

Міністерство освіти і науки України  
Державний заклад  
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

**Нікішин Дмитро Євгенович**

**РОЗРОБКА АЛГОРИТМУ СТВОРЕННЯ ПОЧАТКОВИХ ВХІДНИХ  
ПОСЛІДОВНОСТЕЙ ДЛЯ ЦИФРОВИХ СХЕМ**

**кваліфікаційна робота  
здобувача вищої освіти другого (магістерського) рівня  
освітньої програми «Мультимедійні системи»  
за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис \_\_\_\_\_ Дмитро НІКІШИН

Науковий керівник \_\_\_\_\_ Світлана ДОНЧЕНКО,  
асистент кафедри інформаційних  
технологій та систем

В.о. завідувача кафедри \_\_\_\_\_ Микола СЕМЕНОВ,  
кандидат педагогічних наук, доцент  
кафедри інформаційних технологій  
та систем

## **АНОТАЦІЯ**

**Нікішин Д. Є.**

**Тема:** Розробка алгоритму створення початкових вхідних послідовностей для цифрових схем.

**Спеціальність:** 121 «Інженерія програмного забезпечення».

**Установа:** ЛНУ імені Тараса Шевченка, 2024р.

**Магістерська робота містить:** 54 с., 11 рис., 16 табл., 30 джерел.

**Об'єкт дослідження** - цифрові пристрої.

**Предмет дослідження** - методи побудови ініціюючих послідовностей.

**Мета даної роботи** - розробка і реалізація нового алгоритму генерації ініціюючих вхідних послідовностей для цифрових схем.

**Результати роботи** – в роботі були вивчені і проаналізовані моделі цифрових пристроїв, основні методи моделювання і тестування цифрових пристроїв.

Розроблено та реалізовано алгоритм генерації ініціюючих вхідних послідовностей для цифрових схем. Представлено програмний додаток, що імплементує вищевказаний алгоритм.

**Ключові слова:** ЦИФРОВІ ПРИСТРОЇ, ІНІЦІАЛІЗУЮЧІ ПОСЛІДОВНОСТІ, ЦИФРОВІ СХЕМИ, МЕТОДИ МОДЕЛЮВАННЯ, АЛГОРИТМ, ПОБУДОВА ТЕСТІВ, АЛФАВІТ МОДЕЛЮВАННЯ, ЛОГІЧНІ ЕЛЕМЕНТИ.

## **ANNOTATION**

**Nikishyn Dmytro**

**Theme: Development of an algorithm for creating initial input sequences for digital circuits.**

**Speciality:** 121 "Software Engineering".

**Institution:** Luhansk Taras Shevchenko National University (LTSNU),  
2024 year.

**Master's work of:** 54 p., 11 im, 30 sources.

**A research object of:** - digital devices.

**The article of research-** methods of building initiating sequences.

**An aim of research is** - development and implementation of a new algorithm for generating initiating input sequences for digital circuits.

**Job performanes** - the work studied and analyzed the models of digital devices, the main methods of modeling and testing digital devices.

An algorithm for generating initiating input sequences for digital circuits has been developed and implemented. A software application implementing the above algorithm is presented.

**Keywords:** DIGITAL DEVICES, INITIALIZING SEQUENCES, DIGITAL CIRCUITS, SIMULATION METHODS, ALGORITHM, STRUCTURE OF TESTS, SIMULATION ALPHABET, LOGIC ELEMENTS.

## ЗМІСТ

<b>ВСТУП.....</b>	<b>5</b>
<b>РОЗДІЛ 1 АНАЛІЗ МОДЕЛЕЙ, МЕТОДІВ МОДЕЛЮВАННЯ ТА</b>	
<b>ТЕСТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ .....</b>	<b>7</b>
1.1. Математичні моделі цифрових пристроїв.....	7
1.2. Аналіз методів моделювання несправних ЦП.....	15
1.3. Аналіз методів побудови тестів .....	26
1.4. Висновки до розділу .....	31
<b>РОЗДІЛ 2 РЕАЛІЗАЦІЯ АЛГОРИТМУ ІНІЦІАЛІЗАЦІЇ ЦИФРОВИХ</b>	
<b>СХЕМ.....</b>	<b>32</b>
2.1. Вибір програмних засобів реалізації .....	32
2.2. Аналіз схеми.....	32
2.3. Формування варіантів вхідних послідовностей .....	39
2.4. Моделювання схеми .....	41
2.5. Виділення ініціюючих послідовностей .....	44
2.6. Тестування розробленої програми.....	46
2.7. Висновки до розділу .....	47
<b>ВИСНОВКИ .....</b>	<b>49</b>
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</b>	<b>51</b>
<b>ДОДАТОК А.....</b>	<b>54</b>

## ВСТУП

В останні роки широке поширення у всіх сферах життєдіяльності отримали комп'ютерні технології. Завдяки їм значно підвищилася ефективність сучасного виробництва, оперативність і якість обробки інформації. У зв'язку з цим зростають вимоги до надійності комп'ютерних систем, що пов'язано з відповідальністю покладених на них функцій. Підвищення якості і надійності обчислювальної техніки досягається впершу чергу застосуванням сучасних автоматизованих систем проектування, найважливішими компонентами яких є засоби логічного моделювання і побудови перевіряючих тестів.

В даний час існує ряд ефективних алгоритмів моделювання та генерації тестів для комбінаційних схем. Однак у зв'язку з різким стрибком складності проєктованих цифрових пристроїв (ЦП) застосування вже існуючих підходів істотно збільшує вартість процесу діагностування, вимагаючи надмірних витрат часових ресурсів і пам'яті. Завдання розробки алгоритмів моделювання та побудови тестів для послідовних схем досліджено значно гірше внаслідок невизначеності початкового стану і явища змагань сигналів. В даний час не існує ефективних методів генерації тестів для складних цифрових схем з пам'яттю.

У зв'язку з вищесказаним необхідна подальша розробка ефективних алгоритмів логічного моделювання і автоматизованої генерації тестів, як для комбінаційних, так і для послідовних цифрових схем.

Дані фактори і зумовили вибір теми дослідження: «Розробка алгоритму генерації ініціюючих вхідних послідовностей для цифрових схем».

**Об'єкт дослідження** - цифрові пристрої.

**Предмет дослідження** - методи побудови ініціюючих послідовностей.

**Мета роботи** - розробка і реалізація нового алгоритму генерації ініціюючих вхідних послідовностей для цифрових схем.

В ході виконання роботи буде розроблено програмний додаток, що реалізує даний алгоритм.

**Для досягнення поставленої мети необхідно вирішити наступні завдання:**

1. Вивчити і проаналізувати математичні моделі цифрових пристроїв, основні методи моделювання і тестування цифрових пристроїв.
2. Розробити і реалізувати алгоритм генерації ініціюючих послідовностей для цифрових схем.
3. Протестувати розроблений програмний додаток і проаналізувати отримані результати.

**Методи дослідження** - техніко-економічний з використанням комп'ютерних технологій, технічний аналіз, методи моделювання інформаційних процесів.

Наукова новизна даної роботи полягає в використанні нового імовірнісного підходу при побудові ініціюючих послідовностей.

Практичною цінністю роботи є розроблюваний додаток, який реалізує імовірнісний підхід для отримання ініціюючих вхідних послідовностей, використовуваних при побудові тестів.

У першому розділі здійснюється аналіз моделей, методів моделювання і тестування цифрових пристроїв. У розділі розглянуті рівні уявлення цифрових пристроїв, їх математичні моделі, методи моделювання несправних цифрових пристроїв, алфавіти, які використовуються для логічного моделювання, і сучасні методи побудови тестів для цифрових пристроїв.

Другий розділ присвячений розробці та реалізації нового алгоритму генерації ініціюючих вхідних послідовностей для цифрових схем. В цьому розділі послідовно описується розроблений алгоритм, наводяться приклади його виконання, аргументується вибір мови програмування і середовища розробки програмного забезпечення, описуються реалізовані процедури і функції, наводяться результати тестування розробленої програми і проводиться їх аналіз.

# **РОЗДІЛ 1**

## **АНАЛІЗ МОДЕЛЕЙ, МЕТОДІВ МОДЕЛЮВАННЯ ТА ТЕСТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ**

### **1.1. Математичні моделі цифрових пристроїв**

У моделюванні цифрових пристроїв зазвичай виділяють п'ять рівнів уявлення: моделювання на рівні напівпровідникових приладів і рівні електронних схем (рівень транзисторів, резисторів і т.д.), моделювання на рівні логічних елементів, моделювання на рівні регістрових передач і імітаційне моделювання на системному рівні.

На системному рівні, що містить найменшу кількість деталей, моделюється архітектура системи. Тут не обчислюються значення сигналів в схемі, але проводиться моделювання потоків даних в системі для визначення "вузьких місць" і обчислення пропускну здатності. Цей рівень використовується для вивчення продуктивності комп'ютерів, комунікаційних мереж і т.д.

Моделювання на рівні регістрових передач, званому також функціональним моделюванням, може розглядатися як найвищий рівень логічного моделювання. Тут модельована схема розглядається як з'єднання деяких функціональних блоків, чия поведінка описується на мовах опису апаратури, зокрема мовами регістрових передач. Основна мета моделювання на даному рівні - верифікація розробки на високому рівні перед її деталізацією. Функціональне моделювання застосовується також для аналізу несправностей, для яких неможливо реалізувати моделювання на більш високому рівні деталізації. Якщо розмір схеми дуже великий, то моделювання на функціональному рівні є хорошою альтернативою вентильному рівню моделювання.

Моделювання на вентильному рівні найбільш широко використовуваний інструмент, як при моделюванні справних ЦП, так і ЦП з

несправностями. На цьому рівні деталізації можливо моделювати не тільки логічну поведінку схеми, а також її деякі тимчасові характеристики.

Нижче вентильного рівня деталізації можна виділити ще два рівня, використовуваних при моделюванні. Це комутаційний і схемний рівні. На обох цих рівнях пристрій моделюється як з'єднання транзисторів, резисторів і конденсаторів. На комутаційному рівні моделювання здійснюється обробкою транзисторів як перемикачів, що володіють деяким набором характеристик, що описуються в спеціальних таблицях. На схемному рівні поведінка пристрою визначається рішенням схемних рівнянь. Комутаційний рівень застосовується для моделювання схем, які містять до декількох тисяч вентилів, тоді як схемний рівень обмежується пристроями з декількома сотнями елементів. Таким чином, обидва цих рівня деталізації придатні для отримання тимчасових діаграм, що показують поведінку елементарних осередків, що реалізують певні нескладні функції, які використовуються далі всередині великих схем.

Іноді, через неможливість реалізувати моделювання на заданому рівні деталізації доводиться використовувати стратегію змішаного за рівнями моделювання. Хоча можливо моделювати ЦУ спільно на функціональному, вентильному і схемному рівнях деталізації, найбільше застосування отримало змішане моделювання на функціональному і вентильному рівнях, а також на вентильному і схемному рівнях. При змішаному моделюванні частини схеми, які необхідно аналізувати, моделюються на рівні з більшим ступенем деталізації, тоді як їх оточення моделюється з меншим ступенем деталізації. Наприклад, якщо несправність можлива в одній частині великої схеми, то ця частина моделюється на вентильному рівні, а частина, що залишилася моделюється на функціональному рівні. При цьому функціональне моделювання використовується для поширення сигналів від зовнішніх входів схеми до частини, описаної на вентильному рівні, і далі від неї до зовнішніх виходів схеми для визначення вихідних значень сигналів.



Моделювання поведінки цифрових пристроїв вимагає побудови моделей цих пристроїв. Ці моделі є абстракцією реальної схеми, яка представляє якомога більше інформації, необхідної для моделювання на обраному рівні деталізації.

На даному етапі, як об'єкт дослідження приймається дискретний пристрій (ДП) - пристрій, призначений для обробки інформації, представленої в дискретній формі. Розрізняють два класи ДП: комбінаційні і послідовні дискретні пристрої.

Якщо значення сигналів в поточний момент часу однозначно визначають значення на вході пристрою, то такий ДП називається комбінаційним (ДП без пам'яті.).

Дискретний пристрій називається послідовним (ДП з пам'яттю), якщо значення сигналів на його виходах визначаються не тільки сигналами на входах в поточний момент часу, але і внутрішнім станом схеми в попередній момент часу.

Розглянемо два підходи побудови моделей ДП: функціональний і структурний.

На функціональному рівні моделювання внутрішня структура пристрою не враховується, а розглядається лише логіка його функціонування. Оскільки ми під ДП розуміємо пристрій для обробки двійкової інформації, то в якості математичної моделі вибирається система булевих функцій. Для комбінаційних пристроїв (ДП без пам'яті) вона виглядає наступним чином:

$$y_1 = f_1(x_1, x_2, \dots, x_n);$$

.

.

$$y_m = f_m(x_1, x_2, \dots, x_n);$$

де  $X = (x_1, x_2, \dots, x_n)$  - вхідні і  $Y = (y_1, y_2, \dots, y_m)$  - вихідні змінні.

Для послідовних ДП в якості математичної моделі вибирається абстрактний кінцевий автомат:  $A = \{S, X, Y, \delta, \lambda\}$ , де  $S$  - кінцева множина станів,

$X$  - вхідний алфавіт,  $Y$  - вихідний алфавіт,  $\delta: S \times X \rightarrow S$  - функція переходу і  $\lambda: S \times X \rightarrow Y$  - функція виходу. Автоматна модель ДП дозволяє уявити послідовний пристрій у вигляді комбінаційного блоку і блоку пам'яті, з'єднаних лініями зворотного зв'язку  $S=(s_1, s_2, \dots, s_k)$  (рис.1.1). При цьому система булевих рівнянь вище ускладнюється і має вигляд:

$$y_1=f_1(x_1,\dots,x_n,s_1,\dots,s_k); \dots; y_m=f_m(x_1,\dots,x_n,s_1,\dots,s_k);$$

$$s_1=g_1(x_1,\dots,x_n,s_1,\dots,s_k);\dots; s_k=g_k(x_1,\dots,x_n,s_1,\dots,s_k).$$



Рис. 1.1. Структура послідовної схеми

Як зазначалося вище, ступінь деталізації на функціональному рівні не дозволяє вирішувати всі завдання, пов'язані з моделюванням роботи ДП. Тому в якості основної вибирається модель ДП вентильного рівня. При цьому ДП описується в вигляді логічної мережі (схеми), вершинами якої є логічні елементи, входи, виходи і вузли розгалужень. Мережа називається правильною, якщо в ній виходи ніяких двох елементів не пов'язані один з одним, а функція, що реалізується схемою, може бути представлена як функція виходу кінцевого автомата.

Для позначення фізичних сигналів, що поширюються по схемі, при логічному моделюванні використовується алфавіт моделювання. У процесі моделювання вхідні, вихідні та змінні стани можуть приймати тільки значення з алфавіту моделювання. Оскільки дискретний пристрій працює з інформацією, представленою в двійковому вигляді, тільки кілька логічних значень сигналів необхідно для подання фізичних сигналів.

Мінімальним алфавітом для моделювання роботи ДП є двійковий алфавіт  $B_2=\{0,1\}$ , в якому, як правило, 0 відповідає низькому рівню сигналу, а 1 - високому.

Для подання невизначених початкових станів і сигналів, чиє значення неможливо визначити через конфлікти сигналів в схемі або генерації, використовується невідоме значення  $u$ . При цьому отримуємо тризначний алфавіт  $E_3=\{0,1, u\}$ . Однак символів цього алфавіту стає недостатньо, коли необхідно моделювати змагання в схемі, пов'язані з невизначеністю початкового стану. Тому на практиці широко застосовуються алфавіти більшої значності.

Природним розширенням алфавіту  $E_3$  є п'ятизначний алфавіт  $E_5=\{0, 1, u, E, H\}$ , де 0 і 1 позначають статичні значення 0 і 1,  $u$  - невизначене значення,  $E$  - зміна сигналу  $0 \rightarrow 1$ , а  $H$  - зміна  $1 \rightarrow 0$ . На практиці використовуються також алфавіти більшої значності, в яких додаткові символи представляють статичні і динамічні змагання.

Процес моделювання роботи ДП зводиться до ітеративного моделювання логічних елементів (вентилів), що реалізують деяку логічну функцію. Алфавіт моделювання при цьому відіграє роль базису, на якому задаються функції. Основними функціями при моделюванні є: логічні І, АБО, НЕ, ЯКИЙ ВИКЛЮЧАЄ-АБО, а також їх комбінації. Найпростішою формою завдання логічних функцій є таблиця істинності. Функції основних вентилів для 3-значного алфавіту представлені в табл.1.1-1.3.

При програмній реалізації символи алфавіту зазвичай кодуються цілими числами, а таблиці - двовимірними масивами. Тоді обчислення багатозначної функції полягає в знаходженні індексів і вибірці потрібного елемента масиву.

