двійковому вигляді, зробити таке:

а) подати F_1 і F_2 у вигляді СДНФ.

б) мінімізувати (за кількістю змінних до ДНФ) F_1 с допомогою карт Карно, F_2 - методом Квайна-МакКласкі.

в) реалізувати як комбінаційної схеми на логічних елементах F_1 – у базисі І – НЕ, F_2 – у базисі АБО – НЕ, попередньо привівши F_1 та F_2 до відповідних базисів.

g_i та z_i обчислювати за виразами:

$$g_i = (z_{i-1} + i) \bmod 16 \qquad (1.1.3)$$

$$z_i = (g_{i-1} + 1) \bmod 16 \qquad (1.1.4)$$

при g_0 = A , z_0 = B. _ Параметр i змінювати від 1 доти, доки не буде отримано 9 різних значень g_i і z_i.

1.2 Теоретичні відомості.

Булевою алгеброю називається безліч S об'єктів A, B, C…, в якому визначено дві бінарні операції (логічне додавання – диз'юнкція (+) та логічне множення – кон'юнкція (·)) та одна унарна операція (логічне заперечення ($\overline{A}$)). Воно має такі властивості:

а) Для $\forall\ A, B, C\ \in\ S$

1) $A + B \in S$, $AB \in S$ (*замкнутість*);

2) $\left.\begin{matrix} A + B = B + A \\ AB = BA \end{matrix}\right\}$ (*Комутативні закони*);

3) $\left.\begin{matrix} A + (B + C) = (A + B) + C \\ A(BC) = (AB)C \end{matrix}\right\}$ (*асоціативні закони*);

4) $\left.\begin{matrix} A(B + C) = AB + AC \\ A + BC = (A + B)(A + C) \end{matrix}\right\}$ (*Дистрибутивні закони*);

5) $A + A = AA = A$ (*Властивості ідемпотентності*);

6) $A + B = B$ у тому й лише тому випадку, якщо

$AB = A$ (*Властивість сумісності*);

7) S містить елементи *1* та *0* такі, що для будь-якого елемента $A \in S$

$A + 0 = A, A1 = A, A0 = 0, A + 1 = 1$;

8) для кожного елемента клас A містить елемент $\tilde{A}$ (доповнення елемента A , часто позначається символами $\bar{A}$ або $1 - A$) такий, *що*

$A + \tilde{A} = I$, $A\tilde{A} = 0$.

У кожній булевій алгебрі

$A(A + B) \equiv A + AB \equiv A$ (*Закони поглинання*),

$(A + B)(A + \overline{B}) = AB + A\overline{B} = A$ (*Закони склеювання*),

$\left.\begin{matrix} \overline{(A + B)} \equiv \overline{A}\,\overline{B} \\ \overline{(AB)} \equiv \overline{A} + \overline{B} \end{matrix}\right\}$ (*Двоїстість, закони де Моргана*).

Якщо дані n булевих змінних X_1 , X_2 ,…, X_n , кожна з яких може дорівнювати будь-якому елементу булевої алгебри, то булевою функцією називається вираз

$$Y = F(X_1, X_2, ..., X_n) \qquad (1.2.1)$$

У кожній булевій алгебрі існує рівно 2^{2^n} *різних булевих функцій* n *змінних.*

Система булевих функцій називається повною (базисом), якщо будь-яка функція може бути представлена як суперпозиції функцій обраної системи.

Під критерієм мінімізації (спрощення) булевих функцій розумітимемо досягнення мінімуму букв у записі функції.

Введемо поняття багатовимірного куба.

Будь-яку булеву функцію n змінних, задану в ДНФ або СДНФ, можна відобразити на n -вимірному кубі, побудованому в ортогональному базисі n булевих змінних. Кожне доданок в ДНФ або СДНФ є гіперплощиною відповідної розмірності: якщо воно є кон'юнкцією n змінних - точка, n -1 змінних - пряма, n -2 змінних - площина і т.д. Елементи n -мірного куба, що мають вимірювання s , назвемо s -кубами.

Комплекс $K(y)$ кубів функції $y = f(x_1, x_2, \ldots, x_n)$ є об'єднання $K^s(y)$ множин її кубів. Відсутні в кон'юнкціях змінні позначатимемо через x.

1.3 Розрахунки та отримані результати.

За варіантом завдання знаходимо g_i і z_i:

i	g_i	z_i
0	*5*	*0*
1	*1*	*6*
2	*8*	*2*
3	*5*	*9*
4	*13*	*6*
5	*11*	*14*
6	*4*	*12*
7	*3*	*5*
8	*13*	*4*
9	*13*	*14*
10	*8*	*14*
11	*9*	*9*
12	*5*	*10*
13	*7*	*6*

Неповторювані значення g_i: 5, 1, 8, 13, 11, 4, 3, 9, 7. Неповторювані значення zi: 0, 6, 2, 9, 14, 12, 5, 4, 10. Таким чином, для F_1 отримуємо вираз

$$F_1(x_1x_2x_3x_4) = [0101,0001,1000,1101,1011,0100,0011,1001,0111](x_1x_2x_3x_4), \quad (1.3.1)$$

для F_2:

$$F_2(x_1x_2x_3x_4) = [0000,0110,0010,1001,1110,1100,0101,0100,1010](x_1x_2x_3x_4). \quad (1.3.2)$$

Для мінімізації першої функції застосовуємо метод карток Карно.

Карта Карно - прямокутник з 2^n клітинами, кожній з яких відповідає своя кон'юнкція з n змінних та їх заперечень (доповнень).

Проставляючи одиниці у відповідних клітинах, вибираємо потім мінімальну з усіх можливих комбінацію покриттів. Застосуємо картку Карно до заданої функції:

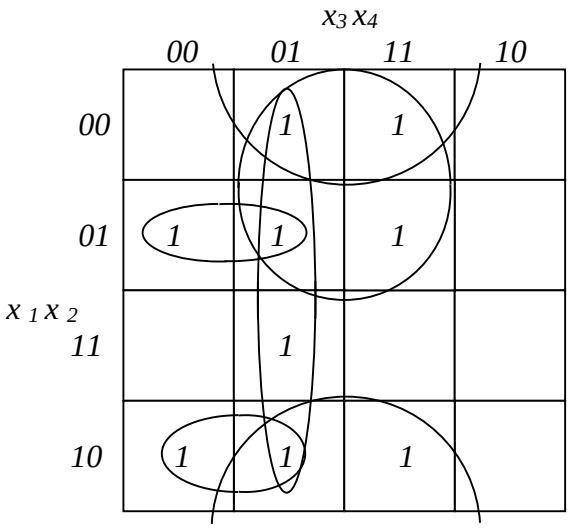

Рисунок 1.2.1 – картка Карно

На підставі обраної комбінації покриттів виписуємо мінімізований вираз для функції F_1:

$$F_1 = \bar{x}_3 x_4 + \bar{x}_2 x_4 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 x_3 x_4 . \qquad (1.3.3)$$

Для другої функції застосовуємо метод Квайна-МакКласкі.

На першому кроці алгоритму виписуємо комплекс K^0-Кубів заданої функції, упорядкованих за зростанням кількості одиниць:

$$K^0 = \left\{ \begin{array}{c|c|c|c} 0 & 0\,0 & 0\,0\,1\,1\,1 & 1 \\ 0 & 0\,1 & 1\,1\,0\,0\,1 & 1 \\ 0 & 1\,0 & 0\,1\,0\,1\,0 & 1 \\ 0 & 0\,0 & 1\,0\,1\,0\,0 & 0 \end{array} \right\} . \qquad (1.3.4)$$

Другий етап ґрунтується на операції склеювання. Кожен із кубів перевіряється на “склеюваність” з усіма іншими. Куби, що склеюються, повинні відрізнятися не більше ніж в одному розряді. Склеєний розряд надалі

позначається як x . Куб, який брав участь у операції склеювання, відповідним чином позначається. Оскільки таких кубів мало, відзначатимемо куби, що не брали участі в операції склеювання. В результаті отримуємо комплекс K^1 -кубів, також упорядкований за зростанням кількості одиниць у розрядах:

$$K^1 = \left\{ \begin{array}{c|c|c} 0\,0 & 0\,x\,0\,0\,x & x\,1\,1 \\ 0\,x & x\,0\,1\,1\,1 & 1\,x\,1 \\ x\,0 & 1\,1\,0\,x\,0 & 1\,1\,x \\ 0\,0 & 0\,0\,x\,0\,0 & 0\,0\,0 \end{array} \right\}. \qquad (1.3.5)$$

Повторюємо вищеописану операцію для комплексу K^1 -кубів, після чого видаляємо з отриманого комплексу K^2 -кубів, що повторюються:

$$K^2 = \left\{ \begin{array}{c|c} 0\,0 & x\,x\,x\,x \\ x\,x & x\,x\,1\,1 \\ x\,x & 1\,1\,x\,x \\ 0\,0 & 0\,0\,0\,0 \end{array} \right\} = \left\{ \begin{array}{c|c} 0 & x\,x \\ x & x\,1 \\ x & 1\,x \\ 0 & 0\,0 \end{array} \right\} \qquad (1.3.6)$$

Ті куби, які не брали участь в операціях склеювання, називаються імплікантами – це кандидати на те, щоб потрапити до підсумкової ДНФ. Їх складаємо таблицю покриттів K^0 -кубів. Імпліканта вважається покриває K^0 - Куб, якщо вони збігаються при x , що приймає довільне значення.

