

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования

ЛУКЬЯНОВИЧ
Александр Сергеевич

**АВТОМАТИЗИРОВАННОЕ ПОСТРОЕНИЕ ТЕСТОВ ДЛЯ КОМБИНА-
ЦИОННЫХ СХЕМ**

Дипломная работа

Научный руководитель:
кандидат технических наук,
доцент А. Е. Люлькин

Допущена к защите

«___» _____ 2020 г.

Зав. кафедрой веб-технологий и компьютерного моделирования
доцент, доктор физико-математических наук В. М. Волков

Минск, 2020

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	4
РЕФЕРАТ.....	5
ВВЕДЕНИЕ	7
1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ	9
2. МЕТОДЫ И АЛГОРИТМЫ МОДЕЛИРОВАНИЯ НЕИСПРАВНОСТЕЙ И ПОСТРОЕНИЯ ТЕСТОВ	11
2.1. Методы построения тестов	11
2.1.1. Метод активизации одномерного пути.....	11
2.1.2. Метод активизации многомерного пути или D-алгоритм..	14
2.1.3. Вероятностный метод.....	18
2.2. Методы моделирования неисправностей	19
2.2.1. Параллельное моделирование неисправностей	19
2.2.2. Дедуктивное моделирование неисправностей.....	20
3. МОДИФИКАЦИЯ МЕТОДОВ И АЛГОРИТМОВ МОДЕЛИРОВАНИЯ НЕИСПРАВНОСТЕЙ И ПОСТРОЕНИЯ ТЕСТОВ	24
3.1. Математическая модель ФСЭ.....	24
3.2. Модификация D-алгоритма для ФСЭ	26
3.3. Параллельное моделирование	30
неисправностей в алфавите $V_2 = \{0,1\}$ для ФСЭ	30
4. АНАЛИЗ РАЗРАБОТКИ CLR-ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ .NET.....	33
4.1 Технологии .NET и CLR.....	33
4.2 Визуальное программирование в Microsoft Visual Studio 2019 .	37
4.3 Проектирование графического интерфейса САПТ КС.....	40
5. ОПИСАНИЕ ПРОГРАММНЫХ СРЕДСТВ	44
5.1. Структура данных подсистемы	44
5.2. Описание основных функций подсистемы анализа и построения тестов вероятностным методом.....	53

5.3. Описание основных функций подсистемы построения тестов регулярным методом.....	59
5.4 Инструкция пользователя	73
ЗАКЛЮЧЕНИЕ	77
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	79
ПРИЛОЖЕНИЕ А	80
ПРИЛОЖЕНИЕ Б	154

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БИС	большая интегральная схема
ДНФ	дизъюнктивная нормальная форма
ИС	интегральная схема
КНФ	конъюнктивная нормальная форма
КС	комбинационная схема
ОС	операционная система
ПЭВМ	персональная электронно-вычислительная машина
САПТ	система автоматизированного построения те- стов
ФСЭ	функционально сложный элемент
ЭВМ	электронно-вычислительная машина

РЕФЕРАТ

В дипломной работе 79 страниц, 19 рисунков, 8 источников, 2 приложения.

Ключевые слова: ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, КОМБИНАЦИОННЫЕ СХЕМЫ, АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ, ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ, ПЛАТФОРМА .NET, .NET ФРЕЙМВОРКИ, CLR, C++.

Дипломная работа посвящена разработке программного комплекса для автоматизированного построения тестов комбинационных (логических) схем на платформе .NET.

Большое место уделено проблеме тестового диагностирования комбинационных схем и подчеркнута необходимость автоматизированного построения тестов.

В дипломной работе рассмотрены вопросы построения тестов с использованием следующих методов тестирования:

- Вероятностный метод
- Регулярный метод

Особенностью дипломной работы является разработка программного комплекса с использованием технологий .NET и CLR-приложений вида Windows Forms Application Visual C++ в среде программирования Microsoft Visual Studio.

В работе излагается технология разработки .NET приложений, подчеркивается практическая польза данной системы.

Предполагается использование разработанного программного средства в учебном процессе и для решения практических задач.

РЭФЕРАТ

У дыпломнай працы 79 старонак, 19 малюнкаў, 8 крыніц, 2 прыкладання.

Ключавыя словы: ІНФАРМАЦЫЙНЫЯ ТЭХНАЛОГІІ, КАМБІНАЦЫЙНЫЯ СХЕМЫ, АЎТАМАТЫЗАВАНАЕ ТЭСЦІРАВАННЕ, ВІЗУАЛЬНАЕ ПРАГРАМІРАВАННЕ, ПЛАТФОРМА .NET, .NET ФРЭЙМВОРКІ, CLR, C++.

Дыпломная праца прысвечана распрацоўцы праграмага комплексу для аўтаматызаванай пабудовы тэстаў камбінацыйных (лагічных) схем на платформе .NET.

Вялікае месца нададзена праблеме тэставага дыягнаставання камбінацыйных схем і падкрэслена неабходнасць аўтаматызаванай пабудовы тэстаў.

У дыпломнай працы разгледжаны пытанні пабудовы тэстаў з выкарыстаннем наступных метадаў тэсціравання:

- Імавернасны метада
- Рэгулярны метада

Асаблівасцю дыпломнай працы з'яўляецца распрацоўка праграмнага комплексу з выкарыстаннем тэхналогій .NET і CLR-праграмм выгляду Windows Forms Application Visual C++ у асяроддзі праграмавання Microsoft Visual Studio.

У працы выкладаецца тэхналогія распрацоўкі .NET праграм, падкрэсліваецца практычная карысць дадзенай сістэмы.

Мяркуюцца выкарыстанне распрацаванага праграмнага сродку ў навучальным працэсе і для вырашэння практычных задач.

ABSTRACT

Thesis: 79 pages, 19 figures, 8 sources, 2 applications.

Keywords: INFORMATION TECHNOLOGIES, COMBINATIONAL CIRCUITRY, AUTOMATED TESTING, VISUAL PROGRAMMING, PLATFORM .NET, .NET FRAMEWORKS, CLR, C++.

The work is devoted to the development of the software package for the automated tests of combinational (logical) circuitry construction on the .NET platform.

Much attention is paid to the problem of combinational circuitry testing and the necessity of automated tests construction is emphasized.

The work deals with the tests construction using the following testing methods:

- Probabilistic method
- Regular method

A feature of the work is the development of the software package using .NET technologies and CLR applications like Windows Forms Application Visual C++ in the Microsoft Visual Studio programming environment.

The work describes the technology for developing .NET applications and emphasizes the practical benefits of this system.

The final developed software is supposed to be used in educational process and practical problem solving.

ВВЕДЕНИЕ

Характерной чертой современного этапа развития средств вычислительной техники является непрерывное увеличение функциональных возможностей и дальнейшее усложнение структуры цифровых схем. Ведь мы становимся на шаг ближе с цифровым миром каждый день. Цифровые схемы присутствуют абсолютно в любой электронике. В связи с этим, неизбежно нарастание сложности цифровых схем, что предопределяет повышенные требования к их надежности. Достижение высокого уровня надежности обеспечивается рядом технологических, эксплуатационных и организационных мероприятий. Среди большого их многообразия можно выделить проведение тестового диагностирования (т.е. проверка устройства путем подачи на его вход специальных воздействий, называемых тестовыми, и анализ выходных реакций).

При разработке методов контроля логических схем необходимо решить ряд задач:

- 1) выделяются наиболее характерные неисправности логической схемы, возникающие при ее изготовлении и эксплуатации;
- 2) выполняется моделирование логических схем с целью проверки правильности их функционирования и обнаружения критических состязаний под воздействием заданных тестов;
- 3) определяется полнота тестов, т. е. вычисляется отношение числа проверяемых неисправностей выбранного класса к их общему числу в данной схеме;
- 4) выполняется автоматическое построение теста для заданной схемы;
- 5) производится минимизация тестов с сохранением их полноты;
- 6) выполняется локализация дефектных элементов;
- 7) разрабатываются универсальные средства для проверки логических схем и диагностики их неисправностей.

В настоящее время опубликованы различные методы и алгоритмы решения перечисленных выше задач [1, 6, 8]. Программные реализации этих алгоритмов объединяются в автоматизированные системы синтеза тестов. Однако нельзя сказать, что эти системы полностью решают все проблемы контроля логических схем. Например, до сих пор нет удовлетворительного алгоритма моделирования логических схем широкого класса с учетом разброса задержек элементов [8]. Для определения полноты тестов в настоящее время используются различные методы моделирования неисправностей, в частности параллельное, дедуктивное и конкурентное моделирование неисправностей. Однако, обработка БИС, содержащих десятки и сотни тысяч вентилях, требует при любом методе чрезмерных затрат машинного времени. Поэтому зачастую приходится довольствоваться

приближенными оценками полноты тестов, которые можно получить статистическими методами.

Наиболее сложной из перечисленных задач является автоматизация синтеза тестов. Большинство проблем, связанных с автоматизированным проектированием тестов для сложных цифровых схем, обусловлено применением в них элементов большой интеграции или функционально сложных элементов. Примерами таких цифровых схем могут служить программируемые БИС, элементы микропроцессорных комплектов, специальные процессоры и др. С целью расширения области применения имеющихся методов и программных средств построения тестов используется представление функционально сложных элементов в виде эквивалентных им схем из более простых элементов.

Даже при наличии алгоритмов, которые гарантируют нахождение теста, если таковой существует, можно построить такие схемы, для которых не удастся найти тест за приемлемое время. В связи с вышеизложенным и в виду широкого распространения БИС, актуальность задачи автоматизированного построения тестов неуклонно растет.

Ценность работы также заключена в разработке САПТ именно с использованием технологии .NET. Обусловлена в первую очередь тем, что приложение, написанное на любом .NET-совместимом языке, является межплатформенным. Приложение, написанное, скажем, на том же C#, не зависит от платформы, на которой будет выполняться, но зато зависит от наличия платформы .NET.

В данной работе решаются следующие задачи:

1. Изучить математические модели и основные методы решения задач тестового диагностирования для комбинационных схем (КС).
2. Изучить структуры данных, алгоритмы и их программную реализацию в системе автоматизированного построения тестов для КС (САПТ КС).
3. Исследовать особенности CLR-приложений и разработать САПТ КС на платформе .NET.

1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Неисправность в технике – нарушение работоспособности логической схемы вследствие недопустимого изменения ее параметров или свойств под влиянием внутренних физико-химических процессов и внешних механических, климатических или иных воздействий. Мы под неисправностью будем понимать математическую модель какого-либо физического дефекта логической схемы.

Неисправности в цифровых устройствах появляются в результате применения неисправных компонент; разрывов и замыканий межкомпонентных соединений; из-за нарушения условий эксплуатации схемы, наличия ошибок при проектировании и производстве и ряда других факторов.

Из множества различных видов неисправностей выделяется класс логических неисправностей. Логическая неисправность – это такая неисправность, которая изменяет логические функции элементов схемы. В данной работе рассматриваются такие неисправности.

Для описания неисправностей используются различные математические модели. Наиболее часто применяемой является модель константных неисправностей. В такой модели неисправность приводит к тому, что в некоторой линии схемы реализуется логическая функция $\equiv 1$ либо $\equiv 0$. Таким образом, число возможных неисправностей в схеме равно $2 \cdot [l + \sum_{i=1}^n (s_i + t_i)]$, где l – число входов схемы, s_i – число входов i -го элемента, t_i – число выходов i -го элемента.

Будем рассматривать комбинационную схему S содержащую k элементов и имеющую n входов и m выходов.

Пусть $X = \{x_1, x_2, \dots, x_n\}$ множество булевых переменных, соответствующих входам схемы; $Z = \{z_1(x_1, x_2, \dots, x_n), z_2(x_1, x_2, \dots, x_n), \dots, z_m(x_1, x_2, \dots, x_n)\}$ – система функций, реализуемых схемой; $F = \{f_1, f_2, \dots, f_r\}$ – множество одиночных константных неисправностей, возможных в схеме; $G = \{g_1, g_2, \dots, g_k\}$ – множество элементов схемы.

Входной набор X является тестом для неисправности f , если при подаче на схему набора X , выходной набор, реализуемый схемой с неисправностью f , отличается от выходного набора, реализуемого исправной схемой.

Множество входных наборов R называется тестом для множества неисправностей F , если для каждой неисправности f из F существует, по крайней мере, один набор из R , являющийся для нее тестом.

Пример. Пусть схема состоит из одного элемента «ИЛИ».

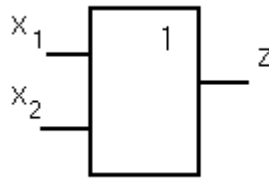


Рис. 1. Функциональный элемент «ИЛИ»

Запишем таблицу значений функций, реализуемой данной схемой с всевозможными константными неисправностями. Булевы переменные x_1 , x_2 соответствуют входам схемы, Z^{f_i} – значение функции, реализуемой схемой с неисправностью f_i ($i = 1, 2, \dots, 6$), Z^{f_0} – функция, реализуемая исправной схемой.

Таблица 1

Таблица значений функций элемента «ИЛИ»

x_1	x_2	Z^{f_0}	Z^{f_1}	Z^{f_2}	Z^{f_3}	Z^{f_4}	Z^{f_5}	Z^{f_6}
0	0	0	0	0	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	1	0	1	1	1	1	0
1	1	1	1	1	1	1	1	0

Тестом для неисправности f_1 может служить входной набор $x = (x_1, x_2) = (1, 0)$. Рассмотрим множество неисправностей $F = \{f_1, f_4, f_5\}$. Тогда множество наборов $\{(1, 0), (0, 0)\}$ будет тестом для множества неисправностей F . Действительно, набор $(1, 0)$ является тестом для f_1 , а набор $(0, 0)$ тест для f_4 и f_5 .

Под полнотой теста будем понимать отношение числа обнаруживаемых им неисправностей к общему их числу.

Длина теста – количество входных наборов, включенных в тест.

2. МЕТОДЫ И АЛГОРИТМЫ МОДЕЛИРОВАНИЯ НЕИСПРАВНОСТЕЙ И ПОСТРОЕНИЯ ТЕСТОВ

2.1. Методы построения тестов

2.1.1. Метод активизации одномерного пути

Исторически одним из первых подходов, применяемым для генерации тестов, является метод активизации одномерного пути. Сущность его заключается в том, что для транспортировки неисправности от места ее возникновения до одного из выходов схемы активизируется одномерный путь. Процедура активизации осуществляется в следующей последовательности:

1. Вычисляется условие, при котором заданная неисправность проявляется в точке ее возникновения. Так, например, для неисправности $\equiv 1$ условием наблюдаемости будет обеспечение значения логического нуля для полюса схемы, на котором возникла данная неисправность.

2. Выбирается путь, по которому неисправность транспортируется на выход. Он задается в виде последовательности элементов, лежащих на нем.

3. Устанавливается условие активности выбранного пути в терминах входных переменных элементов, образующих его. Для элемента выбранного пути входные переменные задаются таким образом, что его выходное значение определяется только входом, подключенным к выходу предыдущего элемента, лежащего на данном пути.

4. Вычисляются значения входных переменных схемы, которые будут определять условия наблюдаемости неисправности и ее транспортировки на выход схемы, т. е. получается условие обнаружения заданной неисправности в терминах входных переменных схемы. Результатом вычислений и будет искомый тест.

Пункты 1–3 приведенного алгоритма часто называют прямой фазой метода активизации одномерного пути, а пункт 4 – обратной фазой.

В качестве примера, иллюстрирующего метод активизации одномерного пути, рассмотрим цифровую схему (рис. 2).

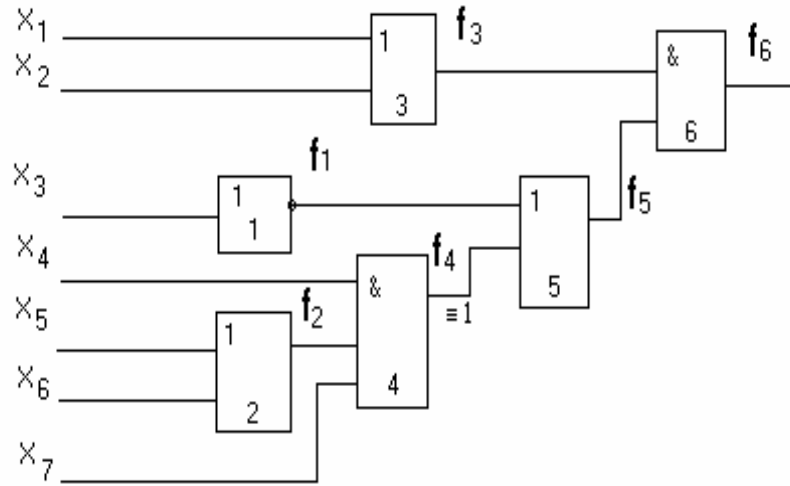


Рис. 2. Цифровая схема

Построим для нее тест, позволяющий обнаруживать неисправность $\equiv 1$ по выходу элемента 4.

Согласно приведенной методике, заданная неисправность будет наблюдаться при условии $f_4 = 0$ или, учитывая функцию, выполняемую элементом 4: $f_4(x_4, f_2(x_5, x_6), x_7) = x_4(x_5 \vee x_6)x_7 = 0$.

Далее выбираем путь, проходящий через элементы 5 и 6. Активность пути через элемент 5 будет определяться значением $f_1 = 0$ на втором его входе, в этом случае изменение f_4 приведет к изменению f_5 . Выполнение равенства $f_3 = 1$ обеспечивает условие активности пути через элемент 6. В результате осуществления прямой фазы метода активизации одномерного пути получаем условие наблюдаемости неисправности в точке возникновения и ее транспортировки на выход в виде системы логических уравнений:

$$\begin{cases} f_4 = 0, \\ f_1 = 0, \\ f_3 = 1. \end{cases} \quad (1)$$

Решение системы (1) содержательно представляет собой процедуру, выполняемую в процессе обратной фазы метода активизации одномерного пути. С учетом функций, реализуемых элементами схемы, данная система примет вид:

$$\begin{cases} x_4x_5x_7 \vee x_4x_6x_7 = 0, \\ \bar{x}_3 = 0, \\ x_1 \vee x_2 = 1. \end{cases}$$

Данная система может иметь множество решений, каждое из которых является тестом, позволяющим обнаруживать заданную неисправность. Действительно, для вектора переменных $x_1x_2x_3x_4x_5x_6x_7 = 1110111$, представляющего одно из возможных решений, значение f_6 на выходе схемы (рис. 2) в случае наличия неисправности $f_4 \equiv 1$ равно 1, а в случае ее отсутствия – 0. Это свидетельствует о наблюдаемости неисправности на выходе схемы.

Однако, несмотря на очевидную простоту реализации, метод активизации одномерного пути не нашел широкого практического применения, так как по приведенным процедурам прямой и обратной фазы не всегда можно получить искомым тест. Это хорошо видно из классического примера Шнейдера (рис. 3).

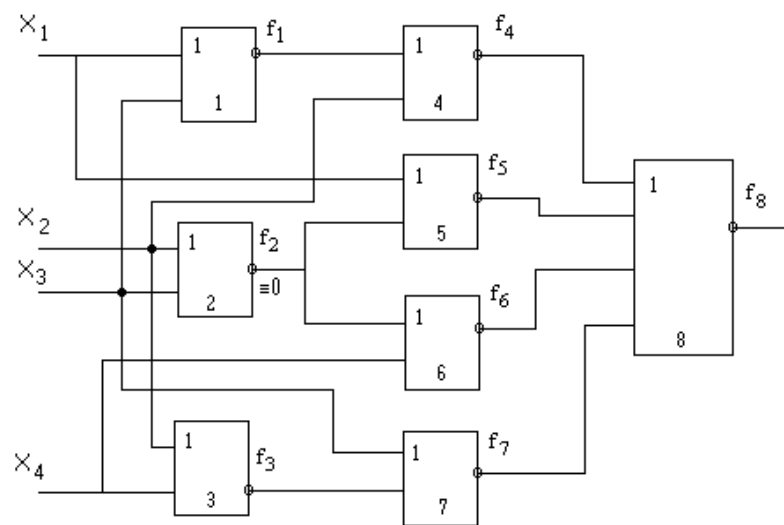


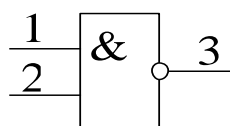
Рис. 3. Схема Шнейдера

В данном случае для неисправности $f_2 \equiv 0$ невозможно построить тест с помощью метода активизации одномерного пути. Действительно, условием наблюдаемости неисправности $f_2 \equiv 0$ в точке возникновения является равенство нулю x_2 и x_3 , а ее транспортировки на выход схемы – соотношения $x_4 = 0$ и $f_4 = f_5 = f_7 = 0$. Для выполнения равенства $f_5 = 0$ x_1 должно равняться 1, соответственно $f_1 = 0$, а с учетом $x_2 = 0$ получим, что $f_4 = 1$, что противоречит условию $f_4 = 0$ активности пути через элемент 8. Аналогично можно показать, что невозможно также активизировать путь через элементы 5 и 8. В то же время очевидно, что соотношение $x_1 = x_4 = 0$ позволяет одновременно активизировать два возможных пути для транспортировки неисправности $f_2 = 0$, а входной набор $x_1x_2x_3x_4 = 0000$ является ее тестом.

Приведенный пример показывает необходимость одновременной активизации нескольких путей для решения задачи построения тестовых воздействий.

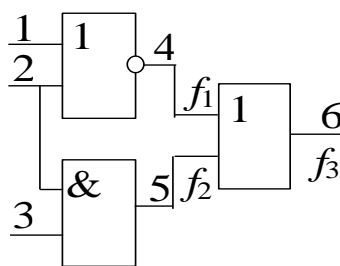
2.1.2. Метод активизации многомерного пути или D-алгоритм

Метод активизации многомерного пути (D-алгоритм), предложенный Ротом, гарантирует нахождение теста для заданной неисправности, если такой тест существует. D-алгоритм Рота основан на применении кубического описания булевых функций, которое является одной из возможных форм их задания. Среди кубических представлений булевых функций, используемых в указанном алгоритме, можно выделить так называемые сингулярные кубы, совокупность которых является сингулярным покрытием схемы, реализующей булеву функцию. Для задания сингулярных кубов применяются символы 0, 1 и \times , где \times означает, что переменная по данной координате может принимать как нулевое, так и единичное значение. На рис. 4, а приведено множество сингулярных кубов для элемента 2И – НЕ, а на рис. 4, б – для более сложной схемы, состоящей из элементов 2И, 2ИЛИ – НЕ, 2ИЛИ.



1	2	3
1	1	0
0	\times	1
\times	0	1

Рис. 4, а



	1	2	3	4	5	6
f_1	0	0		1		
	\times	1		0		
	1	\times		0		
f_2		1	1		1	
		\times	0		0	
		0	\times		0	
f_3				0	0	0
				1	\times	1
				\times	1	1

Рис. 4, б

Рассмотренные сингулярные кубы используются для получения D-кубов, при задании которых применяются символы 0, 1, \times , d , \bar{d} . Символ d может иметь значение, как нуля, так и единицы, а \bar{d} – противоположное значение. Для конкретной цифровой схемы все символы d принимают одно и то же значение 0 или 1, а \bar{d} соответственно 1 или 0.

D-куб, описывающий поведение цифровой схемы, получается в результате выполнения по координатной операции пересечения двух сингулярных кубов из сингулярного покрытия схемы с различным значением по выходной координате на основании следующих соотношений:

$$\begin{aligned}
0 \cap 0 &= 0, 0 \cap \times = 0, \times \cap 0 = 0, \\
1 \cap 1 &= 1, 1 \cap \times = 1, \times \cap 1 = 1, \\
\times \cap \times &= \times, 1 \cap 0 = d, 0 \cap 1 = \bar{d}.
\end{aligned}$$

Так, например, на основании $C_1 = \overset{1 \ 2 \ 3}{111}$ и $C_2 = \overset{1 \ 2 \ 3}{\times 0 1}$ из сингулярного покрытия элемента 2И — НЕ в результате их покоординатного пересечения получим D-куб $C_1 \cap C_2 = \overset{1 \ 2 \ 3}{1 d \bar{d}}$.

Для описания заданной неисправности используются так называемые примитивные D-кубы неисправности, которые состоят из входного набора, позволяющего наблюдать неисправность по выходу элемента, и символа d или \bar{d} в выходной координате. Для примера, приведенного на рис. 4, a , и неисправности $\equiv 0$ по ее выходу, соответствующий примитивный D-куб будет иметь вид $\overset{1 \ 2 \ 3}{0 0 d}$.

В данном случае символ d означает, что в исправном состоянии по выходной координате должен быть символ 1, а в неисправном 0. В то же время для кубов $\overset{1 \ 2 \ 3}{1 \times \bar{d}}$, описывающих поведение элемента 2ИЛИ – НЕ в случае неисправности $\equiv 1$ по ее выходу, символ \bar{d} означает, что в исправном состоянии на его выходе должен быть символ 0, а в неисправном 1.

D-куб неисправности сравнительно легко строится для одного элемента схемы, на выходе которого присутствует константная неисправность. Однако основной задачей D-алгоритма является построение D-куба неисправности всей рассматриваемой схемы. Для этого используется операция D-пересечения, введенная Ротом с целью формализации процедуры активации многомерного пути. Применительно к D-алгоритму процедура D-пересечения двух D-кубов $A = (a_1, a_2, \dots, a_n)$ и $B = (b_1, b_2, \dots, b_n)$, где $a_i, b_i \in \{0, 1, \times, d, \bar{d}\}$, $i = \overline{1, n}$, выполняется по соотношениям:

$$1. \times \bigcap_d b_i = b_i; a_i \bigcap_d \times = a_i.$$

$$2. \text{ Если } a_i \neq \times \text{ и } b_i \neq \times, \text{ то } a_i \bigcap_d b_i = \begin{cases} a_i, & \text{если } b_i = a_i, \\ \emptyset, & \text{в противном случае,} \end{cases}$$

где \emptyset — пустое множество.

Окончательно результатом D-пересечения двух кубов A и B будет являться куб $A \bigcap_d B = (a_1 \bigcap_d b_1, a_2 \bigcap_d b_2, \dots, a_n \bigcap_d b_n)$, если для любого i $a_i \bigcap_d b_i \neq \emptyset$, в противном случае $A \bigcap_d B = \emptyset$.

Непосредственно D-алгоритм состоит из следующих основных этапов:

1. Выбирается примитивный D-куб заданной неисправности элемента цифровой схемы.

2. Активизируются возможные пути от элемента, содержащего неисправность, к выходу схемы. При этом применяется операция D-пересечения D-куба неисправности с D-кубами элементов, последовательно соединенных между исходным неисправным элементом и выходами схемы. В результате первоначального пересечения D-куба неисправности с D-кубом последующего элемента получается D-куб заданной неисправности, описывающий поведение схемы из двух элементов. Последовательность пересечений D-кубов неисправности с D-кубами элементов, лежащих на путях транспортировки неисправности, часто называют процедурой D-прохода. D-проход оканчивается только тогда, когда хотя бы на одной из выходных координат не будет получено значение символа d или \bar{d} .

3. Выполняется обратная фаза как процедура последовательного пересечения полученного D-куба заданной неисправности, описывающего поведение неисправности элемента, и элементов, лежащих на путях транспортировки неисправности с сингулярными кубами остальных элементов.

Практическое применение D-алгоритма рассмотрим на примере цифровой схемы (рис. 5) для случая неисправности $f_2 \equiv 0$.

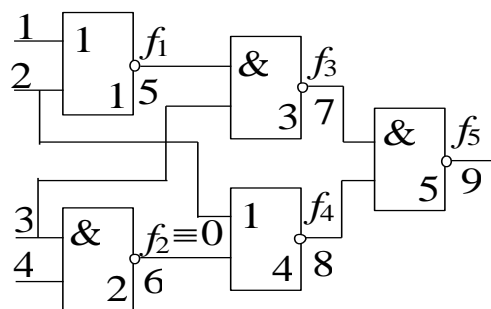


Рис. 5. Логическая схема

Первоначально получим множество сингулярных кубов для элементов 1 и 3, D-кубов для элементов 4 и 5, а также примитивных D-кубов неисправности для элемента 2:

$$\begin{array}{ccccccc}
\frac{f_1}{1 \ 2 \ 5} & \frac{f_2}{3 \ 4 \ 6} & \frac{f_3}{3 \ 5 \ 7} & \frac{f_4}{2 \ 6 \ 8} & \frac{f_5}{7 \ 8 \ 9} \\
\times 10 & 10 d & 0 \times 1 & 0 d \bar{d} & 1 \bar{d} d \\
1 \times 0 & 0 1 d & \times 0 1 & d 0 \bar{d} & \bar{d} 1 d \\
0 0 1 & 1 1 \bar{d} & 1 1 0 & d d \bar{d} & \bar{d} \bar{d} d
\end{array}$$

Из множества D-кубов неисправности элемента 2 выберем куб $C_0 = \frac{346}{10d}$. Далее на первом шаге D-прохода выполним процедуру D-пересечения D-куба неисправности C_0 с D-кубом C_1 , описывающим элемент 4. В результате

$$\begin{array}{l}
\frac{1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9}{C_0 = \times \times 1 0 \times d \times \times \times,} \\
\bigcap_d \\
\frac{C_1 = \times 0 \times \times \times d \times \bar{d} \times}{C_2 = \times 0 1 0 \times d \times \bar{d} \times}
\end{array}$$

D-куб C_2 пересекается с D-кубом C_3 элемента 5

$$\begin{array}{l}
\frac{1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9}{C_2 = \times 0 1 0 \times d \times \bar{d} \times,} \\
\bigcap_d \\
\frac{C_3 = \times \times \times \times \times \times 1 \bar{d} d}{C_4 = \times 0 1 0 \times d 1 \bar{d} d}
\end{array}$$

Анализ куба C_4 показывает, что на полюсе 9 рассматриваемой схемы, который является ее выходом, получен символ d , что свидетельствует об окончании D-прохода.

При выполнении обратной фазы куб C_4 последовательно пересекается с сингулярными кубами элементов 3 и 1. В результате получим

$$\begin{array}{l}
\frac{1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9}{C_4 = \times 0 1 0 \times d 1 \bar{d} d,} \\
\bigcap_d \\
\frac{C_5 = \times \times \times \times 0 \times 1 \times \times}{C_6 = \times 0 1 0 0 d 1 \bar{d} d}, \\
\bigcap_d \\
\frac{C_7 = 1 \times \times \times 0 \times \times \times \times}{C_8 = 1 0 1 0 0 d 1 \bar{d} d}
\end{array}$$

Таким образом, окончательно будем иметь куб C_8 , который показывает, что при подаче на входы схемы набора $x_1x_2x_3x_4 = 1010$ и в случае наличия неисправности $f_2 \equiv 0$ обеспечиваются условия транспортировки указанной неисправности на выход схемы. Другими словами, входной набор 1010 является тестом для неисправности $f_2 \equiv 0$.

Рассмотренный D-алгоритм позволяет формировать тест, если такой существует для каждой неисправности схемы, требует сравнительно небольшого объема памяти и отличается практически реальной трудоемкостью, что позволяет широко использовать его в системах автоматической генерации тестов [7]. Но, как отмечалось выше, для схем, содержащих значительное число элементов, резко возрастает размерность решаемой задачи, что приводит к чрезмерным затратам машинного времени.

2.1.3. Вероятностный метод

Вероятностное тестирование состоит в применении последовательностей случайных независимых двоичных цифр, подаваемых на входы проверяемой схемы. При этом переменная $x_i \in \{0,1\}, i = \overline{1,n}$, подаваемая на i -й вход, описывается вероятностью $p(x_i = 1)$ ее единичного значения.

Классической является схема вероятностного тестирования, состоящего из генератора входных наборов, проверяемой и эталонной цифровых схем, а также блока сравнения выходных реакций. Наиболее простым способом получения входного набора является использование датчика случайных чисел, при этом двоичное представление числа рассматривается как входной набор. Именно такой подход применен в используемом программном модуле. Более эффективным способом является задание входного набора разработчиком схемы, исходя из каких-либо оценок.

В разработанном приложении реализован следующий алгоритм:

1. Пользователь задает целочисленные значения параметров M1, M2, M3.
2. Используя датчик случайных чисел, генерируется входной набор, если он проверяет более M1 неисправностей, то генерируется следующий набор, иначе выполняется пункт 3.
3. Генерируется M2 входных наборов, если эти наборы суммарно проверяют менее чем M3 неисправностей, то выход, иначе выполняется пункт 2.

Таким образом, очевидно, что не всегда вероятностный метод строит тест, проверяющий 100% неисправностей.

Данный метод в сочетании с D-алгоритмом позволяет сократить временные затраты на построение теста. В этом случае на первом этапе применяется вероятностный метод, позволяющий сравнительно быстро построить тест для неисправностей, обнаруживаемых множеством тестовых наборов. На втором этапе для неисправностей, требующих для своего обнаружения конкретного входного набора, используется D-алгоритм [7].

2.2. Методы моделирования неисправностей

Моделирование неисправностей используется при анализе тестов на полноту и построении тестов вероятностным методом для определения проверяющих возможностей входных наборов. Отметим также, что последнюю задачу необходимо решать и при построении тестов направленными методами. Ввиду больших размеров логических схем и функциональной сложности элементов, из которых они состоят, задача моделирования является весьма трудоемкой. Учитывая широкий круг задач, при решении которых применяется моделирование входных наборов, разработка эффективных методов моделирования весьма актуальна. Ниже будут рассмотрены некоторые методы моделирования неисправностей логических схем в алфавите $V_2 = \{0,1\}$.

2.2.1. Параллельное моделирование неисправностей

Пусть $F = \{f_1, f_2, \dots, f_M\}$ – множество константных неисправностей заданной схемы.

Если разрядность компьютера N , то одновременно моделируется $N - 1$ экземпляров неисправных схем, каждый из которых содержит одну неисправность из F , и одна исправная схема. Для моделирования M неисправностей понадобится $\lceil M / (N - 1) \rceil$ циклов моделирования. Рассмотрим пример моделирования неисправностей для схемы, приведенной на рис. 6.

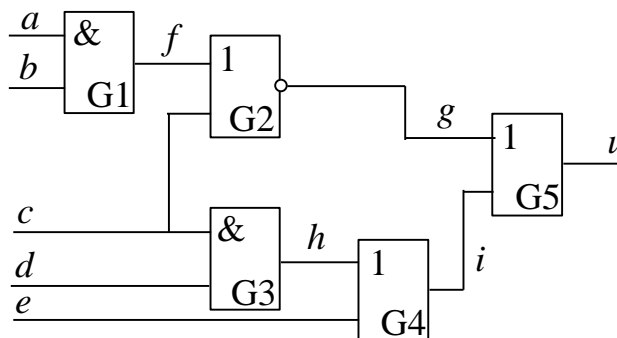


Рис. 6. Логическая схема

Неисправность «константа α » на i -й линии обозначим через i/α , через ff обозначим исправную схему.