Таблиця 1.1

**Завдання вентилі І в 3-значному алфавіті.**

<b>И</b>	<b>0</b>	<b>1</b>	<b>u</b>
----------	----------	----------	----------

<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>1</b>	<b>0</b>	<b>1</b>	<b>u</b>
<b>u</b>	<b>0</b>	<b>u</b>	<b>u</b>

Таблиця 1.2

### Завдання вентиля АБО в 3-значному алфавіті.

<b>АБО</b>	<b>0</b>	<b>1</b>	<b>u</b>
<b>0</b>	<b>0</b>	<b>1</b>	<b>u</b>
<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>u</b>	<b>u</b>	<b>1</b>	<b>u</b>

Таблиця 1.3

### Завдання вентиля НЕ в 3-значному алфавіті.

<b>НЕ</b>	<b>0</b>	<b>1</b>	<b>u</b>
	<b>1</b>	<b>0</b>	<b>u</b>

Ще одним поширеним методом побудови моделей функцій є складання підпрограм. Безпосередньо до логічного виразу функції записуються оператори мови, побудовані за допомогою операцій кон'юнкції, диз'юнкції і заперечення. Таким чином, підпрограма обчислення логічного вентиля буде складатися з декількох логічних операторів і операторів пересилання. Даний метод широко застосовується при паралельному методі моделювання.

Найбільшого поширення при моделюванні ДП отримав 3-значний алфавіт ЕЗ. Основне призначення моделювання в алфавіті ЕЗ - аналіз конфліктів. Конфлікти в реальних ДП виникають внаслідок того, що зміна вхідних сигналів відбувається не одночасно, а також через розкид затримок елементів. Результатом конфліктів може бути неправильне функціонування ДП. Для запобігання виникненню конфліктів використовують протигоничне кодування, вводять спеціальні затримки і т.д.

Для аналізу конфліктів застосовується метод Ейхельбергера. Він ґрунтується на припущенні про довільність затримок в елементах і лініях

зв'язку. Для позначення переходів сигналів  $0 \rightarrow 1$  і  $1 \rightarrow 0$  використовується символ  $\delta$ .

Моделювання вхідного набору  $X_t$  проводиться в два етапи. Спочатку формується проміжний набір  $\delta X_t$ , в якому всім змінним, що змінив значення в порівнянні з набором  $X_{t-1}$ , присвоюється значення  $\delta$ . Решта вхідних змінних залишаються без змін. Після цього проводиться моделювання на отриманому проміжному наборі  $\delta X_t$ . Таким чином, проводиться моделювання перехідного процесу. Всім лініям, які істотно залежать від змінених входів, присвоюється невизначене значення  $\delta$ . На другому етапі проводиться моделювання основного набору  $X_t$ . Всі входи схеми отримують певні значення (0 або 1) і відбувається "зняття" невизначеності. Якщо після закінчення цього етапу на деяких лініях в схемі залишилися невизначені значення  $\delta$ , то це говорить про можливість конфліктів на даних лініях. Якщо всі лінії схеми отримують певні значення, то на модельованому наборі конфлікти неможливі.

Даний метод моделювання конфліктів отримав широке застосування, оскільки аналіз виконується чисто логічними методами без додаткового завдання значень затримок елементів. Однак цей метод може показувати наявність помилкових конфліктів. Це як раз і обумовлено припущенням про довільність затримок елементів і ліній зв'язку. У реальних схемах затримки ліній зв'язку істотно менші затримок елементів. Також не довільне співвідношення затримок різних типів елементів. Помилкові конфлікти виявляються зазвичай в схемах, що містять розгалуження, штучні лінії затримки з кількох елементів і рахункових структурах, коли будь-яке перемикання рахункового входу викликає виявлення конфліктів.

Для аналізу конфліктів також використовуються алфавіти більшої значності. Для виявлення статичних конфліктів використовується алфавіт  $H_6$ . Цей алфавіт є підмножиною  $H_6 \subseteq B^2 \times E^3 \times B^2$ . Для виявлення як статичних, так і динамічних конфліктів використовується 8-значний алфавіт  $H_8$ , що є

підмножиною  $H8 \subseteq B2 \times E5 \times B2$ . Алфавіти  $H6$ ,  $H8$  і їх фізична інтерпретація наведені в табл.1.4 і 1.5 відповідно.

Таблиця 1.4

**Алфавіт  $H6$  і його інтерпретація**

<b>Елемент</b>	<b>Фізична інтерпретація</b>
<b><math>0 \times 0 \times 0</math></b>	<b>статичний 0</b>
<b><math>1 \times 1 \times 1</math></b>	<b>статична 1</b>
<b><math>0 \times u \times 1</math></b>	<b>перехід <math>0 \rightarrow 1</math></b>
<b><math>1 \times u \times 0</math></b>	<b>перехід <math>1 \rightarrow 0</math></b>
<b><math>0 \times u \times 0</math></b>	<b>статичне 0- змагання</b>
<b><math>1 \times u \times 1</math></b>	<b>статичне 1- змагання</b>

На практиці моделювання з аналізом конфліктів проводиться на трьох наборах. Звідси можна простежити математичну структуру даних алфавітів: спочатку виконується моделювання на поточному наборі в двійковому алфавіті  $B2$ , потім на проміжному наборі в алфавіті  $E3$  (для  $H8$  в алфавіті  $E5$ ), і наступному основному знову в алфавіті  $B2$ . При аналізі конфліктів також використовується алфавіт  $H9$ , що містить всі символи алфавіту  $H8$  і додатково символ  $u \times u \times u$ , який відповідає невизначеному стану.

Таблиця 1.5

**Алфавіт  $H8$  і його інтерпретація.**

<b>Елемент</b>	<b>Фізична інтерпретація</b>
<b><math>0 \times 0 \times 0</math></b>	<b>статичний 0</b>
<b><math>1 \times 1 \times 1</math></b>	<b>статична 1</b>
<b><math>0 \times E \times 1</math></b>	<b>перехід <math>0 \rightarrow 1</math></b>

<b>1×N×0</b>	<b>перехід 1→0</b>
<b>0×u×0</b>	<b>статичне 0- змагання</b>
<b>1×u×1</b>	<b>статичне 1- змагання</b>
<b>0×N×0</b>	<b>динамічне 0- змагання</b>
<b>1×E×1</b>	<b>динамічне 1- змагання</b>

## **1.2. Аналіз методів моделювання несправних ЦП**

Логічне моделювання є важливою частиною будь-якої системи моделювання цифрових пристроїв. Це особливо стає очевидним при розробці високо інтегрованих схем.

Розрізняють два принципових додатки логічного моделювання: моделювання справних ЦП і моделювання ЦП з несправностями. Моделювання справних ЦП (верифікація розробки) використовується для визначення поведінки справної схеми. Найбільш часто воно полягає у визначенні вихідної послідовності, яка породжується заздалегідь визначеною вхідною послідовністю. Справне моделювання також використовується для отримання точних часових діаграм для зовнішніх виходів схеми або деяких внутрішніх точок, ґрунтуючись на певних затримках для елементів схеми. Воно також може використовуватися для визначення змагань в схемі на даній вхідній послідовності. Загалом, моделювання справної схеми використовується для підвищення ступеня відповідності між реалізацією схеми і її прототипом. Додатково справне моделювання використовується для отримання даних в алгоритмах тестування. У цьому випадку в процесі моделювання спостерігаються значення сигналів в певних точках схеми. Вони запам'ятовуються в тестуючому пристрої і порівнюються зі значеннями в цих же точках під час тестування для визначення помилкової поведінки схеми.

У загальному випадку алгоритм моделювання можна представити в наступному вигляді. Після запису вхідних впливів на зовнішні входи схеми послідовно від входів схеми до виходів обчислюються нові значення

елементів. Така процедура називається ітерацією. В результаті такого проходу значення сигналів на деяких виходах елементів можуть змінитися. У цьому випадку необхідно провести наступну ітерацію. Процес ітерації повторюється до тих пір, поки всі сигнали не приймуть встановлені значення. У цьому випадку говорять, що процес моделювання на даному вхідному наборі сходиться. Однак можливий випадок генерації схеми - значення сигналів на деяких лініях періодично змінюються. Тому необхідно вибрати критерій завершення процесу моделювання. Зазвичай таких критеріїв два:

- 1) збіг значень сигналів на двох послідовних ітераціях;
- 2) досягнення деякого граничного числа ітерацій.

Порядок обробки елементів всередині однієї ітерації визначається використанням алгоритму. Розрізняють методи наскрізного і подієвого моделювання однієї ітерації.

Наскрізним називається метод моделювання, при якому для поточного вхідного набору обчислюються в певній послідовності значення сигналів для всіх елементів схеми. Такий спосіб обробки елементів призводить до зайвих обчислень для елементів, чиї сигнали на входах не змінилися.

При подієвому методі моделювання обробляються тільки ті елементи, значення вхідних сигналів яких змінилися і тільки в ті моменти часу, коли ці зміни відбуваються. Дана умова особливо істотна для моделювання асинхронних схем з різними затримками елементів. Очевидно, що швидкодія алгоритму подієвого моделювання вище, оскільки в кожен момент часу активні, тобто мають зміни на входах, тільки кілька відсотків логічних елементів.

Моделювання з несправностями (несправне моделювання) визначає поведінку схеми в присутності певної несправності з заздалегідь заданої множини. Строго визначити поняття несправності дуже складно. Ми під несправністю будемо розуміти деякий фізичний дефект одного або декількох компонентів ДП, який може викликати його неправильне функціонування. У роботі буде використовуватися клас правильних несправностей - це



несправності, за наявності яких несправність приладу знаходиться в тому ж класі пристроїв, що і справне. Серед правильних несправностей найбільш широке застосування знайшли логічні несправності, при яких може змінюватися логіка функціонування елементів, але не структура самого пристрою. До логічних несправностей відносяться несправності компонентів ДП і несправності зв'язків, які зводяться до зміни функцій, реалізованих компонентами. Найбільш широко використовуваним класом логічних несправностей є константні несправності. Константні несправності еквівалентні подачі на вхід або вихід елемента схеми постійного сигналу низького рівня (несправність  $\text{const0}$ ) або постійного сигналу високого рівня (несправність  $\text{const1}$ ). Якщо в схемі в один момент часу присутня тільки одна несправність, то говорять про одиночні несправності. Якщо ж число одночасно присутніх в ДП дефектів більше одного, то говорять про кратні несправності. Поодинокі несправності еквівалентні зміні функції тільки одного елемента, наприклад, це відповідає наявності константної несправності на одній лінії логічної схеми. Кратна несправність є довільним, але маючим фізичний сенс, поєднанням поодиноких несправностей.

Через велику кількість потенційних несправностей в будь-якій великій схемі і складності алгоритмів моделювання зазвичай передбачається, що в один момент часу в схемі може бути присутнім тільки одна несправність. У роботі в якості базового класу розглядаються поодинокі константні несправності. Моделювання роботи несправної схеми може бути виконано будь-яким методом моделювання справної схеми шляхом внесення несправності в процесі моделювання. Якщо в схемі можливо  $N$  різних несправностей, то час моделювання може зрости в  $N$  раз. Таким чином, при моделюванні ДП з несправностями основним критерієм є швидкодія алгоритмів. Для зменшення часу моделювання ДП зазвичай застосовують скорочення вихідної множини модельованих несправностей шляхом виявлення еквівалентних несправностей.

Одне із застосувань моделювання ДП з несправностями - визначення повноти заданої тестової послідовності. Повнота тестової послідовності зазвичай визначається як відношення числа несправностей, які виявляються послідовністю,  $m$  до загальної кількості модельованих несправностей  $N$  і виражається у відсотках:

$$P = \frac{m}{N} \cdot 100\%$$

Якщо необхідно, то отримана при моделюванні з несправностями інформація може бути використана для поліпшення якості тестової послідовності. Моделювання ДП з несправностями також використовується для генерації словників сигнатур для різних несправностей з метою визначення місця розташування несправностей. При моделюванні з несправностями виходять набори несправних значень, які порівнюються з сигнатурами несправностей в базі даних для визначення найбільшої відповідності, тобто визначення найбільш вірогідної несправності.

В даний час використовуються кілька основних алгоритмів моделювання несправних ДП, які розрізняються технікою поширення несправностей. Найбільше застосування отримав метод паралельного за несправностями моделювання. Метод ґрунтується на тому, що логічні операції мікропроцесор виконує одночасно над усіма розрядами машинного слова. Якщо інструментальна ЕОМ має  $P$ -розрядне машинне слово, то одночасно моделюється робота  $P$  схем: одна справна і  $P-1$  несправних схем. Таким чином, весь список несправностей розбивається на групи по  $P-1$  несправностей і моделювання повторюється  $\left\lceil \frac{N}{P-1} \right\rceil$  раз. При використанні алфавіту моделювання зі значністю більше двох, коли кожен елемент алфавіту кодується за допомогою декількох машинних слів, встає проблема ефективного кодування алфавіту.

При моделюванні несправних ДП необхідно додатково розробити програмні методи внесення і поширення несправностей. Розглянемо приклад моделювання двухвходового логічного елемента АБО (рис.1.2.). Нехай входи

А і В вентиля містять відповідно машинні слова:  $A=0000$  і  $B=1111$ , а на його лініях моделюються такі несправності:

для входу А -  $H_1=const1$ ;

для входу В -  $H_2=const0$ ;

для виходу С -  $H_3=const0$ ;

У машинному слові несправності розташуються таким чином: в першому справа біті моделюється робота справної схеми, в другому розряді - несправність  $H_1$ , в третьому  $H_2$ , в четвертому - несправність  $H_3$ . Для поширення несправностей через елемент необхідно перед його моделюванням внести несправності входів. Для цього в другий розряд для лінії А необхідно записати одиницю (для несправності  $H_1$ ), а в третій розряд для лінії В - нуль для несправності  $H_2$ . Тоді машинні слова будуть містити:

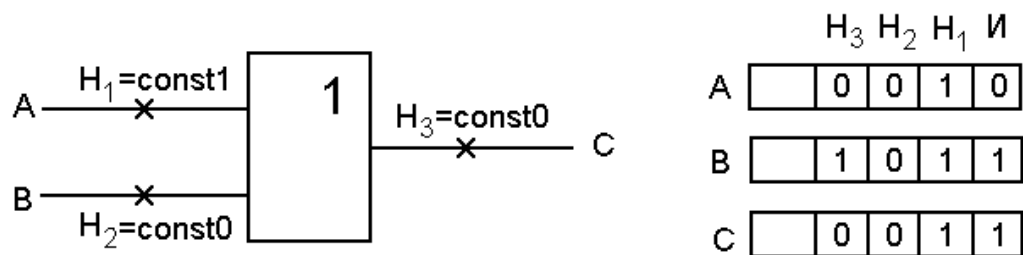


Рис. 1.2. Паралельне моделювання несправностей для вентиля АБО

$A=0010$ ;  $B=1011$ . Після цього необхідно промоделювати елемент - виконати порозрядну операцію  $A|B$ . Лінія С отримає наступні значення:  $C=1011$ . Тепер необхідно внести в четвертий біт одиницю для несправності  $H_3$ . Таким чином, після моделювання отримаємо  $C=0011$ , тобто через елемент поширилися несправності  $H_2$  і  $H_3$ .

З прикладу видно, що при паралельному методі моделювання функціонування логічних елементів має бути виражене за допомогою булевих функцій, оскільки основний виграш у часі виходить при виконанні логічних операцій, які, як зазначено вище, в сучасних ЕОМ виконуються одночасно над усіма розрядами машинного слова.

Внесення несправності в даному методі моделювання здійснюється зазвичай за допомогою масок. При цьому перед моделюванням елемента необхідно перевірити: чи є в поточній групі несправності його входів; якщо так - то підготувати бітові маски і внести вплив цих несправностей. Після моделювання елемента необхідно перевірити: чи є в поточній пачці несправності виходу елемента; якщо так - то підготувати бітові маски і внести вплив несправностей. Видно, що перевірка наявності несправностей на виходах елемента відбувається всякий раз при його обчисленні, навіть в тому випадку, якщо в поточній групі немає несправностей на лініях даного логічного елемента. Якщо врахувати, що в кожній групі моделюється тільки 32 несправності (32 - розрядність інструментальної ЕОМ), то видно, що такий спосіб внесення несправностей веде до надмірних витрат часу при моделюванні схеми.

Щоб уникнути вказаного недоліку, був запропонований метод внесення несправностей шляхом введення в схему фіктивних елементів. Для внесення на лінію схеми несправності `const0` на цю лінію вставляється вентиль АБО. Рис.1.3. показує внесення несправності `const1` в третій біт на лінії А. Спочатку всі елементи-послідовники лінії А стають послідовниками додаткового вентиля АБО. Потім лінія А з'єднується з одним із входів нового вентиля. Інший вхід вентиля АБО - фіктивна лінія, значення сигналу для якої містить нулі у всіх бітових позиціях за винятком позиції для внесення несправності. Таким чином, після обчислення додаткового вентиля біт в третій позиції примусово буде встановлений в одиницю.

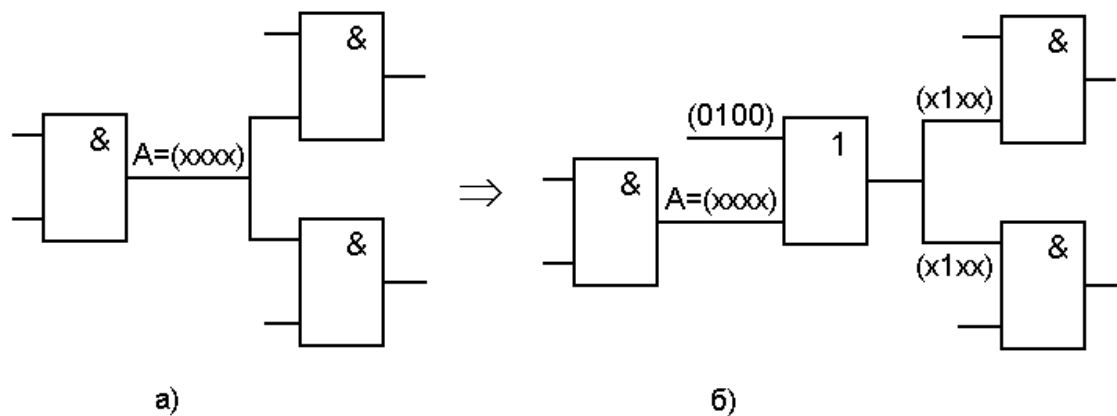


Рис. 1.3. Внесення несправності const1

- а) до внесення несправності (х - довільне значення);  
 б) на лінію А в третій біт внесена несправність const1.