K^0 / z	0 0 0 0	0 0 1 0	0 1 0 0	0 1 1 1	0 1 0 0	1 0 0 1	1 0 1 0	1 1 0 0	1 1 1 0	*Імпліканти*
1001						+				$x_1\bar{x}_2\bar{x}_3x_4$
010x			+	+						$\bar{x}_1x_2\bar{x}_3$
0xx0	+	+	+		+					$\bar{x}_1\bar{x}_4$
xx10		+			+		+		+	$x_3\bar{x}_4$
x1x0			+		+			+	+	$x_2\bar{x}_4$

Таблиця 1.3.1 - Покриття K^0 - кубів

Істотною імплікантою, або екстремаллю, називається така імпліканта, яка в однині покриває хоча б один з K^0-Кубів.

З таблиці випливає, що це імпліканти є екстремалями. Отже, всі вони увійдуть у запис функції у вигляді скороченої ДНФ:

$$F_2 = \bar{x}_1\bar{x}_4 + x_2\bar{x}_4 + x_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3x_4\,. \qquad (1.3.7)$$

Комбінаційна схема - це дискретний пристрій, кожен із вихідних сигналів якого в момент часу t_m визначається так:

$$y_j(t_m) = f\,(x_1(t_m), x_2(t_m),..., x_n(t_m))\,, \qquad (1.3.8)$$

де $x_i, y_j \in \{0;1\}$. Видно, що вихідний сигнал в m-й момент часу визначається тільки комбінацією вхідних сигналів в даний момент і не залежить від попередніх значень. Тому комбінаційну схему можна реалізувати на логічних елементах, що виконують операції з певного базису булевих функцій.

Наведемо F_1 до базису І – НЕ, а F_2 – до базису АБО – НЕ:

$$F_1 = \overline{\overline{x_4(\bar{x}_3 + \bar{x}_2 + \bar{x}_1 x_3)}} + \overline{\overline{\bar{x}_3(\bar{x}_1 x_2 + x_1 \bar{x}_2)}} = \overline{\bar{x}_4 + (\overline{\bar{x}_1 x_3} \cdot x_2 \cdot x_3)} + \overline{\bar{x}_3 \overline{(x_1 \cdot \overline{(x_1 x_2)} \cdot (x_2 \cdot \overline{(x_1 x_2)}))}} =$$

$$= \overline{(x_4 \cdot \overline{(\bar{x}_1 x_3} \cdot x_2 \cdot x_3))} \cdot \overline{(\bar{x}_3 \cdot \overline{(x_1 \cdot \overline{(x_1 x_2)})(x_2 \cdot \overline{(x_1 x_2)})})}; \qquad (1.3.9)$$

$$F_2 = x_1 \bar{x}_2 \bar{x}_3 x_4 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 \bar{x}_4 + \bar{x}_4 x_2 + x_3 \bar{x}_4 =$$

$$= \overline{(\bar{x}_1 + \bar{x}_4 + x_2 + x_3)} + \overline{(x_1 + \bar{x}_2 + x_3)} + \overline{(x_1 + x_4)} + \overline{(\bar{x}_2 + x_4)} + \overline{(\bar{x}_3 + x_4)} \ . \qquad (1.3.10)$$

Отримавши вирази для функцій, приведених до відповідних базисів, можна намалювати комбінаційні схеми, що реалізують ці функції, на елементах одного виду: для першої функції це будуть І – НЕ-елементи, для другої – АБО – НЕ:

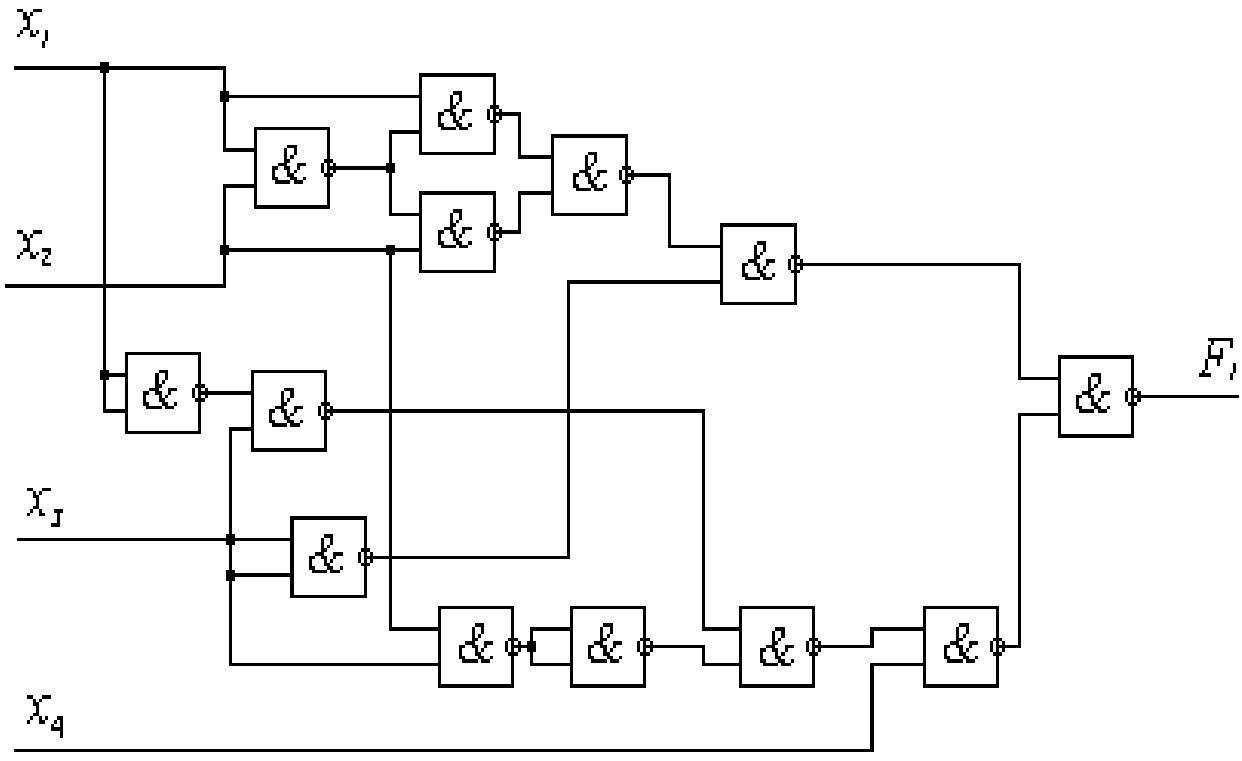

Рисунок 1.3.1 – Схема на І – НЕ-елементах

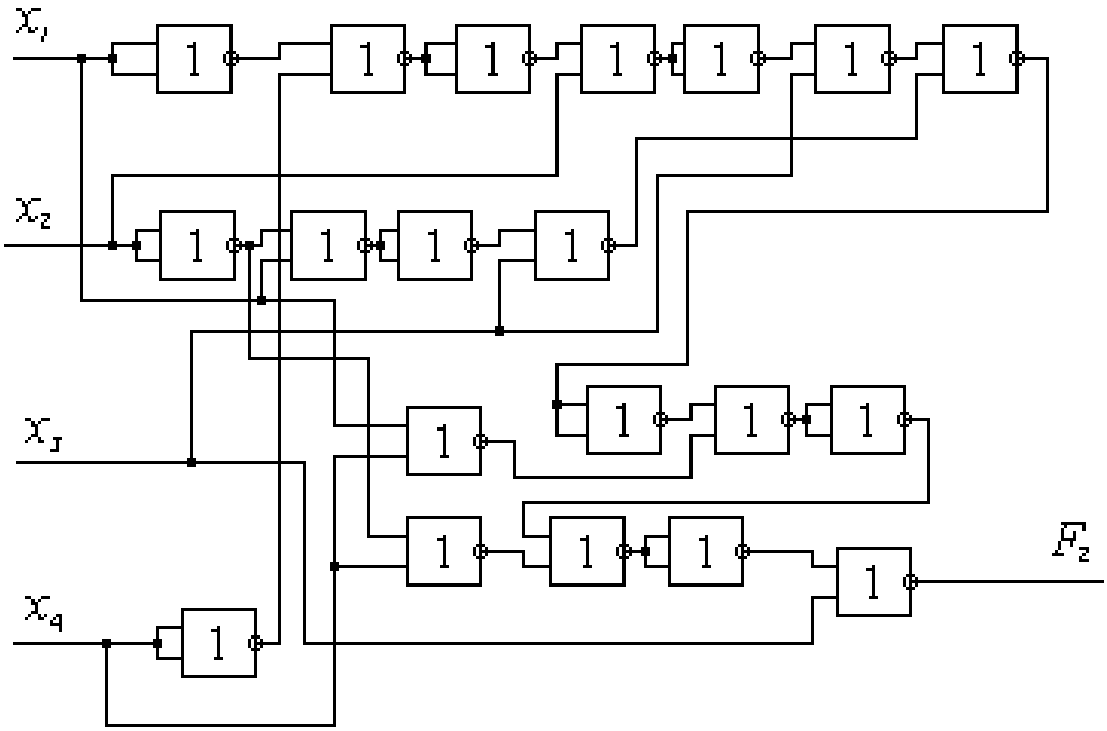

Рисунок 1.3.2 – Схема на АБО – НЕ-елементах

1.4 Висновки у розділі

У першій частині було розглянуто приклади мінімізації (спрощення) булевих функцій двома різними способами. Було практично показано можливість приведення функцій двох аргументів до базису, що складається всього з однієї функції. Було побудовано комбінаційні схеми, що ілюструють отримані результати. Вигода розглянутих перетворень функцій стає очевидною за її практичної реалізації на стандартизованих електронних мікросхемах.