Пусть ЭВМ имеет разрядность $N = 16$. Следовательно, одновременно можно моделировать $N - 1 = 15$ неисправностей. Моделирование проведем для набора $T = (abcde) = (10010)$. Результаты моделирования приведены в таблице 2.

Таблица 2

Результаты параллельного моделирования

	ff	a 0	b 1	c 1	d 0	e 1	f 0	f 1	g 0	g 1	h 0	h 1	i 0	i 1	u 0	u 1
<i>a</i>	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
<i>b</i>	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
<i>c</i>	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
<i>d</i>	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
<i>f</i>	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
<i>g</i>	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	1
<i>h</i>	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0
<i>e</i>	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
<i>i</i>	0	0	0	1	0	1	1	0	0	0	0	1	0	1	0	0
<i>u</i>	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1

Строки соответствуют линиям схемы, столбцы неисправностям. Каждый столбец представляет собой вектор, описывающий поведение схемы с соответствующей неисправностью. Заданный набор является тестом для неисправности, если значения функции, реализуемой исправной схемой и схемой с неисправностью, различны. В нашем случае, входной набор $T = (10010)$ обнаруживает следующие неисправности: $b|1, f|1, g|0, u|0$.

2.2.2. Дедуктивное моделирование неисправностей

Множество проверяемых неисправностей при дедуктивном моделировании вычисляется в процессе одного цикла моделирования заданной схемы на входном наборе. Однако этот цикл требует больших вычислений, чем цикл параллельного моделирования.

Сущность метода состоит в том, чтобы вместо поведения неисправных схем изучать отличие в поведении неисправных и исправной схемы.

Если при параллельном моделировании для линий определялись значения сигналов в каждой из неисправных схем, то в данном методе для каждой

линии схемы перечисляются неисправности, изменяющие значение сигнала на ней по сравнению с сигналом в исправной схеме, а вместо операций над значениями сигналов выполняются операции над списками неисправностей.

Введем операции объединения (\cup), пересечения (\cap) и дополнения (!) над списками неисправностей.

Объединением списков $z_1 \cup z_2$ называется список, включающий все элементы, входящие хотя бы в один из списков z_1 или z_2 . Пересечением списков $z_1 \cap z_2$ называется список, включающий все элементы, входящие одновременно в списки z_1 и z_2 . Разностью списков $z_1 - z_2$ называется список, включающий все элементы, входящие в z_1 , но не входящие в z_2 .

Рассмотрим элемент e_i с k входами и значениями сигналов на входах и выходе $y_1, y_2, \dots, y_j, \dots, y_k$ и y_i соответственно. Пусть для каждого входа j известен входной список неисправностей z_j , для которых значение сигнала на этом входе равно $\overline{y_j}$. Тогда список неисправностей, для которых значение сигнала на выходе равно $\overline{y_i}$, определяется функцией, реализуемой элементом, значениями сигналов на его входах и входными списками неисправностей [1, 8]. Приведем формулы, определяющие выходной список неисправностей для элементов «И», «И-НЕ», «ИЛИ», «ИЛИ-НЕ», обозначив через J_0 подмножество входов элемента, имеющих значение 0, через J_1 – подмножество входов, имеющих значение 1, и $J = J_0 \cup J_1$:

а) для элемента «И»

$$J_0 = \emptyset : z_i = \bigcup_J z_j \bigcup \{i \equiv 0\},$$

$$J_0 \neq \emptyset : z_i = \left[\left(\bigcap_{J_0} z_j \right) - \left(\bigcup_{J_1} z_j \right) \right] \bigcup \{i \equiv 1\}.$$

Действительно, если на всех входах элемента «И» значение сигналов 1, то для изменения значения сигнала на выходе должно измениться хотя бы одно значение на входе. Если же сигнал на выходе равен 0, то для его изменения необходимо, чтобы изменились сигналы на всех входах со значением 0, а сигналы на всех выходах со значением 1 остались неизменными.

b) для элемента «И-НЕ»

$$J_0 = \emptyset : z_i = \bigcup_J z_j \bigcup \{i \equiv 1\},$$

$$J_0 \neq \emptyset : z_i = \left[\left(\bigcap_{J_0} z_j \right) - \left(\bigcup_{J_1} z_j \right) \right] \bigcup \{i \equiv 0\}.$$

c) для элемента «ИЛИ»

$$J_1 = \emptyset : z_i = \bigcup_J z_j \bigcup \{i \equiv 1\},$$

$$J_1 \neq \emptyset : z_i = \left[\left(\bigcap_{J_1} z_j \right) - \left(\bigcup_{J_0} z_j \right) \right] \bigcup \{i \equiv 0\}.$$

d) для элемента «ИЛИ-НЕ»

$$J_1 = \emptyset : z_i = \bigcup_J z_j \bigcup \{i \equiv 0\},$$

$$J_1 \neq \emptyset : z_i = \left[\left(\bigcap_{J_1} z_j \right) - \left(\bigcup_{J_0} z_j \right) \right] \bigcup \{i \equiv 1\}.$$

Для узла разветвления i с исходящими ветвями $i_1, \dots, i_j, \dots, i_l$ справедлива формула: $z_{i_j} = z_i \bigcup \{i_j \equiv \bar{y}_i\}$.

Проиллюстрируем применение дедуктивного алгоритма на схеме, приведенной на рис. 6. Список неисправностей L_a и L_b по входам a и b , соответственно, передается на линию f , список неисправностей L_c и L_f по входу c и линии f передается на линию g и т.д. Процесс продвижения неисправностей от входов схемы к ее выходу изображен на рис. 7.

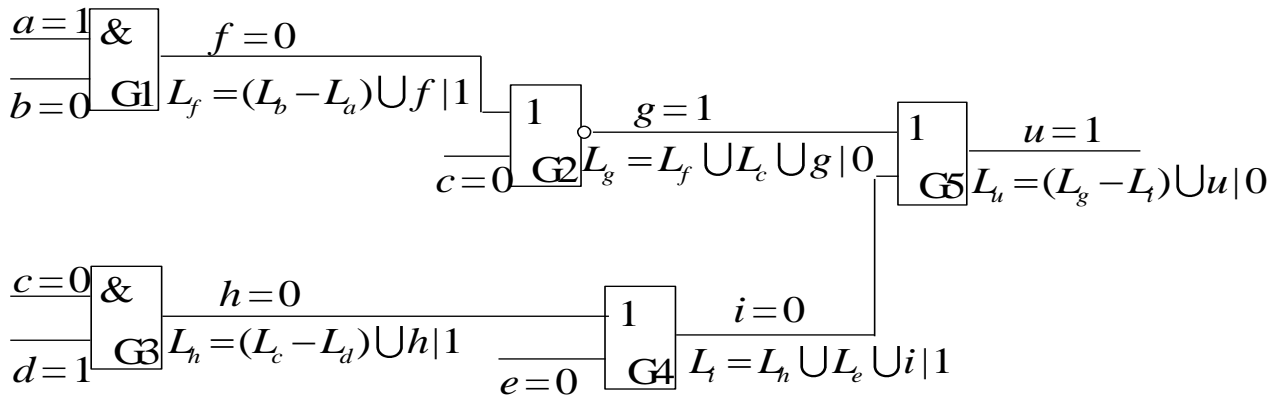


Рис. 7. Процесс продвижения неисправностей

В результате на выходе схемы u получим список неисправностей $L_u = (L_g - L_i) \cup u | 0 = \{b | 1, f | 1, g | 0, u | 0\}$.

Реализация дедуктивного алгоритма связана с серьезными трудностями. Основная из них состоит в распределении оперативной памяти под списки. Т.к. размеры списков определяются входным набором и состоянием схемы, а поэтому не могут быть определены заранее, необходимо динамическое распределение памяти. Кроме того, эффективность алгоритма во многом зависит от качества программной реализации операций над списками.

Кроме вышерассмотренных методов существуют и другие алгоритмы моделирования неисправностей [7, 8], а также их модификации для различных классов схем [2].

3. МОДИФИКАЦИЯ МЕТОДОВ И АЛГОРИТМОВ МОДЕЛИРОВАНИЯ НЕИСПРАВНОСТЕЙ И ПОСТРОЕНИЯ ТЕСТОВ

Актуальна задача разработки и анализа математических моделей элементов большой интеграции, ориентированных на применение в системах автоматизированного проектирования тестов. Среди основных требований, которые должны предъявляться к моделям ФСЭ, отметим такие как удобство для решения задач, возникающих при построении тестов, универсальность, возможность компактного представления в памяти ЭВМ.

3.1. Математическая модель ФСЭ

В качестве одной из моделей элементов, в значительной степени удовлетворяющей перечисленным выше требованиям, может быть использована одна из универсальных аналитических форм представления булевых функций (ДНФ, КНФ и др.). Такая модель всегда применима для логических элементов. Система ДНФ (КНФ и др.) имеет компактное матричное представление, ориентированное на машинную обработку.

Пусть x_1, \dots, x_n – переменные, которые задают значения сигналов на входах элемента; f_1, \dots, f_m – функции, реализуемые на соответствующих выходах элемента; задана система ДНФ, реализуемых элементом функций:

$$\begin{cases} f_1 = D_1(x_1, \dots, x_n), \\ \vdots \\ f_m = D_m(x_1, \dots, x_n), \end{cases}$$

и $D \sqsubset \{D_1, \dots, D_m\}$ – краткое обозначение данной системы ДНФ. В качестве матричного представления D можно использовать следующее. Если $K_i = \tilde{x}_{i_1} \dots \tilde{x}_{i_j} \dots \tilde{x}_{i_r}$ – некоторая конъюнкция из D , то для ее представления можно использовать троичный вектор $T_i = (t_1^i, \dots, t_k^i, \dots, t_n^i)$. Здесь \tilde{x}_{i_σ} означает, что переменная x_{i_σ} входит в K_i с отрицанием или без него;

$$t_k^i \in \{0, 1, -\}, t_k^i = 1 \Leftrightarrow x_k \in K_i, t_k^i = 0 \Leftrightarrow \bar{x}_k \in K_i, t_k^i = - \Leftrightarrow x_k, \bar{x}_k \notin K_i.$$

Если $\{K_1, \dots, K_i, \dots, K_{l_D}\}$ – совокупность конъюнкций из D , то представим данную совокупность троичной матрицей $T = [T_1 \dots T_{l_D}]^t$. Булеву матрицу

$B = [b_{ij}]$, будем использовать для задания отношения принадлежности конъюнкций из D ДНФ функций f_1, \dots, f_m . Здесь $i = \overline{1, l_D}; j = \overline{1, m}; b_{ij} = 1 \Leftrightarrow K_i \in D_j$. Таким образом, совокупность матриц T и B может быть использована для представления системы D . Рассматриваемая модель элементов удобна для решения различных задач, связанных с направленным построением тестов, так как позволяет непосредственно использовать развитый аппарат ДНФ, а также различные соответствующие преобразования троичных и булевых матриц и известные эффективные алгоритмы их выполнения [3, 5].

Пример. Рассмотрим некоторый функциональный элемент с двумя входами и двумя выходами (рис. 8).

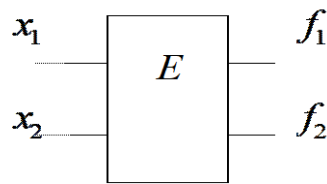


Рис. 8. Функциональный элемент

Пусть указанный элемент реализует систему булевых функций, представленных ДНФ

$$\begin{cases} f_1 = x_1 x_2 \vee \bar{x}_1 \bar{x}_2, \\ f_2 = \bar{x}_1 \bar{x}_2. \end{cases}$$

В качестве модели данной реализации системы функций можно использовать пару матриц

$$T = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}; B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

Здесь T – матрица, задающая конъюнкции $x_1 x_2$ (первая строка) и $\bar{x}_1 \bar{x}_2$ (вторая строка), а матрица B определяет отношение принадлежности данных конъюнкций функциям системы.

Описанная матричная модель элементов может быть использована, например, при построении тестов для логических схем на основе метода активизации существенных путей. Необходимо отметить, что алгоритмы решения основных задач, которые возникают в процессе построения проверяющих наборов на основе метода существенных путей (вычисление условий проявления неисправности – D-кубов неисправности, условий транспортировки неисправности до выходных полюсов путем пересечения куба для логической сети с кубами из кубических покрытий функций, реализуемых элементами сети), либо ориентированы на простые одновыходные элементы типа И, ИЛИ, И-НЕ,

ИЛИ-НЕ, либо применение их в случае элементов, реализующих произвольные булевы функции, становится неэффективным.

3.2. Модификация D-алгоритма для ФСЭ

Рассмотрим модификацию метода активизации существенных путей для случая, когда элементы заданы матричной моделью, описанной выше.

Вначале, остановимся на более узкой задаче, а именно, нахождении входного набора, проверяющего заданную константную неисправность на входном или выходном полюсе ФСЭ.

Если требуется построить проверяющий набор t_{s_i} для неисправности s_i на входном или выходном полюсе ФСЭ с номером i , то нахождение условий проявления неисправности s_i хотя бы на одном из выходных полюсов данного ФСЭ сводится к вычислению в общем случае частично определенного входного набора для данного ФСЭ, на котором значение хотя бы на одном из выходных полюсов ФСЭ зависит от наличия неисправности s_i , т.е. к вычислению D-куба неисправности. Как правило, ФСЭ реализует систему функций зависящих от 10 и более переменных. В связи с этим применение формальных правил для вычисления D-кубов неисправностей и D-кубов становится проблематичным. Поэтому, воспользуемся идеей транспортировки константной неисправности s_i на входном полюсе через ФСЭ с номером i (предполагается, что все элементы схемы пронумерованы) для вычисления D-кубов неисправности s_i . При этом для упрощения вычислений будем находить условия транспортировки неисправности по простым существенным путям. Уточним понятие простого существенного пути через ФСЭ. Будем говорить, что вычислены условия образования простого существенного пути от входного полюса x_j ФСЭ до его выходного полюса y_k , если найден в общем случае частично определенный входной набор для данного ФСЭ такой, что выполняются условия:

1. Существует конъюнкция $K_l \in D$, значение которой на данном входном наборе зависит от значения на входном полюсе x_j .
2. Значение функции, реализованной на k -м выходном полюсе ФСЭ, зависит на данном входном наборе от значения, которое принимает конъюнкция K_l .

Таким образом, простой существенный путь через ФСЭ можно определить тройкой (j, l, k) . Сложный существенный путь от входного полюса x_j ФСЭ до его выходного полюса y_k отличается от простого существенного пути тем, что в этом случае допускается, что значения нескольких конъюнкций

$\{K_{l_1}, \dots, K_{l_m}\} \subset D$ зависят от значения на входном полюсе x_j , а значение функции, реализованной на k -м выходе ФСЭ, зависит от значений, которые принимают конъюнкции K_{l_1}, \dots, K_{l_m} . Сложный существенный путь можно описать вектором (j, l_1, \dots, l_m, k) .

Рассмотрим на примере вычисление D-кубов неисправностей на входных полюсах ФСЭ, используя идею транспортировки через ФСЭ.

Пусть дан ФСЭ (рис. 9), реализующий систему ДНФ

$$\begin{cases} y_1 = \overset{K_1}{x_1 x_2} \vee \overset{K_2}{\bar{x}_1 \bar{x}_2}, \\ y_2 = \overset{K_3}{x_1 \bar{x}_2 \bar{x}_3} \vee \overset{K_4}{\bar{x}_1 x_2 x_3}, \\ y_3 = \overset{K_3}{x_1 \bar{x}_2 \bar{x}_3} \vee \overset{K_5}{x_1 x_2}. \end{cases}$$

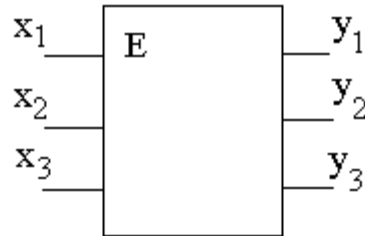


Рис. 9. Функциональный элемент

Пусть задана неисправность типа «константа 1» на входном полюсе x_2 . Попытаемся вычислить условия образования простого существенного пути от входного полюса x_2 до одного из выходных полюсов. От значения на входном полюсе x_2 могут зависеть значения конъюнкций K_1, K_2, K_3, K_4 , так как только в них входит переменная x_2 или ее отрицание. Для того чтобы значение конъюнкции K_1 зависело от значения на входном полюсе x_2 , необходимо и достаточно чтобы $x_1 = 1$. Очевидно, только значение функции, реализованной на выходном полюсе y_1 , может зависеть от значения конъюнкции K_1 . При этом необходимо чтобы $K_2 = 0$. Это условие выполняется, так как при $x_1 = 1$, вычисленном ранее, $K_2 = 0$. Таким образом, найден простой существенный путь $(2, 1, 1)$ при условии $x_1 = 1$. Итак, D-куб неисправности типа «константа 1» на входном полюсе x_2 имеет вид: $(x_1, x_2, x_3, y_1, y_2, y_3) = (1, \bar{d}, x, \bar{d}, x, x)$.

Рассмотрим еще вычисление D-куба неисправности в том случае, когда неисправность возникает на выходном полюсе ФСЭ. Пусть задана неисправ-

ность типа «константа 0» на выходном полюсе y_2 . Необходимым и достаточным условием проявления данной неисправности на выходном полюсе y_2 (на других выходных полюсах данная неисправность проявляться не может) является: либо $K_3 = 1$, либо $K_4 = 1$, либо одновременно $K_3 = 1$ и $K_4 = 1$. Конъюнкция K_3 принимает значение 1 при $x_2 = 0, x_3 = 0$. Таким образом, найден D-куб неисправности $(x, 0, 0, x, d, 1)$. Поиск значений на входных полюсах, которые приводят к выполнению второго условия, т.е. $K_4 = 1$, приводит к второму D-кубу неисправности $(0, 1, 1, 0, d, 0)$. Легко видеть, что не существует входного набора, при котором одновременно $K_3 = 1$ и $K_4 = 1$, т.е. не существует D-куба неисправности, соответствующего выполнению указанного условия. Итак, вычисление условий проявления константной неисправности на выходном полюсе элемента сводится к поиску корней логического уравнения, левая часть которого представляет собой ДНФ, а правая – равна нулю или единице.

Остановимся на особенностях решения задачи нахождения условий транспортировки неисправности через ФСЭ. Очевидно, что необходимым условием того, чтобы некоторый ФСЭ рассматривался при вычислении условий транспортировки неисправности до выходных полюсов схемы, является появление на его входных полюсах одного или нескольких символов d и \bar{d} . Так же, как и в случае вычисления условий проявления неисправности на входных полюсах ФСЭ, будем находить условия транспортировки неисправности через ФСЭ по простым существенным путям (при этом не исключается возможность образования сложного существенного пути). Одна из особенностей вычисления условий транспортировки неисправности через ФСЭ состоит в том, что в этом случае значения на некоторых входных полюсах ФСЭ могут быть определенными, т.е. 0 или 1, и символы d и \bar{d} могут появиться не только на одном, но одновременно на нескольких входных полюсах ФСЭ. Если символы d и \bar{d} присутствуют одновременно на нескольких входных полюсах ФСЭ, то будем поочередно вычислять условия образования простого существенного пути от каждого входного полюса, которому приписан символ d или \bar{d} , не противоречащие уже определенным значениям на некоторых входных полюсах ФСЭ, если таковые имеются. Вычисление условий транспортировки неисправности через ФСЭ соответствует выбору D-куба или тупикового D-куба для данного ФСЭ.

Теперь, когда известно как вычислять условия проявления неисправности (D-кубы неисправности) и условия транспортировки неисправности через

ФСЭ (D-кубы), рассмотрим общую схему алгоритма транспортировки неисправности от места неисправности до одного из выходных полюсов схемы. Будем предполагать, что ФСЭ, входящие в схему, правильно пронумерованы.

Под элементарным путем $P_s^{y_i} = (n_1, \dots, n_{i-1}, n_i, \dots, n_k)$ от места неисправности s до выходного полюса y_i схемы будем понимать упорядоченную в порядке возрастания последовательность номеров ФСЭ такую, что: первый ФСЭ (n_1) из этой последовательности либо соединен с входным полюсом схемы, на котором задана неисправность, либо соединен с некоторыми выходными полюсами другого ФСЭ, у которого задана неисправность на входном полюсе; среди выходных полюсов последнего ФСЭ (n_k) в последовательности есть выходной полюс сети y_i ; среди входных полюсов i -го ФСЭ в последовательности есть такие, которые соединены с выходными полюсами $(i-1)$ -го ФСЭ.

Сложным путем P_s от места неисправности s до всех выходных полюсов схемы будем называть упорядоченную в порядке возрастания последовательность номеров ФСЭ, представляющую собой объединение всех элементарных путей от места неисправности до выходных полюсов сети.

Будем находить условия транспортировки неисправности s от места неисправности до одного из выходных полюсов схемы по сложному пути. Для этого необходимо найти один из вариантов условий проявления неисправности, т.е. вычислить D-куб неисправности. Если при этом символы d и \bar{d} попали на некоторые выходы схемы, то процедура транспортировки неисправности до одного из выходных полюсов схемы завершена. В противном случае, после того как найден один из D-кубов неисправности s , будем поочередно находить условия транспортировки неисправности по простым существенным путям через ФСЭ, входящие в сложный путь от места неисправности, согласно порядку ФСЭ в сложном пути. Если после вычисления условий транспортировки неисправности через очередной ФСЭ оказалось, что символы d или \bar{d} попали на некоторые выходные полюсы схемы, то процедура транспортировки неисправности до одного из выходных полюсов сети завершена.

Рассмотренный алгоритм реализован в программном модуле построения проверяющего теста направленным методом.

3.3. Параллельное моделирование

неисправностей в алфавите $V_2 = \{0,1\}$ для ФСЭ

Напомним, что одновременно моделируются m экземпляров схем. Одна исправная и $m-1$ схем с неисправностями, взятыми из заданного списка.

Для представления значений на i -й линии схемы будем использовать булев вектор $\underline{a}^i = (a_1^i, \dots, a_j^i, \dots, a_m^i)$. Поведение j -го экземпляра схемы моделируется совокупностью значений j -х компонент векторов \underline{a}^i . Для внесения неисправностей поставим в соответствие каждому входному полюсу схемы вектор $\underline{b}^i = (b_1^i, \dots, b_j^i, \dots, b_m^i)$, а для каждого ФСЭ совокупность векторов

$$\begin{aligned}\underline{b}^{u, \beta_1} &= (b_1^{u, \beta_1}, \dots, b_j^{u, \beta_1}, \dots, b_m^{u, \beta_1}), \dots, \\ \underline{b}^{u, s_u} &= (b_1^{u, s_u}, \dots, b_j^{u, s_u}, \dots, b_m^{u, s_u}), \\ \underline{b}^{u, v_1} &= (b_1^{u, v_1}, \dots, b_j^{u, v_1}, \dots, b_m^{u, v_1}), \dots, \\ \underline{b}^{u, v_{t_u}} &= (b_1^{u, v_{t_u}}, \dots, b_j^{u, v_{t_u}}, \dots, b_m^{u, v_{t_u}})\end{aligned}$$

Здесь i – номер входного полюса схемы; u – номер ФСЭ; $\beta_1, \dots, \beta_{s_u}$ – номера входных полюсов u -го ФСЭ; s_u – число входных полюсов u -го ФСЭ; v_1, \dots, v_{t_u} – номера выходных полюсов u -го ФСЭ; t_u – число выходных полюсов u -го ФСЭ. Перечисленные векторы используются для внесения неисправностей в соответствующие экземпляры схемы. Если схема с неисправностью «константа α » ($\alpha \in \{0,1\}$) на i -м входном полюсе схемы моделируется совокупностью r -х компонент векторов \underline{a}^i , то $b_r^i = \alpha, b_j^i = \bar{\alpha}, j = \overline{1, m}, j \neq r$. Аналогично вносятся неисправности типа «константа α » на входных и выходных полюсах ФСЭ с помощью векторов $\underline{b}^{u, \beta_1}, \dots, \underline{b}^{u, s_u}, \underline{b}^{u, v_1}, \dots, \underline{b}^{u, v_{t_u}}$.

Ниже приводятся формулы для вычисления значений сигналов на выходных полюсах некоторых логических элементов.

1) При моделировании неисправностей типа «константа 0»:

а) инвертор

$$\underline{a}^i = \overline{\underline{a}^{p_j} \underline{b}^{i, p_j} \underline{b}^i}$$

б) конъюнктор (приводится рекуррентная формула для вычисления значения на выходе k -входного конъюнктора через значение на выходе $(k-1)$ – входного конъюнктора и значение на k -м входе)

$$k = 1,$$

$$\underline{a}_1^i = \underline{a}^{p_1} \underline{b}^{i, p_1}$$

$$k \geq 2,$$

$$\underline{a}_k^i = \underline{a}_{k-1}^i \underline{a}^{p_k} \underline{b}^{i, p_k}$$

Если $k = n$, где n – число входов в конъюнкторе, то на выход вносятся неисправность с помощью вектора \underline{b}^i : $\underline{a}_n^i = \underline{a}_{n-1}^i \underline{a}^{p_n} \underline{b}^{i, p_n} \underline{b}^i$

с) аналогичные формулы выписываются для дизъюнктора:

$$k = 1,$$

$$\underline{a}_1^i = \underline{a}^{p_1} \underline{b}^{i, p_1}$$

$$k \geq 2,$$

$$\underline{a}_k^i = \underline{a}_{k-1}^i \vee \underline{a}^{p_k} \underline{b}^{i, p_k}$$

$$k = n,$$

$$\underline{a}_n^i = (\underline{a}_{n-1}^i \vee \underline{a}^{p_n} \underline{b}^{i, p_n}) \underline{b}^i$$

2) При моделировании неисправностей типа «константа 1»:

а) инвертор: $\underline{a}^i = \overline{(\underline{a}^{p_j} \vee \underline{b}^{i, p_j})} \vee \underline{b}^i$.

Вычисление значений на выходах n -входовых конъюнкторов и дизъюнкторов ведется с помощью формул, аналогичных формулам для этих элементов при моделировании неисправностей типа «константа 0».

б) конъюнктор:

$$k = 1,$$

$$\underline{a}_1^i = \underline{a}^{p_1} \vee \underline{b}^{i, p_1}$$

$$k \geq 2,$$

$$\underline{a}_k^i = \underline{a}_{k-1}^i (\underline{a}^{p_k} \vee \underline{b}^{i, p_k})$$

$$k = n,$$

$$\underline{a}_n^i = \underline{a}_{n-1}^i (\underline{a}^{p_n} \vee \underline{b}^{i, p_n}) \vee \underline{b}^i$$

с) дизъюнктор:

$$k = 1,$$

$$\underline{a}_1^i = \underline{a}^{p_1} \vee \underline{b}^{i, p_1}$$

$$k \geq 2,$$

$$\underline{a}_k^i = \underline{a}_{k-1}^i \vee \underline{a}^{p_k} \vee \underline{b}^{i, p_k}$$

$$k = n,$$

$$\underline{a}_n^i = \underline{a}_{n-1}^i \vee \underline{a}^{p_n} \vee \underline{b}^{i, p_n} \vee \underline{b}^i$$

Здесь совокупность векторов \underline{a}^{p_j} описывает вектор значений сигналов на входном полюсе p_j элемента; \underline{b}^{i,p_j} – вектор для внесения неисправностей на p_j -й вход, а \underline{b}^i – на выход элемента; \underline{a}^i – описывает вектор значений сигналов на выходе элемента.

В силу того, что ФСЭ представлены в виде системы ДНФ, приведенных выше формул достаточно для вычисления векторов значений на выходных полюсах ФСЭ.

Для входных полюсов схемы вычисления ведутся по формулам: $\underline{a}^i = \underline{a}^i \wedge \underline{b}^i$ для неисправностей типа «константа 0», $\underline{a}^i = \underline{a}^i \vee \underline{b}^i$ для неисправностей типа «константа 1». Здесь в векторе \underline{a}^i все компоненты первоначально равны значению сигнала на i -м входном полюсе схемы в исправном состоянии.

Приведенные выше формулы для внесения и параллельного моделирования неисправностей, использованы в алгоритме построения проверяющего теста, основанного на оценке проверяющих возможностей псевдослучайных входных наборов.

4. АНАЛИЗ РАЗРАБОТКИ CLR-ПРИЛОЖЕНИЙ НА ПЛАТ- ФОРМЕ .NET

Разработка типового проекта Windows Forms с помощью Visual C++ не столь отличается от разработки на других языках .NET, таких как Visual Basic или Visual C#.

Приложения Windows Forms в Visual C++ используют классы .NET Framework и иные функциональные возможности .NET с новым синтаксисом Visual C++.

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Он упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. Имеется в виду программист может использовать Windows Forms как при написании ПО на C# или C++, так и на VB.Net, J# и др.

В среде разработки Microsoft Visual Studio 2012 функция создания приложений вида WF для Visual C++ была устранена. Причиной послужило разделение: приоритет в разработке визуальных приложений с использованием .NET был отдан таким языкам как C# и Visual basic, но C++ в свою очередь является одним из лидеров в работе с функциями более низкого уровня, такими как использование памяти.

4.1 Технологии .NET и CLR.

.NET Framework — платформа, единолично разработанная компанией Microsoft в 2002 году. Основой платформы является общезыковая среда исполнения Common Language Runtime (CLR), что годится для различных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Common Language Runtime (англ. CLR — общезыковая исполняющая среда) — исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования (C#, C++, F# и прочие). CLR является одним из основных компонентов пакета Microsoft .NET Framework.

Среда CLR является реализацией спецификации CLI (англ. Common Language Infrastructure), спецификации общезыковой инфраструктуры компании Microsoft.

Более наглядно это изложено на Рис.9:

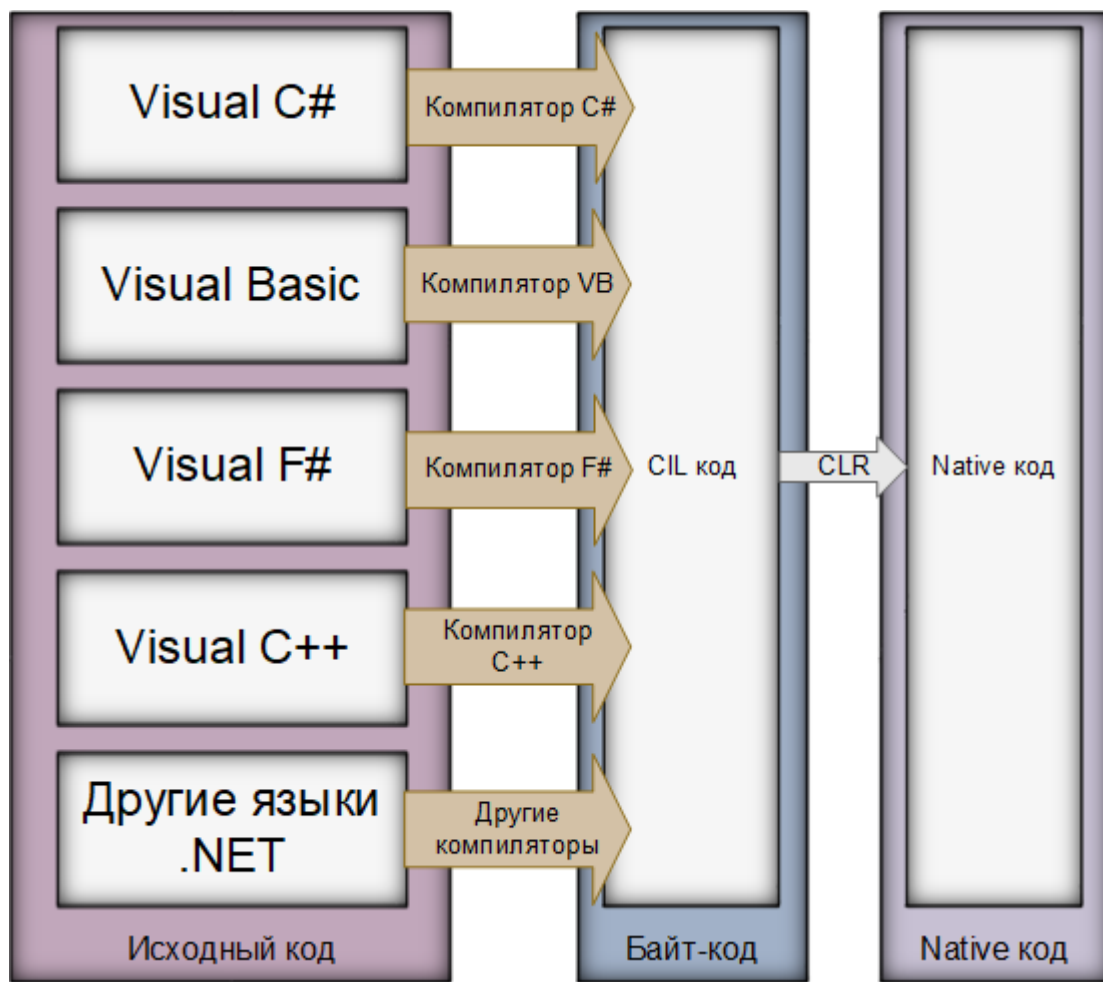


Рис. 9. Принцип работы CLR

Возьмём к примеру рассматриваемое приложение: работа выполнена на C++, но это не совсем C++, а его разновидность. У нас есть фреймворк для работы с формами, он называется .NET Framework Microsoft. Это специальный пакет, который позволяет нам работать именно с формами. И теперь нам нужно связать C++ с этим фреймворком. То есть, сама программа работает на C++, но форма работает по факту на другом языке программирования.

Получается следующее: Microsoft знают, что есть целая куча языков программирования, и как связать эти языки с нативным кодом, чтобы для каждого языка не писать отдельные компоненты, которые будут преобразовывать эти языки в нативный код, который потом преобразуется в ту форму, которую мы должны видеть. Давайте разберём на примере C++ и C#: мы знаем, что это кардинально разные языки, синтаксис у них разный. Но когда запускаются компиляторы у этих языков программирования (у каждого свой), они преобразуют код языка в одинаковый код, байт-код. То есть, раз эти языки поддерживают платформу .NET, в итоге компилятор преобразует этот код в одинако-

вый. И Microsoft сделали так, что этот байт-код может с помощью CLR преобразоваться в нативный код, который в свою очередь уже преобразуется в ту форму, которую мы будем видеть.

CLR поддерживает автоматическую сборку мусора и обеспечивает безопасную передачу данных между различными частями системы. Для обеспечения этих возможностей среда CLR должна владеть полной информацией о типах данных.

В свою очередь, C++ разрабатывался как универсальный язык, в том числе для решения задач реального времени, низкоуровневого программирования и написания драйверов аппаратных устройств, в связи с чем применение в нём автоматической сборки мусора, неявной инициализации переменных и проверки допустимости значений аргументов является неоправданными из-за потери производительности.

Здесь возникает явный конфликт между C++ и CLR. Для его решения было придумано следующее: как было сказано ранее, приложение разрабатывалось не на C++, а на его разновидности (базирующейся на C++). Именно в этой разновидности C++ есть управляемые типы. Управляемый тип сообщает компилятору, что наш класс является управляемым и подчиняется всем правилам среды CLR. Без этого мы не смогли бы взаимодействовать в форме с C++.