Аналогічно, для внесення на лінію схеми несправності const0 використовується додатковий вентиль І з таким фіктивним значенням: всі бітові позиції містять одиниці, а несправна позиція містить нуль.

Даний спосіб внесення несправностей дозволяє уникнути багаторазових перевірок наявності несправностей на лініях вентиля при кожному його моделюванні і істотно скоротити час моделювання схеми в цілому. Недоліком даного методу є те, що для його реалізації необхідно перед моделюванням групи несправностей вставити в схему 32 фіктивних елемента, а після її моделювання - видалити. При цьому необхідно модифікувати схему: видаляти і додавати елементи в списки послідовників, формувати і моделювати фіктивні елементи. З огляду на те, що в кожній групі проводиться моделювання близько 500 вентилів, витрати часу на внесення несправностей таким способом залишаються великими.

Подальші способи збільшення швидкодії алгоритму паралельного моделювання полягають у зменшенні числа подій (обчислень елементів) і обробкою високоактивних несправностей.

Недоліком методів паралельного моделювання є те, що вони не встановлюють зв'язок між несправностями і виявляють їх тестовими наборами, що зменшує ефективність моделювання. Для її підвищення

розроблені методи, які для заданого тестового набору встановлюють всі виявлені несправності.

Дедуктивний метод ґрунтується на аналізі залежності значення сигналів на даному входному наборі від наявності несправностей. Для кожного контакту елементу схеми зберігається правильне значення сигналу на ньому, а також список тих несправностей, які змінюють його на протилежне. Списки таких несправностей для виходу елемента виходять зі списків для входів вентилля за допомогою спеціальних теоретико-множинних операцій, які визначаються виходячи з логіки функціонування елемента. Перевагами даного методу є:

- 1) в результаті одного кроку моделювання визначаються всі несправності, що перевіряються на даному наборі;
- 2) обчислення проводяться лише вздовж шляхів поширення несправностей;
- 3) повністю моделюється тільки поведінка справної схеми.

Недоліками є великий крок моделювання одного набору і високі вимоги ресурсів пам'яті.

Пояснимо суть дедуктивного методу на прикладі поширення несправностей через логічний елемент. Нехай моделюється логічний елемент  $f=A \cdot B \cdot C$ , на виходах якого встановилися значення  $A=1$ ,  $B=0$ ,  $C=1$ . Неправильну поведінку елемента буде викликати будь-яка несправність з  $B$  (тобто  $B=1$ ), але не міститься в  $A$  або  $C$ . Тому для поширення списків несправностей маємо:

$$F=C-(A \cup B).$$

Нехай, наприклад, списки несправностей містять:  $A=\{i1, j0, l0\}$ ,  $B=\{l0, h1\}$ ,  $C=\{j0, l1, h0\}$ , де малі літери позначають лінію в ДП, а індекси - тип несправності. Маємо,  $F=C-(A \cup B)=\{j0, l1, h0\}-\{i1, j0, l0, h1\}=\{l1, h0\}$ . Якщо в клас модельованих несправностей входять несправності виходу даного вентилля, то в список  $F$  необхідно додати несправність  $d1$ :

$$F=C-(A \cup B) \cup d1=\{l1, h0, d1\}.$$

Аналогічним чином можна отримати правила поширення несправностей для найпростіших логічних вентилів (табл.1.6.).

Для довільної булевої функції та будь-яких значень входів правила поширення списків несправностей можна отримати з виразу булева диференціала цієї функції простою заміною змінних  $X_i \rightarrow dx_i$  і булевих функцій відповідними теоретико-множинним операціям:  $\cap \rightarrow \&, \cup \rightarrow \vee, \neg \rightarrow \bar{\phantom{x}}$ .

За допомогою отриманих правил поширення списки несправностей просуваються від входів схеми до виходів. Таким чином, виходить загальний список перевірених даним набором несправностей. Загальна структура алгоритму дедуктивного моделювання подібна до алгоритму подієвого моделювання. Відмінність полягає в тому, що в даному випадку необхідно обробляти два види подій: зміна сигналів справного ДП і зміна списків несправностей. Оскільки довжина списку несправностей змінюється як у часі, так і при просуванні по схемі, це може привести до надмірних витрат пам'яті ЕОМ. Однак за швидкодією дедуктивний метод перевершує паралельний для великих схем.

Таблиця 1.6

### Поширення несправностей в дедуктивному методі

Тип вентиля	Вхід	Список виходу
<b>I, НЕ-I</b>	$I_0 = \emptyset$ $I_1 \neq \emptyset$	$\bigcup_{x_i \in I_1} X_i$ $\bigcap_{x_i \in I_0} X_i - \bigcup_{x_j \in I_1} X_j = \bigcap_{x_i \in I_0} X_i \bigcap_{x_j \in I_1} \bar{X}_j$
<b>АБО, НЕ-АБО</b>	$I_1 = \emptyset$ $I_0 \neq \emptyset$	$\bigcup_{x_i \in I_0} X_i$ $\bigcap_{x_i \in I_1} X_i - \bigcup_{x_j \in I_0} X_j = \bigcap_{x_i \in I_1} X_i \bigcap_{x_j \in I_0} \bar{X}_j$
<b>НЕ</b>		$X_1$

$I_1, I_0$  - безлічі входів вентиля, що мають значення 0 і 1 відповідно,  $x_i$  - список несправностей, пов'язаний з i-м входом вентиля.

Відомі також наближені дедуктивні методи, суть яких полягає в тому, що дедуктивне моделювання виконується в основному в двоїчному алфавіті.

Несправності, які встановлюють сигнал на лінії в невизначене значення, відзначаються знаком '\*' ("зіркова" нотація). Для таких несправностей потрібні додаткові правила обробки. У деяких випадках цей метод дає невірні результати.

Ще одним поширеним методом моделювання несправностей є конкурентний метод моделювання. У ньому для кожного вхідного набору відразу визначаються всі перевірені несправності, тобто він також є «однокроковим». Його відмінність від дедуктивного методу полягає в тому, що список несправностей тут асоціюється не з лінією ДП, а з логічним елементом. При цьому в список включаються як несправності, які змінюють вихід ДП, так і несправності, при яких значення сигналів відмінні від справних і на входах ДП. Видно, що такий список несправностей ширше, ніж при дедуктивному методі, тому він називається суперсписком. В даному методі всі несправності, включені в суперсписок, обробляються нарівні з справним ДП, тобто моделювання кожного несправного ДП виконується явно. Елемент суперсписка включає номер несправності і значення сигналів на входах і виходах.

У довільний момент модельного часу в списку можуть бути елементи, у яких значення сигналів збігаються зі справними. Такі елементи необхідно видалити з суперсписка, який при цьому звужується. При змінах сигналів в справному або несправному елементах вони можуть збігатися. Якщо змінювалися значення сигналів несправних елементів, то всі такі елементи необхідно порівняти зі справним. Якщо ж змінювалися значення в справному елементі суперсписка, то з ним необхідно порівняти всі несправні елементи суперсписка.

Аналогічно, через вплив несправності в процесі моделювання значення сигналів елемента можуть стати відмінними від значень цього елемента в справному ДП. У цьому випадку в суперсписок включається новий елемент, тобто суперсписок розширюється. Якщо для даної несправності значення виходу елемента відрізняється від відповідного справного значення, то вона



включається до списку несправностей всіх елементів-послідовників. Очевидно, що операції пошуку в суперсписку, включення і виключення елементів із суперсписку вимагають витрат часу.

У конкурентному методі моделювання елемента при наявності несправності завжди відбувається явно. Це дозволяє використовувати ефективні моделі елементів: табличні і функціональні. Також не викликає труднощів перехід до багатозначних алфавітів.

Методи моделювання ДП з несправностями, розглянуті вище, вимагають великих обчислювальних ресурсів. Як альтернатива їм були розроблені наближені методи, що оцінюють повноту тестів тільки за результатами моделювання справної схеми. Аналіз впливу несправностей на роботу схеми проводиться неявно. Тому він вимагає значно менших обчислювальних ресурсів.

Одним методом оцінки якості тестів, що не вимагає моделювання дефектів, є метод «статистичного аналізу дефектів» STAFAN. На основі понять «керованості» і «спостережливості» за даними моделювання справної схеми проводиться статистична оцінка ймовірності дефектів довільної схеми. В якості моделі розглядається модель схеми вентильного рівня з безліччю одиночних константних несправностей. Вводяться наступні величини:  $C1(l)$  - 1-керованість в точці  $l$  схеми як ймовірність значення 1 в цій точці на випадкових тестах;  $C0(l)$  - 0 керованість (визначається аналогічно);  $B1(l)$  - 1-спостережливість в точці  $l$  як ймовірність спостереження значення на зовнішньому виході при значенні 1 в точці  $l$ ;  $B0(l)$  - 0-спостережливість (вводиться аналогічно). Дані параметри можуть бути обчислені з тимчасових діаграм моделювання справної схеми. Далі вводяться ймовірності виявлення дефектів типу константа 1 і константа 0 наступним чином:

$$D1(l)=B0(l)*C0(l);$$

$$D0(l)=B1(l)*C1(l).$$

Обсяг додаткових обчислень в порівнянні з моделюванням справної схеми в методі STAFAN невеликий, а отримані експериментальні результати

для схем різних типів складності та обсягу добре узгоджуються з результатами програмного моделювання дефектів.

У дедуктивний метод явно моделюється поведінка тільки справної схеми, а несправних - дедуктивно. У конкурентному методі явно моделюються тільки ті елементи, значення входів або виходів яких відмінні від відповідних значень в справному ДП. У паралельному методі елементи в справному і несправному ДП моделюються явно навіть в тому випадку, якщо значення сигналів в них збігаються. З точних методів найшвидшим був конкурентний метод моделювання, однак, він вимагає дуже великих витрат пам'яті.

Останнім часом був запропонований новий алгоритм паралельного моделювання, що дозволяє в ряді випадків в кілька разів (до 10-ї) збільшити швидкість процесу моделювання. Також запропоновані ефективні методи зменшення подій при паралельному методі: сортування несправностей, обробка високоактивних несправностей. Це дозволяє говорити, що в даний час найефективнішим (по пам'яті і по швидкодії) є паралельний метод моделювання несправностей. Крім того, він дозволяє розширення на деякі нові типи несправностей, не втрачаючи своєї ефективності.

Широке поширення також отримали методи моделювання на функціональному рівні, що пов'язано з ускладненням модельованих пристроїв і необхідністю їх менш детального опису.

### **1.3. Аналіз методів побудови тестів**

Завдання побудови тестів цифрових пристроїв також є однією з центральних в технічній діагностиці. Розрізняють функціональне та діагностичне тестування. У першому випадку на об'єкт надходять впливи, передбачені робочим алгоритмом його функціонування. У другому випадку на об'єкт подаються спеціально організовані тестові впливи, деякі з яких можуть бути неможливі в процесі реальної роботи. Далі буде розглянуто тільки діагностичне тестування.

Нехай задано справний пристрій  $A_0$  і кінцева множина його несправних модифікацій  $A = \{A_1, A_2, \dots, A_n\}$ , і  $A_0 \neq A_i$  для  $i \in \overline{1, n}$ . Тестом, перевіряючим задану несправність, назвемо таку вхідну послідовність, вихідні реакції на яку пристроїв  $A_0$  і  $A_i$  різні. Тестом, перевіряючим всі несправності множини  $A$ , називається така вхідна послідовність, яка є перевіряючою для всіх  $A_i \in A$ .

Також як і при моделюванні ЦП розрізняють побудову тестів комбінаційних і послідовних ЦП. Проблема побудови тестів комбінаційних пристроїв розроблена набагато краще. Існує достатня кількість алгоритмів, що дозволяють домогтися прийнятної повноти генеруємих тестових наборів. Проблема побудови тестів для послідовних пристроїв досліджена значно гірше. Це обумовлено неможливістю однозначно відновити поведінку ЦП під час проведення експериментів з-за невизначеності початкового стану і конфліктів сигналів.

Методи побудови тестів для комбінаційних ЦП можна розбити на імовірнісні і детерміновані. Імовірнісні методи генерації тестів набули поширення завдяки своїй простоті. При їх реалізації використовуються програмні або апаратурні генератори випадкових чисел. Для комбінаційних ЦП вони дозволяють отримувати достатню повноту тестових наборів. До недоліків цих методів слід віднести те, що вони генерують тести великої довжини.

Детерміновані методи генерації тестів для комбінаційних ЦП діляться на алгебраїчні і структурні. До алгебраїчних відносять методи, засновані на застосуванні дужкових і еквівалентних нормальних форм. До основоположних в теоретичному плані відноситься метод розрізняючої функції. На даному методі засновані всі алгоритми, що використовують логічне диференціальне числення. Дані методи шляхом отримання булевих похідних дозволяють аналітично отримати умову поширення несправності.

Активізація шляхів поширення несправностей в багатозначних алфавітах є спільною рисою всіх структурних методів. При цьому ранні

алгоритми реалізовували одновимірну активізацію шляхів в 3-х значному алфавіті. Однак ці методи не гарантували побудови тестів. Тому пізніше звернулися до багатовимірної активізації шляхів у багатозначних алфавітах. Структурні методи в свою чергу розбиваються на методи не орієнтовані на одну несправність і методи орієнтовані на задану несправність. Методи першої групи засновані на припущенні, що активізуюча вхідна послідовність перевіряє половину всіх несправностей на активізованих нею критичних шляхах. Для того щоб отримати тест для більшого числа несправностей, потрібно генерувати послідовності, які породжують якомога більше критичних шляхів. При цьому заздалегідь неможливо сказати, для яких несправностей буде побудований тест, тому даний метод відноситься до наближених.

На наступному етапі застосовуються методи, орієнтовані на задану несправність. Раніше інших був розроблений D-алгоритм, який використовує багатовимірну активізацію критичних шляхів у 6-значному алфавіті. На відміну від методів одновимірної активізації, D-алгоритм гарантує побудову тесту. Пізніше були розроблені різні модифікації D-алгоритму, в тому числі і на функціональному рівні. Однак для схем, що містять велику кількість східних розгалужень, D-алгоритм втрачає свою ефективність, внаслідок великої кількості виникаючих конфліктних ситуацій. Через це час генерації стає дуже великим, а в ряді випадків процес завершити не вдається. Для таких схем був розроблений метод PODEM. Також як і D-алгоритм він використовує 6-значний алфавіт і багатовимірну активізацію шляхів. До відмінностей слід віднести: відмінні процедури D-поширення і повернення, інша стратегія обробки конфліктних ситуацій і виконання процедури імплікації в прямому напрямку (від входів схеми до виходів). Ефективність методу PODEM визначається, перш за все, процедурою зворотного поширення. У ній при пошуку рішення на основі показників керованості вибирається або найбільш перспективний варіант, або дозволяючий виявити суперечливу ситуацію якомога швидше. До переваг даного методу також

відноситься простота реалізації. Показана також перевага даного методу для великих комбінаційних схем. Авторами даного методу також розроблено ефективний псевдовипадковий метод RASP, ідеї якого поширені в методі FAN. У ньому проводиться аналіз ДП, в результаті якого схема розбивається на деревовидні підсхеми. Додатково використовуються прийоми, що дозволяють зробити більш ефективними процедури просування назад і імплікації (шляхом розпаралелювання). Це дозволяє істотно зменшити число повернень по дереву рішення, що знижує час генерації тесту. У всіх цих методах, як вказувалося вище, використовується 6-значний алфавіт. Розроблено методи, в яких для скорочення простору пошуку та більш ефективної активізації застосовуються алфавіти більшої значності.

Також останнім часом набули поширення методи, засновані на «техніці навчання» і ряд інших методів.

Проблема розробки детермінованих методів побудови тестів для послідовних пристроїв досліджена набагато гірше. Це пояснюється двома основними виникаючими при цьому проблемами: невизначеністю початкового стану і конфліктами сигналів. Існуючі методи можна розбити на дві групи: абстрактні і структурні.

У першій групі в якості моделі використовується кінцевий автомат. При цьому структура схеми не відома і не використовується. Як математична модель використовується кінцевий автомат, що задається таблицею або графом переходів і виходів. Перевагою даних методів є висока повнота перевірки несправностей отриманими тестами і гарантія знаходження рішення (якщо таке існує). До їх недоліків відносяться велика довжина тестів, необхідність генерації таблиці переходів і виходів вихідного ДП і ігнорування інформації про структуру ДП.