2 СИНТЕЗ КІНЦЕВИХ АВТОМАТІВ

2.1 Постановка задачі

Кінцевий автомат заданий своїми рівняннями переходів та виходів:

$$s(j+1) = [2 \cdot s(j) + x(j) + B] \bmod 8,$$

$$y(j) = [s(j) + x(j) + A] \bmod 2,$$

$$x(j) \in X = \{0,1,2,3\} .$$

Потрібно:

а) побудувати таблиці переходів, виходів та загальну таблицю переходів автомата;

б) мінімізувати автомат за кількістю станів із використанням таблиць, отриманих раніше;

в) побудувати граф мінімізованого автомата та виписати для нього матрицю переходів;

г) переходячи до двійкового уявлення входу X , виходу Y та стану S скласти таблицю входів і виходів комбінаційної схеми автомата і виконати мінімізацію булевих функцій, відповідних виходам і станам автомата;

д) розробити логічну схему автомата у базисі І – НЕ, реалізуючи елементи пам'яті на тригерах і затримках.

2.2 Теоретичні відомості

Кінцевим автоматом називається такий дискретний пристрій, вихідні сигнали якого в певні моменти часу залежать не тільки від останнього вхідного сигналу, що прийшов, але і від деякої кількості попередніх його значень.

Розрізняють синхронні та асинхронні автомати. У асинхронних зміна вихідних сигналів $y(t_j)$ може відбуватися тільки в моменти зміни вхідних $x(t_j)$, у синхронних – у моменти часу, що визначаються додатковим синхронізуючим сигналом $c(t)$.

Визначимо множини, яким можуть належати вхідні та вихідні сигнали (умовимося позначати t_j як j):

$\begin{cases} x_i \in X_i, \\ y_i \in Y_i \end{cases}$ - безліч вхідних і вихідних сигналів.

Тоді вирази

$$\begin{aligned} X &= X_1 \otimes X_2 \otimes \ldots \otimes X_n, \\ Y &= Y_1 \otimes Y_2 \otimes \ldots \otimes Y_n \end{aligned} \qquad (2.2.1)$$

визначають вхідний та вихідний алфавіти автомата.

Нехай $X_i \in \{0;1\}, Y_i \in \{0;1\}$. Тоді якщо $y(j) = \lambda(x(j))$, то цей автомат є, очевидно, комбінаційною схемою.

Введемо додаткову змінну для того, щоб охарактеризувати стан автомата у кожний момент часу j:

$$\begin{aligned} &s_i(j) \in S_i, \\ &s(j) \in S = S_1 \otimes S_2 \otimes \ldots \otimes S_k \end{aligned} \qquad (2.2.2)$$

В тому випадку якщо X, Y і S - Кінцеві множини, то і сам автомат називають кінцевим.

У вигляді рівнянь будь-який кінцевий автомат можна записати різними способами. Одна з можливих форм запису:

$$\begin{cases} s(j+1) = \delta(x(j), s(j)) \\ y(j) = \lambda(x(j), s(j)) \end{cases} \quad (2.2.3)$$

Записаний у такий спосіб автомат називається автоматом Милі. Зрозуміло, що це – більш інформативна форма запису порівняно з автоматом Мура:

$$\begin{cases} s(j+1) = \delta(x(j), s(j)) \\ y(j) = \lambda(s(j)) \end{cases} \quad (2.2.4)$$

Методи завдання автоматів.

По-перше, автомат може бути заданий безпосередньо рівняннями виду (2.2.3) або (2.2.4).

По - друге, рівняння (2.2.3) та (2.2.4) можуть бути представлені в табличній формі. Табличний аналог першого рівняння (2.2.3) називається таблицею переходів, другого – таблицею виходів.

По - третє, таблиці переходів і виходів можна об'єднати в одну. Вміст кожної клітини є дріб: над косою рисою вписується відповідне значення з таблиці переходів, під косою рисою – значення таблиці виходів. Отримана таким чином таблиця називається загальною таблицею переходів та виходів кінцевого автомата.

Граф автомата – це сигнальний граф, вершини якого позначають стани автомата, на дугах відбито умови переходу зі стану на стан і значення вихідних сигналів як дробу: над косою рисою – x (j*) , під нею – y (j).*

Кінцевий автомат також можна описати за допомогою матриці переходів. Це аналог графа у табличній формі. Вона являє собою квадратну матрицю розмірності кількість станів $\times$ кількість станів, в якій відображені умови переходу зі стану в стан аналогічно зображеним на графі.

Загальне визначення кінцевого автомата:

$$M = (X, Y, S, \delta, \lambda), \quad (2.2.5)$$

де X - вхідний алфавіт, Y - Вихідний алфавіт, S - безліч станів, δ - функція переходів, λ - функція виходів.

Нехай є два автомати: M і M'.

Якщо для будь-якого $s \in S(M)$ існує принаймні одне $s' \in S'(M')$ еквівалентне йому, то кажуть, що M' покриває M : $M' \geq M$.

Якщо одночасно $M' \geq M$ і $M \geq M'$, то $M \sim M'$. Отримуємо еквівалентні автомати. І тут неможливо розрізнити M і M' з їхньої реакції на вхідні сигнали.

Існують два основні положення визначення поняття еквівалентності станів:

а) стани s_i та s_j явно різні, якщо відповідні їм рядки у таблиці виходів різні;

б) стани s_i та s_j явно еквівалентні, якщо відповідні їм рядки у загальній таблиці переходів автомата однакові або стають такими при заміні s_i на s_j або навпаки.

Мінімізація (спрощення) автоматів ґрунтується на понятті еквівалентних автоматів. Під мінімізацією автомата розумітимемо скорочення числа його станів.

2.3 Розрахунки та отримані результати.

Побудова таблиць переходів, виходів та загальної таблиці переходів та виходів на основі заданих рівнянь автомата Мілі очевидно:

x(j) / s(j)	0	1	2	3
0	1	0	1	0
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0
5	0	1	0	1
6	1	0	1	0
7	0	1	0	1

Таблиця 2.3.1 - Таблиця виходів автомата

x(j) / s(j)	0	1	2	3
0	0	1	2	3
1	2	3	4	5
2	4	5	6	7
3	6	7	0	1
4	0	1	2	3
5	2	3	4	5
6	4	5	6	7
7	6	7	0	1

Таблиця 2.3.2 - Таблиця переходів автомата

x(j) / s(j)	0	1	2	3
0	0/1	1/0	2/1	3/0
1	2/0	3/1	4/0	5/1
2	4/1	5/0	6/1	7/0
3	6/0	7/1	0/0	1/1
4	0/1	1/0	2/1	3/0
5	2/0	3/1	4/0	5/1
6	4/1	5/0	6/1	7/0
7	6/0	7/1	0/0	1/1

Таблиця 2.3.3 – Загальна таблиця переходів та виходів автомата

Перейдемо безпосередньо до мінімізації отриманого автомата за станом. Для цього скористаємося алгоритмом, відомим у літературі як метод Гриса – Хопкрофта. Його перевага в тому, що він гарантує мінімальну форму автомата. Однак у загальному випадку він досить трудомісткий і застосовується, як правило, для автоматів з невеликою кількістю станів. Він ґрунтується на властивості транзитивності еквівалентності

$$(s_i \sim s_j) \cap (s_j \sim s_k) \Rightarrow (s_i \sim s_k) \qquad (2.3.1)$$

Пара еквівалентних станів переходить при всіх можливих значеннях входу в пари еквівалентних станів.

Алгоритм складається з наступних кроків.

Спочатку розбиваємо всі стани автомата на множини за ознакою збігу вихідних сигналів. У нашому випадку отримуємо 2 множини: $S_1 = \{0, 2, 4, 6\}$ і $S_2 = \{1, 3, 5, 7\}$.

Щоб назвати кожен із отриманих класів новим станом, потрібно переконатися, що у кожен клас входять лише еквівалентні між собою стану. Для цього складаємо таблицю пар еквівалентних станів. При цьому не забуваємо про те, що еквівалентними можуть бути стани, що належать лише одному класу. У таблицю заносимо всі ті пари, які переходять при відповідних входах вихідні, ймовірно еквівалентні, пари:

пари	*0*	*1*	*2*	*3*
0;2	*0;4*	*1; 5*	*2;6*	*3;7*
0;4	*0; 0*	*1;1*	*2;2*	*3;3*
0;6	*0;4*	*1; 5*	*2;6*	*3;7*
2;4	*4;0*	*5;1*	*6;2*	*3;7*
2;6	*4;4*	*5;5*	*6;6*	*7;7*
4;6	*0;4*	*1; 5*	*2;6*	*3;7*
1;3	*2;6*	*3;7*	*4;0*	*5;1*
1; 5	*2;2*	*3;3*	*4;4*	*5;5*
1;7	*2;6*	*3;7*	*4;0*	*5;1*
3;5	*6;2*	*7;3*	*0;4*	*1; 5*
3;7	*6;6*	*7;7*	*0; 0*	*1;1*
5;7	*2;6*	*3;7*	*4;0*	*5;1*

Таблиця 2.3.4 - Таблиця пар еквівалентних станів

Шукаємо в отриманій таблиці нееквівалентні пари - пари з різних множин. У таблиці таких немає, отже, остаточно отримуємо автомат із двома новими станами – позначимо їх *0* та *1* .

Наступним кроком оформляємо загальну таблицю переходів для мінімізованої форми автомата:

x(j) / s(j)	0	1	2	3
0	0/1	1/0	0/1	1/0
1	0/0	1/1	0/0	1/1

Таблиця 2.3.5 - Нова загальна таблиця переходів.