Ключевое слово `_gc` перед объявлением класса сообщает компилятору, что наш класс является управляемым и подчиняется всем правилам среды CLR. В частности, не нужно вызывать деструктор для удаления объекта из памяти, эту работу сделает CLR. Сейчас используется следующий синтаксис для работы с управляемыми классами, например:

```
String^ _string = gcnew String();
```

Архитектура .NET Framework описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке Framework Class Library (FCL). В FCL входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL).

Достоинства.

1. Вся платформа .NET основана на единой объектно-ориентированной модели. Все сервисы, интерфейсы и объекты, которые платформа предоставляет разработчику объединены в единую иерархию классов. Другими словами, все, что может потребоваться при создании приложений под платформу .NET будет всегда под рукой. Причем, все это сгруппировано очень удобно и интуитивно понятно.
2. Приложение, написанное на любом .NET-совместимом языке является межплатформенным (в идеале). Приложение, написанное на C# не зависит от платформы, на которой будет выполняться, но зато зависит от наличия платформы .NET. Достаточно один раз портировать архитектуру .NET под какую либо систему, после чего без проблем запускать абсолютно любое .NET приложение. Но в настоящий момент платформа .NET портирована только на семейство операционных систем Windows, в том числе на мобильные системы MS Windows mobile.
3. В состав платформы .NET входит так называемый "сборщик мусора", который освобождает ресурсы. Таким образом, приложения защищены от утечки памяти и от необходимости освобождать ресурсы. Это делает программирование более легким и более безопасным.
4. Приложения .NET используют метаданные, что позволяет им не пользоваться системным реестром Windows.
5. Любое .NET приложение является автономным, в том смысле, что не зависит от других программ, в частности от ОС. Установка приложения написанного на одном из .NET языков может быть произведена обычным копированием файлов (исключение составляет создание ярлыков в меню "Пуск" и др. местах).
6. Приложения .NET используют безопасные типы, что повышает их надежность, совместимость и межплатформенность.
7. Приложение, написанное на любом .NET языке взаимодействует с единой моделью обработки ошибок, что значительно упрощает этот процесс.
8. Приложения написанные на разных языках могут легко взаимодействовать. Например, серверная часть может быть написана на C#, а клиентская на Visual Basic.
9. .NET приложения могут быть сертифицированы на безопасность. Это является особенностью промежуточного кода, в который преобразуются все .NET приложения.

10. Абсолютно все ошибки обрабатываются механизмом исключительных ситуаций. Это позволяет избежать разногласия, который иногда возникал при программировании под Win32.

11. Повторное использование кода стало удобнее. Это связано с тем, что промежуточный язык MSIL не зависит от языка программирования. Например, вы можете написать программу на C#, а дополнение к ней писать уже на J#.

Недостатки.

У любого программного продукта есть свои недостатки, есть они и у платформы .NET.

1. Приложения, написанные под платформу .NET работают медленнее. В некоторых случаях скорость может упасть на 15%, что иногда является неприемлемым (например, при создании 3D приложений, где борьба идет за каждый FPS). Задержки в выполнении связаны с промежуточным языком MSIL, ведь для того чтобы его скомпилировать в выполняемый файл тоже нужно время. Разумеется, что приложение компилируется не все сразу, а по частям, равномерно при работе программы.

2. Не на любом языке можно создавать .NET приложения. Дело в том, что первоначально .NET конструировался под C/JAVA-подобные языки. Это породило некоторые трудности с созданием .NET компиляторов для других языков (особенно экзотических и узкоспециализированных). В результате этого некоторые функции пришлось решать нетривиальными способами, что отрицательно сказалось на производительности. Но постепенно данный недостаток сходит на нет, т.к. разработчики компиляторов поняли важность платформы .NET и стараются сделать для своих языков достойные инструменты.

3. Необходимо наличие библиотеки Framework. Данный недостаток устранен с выходом Windows Vista, т.к. данная библиотека встроена в систему по умолчанию.

Можно обратить внимание на то, что достоинств у .NET в сумме получилось больше, чем недостатков. Разумеется, это не является отражением реальности. Это говорит о хорошей маркетинговой кампании проведенной корпорацией Microsoft.

4.2 Визуальное программирование в Microsoft Visual Studio 2019

Как уже говорилось ранее, в среде разработки Microsoft Visual Studio 2019 отсутствует возможность создания приложения вида Windows Forms на языке Visual C++. Однако поддержка данного типа приложений существует, и разработчики могут изменять и дополнять ранее созданные приложения в других версиях среды программирования.

Чтобы создать приложение необходимо:

Шаг 1. Создаем пустой CLR проект на Visual C++.

Но для упрощения я выбрал консольное приложение: сразу будет создан файл .cpp с методом main, а также файл pch.h

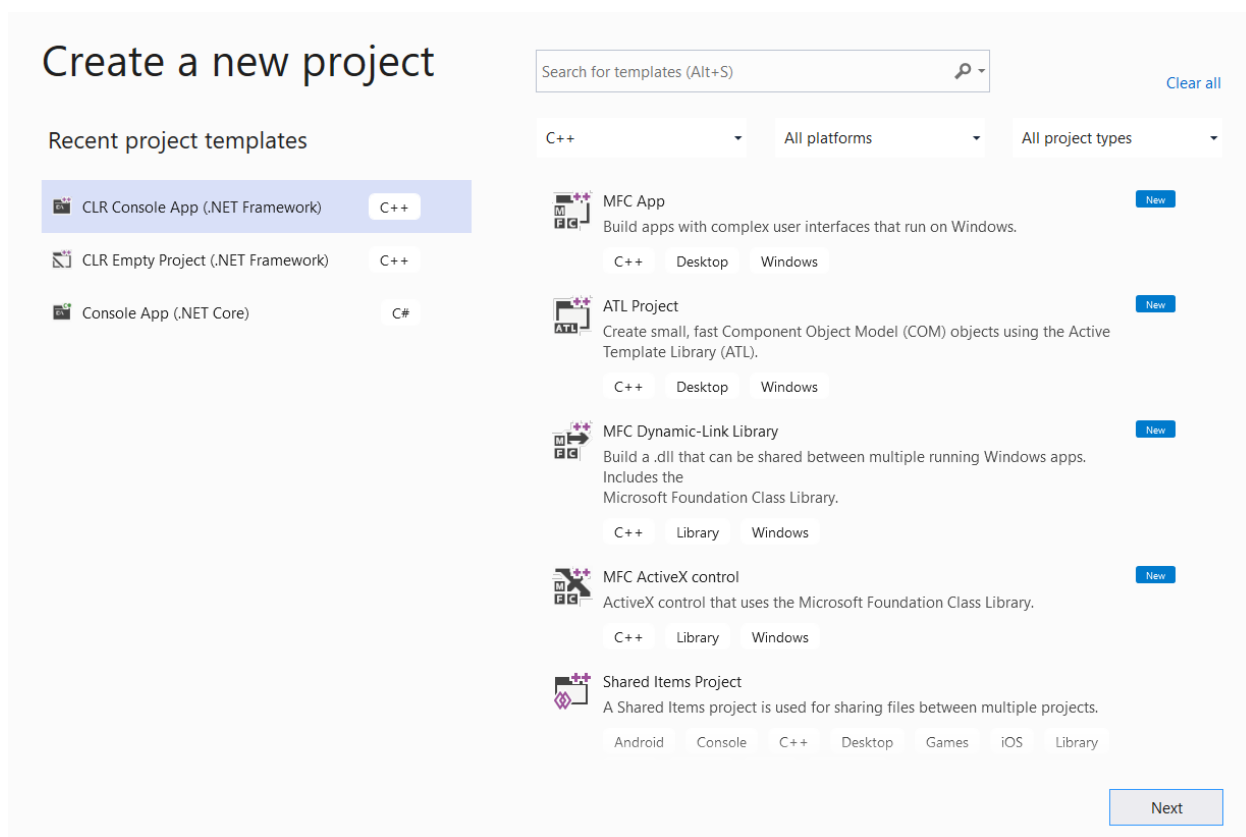


Рис. 10. Создание проекта

Шаг 2. Жмем на проект -> Добавить -> Создать элемент. Выбираем UI -> Форма Windows Forms.

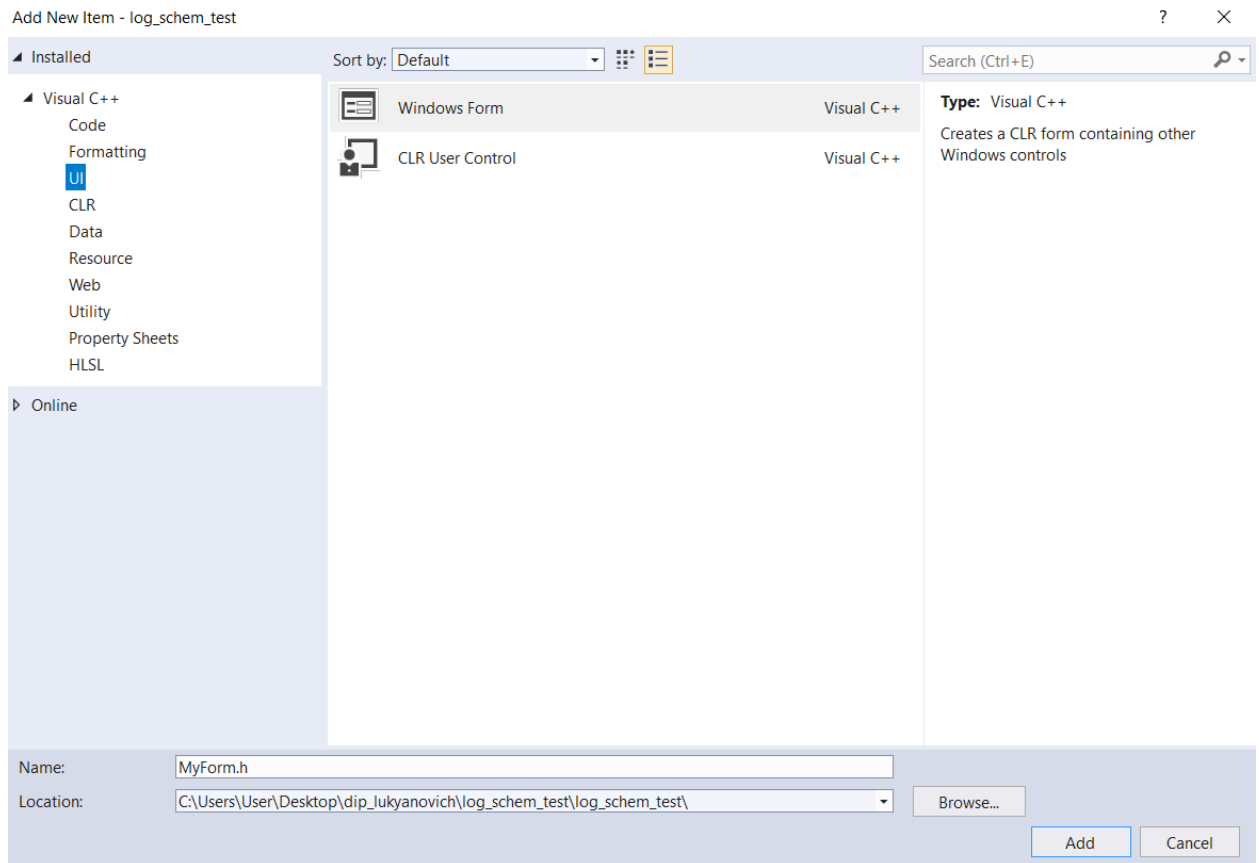


Рис. 11. Добавление элемента Windows Forms в проект

Шаг 3. Дописываем в .cpp стартовый код

```

log_schem_test (Global Scope)
1  #include "pch.h"
2
3  using namespace System;
4  using namespace System::Windows::Forms;
5
6  int main(array<System::String ^> ^args)
7  {
8      Application::EnableVisualStyles();
9      Application::SetCompatibleTextRenderingDefault(false);
10     log_schem_test::form_start start;
11     Application::Run(% start);
12
13     return 0;
14 }
15

```

Рис. 12. Точка входа в программу

4.3 Проектирование графического интерфейса САПТ КС

Графический интерфейс пользователя (англ. *Graphical user interface, GUI*) – разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода – клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Приложение Windows Forms представляет собой событийно-ориентированное приложение, поддерживаемое Microsoft .NET Framework. В отличие от пакетных программ, большая часть времени тратится на ожидание от пользователя каких-либо действий, как, например, ввод текста в текстовое поле или клика мышкой по кнопке.

Для пользователя одним из принципиальных преимуществ работы с Windows является то, что большинство имеющихся приложений выглядят и ведут себя сходным образом. Действие (Action) – реализация некоторого поведения, являющегося реакцией на поступок пользователя, такой, как щелчок на кнопке или разделе меню – инициаторе действия или интерфейсном компоненте действия. Примерами действий являются открытие файла и его загрузка в какой-то приемник, сохранение информации в файле на диске, установка атрибутов шрифта и выравнивание текстов в окнах редактирования, поиск и замена фрагментов текста, навигация по набору данных, выполнение какого-то внешнего приложения, вызов справки и многое другое.

Помимо стандартных действий в реальных приложениях имеются, конечно, и нестандартные, связанные с какими-то расчетами, обработкой данных и т.п. Объекты таких действий надо создать методами визуального программирования и для них надо написать обработчики, реализующие данные действия. Таким образом, прежде чем начинать программирование приложения, надо составить для себя список действий, которые должны быть доступны будущему пользователю через разделы меню, инструментальные панели,

кнопки и другие элементы управления. В Microsoft Visual Studio в приложениях Windows Forms создание полос действий, кнопок и прочих визуальных компонентов происходит с помощью панели элементов простым перетаскиванием необходимых компонентов на форму с помощью мыши. Интерфейсные компоненты действий обычно должны содержать поясняющие их изображения. Эти изображения можно присвоить каждому из элементов непосредственно при настройке свойств (имя, текст, ширина, высота) выбрав необходимый файл с изображением и привязав к нему кнопку. Таким образом, последовательность формирования списка действий, проектирования меню и инструментальных панелей сводится к следующим шагам:

1. Продумывается и составляется список действий, которые должны быть доступны будущему пользователю через разделы меню, инструментальные панели, кнопки и другие элементы управления.

2. Каждому действию задается набор характеристик: Name (имя), Text (надпись в которой выделяется символ быстрого доступа), ShortcutKeys (горячие клавиши), Image (изображения для нестандартных кнопок), ToolTipText (всплывающие тексты подсказок), ссылки на тему справки и др.

Все эти параметры можно легко задать в свойствах используемого визуального компонента. Выбор свойств обширный и включает в себя также положение, размер, выравнивание и прочее (Рис.13).

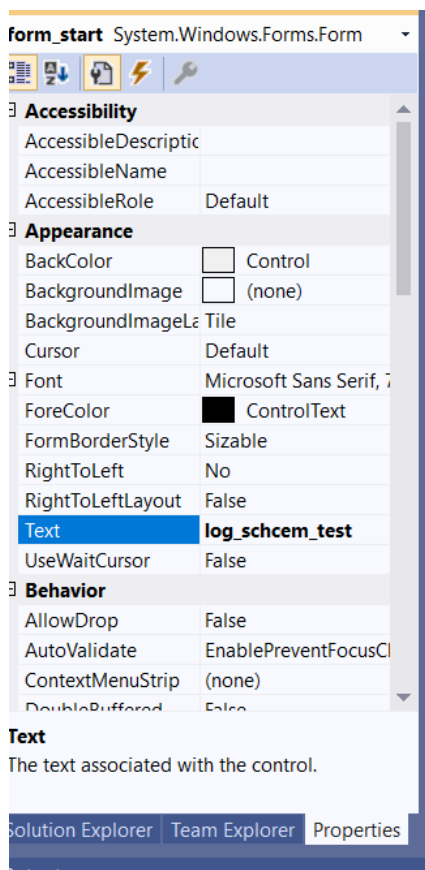


Рис. 13. Свойства объекта Windows Forms

3. Записываются обработчики событий выполнения для всех нестандартных действий.
4. На форму переносится компонент ToolStrip, который будет выполнять функцию главного меню. С помощью ToolStripDropDownButton создаются разделы и подразделы меню, присваиваются необходимые изображения для пунктов меню, названия и прочее.
5. С помощью еще одного ToolStrip создается панель инструментов. Создаются ToolStripButton кнопки для необходимых действий, им присваиваются изображения, название, горячие клавиши и др.
6. Далее непосредственно на форму переносятся другие необходимые компоненты такие, как TextBox, StatusStrip.

Все компоненты из меню, панелей и рабочего окна переносятся мышью из панели элементов Windows Forms (Рис 14).

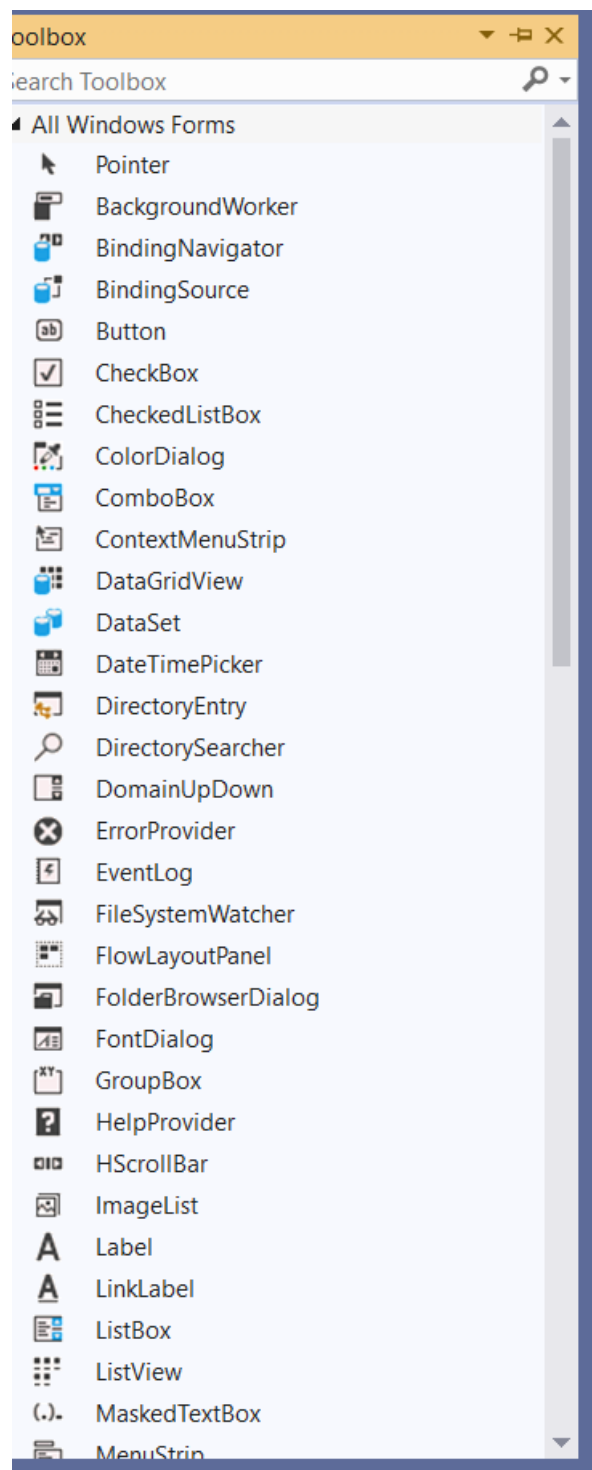


Рис. 14. Панель элементов Windows Forms

5. ОПИСАНИЕ ПРОГРАММНЫХ СРЕДСТВ

Программное средство является одной из подсистем системы VLSI-SIM автоматизированного моделирования больших интегральных схем и предназначено для анализа на полноту и построения тестов логических схем в классе константных неисправностей на основе параллельного моделирования неисправностей. Построение теста может быть выполнено как на основе вероятностного метода, так и направленным (регулярным) методом, представляющим собой модификацию метода существенных путей со структурным подходом к вычислению условий образования существенных путей и их обеспечению [4]. Вероятностный метод, также как и анализ тестов на полноту, основан на параллельном моделировании неисправностей [2].

Программное средство разработано на языке Visual C++ в среде Visual Studio 2019 с использованием библиотек MFC(v142) и ATL(v142) для 32-х разрядных систем под управлением ОС семейства Windows.

5.1. Структура данных подсистемы

Ниже приводится описание основных информационных и рабочих (промежуточных) массивов программной подсистемы, используемых и формируемых различными функциями подсистемы.

Массивы описания множеств конъюнкций систем ДНФ функций, реализуемых логическими элементами схемы (m11 и m12)

Используются для представления совокупности троичных матриц T_i , описывающих множества конъюнкций, входящих в системы ДНФ функций, реализуемых логическими элементами схемы. Массивы целого типа размером 9000. Размер выбран исходя из следующих соображений. Число входов у логического элемента не должно превышать 32. В этом случае число элементов в строке матрицы T_i не превосходит 32. Каждый элемент матрицы T_i представляется парой булевых констант: $1 \rightarrow 10, 0 \rightarrow 01, - \rightarrow 00$. Тогда каждая строка матрицы T_i может быть представлена парой булевых векторов таким образом, что первый вектор содержит первые компоненты из пар булевых констант, кодирующих троичные элементы строки, а второй вектор – вторые компоненты. Следовательно, матрица T_i представляется парой булевых матриц M_i^1 и M_i^2 . Массив m11 предназначен для хранения последовательности матриц $M_1^1, M_2^1, \dots, M_{ns}^1$, а массив m12 – матриц $M_1^2, M_2^2, \dots, M_{ns}^2$. Здесь и далее ns – число логических элементов в схеме. Матрицы расположены в массивах последовательно в соответствии с возрастанием номеров логических элементов,

которым они соответствуют. Строка матрицы представляется одним элементом массива. Таким образом, элемент массива рассматривается как булев вектор. Учитывая размер массивов $m11$ и $m12$ (9000 элементов), видим, что суммарное число конъюнкций в системах ДНФ функций, реализуемых логическими элементами схемы, не должно превышать 9000.

Массив описания множества матриц, задающих отношения вхождения конъюнкций в ДНФ функций, реализуемых логическими элементами схемы ($m2$)

Используется для представления совокупности булевых матриц B_i , задающих отношения вхождения конъюнкций в ДНФ функций, реализуемых логическими элементами схемы. Массив целого типа размером 9000. Матрица B_i задает отношение вхождения конъюнкций, представленных троичной матрицей T_i (или парой булевых матриц M_1 и M_2) в ДНФ функций, реализуемых i -м логическим элементом. Строки матрицы B_i соответствуют конъюнкциям из T_i , а столбцы – ДНФ, реализуемым на выходах логического элемента. Элемент b_{kj}^i матрицы B_i принимает значение 1 тогда и только тогда, когда конъюнкция, представленная k -й строкой матрицы T_i , входит в ДНФ, реализуемую на j -м выходе логического элемента. Массив $m2$ предназначен для хранения последовательности матриц B_1, B_2, \dots, B_n . Матрицы расположены в массиве последовательно в соответствии с возрастанием номеров логических элементов, которым они соответствуют, т.е. в том же порядке, что и матрицы T_i в массивах $m11$ и $m12$. Строка матрицы представляется одним элементом массива.

Таким образом, элемент массива рассматривается как булев вектор. Следовательно, число выходов у каждого элемента схемы не должно превышать 32.

Секционированный массив для массивов $m11$, $m12$, $m2$ (sm)

Массив целого типа размером ns . Используется для секционирования массивов $m11$, $m12$, $m2$. При этом i -й элемент массива sm ($i = 0, 1, 2, \dots$) задает номер элемента массивов $m11$ и $m12$ ($m2$), начиная с которого расположено описание конъюнкций (отношения вхождения конъюнкций в ДНФ), соответствующих $(i+1)$ -му логическому элементу схемы.

Массив, задающий число входных полюсов у элементов, из которых построена схема (ml)

Массив целого типа размером ns . Число входных полюсов у логического элемента с номером i ($i = 1, 2, \dots$) представляется $(i-1)$ -м элементом массива ml .

Массив, задающий число выходных полюсов у элементов, из которых построена схема (mn)

Массив целого типа размером ns . Число выходных полюсов у логического элемента с номером i ($i = 1, 2, \dots$) представляется $(i-1)$ -м элементом массива mp .

Массив, задающий число различных конъюнкций в системах ДНФ функций, реализуемых логическими элементами схемы (mh)

Массив целого типа размером ns . Число различных конъюнкций в системе ДНФ функций, реализуемых логическим элементом с номером i ($i=1,2,\dots$), представляется $(i-1)$ -м элементом массива mh .

Массив описания входных полюсов логических элементов, из которых построена схема (nvx)

Массив целого типа размером $sum1$. Здесь $sum1$ – суммарное число входов у элементов в схеме. Номер любого входного полюса логического элемента схемы определяется номером входного полюса схемы или выходного полюса другого логического элемента, который соединен с данным входным полюсом элемента. Номера входных полюсов логического элемента с номером i представляются последовательными элементами массива nvx , начиная с элемента $s_1+s_2+\dots+s_{i-1}$, где s_i – число входных полюсов у логического элемента с номером i .

Секционирующий массив для массива nvx (svx)

Массив целого типа размером ns . Используется для секционирования описаний входных полюсов логических элементов в массиве nvx . Номер элемента массива nvx , начиная с которого располагается описание входных полюсов логического элемента с номером i , представляется $(i-1)$ -м элементом массива svx , который, таким образом, равен $s_1+s_2+\dots+s_{i-1}$.

Массив описания выходных полюсов логических элементов, из которых построена схема ($nvix$)

Массив целого типа размером $\sum_{i=1}^{ns} t_i$, где t_i – число выходных полюсов у элемента с номером i . Выходные полюсы элементов схемы нумеруются последовательностью целых чисел, начиная с $l+1$ (l – число входных полюсов в схеме), последовательно переходя от элемента с номером 1 к элементу с номером 2 и т. д. Номера выходных полюсов элемента с номером i представляются последовательными элементами массива $nvix$, начиная с элемента $t_1+t_2+\dots+t_{i-1}$ и кончая элементом $t_1+t_2+\dots+t_i-1$.

Секционирующий массив для массива $nvix$ ($svix$)

Массив целого типа размером ns . Используется для секционирования описаний выходных полюсов элементов в массиве $nvix$. Номер элемента мас-

сива $nvix$, начиная с которого располагается описание выходных полюсов элемента с номером i , представляется $(i-1)$ -м элементом массива $svix$, который, таким образом, равен $t_1+t_2+\dots+t_{i-1}$.

Массив описания выходных полюсов логической схемы ($nvixc$)

Массив целого типа размером n , где n – число выходных полюсов в схеме. Используется для представления списка выходных полюсов логической схемы, т.е. номеров выходных полюсов элементов, входящих в схему, которые соединены с выходными полюсами схемы.

Массив, содержащий список неисправностей исходной логической схемы (cf)

Массив целого типа размером $1+4 \times ns$. Используется для представления списка константных неисправностей на входных полюсах схемы, на входных и выходных полюсах элементов, входящих в схему. Список представлен в массиве cf следующим образом. Входному полюсу схемы с номером i соответствует $(i-1)$ -й элемент массива. Первый бит элемента массива соответствует неисправности типа "константа 0" на i -м входном полюсе схемы, второй бит – неисправности типа "константа 1". Если соответствующий бит, равен 1, то неисправность включена в список. Каждому элементу схемы в массиве cf отводятся четыре элемента массива. Если элемент имеет номер i , то это $[1+4(i-1)]$ -й, $[1+4(i-1)+1]$ -й, $[1+4(i-1)+2]$ -й и $[1+4(i-1)+3]$ -й элементы массива. Здесь 1 – число входных полюсов в схеме. В $[1+4(i-1)]$ -м элементе массива представлен список неисправностей типа "константа 0" на входных полюсах элемента, в $[1+4(i-1)+1]$ -м элементе массива – список неисправностей типа "константа 1" на входных полюсах элемента, в $[1+4(i-1)+2]$ -м и $[1+4(i-1)+3]$ -м элементах массива – на выходных полюсах элемента. Первый бит $[1+4(i-1)]$ -го элемента массива соответствует неисправности типа "константа 0" на первом входном полюсе (согласно тому порядку, в котором входные полюсы элемента описаны в массиве nvx), второй бит – неисправности типа "константа 0" на втором входном полюсе, третий бит – неисправности типа "константа 0" на третьем входном полюсе, четвертый бит – неисправности типа "константа 0" на четвертом входном полюсе и т.д. Аналогично, остальные элементы массива используются для представления списков соответствующих константных неисправностей на входных и выходных полюсах элемента. Единичное значение соответствующего бита определяет включение неисправности в список.

Массив, содержащий тест схемы (tst)

Массив целого типа. Предназначен для хранения последовательности пар входной набор – выходной набор, представляющей собой проверяющий

тест. Для каждого входного набора отводится $\lceil l/32 \rceil$ элементов массива. Выходной набор занимает $\lceil n/32 \rceil$ элементов. Здесь n – число выходных полюсов у схемы. Пары входной набор – выходной набор теста располагаются в массиве *tst* последовательно. Каждая пара занимает $\lceil l/32 \rceil + \lceil n/32 \rceil$ элементов. В первых $\lceil l/32 \rceil$ элементах, начиная с первого бита, расположен входной набор. В следующих $\lceil n/32 \rceil$ элементах, начиная с первого бита, расположен соответствующий выходной набор.

Массив с описанием элементов логической сети на основе перечисления их входов (okb)

Массив целого типа размером 20000. Содержит описание элементов логической сети, ориентированной на применение регулярного метода построения теста (преобразование исходной логической схемы к данной логической сети производится функцией *modsx*). Описания элементов логической сети располагаются в массиве *okb* последовательно в соответствии с возрастанием номеров логических элементов. Описание каждого элемента представляет собой последовательность: номер логического элемента (совпадает с номером его выхода), тип элемента, число входов у элемента, номер первого входа (совпадает с номером входного полюса сети или выхода другого логического элемента, который непосредственно соединен с данным входом), номер второго входа и т.д. Таким образом, описание каждого логического элемента занимает $3 + l_i$ элементов массива *okb*, где l_i – число входов у i -го логического элемента сети.

Секционирующий массив для массива okb (cokb)

Массив целого типа размером 9000. Используется для секционирования описаний логических элементов в массиве *okb*. Номер элемента массива *okb*, начиная с которого располагается описание логического элемента с номером i , представляется i -м элементом массива *cokb*.

Массив с описанием связей выходов элементов и входных полюсов логической сети (oid)

Массив целого типа размером 9000. Содержит описание связей выходов логических элементов и входных полюсов с входами других логических элементов в логической сети, ориентированной на применение регулярного метода построения теста. Списки связей для входных полюсов сети и выходов логических элементов располагаются в массиве *oid* последовательно в соответствии с возрастанием их номеров. Описание отдельного списка связей представляет собой последовательность: номер входного полюса сети или выхода логического элемента; число других логических элементов, с входами ко-

торых непосредственно связан данный входной полюс сети или выход логического элемента; номер первого логического элемента; номер второго логического элемента и т.д. Таким образом, отдельный список связей занимает $2+v_i$ элементов массива `oid`, где v_i – число связей у выхода i -го логического элемента или входного полюса логической сети.

Секционирующий массив для массива `oid` (`coid`)

Массив целого типа размером 9000. Используется для секционирования списков связей в массиве `oid`. Номер элемента массива `oid`, начиная с которого располагается описание списка связей i -го входного полюса логической сети или выхода логического элемента, представляется i -м элементом массива `coid`.

Массив описания выходных полюсов логической сети (`vi`)

Массив целого типа размером 300. Используется для представления списка выходных полюсов логической сети, т.е. номеров выходных полюсов логических элементов, входящих в сеть, которые соединены с выходными полюсами сети. Данный массив соответствует массиву `pvixs` в описании исходной логической схемы.

Массив основных параметров логической сети (`pc`)

Массив целого типа размером 25. Используется для представления ряда параметров логической сети:

- `pc[0]` – число наборов, включенных в тест;
- `pc[5]` – число выходных полюсов в сети;
- `pc[6]` – число логических элементов в сети;
- `pc[14]` – число входных полюсов в сети.

Остальные элементы массива в настоящее время не используются.

Массив, содержащий список неисправностей логической сети (`mne`)

Массив целого типа размером 9000. Используется для представления списка константных неисправностей логической сети, соответствующего списку неисправностей исходной логической схемы, представленному массивом `cf`. Список представлен в массиве `mne` следующим образом. Линии сети с номером i (номер линии в сети совпадает с номером соответствующего входного полюса сети или выхода логического элемента сети) соответствует $(i-1)$ -й элемент массива `mne`. 32-й бит элемента массива соответствует неисправности типа "константа 0" на i -й линии сети, 31-й бит – неисправности типа "константа 1". Если соответствующий бит равен 1, то неисправность включена в список.

Массив описания соответствия между номерами линий в исходной схеме и логической сети (сн)

Массив целого типа размером. Используется для описания соответствия между номерами линий в исходной логической схеме и логической сети, ориентированной на применение регулярного метода построения теста. Если i – номер некоторой линии в исходной схеме (номер линии совпадает с номером соответствующего входного полюса схемы или выходного полюса логического элемента), то в i -м элементе массива сн находится номер некоторой линии логической сети, которая соответствует i -й линии исходной схемы.

Массив описания соответствия между номерами линий в логической сети и в исходной схеме (пс)

Массив целого типа размером 9000. Содержит описание соответствия между номерами линий в логической сети, ориентированной на применение регулярного метода построения теста, и исходной логической схеме. Если i – номер некоторой линии в логической сети, то в i -м элементе массива пс находится номер некоторой линии исходной схемы, которая соответствует i -й линии логической сети (если такой линии в исходной схеме нет, то в i -м элементе массива находится 0).

Массив, содержащий D-куб неисправности (f)

Массив символьного типа. Используется для представления D-куба рассматриваемой неисправности. i -й элемент массива соответствует i -й линии логической сети. D-куб неисправности описывается в алфавите $\{0, 1, x, D, B\}$, где B соответствует \bar{D} .

Массив, содержащий D-строку после вычисления условий транспортировки неисправности до выходных полюсов сети (fa)

Массив символьного типа. Имеет такую же структуру, что и массив f.

Массив, содержащий D-строку после обеспечения условий транспортировки неисправности до выходных полюсов сети (fd)

Массив символьного типа. Имеет такую же структуру, что и массив f.

Массив для запоминания варианта условий транспортировки неисправности до выходных полюсов сети (та)

Массив целого типа размером $pc[6] + 1 + 1$. Используется для запоминания вариантов условий транспортировки неисправности через логические элементы, входящие в сложный путь от места неисправности до выходных полюсов сети. i -й элемент массива та соответствует i -му элементу массива ба, в котором находится список логических элементов сложного пути, которые рассматривались при вычислении условий транспортировки неисправности до выходных полюсов сети. При этом $ta[i] = 0$, если i -й логический элемент из

массива ba не участвует в вычислении условий; $ma[i] = 1$ – если выбран первый возможный вариант условий; $ma[i] = 2$ – если выбран второй вариант условий.