Більшість структурних методів генерації тестів для послідовних схем засновані на перетворенні схеми в ітеративний комбінаційний еквівалент шляхом обриву зворотного зв'язку. При цьому найбільшого поширення набув підхід, при якому до отриманої ітеративної схеми застосовується один з

методів генерації тестів для комбінаційних ЦП. У ранніх роботах адаптувався D-алгоритм, пізніше - PODEM. Оскільки при такому підході спостерігаються реакції тільки останнього комбінаційного еквівалента схеми, тобто реалізована одиночна стратегія спостереження, то часто він дає негативні результати, хоча тест існує і може бути побудований аналітичними методами. Щоб уникнути вказаного недоліку пропонується алгоритм, в якому враховуються вихідні реакції всіх копій ітеративної схеми. Однак, в той же час неявно передбачається, що справний і несправний ДП мають один і той же стан.

Серед структурних методів виділяються структурно-аналітичні, в яких умови активізації несправностей виводяться з аналітичних виразів на основі знань про структуру схеми. У них генерація тесту виконується шляхом побудови розрізняючої функції для комбінаційної ітеративної схеми. Метод побудови заснований на булевом диференціальному обчисленні. При цьому враховуються вихідні реакції всіх копій, завдяки застосуванню, так званої, стратегії кратного спостереження. Ці методи за своєю ідеологією аналогічні абстрактним методам, заснованим на побудові дерева попередників. З їх допомогою можуть генеруватися тестові послідовності для несправностей, для яких не можуть бути побудовані тести структурними методами.

Перевага методів, що враховують вихідні реакції всіх копій комбінаційної ітеративної схеми, полягає в тому, що вони враховують ситуацію, коли в залежності від станів справної і несправної схем розрізняються вихідні реакції різних ітеративних копій. Такий підхід називається кратною стратегією спостереження.

Останнім часом набувають поширення методи, засновані на моделюванні. Процес генерації тестів базується на використанні процедури спільного моделювання ДП з несправностями. У методі, заснованому на "техніці навчання", розвинені ідеї, закладені в алгоритмі SOCRATES для комбінаційних схем. Останнім часом деякі дослідники звернулися до, так

званого, змішаного підходу, при якому генерація тестів заснована частково на функціональному і частково на структурному підходах.

#### **1.4. Висновки до розділу**

Моделювання поведінки цифрових пристроїв вимагає побудови моделей цих пристроїв. Ці моделі є абстракцією реальної схеми, яка представляє якомога більше інформації, необхідної для моделювання на обраному рівні уявлення. У зв'язку з цим, в даній роботі, були розглянуті рівні уявлення цифрових пристроїв (системний, рівень реєстрових передач, вентильний, комутаційний і схемний), математичні моделі комбінаційних, послідовних пристроїв і описана модель вентильного рівня уявлення.

Після цього, був проведений аналіз основних методів моделювання цифрових пристроїв: методів наскрізного і подієвого моделювання, методу паралельного за несправностями моделювання, методу введення фіктивних елементів, дедуктивного методу, конкурентного методу.

З результатів проведеного аналізу випливає, що з точних методів моделювання найшвидшим є конкурентний метод моделювання, однак, він вимагає дуже великих витрат пам'яті. В даний час найефективнішим (по пам'яті і по швидкодії) є паралельний метод моделювання.

Так само були розглянуті основні методи побудови тестів: D-алгоритм, метод PODEM, метод RASP, метод SOCRATES.

Підводячи підсумок аналізу, робимо висновок, що в даний час не існує ефективних методів моделювання та генерації тестів для складних цифрових схем, тому що вони вимагають надмірних витрат ресурсів пам'яті і часу.

## РОЗДІЛ 2

### РЕАЛІЗАЦІЯ АЛГОРИТМУ ІНІЦІАЛІЗАЦІЇ ЦИФРОВИХ СХЕМ

#### 2.1. Вибір програмних засобів реалізації

В даний час, велике поширення і популярність отримали різні версії операційної системи сімейства Microsoft Windows. У зв'язку з цим було прийнято рішення розробляти програмний додаток, орієнтований для роботи в даному операційному середовищі. Для вирішення поставленого завдання такий варіант є оптимальним.

Існує безліч засобів розробки програмного забезпечення орієнтованого на роботу в операційному середовищі Windows. Найбільш популярні з них:

- Java
- Visual C ++
- Borland Delphi
- Visual Basic

В якості інструментарію для розробки безпосередньо програмного коду, було вирішено використовувати середовище програмування Borland Delphi 7.0 мови Object Pascal. Вибір обумовлений тим, що дане середовище володіє достатніми можливостями для реалізації поставлених цілей, досить зручним інтерфейсом і структурою. На додаток до цього існує безліч сторонніх компонентів розроблених для середовища програмування Delphi, які розширюють або доповнюють стандартний набір компонентів.

#### 2.2. Аналіз схеми

У даній роботі розглядається новий алгоритм для побудови ініціюючих послідовностей, який можна розділити на чотири етапи: аналіз схеми (виділення підсхем), формування варіантів вхідних послідовностей, моделювання схеми, виділення ініціюючих послідовностей.

На початковому етапі роботи програми проводиться аналіз завантажуючої схеми - з вихідного файлу, що містить опис схеми,



виділяються наступні дані: назва елемента, тип елемента, ранг тригера, вхідні операнди, вихідні операнди, ранг елементу, входу (таблиця 2.1).

Таблиця 2.1

### Структура елемента схеми

Назва	Ідентифікатор	Тип	Опис
Назва елемента	Name	String	Назва елемента в схемі
Тип елемента	EType	TElementType TElementType=0..9	0 - Вхід 1 - Вихід 2 - Тригер 3 - Елемент Not 4 - Елемент And 5 - Елемент Or 6 - Елемент Nand 7 - Елемент Nor 8 - Елемент Xor 9 - Елемент Nxor
Ранг тригера	TriggerRank	Word	Кількість тригерів, через які повинен пройти сигнал від входу схеми до входу тригера по найкоротшому шляху
Вхідні операнди	Operands	array of Longword	Номери елементів, від яких безпосередньо залежить даний елемент
Номер входу	Src	array of Longword	Номери входів, від яких залежить даний елемент
Вихідні операнди	Outputs	array of Longword	Номери елементів, які залежать від даного елемента
Ранг елемента	Rank	Integer	Кількість елементів, через які повинен пройти сигнал від входу схеми до входу даного елемента по найкоротшому шляху

Виділені дані відповідним чином групуються і зберігаються, після чого здійснюється виділення підсхем з даної схеми.

Для виділення підсхем використовується алгоритм, розроблений Чебановим П.О.

Згідно з цим алгоритмом, для побудови ініціалізуючої послідовності вхідних сигналів пропонується використовувати імовірнісний підхід при моделюванні логічної схеми. Для цього використовується алфавіт  $P = \{u, H_p, L_p\}$ , що визначає ймовірність ініціалізації тій чи іншій лінії. У цьому алфавіті символ  $H_p$  використовується при ймовірності ініціалізації лінії що дорівнює 1, символ  $u$  використовується при ймовірності ініціалізації лінії рівної 0, і, відповідно, символ  $L_p$  використовується при ймовірності ініціалізації лінії більше нуля і меншою одиниці. Ці символи можна розуміти таким чином, символ  $H_p$  показує, що значення на цій лінії буде певним незалежно від сигналів на входах схеми, символ  $u$  - значення на цій лінії буде невизначеним незалежно від сигналів на входах схеми і символ  $L_p$  - значення на цій лінії буде певним або невизначеним в залежності від сигналів на входах схеми. Визначимо для такого алфавіту стандартні функції булевої алгебри, використовувані при описі цифрових схем на логічному рівні. Для цього задамо їх наступними таблицями істинності.

Таблиця 2.2

**Таблиця істинності логічного AND**

AND	$u$	$L_p$	$H_p$
$u$	$u$	$L_p$	$L_p$
$L_p$	$L_p$	$L_p$	$L_p$
$H_p$	$L_p$	$L_p$	$H_p$

Таблиця 2.3

**Таблиця істинності логічного OR**

OR	$u$	$L_p$	$H_p$
$u$	$u$	$L_p$	$L_p$
$L_p$	$L_p$	$L_p$	$L_p$
$H_p$	$L_p$	$L_p$	$H_p$

Таблиця істинності логічного XOR

XOR	u	Lp	Hp
u	u	Lp	Lp
Lp	Lp	Lp	Lp
Hp	Lp	Lp	Hp

Таблиця 2.5

Таблиця істинності логічного NOT

NOT	u	Lp	Hp
A	u	Lp	Hp
$\bar{A}$	u	Lp	Hp

Всі таблиці істинності елементарно виходять з таблиць істинності логічних елементів на тризначному алфавіті ЕЗ. Наприклад, при моделюванні логічного множення, якщо на обидва входи подати значення з ймовірністю ініціалізації що дорівнює 1, тобто на входах вентиля будуть значення або 0, або 1, вентиль на виході дасть певне значення за своїм табличним описом в алфавіті ЕЗ. Якщо ж один з входів ініціалізується з ймовірністю 0, тобто на ньому буде значення u, то ймовірність ініціалізації виходу вентиля буде менше 1, але більше нуля. Це легко пояснити тим фактом, що якщо ініціалізований вхід прийме значення 0, то на виході однозначно буде 0, а якщо ініціалізований вхід прийме значення 1, то значення на виході дорівнюватиме u. Відповідно, при ймовірності ініціалізації обох входів більшою нуля, ймовірність ініціалізації виходу буде також більше нуля, але менше 1.

Моделювання схеми на такому алфавіті має на меті розбиття послідовних схем на ряд підсхем, причому кількість цих підсхем дорівнюватиме послідовній глибині схеми. Розбиття на підсхеми виробляється в такий спосіб:

1. Нехай  $n$  - це послідовна глибина схем.

2. Подаємо послідовність вхідних сигналів довжиною 1 що складається з елементів  $H_r$  на зовнішні входи схеми (при моделюванні в алфавіті  $P$  всієї схеми, ми також замінюємо входи тригерів на псевдовиходи схеми, а виходи тригерів - на псевдовходи схеми). Після моделювання ми отримаємо схему, в якій буде видно всі лінії і тригера, тому що на цих лініях значення будуть відмінні від  $u$ , для яких можна побудувати ініціалізовану послідовність.
3. Видаляємо з схеми елементи, які мають на виходах значення  $u$ , і отримуємо таким чином першу підсхему. Якщо в результаті видалення вентиля виходить лінія з невідомим значенням, то замінюємо цю лінію на сигнал  $constU$ .
4. Подаємо послідовність вхідних сигналів довжиною 1 що складається з елементів  $H_r$  на зовнішні входи повної схеми і псевдовходи першого рангу.
5. Видаляємо з схеми елементи, які мають на виходах значення  $u$ , і отримуємо таким чином другу підсхему.
6. Повторюємо кроки 4,5  $n-2$  раз, кожного разу збільшуючи розмір послідовності вхідних сигналів на кількість тригерів відповідного рангу.
7. В результаті отримуємо  $n$  підсхем, необхідних для моделювання.

Розглянемо наведений алгоритм на прикладі схеми, представленої на рис. 2.1.

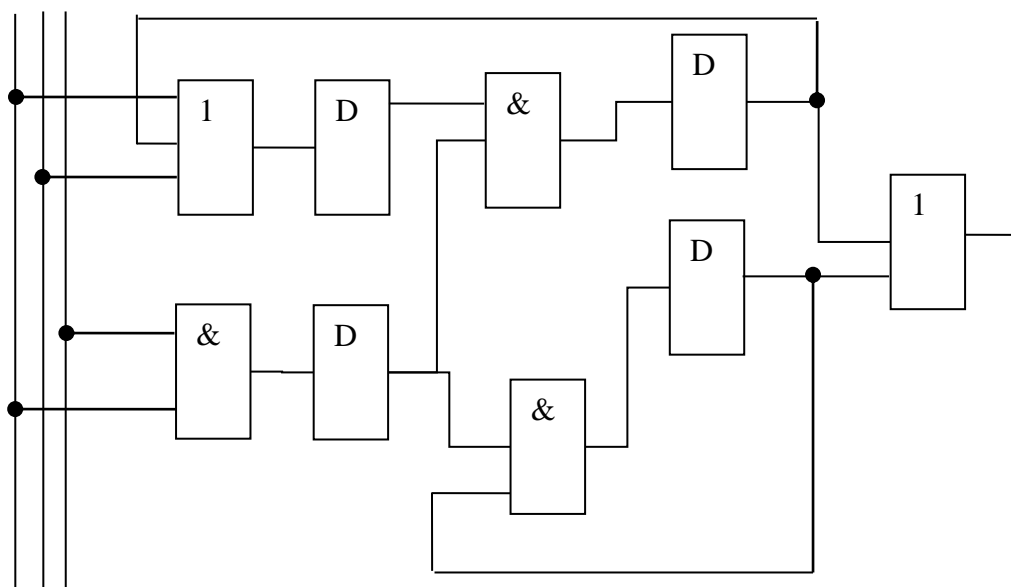


Рис. 2.1. Вихідна схема

Після виконання 1 кроку алгоритму отримуємо наступні значення сигналів на лініях (рис. 2.2).

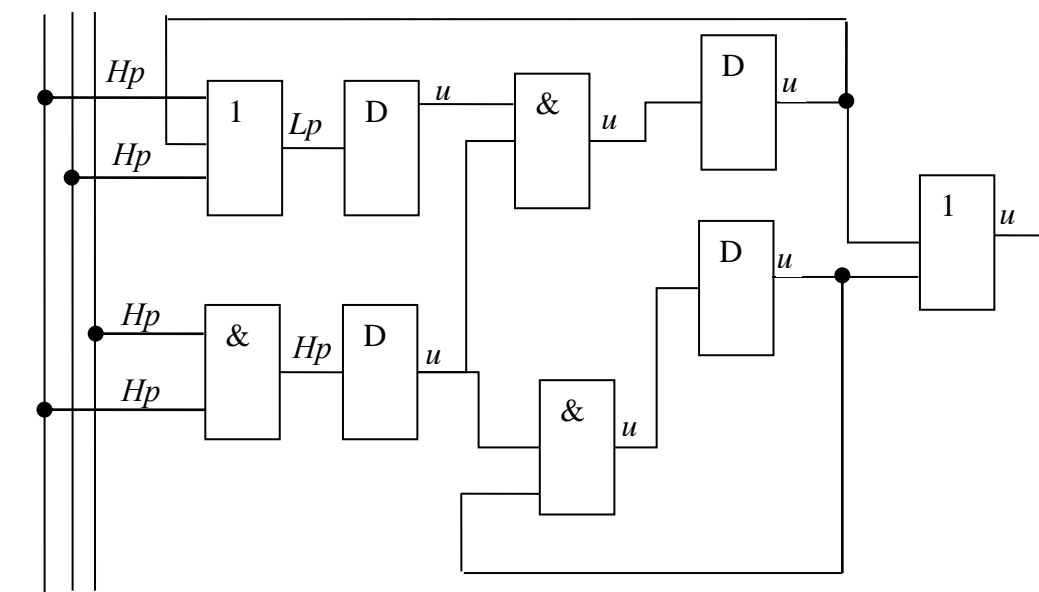


Рис. 2.2. Значення ліній схеми

Після видалення невизначених елементів і заміни невизначених ліній, отримуємо першу підсхему (рис. 2.3).

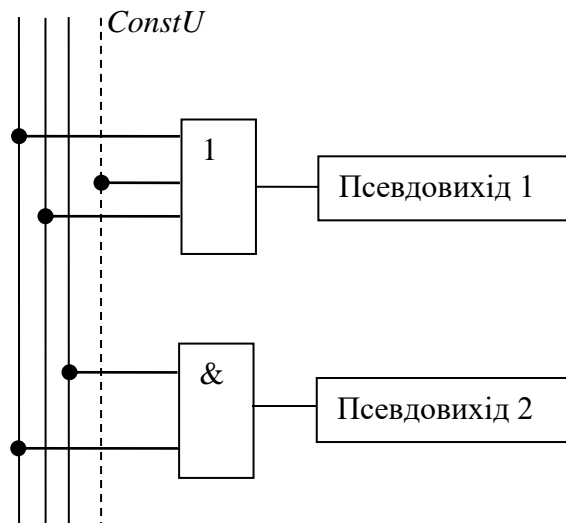


Рис. 2.3 Підсхема №1

Далі, моделюємо повну схему на алфавіті  $P$ , задаючи значеннями  $H_p$  псевдовиходи першого рангу. В результаті отримуємо такі значення ліній схеми (рис. 2.4). Після виконання кроку 5 ми отримаємо другу підсхему, зображену на рис. 2.5.