На підставі отриманої загальної таблиці переходів та виходів можна намалювати граф мінімізованого автомата з двома станами:

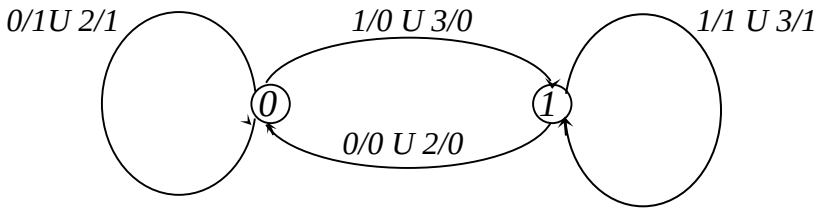

Рисунок 2.3.1 – Граф мінімізованого автомата

Для практичної реалізації отриманого автомата треба двійково закодувати всі сигнали. Для кодування *y* і *s* достатньо одного двійкового розряду, *x* вимагає двох - x_1 і x_2:

x	x_1	x_2
0	0	0
1	0	1
2	1	0
3	1	1

Таблиця 2.3.6 – Двійкове кодування *x*

Складаємо таблицю істинності для комбінаційної частини схеми з урахуванням таблиці (2.3.5). Отримуємо дві функції трьох аргументів :

$x_1(j)$	0	0	0	0	1	1	1	1
$x_2(j)$	0	0	1	1	0	0	1	1
$s(j)$	0	1	0	1	0	1	0	1
$y(j)$	1	0	0	1	1	0	0	1
$s(j+1)$	0	0	1	1	0	0	1	1

Таблиця 2.3.7 - Таблиця істинності комбінаційної частини

Кожну з функцій $y(j)$ та $s(j+1)$ мінімізуємо за допомогою карт Карно:

Рисунок 2.3.2 – Карти Карно для комбінаційної частини

На підставі вибраних покриттів записуємо мінімізовані вирази для функцій переходів та виходів:

$$y(j) = x_2(j) \cdot s(j) + \bar{x}_2(j) \cdot \bar{s}_2(j) = \overline{x_2(j) \oplus s(j)} \qquad (2.3.2)$$

$$s(j+1) = x_2(j) \qquad (2.3.3)$$

Реалізуємо отримані функції як комбінаційної схеми, додаючи до неї елементи пам'яті – D - тригер і затримку. Комбінаційну частину реалізуємо в базисі І – АБО – НЕ.

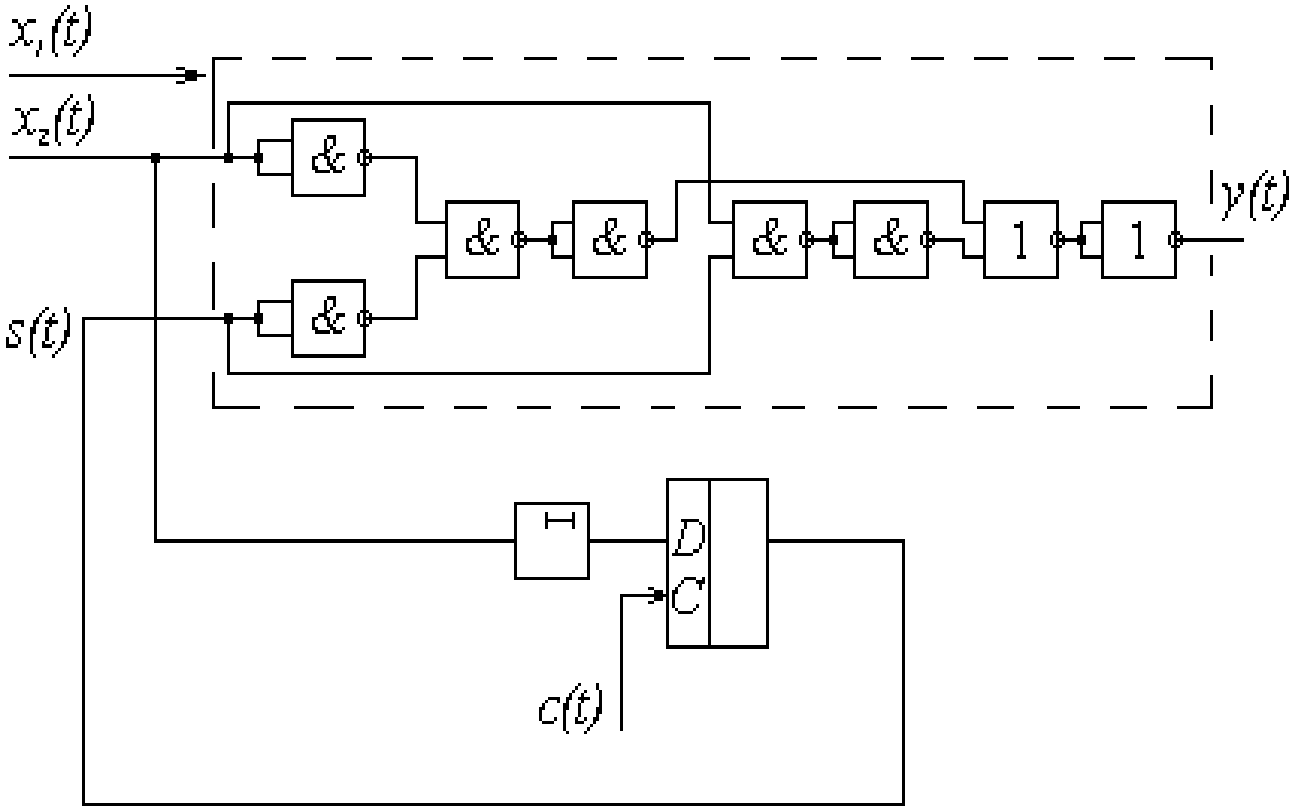

Рисунок 2.3.2 – Схема мінімізованого автомата в базисі І – АБО – НЕ

2.4 Висновки у розділі

У цьому розділі було показано приклад мінімізації (спрощення) кінцевого автомата зі скороченням кількості станів, і навіть приклад реалізації автомата на логічних елементах і пам'яті. Ми переконалися в тому, що кінцевий автомат є розширенням поняття комбінаційної схеми на випадок, коли для отримання вихідного сигналу в даний момент потрібно "пам'ятати" деяку кількість попередніх значень вхідного сигналу, а не тільки його поточне значення. При практичній реалізації автомата стала очевидною користь проведених операцій зі спрощення вихідного автомата та приведення його комбінаційної частини до конкретного базису.

3 МЕРЕЖІ ПЕТРІ

3.1 Постановка задачі

Для заданої мережі Петрі, що описує розподіл ресурсів для двох процесів, зробити таке:

а) виписати матричне рівняння зміни маркувань;

б) побудувати дерево та граф покриваності маркувань;

в) описати поведінкові властивості мережі на основі графа покриваності та матричних рівнянь;

г) виписати безліч досяжних з μ_0 маркувань;

д) розробити програму моделювання мережі Петрі.

3.2 Теоретичні відомості

Мережі Петрі – найбільш вдалий з існуючих математичний апарат для моделювання, аналізу, синтезу та проектування різних дискретних систем з паралельно протікаючими процесами.

Визначення. Мережею Петрі називається четвірка елементів

$$C = (P, T, I, O), \tag{3.2.1}$$

де

$$P = \{p_1, p_2, ..., p_n\}, n > 0 \tag{3.2.2}$$

безліч позицій (кінцеве),

$$T = \{t_1, t_2, ..., t_m\}, m > 0 \tag{3.2.3}$$

безліч переходів (кінцеве),

$$I: T \rightarrow P \qquad (3.2.4)$$

функція входів (відображення безлічі переходів у вхідні позиції),

$$O: T \rightarrow P \qquad (3.2.5)$$

функція виходів (відображення безлічі переходів у вихідні позиції).

Якщо $p_i \in I(t_j)$, то p_i - Вхідна позиція j-го переходу, якщо $p_i \in I(t_j)$, то p_i - Вихідна позиція j - го переходу.

Для наочного уявлення мереж Петрі використовуються графи.

Граф мережі Петрі є дводольним орієнтованим мультиграфом.

$$G = (V, \vec{E}), \qquad (3.2.6)$$

де $V = P \cup T$, причому $P \cap T = \varnothing$.

Виходячи з графічного уявлення мережі Петрі, її можна визначити і так:

$$C = (P, T, A), \qquad (3.2.7)$$

де A - матриця інцидентності графа мережі.

Визначимо поняття маркованої мережі Петрі - є ключовим для будь-якої мережі.

Маркування μ мережі Петрі $C = (P, T, I, O)$ є функція:

$$N = \mu(P), N \in \mathbf{N}, \qquad (3.2.8)$$

відображає безліч позицій на множину натуральних чисел. Маркування можна також визначити як вектор:

$$\mu = \{\mu_1, \mu_2, \ldots, \mu_n\}, \qquad (3.2.9)$$

де $n = \left| P \right|$, а $\mu_i \in \boldsymbol{N}$. Між цими визначеннями є зв'язок:

$$\mu_i = \mu\,(p_i) \qquad (3.2.10)$$

На графі маркування відображається з відповідним числом точок у кожній позиції. Крапки називаються маркерами або фішками. Якщо фішок багато (більше трьох), їх кількість відображається числом.