Массив для запоминания списка элементов сложного пути, которые использовались при вычислении условий транспортировки неисправности до выходных полюсов сети (ba)

Массив целого типа размером $pc[6] + 1 + 1$. Используется для представления списка логических элементов, принадлежащих сложному существенному пути (сложный существенный путь хранится в массиве $nlin$), которые действительно использовались при вычислении условий транспортировки неисправности до выходных полюсов сети. Номера логических элементов расположены в массиве ba в том же порядке, что и в массиве $nlin$.

Массив для хранения сложного пути ($nlin$)

Массив целого типа размером $pc[6] + 1 + 1$. Используется для представления списка логических элементов, принадлежащих сложному пути (т.е. совокупности всех простых путей) от места неисправности до выходных полюсов сети. В 0-м элементе массива $nlin$ находится номер неисправного элемента или входного полюса логической сети. Номера логических элементов расположены в массиве в порядке возрастания номеров.

Массив для хранения списка логических элементов, на выходах которых необходимо обеспечить заданные значения сигналов (bd)

Массив целого типа размером $pc[6] + 1 + 1$. Используется для представления списка логических элементов, на выходах которых необходимо обеспечить заданные значения сигналов (список элементов для выполнения операции расширения фиксации при обеспечении условий образования существенного пути). Номера логических элементов расположены в массиве bd в том порядке, в котором они появлялись при вычислении условий транспортировки неисправности до выходных полюсов сети.

Массив для запоминания вариантов обеспечения определенных значений сигналов на выходах заданных элементов (md)

Массив целого типа размером 4000. Используется для запоминания вариантов обеспечения определенных значений сигналов на выходах заданных логических элементов при выполнении операции расширения фиксации. i -й элемент массива md соответствует i -м элементам массивов bdr и $bdr1$, в которых находится список логических элементов, на выходах которых необходимо обеспечить заданные значения сигналов. При этом $md[i] = 0$, если на выходе i -го логического элемента сети не нужно обеспечивать определенное значение сигнала (т.е. i -й логический элемент не принадлежит списку) или же на выходе

i -го элемента определенное значение сигнала еще не обеспечивалось; $md[i] = 1$ – если выбран первый возможный вариант обеспечения заданного значения сигнала на выходе логического элемента; $md[i] = 2$ – если выбран второй возможный вариант обеспечения заданного значения сигнала на выходе логического элемента или заданное значение сигнала уже обеспечено (т.е. обеспечено автоматически).

Массивы для хранения списка логических элементов, на выходах которых необходимо обеспечить заданные значения сигналов (bdr и $bdr1$)

Массив целого типа размером 4000. Используются для представления списка логических элементов, на выходах которых необходимо обеспечить заданные значения сигналов. Отличаются от массива bd представлением списка элементов. Если логический элемент с номером i принадлежит списку, то $bdr[i] = bdr1[i] = 1$, в противном случае – $bdr[i] = bdr1[i] = 0$.

Кроме описанных выше массивов используется также ряд переменных, которые задают основные параметры логической схемы:

- l – переменная целого типа, задающая число входных полюсов в исходной схеме;
- n – переменная целого типа, задающая число выходных полюсов в исходной схеме;
- ns – переменная целого типа, задающая число логических элементов в исходной схеме;
- nt – переменная целого типа, задающая длину теста в массиве tst , т.е. число наборов включенных в тест;
- $nhcf$ – переменная целого типа, задающая число неисправностей в исходном списке неисправностей, представленном массивом cf ;
- $nhcf1$ – переменная целого типа, задающая число неисправностей из списка, представленного массивом cf , которые не обнаруживаются тестом, помещенным в массив tst .

5.2. Описание основных функций подсистемы анализа и построения тестов вероятностным методом

Программный модуль включает в себя ряд функций, реализованных на языке C++. Ниже приводится их краткое описание:

`convertor` – конвертирует описание схемы, полученное в результате трансляции и содержащееся в файле `имя_схемы.tsc`, в совокупность массивов, которые представляют собой описание схемы;

`anal` – вычисляет множество неисправностей из заданного списка, проверяемых заданной последовательностью входных наборов;

`compe` – реализует алгоритм параллельного моделирования неисправностей из заданного списка;

`iskl` – производит подсчет числа неисправностей, проверяемых заданным входным набором, и удаляет соответствующие неисправности из списка;

`simul` – вычисляет значения сигналов на линиях 32 экземпляров схемы;

`PatGener` – генерирует и анализирует входной набор;

`vermet` – строит тест вероятностным методом;

`intst` – включает тест, в массив с тестом схемы;

`printtst` – выводит в текстовый файл `имя_схемы.tsd` тест схемы в документированной форме (в виде таблицы);

`printf` – выводит в текстовый файл `имя_схемы.fff` список непроверенных неисправностей в документированной форме.

Описание функции `convertor`

Назначение

Преобразование описания схемы, полученного в результате трансляции (файл `имя_схемы.tsc`), в структуру данных, требуемую для программ параллельного моделирования неисправностей.

Исходные данные

Файл `имя_схемы.tsc`, содержащий результаты трансляции заданной схемы с именем `имя_схемы`.

Результаты

Массивы и переменные `m11`, `m12`, `m2`, `sm`, `ml`, `mn`, `mh`, `nvx`, `nvix`, `svix`, `nvixc`, `cf`, `l`, `n`, `ns`, `nhcf`.

Описание функции `anal`

Назначение

Функция вычисляет множество неисправностей из заданного списка, проверяемых заданной последовательностью входных наборов.

Исходные данные

Массивы и переменные: m11, m12, m2, sm, ml, mn, mh, nvx, svx, nvix, svix, nvixc, cf, l, n, ns, nt, nhcf, nhcf1, tst, fa, hWnd, где hWnd – дескриптор окна, используемый для передачи сообщений.

Результаты

Текстовые файлы: имя_схемы.pte – содержит протокол анализа заданного теста на полноту.

Массивы и переменные: tst, nhcf, nhcf1.

Обращение

anal(hWnd).

Вызываемые функции

comne.

Алгоритм

Последовательно выбираются входные наборы теста и для каждого входного набора выполняется следующее.

Инициализируются значения входных полюсов для заданного входного набора, обнуляет счетчик обнаруживаемых неисправностей (nh) и выполняется процедура параллельного моделирования неисправностей из заданного списка (обеспечивается обращением к функции comne).

После завершения моделирования производится включение заданного входного набора и соответствующей ему выходной реакции исправной схемы в тест схемы (обеспечивается обращением к функции intst).

Далее производится вывод результатов моделирования неисправностей на заданном входном наборе (количество неисправностей, обнаруживаемых данным входным набором – nh, и общее количество неисправностей, обнаруженных введенной тестовой последовательностью – nhcf - nhcf1, входной набор и соответствующая ему выходная реакция) в файл протокола имя_схемы.pte.

Описание функции comne

Назначение

Функция реализует параллельное моделирование неисправностей принадлежащих заданному списку, на заданном входном наборе.

Исходные данные

Массивы и переменные: m11, m12, m2, sm, ml, mn, mh, nvx, svx, nvix, svix, nvixc, cf, l, n, ns, nh.

Результаты

Массивы и переменные: nh, cf.

Обращение

comne()

Вызываемые функции

simul, iskl.

Алгоритм

1. Устанавливается признак моделирования неисправностей типа "константа 0" (переменная $pr = 0$) и обнуляется счетчик неисправностей, обнаруживаемых входным набором ($nh = 0$).

2. Подготавливаются маски неисправностей для моделирования исправной схемы ($rap[i] = -1$, т.е. неисправности не вносятся ни на один полюс).

3. Моделируется исправная схема, что достигается обращением к функции simul.

4. Подготавливаются значения сигналов на линиях схемы для моделирования схемы с неисправностями.

5. Организуется цикл по элементам массива cf , задающего список неисправностей (от 1 до $4 \times ns + 1$), для формирования групп неисправностей, каждая из которых содержит не более 31 неисправности, моделируемых одновременно.

6. Строятся маски неисправностей $rap[i]$ для входов схемы (в одном цикле моделируется до 31 неисправности): если значение сигнала на входе отличается от типа вносимой неисправности и такая неисправность еще не обнаружена, то неисправность включается в группу.

7. Повторяются действия, описанные в пункте 6, но уже для элементов схемы. Однако имеются отличия в вычислении номеров элементов массива cf (используются не номера полюсов, а номера функциональных элементов с учетом количества элементов массива cf , отводимых под неисправности одного элемента схемы).

8. После завершения формирования очередной группы неисправностей выполняется ее моделирование, находятся неисправности, которые проверяются, и они удаляются из списка (это достигается путем обращения к функциям simul и iskl).

9. При наличии в списке еще непроверенных неисправностей осуществляется переход к пункту 6.

10. Устанавливается признак моделирования неисправностей типа "константа 1" ($pr=1$) и осуществляется повторение пунктов 2-10, за исключением того, что в пункте 2 маски формируются таким образом, что $rap[i]=0$, т.е. векторы для внесения неисправностей на полюсы схемы вначале не содержат неисправностей.

Описание функции **iskl**

Назначение

Функция выполняет подсчет числа неисправностей, проверяемых заданным входным набором, и удаляет соответствующие неисправности из списка.

Исходные данные

Массивы и переменные: *ml*, *mn*, *cf*, *nvixc*, *l*, *n*, *ns*, *nh*, *ncf*, *rdcf*, *pr*.

Результаты

Массивы и переменные: *cf*, *nh*.

Обращение

iskl(*pr*).

Вызываемые функции

simul.

Алгоритм

Вызывается функция *simul*, которая выполняет параллельное моделирование очередной группы неисправностей на основе простых итераций, и производится подсчет проверенных неисправностей (для этого используется вектор обнаружимости *w[i]*, в котором *i*-я компонента равна 1, если *i*-я неисправность из группы проверяется, или равна 0 – в противном случае). Неисправность считается проверенной (*w[i]=1*), если состояние какого-либо выхода схемы в исправной и неисправной схемах отличается и имеет значение 0 или 1.

После формирования вектора обнаружимости производится исключение проверенных неисправностей из списка (массив *cf*) и увеличивается счетчик числа проверенных неисправностей (*nh*) на количество ненулевых компонент вектора обнаружимости *w*. Затем происходит подготовка массивов для моделирования следующей группы неисправностей.

Описание функции **simul**

Назначение

Функция вычисляет значения сигналов на линиях в 32 экземплярах логической схемы (с учетом признака моделирования в отдельные экземпляры схемы могут вноситься неисправности типа «константа 0» или «константа 1») на основе синхронного моделирования на заданном входном наборе.

Исходные данные

Массивы и переменные *m11*, *m12*, *m2*, *sm*, *ml*, *mn*, *mh*, *nvx*, *svx*, *nvix*, *svix*, *nvixc*, *l*, *n*, *ns*, *nh*.

Результаты

Массив *z* описывающий сигналы в линиях схемы, полученные в результате моделирования.

Обращение

simul(pr)

Вызываемые функции

Нет.

Алгоритм

Функция реализует синхронное моделирование элементов схемы в порядке возрастания их номеров.

Для моделирования функциональных элементов использованы формулы, приведенные в подразделе 3.3.

Описание функции PatGener

Назначение

Функция генерирует и анализирует (вычисляет множество неисправностей проверяемых набором) случайный входной набор.

Исходные данные

Массивы и переменные: ch, lp.

Результаты

Массивы и переменные: nh, nn, tst, nhcf1.

Обращение

PatGener(ch,lp).

Вызываемые функции

anal.

Алгоритм

Используя датчик случайных чисел, генерируется входной набор. Затем, выполняется анализ входного набора.

Описание функции vermet

Назначение

Построение теста вероятностным методом.

Исходные данные

Массивы и переменные: l, nn, nhcf, nh, lp, структура mps, из которой считываются параметры M1, M2, M3, и дескриптор окна hWnd.

Результаты

Массивы и переменные: nh, nn, tst, nhcf1.

Обращение

vermet(mps).

Вызываемые функции

PatGener.

Алгоритм

Функция реализует алгоритм, описанный в разделе 2.

Описание функции `intst`

Назначение

Включает тест (входной набор и соответствующий ему выходной набор) в массив с тестом схемы `tst`.

Исходные данные

Переменные `l` и `n`, а также массивы символьного типа `InpPat` и `OutPat`, содержащие входной и выходной наборы теста, соответственно.

Результаты

Массив `tst`, содержащий тест схемы, в конец которого добавлен заданный тестовый набор.

Обращение

`intst()`.

Вызываемые функции

Нет.

Описание функции `printtst`

Назначение

Выводит в текстовый файл `имя_схемы.tsd` тест схемы в документированной форме (в виде таблицы).

Исходные данные

Переменные `l` и `n`, массив `tst`.

Результаты

Текстовый файл `имя_схемы.tsd`, содержащий тест схемы в документированной форме.

Обращение

`printtst()`.

Вызываемые функции

Нет.

Описание функции `printcf`

Назначение

Выводит в текстовый файл `имя_схемы.fff` список непроверенных неисправностей в документированной форме.

Исходные данные

Массивы и переменные `l`, `ns`, `ml`, `mn`, `nvx`, `svx`, `nvix`, `svix`, `cf`, `nhcf1`.

Результаты

Текстовый файл `имя_схемы.fff`, содержащий список непроверенных неисправностей в документированной форме.

Обращение

`printcf()`.

Нет.

5.3. Описание основных функций подсистемы построения тестов регулярным методом

Ниже приводится краткое описание функций подсистемы:

`modsx` – строит модель схемы, ориентированную на решение основных задач, возникающих при вычислении проверяющих наборов теста на основе регулярного метода;

`kode` – строит список неисправностей логической сети, ориентированной на применение регулярного метода построения теста;

`regme` – реализует общую схему направленного метода построения теста для заданного списка неисправностей логической сети;

`dkni` – строит D-кубы неисправностей логического элемента, реализующего конъюнкцию;

`dknili` – строит D-кубы неисправностей логического элемента, реализующего дизъюнкцию;

`dknne` – строит D-кубы неисправностей логического элемента, реализующего отрицание;

`clput` – построение сложного пути, т.е. совокупности простых путей, от места неисправности до выходных полюсов сети;

`ucltr` – вычисляет условия образования существенного пути от места неисправности до выходных полюсов логической сети;

`uti` – вычисляет условия образования существенного пути через логический элемент, реализующий конъюнкцию;

`utili` – вычисляет условия образования существенного пути через логический элемент, реализующий дизъюнкцию;

`utne` – вычисляет условия образования существенного пути через инвертор;

`orascf` – вычисляет входной набор (возможно не полностью определенный) логической сети, обеспечивающий найденные функцией `ucltr` условия образования существенного пути;

`orfi` – вычисляет вариант значений сигналов на входах логического элемента, реализующего конъюнкцию, обеспечивающих заданное значение сигнала на выходе элемента;

`orfil` – вычисляет вариант значений сигналов на входах логического элемента, реализующего дизъюнкцию, обеспечивающих заданное значение

сигнала на выходе элемента;

orfne – вычисляет значение сигнала на входе логического элемента, реализующего отрицание, обеспечивающего заданное значение сигнала на выходе логического элемента.

Следует отметить, что подсистема выполняет не только построение теста направленным методом, но и анализирует тест на основе параллельного моделирования неисправностей; соответствующие функции были описаны выше.

Описание функции modcx

Назначение

Построение модели исходной схемы, ориентированной на решение основных задач, возникающих при вычислении проверяющих наборов теста на основе регулярного метода (вычисление условий транспортировки влияния неисправности через элементы, обеспечение заданных значений сигналов в узлах сети на основе решения логических уравнений методом отраженных волн и др.).

Исходные данные

Массивы и переменные: m11, m12, m2, sm, ml, mn, mh, nvx, svx, nvix, svix, nvixc, cf, l, n, ns, hWnd. Дескриптор окна – hWnd, используется для передачи сообщений.

Результаты

1. Массивы и переменные, задающие описание логической сети, ориентированной на применение регулярного метода построения теста: okb, sokb, mne, vi, pc, oid, coid.

2. Массивы nc и cn, задающие соответствие между линиями исходной схемы и линиями логической сети.

3. Результаты работы функции regme, к которой строится обращение.

Обращение

modcx (hWnd).

Вызываемые функции

kodne, regme.

Алгоритм

В программе последовательно в соответствии с нумерацией рассматриваются элементы исходной схемы и заменяются функционально эквивалентными сетями из элементов классического базиса (конъюнктор, дизъюнктор, инвертор). Предполагается, что элементы исходной схемы правильно пронумерованы числами от 1 до ns, т.е. номер любого элемента больше, чем номера элементов, некоторые выходы которых связаны со входами данного элемента.

Здесь n_s – число элементов в исходной схеме. В результате такой замены элементы эквивалентной логической сети (модели исходной схемы) также оказываются правильно пронумерованными числами от $l+1$ до $l+n_e$, где n_e – число элементов в полученной сети, l – число входов в схеме. При этом номер элемента используется также для идентификации выхода (или выходной связи) элемента. Входы полученной логической сети, как и входы исходной схемы нумеруются числами от 1 до l .

Построение эквивалентной логической сети для элемента исходной схемы состоит в следующем. Для каждого входа элемента просматриваются все конъюнкции из системы ДНФ, задающей функции данного элемента, с целью определить, входит ли переменная, соответствующая данному входу, в некоторую конъюнкцию с отрицанием (для этого анализируются соответствующие данному входу компоненты элементов массива $m12$, описывающих конъюнкции рассматриваемого элемента схемы). В случае положительного ответа строится описание инвертора и помещается в массив okb . Формируется также соответствующий элемент массива $sokb$. Далее последовательно рассматриваются конъюнкции, входящие в систему ДНФ, задающую функции рассматриваемого элемента схемы, и каждая из них заменяется логической сетью из конъюнкторов. При этом первый конъюнктор реализует конъюнкцию первых двух литералов из исходной конъюнкции, второй – конъюнкцию над переменной, соответствующей выходу первого конъюнктора, и третьим литералом из исходной конъюнкции и т.д. Описания соответствующих конъюнкторов включаются в массив okb и формируются соответствующие элементы массива $sokb$.

Далее последовательно рассматриваются дизъюнкции системы ДНФ для исходного элемента и каждая из них аналогичным образом заменяется логической сетью из дизъюнкторов. В процессе формирования для каждой дизъюнкции из ДНФ выходного дизъюнктора эквивалентной логической сети формируются элементы массивов cn и pc , задающие соответствие между старыми и новыми номерами узлов в исходной схеме и в эквивалентной логической сети и наоборот.

Формируются некоторые элементы массива pc , задающего основные параметры эквивалентной логической сети и другие характеристики. Так, $pc[0]:=nt$, т.е. устанавливается первоначальное значение числа входных наборов в тесте (если $nt=0$, то тест не содержит наборов, в противном случае уже имеется некоторая последовательность наборов теста и требуется тест достроить); $pc[5]:=n$, т.е. задается число выходов в сети; $pc[6]:=n_e$; $c[14]:=1$.

Формируется массив выходов vi логической сети.

Далее строится описание логической сети в виде списка исходящих дуг (массив `oid`) и секционирующий для него массив `coid`.

Вызывается функция `kodne`, которая строит список неисправностей логической сети (массив `mne`), соответствующий списку неисправностей (массив `cf`) исходной схемы.

Вызывается функция `regme`, реализующая общую схему регулярного метода построения теста.

Выводится список необнаруженных неисправностей исходной схемы (массив `cf`, в котором единицами отмечены обнаруживаемые построенным тестом неисправности).

Описание функции `kodne`

Назначение

Строит список неисправностей логической сети, ориентированной на применение регулярного метода построения теста, соответствующий списку неисправностей исходной схемы.

Исходные данные

Массивы и переменные: `cf`, `l`, `nc`, `mn`, `nvix`, `svix`, `cn`.

Результаты

Список неисправностей логической сети, соответствующий списку неисправностей исходной схемы. Список помещается в массив `mne`.

Обращение

`kodne ()`.

Алгоритм

Просматриваются все линии исходной схемы в соответствии с возрастанием их номеров (номер линии определяется номером входного полюса схемы или выходного полюса логического элемента, из которого она исходит) и анализируется наличие на них неисправностей из списка `cf`. Если найдена некоторая неисправность, то такая же неисправность вносится на линию логической сети, соответствующую рассматриваемой линии исходной схемы.

Описание функции `regme`

Назначение

Реализует общую схему направленного метода построения теста для заданного списка неисправностей логической сети.

Исходные данные

Массивы и переменные: `m11`, `m12`, `m2`, `sm`, `ml`, `mn`, `mh`, `nvx`, `svx`, `nvix`, `svix`, `nvixc`, `cf`, `l`, `n`, `ns`, `nt`, `nhcf`, `okb`, `cokb`, `mne`, `vi`, `pc`, `oid`, `coid`, `nc`, `cn`, `hWnd`.
Дескриптор окна – `hWnd`, используется для передачи сообщений.

Результаты

1. Построенный тест, помещенный в массив `tst`. Переменная `nt` задает длину теста (число проверяющих наборов, включенных в тест).

2. Список неисправностей, которые не проверяются тестом (массивы `cf` и `mne`). Число `nhcfl` неисправностей, которые не проверяются тестом.

Обращение

`regme (hWnd)`.

Вызываемые функции

`dknne, dkni, dknili, clput, ucltr, opacf, anal, kodne`.

Алгоритм

1. Выбирается очередная неисправность из списка неисправностей, представленного массивом `mne`. Если список исчерпан, то выводится текст "Тест построен" и алгоритм заканчивает работу. Иначе выполняется анализ вида выбранной неисправности. Если выбрана неисправность на входном полюсе сети, то переход к п. 2 (в программе `me82`). Если выбрана неисправность инвертора, то переход к п. 3. В случае неисправности конъюнктора - переход к п. 4 (`me11`), дизъюнктора - к п. 5 (`me12`).

2. Формируется D-строка (массив `f`), соответствующая неисправности на входе сети, и осуществляется переход к п. 7.

3. Выбирается D-куб неисправности инвертора и формируется соответствующая ему D-строка, что достигается вызовом функции `dknne`. Переход к п. 6 (`me4`).

4. Выбирается D-куб неисправности конъюнктора и формируется соответствующая D-строка, что достигается вызовом функции `dkni`. Переход к п. 6 (`me4`).

5. Выбирается D-куб неисправности дизъюнктора и формируется соответствующая ему D-строка, что достигается вызовом функции `dknili`.

6. Выполняется проверка: является ли выход логического элемента выходом схемы? В случае положительного ответа осуществляется переход к п. 9 (`me10`).

7. Строится сложный путь от места неисправности до выходных полюсов сети (вызов функции `clput`).

8. Выполняется попытка найти условия транспортировки неисправности до выходных полюсов сети (вызов функции `ucltr`). Если таких условий не существует, то переход к одному из п.п. 3 – 5 с целью выбора другого D-куба неисправности или перехода к очередной непроверенной неисправности из заданного списка (п. 1).

9. Выполняется попытка обеспечить найденные условия транспортировки неисправности до выходных полюсов сети (вызов функции `opacf`). Если это

сделать не удастся, то переход к п. 8 с целью поиска другого варианта условий транспортировки неисправности до выходных полюсов сети. Иначе (условия обеспечены) найден проверяющий тест для заданной неисправности, и он выводится в файл regtest. Далее выполняется моделирование оставшихся в списке cf неисправностей на найденном входном наборе с целью удаления из списка неисправностей тех из них, которые проверяются набором. Если значения на некоторых входах в найденном проверяющем наборе не определены, то неопределенное значение предварительно заменяется нулем. Далее осуществляется переход к п. 1.

Описание функции dkni

Назначение

Строит D-кубы неисправностей логического элемента, реализующего конъюнкцию.

Исходные данные

Массивы okb, sokb, pc. Переменная im, задающая номер логического элемента; переменная c, задающая неисправность элемента; переменная d, задающая номер D-куба неисправности.

Результаты

1. Массив f, содержащий D-куб неисправности.
2. Массив bd со списком логических элементов, на выходах которых необходимо обеспечить определенные значения сигналов.
3. Переменная w, указывающая на первый свободный элемент массива bd.

Обращение

dkni ().

Алгоритм

Переменная c указывает, для какой неисправности необходимо построить D-куб неисправности. При этом $c = 1$ соответствует неисправности "константа 0" на выходе элемента, $c = 2$ – неисправности "константа 1" на выходе, $c = 3$ – неисправности "константа 0" на первом входе, $c = 4$ – неисправности "константа 1" на первом входе и т.д. Если $d = 1$, то выбирается первый из двух возможных D-кубов неисправностей (для тех неисправностей, для которых существуют два D-куба неисправности), если $d = 2$, то – второй D-куб неисправности.

Алгоритм программы тривиален и состоит в занесении требуемого D-куба заданной неисправности в массив f. При этом в массив bd включаются

логические элементы, на выходах которых необходимо обеспечить определенные значения сигналов, появившиеся в результате выбора D-куба неисправности.

Описание функции dknili

Назначение

Строит D-кубы неисправностей логического элемента, реализующего дизъюнкцию.

Исходные данные

Массивы и переменные okb, sokb, pc, im, c, d (см. описание функции dknj).

Результаты

Массивы и переменные f, bd, w (см. описание функции dknj).

Обращение

dknili ().

Алгоритм

Аналогичен алгоритму функции dknj.

Описание функции dknne

Назначение

Строит D-кубы неисправностей логического элемента, реализующего отрицание.

Исходные данные

Массивы и переменные okb, sokb, pc, im, c (см. описание функции dknj).

Результаты

Массивы и переменные f, bd, w (см. описание функции dknj).

Обращение

dknne ().

Алгоритм

См. алгоритм функции dknj. В отличие от данной функции не используется переменная d, так как каждой неисправности соответствует только один D-куб неисправности.

Описание функции clput

Назначение

Построение сложного пути, т.е. совокупности простых путей, от места неисправности до выходных полюсов сети.

Исходные данные

Массивы и переменные oid, cojd, im, pc. Здесь im – переменная, которая задает номер линии сети, от которой необходимо построить путь.

Результаты

Сложный путь, помещенный в массив `nlin`.

Обращение

`clput ()`.

Алгоритм

Последовательно просматриваются номера узлов (номер узла это номер входного полюса сети или выхода логического элемента), включенных в путь (первоначально в путь включается узел, заданный переменной `im`) и для каждого из них находятся все преемники. Те из преемников, которые еще не включались в путь, т.е. их еще нет в массиве `nlin`, включаются в него. Процесс продолжается до тех пор, пока все узлы, включенные в путь, не рассмотрены.

В заключение номера узлов, включенных в путь, упорядочиваются в порядке возрастания их номеров.

Описание функции `ucltr`

Назначение

Вычисляет условия образования существенного пути от места неисправности до выходных полюсов логической сети.

Исходные данные

Массивы и переменные `okb`, `sokb`, `vi`, `f`, `ma`, `nlin`, `oid`, `coid`, `pc`, а также переменная `w`, указывающая на первый свободный элемент массива `bd` список элементов для операции расширения фиксации).

Результаты

1. Признак `s`, указывающий на результат вычисления условий образования существенного пути. `s=1`, если условия найдены, и `s=0` – в противном случае.
2. Массив `fa`, содержащий D-строку.
3. Массив `ma`, содержащий условное представление найденного варианта условий образования существенного пути.
4. Массив `ba` со списком элементов сложного пути, которые использовались при вычислении условий образования существенного пути.
5. Массив `bd` со списком логических элементов, на выходах которых необходимо обеспечить определенные значения сигналов.
6. Переменная `v`, указывающая на первый свободный элемент массива `bd`.
7. Переменная `f1`, указывающая на первый свободный элемент массива `ba`.

Обращение

`ucltr ()`.

Вызываемые функции

`uti`, `utili`, `utne`.

Алгоритм

Последовательно рассматриваются логические элементы, принадлежащие сложному пути от места неисправности до выходных полюсов сети (массив $nlin$). Для очередного выбранного элемента делается попытка найти условия транспортировки неисправности через него (в зависимости от типа логического элемента для этого строится обращение к одной из функций uti , $utili$, $utne$). Если после рассмотрения очередного логического элемента найдены условия транспортировки неисправности до выходного полюса сети, то выполнение алгоритма завершается. Если такие условия не найдены, но вектор активности (q) не равен нулю (вектор активности – число логических элементов сложного пути, которые еще не рассматривались и на входах которых проявляется заданная неисправность), то – переход к рассмотрению очередного элемента сложного пути. Если вектор активности равен нулю, то осуществляется попытка возврата к последнему логическому элементу из уже рассмотренных элементов сложного пути с целью вычисления других условий транспортировки неисправности через него. Если такого элемента нет, то $s = 0$ и выполнение алгоритма завершается.

Описание функции uti

Назначение

Вычисляет условия образования существенного пути через логический элемент, реализующий конъюнкцию (т.е. D-куб или тупиковый D-куб).

Исходные данные

Массивы и переменные: vi , okb , $sokb$, fa , ma , oid , $coid$, $nlin$, pc .

Результаты

Массивы и переменные, модифицированные в соответствии с выбранным D-кубом: fa , ma , ba , bd , fl , il , $j1$, s , q (вектор активности), v .

Обращение

$uti(i1, j1, q)$.

Алгоритм

Выполняется анализ значений сигналов на входах логического элемента и в зависимости от требуемого варианта условий транспортировки неисправности через логический элемент, который определяется значением соответствующего данному логическому элементу элемента массива ma , выбирается D-куб или тупиковый D-куб, который не противоречит D-строке (D-строка хранится в массиве fa). Модифицируется элемент массива ma , соответствующий рассматриваемому логическому элементу: 1) $ma[i1] = 0$, если D-куба для рассматриваемого логического элемента не существует, 2) $ma[i1] = 2$, если существует и выбран единственный возможный вариант D-куба для логического

элемента или выбран второй вариант D-куба (если возможны два варианта и первый был использован ранее), 3) $ma[i1] = 1$, если выбран первый из нескольких возможных вариантов D-кубов для рассматриваемого логического элемента. Выбранный D-куб или тупиковый D-куб заносится в массив fa , соответствующим образом корректируются массивы ba , bd , переменные $f1$, $i1$, $j1$, q (вектор активности), s (если символ D или D попадает на выходной полюс логической сети), v .

Описание функции utili

Назначение

Вычисляет условия образования существенного пути через логический элемент, реализующий дизъюнкцию (т.е. D-куб или тупиковый D-куб).

Исходные данные

Массивы и переменные, модифицированные в соответствии с выбранным D-кубом: fa , ma , ba , bd , $f1$, $i1$, $j1$, s , q (вектор активности), v .

Обращение

$utili(i1, j1, q)$.

Алгоритм

Выполняется анализ значений сигналов на входах логического элемента и в зависимости от требуемого варианта условий транспортировки неисправности через логический элемент, который определяется значением соответствующего данному логическому элементу массива ma , выбирается D-куб или тупиковый D-куб, который не противоречит D-строке (D-строка хранится в массиве fa). Модифицируется элемент массива ma , соответствующий рассматриваемому логическому элементу: 1) $ma[i1]=0$, если D-куба для рассматриваемого логического элемента не существует, 2) $ma[i1]=2$, если существует и выбран единственный возможный вариант D-куба для логического элемента или выбран второй вариант D-куба (если возможны два варианта и первый был использован ранее), 3) $ma[i1]=1$, если выбран первый из нескольких возможных вариантов D-кубов для рассматриваемого логического элемента. Выбранный D-куб или тупиковый D-куб заносится в массив fa , соответствующим образом корректируются массивы ba , bd , переменные $f1$, $i1$, $j1$, q (вектор активности), s (если символ D или D попадает на выходной полюс логической сети), v .

Описание функции utne

Назначение

Вычисляет условия образования существенного пути через инвертор (т.е. D-куб).

Исходные данные

Массивы и переменные: v_i , okb , $sokb$, fa , oid , $coid$, $nlin$, pc .

Результаты

Массивы и переменные, модифицированные в соответствии с выбранным Дкубом: fa , ma , ba , fl , $i1$, $j1$, s , q (вектор активности).

Обращение

$utne(i1, j1, q)$.

Алгоритм

Выполняется анализ значения сигнала на входе логического элемента и выходу элемента приписывается соответствующее значение сигнала. Модифицируется элемент массива ma , соответствующий рассматриваемому логическому элементу: 1) $ma[i1]=0$, если входу элемента приписано значение сигнала x , 2) $ma[i1]=2$ – в противном случае. Соответственно корректируются массивы и переменные fa , ba , fl , $i1$, $j1$, q , s (если символ D или D попадает на выходной полюс логической сети).

Описание функции **orasf**

Назначение

Вычисляет входной набор (возможно не полностью определенный) логической сети, обеспечивающий найденные функцией $ucltr$ условия образования существенного пути, т.е. реализует операцию расширения фиксации для логической сети в целом.

Исходные данные

Массивы и переменные okb , $sokb$, fa , bd , pc , v (см. функцию $ucltr$).

Результаты

Признак s , указывающий на результат обеспечения условий: $s=1$, если условия удалось обеспечить, и $s=0$ – в противном случае. Массив fd , содержащий обеспеченную D -строку.

Обращение

$orasf()$.

Вызываемые функции

$orfi$, $orfil$, $orfne$.

Алгоритм

Последовательно в порядке убывания номеров рассматриваются логические элементы, на выходах которых необходимо обеспечить определенные значения сигналов. Список таких элементов задается массивами bd , bdr и $bdr1$. Данный список в процессе рассмотрения элементов может пополняться. Для очередного выбранного элемента делается попытка обеспечить заданное значение сигнала на его выходе (в зависимости от типа логического элемента для

этого строится обращение к одной из функций orf_i , orf_{il} , orf_{ne}). Если для очередного логического элемента обеспечено заданное значение сигнала на его выходе, то осуществляется переход к рассмотрению очередного элемента из списка. Если же требуемое значение сигнала обеспечить не удастся, то осуществляется возврат к последнему рассмотренному элементу из списка, для которых существует другой вариант обеспечения заданного значения сигнала на его выходе. Если такого элемента не существует, то $s=0$ и выполнение алгоритма завершается. В том случае, когда список исчерпан, то $s=1$ и выполнение алгоритма также завершается.

Описание функции orf_i

Назначение

Вычисляет вариант значений сигналов на входах логического элемента, реализующего конъюнкцию, обеспечивающих заданное значение сигнала на выходе элемента.

Исходные данные

Массивы и переменные: fd , bdr , okb , $sokb$, pc , а также массив md , используемый для запоминания выбранных вариантов обеспечения заданных значений сигналов на выходах логических элементов; переменная $i1$, задающая номер рассматриваемого логического элемента; переменная $j1$, указывающая на элемент массива md , соответствующий рассматриваемому логическому элементу.