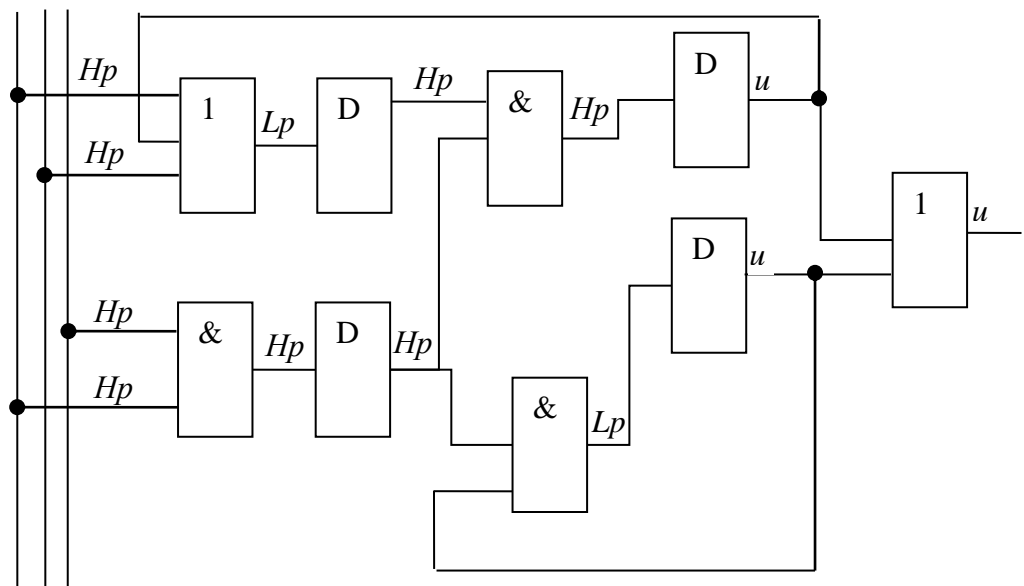


Рис. 2.4. Значення ліній схеми після кроку 4

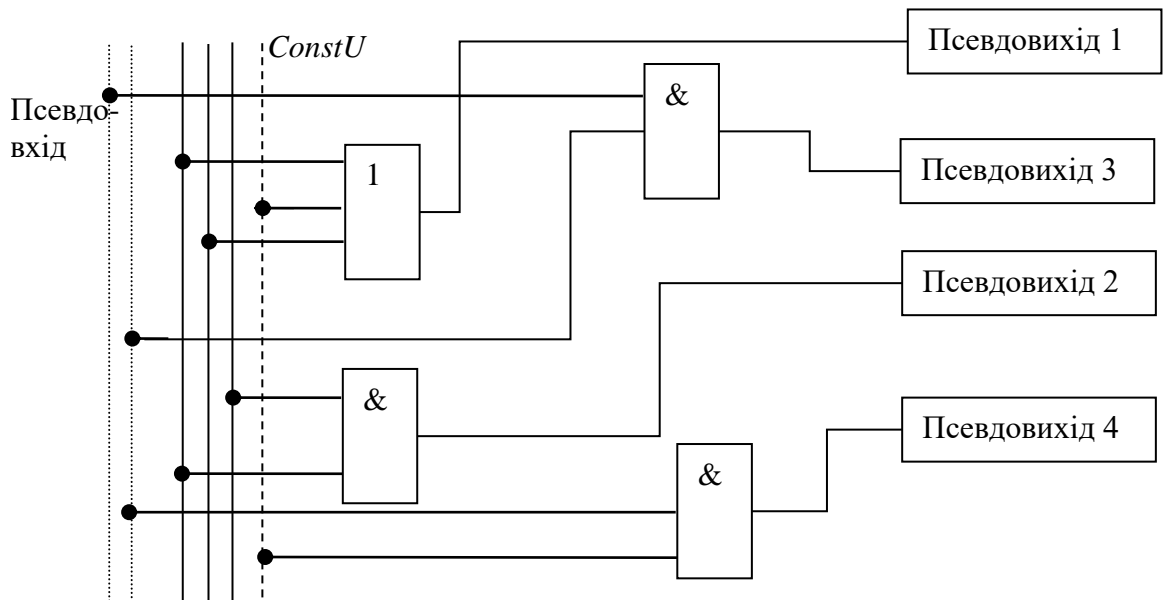


Рис. 2.5. Підсхема №2

Після виділення всіх підсхем вони групуються по порядку і зберігаються.

Перебираючи тим чи іншим способом послідовності вхідних сигналів довжини 1 і моделюючи першу отриману схему на двозначному алфавіті В2, ми можемо отримати всі ініціалізуючі послідовності для тригерів 1 рангу. Після отримання таких послідовностей ми можемо приступити до моделювання другої підсхеми, використовуючи підготовлені на попередньому етапі послідовності для ініціалізації тригерів 2 рангу і т.д. В кінцевому підсумку ми отримаємо ініціалізуючі послідовності для всіх тригерів схеми.

### 2.3. Формування варіантів вхідних послідовностей

Після того, як був проведений аналіз схеми і були сформовані підсхеми, згідно розроблюваного алгоритму, проводиться формування всіх можливих варіантів вхідних послідовностей. Кількість можливих вхідних послідовностей визначається за формулою:

$$M = 2^N$$

де М - кількість всіх можливих вхідних послідовностей, а N - кількість входів схеми.

Знаючи загальну кількість вхідних послідовностей, методом простого перебору, формуються всі можливі варіанти сигналів для кожного входу.

Таким чином, отримуються всі можливі вхідні послідовності на двозначному алфавіті В2. Але при проведенні ініціалізації схеми (підсхеми) на деякі елементи можуть надходити невизначені сигнали *u*, в результаті чого виникає необхідність ініціалізації схеми в тризначному алфавіті Е3.

Для сумісності алфавітів і зручності ініціалізації, всі сигнали кодуються за наступною схемою (таблиця 2.6):

Таблиця 2.6

**Кодування сигналів**

Сигнал	Закодоване значення	
	Молодший біт	Старший біт
0	1	0
1	0	1
<i>u</i>	0	0

Відповідно до такої схеми кожен сигнал представляється 2 числами. Це уявлення дає можливість, посилаючи на входи схеми сигнали з високим і низьким рівнем, які сприймаються як набори нулів та одиниць (двозначний алфавіт В2), проводити ініціалізацію з урахуванням невизначеного сигналу *u* (тризначний алфавіт Е3).

З наведеної вище формули знаходження всіх можливих вхідних послідовностей видно їх залежність від кількості входів схеми. Чим більше входів у схемі, тим більше варіантів вхідних послідовностей необхідно промодельовувати, що відповідно тягне за собою збільшення часу для знаходження ініціюючих послідовностей. Для оптимізації обчислень застосовується кодування всіх можливих вхідних послідовностей: кожна вхідна послідовність представляється у вигляді числа в десятковій системі числення, за допомогою запису відповідного значення кожної послідовності в відповідний біт десяткового числа.

Розглянемо принцип даного алгоритму для схеми, що містить два входи.



Спочатку перетворимо наявні вхідні послідовності згідно з розробленою схемою кодування (таблиця 2.7):

Таблиця 2.7

### Перетворені вхідні послідовності

	Вхід 1		Вхід 2	
	Молодший біт	Старший біт	Молодший біт	Старший біт
<b>1 послідовність</b>	1	0	1	0
<b>2 послідовність</b>	1	0	0	1
<b>3 послідовність</b>	0	1	1	0
<b>4 послідовність</b>	0	1	0	1

Після перетворення, запишемо відповідне значення сигналу в відповідний біт числа, т. т. представимо всі вхідні послідовності як числа двійкової системи числення і проведемо операцію перекладу значень всіх вхідних послідовностей кожного входу з двійкової системи числення в десяткову. Таким чином, для Входу 1 ми отримаємо два десяткових числа (закодованих вхідних послідовностей):

$$0011_2 = 3_{10}$$

$$1100_2 = 12_{10}$$

Приклад закодованих вхідних послідовностей представлений в таблиці 2.8.

Таблиця 2.8

### Приклад закодованих вхідних послідовностей для схеми з двома входами

	Вхід 1		Вхід 2	
<b>Закодована послідовність</b>	3	12	5	10

## 2.4. Моделювання схеми

Після того, як були сформовані і закодовані всі вхідні послідовності, здійснюється моделювання кожної підсхеми.

Перед здійсненням безпосередньої ініціалізації кожного елемента організується циклічна черга - упорядкований замкнутий список, елементи якого додаються в один кінець списку, а видаляються з іншого кінця. Як

елементи черги використовуються порядкові номери елементів підсхеми (рис. 2.6).

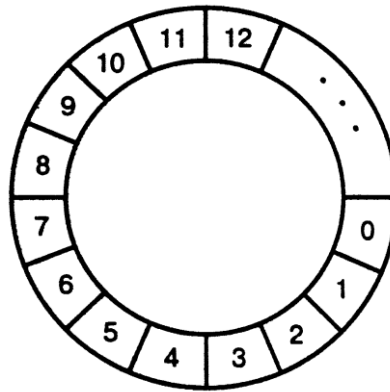


Рис. 2.6. Графічне представлення циклічної черги

Моделювання підсхеми здійснюється за наступним алгоритмом:

1. Циклічна черга заповнюється елементами підсхеми, у яких властивість Rank має значення 1 (таблиця 2.1).
2. З черги береться перший елемент і запам'ятовується його поточне значення.
3. Здійснюється моделювання елемента.
4. Перевіряється, чи змінилося значення сигналу після моделювання. Якщо значення змінилося, номери елементів, які знаходяться у властивості Outputs, додаються в кінець черги (рис. 2.7).
5. Промодельований об'єкт видаляється з черги (рис. 2.8).
6. Повторюються пункти 2-5 доки черга не очиститься.

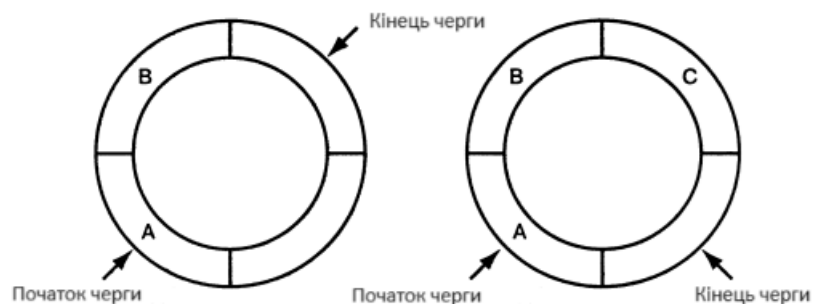


Рис. 2.7. Графічне представлення додавання елемента С в кінець

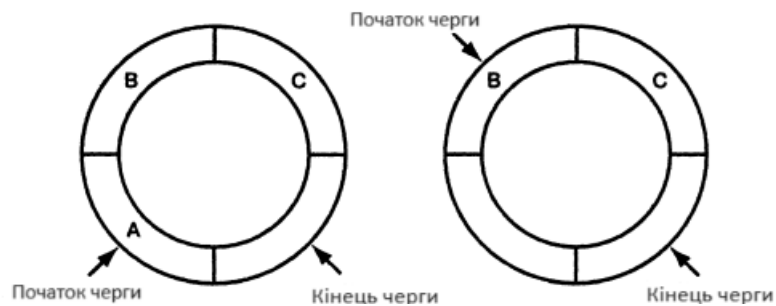


Рис. 2.8. Графічне представлення видалення елемента А з черги

Розглянемо процес моделювання елемента (3 пункт алгоритму моделювання підсхеми), який полягає у виконанні двох побітових операцій над закодованими значеннями сигналів.

Кожен елемент підсхеми має властивість EType (таблиця 2.1), яка визначає тип елемента, в залежності від якого виконується відповідний набір побітових операцій:

- елемент Not - моделювання здійснюється за допомогою перестановки місцями значень що подаються (значення  $X_{11}$  записується в  $X_{12}$ , а початкове значення  $X_{12}$  записується в  $X_{11}$ ).

- елемент And - моделюється за допомогою двох логічних операцій:

$$X_{11} \text{ or } X_{21}$$

$$X_{12} \text{ and } X_{22}$$

- елемент Or - моделюється наступними побітовими операціями:

$$X_{11} \text{ and } X_{21}$$

$$X_{12} \text{ or } X_{22}$$

- елемент Nand - моделюється за допомогою почергового виконання двох ініціюючих операцій для елемента And і Not.

- елемент Nor - моделюється шляхом почергового виконання операцій елементів Or і Not.

- елемент Xor - моделюється набором наступних побітових операцій:

$$(X_{11} \text{ and } X_{21}) \text{ or } (X_{12} \text{ and } X_{22})$$

$$(X_{12} \text{ and } X_{21}) \text{ or } (X_{11} \text{ and } X_{22})$$

- елемент Nxor - моделюється за допомогою почергового виконання ініціалізуючої операції елемента Xor і елемента Not.

Під значеннями  $X_{11}$  і  $X_{12}$  маються на увазі десяткові числа - закодовані складові послідовності для першого вхідного елемента (приклад в таблиці 2.8), а  $X_{21}$  і  $X_{22}$  - для другого вхідного елемента (якщо йдеться про 2-вхідний елемент).

Як приклад розглянемо алгоритм моделювання 2-вхідного вентилі And. Входи і виходи цього елемента приймають значення з тризначного алфавіту  $E_3 = \{0, 1, u\}$ . При моделюванні підсхеми ми розглядали сигнал як комбінацію двох чисел (таблиця 2.6), набір яких, згодом, кодується в десяткове число. Оскільки над закодованими сигналами виконуються побітові логічні операції, в даному прикладі не будемо розглядати десяткові числа, а тільки один з них бітів - т. т. уявлення сигналу з таблиці 2.7 (інші біти десяткового числа моделюються аналогічно).

Нехай на входи подані сигнали 1 і  $u$ , де перший сигнал представлений у вигляді  $x_{11}=0$ ,  $x_{12}=1$ , а другий - у вигляді  $x_{21}=0$ ,  $x_{22}=0$ . В процесі обробки сигналів виконуються дві побітові операції:

1) Перша операція or здійснюється над першими складовими кожного уявлення сигналу:  $x_{11}$  or  $x_{21}$ ;

2) Друга операція and здійснюється над другими складовими кожного уявлення сигналу:  $x_{12}$  and  $x_{22}$ .

У підсумку на виході ми отримуємо два значення 0 і 0, що відповідає невизначеному сигналу  $u$  троїчному алфавіту  $E_3$  в нашій системі кодування (таблиця 2.6).

## **2.5. Виділення ініціюючих послідовностей**

Після етапу моделювання схеми ми отримуємо всі можливі набори вихідних сигналів в закодованому вигляді, з яких необхідно виділити ті унікальні набори, які не містять невизначених значень  $u$ . Для цього використовується наступний алгоритм:

1. Декодування обчислених вихідних наборів.
2. Виділення унікальних наборів.
3. Пошук відповідних вхідних послідовностей.

4. Збереження ініціюючих вхідних послідовностей.
5. Формування нових вхідних послідовностей.

Розглянемо детальніше процес реалізації даного алгоритму.

Процес декодування вихідних сигналів є зворотним процесу кодування: обчислені десяткові значення переводяться в числа двійкової системи числення, які групуються відповідно до сформувавших їх вхідних послідовностей (таблиця 2.7), і переводяться в тризначний алфавіт E3 згідно схеми кодування з таблиці 2.6.

Елементи схеми можуть бути розташовані так, що різні вхідні послідовності, під час ініціалізації, призводитимуть до одних і тих же вихідних наборів. Тому, після етапу декодування, проводиться відбір унікальних (неповторюваних) наборів вихідних послідовностей. Якщо на виході було отримано кілька однакових вихідних послідовностей, унікальною вважатиметься перша отримана послідовність.

Коли всі унікальні набори вихідних послідовностей були виділені, проводиться перевірка відсутності в них невизначеного сигналу  $u$ , після чого здійснюється відбір відповідних вхідних послідовностей.

Відібрані таким чином вхідні послідовності і будуть ініціалізуючими вхідними послідовностями для даної цифрової підсхеми (схеми).

Якщо схема була розділена на кілька підсхем, здійснюється формування нових вхідних послідовностей для наступної підсхеми: до вхідних послідовностей, що отримуються методом простого перебору, додаються набори вихідних послідовностей, отримані при ініціалізації попередньої підсхеми. Ці набори вихідних послідовностей надсилаються на відповідні тригери підсхеми. Після цього здійснюється ініціалізація даної підсхеми.

Коли всі підсхеми промодельовані, обчислені ініціалізуючі вхідні послідовності зберігаються в файл результатів.

## 2.6. Тестування розробленої програми

Розроблений алгоритм був реалізований в вибраному середовищі розробки Borland Delphi 7.0 у вигляді 2 процедур і 3 функцій:

- CalcSetupSignal - процедура, в якій здійснюється аналіз схеми і виділення підсхем.
- Initialize - процедура, в якій здійснюється формування вхідних послідовностей, моделювання елементів і виділення ініціюючих вхідних послідовностей.
- SetBit - функція кодування вхідних послідовностей.
- Del\_elem - функція видалення елемента з черги.
- Add\_elem - функція додавання елемента в чергу.

Цей програмний додаток було протестовано за допомогою стандартних схем з міжнародного каталогу ISCAS-89, дані про які представлені в таблиці 2.9.

Таблиця 2.9

**Характеристики тестових схем**

Назва схеми	Кількість елементів	Кількість входів	Кількість виходів	Кількість тригерів
S27	10	4	1	3
S400	164	3	6	21
S1	7	3	1	5
S298	139	3	6	14
S344	160	9	11	15
S349	161	9	11	15
S382	158	3	6	21
S510	211	19	7	6
S635	286	2	1	32
S4201	218	18	1	16

Дані схеми були по черзі завантажені в розроблений додаток. За результатами роботи була сформована таблиця 2.10.

## Результати тестування програми

Назва схеми	Послідовнісна глибина	Загальна кількість послідовностей	Кількість ініціюючих послідовностей	Час роботи (сек.)
S27	1	9	3	0,0007
S400	2	4	1	0,001
S1	3	6	6	0,0002
S298	2	5	3	0,001
S344	2	258	256	0,031
S349	2	258	256	0,031
S382	2	5	1	0,0008
S510	3	4	1	9,797
S635	2	4	1	0,016
S4201	2	4	0	3,046

Після цього була проведена перевірка результатів роботи програми і зроблені нижченаведені висновки.