Таким чином, маркована мережа Петрі є п'ятіркою елементів:

$$M = (P, T, I, O, \mu). \qquad (3.2.11)$$

Приклад найпростішої мережі Петрі:

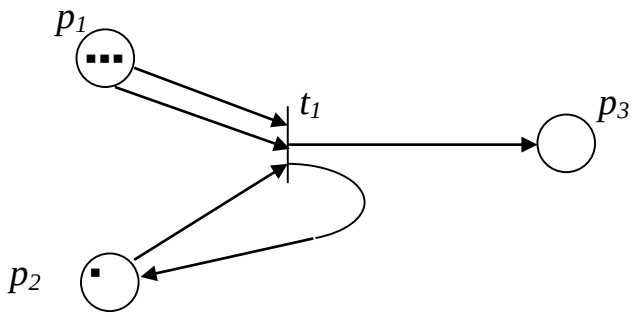

Рисунок 3.2.1 – Приклад мережі Петрі

Правила роботи із мережами Петрі.

Мережа Петрі виконується у вигляді запуску переходів. Перехід може бути запущений у тому випадку, коли його дозволено. Перехід є дозволеним, якщо кожна з його вхідних позицій містить число фішок не менше ніж число дуг з неї в даний перехід.

Процедура запуску полягає у видаленні з кожної вхідної позиції переходу числа фішок, рівного числу дуг з неї, і у виставленні в кожній вихідній позиції числа фішок, що дорівнює кількості дуг, що входить до неї.

Проілюструємо сказане з прикладу вже намальованої мережі Петрі. Запустимо в ній перехід t_1 – він є дозволеним:

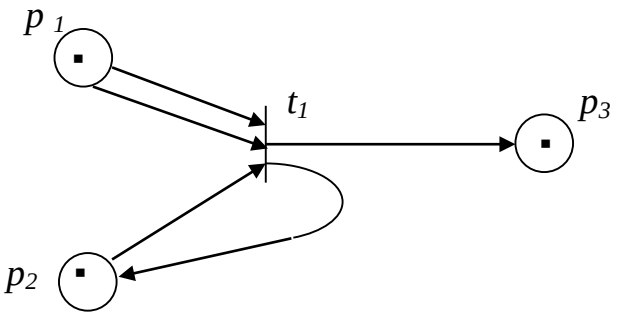

Рисунок 3.2.2 – Приклад запуску переходу мережі Петрі

Простір станів та поведінкові властивості мереж Петрі.

Нехай є маркована мережа Петрі:

$$M = (P, T, I, O, \mu) \qquad (3.2.12)$$

У неї n позицій. У кожній позиції не більше N фішок. Тоді простір стоянь є безліч всіх можливих маркувань мережі. Визначимо δ – функцію наступного стану.

Якщо перехід t_j дозволений під час поточного маркування μ, то наступне маркування μ' визначиться так:

$$\mu' = \delta (\mu, t_j) \qquad (3.2.13)$$

Якщо перехід t_j не дозволено, то δ не визначено.

Нехай $\{t_{j\ 0},\ t_{j1},\ldots,\ t_{js}\}$ - Послідовність запущених переходів. Тоді їй відповідатиме послідовність $\{\mu^0, \mu^1, \ldots, \mu^{s+1}\}$, тобто

$$\mu^{k+1} = \delta (\mu^k, t_{jk}) \qquad (3.2.14)$$

З останньої рівності можна визначити поняття безпосередньо досяжної маркування. Для мережі $C = (P, T, I, O)$ маркування μ' називається безпосередньо досяжним з μ, якщо існує такий перехід $t_j \in T$, при якому

$$\mu' = \delta (\mu, t_j) \qquad (3.2.15)$$

Можна розповсюдити це поняття на безліч досяжних з цієї маркувань. Визначимо безліч досяжних з маркування μ $R(C, \mu)$ наступним чином:

по - перше, $\mu \in R(C, \mu)$;

по - друге, якщо $\mu' \in R(C, \mu)$, $\mu' = \delta(\mu, t_j)$ і $\mu'' = \delta(\mu', t_k)$, то і $\mu'' \in R(C, \mu)$

.

На основі введених понять можна сформулювати низку властивостей мережі Петрі, що характеризують її в процесі зміни маркувань – назвемо їх поведінковими властивостями мережі Петрі. Визначимо найважливіші їх.

1 Досяжність даного маркування. Нехай є деяке маркування μ, відмінне від початкового. Тоді виникає питання досяжності: чи можна шляхом запуску певної послідовності переходів перейти з початкового в задане маркування.

2 Обмеженість. Мережа Петрі називається k - обмеженою, якщо за будь-якого маркування кількість фішок у будь-якій з позицій не перевищує k. Зокрема, мережа називається безпечною, якщо k дорівнює 1. Крім того, мережа називається однорідною, якщо в ній відсутні петлі та одинарний (простий), якщо в ній немає кратних дуг.

3 активність. Мережа Петрі називається активною, якщо незалежно від досягнутої з μ_0 маркування існує послідовність запусків, що веде до запуску цього переходу.

Реально вводять поняття кількох рівнів активності для конкретних переходів. Перехід $t_j \in T$ називається:

а) пасивним (*L0* - активним), якщо він ніколи не може бути запущений;

б) *L1* активним, якщо він може бути запущений послідовністю переходів з μ_0 хоча б один раз;

в) *L2* активним, якщо для будь-якого числа K існує послідовність запусків переходів з μ_0, при якій даний перехід може спрацювати K і більше разів;

г) *L3* - активним, якщо він є *L2* - активним при $K \to \infty$.

4 Оборотність. Мережа Петрі оборотна, якщо для будь-якого маркування $\mu \in R(C, \mu_0)$ маркування μ_0 досяжна з μ.

5 Покриваність. Маркування μ покривається, якщо існує інше маркування $\mu' \in R(C, \mu_0)$ така, що у кожній позиції μ' фішок не менше, ніж у позиціях маркування μ.

6 Стійкість. Мережа Петрі називається стійкою, якщо будь-яких двох дозволених переходів спрацьовування однієї з них не призводить до заборони спрацьовування іншого.

Існують два основні методи аналізу мереж Петрі: матричні та засновані на побудові дерева покриваності.

Перша група методів заснована на матричному поданні маркувань та послідовностей запуску переходів. Для цього визначимо дві матриці розмірності, кількість позицій, × кількість переходів, пов'язаних зі структурою мережі. Перша матриця називається матрицею входів:

$$\boldsymbol{D}^{-}[i, j] = \#(p_i, I(t_j)), \qquad (3.2.16)$$

кожен її елемент дорівнює числу фішок, що йдуть з j -ї позиції при запуску i -го переходу. Друга матриця називається матрицею виходів:

$$\boldsymbol{D}^{+}[i, j] = \#(p_i, O(t_j)), \qquad (3.2.17)$$

кожен її елемент дорівнює числу фішок, що приходять в j -ю позицію при запуску i -го переходу. Визначимо одиничний вектор $e[j]$ розмірності m, що містить нулі у всіх позиціях крім тієї, яка відповідає переходу, що запускається в даний момент. Вочевидь, що перехід дозволено, якщо $\mu \geq e[j] \cdot \boldsymbol{D}^{-}$. Тоді результат запуску j -го переходу можна описати так:

$$\mu' = \mu + e[j] \cdot \boldsymbol{D}, \qquad (3.2.18)$$

де $\boldsymbol{D} = (\boldsymbol{D}^{+} - \boldsymbol{D}^{-})$ – матриця змін. Тоді всі сформульовані раніше проблеми мережі Петрі легко інтерпретуються матричними рівняннями виду

$$\mu = \mu_0 + \sigma \cdot \boldsymbol{D}\,, \qquad (3.2.19)$$

де μ – досліджуване маркування, σ – вектор, компоненти якого показують скільки разів спрацьовує кожен перехід.

Хоча цей метод досить простий, він не позбавлений деяких недоліків. А саме, його застосування дає лише необхідні умови існування будь-якої властивості, іншими словами, може гарантувати лише його відсутність, а про присутність ми зможемо говорити з упевненістю, лише проаналізувавши дерево покриваності (зміни) маркувань.

Дерево маркувань мережі – це зв'язаний граф, у вершинах якого знаходяться маркування, яких ми досягли в результаті послідовного запуску дозволених переходів, а на дугах, що з'єднують вершини – переходи, що спускаються. Шлях від кореня до кожного маркування відбиває послідовність запусків, що призвела до неї. Коренем дерева є початкове маркування. При необмеженому накопиченні фішок в позиції на дереві утворюється петля, а в маркуванні на місці, що відповідає позиції, що зациклилася, ставиться ω - символ нескінченно великого числа.

Зрозуміло, що цей метод хоч і вимагає стомливого перебору всіх можливих маркувань мережі, зате вже готовому дереву досить легко аналізувати проблеми досяжності, покриваності, активності, оборотності мережі.

Описавши поведінкові властивості та методи аналізу, можна перейти безпосередньо до аналізу конкретної мережі Петрі.