Результаты

1. Массив $bdr1$, содержащий пополненный список логических элементов, на выходах которых необходимо обеспечить заданные значения сигналов.
2. Массив fd , содержащий обеспеченную D-строку, т.е. тест для заданной неисправности.
3. Массив md , модифицированный в соответствии с выбранным вариантом обеспечения заданного значения сигнала на выходе логического элемента.
4. Переменная pr , которая принимает значение 0, если заданное значение сигнала на выходе логического элемента обеспечено; $pr=2$, если не существует входного набора, обеспечивающего заданные значения сигналов на выходах логических элементов, заданных списком (массивом) bdr ; $pr=1$, если необходимо и возможно вернуться к уже рассмотренным логическим элементам из заданного списка с целью выбора другого варианта обеспечения заданных значений сигналов на их выходах.

Обращение

$orf_i(i1, j1, pr)$.

Алгоритм

Выполняется анализ значений сигналов на выходе и входах заданного логического элемента и в зависимости от требуемого варианта обеспечения заданного значения сигнала на выходе логического элемента (определяемого элементом массива $md[j1]$) такой вариант выбирается. Модифицируется элемент массива md , соответствующий рассматриваемому логическому элементу: 1) $md[j1]=2$, если существует и выбран единственный возможный вариант обеспечения заданного значения сигнала на выходе логического элемента или выбран второй возможный вариант обеспечения условий (если возможны два варианта и первый был использован ранее), 2) $md[j1]=1$, если выбран первый из нескольких возможных вариантов обеспечения заданного значения сигнала на выходе логического элемента.

Выбранный вариант заносится в массив fd , соответствующим образом корректируется массив $bdr1$, $pr=0$. Если же не существует варианта

обеспечения заданного значения сигнала на выходе логического элемента и для уже рассмотренных логических элементов из списка bdr невозможно выбрать другие варианты обеспечения заданных значений сигналов на их выходах, то $pr=2$ и выполнение алгоритма завершается. Иначе $pr=1$ и также выполнение алгоритма завершается.

Описание функции **orfil**

Назначение

Вычисляет вариант значений сигналов на входах логического элемента, реализующего дизъюнкцию, обеспечивающих заданное значение сигнала на выходе элемента.

Исходные данные

Массивы и переменные: fd , bdr , okb , $sokb$, pc , md , $i1$, $j1$ (см. описание функции **orfi**).

Результаты

Модифицированные массивы и переменные: $bdr1$, fd , md , pr (см. описание функции **orfi**).

Обращение

orfil ($i1$, $j1$, pr).

Алгоритм

Аналогичен алгоритму функции **orfi**.

Описание функции `orfne`

Назначение

Вычисляет значение сигнала на входе логического элемента, реализующего отрицание, обеспечивающего заданное значение сигнала на выходе логического элемента.

Исходные данные

Массивы и переменные: `fd`, `bdr`, `okb`, `sokb`, `pc`, `md`, `i1`, `j1` (см. описание функции `orf1`).

Результаты

Модифицированные массивы и переменные: `bdr1`, `fd`, `md`, `pr` (см. описание функции `orf1`).

Алгоритм

Аналогичен алгоритму функции `orf1`.

5.4 Инструкция пользователя

После запуска приложения на экране появится его главное окно (рис. 15)

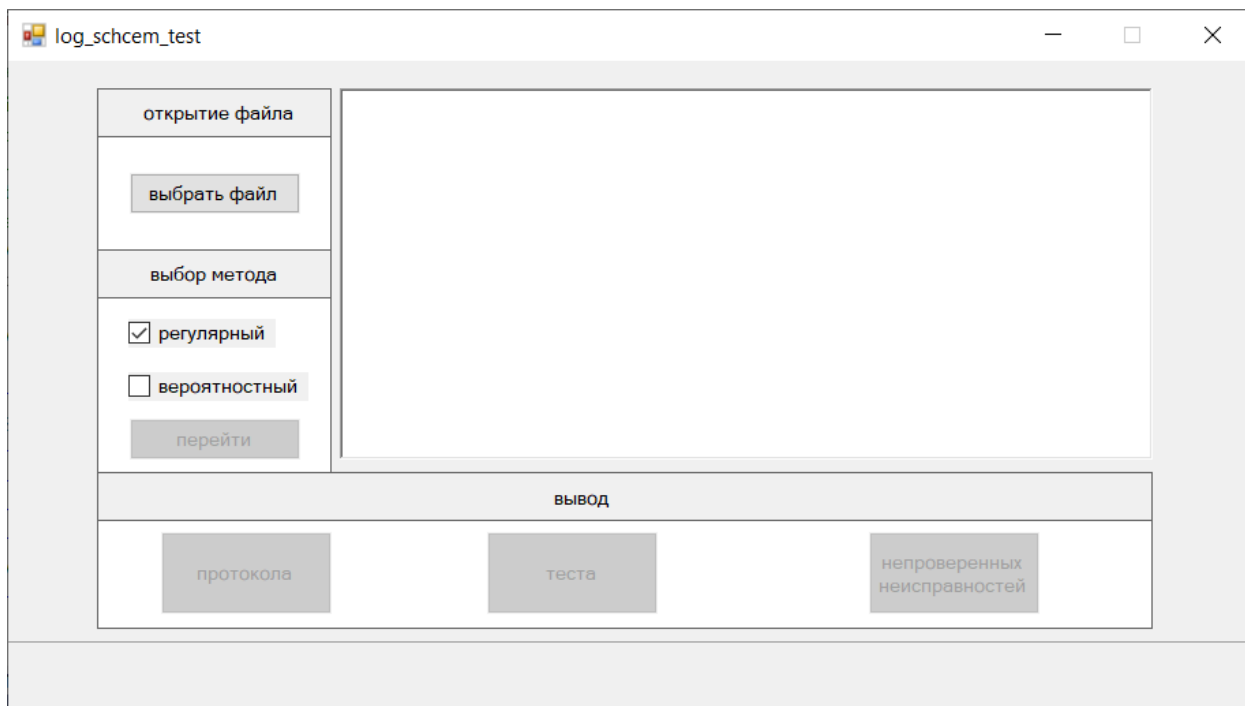


Рис. 15. Главное окно приложения

Мы видим, что кнопки «перейти» и различные выводы результатов недоступны, чтобы не запутать пользователя и избежать лишних ошибок. Само окно построено логично: пользователь интуитивно будет следовать по кнопкам сверху вниз.

При нажатии «выбрать файл» открывается окно выбора файла определённого формата «.TSC», в котором содержится описание схемы (сгенерировано в системе MODELSIM. Представление схемы, используемой в этой главе, находится в Приложении Б). По умолчанию открывается корневая папка приложения с переходом в папку с примерами «Ex» (см. инструкцию по установке). После выбора и загрузки мы видим сообщение об удачной загрузке файла (либо об ошибке), и кнопки главного окна становятся доступны. Также, для удобства, внизу окна выводится информация о загруженном в данный момент файле (рис. 16).

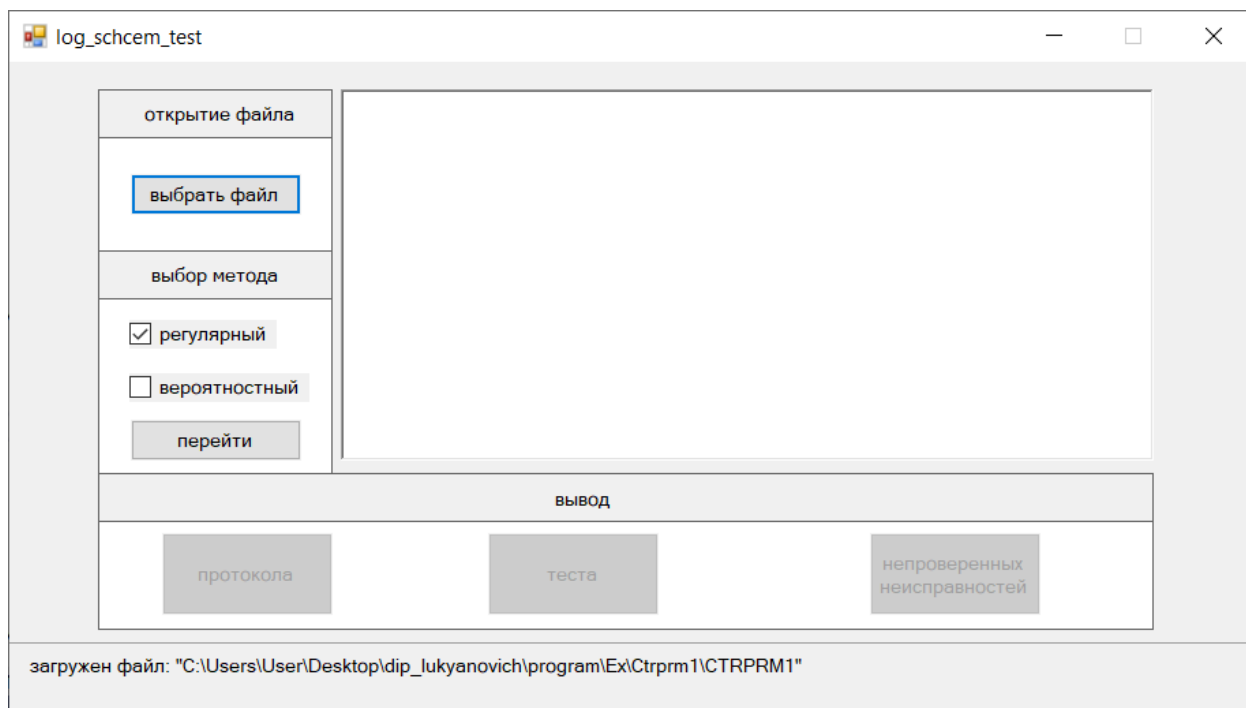


Рис. 16. Главное окно после загрузки файла

Далее следует выбор метода построения теста (система не даст поставить сразу 2 галочки) и для перехода к построению необходимо нажать кнопку «перейти». Мы увидим данное окно (рис.17):

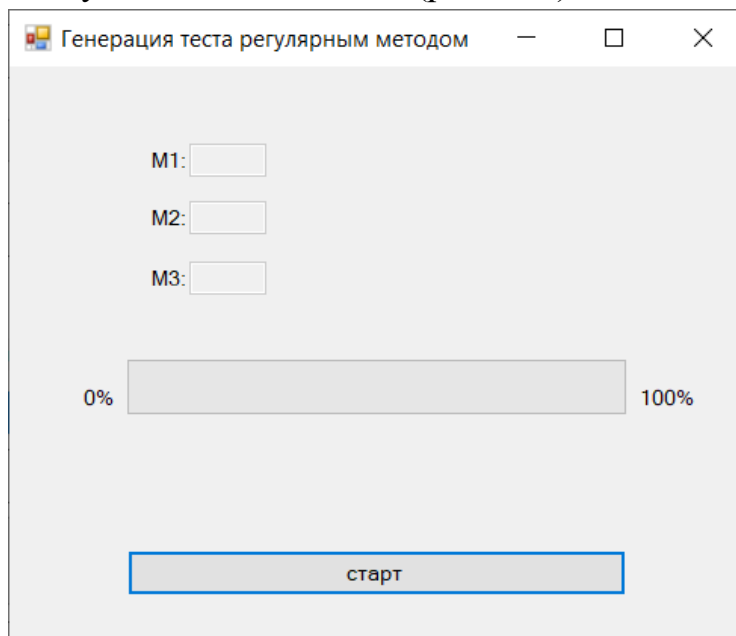


Рис. 17. Окно построения тестов

Это общее окно для обоих методов построения тестов, реализованных в системе. Но так как был выбран «регулярный метод», поля «M1», «M2» «M3» недоступны для заполнения (при выборе вероятностного метода и неверном вводе параметров будет выводиться ошибка). Для начала построения

необходимо нажать кнопку «старт» и дождаться выполнения, далее будет выведена информация (рис. 18):

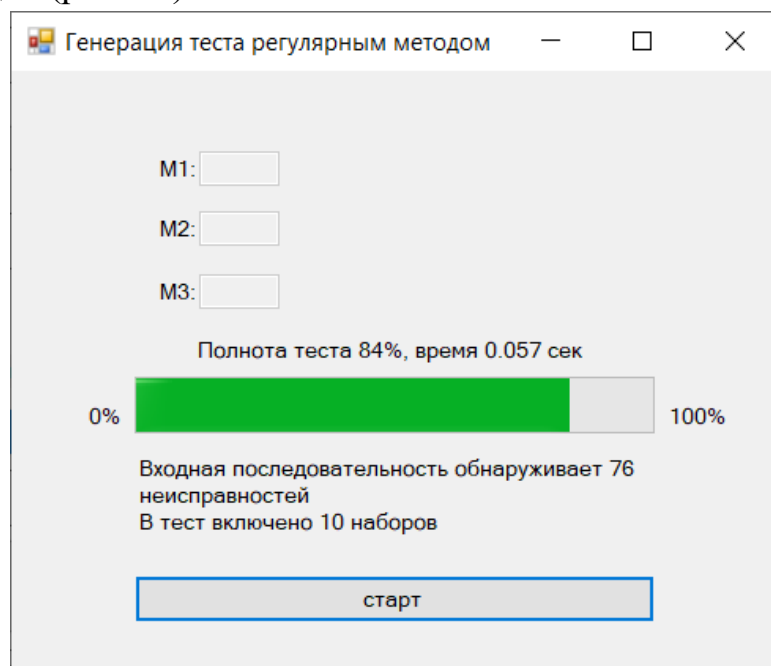


Рис. 18. Окно построения тестов после генерации

Здесь выводится информация о полноте, затраченном времени в секундах, общем числе обнаруживаемых неисправностей и количестве наборов тестирования.

Далее необходимо закрыть модальное окно и возвратиться к главному окну программы. Для вывода подробной информации о построении теста, такой как протокол, сам тест либо списка непроверенных неисправностей, необходимо нажать одну из кнопок в поле выбора «протокол», «теста» «непроверенных неисправностей» соответственно. Вывод информации осуществляется в текстовом окне (для удобства переноса данной информации в иную программу)

Например, при нажатии кнопки «теста» окно будет выглядеть следующим образом (рис. 19):

log_schcem_test

открытие файла

выбрать файл

выбор метода

☒ регулярный

☐ вероятностный

перейти

ПРОВЕРЯЮЩИЙ ТЕСТ

ВХОДЫ	ВЫХОДЫ
1) 11100000000	0101
2) 01101111111	1100
3) 10101111111	1100
4) 11001111111	1100
5) 11111000000	1011
6) 11101001111	0110
7) 11110001111	0110
8) 11111100000	0111
9) 11111010000	0111
10) 00011100000	1111

вывод

протокола

теста

непроверенных
неисправностей

загружен файл: "C:\Users\User\Desktop\dip_lukyanovich\program\Ex\Ctrprm1\CTRPRM1"

Рис. 19. Вывод подробностей о построенных тестах

ЗАКЛЮЧЕНИЕ

В логических схемах того или иного рода бывают неисправности. Работа посвящена своевременному диагностированию неисправностей на моменте логического проектирования. В связи с большой сложностью устройств, высокой трудоёмкостью построения тестов на этапе проектирования важно иметь средства автоматизированного построения тестов для неисправностей.

В ходе работы были изучены математические модели и основные методы решения задач тестового диагностирования. Исследованы алгоритмы построения тестов для КС, реализованные в программном комплексе для ПЭВМ, который позволяет строить тесты с помощью вероятностного и методов.

В настоящее время, в связи с развитием вычислительной техники, стоит задача использования комплексов для автоматизированной диагностики на любых платформах, имеющихся у пользователей. Именно поэтому система программ, выполняющих функции построения тестов, была выполнена с использованием технологий .NET и CLR. Использование .NET предполагает межплатформенность приложения и возможность портирования программы на любую систему после установки .NET Framework.

Программный комплекс разработан как приложение Windows Forms в среде Microsoft Visual Studio 2019. Использование визуального программирования предполагает наличие дружественного интерфейса с понятными пользователю обозначениями и принципами работы.

Для тестирования была разработана система окон, позволяющая пользователю наблюдать результаты работы теста. В ходе построения теста на экран выводится информация о времени проведения теста (затратах времени), длине, полноте выполненного тестирования. Вся информация о результатах тестирования выводится (при нажатии соответствующей кнопки) в главном окне, позволяя пользователю видеть непроверенные неисправности, общее число неисправностей, полноту теста в процентах.

В процессе создания программного комплекса методы построения тестов были реализованы средствами Visual C++ CLR. Считаю использование платформы .NET Framework и CLR приложений в дальнейшем целесообразным и обоснованным. Программы, работающие независимо от платформы, несомненно необходимы в условиях быстрого развития ПЭВМ.

В связи с ростом производительности компьютерной техники, потеря скорости до 15% в программах, использующих CLR, за счет промежуточного языка CIL, оправдана возможностью написания дополнений и исправлений на любом из .NET совместимых языков. Исключениями являются случаи разработки 3D приложений, где потеря производительности недопустима.

Несмотря на все преимущества платформы .NET, для углубленной работы с памятью, потоками и прочими компонентами низкого уровня, рекомендуется все же не прибегать к средствам .NET именно из-за ограничений на работу с памятью.

Таким образом, в ходе выполнения дипломной работы «Автоматизированное построение тестов для комбинационных схем» была проанализирована проблема тестового диагностирования, исследованы методы построения тестов; разработан графический интерфейс пользователя для программного комплекса автоматизированного построения тестов; создана система окон для выполнения тестирования комбинационных схем; сделаны выводы об актуальности использования средств .NET.

Считаю, что задача дипломной работы изучить способы диагностирования неисправностей и разработать программный комплекс автоматизированного построения тестов для КС с использованием технологий .NET и CLR решена полностью.

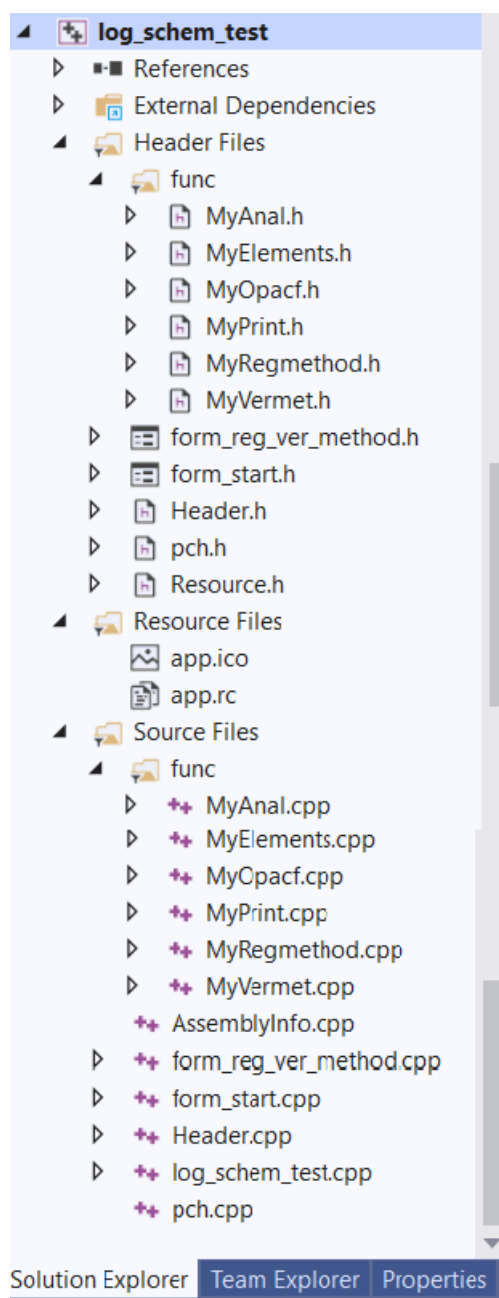
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Бадулин С.С., Барнаулов Ю.М., Бердышев А.А. и др./ Автоматизированное проектирование цифровых устройств.- М: Радио и связь, 1981
2. Коротаев Н. А., Люлькин А. Е. Автоматизация тестового диагностирования дискретных устройств. Минск: БГУ, 1983.
3. Люлькин А.Е. Математические модели элементов большой интеграции в системах автоматизированного проектирования тестов.— Микроэлектроника.—1991.—Том 20.— № 2.— С.208-215.
4. Люлькин А.Е. Построение проверяющих наборов для неисправностей в логических сетях из функционально сложных элементов.— Автоматика и вычислительная техника.—1986.—.№1.— С.91
5. Уткин А.А. Анализ логических сетей и техника булевых вычислений. — Мн.: Наука и техника, 1979.
6. Яблонский С.В. Введение в дискретную математику. — М: Наука, 2010
7. Ярмолик В.Н. Контроль и диагностика цифровых узлов ЭВМ. — Минск: Наука и техника, 1988
8. Mourad S., Zorian Y. Principles of Testing Electronic Systems. — New York, 2000. — 420 p.

ПРИЛОЖЕНИЕ А

Листинг программы

Древо проекта выглядит следующим образом:



MyAnal.h

```
//#pragma once

#if !defined (_ANAL_HEADER)
#define _ANAL_HEADER

void anal(HWND hWnd);
void intst();
void comne();
void iskl(int pr);
void simul(int pr);

#endif
```

MyElements.h

```
//#pragma once

#if !defined (_ELEMENTS_HEADER)
#define _ELEMENTS_HEADER

void ucltr();
void uti(int* i1, int* j1, int* q);
void utili(int* i1, int* j1, int* q);
void utne(int* i1, int* j1, int* q);

#endif
```

MyOpacf.h

```
//#pragma once

#if !defined (_OPACF_HEADER)
#define _OPACF_HEADER

void opacf();
void opfi(int* i1, int* j1, int* pr);
void opfil(int* i1, int* j1, int* pr);
void opfne(int* i1, int* j1, int* pr);

#endif
```

MyPrint.h

```
//#pragma once

#if !defined (_PRINT_HEADER)
#define _PRINT_HEADER

void printtst();
void printcf();

#endif
```

MyRegmethod.h

```
//#pragma once

#include <Windows.h>

#if !defined (_REGMETHOD_HEADER)
#define _REGMETHOD_HEADER

int modcx (HWND hWnd);
int regme (HWND hWnd);
void kodne ();
void clput ();
void dknne ();
void dknili ();
void dkni ();
#endif
```

MyVermet.h

```
//#pragma once

#if !defined (_VERMET_HEADER)
#define _VERMET_HEADER

UINT vermet (LPVOID pParam);
void PatGener (int* ch, int lp);

#endif
```

form_reg_ver_method.h

```
#pragma once

namespace logschemtest {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for form_reg_ver_method
    /// </summary>
    public ref class form_reg_ver_method : public System::Win-
dows::Forms::Form
    {
    public:
        form_reg_ver_method(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }
    };
}
```

```

    }

    form_reg_ver_method(bool checked)
    {
        isVerMethod = checked;
        if (!checked)
            name = "❖"enerationa теста регулярным методом";
        else
            name = "❖"enerationa теста вероятностным мето-
дом";

        InitializeComponent();
    }

    bool isVerMethod;
    System::String^ name;

protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~form_reg_ver_method()
    {
        if (components)
        {
            delete components;
        }
    }

private: System::Windows::Forms::ProgressBar^ progressBar1;
protected:
private: System::Windows::Forms::TextBox^ textBoxM1;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::TextBox^ textBoxM2;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::TextBox^ textBoxM3;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Button^ buttonStart;

private: System::Windows::Forms::Label^ labelResult;
private: System::Windows::Forms::Label^ label4;
private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Label^ labelPolnota;

private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->progressBar1 = (gcnew System::Win-
dows::Forms::ProgressBar());
        this->textBoxM1 = (gcnew System::Windows::Forms::Text-
Box());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->textBoxM2 = (gcnew System::Windows::Forms::Text-
Box());

```

```

        this->label2 = (gcnew System::Windows::Forms::Label());
        this->textBoxM3 = (gcnew System::Windows::Forms::Text-
Box());

        this->label3 = (gcnew System::Windows::Forms::Label());
        this->buttonStart = (gcnew System::Windows::Forms::But-
ton());

        this->labelResult = (gcnew System::Windows::Forms::La-
bel());

        this->label4 = (gcnew System::Windows::Forms::Label());
        this->label5 = (gcnew System::Windows::Forms::Label());
        this->labelPolnota = (gcnew System::Windows::Forms::La-
bel());

        this->SuspendLayout();
        //
        // progressBar1
        //
        this->progressBar1->Location = System::Draw-
ing::Point(78, 194);

        this->progressBar1->Name = L"progressBar1";
        this->progressBar1->Size = System::Drawing::Size(330,
36);

        this->progressBar1->TabIndex = 0;
        //
        // textBoxM1
        //
        this->textBoxM1->Enabled = isVerMethod;//true;
        this->textBoxM1->Location = System::Drawing::Point(119,
51);

        this->textBoxM1->Name = L"textBoxM1";
        this->textBoxM1->Size = System::Drawing::Size(51, 22);
        this->textBoxM1->TabIndex = 1;
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Location = System::Drawing::Point(91,
53);

        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(31, 17);
        this->label1->TabIndex = 4;
        this->label1->Text = L"M1:";
        //
        // textBoxM2
        //
        this->textBoxM2->Enabled = isVerMethod;//true;
        this->textBoxM2->Location = System::Drawing::Point(119,
89);

        this->textBoxM2->Name = L"textBoxM2";
        this->textBoxM2->Size = System::Drawing::Size(51, 22);
        this->textBoxM2->TabIndex = 5;
        //
        // label2
        //
        this->label2->AutoSize = true;
        this->label2->Location = System::Drawing::Point(91,
91);

        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(31, 17);
        this->label2->TabIndex = 6;
        this->label2->Text = L"M2:";
        //
        // textBoxM3
        //

```

```

        this->textBoxM3->Enabled = isVerMethod;//true;
        this->textBoxM3->Location = System::Drawing::Point(119,
129);

        this->textBoxM3->Name = L"textBoxM3";
        this->textBoxM3->Size = System::Drawing::Size(51, 22);
        this->textBoxM3->TabIndex = 7;
        //
        // label3
        //
        this->label3->AutoSize = true;
        this->label3->Location = System::Drawing::Point(91,
131);

        this->label3->Name = L"label3";
        this->label3->Size = System::Drawing::Size(31, 17);
        this->label3->TabIndex = 8;
        this->label3->Text = L"M3.";
        //
        // buttonStart
        //
        this->buttonStart->Location = System::Draw-
ing::Point(78, 320);

        this->buttonStart->Name = L"buttonStart";
        this->buttonStart->Size = System::Drawing::Size(330,
30);

        this->buttonStart->TabIndex = 9;
        this->buttonStart->Text = L"crapt";
        this->buttonStart->UseVisualStyleBackColor = true;
        this->buttonStart->Click += gcnew Sys-
tem::EventHandler(this, &form_reg_ver_method::buttonStart_Click);
        //
        // labelResult
        //
        this->labelResult->Location = System::Draw-
ing::Point(78, 243);

        this->labelResult->Name = L"labelResult";
        this->labelResult->Size = System::Drawing::Size(330,
75);

        this->labelResult->TabIndex = 11;
        //
        // label4
        //
        this->label4->AutoSize = true;
        this->label4->Location = System::Drawing::Point(46,
210);

        this->label4->Name = L"label4";
        this->label4->Size = System::Drawing::Size(28, 17);
        this->label4->TabIndex = 12;
        this->label4->Text = L"0%";
        //
        // label5
        //
        this->label5->AutoSize = true;
        this->label5->Location = System::Drawing::Point(414,
210);

        this->label5->Name = L"label5";
        this->label5->Size = System::Drawing::Size(44, 17);
        this->label5->TabIndex = 13;
        this->label5->Text = L"100%";
        //
        // labelPolnota
        //
        this->labelPolnota->AutoSize = true;

```

```

        this->labelPolnota->Location = System::Draw-
ing::Point(115, 168);
        this->labelPolnota->Name = L"labelPolnota";
        this->labelPolnota->Size = System::Drawing::Size(0,
17);

        this->labelPolnota->TabIndex = 14;
        //
        // form_reg_ver_method
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(8,
16);

        this->AutoScaleMode = System::Windows::Forms::Au-
toScaleMode::Font;

        this->ClientSize = System::Drawing::Size(489, 381);
        this->Controls->Add(this->labelPolnota);
        this->Controls->Add(this->label5);
        this->Controls->Add(this->label4);
        this->Controls->Add(this->labelResult);
        this->Controls->Add(this->buttonStart);
        this->Controls->Add(this->textBoxM3);
        this->Controls->Add(this->label3);
        this->Controls->Add(this->textBoxM2);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->textBoxM1);
        this->Controls->Add(this->progressBar1);
        this->Controls->Add(this->label1);
        this->Name = L"form_reg_ver_method";
        this->StartPosition = System::Win-
dows::Forms::FormStartPosition::CenterParent;
        this->Text = name; // L"TeCT";
        this->FormClosed += gcnew System::Windows::Forms::Form-
ClosedEventHandler(this, &form_reg_ver_method::form_reg_ver_method_Form-
Closed);

        this->ResumeLayout(false);
        this->PerformLayout();

    }

#pragma endregion
    private: System::Void buttonStart_Click(System::Object^ sender, Sys-
tem::EventArgs^ e);
    private: System::Void form_reg_ver_method_FormClosed(System::Object^
sender, System::Windows::Forms::FormClosedEventArgs^ e);
    private: System::Void regMethod();
    private: System::Void verMethod();
};

}

struct MPStruct {
    CUIntArray M;
    HWND hw;
    LPCSTR st;
};
};

```

form_start.h

```

#pragma once

namespace logschemtest {

```

```

//using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

/// <summary>
/// Summary for form_start
/// </summary>
public ref class form_start : public System::Windows::Forms::Form
{
public:
    form_start(void)
    {
        InitializeComponent();
        //
        //TODO: Add the constructor code here
        //
    }

protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~form_start()
    {
        if (components)
        {
            delete components;
        }
    }

protected:

protected:

private: System::Windows::Forms::RichTextBox^ richTextBox1;
private: System::Windows::Forms::Button^ buttonPrintTest;

private: System::Windows::Forms::Button^ buttonOpen;
private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Button^ buttonStartMethodForm;

private: System::Windows::Forms::Label^ label4;
private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Label^ label6;
private: System::Windows::Forms::Label^ label7;
private: System::Windows::Forms::Label^ label8;
private: System::Windows::Forms::Label^ label9;
private: System::Windows::Forms::Button^ buttonPrintProtocol;
private: System::Windows::Forms::Button^ buttonPrintUntesting;

```

```

private: System::Windows::Forms::CheckBox^ checkBoxRegMethod;
private: System::Windows::Forms::CheckBox^ checkBoxVerMethod;
private: System::Windows::Forms::Label^ label10;
private: System::Windows::Forms::Label^ labelFileName;

protected:

private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->richTextBox1 = (gcnew System::Win-
dows::Forms::RichTextBox());
        this->buttonPrintTest = (gcnew System::Win-
dows::Forms::Button());
        this->buttonOpen = (gcnew System::Windows::Forms::But-
ton());
        this->openFileDialog1 = (gcnew System::Win-
dows::Forms::OpenFileDialog());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->label2 = (gcnew System::Windows::Forms::Label());
        this->label3 = (gcnew System::Windows::Forms::Label());
        this->buttonStartMethodForm = (gcnew System::Win-
dows::Forms::Button());
        this->label4 = (gcnew System::Windows::Forms::Label());
        this->label5 = (gcnew System::Windows::Forms::Label());
        this->label6 = (gcnew System::Windows::Forms::Label());
        this->label7 = (gcnew System::Windows::Forms::Label());
        this->label8 = (gcnew System::Windows::Forms::Label());
        this->label9 = (gcnew System::Windows::Forms::Label());
        this->buttonPrintProtocol = (gcnew System::Win-
dows::Forms::Button());
        this->buttonPrintUntesting = (gcnew System::Win-
dows::Forms::Button());
        this->checkBoxRegMethod = (gcnew System::Win-
dows::Forms::CheckBox());
        this->checkBoxVerMethod = (gcnew System::Win-
dows::Forms::CheckBox());
        this->label10 = (gcnew System::Windows::Forms::La-
bel());
        this->labelFileName = (gcnew System::Win-
dows::Forms::Label());
        this->SuspendLayout();
        //
        // richTextBox1
        //
        this->richTextBox1->Location = System::Draw-
ing::Point(243, 20);
        this->richTextBox1->Name = L"richTextBox1";
        this->richTextBox1->Size = System::Drawing::Size(596,
272);
        this->richTextBox1->TabIndex = 2;

```



```

        this->richTextBox1->Text = L"";
        //
        // buttonPrintTest
        //
        this->buttonPrintTest->Enabled = false;
        this->buttonPrintTest->Location = System::Draw-
ing::Point(351, 345);
        this->buttonPrintTest->Name = L"buttonPrintTest";
        this->buttonPrintTest->Size = System::Draw-
ing::Size(125, 60);
        this->buttonPrintTest->TabIndex = 3;
        this->buttonPrintTest->Text = L"теста";
        this->buttonPrintTest->UseVisualStyleBackColor = true;
        this->buttonPrintTest->Click += gcnew Sys-
tem::EventHandler(this, &form_start::buttonPrintTest_Click);
        //
        // buttonOpen
        //
        this->buttonOpen->Location = System::Drawing::Point(89,
82);
        this->buttonOpen->Name = L"buttonOpen";
        this->buttonOpen->Size = System::Drawing::Size(125,
30);
        this->buttonOpen->TabIndex = 4;
        this->buttonOpen->Text = L"выбрать файл";
        this->buttonOpen->UseVisualStyleBackColor = true;
        this->buttonOpen->Click += gcnew Sys-
tem::EventHandler(this, &form_start::buttonOpen_Click);
        //
        // openFileDialog1
        //
        this->openFileDialog1->FileName = L"openFileDialog1";
        this->openFileDialog1->FileOk += gcnew System::Compo-
nentModel::CancelEventHandler(this, &form_start::openFileDialog1_FileOk);
        //
        // label1
        //
        this->label1->BackColor = System::Drawing::SystemCol-
ors::ButtonHighlight;
        this->label1->BorderStyle = System::Win-
dows::Forms::BorderStyle::FixedSingle;
        this->label1->Location = System::Drawing::Point(65,
20);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(172, 119);
        this->label1->TabIndex = 5;
        //
        // label2
        //
        this->label2->BorderStyle = System::Win-
dows::Forms::BorderStyle::FixedSingle;
        this->label2->Location = System::Drawing::Point(65,
20);
        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(172, 36);
        this->label2->TabIndex = 6;
        //
        // label3
        //
        this->label3->AutoSize = true;
        this->label3->Location = System::Drawing::Point(96,
29);
        this->label3->Name = L"label3";