### 2.7. Висновки до розділу

В даному розділі представлений новий алгоритм для побудови ініціюючих послідовностей, який ділиться на чотири етапи: аналіз схеми (виділення підсхем), формування варіантів вхідних послідовностей, ініціалізація схеми, виділення ініціюючих послідовностей.

На першому етапі алгоритму здійснюється загальний програмний аналіз схеми і відбувається її розбиття на підсхеми.

На другому етапі, методом перебору, будуються всі можливі вхідні послідовності.

Наступний етап алгоритму реалізує процес ініціалізації схеми шляхом її моделювання.

На останньому етапі здійснюється виділення і збереження ініціюючих вхідних послідовностей.

Даний алгоритм реалізований за допомогою середовища програмування Borland Delphi 7.0, в результаті чого було отримано програмний додаток (Додаток А).

Було проведено тестування розробленого додатка.

Проаналізувавши отримані результати, можна зробити наступні висновки:

- розроблений алгоритм застосуємо для знаходження ініціюючих вхідних послідовностей;
- програма, що реалізує даний алгоритм, видала коректні результати для розглянутих схем;
- додаток показав високу швидкість ініціалізації тестових схем.



## ВИСНОВКИ

У магістерській роботі було розглянуто розробку алгоритму генерації ініціюючих вхідних послідовностей для цифрових схем.

В ході виконання магістерської роботи були проведені дослідження в області моделювання і тестування цифрових пристроїв.

У даній роботі були розглянуті п'ять рівнів уявлення цифрових пристроїв: системний рівень, рівень реєстрових передач, вентиляний рівень, комутаційний і схемний рівні. Моделювання поведінки цифрових пристроїв вимагає побудови моделей цих пристроїв. Ці моделі є абстракцією реальної схеми, яка представляє якомога більше інформації, необхідної для моделювання на обраному рівні уявлення. У зв'язку з цим були розглянуті математичні моделі комбінаційних, послідовних пристроїв і описана модель вентиляного рівня уявлення.

Було проведено аналіз основних методів моделювання цифрових пристроїв: методів наскрізного і подієвого моделювання, методу паралельного по несправностям моделювання, методу введення фіктивних елементів, дедуктивного методу, конкурентного методу. Аналіз показав, що з точних методів моделювання найшвидшим є конкурентний метод моделювання, однак, він вимагає дуже великих витрат пам'яті. В даний час найефективнішим (по пам'яті і по швидкодії) є паралельний метод моделювання.

Так само були розглянуті основні методи побудови тестів: D-алгоритм, метод PODEM, метод RASP, метод SOCRATES.

Провівши аналіз методів моделювання і тестування цифрових пристроїв, був зроблений висновок про те, що в даний час не існує ефективних методів моделювання та генерації тестів для складних цифрових схем, тому що вони вимагають надмірних витрат ресурсів пам'яті і часу.

У зв'язку з цим було запропоновано новий алгоритм генерації ініціюючих вхідних послідовностей для цифрових схем, що базується на імовірнісному підході.

Даний алгоритм був розділений на чотири етапи:

1. Аналіз схеми (виділення підсхем).
2. Формування варіантів вхідних послідовностей.
3. Моделювання схеми.
4. Виділення ініціюючих послідовностей.

Відповідно до даного алгоритму було розроблено програмний додаток і проведено його тестування.

Проаналізувавши отримані результати, був зроблений висновок про високу швидкість виконання алгоритму для тестованих схем.

Усі поставлені цілі і завдання магістерської роботи були досягнуті.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ангер С. Асинхронні послідовні схеми. М.: Наука, 1977.
2. Артюхов В.Л., Кондратьєв В.М., Шалит А.А. Реалізація булевих функцій арифметичними поліномами // Автоматика і телемеханіка, 1988. № 4.
3. Архагельскій А.Я. Delphi 7. Довідковий посібник - М.: ТОВ «Біном-Пресс», 2003 р
4. Баркер С. Професійне програмування в Microsoft Access 2002.: Пер. з англ. - М.: Видавничий дім «Вільямс», 2002. - 992 с.: іл. - Парал. тит.
5. Боровський А.Н. Програмування в Delphi 2005. - СПб.: БХВ - Петербург, 2005. - 448 с.: іл.
6. Боуман Дж., Емерсон С., Дарновські М. Практичний посібник з SQL СПб.: БХВ - Санкт-Петербург, 1999. - 322 с.: іл.
7. Гелль П.П., Іванов-Есипович Н.К. Конструювання і мікромініатюризація радіоелектронної апаратури: Навч. для вузів. - Л.: Вища школа, 1984. - 536 с.
8. Єлманова Н., Трепалін С., Тецнер А. Delphi 6 і технологія COM. - СПб.: Пітер, 2002. - 640 с.: іл. - (Серія «Майстер-клас»).
9. Зайдуллін С.С. Інформаційна технологія синтезу та управління експлуатацією мобільних компонентів територіально розподілених систем: Автореферат дисертації на здобуття наукового ступеня канд. техн. наук: 05.13.01. -Казань, 2001. -19 с. - Бібліогр.: с. 16-19 (27 назв.) ДРНТІ 50.49
10. Іванов Д.Є., Скобцов Ю.А. Генерація тестів цифрових пристроїв з використанням генетичних алгоритмів // Праці інституту прикладної математики і механіки НАН України. - Т.4. - Донецьк, ПІММ. - 1999. - С.82-88.

- 11.Калабеков Б.А. Мікропроцесори та їх застосування в системах передачі та обробки сигналів: Навч. посібник для вузів. - М.: Радио и связь, 1988. - 368 с.
- 12.Кузан Д.Я., Шапоров В.Н. Програмування Win32 API в Delphi. - СПб .: БХВ - Петербург, 2005. - 368 с.: іл.
- 13.Ляпунов А.А. Про логічні схеми програм // Проблеми кібернетики. Вип.1. М .: Физматгиз. 1985.
- 14.Мішель Ж. Програмовані контролери. Архітектура і застосування. М .: Машинобудування, 1992.
- 15.Моїсєєв В.С., Дудкін Г.Г., Горбунов Д.А., Зайдуллін С.С.Мобільніє автоматизовані комплекси збору, обробки і передачі інформації при технічному обслуговуванні і моніторингу територіально розподілених систем - Вісник КДТУ ім. А.Н.Туполева, 1997, №3
- 16.Моїсєєв В.С., Зайдуллін С.С. Завдання аналізу та синтезу складних територіально розподілених систем - Вісник КДТУ ім. А.Н.Туполева, 2000., №2
- 17.Моїсєєв В.С., Зайдуллін С.С. Моделювання динаміки кінцевих дискретних множин - Вісник КДТУ ім. А.Н.Туполева, 2003 №4
- 18.Моїсєєв В.С., Комаров Ю.Л., Зайдуллін С.С., Рахматуллін А.І. Методика аналізу систем обслуговування виробів в польових умовах - Вісник КДТУ ім. А.Н.Туполева, 1999, №3
- 19.Ненашев А.П. Конструювання радіоелектронних засобів: Навч. посібник для радіотехніч. спец. вузів. - М .: Вища. шк., 1990. - 432 с.
- 20.Паризький С.М. Вчимося на прикладах / Под ред. Ю.А. Шпака - К: «МК-Пресс», 2005. - 216 с.: іл.
- 21.C. Pixley, S. Jeong, G. Hatchel, "Exact Calculation of Synchronization Sequences based on Binary Decision Diagrams", IEEE Trans. on CAD, Vol.13, pp.1024-1034, 1994.
- 22.Скобцов Ю.А., Скобцов В.Ю. Логічне моделювання і тестування цифрових пристроїв. - Донецьк, 2005.

- 23.Стівенс Р. Delphi. Готові алгоритми: Пер. з англ. - М.: ДМК Пресс, 2001. - 384 с. : іл.
- 24.Технологія і автоматизація виробництва радіоелектронної апаратури: Навч. для вузів / І.П. Бушмінській, О.Ш. Даутов, А.П. Достанко і ін.; Під ред. А.П. Достанко. - М.: Радио и связь, 1989. - 624 с.
- 25.Фаронов В.В. Delphi 6. Навчальний курс. - М.: Видавець Молгачева С.В., 2003. - 672 с., Іл.
26. Фаронов В.В., Шумаков П.В. Delphi 5. Керівництво розробника баз даних - М.: «Нолидж», 2001. - 640, мул.
27. Федеоров А., Єлманова Н. ADO в Delphi: Пер. з англ. - СПб.: БХВ-Петербург, 2002. - 816 с. іл.
- 28.Філд А., Харрісон П. Функціональне програмування. М.: МИР, 1993.
- 29.Фленов М.Є. Біблія Delphi. - СПб. : БХВ - Петербург, 2004. - 880 с.: іл.
- 30.Цифрові і аналогові інтегральні мікросхеми: Довідник С.В. Якубовський, Л.І. Ніссельсон, В.І.Кулешова і ін.; Під ред. С.В.Якубовського. - М.: Радио и связь, 1988. - 363 с.

## ДОДАТОК А

### Лістинг вихідного коду розробленого додатка

```
unit Unit2;
interface
const
Const0=1;Const1=2;ConstU=0;ConstC=3;ConU=0;ConHp=2;ConLp=1;
type
masZnZ=record
MasIni:array of array of LongWord;
VariantZnac:array of LongWord;
end;
mas=record
v0:LongWord;
v1:LongWord;
end;
b=^a;
a=record
pred,next:b;
data:integer;
end;
TBoolMass=0..3;
TElementType=0..9;
InnerMass=array of array of TBoolMass;
TInputElements=array of array of byte;
TElement=record
Name:String;
EType:TElementType;
TriggerRank:word;
Inputs:Word;
Operands:array of Longword;
src:array of Longword;
outputs:array of Longword;
Rank:integer;
end;
TMicro=class
private
```

```

FFaultFreeArr:InnerMass;
FFaultArr:InnerMass;
FCurrentRow,FRowCount:word;
FOUTPUTS:array of integer;
FINPUTS:array of integer;
FTRIG:array of integer;
FElementCount:LongWord;
Glubina:word;
FProbArr:array of array of TElement;
constructor Create(FileName:string);
public
    InputCount:word;
    OutputCount:word;
    TriggerCount:word;
    MAX_RANK:word;
    ElementCount:integer;
    FEIArr:array of TElement;
MasVhod:array of array of mas;
MasVih:array of mas;
MasVihR:array of array of LongWord;
KolPVih:LongWord;
KolUnVih:LongWord;
MasVhVih:array of array of LongWord;
masZn:array of array of masZnZ;
KolTrig:array of integer;
fileVV:TextFile;
function SetBit(Src: Integer; bit: Integer): Integer;
procedure add_elem(first:b;elem:integer);
function del_elem(first,num:b):b;
procedure Initialize(CurScheme:word);
procedure CalcSetupSignal;
implementation
uses Classes, SysUtils, math, dialogs, windows, unit1;
constructor TMicro.Create(FileName:string);
var
q,w,e,r,newi:integer;

```

```

sl:TStringList;
i,j,n,k,CommaPos:Word;
s:String;
ElN,OperName,OperType:string;
El:TElement;
ic,oc:word;
b:boolean;
DigitCount:byte;
ElNo,ii:longword;
CurTrigRank,MaxTrigRank,CurRank:word;
CurElem:word;
MinRank:longword;
InputMass:array of word;
begin
ic:=0; oc:=0;
sl:=TStringList.Create;
sl.LoadFromFile(FileName);
i:=0;n:=0;
while i<sl.Count do
begin
s:=sl.Strings[i];
s:=TrimLeft(s);
if (Length(s)=0)or (s[1]='#' )then
begin
sl.Delete(i);
continue;
end;
if pos('INPUT',s)>0 then
begin
ElN:=copy(s,pos('(' ,s)+1, pos(')',s)-pos('(' ,s)-1);
El.Name:=ElN;
El.EType:=0;
El.Inputs:=n;
n:=Length(FElArr);
SetLength(FElArr,n+1);
FElArr[n]:=El;

```



```

k:=Length(FINPUTS);
SetLength(FINPUTS,k+1);
FINPUTS[k]:=n;
inc(ic);
sl.Delete(i);
continue;
end;
if pos('OUTPUT',s)>0 then
begin
  ELN:=copy(s,pos('('s)+1, pos(')',s)-pos('('s)-1);
  El.Name:=ElN+'# OUTPUT';
  El.EType:=1;
  El.Inputs:=n;
  n:=Length(FElArr);
  SetLength(FElArr,n+1);
  FElArr[n]:=El;
  inc(i);
  k:=Length(FOUTPUTS);
  SetLength(FOUTPUTS,k+1);
  FOUTPUTS[k]:=n;
  inc(oc);
  continue;
end;
ELN:=Trim(copy(s,1,pos('='s)-1));
El.Name:=ElN;
El.EType:=ElemType(trim(copy(s,pos('='s)+1,pos('('s)-pos('='s)-1)));
inc(i);
El.Inputs:=n;
n:=Length(FElArr);
SetLength(FElArr,n+1);
FElArr[n]:=El;
if El.EType=2 then inc(TriggerCount);
InputCount:=ic;
OutputCount:=oc;
end;
for i:=0 to sl.Count-1 do

```

```

begin
  s:=trim(sl[i]);
  n:=CommaCount(s)+1;
  ElN:=Trim(copy(s,1,pos('=',s)-1));
  if ElN="" then
    if pos('OUTPUT',s)>0 then
      begin
        for ii:=low(FElArr) to high(FElArr) do
          if (FElArr[ii].EType=1)and(length(FElArr[ii].Operands)=0)then
            begin
              ElN:=copy(FElArr[ii].Name,1,pos('#',FElArr[ii].Name)-1);
              break;
            end;
          ElNo:=FindElementNoByName(ElN);
          SetLength(FElArr[ii].Operands,1);
          FElArr[ii].Operands[0]:=ElNo;
          continue
        end;
        ElNo:=FindElementNoByName(ElN);
        SetLength(FElArr[ElNo].Operands,n);
        Delete(s,1,pos('(',s));
        s:=TrimLeft(s);
        Delete(s,length(s),1);
        for j:=0 to n-1 do
          begin
            CommaPos:=pos(',',s);
            if CommaPos>0 then
              begin
                OperName:=copy(s,1, CommaPos-1);
                Delete(s,1,CommaPos);
                s:=TrimLeft(s);
              end
            else
              OperName:=TrimLeft(s);
            FElArr[ElNo].Operands[j]:=FindElementNoByName(OperName);
            FElArr[ElNo].TriggerRank:=0;

```

```

    end;
end;
FElementCount:=Length(FElArr);
ElementCount:=FElementCount;
SetLength(FFaultArr,FElementCount);
SetLength(FFaultFreeArr,FElementCount);
SetLength(FFaultList,FElementCount*2);
for i:=0 to FElementCount*2-1 do
    begin
        FFaultList[i].Checked:=false;
        FFaultList[i].Element:= i div 2;
        FFaultList[i].FaultType:= i mod 2;
        FFaultList[i].SourcePos:=false;
    end;
FStartFrom:=InputCount+OutputCount;
for i:=0 to FElementCount-1 do
    FElArr[i].Rank:=-1;
MAX_RANK:=0;
for i:=low(Finputs) to high(Finputs) do
    FElArr[i].Rank:=0;
b:=true;
while b do
    Begin
        for i:=0 to FElementCount-1 do
            begin
                if FElArr[i].Rank>-1 then
                    begin
                        continue;
                    end;
                if (Length(FElArr[i].Operands)>0) then
                    begin
                        MinRank:=1000000;
                        for j:=low(FElArr[i].operands) to high(FElArr[i].operands) do
                            if FElArr[FElArr[i].Operands[j]].Rank>=0 then
                                MinRank:=Min(MinRank,FElArr[FElArr[i].Operands[j]].Rank);
                            if MinRank<>1000000 then

```

```

    if FEIArr[i].EType<>1 then
        FEIArr[i].Rank:=MinRank+1
    else
        FEIArr[i].Rank:=MinRank;
    if FEIArr[i].Rank>MAX_RANK then
        MAX_RANK:=FEIArr[i].Rank;
    end;
end;
b:=false;
for i:=0 to FElementCount-1 do
    if FEIArr[i].Rank=-1 then b:=true
end;
for j:=FStartFrom to FElementCount-1 do
    if (FEIArr[j].EType=2) then
        begin
            n:=Length(FTRIG);
            SetLength(FTRIG,n+1);
            FTRIG[n]:=j;
        end;
MaxTrigRank:=0;
b:=false;
for i:=0 to FElementCount-1 do
    if (FEIArr[i].Rank=MAX_RANK)then
        begin
            CurRank:=CalcElementTriggerRank(i);
            If CurTrigRank=0 then
                CurTrigRank:=CurRank
            else
                CurTrigRank:=max(CurRank,CurTrigRank)
            end;
if CurTrigRank=0 then
    CurTrigRank:=1;
Glubina:=CurTrigRank;
SetLength(FFaultTrig,TriggerCount);
for i:=0 to TriggerCount-1 do
    begin