3.3 Розрахунки та отримані результати

Вихідна мережа у вигляді графа:

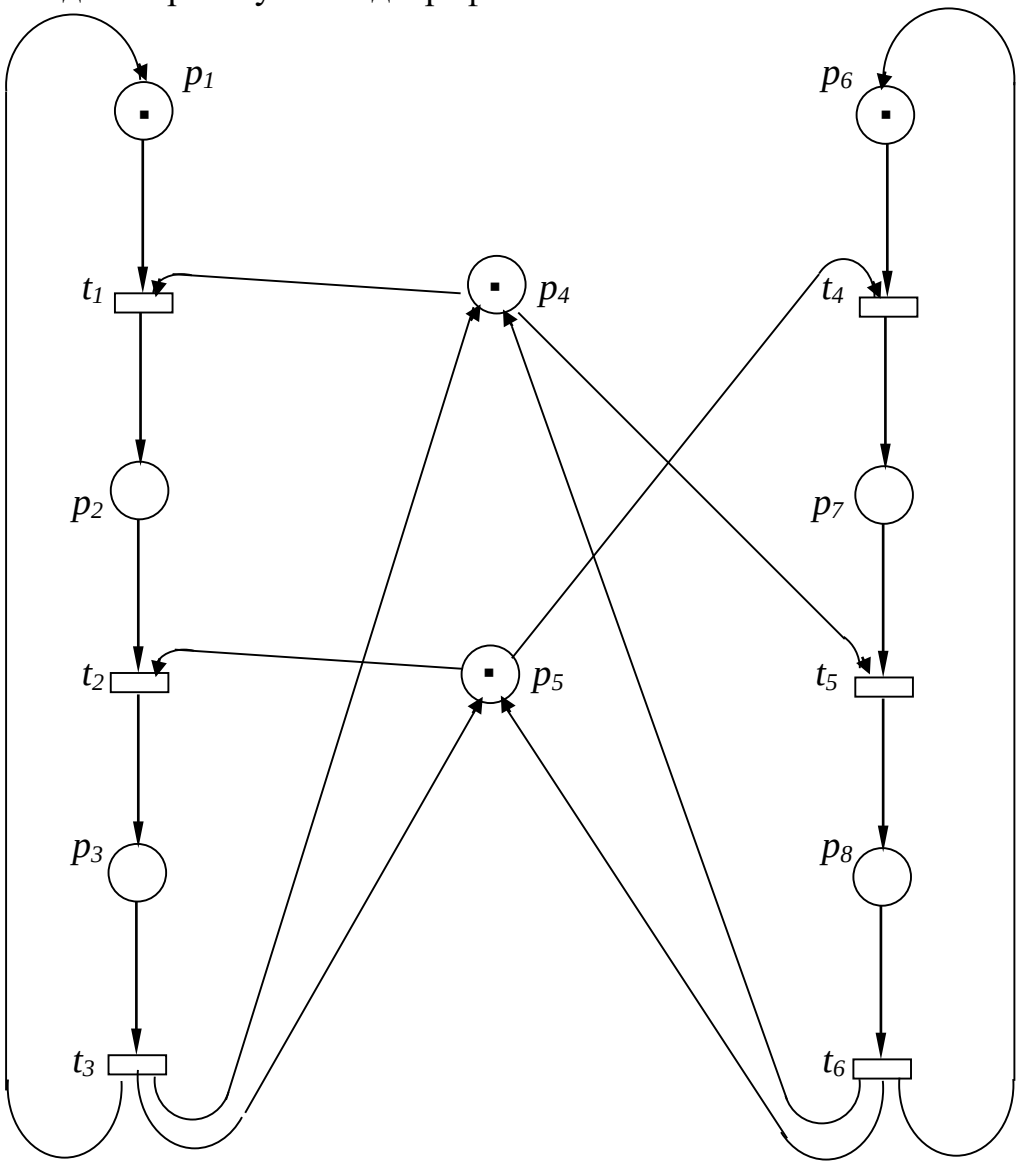

Рисунок 3.3.1 – Початкова мережа Петрі

Для матричного аналізу мережі знайдемо її матрицю змін.

$$D^{-} = \begin{vmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{vmatrix} \qquad (3.3.1)$$

$$D^{+} = \begin{vmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{vmatrix} \quad (3.3.2)$$

Матрицю змін знайдемо як різницю між (3.3.2) та (3.3.1):

$$D = \begin{vmatrix} -1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & -1 \end{vmatrix} \quad (3.3.3)$$

Таким чином, отримавши матрицю змін можна записати матричне рівняння зміни маркувань виду (3.2.19). Вектор початкового маркування визначимо так:

$$\mu_0 = (10011100) \quad (3.3.4)$$

Складемо дерево покриваності маркувань мережі.

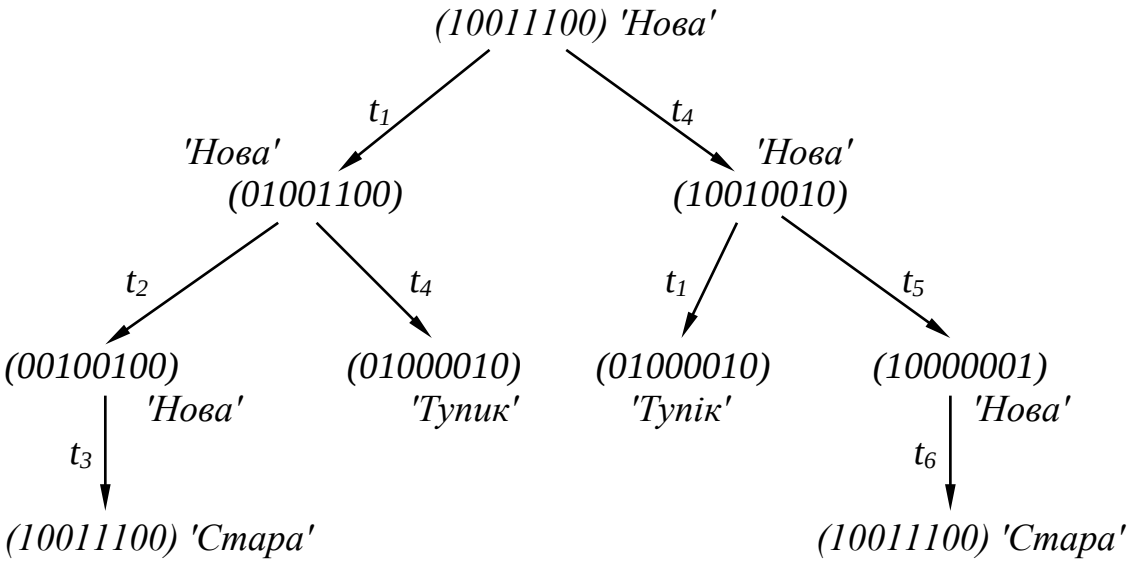

Рисунок 3.3.1 – Дерево покриваності маркувань

Дерево покриття зручно оформити у вигляді графа. При цьому більш наочно видно переходи, що зациклюються, тупикові маркування ніякими додатковими поясненнями постачати не потрібно - відсутність дуг, що виходять з даного маркування, говорить саме за себе. При досягненні старого маркування її не потрібно заново наносити на граф - достатньо з'єднати дугою попереднє маркування і вже існуюче "старе".

Граф покриття мережі виглядає наступним чином:

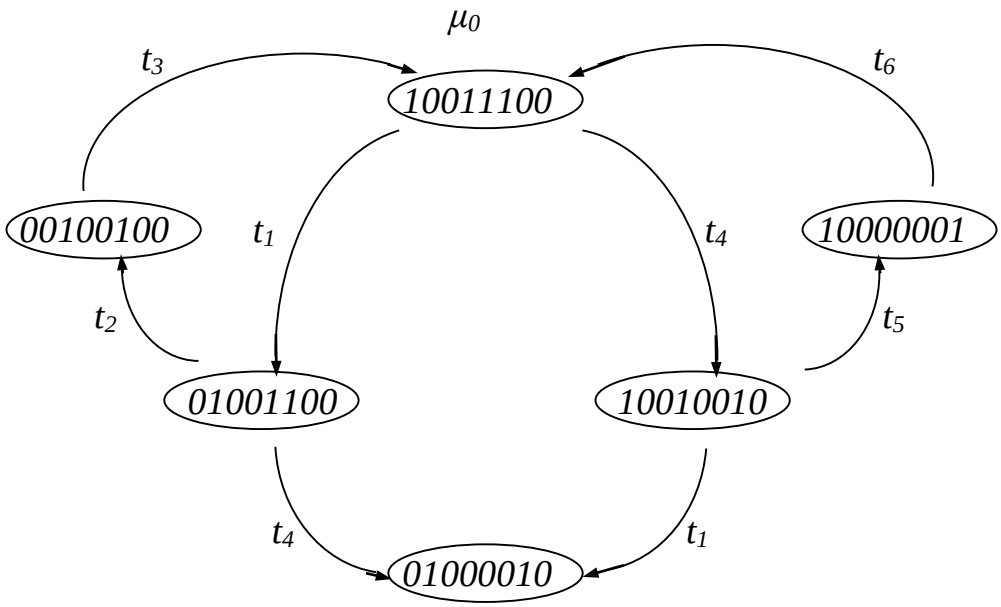

Рисунок 3.3.2 – Граф покриваності маркувань мережі Петрі

Проаналізуємо мережу двома методами – матричним та графічним та порівняємо отримані результати.

Питання досяжності будь-якої маркування найлегше вирішується, дивлячись на граф покриваності. Справді, візьмемо для прикладу два маркування: $\mu_1 = (01000010)$ та $\mu_2 = (00100010)$. Перша з них є досяжною, і можливі два шляхи приходу до неї: t_1 , t_4 або $t4$, $t1$. Однак вони не єдині, перед другим запуском переходу можливо нескінченну кількість разів запустити для першого випадку послідовність t_2 , t_3 для другого випадку – t_5 , t_6 . Друге маркування явно недосяжна, оскільки його немає на графі.