```

```

        this->label3->Size = System::Drawing::Size(118, 17);
        this->label3->TabIndex = 7;
        this->label3->Text = L"открытие файла";
        //
        // buttonStartMethodForm
        //
        this->buttonStartMethodForm->Enabled = false;
        this->buttonStartMethodForm->Location = System::Draw-
ing::Point(89, 262);
        this->buttonStartMethodForm->Name = L"buttonStartMe-
thodForm";
        this->buttonStartMethodForm->Size = System::Draw-
ing::Size(125, 30);
        this->buttonStartMethodForm->TabIndex = 14;
        this->buttonStartMethodForm->Text = L"перейти";
        this->buttonStartMethodForm->UseVisualStyleBackColor =
true;
        this->buttonStartMethodForm->Click += gcnew Sys-
tem::EventHandler(this, &form_start::buttonVerMethod_Click);
        //
        // label4
        //
        this->label4->AutoSize = true;
        this->label4->Location = System::Drawing::Point(101,
147);
        this->label4->Name = L"label4";
        this->label4->Size = System::Drawing::Size(101, 17);
        this->label4->TabIndex = 17;
        this->label4->Text = L"выбор метода";
        //
        // label5
        //
        this->label5->BorderStyle = System::Win-
dows::Forms::BorderStyle::FixedSingle;
        this->label5->Location = System::Drawing::Point(65,
138);
        this->label5->Name = L"label5";
        this->label5->Size = System::Drawing::Size(172, 36);
        this->label5->TabIndex = 16;
        //
        // label6
        //
        this->label6->BackColor = System::Drawing::SystemCol-
ors::ButtonHighlight;
        this->label6->BorderStyle = System::Win-
dows::Forms::BorderStyle::FixedSingle;
        this->label6->Location = System::Drawing::Point(65,
138);
        this->label6->Name = L"label6";
        this->label6->Size = System::Drawing::Size(172, 164);
        this->label6->TabIndex = 15;
        //
        // label7
        //
        this->label7->AutoSize = true;
        this->label7->Location = System::Drawing::Point(397,
311);
        this->label7->Name = L"label7";
        this->label7->Size = System::Drawing::Size(48, 17);
        this->label7->TabIndex = 20;
        this->label7->Text = L"ВЫВОД";
        //
        // label8

```

```

//
this->label8->BorderStyle = System::Win-
dows::Forms::BorderStyle::FixedSingle;
this->label8->Location = System::Drawing::Point(65,
301);

this->label8->Name = L"label8";
this->label8->Size = System::Drawing::Size(774, 36);
this->label8->TabIndex = 19;
//
// label9
//
this->label9->BackColor = System::Drawing::SystemCol-
ors::ButtonHighlight;
this->label9->BorderStyle = System::Win-
dows::Forms::BorderStyle::FixedSingle;
this->label9->Location = System::Drawing::Point(65,
301);

this->label9->Name = L"label9";
this->label9->Size = System::Drawing::Size(774, 115);
this->label9->TabIndex = 18;
//
// buttonPrintProtocol
//
this->buttonPrintProtocol->Enabled = false;
this->buttonPrintProtocol->Location = System::Draw-
ing::Point(112, 345);
this->buttonPrintProtocol->Name = L"buttonPrintProto-
col";
this->buttonPrintProtocol->Size = System::Draw-
ing::Size(125, 60);

this->buttonPrintProtocol->TabIndex = 21;
this->buttonPrintProtocol->Text = L"протокола";
this->buttonPrintProtocol->UseVisualStyleBackColor =
true;

this->buttonPrintProtocol->Click += gcnew Sys-
tem::EventHandler(this, &form_start::buttonPrintProtocol_Click);
//
// buttonPrintUntesting
//
this->buttonPrintUntesting->Enabled = false;
this->buttonPrintUntesting->Location = System::Draw-
ing::Point(631, 345);
this->buttonPrintUntesting->Name = L"buttonPrintUntest-
ing";
this->buttonPrintUntesting->Size = System::Draw-
ing::Size(125, 60);

this->buttonPrintUntesting->TabIndex = 22;
this->buttonPrintUntesting->Text = L"непроверенных
неисправностей";
this->buttonPrintUntesting->UseVisualStyleBackColor =
true;

this->buttonPrintUntesting->Click += gcnew Sys-
tem::EventHandler(this, &form_start::buttonPrintUntesting_Click);
//
// checkBoxRegMethod
//
this->checkBoxRegMethod->AutoSize = true;
this->checkBoxRegMethod->Checked = true;
this->checkBoxRegMethod->CheckState = System::Win-
dows::Forms::CheckState::Checked;
this->checkBoxRegMethod->Location = System::Draw-
ing::Point(88, 189);
this->checkBoxRegMethod->Name = L"checkBoxRegMethod";

```

```

ing::Size(108, 21);
    this->checkBoxRegMethod->Size = System::Draw-
    this->checkBoxRegMethod->TabIndex = 25;
    this->checkBoxRegMethod->Text = L"регулярный";
    this->checkBoxRegMethod->UseVisualStyleBackColor =
true;
    this->checkBoxRegMethod->CheckedChanged += gcnew Sys-
tem::EventHandler(this, &form_start::checkBoxRegMethod_CheckedChanged);
    //
    // checkBoxVerMethod
    //
    this->checkBoxVerMethod->AutoSize = true;
    this->checkBoxVerMethod->Location = System::Draw-
ing::Point(88, 228);
    this->checkBoxVerMethod->Name = L"checkBoxVerMethod";
    this->checkBoxVerMethod->Size = System::Draw-
ing::Size(132, 21);
    this->checkBoxVerMethod->TabIndex = 26;
    this->checkBoxVerMethod->Text = L"вероятностный";
    this->checkBoxVerMethod->UseVisualStyleBackColor =
true;
    this->checkBoxVerMethod->CheckedChanged += gcnew Sys-
tem::EventHandler(this, &form_start::checkBoxVerMethod_CheckedChanged);
    //
    // label10
    //
    this->label10->BorderStyle = System::Win-
dows::Forms::BorderStyle::Fixed3D;
    this->label10->Location = System::Drawing::Point(-19,
425);
    this->label10->Name = L"label10";
    this->label10->Size = System::Drawing::Size(945, 56);
    this->label10->TabIndex = 27;
    //
    // labelFileName
    //
    this->labelFileName->AutoSize = true;
    this->labelFileName->Location = System::Draw-
ing::Point(12, 433);
    this->labelFileName->MaximumSize = System::Draw-
ing::Size(897, 490);
    this->labelFileName->Name = L"labelFileName";
    this->labelFileName->Size = System::Drawing::Size(0,
17);
    this->labelFileName->TabIndex = 28;
    //
    // form_start
    //
    this->AutoScaleDimensions = System::Drawing::SizeF(8,
16);
    this->AutoScaleMode = System::Windows::Forms::Au-
toScaleMode::Font;
    this->ClientSize = System::Drawing::Size(913, 478);
    this->Controls->Add(this->labelFileName);
    this->Controls->Add(this->checkBoxVerMethod);
    this->Controls->Add(this->checkBoxRegMethod);
    this->Controls->Add(this->buttonPrintUntesting);
    this->Controls->Add(this->buttonPrintProtocol);
    this->Controls->Add(this->label7);
    this->Controls->Add(this->label8);
    this->Controls->Add(this->label4);
    this->Controls->Add(this->label5);
    this->Controls->Add(this->buttonStartMethodForm);

```

```

        this->Controls->Add(this->label3);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->buttonOpen);
        this->Controls->Add(this->buttonPrintTest);
        this->Controls->Add(this->richTextBox1);
        this->Controls->Add(this->label1);
        this->Controls->Add(this->label6);
        this->Controls->Add(this->label9);
        this->Controls->Add(this->label10);
        this->MaximizeBox = false;
        this->Name = L"form_start";
        this->StartPosition = System::Win-
dows::Forms::FormStartPosition::CenterScreen;
        this->Text = L"log_schcem_test";
        this->FormClosed += gcnw System::Windows::Forms::Form-
ClosedEventHandler(this, &form_start::form_start_FormClosed);
        this->Load += gcnw System::EventHandler(this,
&form_start::form_start_Load);
        this->ResumeLayout(false);
        this->PerformLayout();

    }
#pragma endregion
    //private: System::Void buttonRegMethod_Click(System::Object^
sender, System::EventArgs^ e);
    //private: System::Void buttonResult_Click(System::Object^
sender, System::EventArgs^ e);
    private: System::Void buttonOpen_Click(System::Object^ sender, Sys-
tem::EventArgs^ e);
    private: System::Void openFileDialog1_FileOk(System::Object^ sender,
System::ComponentModel::CancelEventArgs^ e) {}
    private: System::Void buttonVerMethod_Click(System::Object^ sender,
System::EventArgs^ e);
    private: System::Void form_start_Load(System::Object^ sender, Sys-
tem::EventArgs^ e) { }
    private: System::Void checkBoxVerMethod_CheckedChanged(System::Object^
sender, System::EventArgs^ e);
    private: System::Void buttonPrintUntesting_Click(System::Object^
sender, System::EventArgs^ e);
    private: System::Void buttonPrintProtocol_Click(System::Object^
sender, System::EventArgs^ e);
    private: System::Void buttonPrintTest_Click(System::Object^ sender,
System::EventArgs^ e);
    private: System::Void form_start_FormClosed(System::Object^ sender,
System::Windows::Forms::FormClosedEventArgs^ e);
    private: System::Void checkBoxRegMethod_CheckedChanged(System::Object^
sender, System::EventArgs^ e);

    private: System::Void printResult(char*);
    };
}

////////////////////////////////////
////////////////////////////////////
#define INT SIZE 32

UINT Mainpr(LPVOID pParam);
UINT FromFile(LPVOID pParam);
int Pow2(int);
void ReadElement(FILE*);
int Convertor();
void ClearArrays();

```

```

int ReadData ();
void SaveData ();

void DeleteFiles ();

```

Header.h

```

#pragma once
#include <cstring>
#include <string>

//using namespace System;
//using namespace System::Runtime::InteropServices;

System::String^ charToString(char*);
char* stringToChar(System::String^);

void myAlert(char* msg);
void myAlert(const char* msg);
//void myAlert(CStringA msg);
void myAlert(System::String^ msg);

```

pch.h

```

// pch.h: This is a precompiled header file.
// Files listed below are compiled only once, improving build performance for
// future builds.
// This also affects IntelliSense performance, including code completion and
// many code browsing features.
// However, files listed here are ALL re-compiled if any one of them is up-
// dated between builds.
// Do not add files here that you will be updating frequently as this negates
// the performance advantage.

#ifndef PCH_H
#define PCH_H

#define _CRT_SECURE_NO_WARNINGS

#include "Header.h"
#include "StdAfx.h"
#include "form_reg_ver_method.h"
#include "form_start.h"

// add headers that you want to pre-compile here

#endif //PCH_H

```

Resource.h

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by KS.rc

```

```

//
#define IDD_ABOUTBOX 100
#define IDP_OLE_INIT_FAILED 100
#define IDR_MAINFRAME 128
#define IDR_KSTYPE 129
#define IDD_RESULT 134
#define IDD_HANDTEST 140
#define IDD_PRDLG 142
#define IDI_NORMFACE 144
#define IDI_WORK 145
#define IDI_OKFACE 146
#define IDI_ABORTFACE 147
#define IDD_VERMET 148
#define IDD_ALLPRDLG 149
#define IDC_RICHEDIT1 1013
#define IDC_BUTTON 1014
#define IDC_INPAT 1017
#define IDC_OUTPAT 1018
#define IDC_MODEL 1019
#define IDC_LISTPAT 1020
#define IDC_LISTOUT 1021
#define IDC_KURPATLENGTH 1022
#define IDC_L 1023
#define IDC_ACTION 1027
#define IDC_START 1028
#define IDC_NUMFAULTS 1029
#define IDC_PROGFAULTS 1030
#define IDC_TESTCOUNT 1031
#define IDC_STOP 1032
#define IDC_NUMPAT 1033
#define IDC_INFO 1035
#define IDC_PatFaults 1036
#define IDC_AllPatFaults 1037
#define IDC_EDIT1 1038
#define IDC_EDIT2 1039
#define IDC_EDIT3 1040
#define IDC_GENERATE 1041
#define IDC_BAR 1042
#define IDC_NUMPATGEN 1044
#define IDC_PROGRESS1 1045
#define ID_RESULT_TEST 32771
#define ID_RESULT_FAULTS 32772
#define ID_RESULT_PROTOKOL 32773
#define ID_TESTING_RM 32774
#define ID_MENUITEM32775 32775
#define ID_RESULT_CLEAR 32776
#define ID__TEXNOLOGI 32777
#define ID_HAND_TEST 32778
#define ID_TEST_FROM_FILE 32779
#define ID_VERMET 32785
#define ID_INDICATOR_INFO 59142

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS 1
#define _APS_NEXT_RESOURCE_VALUE 150
#define _APS_NEXT_COMMAND_VALUE 32788
#define _APS_NEXT_CONTROL_VALUE 1046
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif

```

MyAnal.cpp

```

/*****
/*
/*      Функция вычисляет множество константных
/*      неисправностей из заданного списка, прове-
/*      ряемых данным входным набором.
/*
/*
*****/
#include "pch.h"
#include "MyAnal.h"
#include "MyRegmethod.h"

extern char* fa;
extern int* m11, * m12, * m2, * sm, * ml, * mn, * mh, * nvx, * svx,
* nvix, * svix, * nvixc, * cf;
extern int l, n, ns;
extern int* tst, nhcf, nhcfl, nt, nn;
extern char* lpName;
int c11[32] = { 0x00000001,0x00000002,0x00000004,0x00000008,
                0x00000010,0x00000020,0x00000040,0x00000080,
                0x00000100,0x00000200,0x00000400,0x00000800,
                0x00001000,0x00002000,0x00004000,0x00008000,
                0x00010000,0x00020000,0x00040000,0x00080000,
                0x00100000,0x00200000,0x00400000,0x00800000,
                0x01000000,0x02000000,0x04000000,0x08000000,
                0x10000000,0x20000000,0x40000000,0x80000000 };

CString message;

int z[2000];      /* массив значений сигналов на линиях схемы */
int pan[6000];   /* маска для внесения выбранного
                  /* подмножества неисправностей
int ipan[2000];  /* секционирующий массив для pan
char InpPat[300], OutPat[300];
int nh;
int ncf[32], rdcf[32];

void anal(HWND hWnd)
{
    FILE* fptr;
    //extern UINT WM_MYUPDATE;
    int i, j;
    // int col,lin;
    /* формирование ipan */
    for (j = 0; j <= l; j++)
        ipan[j] = j;
    for (j = 1; j <= ns + l - 2; j++)
        ipan[j + 1] = ipan[j] + ml[j - 1] + mn[j - 1];
    message += "Файл " + CString(lpName) + ".tsc ";

    /*  for(i=0;i<=l+ns-1;i++) printf("ipan[%d]=%d\n",i,ipan[i]); */

    /* подача входного набора на входы схемы */

    for (i = 0; i < l; i++)
    {
        if (fa[i + 1] == '0')
            z[i] = 0;
        else
            if (fa[i + 1] == '1')
                z[i] = -1;
    }
}

```



```

        else
        {
            message += "непонятный сигнал на входе";
            myAlert((const char*)(CStringA)message);
            //MessageBox(NULL, message, "Открытие файла", MB_OK |
            MB_ICONINFORMATION);
            exit(1);
            return;
        }
        /* printf("вх.н.:z[%d]=%x\n",i,z[i]); */
    }

    nh = 0;          /* число неисправностей, проверяемых */
                    /* заданным входным набором          */

    /* *****
    */
    comne();          /* реализация алгоритма парал- */
                    /* лельного моделирования неис- */
                    /* правностей типа "const 0(1)" */
                    /* для зад. входного набора      */

    /* *****
    */

    nn += nh;          /* количество неисправностей, обнаруживаемых */
                    /* входной последовательностью */
    strcat(LpName, ".pte");
    //fptr = fopen(LpName, "a");
    if ((fptr = fopen(LpName, "a")) == NULL)
        //MessageBox(NULL, "Ошибка открытия файла", "Невозможно открыть файл
        содержащий протокол", MB_OK | MB_ICONINFORMATION);
    myAlert("Ошибка открытия файла. Невозможно открыть файл содержащий
    протокол");
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    fprintf(fptr, "входной набор: ");
    for (i = 0; i < l; i++)
    {
        if ((z[i] & c11[31]) == c11[31])
        {
            InpPat[i] = '1';
            ipan[i] = 1;
        }
        else
        {
            InpPat[i] = '0';
            ipan[i] = 0;
        }
        fprintf(fptr, "%d", ipan[i]);
    }
    fprintf(fptr, "\nвыходная реакция: ");
    for (i = 0; i < n; i++)
    {
        if ((z[nvixc[i] - 1] & c11[31]) == c11[31])
        {
            OutPat[i] = '1';
            ipan[i] = 1;
        }
        else
        {
            OutPat[i] = '0';
            ipan[i] = 0;
        }
    }

```

```

    }
    fprintf(fptr, "%d", ipan[i]);
}

/* формирование теста схемы */
if (nh != 0)
{
    intst();
    nt++;
}

fprintf(fptr, "\nвходной набор обн. %d неисправностей\n", nh);
nhcf1 -= nh; /* количество необнаруженных неисправностей */
fprintf(fptr, "входная последовательность обн. %d неисправностей\n\n",
nhcf - nhcf1); /* nn
//if (hWnd != NULL) SendMessage(hWnd, WM_MYUPDATE, 4, 0);
//Sleep(1000);

fclose(fptr);
}

//❖'ключение тестового набора в тест схемы
void intst()
{
    int c1[32] = { 0x80000000, 0x40000000, 0x20000000, 0x10000000,
                  0x8000000, 0x4000000, 0x2000000, 0x1000000,
                  0x800000, 0x400000, 0x200000, 0x100000,
                  0x80000, 0x40000, 0x20000, 0x10000,
                  0x8000, 0x4000, 0x2000, 0x1000,
                  0x800, 0x400, 0x200, 0x100,
                  0x80, 0x40, 0x20, 0x10,
                  0x8, 0x4, 0x2, 0x1 };

    int i, j, k, i1;
    i1 = 1 / 32 + n / 32;
    if ((j = 1 % 32) != 0) i1++;
    if ((j = n % 32) != 0) i1++;
    j = i1 * tst[0] + 1;
    k = 0;
    for (i = 0; i < 1; i++)
    {
        if (InpPat[i] == '1') tst[j] |= c1[k];
        if (k != 31) k++;
        else
        {
            k = 0;
            j++;
        }
    }
    if (k != 0)
    {
        k = 0;
        j++;
    }
    for (i = 0; i < n; i++)
    {
        if (OutPat[i] == '1') tst[j] |= c1[k];
        if (k != 31) k++;
        else
        {
            k = 0;
            j++;
        }
    }
}

```

```

    }
    tst[0]++;
}

/*****
/*
/*      Функция реализует алгоритм параллельного моделирования
/*      неисправностей типа const 0(1) из заданного списка для задан-
/*      ного входного набора.
*****/

void comne()
{
    int i, j, k, pr, os, nrel;
    nh = 0;
    pr = 0; /* установка признака моделирования */
            /* неисправности типа const 0 */

    k = ipan[l + ns - 1] + ml[ns - 1] + mn[ns - 1] - 1; /* подготовка масок
для */
    for (i = 0; i <= k; i++) /* внесения неисправностей
*/
        pan[i] = -1; /* к моделированию исправной
схемы */
    /* printf("ipan[%d]=%d\n",l+ns-1,ipan[l+ns-1]); */
    /* printf("k=%d\n",k); */
    /* printf("в comne:\n"); */
    /* for(i=0;i<=k;i++) printf("pan[%d]=%x\n",i,pan[i]); */
    /* printf("const 0\n"); */

/*****
simul(pr); /* моделирование */
            /* исправной схемы */
*****/

    for (i = 0; i < l + ns; i++)
    {
        /* подготовка массива значений на линиях */
        if ((z[i] & c11[31]) == c11[31]) /* схемы к внесению неис-
правностей */
            z[i] = -1;
        else
            z[i] = 0;
        /* printf("мод.ис.сх.:z[%d]=%d\n",i,z[i]); */
    }

    i = 0;
    j = 0;
    k = 0;
    while (i < (4 * ns + 1))
    {
        if (i < l) /* на входах схемы */
        {
            /* printf("входы схемы:\n"); */
            if ((z[i] & cf[i] & c11[31]) != 0) /* тип неисправности не совпа-
дает */
                /* со значением на линии испр.схемы?
*/
            {

```

```

        if (j == 31)          /* внесена 31 неисправность из списка */
        {

/*****
        iskl(pr); /* удаление проверяемых */
        /* неисправностей из списка */
        /* ска и подсчет их числа */

/*****
        j = 0;          /* очистка счетчика вносимых неисправностей */
        }
        j++;
        /* printf("j0=%d\n",j); */
        pan[i] = ~c11[31 - j]; /* внесение неисправности */
        /* printf("pan[%d]=%x\n",i,pan[i]); */
        ncf[j] = i;          /* номер соотв. элемента cf */

        rdcf[j] = 31;          /* разряд элемента cf */
    }
    if (i > (l - 1))          /* не на первичных входах */
    {
        os = (i - 1) % 4;
        nrel = (i - 1) / 4 - os / 4;
        /* printf("элемент N %d\n",nrel+1); */
        if (os == 0)          /* неисправности на входах элемента */
        {
            /* printf("вход эл-та\n"); */
            while (k < ml[nrel])
            {
                if (j == 31)
                {

/*****
                iskl(pr); /* подсчет числа неисправностей и удаления их из списка */

/*****
                j = 0; /* очистка счетчика вносимых неисправностей */
                }
                if (((z[nvx[svx[nrel] + k] - 1] & c11[31]) != 0) &&
((cf[i] & c11[31 - k]) != 0)) /* сигнал на входе */

/* элемента не совпадает с типом */

/*      неисправности ?      */
                {
                    j++;
                    /* printf("j0=%d\n",j); */
                    pan[ipan[l + nrel] + k] = ~c11[31 - j]; /* внесение
неисправности */

                    /* printf("pah=%x\n",pan[ipan[l+nrel]+k]); */
                    ncf[j] = i;          /* номер элемента cf */
                    rdcf[j] = 31 - k;    /* разряд элемента cf

                */

                }
                k++;
            }
            k = 0;
        }
    }

```



```

*/      pan[i] = 0;                                /* моделированию исправной схемы
*/
/* **** */
simul(pr); /* моделирование ис- */
/* правой схемы */
/* **** */
for (i = 0; i < 1 + ns; i++)
{
    /* получение 32 экземпляров */
    if ((z[i] & c11[31]) == c11[31])
        z[i] = -1;                                /* моделей */
    else
        z[i] = 0;                                /* исправной */
    /* printf("м.и.с:z[%d]=%d\n", i, z[i]); */
}
/* схемы */

i = 0;
j = 0;
k = 0;
while (i < (4 * ns + 1))
{
    if (i < 1)                                    /* входы схемы */
    {
        /* printf("входы схемы:\n"); */
        if (((z[i] & c11[31]) == 0) && ((cf[i] & c11[30]) != 0))
            /* сигнал на входе не совп.с типом */
            /* неисправности? */
            {
                if (j == 31)                        /* внесена 31 неисправность */
                {
                    /* **** */
                    iskl(pr); /* подсчет числа про- */
                                /* веряемых неисправностей */
                                /* и исключение их из списка */
                    /* **** */

                    j = 0;                          /* сброс счетчика вносимых неисправностей
*/
                }
                j++;
                /* printf("j1=%d\n", j); */
                pan[i] = c11[31 - j];                /* внесение неисправности */
                /* printf("pan[%d]=%x\n", i, pan[i]); */
                ncf[j] = i;                          /* номер элемента cf */
                rdcf[j] = 30;                        /* номер разряда элемента cf */
            }
        else
            /* printf("Z[i]=%d\n", z[i]) */;
    }
    if (i > (1 - 1))                                /* линии, не явл. первичными входами */
    {
        os = (i - 1) % 4;
        nrel = (i - 1) / 4 - os / 4;
        /* printf("элемент N %d\n", nrel+1); */
        if (os == 1)                                /* элемент cf, соотв. const 1 на входах элемента
*/
        {
            /* printf("вход эл-та\n"); */
            while (k < ml[nrel])
            {
                if (j == 31)                        /* внесена 31 неисправность */
                {

```



```

        rdcf[j] = 31 - k;
    }
    k++;
}
k = 0;
}
}
i++;
}
if (j != 0) /* внесено менее 31 неисправности данного типа */
{
    /* printf("Jl===%d\n", j); */
    /******
    iskl(pr); /* подсчет числа про- */
              /*веряемых неисрав- */
              /*ностей и удаление их из списка */
    /******
    j = 0;
    /* printf("nh в comne=%d\n", *nh); */
}
}

/******
/*
/*      Функция производит подсчет числа неисправностей ,
/*      проверяемых данным входным набором , и удаляет
/*      соответствующие неисправности из списка
/******

void iskl(int pr)
{
    int i, j, k, a, b, aijk;
    int w[32]; /* вектор обнаружимости */

    /******
    simul(pr); /* параллельное модели-
                /*рование неисправностей */
    /******
    /* printf("\n iskl:z[0]=%x,z[1]=%x\n", z[0], z[1]); */
    for (j = 1; j < 32; j++) //16 - 32
    {
        /* формирование */
        w[j] = 0;
        for (k = 0; k < n; k++) /* вектора */
        {
            a = z[nvixc[k] - 1] & c11[31]; /* обнаружимости */
            b = z[nvixc[k] - 1] & c11[31 - j];
            if (a != 0)
                a = 1;
            if (b != 0)
                b = 1;
            if (a == b)
                aijk = 0;
            else
                aijk = 1;
            w[j] = w[j] | aijk;
        }
    }
    for (j = 1; j < 32; j++)
    {
        /* подсчет проверяемых несиправностей */
        if (w[j] != 0)
        {
            /* и исключение их */
            if ((cf[ncf[j]] & c11[rdcf[j]]) != 0)

```



```

        {
            cf[ncf[j]] = cf[ncf[j]] & (~(c11[rdcf[j]]));
            nh++; /* из списка */
        }
    }
}
/* printf("в iskl nh=%d\n", *nh); */
k = ipan[l + ns - 1] + ml[ns - 1] + mn[ns - 1] - 1;
for (i = 0; i <= k; i++) /* подготовка к внесению */
{
    if (pr == 0)
        pan[i] = -1; /* очередной пачки неисправностей
*/
    else
        pan[i] = 0;
}
for (i = 0; i < l; i++)
{
    if ((z[i] & c11[31]) == c11[31])
        z[i] = -1;
    else
        z[i] = 0;
}
}

/*****
/*
/* Функция вычисляет значения сигналов на линиях 32
/* экземпляров логической схемы
/*
/*
/*****

void simul(int pr)
{
    int i, j, k, t, in, ih, m, zh, nzh;
    /* printf("SIMUL:\n"); */
    if (pr == 0) /* const 0 */
    { /* printf("неисправности на входах схемы:\n"); */
        for (i = 0; i < l; i++) /* внесение неисправностей */
        {
            z[i] = z[i] & pan[i]; /* printf("z[%d]=%x\n", i, z[i]); */
        } /* на входы схемы */
        for (i = 0; i < ns; i++)
        {
            k = ml[i]; /* количество входов i-го элемента */
            in = mn[i]; /* количество выходов i-го элемента */
            ih = mh[i]; /* число различных элементарных конъюнкций */
            /* в системе "НФ функций i-го элемента */
            for (t = 0; t < in; t++)
            { /* реализация формального аппарата
*/
                z[nvix[svix[i] + t] - 1] = 0;
                for (j = 0; j < ih; j++) /* параллельного моде-
лирования */
                {
                    zh = -1; /* значений сигналов */
                    nzh = -1;
                    for (m = 0; m < k; m++) /* на линиях
исправной схемы */
                    {

```

```

        if ((m11[sm[i] + j] & c11[31 - m]) != 0) /* и с
внесенными неисправностями */
            zh = zh & (z[nvx[svx[i] + m] - 1] & pan[ipan[l +
i] + m]);
        if ((m12[sm[i] + j] & c11[31 - m]) != 0)
/* типа const 0 */
            nzh = nzh & (~ (z[nvx[svx[i] + m] - 1] &
pan[ipan[l + i] + m]));
    }
    if ((m2[sm[i] + j] & c11[31 - t]) != 0)
        z[nvix[svix[i] + t] - 1] = z[nvix[svix[i] + t] - 1] |
(zh & nzh);
    }
    z[nvix[svix[i] + t] - 1] = z[nvix[svix[i] + t] - 1] &
pan[ipan[i + l] + k + t];
}
}
}
/*-----
-*/
if (pr == 1) /* const 1 */
{
    /* printf("неисправности на входах схемы;\n"); */
    for (i = 0; i < l; i++) /* внесение неисправностей */
    {
        z[i] = z[i] | pan[i];
        /*printf("z[%d]=%x\n", i, z[i]);*/
    }
    /* на входы схемы */
    for (i = 0; i < ns; i++)
    {
        k = ml[i]; /* число входов i-го элемента */
        in = mn[i]; /* число выходов i-го элемента */
        ih = mh[i]; /* число различных элементарных
*/
/* конъюнкций в системе ? "НФ функ-
ций */
/* i-го элемента схемы */
        for (t = 0; t < in; t++)
        {
            z[nvix[svix[i] + t] - 1] = 0; /* реализация формального
*/
            for (j = 0; j < ih; j++) /* аппарата параллельного */
            {
                zh = -1;
                nzh = -1; /* моделирования значений
*/
                for (m = 0; m < k; m++) /* сигналов на линиях исправ-
ной */
                {
                    if ((m11[sm[i] + j] & c11[31 - m]) != 0)
                        zh = zh & (z[nvx[svx[i] + m] - 1] | pan[ipan[l +
i] + m]); /* схемы и схемы */
                    if ((m12[sm[i] + j] & c11[31 - m]) != 0)
                        nzh = nzh & (~ (z[nvx[svx[i] + m] - 1] |
pan[ipan[l + i] + m])); /* с внесенными */
                    if ((m2[sm[i] + j] & c11[31 - t]) != 0) /* неис-
правностями типа */
                        z[nvix[svix[i] + t] - 1] = z[nvix[svix[i] + t] - 1] |
(zh & nzh); /* const 1 */
                    }
                    z[nvix[svix[i] + t] - 1] = z[nvix[svix[i] + t] - 1] |
pan[ipan[l + i] + k + t];

```

```

    }
}
}

```

MyElements.cpp

```

/* ♦ 'ычисление условий образования существенного пути от места */
/* неисправности до выходных полюсов логической сети */
#include "pch.h"
#include "Myelements.h"
#include "MyRegmethod.h"

extern int* okb, * cokb, * vi, * ma, * nlin, * ba,
* bd, * oid, * coid, pc[], fl, s, w, v;
extern char* f, * fa;
void ucltr()
{
    int il, j1, q;
    int i, iot;
    iot = 0;
    s = 0;
me2: for (i = 0; i <= (pc[6] + pc[14]); i++)
fa[i] = f[i];
v = w;
ba[0] = nlin[0];
il = 1;
j1 = 1;
fl = 1;
q = oid[coid[nlin[0]] + 1];
while (q > 0)
{ /* q - признак активности */
    if (okb[cokb[nlin[il]] + 1] == 1)
        /* активизация пути через элемент типа ♦ */
        uti(&il, &j1, &q);
    else
        if (okb[cokb[nlin[il]] + 1] == 2)
            /* активизация пути через элемент типа ♦Л♦ */
            utili(&il, &j1, &q);
        else
            /* активизация пути через элемент типа НЕ */
            utne(&il, &j1, &q);
    /* printf("ucltr: э----|т: %d, fa: %c%c%c%c%c%c%c%c%c%c%c%c\n",nlin[il],
*/
    /* fa[1],fa[2],fa[3],fa[4],fa[5],fa[6],fa[7],fa[8],fa[9], */
    /* fa[10],fa[11],fa[12]); */
    if (s > 0) return;
}
iot++;
if (iot == 101) return;
for (il = il - 1; il > 0; il--)
{
    if (ma[il] == 1)
    {
        ma[il] = 2;
        goto me2;
    }
    if (ma[il] > 1)
        ma[il] = 0;
}
}

```

```

}

/* Активизация пути через элемент типа ? */
void uti(int* i1, int* j1, int* q)
{
    int i;
    /* Анализ значений на входах элемента и выбор варианта активизации */
    if (fa[okb[cokb[nlin[*i1]] + 3]] == 'D') goto m13;
    if (fa[okb[cokb[nlin[*i1]] + 3]] == '1') goto m35;
    if (fa[okb[cokb[nlin[*i1]] + 3]] == 'B') goto m24;
    if (fa[okb[cokb[nlin[*i1]] + 3]] == '0') goto m44;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m5;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m3;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m2;
    ma[*i1] = 0;
    *i1 = *i1 + 1;
    return;
m2: fa[nlin[*i1]] = '0';
    ma[*i1] = 2;
    goto m11;
m3: *q = *q - 1;
    if (ma[*i1] == 0) goto m4;
    if (ma[*i1] == 2) goto m6;
    fa[okb[cokb[nlin[*i1]] + 3]] = '1';
    fa[nlin[*i1]] = 'B';
    *q = *q + oid[coid[nlin[*i1]] + 1];
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
    goto m9;
m4: fa[okb[cokb[nlin[*i1]] + 3]] = '1';
    fa[nlin[*i1]] = 'B';
    *q = *q + oid[coid[nlin[*i1]] + 1];
    goto m8;
m5: *q = *q - 1;
    if (ma[*i1] == 0) goto m7;
    if (ma[*i1] == 2) goto m6;
    fa[okb[cokb[nlin[*i1]] + 3]] = '1';
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v = v + 1;
    fa[nlin[*i1]] = 'D';
    q = q + oid[coid[nlin[*i1]] + 1];
    goto m9;
m6: fa[okb[cokb[nlin[*i1]] + 3]] = '0';
    fa[nlin[*i1]] = '0';
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
    goto m11;
m7: fa[okb[cokb[nlin[*i1]] + 3]] = '1';
    fa[nlin[*i1]] = 'D';
    *q = *q + oid[coid[nlin[*i1]] + 1];
m8: ma[*i1] = 1;
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
m9: for (i = 0; i < pc[5]; i++)
    {
        if (nlin[*i1] == vi[i]) goto m12;
    }
m11: ba[*j1] = nlin[*i1];
    f1++;
    *i1 = *i1 + 1;
    *j1 = *j1 + 1;
    return;
m12: ba[*j1] = nlin[*i1];

```