```

```

    FFaultTrig[i]:=0;
end;
if InputCount=1 then
    for i:=0 to FElementCount-1 do
        begin
            SetLength(FElArr[i].src,1);
            FElArr[i].src[1]:=FInputs[0];
        end
    else
        begin
            b:=true;
            CurTrigRank:=0;
            for i:=low(FINPUTS) to high(FINPUTS) do
                begin
                    SetLength(FElArr[FINPUTS[i]].src,1);
                    FElArr[FINPUTS[i]].src[0]:=FINPUTS[i];
                end;
            while b do
                begin
                    b:=false;
                    for i:=low(FINPUTS) to high(FINPUTS) do
                        begin
                            CurTrigRank:=FINPUTS[i];
                            for j:=1 to MAX_RANK do
                                for ii:=FStartFrom to FElementCount-1 do
                                    if FElArr[ii].Rank=j then
                                        if not(CheckInputInSrc(CurTrigRank,ii))and
CheckInputInOperands(CurTrigRank,ii) then
                                            begin
                                                AddInputToSrc(CurTrigRank,ii);
                                                b:=true;
                                            end;
                                        end;
                                    end;
                                end;
                            end;
                        for ii:=FStartFrom to FElementCount-1 do
                            if length(FElArr[ii].Operands)>0 then
                                begin

```

```

    SetLength(InputMass,0);
    for i:=low(FElArr[ii].Operands) to high(FElArr[ii].Operands) do
        if length(FElArr[FElArr[ii].Operands[i]].src)>0 then
            for j:=low(FElArr[FElArr[ii].Operands[i]].src) to
high(FElArr[FElArr[ii].Operands[i]].src) do
                if not(InMass(FElArr[FElArr[ii].Operands[i]].src[j],InputMass)) then
                    begin
                        CurTrigRank:=length(InputMass);
                        SetLength(InputMass,CurTrigRank+1);
                        InputMass[CurTrigRank]:=FElArr[FElArr[ii].Operands[i]].src[j];
                    end;
                end;
            end;
        end;
    end;
    AddBranchesFault;
    FaultCount:=Length(FFaultList);
    for q:=low(FElArr) to high(FElArr) do
        for w:=low(FElArr[q].Operands) to high(FElArr[q].Operands) do begin
            e:=length(FElArr[FElArr[q].Operands[w]].outputs);
            Setlength(FElArr[FElArr[q].Operands[w]].outputs,e+1);
            FElArr[FElArr[q].Operands[w]].outputs[e]:=q;
        end;
    end;
    Glubina:=0;
    for i:=0 to FElementCount-1 do
        begin
            SetLength(FFaultFreeArr[i],1);
        end;
    end;
    for newi:=0 to FElementCount-1 do
        FFaultFreeArr[newi][1]:=ConU;
    end;
    repeat
        q:=0;
        for newi:=low(FInputs) to high(FInputs) do
            FFaultFreeArr[FINPUTS[newi]][Glubina]:=ConHp;
            FCurrentRow:=Glubina;
            for i:=low(Ftrig) to High(FTRIG) do
                IniDFF(Ftrig[i]);
                for ii:=1 to MAX_RANK do
                    for i:=FStartFrom to FElementCount-1 do

```

```

    if (FEIArr[i].Rank<=ii) then
        case FEIArr[i].EType of
            3: IniNot(i);
            4,5,6,7,8,9: IniAnd(i);
        end;
        for i:=0 to OutputCount-1 do
            begin
FFaultFreeArr[FOUTPUTS[i]][FCurrentRow]:=FFaultFreeArr[FEIArr[FOUTPUTS[i]].Opera
nds[0]][FCurrentRow];
                end;
                for i:=low(Ftrig) to High(FTRIG) do
                    IniDFF(Ftrig[i]);
                    for newi:=low(FFaultFreeArr) to high(FFaultFreeArr) do
                        if FFaultFreeArr[newi][Glubina]=ConU then q:=1;
                        if q=1 then Glubina:=Glubina+1;
                    until q=0;
                    FaultDetected:=0;
                end;
            procedure TMicro.CalcSetupSignal;
            var
                newi,i,ii,iii,j,ActivatedCount:Longword;
                TempMass:array of integer;
                TempK,PseudoCount:integer;
                a,q,w,f:Longword;
                timest,timeen:integer;
                timere:real;
            begin
                timest:=GetTickCount();
                ConUInput.EType:=0;
                ConUInput.Name:='DopElement';
                FRowCount:=1;
                SetLength(FProbArr,Glubina);
                for i:=0 to FElementCount-1 do
                    begin
                        SetLength(FFaultFreeArr[i],FRowCount);
                    end;
                for newi:=0 to FElementCount-1 do

```

```

    FFaultFreeArr[newi][FRowCount]:=ConU;
for j:=0 to Glubina-1 do
begin
for newi:=low(FInputs) to high(FInputs) do
    FFaultFreeArr[FINPUTS[newi]][j]:=ConHp;
    SetLength(TempMass,0);
    FCurrentRow:=j;
    for i:=low(Ftrig) to High(FTRIG) do
        IniDFF(Ftrig[i]);
        for ii:=1 to MAX_RANK do
            for i:=FStartFrom to FElementCount-1 do
                if (FEIArr[i].Rank<=ii) then
                    case FEIArr[i].EType of
                        3: IniNot(i);
                        4,5,6,7,8,9: IniAnd(i);
                    end;
                    for i:=0 to OutputCount-1 do
                        begin
FFaultFreeArr[FOUTPUTS[i]][FCurrentRow]:=FFaultFreeArr[FEIArr[FOUTPUTS[i]].Operands[0]][FCurrentRow];
                        end;
                    for i:=low(Ftrig) to High(FTRIG) do
                        IniDFF(Ftrig[i]);
ActivatedCount:=InputCount+1;
                    for i:=FStartFrom to FElementCount-1 do
                        if (FFaultFreeArr[i][FCurrentRow]<>ConU)then
                            inc(ActivatedCount);
                    SetLength(FProbArr[j],ActivatedCount);
                    SetLength(TempMass,FElementCount);
                    for i:=0 to InputCount-1 do
                        begin
                            FProbArr[j][i].Name:=FEIArr[i].Name;
                            FProbArr[j][i].EType:=FEIArr[i].EType;
                            FProbArr[j][i].TriggerRank:=FEIArr[i].TriggerRank;
                            FProbArr[j][i].Inputs:=FEIArr[i].Inputs;
                            if length(FEIArr[i].Operands)>0 then
                                begin

```



```

    Setlength(FProbArr[j][i].Operands,length(FElArr[i].Operands));
    for f:=low(FElArr[i].Operands) to high(FElArr[i].Operands) do
        FProbArr[j][i].Operands[f]:=FElArr[i].Operands[f];
    end;
    if length(FElArr[i].src)>0 then
        begin
            Setlength(FProbArr[j][i].src,length(FElArr[i].src));
            for f:=low(FElArr[i].src) to high(FElArr[i].src) do
                FProbArr[j][i].src[f]:=FElArr[i].src[f];
            end;
        end;
    if length(FElArr[i].outputs)>0 then
        begin
            Setlength(FProbArr[j][i].outputs,length(FElArr[i].outputs));
            for f:=low(FElArr[i].outputs) to high(FElArr[i].outputs) do
                FProbArr[j][i].outputs[f]:=FElArr[i].outputs[f];
            end;
        end;
    FProbArr[j][i].Rank:=FElArr[i].Rank;
end;
a:=InputCount+1;
FProbArr[j][InputCount]:=ConUInput;
if j=0 then PseudoCount:=0;
    w:=length(FProbArr[j]);
    Setlength(FProbArr[j],w-PseudoCount);
    PseudoCount:=0;
    for q:=low(FTrig) to high(FTrig) do
        if FFaultFreeArr[FElArr[FTrig[q]].Operands[0]][FCurrentRow]<>ConU
        then
            begin
                w:=length(FProbArr[j]);
                Setlength(FProbArr[j],w+1);
                if FElArr[FTrig[q]].TriggerRank=0 then FElArr[FTrig[q]].TriggerRank:=j+1;
                FProbArr[j][a].Name:=FElArr[FTrig[q]].Name;
                FProbArr[j][a].EType:=FElArr[FTrig[q]].EType;
                FProbArr[j][a].TriggerRank:=FElArr[FTrig[q]].TriggerRank;
                FProbArr[j][a].Inputs:=FElArr[FTrig[q]].Inputs;
                if length(FElArr[FTrig[q]].Operands)>0 then

```

```

begin
  Setlength(FProbArr[j][a].Operands,length(FElArr[FTrig[q]].Operands));
  for f:=low(FElArr[FTrig[q]].Operands) to high(FElArr[FTrig[q]].Operands) do
    FProbArr[j][a].Operands[f]:=FElArr[FTrig[q]].Operands[f];
  end;
if length(FElArr[FTrig[q]].src)>0 then
  begin
    Setlength(FProbArr[j][a].src,length(FElArr[FTrig[q]].src));
    for f:=low(FElArr[FTrig[q]].src) to high(FElArr[FTrig[q]].src) do
      FProbArr[j][a].src[f]:=FElArr[FTrig[q]].src[f];
    end;
  if length(FElArr[FTrig[q]].outputs)>0 then
    begin
      Setlength(FProbArr[j][a].outputs,length(FElArr[FTrig[q]].outputs));
      for f:=low(FElArr[FTrig[q]].outputs) to high(FElArr[FTrig[q]].outputs) do
        FProbArr[j][a].outputs[f]:=FElArr[FTrig[q]].outputs[f];
      end;
    FProbArr[j][a].Rank:=FElArr[FTrig[q]].Rank;
    inc(PseudoCount);
    inc(a);
  end;
for i:=InputCount to FElementCount-1 do
  if (FFaultFreeArr[i][FCurrentRow]<>ConU) and (FElArr[i].EType<>2) then begin
    if FElArr[i].EType=1 then begin
      w:=length(FProbArr[j]);
      Setlength(FProbArr[j],w+1);
    end;
    FProbArr[j][a].Name:=FElArr[i].Name;
    FProbArr[j][a].EType:=FElArr[i].EType;
    FProbArr[j][a].TriggerRank:=FElArr[i].TriggerRank;
    FProbArr[j][a].Inputs:=FElArr[i].Inputs;
    if length(FElArr[i].Operands)>0 then
      begin
        Setlength(FProbArr[j][a].Operands,length(FElArr[i].Operands));
        for f:=low(FElArr[i].Operands) to high(FElArr[i].Operands) do
          FProbArr[j][a].Operands[f]:=FElArr[i].Operands[f];
        end;
      end;
    end;
  end;
end;

```

```

end;
if length(FElArr[i].src)>0 then
begin
  Setlength(FProbArr[j][a].src,length(FElArr[i].src));
  for f:=low(FElArr[i].src) to high(FElArr[i].src) do
    FProbArr[j][a].src[f]:=FElArr[i].src[f];
  end;
if length(FElArr[i].outputs)>0 then
begin
  Setlength(FProbArr[j][a].outputs,length(FElArr[i].outputs));
  for f:=low(FElArr[i].outputs) to high(FElArr[i].outputs) do
    FProbArr[j][a].outputs[f]:=FElArr[i].outputs[f];
  end;
FProbArr[j][a].Rank:=FElArr[i].Rank;
if TempMass[i]=0 then TempMass[i]:=a;
inc(a);
end
end;
if (high(FElArr)+1)<>high(FProbArr[Glubina-1]) then
begin
  Glubina:=Glubina+1;
  SetLength(FProbArr,Glubina);
  SetLength(FProbArr[Glubina-1],length(FElArr)+1);
  for i:=0 to high(FElArr) do
    FProbArr[Glubina-1,i]:=FElArr[i];
    FProbArr[Glubina-1,high(FElArr)+1]:=ConUInput;
  end;
Setlength(KolTrig,Glubina);
for i:=0 to Glubina-1 do
  KolTrig[i]:=0;
for i:=Low(FProbArr) to High(FProbArr) do
  for j:=Low(FProbArr[i]) to High(FProbArr[i]) do
    if (FProbArr[i,j].EType=2) and (FProbArr[i,j].TriggerRank=i+1) then
      KolTrig[i]:=KolTrig[i]+1;
    for i:=0 to Glubina-1 do
      for j:=0 to high(FProbArr[i]) do
        begin

```

```

if length(FProbArr[i,j].Operands)>0 then
  for w:=low(FProbArr[i,j].Operands) to high(FProbArr[i,j].Operands) do
    begin
      q:=0;
      for a:=low(FProbArr[i]) to high(FProbArr[i]) do
        if q=0 then
          if FELArr[FProbArr[i,j].Operands[w]].Name=FProbArr[i,a].Name then begin
            FProbArr[i,j].Operands[w]:=a;
            q:=1;
          end;
        end;
      end;
    if length(FProbArr[i,j].outputs)>0 then
      for w:=low(FProbArr[i,j].outputs) to high(FProbArr[i,j].outputs) do
        begin
          q:=0;
          for a:=0 to high(FProbArr[i]) do
            if q=0 then
              if FELArr[FProbArr[i,j].outputs[w]].Name=FProbArr[i,a].Name then begin
                FProbArr[i,j].outputs[w]:=a;
                q:=1;
              end;
            end;
          end;
        end;
      end;
    if InputCount<20 then begin
      for i:=0 to Glubina-1 do
        initialize(i);
        timeen:=GetTickCount();
        timere:=(timeen-timest)/1000;
        AssignFile(fileVV, 'fileVV.txt');
        Rewrite(fileVV);
        for i:=0 to Glubina-1 do
          if KolTrig[i]<>0 then
            begin
              writeln(fileVV, '      '+IntToStr(i+1)+' подcхема:');
              writeln(fileVV, '*****');
              for j:=0 to high(masZn[i]) do

```

```

begin
  writeln(fileVV, '  '+IntToStr(j+1)+' последовательность:');
  writeln(fileVV, '-----');
  writeln(fileVV, 'значения на входах:');
  w:=high(masZn[i,j].MasIni);
  for a:=0 to w do begin
    for q:=0 to InputCount-1 do
      write(fileVV, IntToStr(masZn[i,j].MasIni[a,q]));
      writeln(fileVV, ' ');
    end;
    writeln(fileVV, 'значения на псевдовыходах:');
    for a:=0 to high(masZn[i,j].VariantZnac) do
      write(fileVV, IntToStr(masZn[i,j].VariantZnac[a]));
      writeln(fileVV, ' ');
    end;
    writeln(fileVV, '-----');
  end;
  writeln(fileVV, '*****');
  writeln(fileVV, ' ');
end;
writeln(fileVV, 'Время расчета = '+FloatToStr(timere)+' секунд');
CloseFile(fileVV);
end
else ShowMessage('Программа не сможет обработать данную схему');
end;

procedure TMicro.Initialize(CurScheme:word);
var
  y,t,h,v,x,z,p,i,j,k,c,a,q,m,mm,n,KolElOc,KolVarVh:integer;
  kk:double;
  s:string;
  mask:longword;
  start,qwe,asd:b;
  znavih:mas;
  MasVhodN:array of array of longword;
  MasVh:array of array of word;
  MasNomUnTr:array of integer;

```

```

MasVhodVrem:array of array of mas;
begin
mask:=0;
n:=InputCount;
m:=round(power(2,n));
Setlength(MasVh,m,n);
q:=m;
for j:=0 to n-1 do
begin
q:=round(q/2);
p:=0;
k:=-1;
for i:=0 to m-1 do
begin
k:=k+1;
if k=q then if p=0 then begin
p:=1;
k:=0;
end
else begin
p:=0;
k:=0;
end;
MasVh[i,j]:=p;
end;
end;
Setlength(MasVhod,0,0);
Setlength(MasVhodN,0,0);
if CurScheme<>0 then
begin
KolVarVh:=m;
for i:=0 to CurScheme-1 do
KolVarVh:=KolVarVh*length(masZn[i]);
Setlength(MasVhodN,KolVarVh,n+KolPVih);
q:=round(KolVarVh/m);
k:=-1;

```

```

p:=0;
for i:=0 to KolVarVh-1 do
begin
k:=k+1;
for j:=0 to n-1 do
begin
if k=q then begin
if p=m-1 then p:=0
else p:=p+1;
k:=0;
end;
MasVhodN[i,j]:=MasVh[p,j];
end;
end;
a:=n;
for c:=0 to CurScheme-1 do
begin
q:=round(q/length(masZn[c]));
k:=-1;
p:=0;
for i:=0 to KolVarVh-1 do
begin
k:=k+1;
for j:=a to a+length(masZn[c,0].VariantZnac)-1 do
begin
if k=q then begin
if p=length(masZn[c])-1 then p:=0
else p:=p+1;
k:=0;
end;
MasVhodN[i,j]:=masZn[c,p].VariantZnac[j-a];
end;
end;
a:=a+length(masZn[c,0].VariantZnac);
end;
kk:=KolVarVh/32;