За допомогою матриць це питання вирішується так. Складаємо рівняння виду (3.2.19), в якому замість σ ставимо невідомий вектор x тієї ж розмірності,

а замість μ – цікаве для нас маркування μ_1. У результаті отримуємо систему з 8 рівнянь щодо 6 невідомих компонентів вектора x.

$$\begin{cases} -x_1 + x_3 = -1 \\ x_1 - x_2 = 1 \\ x_2 - x_3 = 0 \\ -x_1 + x_3 - x_5 = -1 \\ -x_2 + x_3 - x_4 + x_6 = -1 \\ -x_4 + x_6 = -1 \\ x_4 - x_5 = 1 \\ x_5 - x_6 = 0 \end{cases} \qquad (3.3.5)$$

Проаналізувавши цю систему, бачимо, що п'яте рівняння є наслідком із третього та шостого, шосте – з сьомого та восьмого, перше – з другого та третього. З (1) і (4) випливає, що $x_5 = 0$, $x_6 = 0$, з (7) випливає, що $x_4 = 1$. Перші три рівняння (3.3.5) є лінійно залежними, тому за вільне невідоме приймемо x_1. Тоді одержуємо рішення у вигляді $x_1 = \{ y\ y\ -1\ y\ -1\ 1\ 0\ 0 \}$, де y - будь-яке ціле число. Отримане рішення говорить про досяжність маркування μ_1 і вказує, які з переходів та скільки разів мають бути для цього запущені.

Порівнявши обидва способи вирішення, відразу можна побачити недоліки другого. По-перше, рішення (3.3.5) не вказує, у якій саме послідовності мають бути запущені зазначені переходи. По-друге, дивлячись на матрицю змін, ми можемо будувати висновки про наявність у мережі петель. Крім того, отримане матричне рішення не дає, взагалі кажучи, гарантій своєї реалізації – воно є лише необхідною умовою досяжності. Однак, не отримавши рішення, можна говорити про недосяжність маркування.

Дійсно, записавши (3.2.19) для μ_2 отримуємо систему:

$$\begin{cases} -x_1 + x_3 = -1 \\ x_1 - x_2 = 0 \\ x_2 - x_3 = 1 \\ -x_1 + x_3 - x_5 = -1 \\ -x_2 + x_3 - x_4 + x_6 = -1 \\ -x_4 + x_6 = -1 \\ x_4 - x_5 = 1 \\ x_5 - x_6 = 0 \end{cases} \qquad (3.3.6)$$

Система є несумісною, оскільки після віднімання третього рівняння з шостого отримуємо рівняння, що суперечить п'ятому. Тому можна зробити висновок про недосяжність μ_2 збігається з отриманим з графа покриваності маркувань.

Виходячи з графа (3.3.2), можна зробити висновок, що мережа є безпечною. Справді, в жодній з позицій на маркування не накопичується більше однієї фішки. Це говорить про те, що реальний процес, який описує мережа, протікає без конфліктів. Однак про повну відсутність конфліктів говорити поки що зарано. Цей висновок неможливо одержати з матричного рівняння, оскільки він є узагальненням, зробленим з урахуванням знання всіх можливих маркувань, які у мережі.

Ця мережа є активною – у ній кожен перехід може спрацювати хоча б один раз. Проаналізуємо рівні активності окремих переходів. Переходи t_1 і t_4 є *L1* - активними, оскільки вони у гіршому випадку (тобто при одержанні тупикового маркування) можуть спрацювати хоча б один раз. Переходи t_2, t_3, t_5 і t_6 є *L2* - активними, тому що вони можуть спрацювати будь-яке наперед задане число разів і навіть більше.

Звідси можна зробити висновок про те, що дана мережа не є безконфліктною – вона має тупиковий стан.

Можна також сказати, що мережа є оборотною. Цей висновок можна отримати і матричним шляхом – вирішивши рівняння

$$x \cdot \boldsymbol{D} = 0 \qquad (3.3.7)$$

Отримуємо систему

$$\begin{cases} -x_1 + x_3 = 0 \\ x_1 - x_2 = 0 \\ x_2 - x_3 = 0 \\ -x_1 + x_3 - x_5 = 0 \\ -x_2 + x_3 - x_4 + x_6 = 0 \\ -x_4 + x_6 = 0 \\ x_4 - x_5 = 0 \\ x_5 - x_6 = 0 \end{cases} \qquad (3.3.8)$$

Ця система має 2 рішення: *{ y y y 0 0 0}* і *{0 0 0 y y y }* , де *y* – будь-яке. Дійсно, запускаючи будь-яку кількість разів послідовності $t_1\ t_2\ t_3$ або $t_4\ t_5\ t_6$ щоразу ми повертаємося до вихідного маркування.

З графа (3.3.2) також випливає, що жодне з маркувань мережі не є покривається. Дійсно, для жодного маркування не існує іншого такого, для якого в кожній позиції було б не менше фішок, ніж у вихідній.

Можна сказати, що ця мережа не є стійкою. У неї є глухий кут, і, крім того, безпосередньо перед переходом у глухий стан завжди існують два дозволені переходи. Запускаючи 'неправильний' перехід, ми забороняємо обидва – і опиняємось у безвиході. Така властивість мережі свідчить про наявність потенційно можливих конфліктів.

На підставі графа (3.3.2) можна виписати безліч досяжних з μ_0 маркувань:

$$R(C,\mu_0)=\begin{Bmatrix}01001100\\10010010\\00100100\\10000001\\01000010\end{Bmatrix} \qquad (3.3.9)$$

Для моделювання мережі було написано програму мовою *Turbo Pascal* . Вона відображає стан мережі та дозволені у кожний момент переходи. Для вибору переходу, що запускається, використовується миша.

3.4 Висновки у розділі

У даному розділі була проаналізована і змодельована мережа Петрі, яка служить моделлю функціонування двох виробничих процесів, пов'язаних двома загальними ресурсами. У результаті можна зробити висновок про принципову наявність у системі тупикової ситуації, що виникає при спробі одночасного запуску обох процесів на виконання. Щоб не виникало глухого кута, необхідно кожен з процесів доводити до завершення, і не запускати інший процес, поки не закінчені всі три цикли першого. Все вищесказане повністю підтверджується написаною програмою, яка моделює всі описані ситуації, що виникають у мережі.

ВИСНОВОК

У роботі було розглянуто питання спрощення та синтезу дискретних двійкових пристроїв з 'пам'яттю' і без неї, а також проаналізовано мережу Петрі, що моделює конкретний виробничий процес та в кожному розділі зроблено відповідні висновки щодо самого процесу.

У першій частині була розглянута мінімізація булевих функцій, заданих у вигляді СДНФ, за допомогою двох різних способів: карт Карно та методу склеювання Квайна – МакКласкі. Отримані у вигляді мінімізованих ДНФ функції були приведені до базисів, що складаються лише з однієї функції: І – НЕ та АБО – НЕ, а потім реалізовані у вигляді комбінаційних схем на відповідних логічних елементах.

У другій частині був заданий за умовою у функціональному вигляді кінцевий автомат було мінімізовано за кількістю станів. Для отриманого автомата було збудовано граф станів. Потім, перейшовши до двійкового подання вхідних, вихідних сигналів і сигналів стану, в автоматі були виділені елементи пам'яті і комбінаційна частина, яка була мінімізована за кількістю змінних. Автомат був реалізований у базисі І – АБО – НЕ з використанням D – тригера та затримки.

У третій частині була проаналізована задана мережа Петрі за допомогою двох способів: матричного та заснованого на побудові дерева покриваності, а також написана програма для її моделювання.

ЛІТЕРАТУРА

1 Сигірський В.П. Математичний апарат інженера. - Київ: Техніка, 1975. -538 с.
2 Г.Корн, Т.Корн Довідник з математики для науковців та інженерів. - М .: Наука, 1984. -831 с.
3 В. Брауер Введення в теорію кінцевих автоматів. - М.: Радіо і зв'язок, 1987. -392 с.
4 Фаронов В.В. Турбо Паскаль 7.0: практика програмування - М.: Нолідж, 1997. -432 с.
5 Зайцев Д.А. Мережі Петрі і моделювання систем: Навчальний посібник, Одеса 2006
6 Зайцев Д.А. Математичні моделі дискретних систем: Навчальний посібник Одеса: ОНАЗ ім. О.С. Попова, 2004. - 40 с.
7 Математичні засади теорії телекомунікаційних систем / Підручник за загальну редакцію В.В. Поповського. - Харків, ТОВ "Компанія СМІТ", 2006. - 564 с.
8 Пітерсон Дж. Теорія мереж Петрі та моделювання систем. - М. Світ, 1984. - 264 с.
9 Ачасова С.М., Бандман О.Л. Коректність паралельних обчислювальних процесів. - Н.: Наука, 1990. - 253 с.
10 Слєпцов А.І., Юрасов А.А. Автоматизація проектування керуючих систем гнучких автоматизованих виробництв / Под ред. Б.Н.Малиновського. – К.: Техніка, 1986. – 160 с.
11 Мараховський В. Б., Розенблюм Л. Я., Яковлєв А. В. Моделювання паралельних процесів. Мережі Петрі. Курс для системних архітекторів, програмістів, системних аналітиків, проектувальників складних систем керування. - Санкт-Петербург: Професійна література, АйТі-Підготовка, 2014. - 400 с.
12 Норенков І. П., Кузьмік П. К. Інформаційна підтримка наукомістких виробів. М.: МДТУ ім. Н. Е. Баумана. 2002. - 347с.
13 Малиновського. - К.: Техніка, 1986. - 160 с.
14 Терано, Т.; Асаї, До.; Сугено, М. Прикладні нечіткі системи. М: Світ. 1993. - 368с.