```

f1++;
s = 1;
return;
m13: *q = *q - 1;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m14;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m18;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m19;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m115;
if (ma[*i1] == 0) goto m20;
if (ma[*i1] == 1) goto m21;
fa[okb[cokb[nlin[*i1]] + 4]] = '0';
fa[nlin[*i1]] = '0';
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
goto m11;
m14: *q = *q - 1;
m115: fa[nlin[*i1]] = 'D';
*q = *q + oid[coid[nlin[*i1]] + 1];
ma[*i1] = 2;
goto m9;
m18: *q = *q - 1;
m19: fa[nlin[*i1]] = '0';
ma[*i1] = 2;
goto m11;
m20: fa[okb[cokb[nlin[*i1]] + 4]] = '1';
fa[nlin[*i1]] = 'D';
ma[*i1] = 1;
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m21: fa[okb[cokb[nlin[*i1]] + 4]] = '1';
fa[nlin[*i1]] = 'D';
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m24: *q = *q - 1;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m31;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m25;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m32;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m26;
if (ma[*i1] == 0) goto m33;
if (ma[*i1] == 1) goto m34;
fa[okb[cokb[nlin[*i1]] + 4]] = '0';
fa[nlin[*i1]] = '0';
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
goto m11;
m25: *q = *q - 1;
m26: fa[nlin[*i1]] = 'B';
ma[*i1] = 2;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m31: *q = *q - 1;
m32: fa[nlin[*i1]] = '0';
ma[*i1] = 2;
goto m11;
m33: fa[okb[cokb[nlin[*i1]] + 4]] = '1';
fa[nlin[*i1]] = 'B';
ma[*i1] = 1;
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;

```

```

*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m34: fa[okb[cokb[nlin[*i1]] + 4]] = '1';
fa[nlin[*i1]] = 'B';
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m35: if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m36;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m41;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m43;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m42;
ma[*i1] = 0;
*i1 = *i1 + 1;
return;
m36: *q = *q - 1;
fa[nlin[*i1]] = 'D';
ma[*i1] = 2;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m41: *q = *q - 1;
fa[nlin[*i1]] = 'B';
ma[*i1] = 2;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m42: fa[nlin[*i1]] = '1';
ma[*i1] = 2;
goto m11;
m43: fa[nlin[*i1]] = '0';
ma[*i1] = 2;
goto m11;
m44: fa[nlin[*i1]] = '0';
if ((fa[okb[cokb[nlin[*i1]] + 4]] != 'D') && (fa[okb[cokb[nlin[*i1]] + 4]] !=
'B'))
goto m71;
*q = *q - 1;
m71: ma[*i1] = 2;
ba[*j1] = nlin[*i1];
f1++;
*i1 = *i1 + 1;
*j1 = *j1 + 1;
return;
}

/* Активизация пути через элемент типа  */
void utili(int* i1, int* j1, int* q)
{
    int i;
    /* анализ значений на входах элемента и выбор варианта активизации */
    if (fa[okb[cokb[nlin[*i1]] + 3]] == 'D') goto m13;
    if (fa[okb[cokb[nlin[*i1]] + 3]] == 'B') goto m34;
    if (fa[okb[cokb[nlin[*i1]] + 3]] == '1') goto m33;
    if (fa[okb[cokb[nlin[*i1]] + 3]] == '0') goto m24;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m5;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m3;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m2;
    ma[*i1] = 0;
    *i1 = *i1 + 1;
    return;
m2: fa[nlin[*i1]] = '1';
ma[*i1] = 2;
goto m11;
m3: *q = *q - 1;

```

```

    if (ma[*i1] == 0) goto m4;
    if (ma[*i1] == 2) goto m6;
    fa[okb[cokb[nlin[*i1]] + 3]] = '0';
    fa[nlin[*i1]] = 'B';
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
    *q = *q + oid[coid[nlin[*i1]] + 1];
    goto m9;
m4:  fa[okb[cokb[nlin[*i1]] + 3]] = '0';
    fa[nlin[*i1]] = 'B';
    *q = *q + oid[coid[nlin[*i1]] + 1];
    goto m8;
m5:  *q = *q - 1;
    if (ma[*i1] == 2) goto m6;
    if (ma[*i1] == 0) goto m7;
    fa[okb[cokb[nlin[*i1]] + 3]] = '0';
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
    fa[nlin[*i1]] = 'D';
    *q = *q + oid[coid[nlin[*i1]] + 1];
    goto m9;
m6:  fa[okb[cokb[nlin[*i1]] + 3]] = '1';
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
    fa[nlin[*i1]] = '1';
    goto m11;
m7:  fa[okb[cokb[nlin[*i1]] + 3]] = '0';
    fa[nlin[*i1]] = 'D';
    *q = *q + oid[coid[nlin[*i1]] + 1];
m8:  ma[*i1] = 1;
    bd[v] = okb[cokb[nlin[*i1]] + 3];
    v++;
m9:  for (i = 0; i < pc[5]; i++)
    {
        if (nlin[*i1] == vi[i]) goto m12;
    }
m11: ba[*j1] = nlin[*i1];
    fl++;
    *i1 = *i1 + 1;
    *j1 = *j1 + 1;
    return;
m12: ba[*j1] = nlin[*i1];
    fl++;
    s = 1;
    return;
m13: *q = *q - 1;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m18;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m15;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m19;
    if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m16;
    if (ma[*i1] == 0) goto m17;
    if (ma[*i1] == 1) goto m14;
    fa[okb[cokb[nlin[*i1]] + 4]] = '1';
    fa[nlin[*i1]] = '1';
    bd[v] = okb[cokb[nlin[*i1]] + 4];
    v++;
    goto m11;
m14: fa[okb[cokb[nlin[*i1]] + 4]] = '0';
    bd[v] = okb[cokb[nlin[*i1]] + 4];
    v++;
    fa[nlin[*i1]] = 'D';
    *q = *q + oid[coid[nlin[*i1]] + 1];
    goto m9;

```

```

m15: *q = *q - 1;
m16: fa[nlin[*i1]] = '1';
ma[*i1] = 2;
goto m11;
m17: fa[okb[cokb[nlin[*i1]] + 4]] = '0';
fa[nlin[*i1]] = 'D';
*q = *q + oid[coid[nlin[*i1]] + 1];
ma[*i1] = 1;
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
goto m9;
m18: *q = *q - 1;
m19: fa[nlin[*i1]] = 'D';
*q = *q + oid[coid[nlin[*i1]] + 1];
ma[*i1] = 2;
goto m9;
m24: if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m28;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m25;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m26;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m27;
ma[*i1] = 0;
*i1 = *i1 + 1;
return;
m25: *q = *q - 1;
fa[nlin[*i1]] = 'B';
ma[*i1] = 2;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m26: fa[nlin[*i1]] = '0';
ma[*i1] = 2;
goto m11;
m27: fa[nlin[*i1]] = '1';
ma[*i1] = 2;
goto m11;
m28: *q = *q - 1;
fa[nlin[*i1]] = 'D';
ma[*i1] = 2;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m33: fa[nlin[*i1]] = '1';
if ((fa[okb[cokb[nlin[*i1]] + 4]] == 'D') || (fa[okb[cokb[nlin[*i1]] + 4]] ==
'B'))
*q = *q - 1;
ma[*i1] = 2;
ba[*j1] = nlin[*i1];
f1++;
*i1 = *i1 + 1;
*j1 = *j1 + 1;
return;
m34: *q = *q - 1;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'D') goto m37;
if (fa[okb[cokb[nlin[*i1]] + 4]] == 'B') goto m39;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '0') goto m40;
if (fa[okb[cokb[nlin[*i1]] + 4]] == '1') goto m38;
if (ma[*i1] == 1) goto m35;
if (ma[*i1] == 0) goto m36;
fa[okb[cokb[nlin[*i1]] + 4]] = '1';
fa[nlin[*i1]] = '1';
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
goto m11;
m35: fa[okb[cokb[nlin[*i1]] + 4]] = '0';
fa[nlin[*i1]] = 'B';

```



```

bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m36: fa[okb[cokb[nlin[*i1]] + 4]] = '0';
fa[nlin[*i1]] = 'B';
ma[*i1] = 1;
bd[v] = okb[cokb[nlin[*i1]] + 4];
v++;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
m37: *q = *q - 1;
m38: fa[nlin[*i1]] = '1';
ma[*i1] = 2;
goto m11;
m39: *q = *q - 1;
m40: fa[nlin[*i1]] = 'B';
ma[*i1] = 2;
*q = *q + oid[coid[nlin[*i1]] + 1];
goto m9;
}

```

/* Активизация пути через элемент типа HE */

```

void utne(int* i1, int* j1, int* q)
{
    int i;
    /* Анализ значения на входе и выбор варианта активизации */
    if (fa[okb[cokb[nlin[*i1]] + 3]] == 'X')
    {
        ma[*i1] = 0;
        *i1 = *i1 + 1;
        return;
    }
    if (fa[okb[cokb[nlin[*i1]] + 3]] == '1')
    {
        fa[nlin[*i1]] = '0';
        ma[*i1] = 2;
        ba[*j1] = nlin[*i1];
        f1++;
        *j1 = *j1 + 1;
        *i1 = *i1 + 1;
        return;
    }
    if (fa[okb[cokb[nlin[*i1]] + 3]] == '0')
    {
        fa[nlin[*i1]] = '1';
        ma[*i1] = 2;
        ba[*j1] = nlin[*i1];
        f1++;
        *j1 = *j1 + 1;
        *i1 = *i1 + 1;
        return;
    }
    if (fa[okb[cokb[nlin[*i1]] + 3]] == 'D')
    {
        *q = *q - 1;
        fa[nlin[*i1]] = 'B';
m1: *q = *q + oid[coid[nlin[*i1]] + 1];
        ma[*i1] = 2;
        for (i = 0; i < pc[5]; i++)
        {

```

```

        if (nlin[*i1] == vi[i])
        {
            ba[*j1] = nlin[*i1];
            fl++;
            s = 1;
            return;
        }
        ba[*j1] = nlin[*i1];
        fl++;
        *j1 = *j1 + 1;
        *i1 = *i1 + 1;
        return;
    }
    *q = *q - 1;
    fa[nlin[*i1]] = 'D';
    goto m1;
}

```

MyOpacf.cpp

```

/* Обеспечение условий образования существенного пути */
#include "pch.h"
#include "myopacf.h"
#include "myregmethod.h"

extern int* okb, * cokb, * bd, pc[], v, s;
extern char* fa, * fd;
int md[4000], bdr[4000], bdr1[4000];
void opacf()
{
    int i1, j1, pr;
    int i, iot;
    iot = 0;
    s = 0; /* начальная установка признака результата решения задачи */
    for (i = 0; i <= (pc[6] + pc[14]); i++)
    {
        md[i] = 0;
        bdr[i] = 0;
    }
    for (i = 0; i < v; i++)
        bdr[bd[i]] = bd[i];
    for (i = 0; i <= (pc[6] + pc[14]); i++)
        bdr1[i] = bdr[i];

m4: for (i = 0; i <= (pc[6] + pc[14]); i++)
    fd[i] = fa[i];
    j1 = 0;
    /* Обеспечение заданных значений на выходах элементов из списка bdr1 */
    for (i1 = pc[6] + pc[14]; i1 > pc[14];)
    {
        if (bdr1[i1] != 0) /* выбор элемента из списка bdr1 */
        {
            pr = 0;
            if (okb[cokb[i1] + 1] == 1)
                /* Обеспечение заданного значения на выходе элемента типа ? */
                opfi(&i1, &j1, &pr);
            else
                if (okb[cokb[i1] + 1] == 2)
                    /* Обеспечение заданного значения на выходе элемента типа
?Л? */

```

```

        opfil(&i1, &j1, &pr);
    else
        /* Обеспечение заданного значения на выходе элемента типа НЕ
*/
        opfne(&i1, &j1, &pr);
    if (pr == 1) /* возврат назад */
    {
        iot++;
        if (iot == 100) return;
        goto m4;
    }
    if (pr > 1) return; /* обеспечить условия активизации нельзя */
}
else
    i1--; /* переход к выбору очередного элемента */
}
s = 1; /* условия образования существенного пути обеспечены */
}

/* Обеспечение заданного значения на выходе элемента типа ? */
void opfi(int* i1, int* j1, int* pr)
{
    int i;
    if (fd[*i1] == '1') goto m15; /* значение 1 на выходе элемента */
/* Обеспечение значения 0 на выходе элемента */
/* Анализ значений на входах элемента */
    if (fd[okb[cokb[*i1] + 3]] == '0') goto m6;
    if (fd[okb[cokb[*i1] + 3]] == '1') goto m7;
    if (fd[okb[cokb[*i1] + 4]] == '0') goto m3;
    if (fd[okb[cokb[*i1] + 4]] == '1') goto m4;
    if (md[*j1] == 0) goto m5; /* выбор первого варианта обеспечения */
    if (md[*j1] == 1) goto m2; /* выбор второго варианта обеспечения */
    fd[okb[cokb[*i1] + 4]] = '0';
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    if (okb[cokb[*i1 + 1] + 4] > pc[14])
        bdr1[okb[cokb[*i1 + 1] + 4]] = okb[cokb[*i1 + 1] + 4];
    return;
m2: fd[okb[cokb[*i1] + 3]] = '0';
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    if (okb[cokb[*i1 + 1] + 3] > pc[14])
        bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
    return;
m3: md[*j1] = 2;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    return;
m4: fd[okb[cokb[*i1] + 3]] = '0';
    md[*j1] = 2;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    if (okb[cokb[*i1 + 1] + 3] > pc[14])
        bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
    return;
m5: fd[okb[cokb[*i1] + 3]] = '0';
    md[*j1] = 1;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    if (okb[cokb[*i1 + 1] + 3] > pc[14])
        bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
    return;

```

```

m6: md[*j1] = 2;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    return;
m7: if (fd[okb[cokb[*i1] + 4]] == '0')
{
    md[*j1] = 2;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    return;
}
if (fd[okb[cokb[*i1] + 4]] == '1') goto m9;
fd[okb[cokb[*i1] + 4]] = '0';
md[*j1] = 2;
*j1 = *j1 + 1;
*i1 = *i1 - 1;
if (okb[cokb[*i1 + 1] + 4] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 4]] = okb[cokb[*i1 + 1] + 4];
return;
m9: md[*j1] = 0;
if (*j1 > 0) goto m10;
*pr = 2;
return;
m10: *j1 = *j1 - 1;
if (*j1 >= 0) goto m11;
*pr = 2;
return;
m11: if (md[*j1] == 0) goto m10;
if (md[*j1] == 1) goto m12;
md[*j1] = 0;
goto m10;
m12: md[*j1] = 2;
for (i = 0; i <= (pc[6] + pc[14]); i++)
bdr1[i] = bdr[i];
*pr = 1;
return;
/* Обеспечение значения 1 на выходе элемента */
m15: if (fd[okb[cokb[*i1] + 3]] == '0') goto m9; /* возврат назад */
if (fd[okb[cokb[*i1] + 3]] == '1') goto m18;
if (fd[okb[cokb[*i1] + 4]] == '0') goto m9; /* возврат назад */
if (fd[okb[cokb[*i1] + 4]] == '1') goto m17;
fd[okb[cokb[*i1] + 3]] = '1';
fd[okb[cokb[*i1] + 4]] = '1';
md[*j1] = 2;
*j1 = *j1 + 1;
*i1 = *i1 - 1;
if (okb[cokb[*i1 + 1] + 3] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
if (okb[cokb[*i1 + 1] + 4] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 4]] = okb[cokb[*i1 + 1] + 4];
return;
m17: fd[okb[cokb[*i1] + 3]] = '1';
md[*j1] = 2;
*j1 = *j1 + 1;
*i1 = *i1 - 1;
if (okb[cokb[*i1 + 1] + 3] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
return;
m18: if (fd[okb[cokb[*i1] + 4]] == '0') goto m9;
if (fd[okb[cokb[*i1] + 4]] == '1') goto m19;
fd[okb[cokb[*i1] + 4]] = '1';
md[*j1] = 2;
*j1 = *j1 + 1;

```

```

*i1 = *i1 - 1;
if (okb[cokb[*i1 + 1] + 4] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 4]] = okb[cokb[*i1 + 1] + 4];
return;
m19: md[*j1] = 2;
*j1 = *j1 + 1;
*i1 = *i1 - 1;
return;
}

// Обеспечение заданного значения на выходе элемента типа Л
void opfil(int* i1, int* j1, int* pr)
{
    int i;
    if (fd[*i1] == '1') goto m7; // значение 1 на выходе элемента
    // Обеспечение значения 0 на выходе элемента
    // Анализ значений на входах элемента
    if (fd[okb[cokb[*i1] + 3]] == '0') goto m5;
    if (fd[okb[cokb[*i1] + 3]] == '1') goto m15; // возврат назад
    if (fd[okb[cokb[*i1] + 4]] == '0') goto m4;
    if (fd[okb[cokb[*i1] + 4]] == '1') goto m15; // возврат назад
    fd[okb[cokb[*i1] + 3]] = '0';
    fd[okb[cokb[*i1] + 4]] = '0';
    md[*j1] = 2;
    *i1 = *i1 - 1;
    *j1 = *j1 + 1;
    if (okb[cokb[*i1 + 1] + 3] > pc[14])
        bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
m3:  if (okb[cokb[*i1 + 1] + 4] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 4]] = okb[cokb[*i1 + 1] + 4];
return;
m4:  fd[okb[cokb[*i1] + 3]] = '0';
md[*j1] = 2;
m33: *i1 = *i1 - 1;
*j1 = *j1 + 1;
if (okb[cokb[*i1 + 1] + 3] > pc[14])
bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
return;
m5:  if (fd[okb[cokb[*i1] + 4]] == '0') goto m6;
if (fd[okb[cokb[*i1] + 4]] == '1') goto m15;
fd[okb[cokb[*i1] + 4]] = '0';
md[*j1] = 2;
m32: *i1 = *i1 - 1;
*j1 = *j1 + 1;
goto m3;
m6:  md[*j1] = 2;
*i1 = *i1 - 1;
*j1 = *j1 + 1;
return;
m7:  if (fd[okb[cokb[*i1] + 3]] == '0') goto m13;
if (fd[okb[cokb[*i1] + 3]] == '1') goto m6;
if (fd[okb[cokb[*i1] + 4]] == '0') goto m11;
if (fd[okb[cokb[*i1] + 4]] == '1') goto m6;
if (md[*j1] == 0) goto m9;
if (md[*j1] == 1) goto m8;
m31: fd[okb[cokb[*i1] + 4]] = '1';
goto m32;
m8:  fd[okb[cokb[*i1] + 3]] = '1';
goto m33;
m9:  fd[okb[cokb[*i1] + 3]] = '1';
md[*j1] = 1;
goto m33;
m11: md[*j1] = 2;

```

```

goto m8;
// Анализ значения на втором входе
m13:  if (fd[okb[cokb[*i1] + 4]] == '0') goto m15; // возврат назад
if (fd[okb[cokb[*i1] + 4]] == '1') goto m6;
md[*j1] = 2;
goto m31;
m15:  md[*j1] = 0;
if (*j1 > 0) goto m16;
m35:  *pr = 2; // обеспечить заданное значение нельзя
return;
m16:  *j1 = *j1 - 1;
if (*j1 < 0) goto m35;
if (md[*j1] == 0) goto m16;
if (md[*j1] == 1) goto m18;
md[*j1] = 0;
goto m16;
m18:  md[*j1] = 2;
for (i = 0; i <= (pc[6] + pc[14]); i++)
bdr1[i] = bdr[i];
*pr = 1; // возврат назад
return;
}

// Обеспечение заданного значения на выходе элемента типа HE
void opfne(int* i1, int* j1, int* pr)
{
    int i;
    if (fd[*i1] == '1') goto m2; // значение 1 на выходе элемента
// значение 0 на выходе элемента
// Анализ значения на входе элемента
    if ((fd[okb[cokb[*i1] + 3]] == '0') || (fd[okb[cokb[*i1] + 3]] == 'X'))
        goto m4;
m13:  md[*j1] = 2;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    return;
m2:  if (fd[okb[cokb[*i1] + 3]] == '0') goto m13;
    if (fd[okb[cokb[*i1] + 3]] == '1') goto m5;
    fd[okb[cokb[*i1] + 3]] = '0';
m17:  md[*j1] = 2;
    *j1 = *j1 + 1;
    *i1 = *i1 - 1;
    if (okb[cokb[*i1 + 1] + 3] > pc[14])
        bdr1[okb[cokb[*i1 + 1] + 3]] = okb[cokb[*i1 + 1] + 3];
    return;
m4:  if (fd[okb[cokb[*i1] + 3]] == '0') goto m5;
    fd[okb[cokb[*i1] + 3]] = '1';
    goto m17;
m5:  md[*j1] = 0;
    if (*j1 > 0) goto m6;
m18:  *pr = 2; // Обеспечить заданное значение нельзя
    return;
m6:  *j1 = *j1 - 1;
    if (*j1 < 0) goto m18;
    if (md[*j1] == 0) goto m6;
    if (md[*j1] == 1) goto m8;
    md[*j1] = 0;
    goto m6;
m8:  md[*j1] = 2;
    for (i = 0; i <= (pc[6] + pc[14]); i++)

```



```

        j++;
    }
    for (i1 = 0; i1 < n; i1++)
    {
        if ((tst[j] & c1[k]) != 0)
            fprintf(ftstd, "%c", '1');
        else
            fprintf(ftstd, "%c", '0');
        if (k != 31) k++;
        else
        {
            k = 0;
            j++;
        }
    }
    fprintf(ftstd, "\n");
    if (k != 0)
    {
        k = 0;
        j++;
    }
}
fclose(ftstd);
}

extern int PointIdent[];
extern char Ident[];

extern int* ml, * mn, * nvx, * svx, * nvix, * svix, * nct, * cf, nhcf1;

void printcf()
{
    int c1[32] = { 0x80000000, 0x40000000, 0x20000000, 0x10000000
        , 0x08000000, 0x04000000, 0x02000000, 0x01000000
        , 0x00800000, 0x00400000, 0x00200000, 0x00100000
        , 0x00080000, 0x00040000, 0x00020000, 0x00010000
        , 0x00008000, 0x00004000, 0x00002000, 0x00001000
        , 0x00000800, 0x00000400, 0x00000200, 0x00000100
        , 0x00000080, 0x00000040, 0x00000020, 0x00000010
        , 0x00000008, 0x00000004, 0x00000002, 0x00000001 };

    FILE* fcfid;
    int i, j, k, i1, count, pr;
    char IdentEl[40]; /* идентификатор элемента*/
    strcat(LpName, ".fff");
    //fcfid = fopen(LpName, "w");
    if ((fcfid = fopen(LpName, "w")) == NULL)
        //MessageBox(NULL, "Невозможно открыть файл типа .fff", "Ошибка ",
        MB_OK | MB_ICONERROR);
    myAlert("невозможно открыть файл типа .fff");
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения

    fprintf(fcfid, "
        СПСОК НЕПРОЕРЕННЫХ НЕСПРАНО-
        СТЕЙ\n");
    for (i = 0; i < 70; i++) fprintf(fcfid, "*");
    fprintf(fcfid, "\n список %d неисправностей\n", nhcf1);
    i1 = 0;

    for (i = 0; i < 1; i++)
    {

```



```

    if ((cf[i] & c1[0]) != 0)
    {
        count = PointIdent[i + 3] - PointIdent[i + 2];
        fprintf(fcfd, " ");
        for (j = 0; j < count; j++)
            fprintf(fcfd, "%c", Ident[PointIdent[i + 2] + j]);
        fprintf(fcfd, " 0");
        if (i1 != 4) i1++;
        else
        {
            i1 = 0;
            fprintf(fcfd, "\n");
        }
    }
    if ((cf[i] & c1[1]) != 0)
    {
        count = PointIdent[i + 3] - PointIdent[i + 2];
        fprintf(fcfd, " ");
        for (j = 0; j < count; j++)
            fprintf(fcfd, "%c", Ident[PointIdent[i + 2] + j]);
        fprintf(fcfd, " 1");
        if (i1 != 4) i1++;
        else
        {
            i1 = 0;
            fprintf(fcfd, "\n");
        }
    }
}
if (i1 != 0) fprintf(fcfd, "\n");
/*❖'ывод списка неисправностей для элементов */

for (i = 0; i < ns; i++)
{
    pr = 0;
    count = PointIdent[nct[nvix[svix[i]]] + 2] - PointI-
dent[nct[nvix[svix[i]]] + 1];
    i1 = 0; /* i1= числу символов в индефикаторе элементов */
    for (j = 0; j < count; j++)
    {
        if (Ident[PointIdent[nct[nvix[svix[i]]] + 1] + j] == '(') break;
        else
        {
            i1++;
            IdentEl[j] = Ident[PointIdent[nct[nvix[svix[i]]] + 1] + j];
        }
    }
    for (j = 0; j < ml[i]; j++)
    {
        if ((cf[l + i * 4] & c1[j]) != 0)
        {
            pr++;
            for (k = 0; k < i1; k++)
                fprintf(fcfd, "%c", IdentEl[k]);
            fprintf(fcfd, ":");
            count = PointIdent[nct[nvx[svx[i] + j]] + 2] -
                PointIdent[nct[nvx[svx[i] + j]] + 1];

            for (k = 0; k < count; k++)
                fprintf(fcfd, "%c", Ident[PointIdent[nct[nvx[svx[i] + j]]
+ 1] + k]);

            fprintf(fcfd, " 0");//
        }
    }
}

```

```

        if ((cf[l + i * 4 + 1] & c1[j]) != 0)
        {
            pr++;
            for (k = 0; k < i1; k++)
                fprintf(fcfd, "%c", IdentEl[k]);
            fprintf(fcfd, ":");
            count = PointIdent[nct[nvx[svx[i] + j]] + 2] -
                PointIdent[nct[nvx[svx[i] + j]] + 1];

            for (k = 0; k < count; k++)
                fprintf(fcfd, "%c", Ident[PointIdent[nct[nvx[svx[i] + j]]
+ 1] + k]);

            fprintf(fcfd, " 1");//
        }
    }
    for (j = 0; j < mn[i]; j++)
    {
        if ((cf[l + i * 4 + 2] & c1[j]) != 0)
        {
            pr++;
            count = PointIdent[nct[nvix[svix[i] + j]] + 2] -
                PointIdent[nct[nvix[svix[i] + j]] + 1];
            for (k = 0; k < count; k++)
                fprintf(fcfd, "%c", Ident[PointIdent[nct[nvix[svix[i] +
j]] + 1] + k]);

            fprintf(fcfd, " 0");//
        }
        if ((cf[l + i * 4 + 3] & c1[j]) != 0)
        {
            pr++;
            count = PointIdent[nct[nvix[svix[i] + j]] + 2] -
                PointIdent[nct[nvix[svix[i] + j]] + 1];
            for (k = 0; k < count; k++)
                fprintf(fcfd, "%c", Ident[PointIdent[nct[nvix[svix[i] +
j]] + 1] + k]);

            fprintf(fcfd, " 1");//
        }
    }
    if (pr != 0) fprintf(fcfd, "\n");
}
fclose(fcfd);
return;
}/* end printcf*/

```

MyRegmethod.cpp

```

// Построение модели исходной схемы, ориентированной на решение
// основных задач, возникающих при вычислении проверяющих
// наборов теста на основе регулярного метода
//////////
#include "pch.h"
#include "myregmethod.h"
#include "form_start.h"
// #include "KSDoc.h"
#include "myelements.h" !!
#include "myanal.h"
#include "myopacf.h" !!
//////////
extern int g_NumFaults;
extern char* LpName;

```

```

extern int* m11, * m12, * m2, * sm, * ml, * mn, * mh, * nvx, * svx, * nvix, *
svix,
* nvixc, * cf, l, n, ns, nt, nhcf, nhcfl, * tst, * nctl, * ctn, * nct;

int* ba, * ma, * nlin, * bd;
int w, im, c, fl, v, s, nn;
extern FILE* fptr;
int* cn, * nc, * cokb;
int* mne, * okb, * vi, pc[25], * oid, * coid;
char* f, * fd;
extern char* fa;
int d;
int c1[32] = { 0x80000000,0x40000000,0x20000000,0x10000000
,0x08000000,0x04000000,0x02000000,0x01000000
,0x00800000,0x00400000,0x00200000,0x00100000
,0x00080000,0x00040000,0x00020000,0x00010000
,0x00008000,0x00004000,0x00002000,0x00001000
,0x00000800,0x00000400,0x00000200,0x00000100
,0x00000080,0x00000040,0x00000020,0x00000010
,0x00000008,0x00000004,0x00000002,0x00000001 };
int modcx(HWND hWnd)
{
    int nipla[64], nppla[48], nct[48], iokb, neb, is, npla, ioid, ioid1;
    int s, q, t, i, il, j1, k/*,k1,i2,i5,i11*/ , j;

    il = l + 1;
    for (i = 0; i < ns; i++)
        il += mn[i];

    if ((cn = new int[il]) == NULL)        return 4;
    if ((nc = new int[9000]) == NULL)     return 4;
    if ((cokb = new int[9000]) == NULL)   return 4;
    if ((okb = new int[20000]) == NULL)   return 4;
    if ((vi = new int[300]) == NULL)      return 4;
    if ((oid = new int[20000]) == NULL)   return 4;
    if ((coid = new int[9000]) == NULL)   return 4;
    if ((mne = new int[9000]) == NULL)    return 4;

    for (i = 0; i < il; i++)
        cn[i] = 0;
    for (i = 0; i < 9000; i++)
        nc[i] = cokb[i] = coid[i] = mne[i] = 0;
    for (i = 0; i < 20000; i++)
        okb[i] = oid[i] = 0;
    for (i = 0; i < 300; i++)
        vi[i] = 0;
    // запись в файл
    strcat(LpName, ".pte");
    if ((fptr = fopen(LpName, "a")) == NULL) return 13; //файл отч'та
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    fprintf(fptr, "РЕ-УЛЬТАТЫ ПОСТРОЕНИЯ ТЕСТА " "ЛЯ СХЕМЫ\n");
    fprintf(fptr, "РЕ-УЛЯРНЫМ МЕТО-ОМ\n\n");
    for (int I = 0; I < l + 20; I++) fprintf(fptr, "*");
    fprintf(fptr, "*\n");
    fprintf(fptr, "l= %d, n= %d, ns= %d\n", l, n, ns);
    fclose(fptr);
    //
    // данном месте кода программы должно находится описания
    //сообщений о процессе генерации тест
    //
    //

```

```

for (i = 0; i < 9000; i++)
    nc[i] = 0;
// Построение массивов okb, cokb, mne, vi, ps, oid, coid, задающих
// описание логической сети, эквивалентной исходной схеме
for (i = 0; i <= 1; i++)
{
    cokb[i] = 0;
    nc[i] = i;
    cn[i] = i;
}
neb = 1 + 1;
iokb = 0;

for (npla = 0; npla < ns; npla++)
{
    s = m1[npla];
    q = mh[npla];
    t = mn[npla];

    for (i = 0; i < s; i++)
    {
        nipla[2 * i] = cn[nvx[svx[npla] + i]];

        for (j = 0; j < q; j++)
        {
            if ((m12[sm[npla] + j] & c1[i]) != 0x00000000)
            {
                okb[iokb] = neb;
                okb[iokb + 1] = 3;
                okb[iokb + 2] = 1;
                okb[iokb + 3] = cn[nvx[svx[npla] + i]];
                cokb[neb] = iokb;
                iokb = iokb + 4;
                nipla[2 * i + 1] = neb;
                neb++;
                j = q;
            }
        }
    }

    for (i = 0; i < q; i++)
    {
        is = 0;
        for (j = 0; j < s; j++)
        {
            if ((m11[sm[npla] + i] & c1[j]) != 0x00000000)
            {
                nct[is] = nipla[2 * j];
                is++;
            }
            if ((m12[sm[npla] + i] & c1[j]) != 0x00000000)
            {
                nct[is] = nipla[2 * j + 1];
                is++;
            }
        }
        if (is < 2)
            nppla[i] = nct[0];
        else
        {
            for (j = 0; j < is - 1; j++)
            {
                okb[iokb] = neb;

```

```

        okb[iokb + 1] = 1;
        okb[iokb + 2] = 2;
        okb[iokb + 3] = nct[j];
        okb[iokb + 4] = nct[j + 1];
        cokb[neb] = iokb;
        iokb = iokb + 5;
        nct[j + 1] = neb;
        neb++;
    }
    nppla[i] = neb - 1;
}

for (j = 0; j < t; j++)
{
    is = 0;
    for (i = 0; i < q; i++)
    {
        if ((m2[sm[npla] + i] & c1[j]) != 0x00000000)
        {
            nct[is] = nppla[i];
            is++;
        }
    }

    if (is < 2)
    {
        cn[nvix[svix[npla] + j]] = nct[0];
        nc[nct[0]] = nvix[svix[npla] + j];
    }
    else
    {
        for (i = 0; i < is - 1; i++)
        {
            okb[iokb] = neb;
            okb[iokb + 1] = 2;
            okb[iokb + 2] = 2;
            okb[iokb + 3] = nct[i];
            okb[iokb + 4] = nct[i + 1];
            cokb[neb] = iokb;
            iokb = iokb + 5;
            nct[i + 1] = neb;
            neb++;
        }
        cn[nvix[svix[npla] + j]] = neb - 1;
        nc[neb - 1] = nvix[svix[npla] + j];
    }
}

neb--;

pc[0] = nt;
pc[5] = n;
pc[6] = neb - 1;
pc[14] = 1;
for (i = 0; i < n; i++)
    vi[i] = cn[nvixc[i]];
ioid = 0;

```

```

for (i = 1; i <= neb; i++)
{
    ioid1 = ioid + 2;
    coid[i] = ioid;
    oid[ioid] = i;
    oid[ioid + 1] = 0;
    if (i != neb)
    {
        if (i > 1)
            i1 = i + 1;
        else
            i1 = 1 + 1;
        for (j = i1; j <= neb; j++)
        {
            k = okb[cokb[j] + 2];
            for (j1 = 0; j1 < k; j1++)
            {
                if (i == okb[cokb[j] + 3 + j1])
                {
                    oid[ioid + 1]++;
                    oid[ioid1] = j;
                    ioid1++;
                    j1 = k;
                }
            }
            ioid = ioid1;
        }
    }
}

coid[neb + 1] = ioid1;

// Построение списка неисправностей модели схемы соответствующей
// списку неисправностей исходной схем
kodne()

// Построение теста для заданного списка неисправностей
regme(hWnd)

// вывод списка необнаруженных неисправностей
/*
strcat( LpName, ".pte");
if((fptr=fopen(LpName,"a"))==NULL) return 18;//файл протокол
LpName[strlen(LpName)-4] = 0x00; // Удаление расширения
k1=1;
fprintf(fptr,"Необнаруженные неисправности\n");
for(i1=1;i1<=l;i1=i1+10)
{
    if((i1+9)<=k1)
        i2=i1+9;
    else
        i2=k1;
    for(i5=i1;i5<=i2;i5++)
        fprintf(fptr,"%2d ",i5);
    fprintf(fptr,"\n");
    for(i5=i1;i5<=i2;i5++)
        fprintf(fptr,"%4x ",cf[i5-1]);
    fprintf(fptr,"\n");
}
k1=l+ns;
for(i1=l+1;i1<=l+ns;i1=i1+10)
    if((i1+9)<=k1)

```

```

        i2=i1+9;
    else
        i2=k1;
    for(i5=i1;i5<=i2;i5++)
        fprintf(fptr,"%2d",nct1[i5]);
    fprintf(fptr,"\n");
    for(i=0;i<2;i++)
        i11=1+(i1-1-1)*4+2;
        for(i5=i1;i5<=i2;i5++)
            fprintf(fptr,"%4x",cf[i11+i]);
        i11=i11+4;
    }
    fprintf(fptr,"\n");