```

```

kk:=kk+1-frac(kk);
mm:=round(kk);
Setlength(MasVhod,mm,n);
for i:=0 to mm-1 do
  for j:=0 to n-1 do
    begin
      MasVhod[i,j].v0:=0;
      MasVhod[i,j].v1:=0;
    end;
m:=KolVarVh;
for i:=0 to m-1 do
  begin
    k:=i div 32;
    c:=i mod 32;
    for j:=0 to n-1 do
      begin
        if MasVhodN[i,j] = 0 then MasVhod[k,j].v0:=SetBit(MasVhod[k,j].v0,c);
        if MasVhodN[i,j] = 1 then MasVhod[k,j].v1:=SetBit(MasVhod[k,j].v1,c);
      end;
    end;
    n:=n+KolPVih;
  end
  else
    begin
      kk:=m/32;
      kk:=kk+1-frac(kk);
      mm:=round(kk);
      Setlength(MasVhod,mm,n);
      Setlength(MasVhodVrem,mm,n);
      //обнуление массива
      for i:=0 to mm-1 do
        for j:=0 to n-1 do
          begin
            MasVhodVrem[i,j].v0:=0;
            MasVhodVrem[i,j].v1:=0;
          end;

```



```

for i:=0 to m-1 do
begin
k:=i div 32;
c:=i mod 32;
for j:=0 to n-1 do
begin
if i and (1 shl j) = 0 then
MasVhodVrem[k,j].v0:=SetBit(MasVhodVrem[k,j].v0,c)
else
MasVhodVrem[k,j].v1:=SetBit(MasVhodVrem[k,j].v1,c);
end;
end;
for i:=0 to mm-1 do
for j:=0 to n-1 do
begin
MasVhod[i,j].v0:=MasVhodVrem[i,n-1-j].v0;
MasVhod[i,j].v1:=MasVhodVrem[i,n-1-j].v1;
end;
Setlength(MasVhodVrem,0,0);
Setlength(MasVhodN,m,n);
for i:=0 to m-1 do
for j:=0 to n-1 do
MasVhodN[i,j]:=MasVh[i,j];
end;
i:=CurScheme;
Setlength(MasVih,High(FProbArr[i])+1);
for j:=Low(FProbArr[i]) to High(FProbArr[i]) do
begin
MasVih[j].v0:=0;
MasVih[j].v1:=0;
end;
KolPVih:=0;
for j:=Low(FProbArr[i]) to High(FProbArr[i]) do
if FProbArr[i,j].EType=2 then KolPVih:=KolPVih+1;
Setlength(MasVihR,0,0);
Setlength(MasVihR,KolPVih,m);

```

```

t:=0;
if mm>1 then y:=32
    else y:=m;
for h:=0 to mm-1 do
begin
    for q:=Low(FINPUTS) to High(FINPUTS) do
    begin
        MasVih[FINPUTS[q]].v0:=MasVhod[h,q].v0;
        MasVih[FINPUTS[q]].v1:=MasVhod[h,q].v1;
    end;
KolElOc:=0;
new(start);
asd:=start;
for j:=Low(FProbArr[i]) to High(FProbArr[i]) do
if FProbArr[i,j].Rank=1 then begin
        KolElOc:=KolElOc+1;
        start^.data:=j;
        new(qwe);
        start^.next:=qwe;
        qwe^.pred:=start;
        start:=qwe;
    end;

start:=start^.pred;
start^.next:=asd;
asd^.pred:=start;
start:=asd;
dispose(qwe);
c:=0;
repeat
j:=start^.data;
znavih.v0:=MasVih[j].v0;
znavih.v1:=MasVih[j].v1;
case FProbArr[i,j].EType of
    2: begin
        if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
            begin

```

```

        MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
        MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
    end
    else begin
        MasVih[j].v0:=MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[InputCount].v1;
    end;
end;
3: begin
//N
    if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v1;
            MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v0;
        end
    else begin
        MasVih[j].v0:=MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[InputCount].v1;
    end;
    mask:=not(MasVih[j].v0 and MasVih[j].v1);
    if mask<>0 then begin
        MasVih[j].v0:=MasVih[j].v0 and mask;
        MasVih[j].v1:=MasVih[j].v1 and mask;
    end;
    mask:=0;
end;
4: begin
//AND
    if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
            MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
        end
    else begin
        MasVih[j].v0:=MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[InputCount].v1;
    end;
end;

```

```

    end;
for a:=Low(FProbArr[i,j].Operands)+1 to High(FProbArr[i,j].Operands) do begin
    if FProbArr[i,j].Operands[a]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[j].v0 or MasVih[FProbArr[i,j].Operands[a]].v0;
            MasVih[j].v1:=MasVih[j].v1 and MasVih[FProbArr[i,j].Operands[a]].v1;
        end
    else begin
        MasVih[j].v0:=MasVih[j].v0 or MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[j].v1 and MasVih[InputCount].v1;
    end;
end;
end;
5: begin
//OR
if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
    begin
        MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
        MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
    end
else begin
        MasVih[j].v0:=MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[InputCount].v1;
    end;
for a:=Low(FProbArr[i,j].Operands)+1 to High(FProbArr[i,j].Operands) do begin
    if FProbArr[i,j].Operands[a]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[j].v0 and MasVih[FProbArr[i,j].Operands[a]].v0;
            MasVih[j].v1:=MasVih[j].v1 or MasVih[FProbArr[i,j].Operands[a]].v1;
        end
    else begin
        MasVih[j].v0:=MasVih[j].v0 and MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[j].v1 or MasVih[InputCount].v1;
    end;
end;
end;
end;

```

```

6: begin
  //NAND
  if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
    begin
      MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
      MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
    end
  else begin
    MasVih[j].v0:=MasVih[InputCount].v0;
    MasVih[j].v1:=MasVih[InputCount].v1;
  end;
  for a:=Low(FProbArr[i,j].Operands)+1 to High(FProbArr[i,j].Operands) do begin
    if FProbArr[i,j].Operands[a]<=high(FProbArr[i]) then
      begin
        MasVih[j].v0:=MasVih[j].v0 or MasVih[FProbArr[i,j].Operands[a]].v0;
        MasVih[j].v1:=MasVih[j].v1 and MasVih[FProbArr[i,j].Operands[a]].v1;
      end
    else begin
      MasVih[j].v0:=MasVih[j].v0 or MasVih[InputCount].v0;
      MasVih[j].v1:=MasVih[j].v1 and MasVih[InputCount].v1;
    end;
  end;
  c:=MasVih[j].v0;
  MasVih[j].v0:=MasVih[j].v1;
  MasVih[j].v1:=c;
  mask:=not(MasVih[j].v0 and MasVih[j].v1);
  if mask<>0 then begin
    MasVih[j].v0:=MasVih[j].v0 and mask;
    MasVih[j].v1:=MasVih[j].v1 and mask;
  end;
  mask:=0;
end;
7: begin
  //NOR
  if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
    begin

```

```

    MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
    MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
end
else begin
    MasVih[j].v0:=MasVih[InputCount].v0;
    MasVih[j].v1:=MasVih[InputCount].v1;
end;
for a:=Low(FProbArr[i,j].Operands)+1 to High(FProbArr[i,j].Operands) do begin
    if FProbArr[i,j].Operands[a]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[j].v0 and MasVih[FProbArr[i,j].Operands[a]].v0;
            MasVih[j].v1:=MasVih[j].v1 or MasVih[FProbArr[i,j].Operands[a]].v1;
        end
    else begin
        MasVih[j].v0:=MasVih[j].v0 and MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[j].v1 or MasVih[InputCount].v1;
    end;
end;
c:=MasVih[j].v0;
MasVih[j].v0:=MasVih[j].v1;
MasVih[j].v1:=c;
mask:=not(MasVih[j].v0 and MasVih[j].v1);
if mask<>0 then begin
    MasVih[j].v0:=MasVih[j].v0 and mask;
    MasVih[j].v1:=MasVih[j].v1 and mask;
end;
mask:=0;
end;
8: begin
    //XOR
    if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
            MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
        end
    else begin

```

```

        MasVih[j].v0:=MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[InputCount].v1;
    end;
    for a:=Low(FProbArr[i,j].Operands)+1 to High(FProbArr[i,j].Operands) do begin
        if FProbArr[i,j].Operands[a]<=high(FProbArr[i]) then
            begin
                MasVih[j].v0:=(MasVih[j].v0 and
                MasVih[FProbArr[i,j].Operands[a]].v0)or(MasVih[j].v1 and
                MasVih[FProbArr[i,j].Operands[a]].v1);
                MasVih[j].v1:=(MasVih[j].v1 and
                MasVih[FProbArr[i,j].Operands[a]].v0)or(MasVih[j].v0 and
                MasVih[FProbArr[i,j].Operands[a]].v1);
            end
        else begin
            MasVih[j].v0:=(MasVih[j].v0 and MasVih[InputCount].v0)or(MasVih[j].v1
            and MasVih[InputCount].v1);
            MasVih[j].v1:=(MasVih[j].v1 and MasVih[InputCount].v0)or(MasVih[j].v0
            and MasVih[InputCount].v1);
        end;
    end;
end;
9: begin
    //NXOR
    if FProbArr[i,j].Operands[0]<=high(FProbArr[i]) then
        begin
            MasVih[j].v0:=MasVih[FProbArr[i,j].Operands[0]].v0;
            MasVih[j].v1:=MasVih[FProbArr[i,j].Operands[0]].v1;
        end
    else begin
        MasVih[j].v0:=MasVih[InputCount].v0;
        MasVih[j].v1:=MasVih[InputCount].v1;
    end;
    for a:=Low(FProbArr[i,j].Operands)+1 to High(FProbArr[i,j].Operands) do begin
        if FProbArr[i,j].Operands[a]<=high(FProbArr[i]) then
            begin
                MasVih[j].v0:=(MasVih[j].v0 and
                MasVih[FProbArr[i,j].Operands[a]].v0)or(MasVih[j].v1 and
                MasVih[FProbArr[i,j].Operands[a]].v1);

```

```

        MasVih[j].v1:=(MasVih[j].v1 and
MasVih[FProbArr[i,j].Operands[a]].v0)or(MasVih[j].v0 and
MasVih[FProbArr[i,j].Operands[a]].v1);
    end
    else begin
        MasVih[j].v0:=(MasVih[j].v0 and MasVih[InputCount].v0)or(MasVih[j].v1
and MasVih[InputCount].v1);
        MasVih[j].v1:=(MasVih[j].v1 and MasVih[InputCount].v0)or(MasVih[j].v0
and MasVih[InputCount].v1);
    end;
end;
c:=MasVih[j].v0;
MasVih[j].v0:=MasVih[j].v1;
MasVih[j].v1:=c;
mask:=not(MasVih[j].v0 and MasVih[j].v1);
if mask<>0 then begin
    MasVih[j].v0:=MasVih[j].v0 and mask;
    MasVih[j].v1:=MasVih[j].v1 and mask;
end;
mask:=0;
end;
end;
if (znnavih.v0<>MasVih[j].v0) or (znnavih.v1<>MasVih[j].v1) then for
a:=Low(FProbArr[i,j].outputs) to High(FProbArr[i,j].outputs) do
    if FProbArr[i,j].outputs[a]<=high(FProbArr[i]) then
begin
add_elem(start,FProbArr[i,j].outputs[a]);
KolElOc:=KolElOc+1;
end;

start:=del_elem(start,start);
KolElOc:=KolElOc-1;
until KolElOc=0;
while start<>nil do
begin
qwe:=start;
start:=start^.next;
dispose(qwe);
end;

```



```

j:=-1;
for a:=Low(FProbArr[i]) to High(FProbArr[i]) do
  if FProbArr[i,a].EType=2 then
    begin
      x:=-1;
      j:=j+1;
      for c:=t to y-1 do
        begin
          x:=x+1;
          if (MasVih[a].v0 and (1 shl x) <> 0) and (MasVih[a].v1 and (1 shl x) = 0) then
            MasVihR[j,c]:=0;
          if (MasVih[a].v0 and (1 shl x) = 0) and (MasVih[a].v1 and (1 shl x) <> 0) then
            MasVihR[j,c]:=1;
          if (MasVih[a].v0 and (1 shl x) = 0) and (MasVih[a].v1 and (1 shl x) = 0) then
            MasVihR[j,c]:=2;
          if (MasVih[a].v0 and (1 shl x) <> 0) and (MasVih[a].v1 and (1 shl x) <> 0) then
            MasVihR[j,c]:=3;
        end;
      end;
      t:=y;
      if m-t<32 then y:=m
        else y:=y+32;
    end;
    p:=0;
    for q:=0 to i do
      begin
        k:=0;
        j:=0;
        for c:=0 to m-2 do
          begin
            j:=c+1;
            if MasVihR[p,c]<>30 then
              begin
                while (j<=m) do
                  begin
                    for a:=p to p+KolTrig[q]-1 do
                      begin

```

```

        if MasVihR[a,c]<>MasVihR[a,j] then k:=1;
    end;
    if k=1 then k:=0
        else begin
            for a:=p to p+KolTrig[q]-1 do
                MasVihR[a,j]:=30;
            end;
            j:=j+1;
        end;
    end;
    end;
    p:=p+KolTrig[q];
    if (q=i-1) and (KolTrig[i]=0) then p:=p-1;
end;
z:=KolPVih;
for p:=i downto 0 do
    if KolTrig[p]<>0 then
        begin
            z:=z-KolTrig[p];
            KolUnVih:=0;
            for q:=0 to m-1 do
                if MasVihR[z,q]<>30 then KolUnVih:=KolUnVih+1;
            Setlength(MasNomUnTr,0);
            Setlength(MasNomUnTr,KolUnVih);
            Setlength(MasVhVih,0,0);
            Setlength(MasVhVih,KolUnVih,KolTrig[p]+2+InputCount);
            a:=-1;
            for c:=0 to m-1 do
                if MasVihR[z,c]<>30 then
                    begin
                        q:=-1;
                        a:=a+1;
                        for j:=0 to InputCount-1 do
                            begin
                                q:=q+1;
                                MasVhVih[a,j]:=MasVhodN[c,j];

```

```

        MasNomUnTr[a]:=c;
    end;
    q:=q+1;
    MasVhVih[a,q]:=2;
    q:=q+1;
    MasVhVih[a,q]:=40;
    for k:=0 to KolTrig[p]-1 do
        begin
            q:=q+1;
            MasVhVih[a,q]:=MasVihR[z+k,c];
        end;
    end;
    if i=p then
        begin
            Setlength(masZn,i+1);
            Setlength(masZn[i],KolUnVih);
            for j:=0 to KolUnVih-1 do
                begin
                    Setlength(masZn[i,j].MasIni,1,InputCount);
                    for q:=0 to InputCount-1 do
                        masZn[i,j].MasIni[0,q]:=MasVhVih[j,q];
                    Setlength(masZn[i,j].VariantZnac,KolTrig[i]);
                    for q:=0 to KolTrig[i]-1 do
                        masZn[i,j].VariantZnac[q]:=MasVhVih[j,q+InputCount+2];
                    end;
                end
            end
        else
            begin
                q:=0;
                for k:=0 to KolUnVih-1 do
                    begin
                        c:=0;
                        for a:=0 to high(masZn[p]) do
                            begin
                                for j:=0 to KolTrig[p]-1 do
                                    if masZn[p,a].VariantZnac[j]=MasVhVih[k,j+InputCount+2] then q:=q+1;

```

```

        if q=KolTrig[p] then c:=1;
        q:=0;
    end;
    if c=0 then
        begin
            Setlength(masZn[p],length(masZn[p])+1);
            c:=length(masZn[p])-1;
            q:=-1;
            repeat
                q:=q+1;
                a:=0;
                for j:=0 to InputCount-1 do
                    if
masZn[p,q].MasIni[high(masZn[p,q].MasIni),j]=MasVhodN[MasNomUnTr[k],j] then
a:=a+1;

                    if a=InputCount then
                        begin
                            for x:=0 to length(masZn[p,q].MasIni)-1 do
                                begin
                                    Setlength(masZn[p,c].MasIni,length(masZn[p,c].MasIni)+1,InputCount);
                                    for v:=0 to InputCount-1 do
                                        masZn[p,c].MasIni[length(masZn[p,c].MasIni)-
1,v]:=masZn[p,q].MasIni[x,v];
                                    end;
                                    q:=c-1;
                                end;
                            until q=c-1;
                            Setlength(masZn[p,c].VariantZnac,KolTrig[p]);
                            for j:=0 to KolTrig[p]-1 do
                                masZn[p,c].VariantZnac[j]:=MasVhVih[k,j+InputCount+2];

                                Setlength(masZn[p,c].MasIni,length(masZn[p,c].MasIni)+1,InputCount);
                                for q:=0 to InputCount-1 do
                                    masZn[p,c].MasIni[length(masZn[p,c].MasIni)-1,q]:=MasVhVih[k,q];
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;

```

```

    end;
end;

function TMicro.SetBit(Src: Integer; bit: Integer): Integer;
begin
    Result := Src or (1 shl Bit);
end;

procedure TMicro.add_elem(first:b;elem:integer);
var
    k,kk,c:b;
begin
    c:=first;
    c:=c^.pred;
    new(k);
    k^.data:=elem;
    k^.pred:=c;
    k^.next:=first;
    kk:=first;
    kk^.pred:=k;
    c^.next:=k;
end;

function TMicro.del_elem(first,num:b):b;
var
    k,kk:b;
begin
    k:=first;
    kk:=nil;
    if k=nil then exit;
    if first.next<>first then
    begin
        while k<>num do
            k:=k^.next;
            if k^.pred=k^.next then begin
                kk:=first^.next;

```

```

                                kk^.next:=kk;
                                kk^.pred:=kk;
                                end
else begin
                                kk:=k^.pred;
                                kk^.next:=k^.next;
                                kk:=k^.next;
                                kk^.pred:=k^.pred;
                                end;
                                end;

dispose(k);
Result:=kk;
end;

```