ДОДАТКИ

Додаток А
Програма моделювання мережі Петрі

```
Program Farewell_Pascal_Please_Forgive_Me;
Uses graph,crt;
Const m_0=$9C;
      r_0=$90;
     path='cursor.dat';
 mask:array[0..5] of byte = ($90,$48,$20,$0C,$12,$01);
 jump:array[0..5] of word = ($406F,$20B7,$98DF,$02F3,$01ED,$1CFE);
Var
  i,j,counter,number:integer;
  flag_of_exit:boolean;
  ok:word;
  bm:integer;
  ScrMask:array[1..64] of byte;
  r,m,old_m,old_r:byte;
  f:file of byte;
  procedure Init_Graph_Mode;
  var
    Driver,
    Mode,
    ErrCode: Integer;
  begin
   Driver := Detect;
   InitGraph(Driver, Mode, '');

   ErrCode := GraphResult;
   if ErrCode <> grOk then
   begin
     Writeln('Помилка графічного режиму:',
                                      GraphErrorMSG(ErrCode));
     Halt(1);
  end;
  SetTextStyle(DefaultFont, HorizDir, 1);
  SetColor(15);
  SetLineStyle(0,0,1);
  SetFillStyle(1,0)
  end;
  function Init_Mouse:word;
  begin
    asm
      push ax
      mov ax,00h
      int 33h
      mov @Result,ax
      pop ax
    end
  end;
  procedure Show_Mouse;
  begin
    asm
      push ax
      mov ax,01h
      int 33h
      pop ax
    end
  end;
  procedure Hide_Mouse;
  begin
    asm
      push ax
      mov ax,02h
```

```
    int 33h
    pop ax
  end
end;
procedure Set_Graph_Cursor(segm,ofst:word;x,y:integer);
begin
  asm
    push ax
    push bx
    push cx
    push dx
    mov bx,x
    mov cx,y
    mov es,segm
    mov dx,ofst
    mov ax,09h
    int 33h
    pop dx
    pop cx
    pop bx
    pop ax
  end
end;
procedure Get_Mouse_State(var bt,x,y:integer);
begin
  asm
    push ax
    push bx
    push cx
    push dx
    mov ax,03h
    int 33h
    lds di,bt
    mov [di],bx
    lds di,x
    mov [di],cx
    lds di,y
    mov [di],dx
    pop dx
    pop cx
    pop bx
    pop ax
  end
end;
procedure Get_Web_State;
begin
  r := 0;
  for counter:= 0 to 5 do
    if (mask[counter] and m) = mask[counter] then
      r := r or ($80 shr counter)
end;
procedure Design_Kernel;
begin
  OutTextXY(190,20, 'Розподіл ресурсів для');
  OutTextXY(207,27, 'випадку двох процесів');
  for counter := 0 to 2 do
    Circle(150,counter*150+50,15);
  for counter := 0 to 2 do
    Circle(450,counter*150+50,15);
  for counter := 0 to 1 do
    Circle(300,counter*150+120,15);
  for counter := 0 to 2 do
    begin
      Line(140,counter*150+123,160,counter*150+123);
      Line(140,counter*150+127,160,counter*150+127);
      Line(140,counter*150+123,140,counter*150+127);
      Line(160,counter*150+123,160,counter*150+127)
```

```
    end;
  for counter := 0 to 2 do
    begin
      Line(440,counter*150+123,460,counter*150+123);
      Line(440,counter*150+127,460,counter*150+127);
      Line(440,counter*150+123,440,counter*150+127);
      Line(460,counter*150+123,460,counter*150+127)
    end;
  for counter := 0 to 1 do
    begin
      Line(counter*300+150,65,counter*300+150,123);
      Line(counter*300+150,127,counter*300+150,185);
      Line(counter*300+150,215,counter*300+150,273);
      Line(counter*300+150,277,counter*300+150,335);
      Line(counter*300+150,365,counter*300+150,423);
      Line(counter*300+150,123,counter*300+148,114);
      Line(counter*300+150,123,counter*300+152,114);
      Line(counter*300+150,185,counter*300+148,176);
      Line(counter*300+150,185,counter*300+152,176);
      Line(counter*300+150,273,counter*300+148,264);
      Line(counter*300+150,273,counter*300+152,264);
      Line(counter*300+150,335,counter*300+148,326);
      Line(counter*300+150,335,counter*300+152,326);
      Line(counter*300+150,423,counter*300+148,414);
      Line(counter*300+150,423,counter*300+152,414)
    end;
  Arc(120,427,180,360,25);Arc(480,427,180,360,25);
  Arc(122,35,0,180,27);Arc(478,35,0,180,27);
  Line(95,35,95,425);Line(505,35,505,425);
  Line(293,134,163,431);Arc(159,427,180,330,5);
  Line(290,281,170,436);Arc(162,427,180,320,12);
  Line(307,134,436,431);Arc(440,427,210,360,5);
  Line(310,281,429,436);Arc(438,427,220,360,12);
  Line(283,117,169,106);Arc(171,121,80,180,15);
  Line(312,129,439,262);Arc(429,273,0,45,15);
  Line(283,267,169,256);Arc(171,271,80,180,15);
  Line(311,257,426,110);Arc(432,121,0,160,12);
  Line(150,35,145,26);Line(150,35,150,26);
  Line(450,35,455,26);Line(450,35,450,26);
  Line(155,123,156,114);Line(155,123,159,115);
  Line(155,273,156,264);Line(155,273,159,265);
  Line(445,123,444,114);Line(445,123,440,115);
  Line(445,123,444,114);Line(445,123,441,116);
  Line(445,273,444,264);Line(445,273,440,265);
  Line(293,135,287,142);Line(293,135,291,143);
  Line(307,135,309,143);Line(307,135,312,142);
  Line(290,282,282,288);Line(290,282,285,290);
  Line(311,282,315,290);Line(311,282,317,288);
  SetFillStyle(1,8);
  for counter := 0 to 1 do
    begin
      Line(540,counter*70+150,600,counter*70+150);
      Line(540,counter*70+170,600,counter*70+170);
      Line(600,counter*70+150,600,counter*70+170);
      Line(540,counter*70+150,540,counter*70+170);
      FloodFill(570,counter*70+160,15)
    end;
  SetFillStyle(1,15);
  OutTextXY(543,159,'Restore');
  OutTextXY(555,229,'Exit');
end;
procedure Design_Mark_and_Jumps;
begin
    SetColor(15);
    SetLineStyle(0,0,3);
    SetFillStyle(1,15);
    Hide_Mouse;
```

```
    for counter := 0 to 2 do
      if ((m shr (7 - counter)) and 1) = 1 then
        begin
          SetColor(15);
          SetFillStyle(1,15);
          FillEllipse(150,counter*150+50,1,1)
        end
        else
        begin
          SetColor(0);
          SetFillStyle(1,0);
          FillEllipse(150,counter*150+50,1,1)
        end;
    for counter := 3 to 4 do
      if ((m shr (7 - counter)) and 1) = 1 then
        begin
          SetColor(15);
          SetFillStyle(1,15);
          FillEllipse(300,(counter-3)*150+120,1,1)
        end
        else
        begin
          SetColor(0);
          SetFillStyle(1,0);
          FillEllipse(300,(counter-3)*150+120,1,1)
        end;
    for counter := 5 to 7 do
      if ((m shr (7 - counter)) and 1) = 1 then
        begin
          SetColor(15);
          SetFillStyle(1,15);
          FillEllipse(450,(counter-5)*150+50,1,1)
        end
        else
        begin
          SetColor(0);
          SetFillStyle(1,0);
          FillEllipse(450,(counter-5)*150+50,1,1)
        end;
    for counter := 0 to 2 do
      if ((r shr (7 - counter)) and 1) = 1 then
        begin
          SetFillStyle(1,10);
          FloodFill(150,counter*150+125,15)
        end
        else
        begin
          SetFillStyle(1,12);
          FloodFill(150,counter*150+125,15)
        end;
    for counter := 3 to 5 do
      if ((r shr (7 - counter)) and 1) = 1 then
        begin
          SetFillStyle(1,10);
          FloodFill(450,(counter-3)*150+125,15)
        end
        else
        begin
          SetFillStyle(1,12);
          FloodFill(450,(counter-3)*150+125,15)
        end;
    SetColor(15);
    SetFillStyle(1,15);
    Show_Mouse
end;
Begin
  Init_Graph_Mode;
```

```
    ok := Init_Mouse;
    flag_of_exit := false;
    m := m_0;
    r := r_0;
    old_m := 0;
    old_r := 0;
    if ok = $FFFF then
      begin
{$I-}   assign(f,path);
        reset(f);
        ok := filesize(f);
{$I+}   if (IOResult = 0) and (ok = 64) then
          begin
            for i := 0 to 63 do
              read(f,ScrMask[i]);
            Set_Graph_Cursor(seg(ScrMask),ofs(ScrMask),2,2)
          end;
        Design_Kernel;
        Show_Mouse;
        repeat
          Get_Mouse_State(bm,i,j);
          if (m <> old_m) or (r <> old_r) then
            begin
              Get_Web_State;
              Design_Mark_and_Jumps;
              old_m := m;
              old_r := r
            end;
          if bm = 1 then
          begin
            number := 6;
            for counter := 0 to 2 do
              if (i < 165) and (i > 135) and
                 (j < counter*150+130) and (j > counter*150+120)
              then
                number := counter;
            for counter := 3 to 5 do
              if (i < 465) and (i > 435) and
                 (j < (counter-3)*150+130) and (j > (counter-3)*150+120)
              then
                number := counter;
            if (number < 6) and (((1 shl (7-number)) and r) <> 0) then
              begin
                m := m and (jump[number] and $FF);
                m := m or (jump[number] shr 8)
              end;
            if (i < 600) and (i > 540) and (j < 170) and (j > 150)
              then
                m := m_0;
            if (i < 600) and (i > 540) and (j < 240) and (j > 220)
              then
                flag_of_exit := true
          end;
        until flag_of_exit;
        Hide_Mouse;
        CloseGraph
      end
      else
        begin
          CloseGraph;
          WriteLn('Помилка миші: Device or driver not found.')
        end
  End.
```