}
delete[] mne;
delete[] coid;
delete[] oid;
delete[] vi;
delete[] okb;
delete[] cokb;
delete[] nc;
delete[] cn;
return 14;//тест построе
}

// Реализация направленного метода построения теста для заданног
// списка неисправностей логической сети
int regme(HWND hWnd)
{
    int kc, i, r, /*d,*/p, iot, pr/*,j*/;
    static int conl[6] = {
        0x00000001,0x00000002,0x00000004,0x00000008,0x00000010,0x00000020 };

    if ((ba = new int[pc[6] + 1 + 1]) == NULL) return 4;
    if ((ma = new int[pc[6] + 1 + 1]) == NULL) return 4;
    if ((nlin = new int[pc[6] + 1 + 1]) == NULL) return 4;
    if ((bd = new int[pc[6] + 1 + 1]) == NULL) return 4;
    if ((f = new char[pc[6] + 1 + 1]) == NULL) return 4;
    if ((fa = new char[pc[6] + 1 + 1]) == NULL) return 4;
    if ((fd = new char[pc[6] + 1 + 1]) == NULL) return 4;
    for (i = 0; i <= (pc[6] + 1); i++)
        ba[i] = ma[i] = nlin[i] = bd[i] = 0;

    strcat(LpName, ".pte");
    if ((fptr = fopen(LpName, "a")) == NULL) return 13;//файл отч'т
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    iot = nn = 0;
    kc = 1;
    for (i = 0; i < ns; i++)
        kc += mn[i];
    // 'ыбор очередной неисправности из массива неисправности
    im = 1;
me2:  if (im > (pc[6] + pc[14])) goto me28; // список исчерпа
    r = 0;
    if (im <= pc[14]) goto me25; // неисправность на входе схем
// выбирается неисправность логического элемент
    c = 1;
me3:  if ((mne[im - 1] & conl[c - 1]) == 0x00000000) goto me22;
    d = 1;
    if (okb[cokb[im] + 1] == 1) goto me11; // элемент типа

```

```

        if (okb[cokb[im] + 1] == 2) goto me12; // элемент типа ?Л?
        dknne(); // элемент типа Н?
me4:   fprintf(fptr, "Связь %d, неисправность %d;\n", nct[nc[im]], c);
        /* ?'ыход элемента является выходом логической сети ? */
        for (i = 0; i < pc[5]; i++)
        {
            if (im == vi[i]) goto me10;
        }
me6:   if (r == 1) goto me7; /* r=1 --> путь уже построен */
        p = 0;
        clput(); /* построение пути к выходам схемы */
        /* printf("Сложный путь \n"); */
        /* printf("%d %d %d %d %d %d %d %d %d \n", nc[nlin[0]], nc[nlin[1]], *
        /* nc[nlin[2]], nc[nlin[3]], nc[nlin[4]], nc[nlin[5]], nc[nlin[6]], *
        /* nc[nlin[7]], nc[nlin[8]], nc[nlin[9]]); */
        /* ?'ычисление условий образования активизированного пути */
me7:   r = 1;
        for (i = 0; i < (pc[6] + 1 + 1); i++) ma[i] = 0;
me8:   ucltr();
        iot++;
        if (s == 0) goto me13; /* ограничение перебора */
        if (iot == 100)
        {
            fprintf(fptr, "Ограничение перебора\n");
            goto me13;
        }
        if (v == 0) goto me16; /* список для доопределения пуст */
        /* Обеспечение условий образования активизированного пути */
me9:   opacf();
        if (s == 0) goto me19; /* возврат к поиску других условий акт. пути */
        /* ?'ывод построенного теста */
me60:  fprintf(fptr, "Тест\n");
        /* for(i=1;i<=kc;i++)
            fprintf(fptr,"%c",fd[cn[ctn[i]]]);
            fprintf(fptr,"\n"); */
        iot = 0;
        pr = 0;
me46:  for (i = 1; i <= pc[14]; i++)
        {
            if (fd[i] == '0') goto me91;
            if (fd[i] == '1') goto me92;
            if (fd[i] == 'D') goto me95;
            if (pr == 0) goto me91;
me92:   fa[i] = '1';
            continue;
me91:   fa[i] = '0';
            continue;
me95:   if (pr == 0) goto me92;
            goto me91;
        }
        fclose(fptr);
        /* ?'ычисление множества неисправностей, проверяемых построенным */
        /* тестом, и удаление их из списка неисправностей */
        anal(hWnd);
        strcat(LpName, ".pte");
        if ((fptr = fopen(LpName, "a")) == NULL) return 13; // файл отч?т?
        LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
        kodne();
        if ((im > pc[14]) || (pr == 1)) goto me74; /* выбор очередной неисправности */
        pr = 1;
        goto me46;

```



```

me74:  if (im <= pc[14]) goto me83;
goto me22;
/*  'выход элемента является выходом сети --> путь активизирован */
me10:  p = 1;
for (i = 0; i <= (pc[6] + pc[14]); i++)
fa[i] = f[i];
v = w;
goto me9;
/* Построение D-куба неисправности для элемента типа  *
me11:  dkni();
goto me4;
/* Построение D-куба неисправности для элемента типа  *
me12:  dknili();
goto me4;
me13:  fprintf(fp, "Необнаружимая неисправность\n");
/* Переход к выбору очередной неисправности */
iot = 0;
if (im > pc[14]) goto me67;
im++;
goto me2;
me67:  if (okb[cokb[im] + 1] == 1) goto me15;
if (okb[cokb[im] + 1] == 3) goto me23;
if ((c > 1) || (d == 2)) goto me22;
d++;
goto me12;
me15:  if ((c != 2) || (d == 2)) goto me22;
d++;
goto me11;
me16:  if (p > 0) goto me18;
for (i = 0; i <= (pc[6] + pc[14]); i++)
fd[i] = fa[i];
goto me60;
/* me18:  for(i=1;i<=kc;i++)
        fprintf(fp, "%c", f[cn[ctn[i]]]);
        fprintf(fp, "\n"); */
me18:  goto me22;
me19:  if (p > 0) goto me13;
for (i = 1 + pc[6]; i > 0; i--)
{
    if (ma[i] == 1) goto me21;
    ma[i] = 0;
    continue;
me21:  ma[i] = 2;
    goto me8;
}
goto me13;
me22:  if (okb[cokb[im] + 1] == 3) goto me23;
if (c == 6) goto me24;
me79:  c++;
goto me3;
me23:  if (c < 4) goto me79;
me24:  im++;
goto me2;
me25:  if (((mne[im - 1] & conl[0]) != 0x00000000) || ((mne[im - 1] &
conl[1]) != 0x00000000))
goto me82;
me83:  im++;
goto me2;
me82:  w = 0;
for (i = 1; i <= (pc[6] + pc[14]); i++)
f[i] = 'X';
f[im] = 'D';

```

```

fprintf(fptr, "Неисправность на %d входе\n", im);
goto me6;
me28: fprintf(fptr, "Тест построен\n");
fclose(fptr);

delete[] fd;
delete[] fa;
delete[] f;
delete[] bd;
delete[] nlin;
delete[] ma;
delete[] ba;
return 14;

/* Построение списка неисправностей модели схемы, соответствующего *
/* списку неисправностей исходной схемы */
void kodne()
{
    int i, j;
    for (i = 0; i < 9000; i++)
        mne[i] = 0x00000000;
    /* Построение списка неисправностей на входах схемы */
    for (i = 0; i < 1; i++)
    {
        if ((cf[i] & c1[0]) != 0x00000000) // 0x0000
            mne[i] = mne[i] | c1[31];
        if ((cf[i] & c1[1]) != 0x00000000)
            mne[i] = mne[i] | c1[30];
    }
    /* Построение списка неисправностей логических элементов */
    for (i = 0; i < ns; i++)
    {
        for (j = 0; j < mn[i]; j++)
        {
            if ((cf[1 + 4 * i + 2] & c1[j]) != 0x00000000)
                mne[cn[nvix[svix[i] + j]] - 1] = c1[31];
            if ((cf[1 + 4 * i + 3] & c1[j]) != 0x0000)
                mne[cn[nvix[svix[i] + j]] - 1] = mne[cn[nvix[svix[i] + j]] -
1] | c1[30];
        }
    }
    /* printf("mne: %x %x %x %x %x %x %x %x %x %x %x %x\n", mne[0], mne[1], *
    /* mne[2], mne[3], mne[4], mne[5], mne[6], mne[7], mne[8], mne[9], mne[10], *
    /* mne[11]); */

    /* Построение сложного пути от места неисправности до выходов сети */
    void clput()
    {
        int i, j, k, n;
        nlin[0] = im;
        n = 0;
        for (i = 0; i <= n; i++)
        {
            if (oid[coid[nlin[i]] + 1] != 0)
            {
                for (j = 0; j < oid[coid[nlin[i]] + 1]; j++)
                {
                    for (k = 0; k <= n; k++)
                    {
                        if (nlin[k] == oid[coid[nlin[i]] + 2 + j])
                            k = n + 2;
                    }
                }
            }
        }
    }
}

```

```

        if (k == (n + 1))
            nlin[++n] = oid[coid[nlin[i]] + 2 + j];
    }
}

if (n > 0)
{
    /* Упорядочение элементов пути в порядке возрастания номеров */
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j <= n; j++)
        {
            if (nlin[j] < nlin[i])
            {
                k = nlin[i];
                nlin[i] = nlin[j];
                nlin[j] = k;
            }
        }
    }
}

/* ?'ычисление условий проявления неисправностей для элемента типа НЕ */
void dknne()
{
    int i;
    for (i = 0; i <= (pc[14] + pc[6]); i++)
        f[i] = 'X';
    /* Какая неисправность рассматривается */
    if ((c == 1) || (c == 4))
    {
        /* Неисправность типа "константа 1" на входе или "константа 0" на вы-
ходе */
        f[okb[cokb[im] + 3]] = '0';
        bd[0] = okb[cokb[im] + 3];
        w = 1;
        f[im] = 'D';
        return;
    }
    if ((c == 2) || (c == 3))
    {
        /* Неисправность типа "константа 0" на входе или "константа 1" на вы-
ходе */
        f[okb[cokb[im] + 3]] = '1';
        bd[0] = okb[cokb[im] + 3];
        w = 1;
        f[im] = 'B';
        return;
    }
    //printf("? ' программе dknne c!=1,2,3,4 \n");
}

/* ?'ычисление условий проявления неисправностей для элемента типа ?Л? */
void dknili()
{
    int i;
    for (i = 0; i <= (pc[14] + pc[6]); i++)
        f[i] = 'X';
}

```

```

if (c == 1)
/* Неисправность типа "константа 0" на выходе элемента */
{
    if (d == 1)
        /* 'ыбор первого варианта условий проявления неисправности */
        {
            f[okb[cokb[im] + 4]] = '1';
            bd[0] = okb[cokb[im] + 4];
            w = 1;
            f[im] = 'D';
            return;
        }
        if (d == 2)
            /* 'ыбор второго варианта условий проявления неисправности */
            {
                f[okb[cokb[im] + 3]] = '1';
                bd[0] = okb[cokb[im] + 3];
                w = 1;
                f[im] = 'D';
                return;
            }
        // printf(" ' программе dknili d != 1 и d != 2 \n");
        return;
    }
    if ((c == 2) || (c == 4) || (c == 6))
        /* Неисправность типа "константа 1" на входах и выходе элемента */
        {
            f[okb[cokb[im] + 3]] = '0';
            f[okb[cokb[im] + 4]] = '0';
            bd[0] = okb[cokb[im] + 3];
            bd[1] = okb[cokb[im] + 4];
            w = 2;
            f[im] = 'B';
            return;
        }
        if (c == 3)
            /* Неисправность типа "константа 0" на первом входе элемента */
            {
                f[okb[cokb[im] + 3]] = '1';
                f[okb[cokb[im] + 4]] = '0';
                bd[0] = okb[cokb[im] + 3];
                bd[1] = okb[cokb[im] + 4];
                w = 2;
                f[im] = 'D';
                return;
            }
            if (c == 5)
                /* Неисправность типа "константа 0" на втором входе элемента */
                {
                    f[okb[cokb[im] + 3]] = '0';
                    f[okb[cokb[im] + 4]] = '1';
                    bd[0] = okb[cokb[im] + 3];
                    bd[1] = okb[cokb[im] + 4];
                    w = 2;
                    f[im] = 'D';
                    return;
                }
                // printf(" ' программе dknili c!=1,2,3,4,5,6 \n");
            }
        /* 'ычисление условий проявления неисправностей для элемента типа ' */

```

```

void dkni()
{
    int i;
    for (i = 0; i <= (pc[14] + pc[6]); i++)
        f[i] = 'X';
    /* Какая неисправность рассматривается ? */
    if ((c == 1) || (c == 3) || (c == 5))
        /* Неисправность типа "константа 0" на входах или выходе элемента */
        {
            f[okb[cokb[im] + 3]] = '1';
            f[okb[cokb[im] + 4]] = '1';
            bd[0] = okb[cokb[im] + 3];
            bd[1] = okb[cokb[im] + 4];
            w = 2;
            f[im] = 'D';
            return;
        }
    if (c == 2)
        /* Неисправность типа "константа 1" на выходе элемента */
        {
            /* Выбор первого варианта условий проявления неисправности */
            if (d == 1)
                {
                    f[okb[cokb[im] + 3]] = '0';
                    bd[0] = okb[cokb[im] + 3];
                    w = 1;
                    f[im] = 'B';
                    return;
                }
            if (d == 2)
                /* Выбор второго варианта условий проявления неисправности */
                {
                    f[okb[cokb[im] + 4]] = '0';
                    bd[0] = okb[cokb[im] + 4];
                    w = 1;
                    f[im] = 'B';
                    return;
                }
            // printf("In the programm DKNI d!=1 and d!=2 \n");
            return;
        }
    if (c == 4)
        /* Неисправность типа "константа 1" на первом входе элемента */
        {
            f[okb[cokb[im] + 3]] = '0';
            f[okb[cokb[im] + 4]] = '1';
            bd[0] = okb[cokb[im] + 3];
            bd[1] = okb[cokb[im] + 4];
            w = 2;
            f[im] = 'B';
            return;
        }
    if (c == 6)
        /* Неисправность типа "константа 1" на втором входе элемента */
        {
            f[okb[cokb[im] + 3]] = '1';
            f[okb[cokb[im] + 4]] = '0';
            bd[0] = okb[cokb[im] + 3];
            bd[1] = okb[cokb[im] + 4];
            w = 2;
            f[im] = 'B';
            return;
        }
}

```

```
// printf("In the programm DKNI c!= 1,2,3,4,5,6 \n");
```

MyVermet.cpp

```
//#include "stdafx.h"
#include "pch.h"
#include "myvermet.h"
#include "ksdoc.h"
#include "myanal.h"

int NumPatGen = 0;

UINT vermet(LPVOID pParam) {
    extern char* fa, * LpName;
    extern int l, nn, nhcf, nh;
    int lp, grsum, c11[32] = {
        0x00000001, 0x00000002, 0x00000004, 0x00000008,
        0x00000010, 0x00000020, 0x00000040, 0x00000080,
        0x00000100, 0x00000200, 0x00000400, 0x00000800,
        0x00001000, 0x00002000, 0x00004000, 0x00008000,
        0x00010000, 0x00020000, 0x00040000, 0x00080000,
        0x00100000, 0x00200000, 0x00400000, 0x00800000,
        0x01000000, 0x02000000, 0x04000000, 0x08000000,
        0x10000000, 0x20000000, 0x40000000, 0x80000000 };

    int* ch;
    NumPatGen = 0;
    extern char* fa;

    MPStruct* mps = (MPStruct*)pParam;
    //CUIntArray *M=(CUIntArray*)pParam;
    CUIntArray* M = &mps->M;

    lp = l / INT_SIZE;
    ch = new int[lp];
    fa = new char[l + 1];

    //srand(1);

    while (nn < nhcf) {
        SendMessage(mps->hw, M->GetAt(3), 0, NumPatGen);
        PatGener(ch, lp);
        NumPatGen++;
        if (nh < M->GetAt(0)) {
            grsum = 0;
            SendMessage(mps->hw, M->GetAt(3), 1, NumPatGen);
            for (int i = 0; i < M->GetAt(1); i++) {
                PatGener(ch, lp);
                NumPatGen++;
            }
        }
    }
}
```

```

        SendMessage(mps->hw, M->GetAt(3), 5,
NumPatGen);

        grsum += nh;
        if (nn == nhcf) { SendMessage(mps->hw, M-
>GetAt(3), 2, NumPatGen); return 1; }
    }
    if (grsum < M->GetAt(2)) { SendMessage(mps->hw, M-
>GetAt(3), 2, NumPatGen); return 1; }
    }
    SendMessage(mps->hw, M->GetAt(3), 2, NumPatGen);
    delete[] ch;
    delete[] fa;
    return 1;
}

void PatGener(int* ch, int lp) {
    int c11[32] = { 0x00000001,0x00000002,0x00000004,0x00000008,
0x00000010,0x00000020,0x00000040,0x00000080,
0x00000100,0x00000200,0x00000400,0x00000800,
0x00001000,0x00002000,0x00004000,0x00008000,
0x00010000,0x00020000,0x00040000,0x00080000,
0x00100000,0x00200000,0x00400000,0x00800000,
0x01000000,0x02000000,0x04000000,0x08000000,
0x10000000,0x20000000,0x40000000,0x80000000 };
    extern int l;
    extern char* fa;

    for (int i = 0; i < lp + 1; i++) ch[i] = rand(); //генерация вх. набора
    int k = 0;
    while (k < l) {
        if ((ch[k / INT_SIZE] & c11[((k + 1) % INT_SIZE) - 1]) == 0)
            fa[k + 1] = '0'; else fa[k + 1] = '1';
        k++;
    }
    anal(NULL);
    //delete [] fa;
}

```

AssemblyInfo.cpp

```

#include "pch.h"

using namespace System;
using namespace System::Reflection;
using namespace System::Runtime::CompilerServices;
using namespace System::Runtime::InteropServices;
using namespace System::Security::Permissions;

[assembly: AssemblyTitle(L"logschemtest")];
[assembly: AssemblyDescriptionAttribute(L"")];
[assembly: AssemblyConfigurationAttribute(L"")];
[assembly: AssemblyCompanyAttribute(L"")];

```

```
[assembly:AssemblyProductAttribute(L"logschemtest")];
[assembly:AssemblyCopyrightAttribute(L"Copyright (c) 2020")];
[assembly:AssemblyTrademarkAttribute(L"")];
[assembly:AssemblyCultureAttribute(L"")];

[assembly:AssemblyVersionAttribute("1.0.*")];

[assembly:ComVisible(false)];

[assembly:CLSCompliantAttribute(true)];
```

form_reg_ver_method.cpp

```
#include "pch.h"
#include "form_reg_ver_method.h"
#include "MyPrint.h"
#include "MyRegmethod.h"
#include "MyVermet.h"
#include "time.h"

extern char* LpName;
extern FILE* in_file;
extern int l, n, ns, nhcfl, nn, nhcf, NumPatGen;
extern int* tst,*cf;

System::Void logschemtest::form_reg_ver_method::buttonStart_Click(System::Object^ sender, System::EventArgs^ e) {
    clock_t start, end;
    start = clock();
    if (isVerMethod)
        verMethod();
    else
        regMethod();
    end = clock();
    int percent = nn * 100 / nhcf;
    labelPolnota->Text = "Полнота теста " + Convert::ToString(percent) +
"%";
    labelPolnota->Text += ", время " + Convert::ToString(((double)end -
start) / ((double)CLOCKS_PER_SEC)) + " сек";
    labelResult->Text = "❖'ходная последовательность обнаруживает " + nn
+ " неисправностей";
    labelResult->Text += "\n❖' тест включено " + tst[0] + " наборов";
    if (NumPatGen)
        labelResult->Text += "\nСгенерировано " + Con-
vert::ToString(NumPatGen) + " входных наборов";
    progressBar1->Value = percent;
}

System::Void logschemtest::form_reg_ver_method::form_reg_ver_method_Form-
Closed(System::Object^ sender, System::Windows::Forms::FormClosedEventArgs^
e)
{
    //this->Parent->Show();
    return;
}

System::Void logschemtest::form_reg_ver_method::regMethod()
{
```



```

    HWND hWnd;
    int i1, j;

    modcx(hWnd);
    strcat(LpName, ".tst");
    //in_file = fopen(LpName, "wb");
    if ((in_file = fopen(LpName, "wb")) == NULL)
        //MessageBox(NULL, "Невозможно открыть файл с тестом", "Ошибка
", MB_OK | MB_ICONERROR); //файл с тестом
        myAlert("невозможно открыть файл с тестом");
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    i1 = 1 / 32 + n / 32;
    if ((j = 1 % 32) != 0) i1++;
    if ((j = n % 32) != 0) i1++;
    fwrite(tst, sizeof(int), tst[0] * i1 + 1, in_file);
    fclose(in_file);
    strcat(LpName, ".cff");
    //in_file = fopen(LpName, "wb");
    if ((in_file = fopen(LpName, "wb")) == NULL)
        //MessageBox(NULL, "Невозможно открыть файл .cff", "Ошибка ",
MB_OK | MB_ICONERROR); //файл с неисправностями
        myAlert("невозможно открыть файл .cff");
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    fwrite(&nhcfl, sizeof(int), 2, in_file);
    fwrite(cf, sizeof(int), 1 + ns * 4, in_file);
    printf();
    printtst();
}

```

```

System::Void logschemtest::form_reg_ver_method::verMethod()
{
    MPStruct m_MPStruct;
    m_MPStruct.M.SetSize(4);
    m_MPStruct.M.SetAt(0, Convert::ToInt32(textBoxM1->Text));
    m_MPStruct.M.SetAt(1, Convert::ToInt32(textBoxM1->Text));
    m_MPStruct.M.SetAt(2, Convert::ToInt32(textBoxM1->Text));
    //m_MPStruct.M.SetAt(3, WM_VERMETUPDATE);
    //m_MPStruct.hw = hWnd;

    if ((m_MPStruct.M.GetAt(0) <= 0) | (m_MPStruct.M.GetAt(2) <= 0) |
(m_MPStruct.M.GetAt(1) <= 0))
    {
        myAlert("неверно введены параметры M1, M2, M3");
        return;
    }
    vermet(&m_MPStruct);
    printf();
    printtst();
}

```

form_start.cpp

```

#include "pch.h"
#include "MyPrint.h"
//#include "MyRegmethod.h"
#include <cstring>
#include <iostream>
#include <fstream>

/*
#ifdef _DEBUG

```

```

#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
*/
#define HIGH_BIT 0x80000000

// #define INT_SIZE 32

char* appDir;
CString CSfilePath = "";
int flagFileOpen = 0;
char* LpName; // Путь и имя схемы
char Ident[20000];
int PointIdent[5000];
// CRecentFileList recFList(0, "Recent File List", "File%d", 4, 20);
char* errors[] = {
    "успешно конвертирован.", // 0
    "Ошибка открытия файла BIBLEL.", // 1
    "Ошибка открытия файла описания.", // 2
    "Ошибка открытия файла неисправностей.", // 3
    "Недостаточно памяти.", // 4
    "Ошибка открытия файла протокола.", // 5
    "Ошибка открытия файла идентификаторов линий схемы.", // 6
    "Файл неисправностей успешно сохранен.", // 7
    "Недопустимое значение на входе схемы.", // 8
    "Моделирование успешно завершено.", // 9
    "❖-акончил чтение файла", // 10
    "Ошибка открытия файла с тестом.", // 11
    "Тест успешно построен." // 12
    "", //13 Ошибка открытия файла протокола
    "", //14 Ошибка открытия файла .pnt
    "", //15
    "", //16
    "00001", //17 Ошибка открытия файла .inf
    "00002", //18
    "00003", //19
    "00004", //20
    "00005", //21
    "00006", //22
    "00007", //23
    "00008", //24
    "Ошибка открытия файла INF - выходного", // 25
    "Ошибка открытия файла CFF - выходного" // 26
};

//int il; //?
int* m11, * m12, * m2;
int* sm, * ml, * mn, * mh, * nvx, * svx, * nvix, * svix, * nvixc, * cf, *
nct, * ctn, * ncte, * ctne;
int ns; // кол-во элементов в схеме
int nl; // кол-во узлов в схеме
int l; // кол-во первичных входов
int n; // кол-во выходов и контр.точек
int nt = 0, nhcfl, nhcf, * tst;
static int* m11bib, * m12bib, * m2bib, * sbib;
static int* mlbib, * mnbib, * mhbib;

FILE* stream, * fptr, * fcf, * in_file;

static int el_n;
static int mt_n;

```

```

char* fa;

////////////////////////////////////
////////////////////////////////////

System::Void logschemtest::form_start::buttonOpen_Click(System::Object^
sender, System::EventArgs^ e)
{
    //richTextBox1->Text = System::Environment::CurrentDirectory; //путь
    appDir = new char[255]; // расположение bible1
    //strcpy(appDir, "C://Users//User//Desktop//dip_lukya-
novich//log_schem_test");
    strcpy(appDir, stringToChar(System::Environment::CurrentDirectory));

    int open_tsc_error; // i, j=0;
    CString message;
    LpName = new char[255];
    CString tscFile;
    openFileDialog1->InitialDirectory = charToString(appDir)+"\\Ex"; //
    старт директория
    openFileDialog1->Filter = "(*.TSC)|*.TSC";
    if (openFileDialog1->ShowDialog() == System::Windows::Forms::Di-
alogResult::OK) {
        String^ path = openFileDialog1->FileName;
        path=path->Remove(path->Length - 4, 4);
        tscFile = stringToChar(path);
    }

    /*
    //CString tscFile = "C://Users//User//Desktop//dip_lukya-
novich//log_schem_test//Ex//Ctrprm1//CTRPRM1";
    CString tscFile = appDir;
    tscFile += "//Ex//";
    tscFile += "Ctrprm1//CTRPRM1";
    */

    //tscFile += "shn2//SHN2";
    //strcpy(LpName, LPCSTR(tscFile->GetFilePath().Left(tscFile->Get-
FilePath().GetLength() - 4));
    strcpy(LpName, LPCSTR((CStringA)tscFile));
    DeleteFiles();
    open_tsc_error = Convertor();
    if (open_tsc_error == 0) {
        flagFileOpen = 1;
        ReadData();

        //message += "Файл " + tscFile->GetFileName() + " " + CString(er-
rors[0]);
        //MessageBox(NULL, message, "Открытие файла", MB_OK | MB_ICONINFOR-
MATION);

        message += "Файл \"" + tscFile + ".TSC\" " + CString(er-
rors[0]);
        myAlert((const char*)(CStringA)message);

        buttonStartMethodForm->Enabled = true;

        labelFileName->Text = "загружен файл: \"" + charTo-
String((char*)(const char*)(CStringA)tscFile) + "\"";
    }
    else {

```

```

        //MessageBox(NULL, errors[open_tsc_error], "Ошибка " +
CSfilePath, MB_OK | MB_ICONERROR);
        myAlert(errors[open_tsc_error]);
    }
}

System::Void logschemtest::form_start::buttonVerMethod_Click(System::Object^
sender, System::EventArgs^ e)
{
    logschemtest::form_reg_ver_method form2(checkBoxVerMethod->Checked);
    //this->Hide();
    form2.ShowDialog();
    //this->Show();
    buttonPrintProtocol->Enabled = true;
    buttonPrintTest->Enabled = true;
    buttonPrintUntesting->Enabled = true;
}

System::Void logschemtest::form_start::checkBoxVerMethod_CheckedChanged(System::Object^ sender, System::EventArgs^ e)
{
    checkBoxRegMethod->Checked = !checkBoxVerMethod->Checked;
}

System::Void logschemtest::form_start::checkBoxRegMethod_CheckedChanged(System::Object^ sender, System::EventArgs^ e)
{
    checkBoxVerMethod->Checked = !checkBoxRegMethod->Checked;
}

System::Void logschemtest::form_start::printResult(char* typeFile)
{
    char* resFilePath = new char[255];
    strcpy(resFilePath, lpName);
    strcat(resFilePath, typeFile);
    std::ifstream in(resFilePath);
    std::string line, res;
    while (getline(in, line)) { res += line + "\n"; }
    in.close();
    if (res.empty())
        myAlert("файл пуст");
    else
        richTextBox1->Text = charToString((char*)res.c_str());
}

System::Void logschemtest::form_start::buttonPrintUntesting_Click(System::Object^ sender, System::EventArgs^ e)
{
    //char* resFilePath = "C://Users//User//Desktop//dip_lukyanovich//log_schem_test//Ex//Ctrprml//CTRPRM1.fff";
    printResult(".fff");
}

System::Void logschemtest::form_start::buttonPrintProtocol_Click(System::Object^ sender, System::EventArgs^ e)
{
    //char* resFilePath = "C://Users//User//Desktop//dip_lukyanovich//log_schem_test//Ex//Ctrprml//CTRPRM1.pte";
    printResult(".pte");
}

```

```

System::Void logschemtest::form_start::buttonPrintTest_Click(System::Object^
sender, System::EventArgs^ e)
{
    //char* resFilePath = "C://Users//User//Desktop//dip_lukya-
novich//log_schem_test//Ex//Ctrprml//CTRPRM1.tsd";
    printResult(".tsd");
}

```

```

System::Void logschemtest::form_start::form_start_FormClosed(System::Object^
sender, System::Windows::Forms::FormClosedEventArgs^ e)
{
    DeleteFiles();
    return System::Void();
}

```

```

////////////////////////////////////
////////////////////////////////////

```

```

void DeleteFiles() {
    if (LpName != nullptr) {
        DeleteFile(LPCTSTR(CString(LpName) + ".tst"));
        DeleteFile(LPCTSTR(CString(LpName) + ".tsd"));
        DeleteFile(LPCTSTR(CString(LpName) + ".pte"));
        DeleteFile(LPCTSTR(CString(LpName) + ".fff"));
    }
}

```

```

int Convertor()
{
    int boom = 0;
    int* pel, * uzlel, * puzl;
    int swinf[4];
    int il;
    int i, k, i2, t, j, vs, sum = 0, sum1 = 0;
    unsigned short* tt, c14 = 0x7FFF;
    static int c11[32] = { 0x80000000,0xC0000000,0xE0000000,0xF0000000,
        0xF8000000,0xFC000000,0xFE000000,0xFF000000,
        0xFF800000,0xFFC00000,0xFFE00000,0xFFFF0000,
        0xFFFF8000,0xFFFFC000,0xFFFFE000,0xFFFFF000,
        0xFFFFF800,0xFFFFFC00,0xFFFFFE00,0xFFFFFF00,
        0xFFFFF800,0xFFFFFC00,0xFFFFFE00,0xFFFFFF00,
        0xFFFFF80,0xFFFFFC0,0xFFFFFE0,0xFFFFFF0,
        0xFFFFF8,0xFFFFFC,0xFFFFFE,0xFFFFF };

    mt_n = 0;

    if ((m11bib = new int[5000]) == NULL) return 4;
    if ((m12bib = new int[5000]) == NULL) return 4;
    if ((m2bib = new int[5000]) == NULL) return 4;
}

```

```

if ((sbib = new int[1000]) == NULL) return 4;
if ((mlbib = new int[1000]) == NULL) return 4;
if ((mnbib = new int[1000]) == NULL) return 4;
if ((mhbib = new int[1000]) == NULL) return 4;
if ((m11 = new int[9000]) == NULL) return 4;
if ((m12 = new int[9000]) == NULL) return 4;
if ((m2 = new int[9000]) == NULL) return 4;

// Обнуление элементов массивов
for (i = 0; i < 5000; i++) m11bib[i] = m12bib[i] = m2bib[i] = 0;
for (i = 0; i < 9000; i++) m11[i] = m12[i] = m2[i] = 0;
for (i = 0; i < 1000; i++) sbib[i] = mlbib[i] = mnbib[i] = mhbib[i] =
0;

// Открываем файл с библиотекой элементов
strcat(appDir, "//bible1");
if ((stream = fopen(appDir, "rt")) == NULL) return 1;
appDir[strlen(appDir) - 6] = 0x00;
do { ReadElement(stream); } while (!feof(stream));
fclose(stream);

strcat(LpName, ".tsc");
if ((in_file = fopen(LpName, "rb")) == NULL) return 2;
LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения

fread(swinf, sizeof(int), 4, in_file);
ns = (int)swinf[0]; // кол-во элементов в схеме
nl = (int)swinf[1]; // кол-во узлов в схеме
l = (int)swinf[2]; // кол-во первичных входов
n = (int)swinf[3]; // кол-во выходов и контр.точек
// пропускаем 4 числа блока swinf =16 байт
fseek(in_file, 16, SEEK_CUR);
// считываем TT размером ns (ns*2 байт)
if ((tt = new unsigned short[ns]) == NULL) return 4;
// for ( i=0; i<=ns-1; i++) tt[i]=0; // Очистка массива нулями
fread(tt, 2, ns, in_file);
for (i = 0; i < ns; i++)
    tt[i] &= c14;

// считываем Puzl размером ns+1 ((ns+1)*4 байт)
if ((puzl = new int[ns + 1]) == NULL) return 4;
// for ( i=0; i<=ns; i++) puzl[i]=0; // Очистка массива нулями
fread(puzl, 4, ns + 1, in_file);
if ((uzlel = new int[puzl[ns]]) == NULL) return 4;
// for ( i=0; i<=puzl[ns]-1; i++) uzlel[i]=0; // Очистка массива
нулями
fread(uzlel, 4, puzl[ns], in_file);
for (i = 0; i < puzl[ns]; i++) uzlel[i]--;

// пропускаем блок pel до последнего элемента
fseek(in_file, nl * 4, SEEK_CUR);
if ((pel = new int[4]) == NULL) return 4;
// for ( i=0; i<=3; i++) pel[i]=0; // Очистка массива нулями
fread(pel, 4, 1, in_file);
fseek(in_file, (*pel) * 4, SEEK_CUR);

// считываем Nviuzl размером n
if ((nvixc = new int[n]) == NULL) return 4;
fread(nvixc, 4, n, in_file);
for (i = 0; i < n; i++) nvixc[i]--;

```

```

fclose(in_file);
// ФОРМАТО 'АН'Е МАСС' 'О' "ЛЯ ЭЛЕМЕНТО' 1,2,3,4,5
if ((sm = new int[ns]) == NULL) return 4;
if ((ml = new int[ns]) == NULL) return 4;
if ((mn = new int[ns]) == NULL) return 4;
if ((mh = new int[ns]) == NULL) return 4;
for (i = 0; i < ns; i++) sm[i] = ml[i] = mn[i] = mh[i] = 0;
// формирование массивов m11,m12,m2,ml,mn,mh,sm
i1 = 0;
for (i = 0; i < ns; i++)
{
    switch (tt[i])
    {
    case 1: m11[i1] = 0;
            ml[i] = puzl[i + 1] - puzl[i] - 1;
            for (j = 0; j <= ml[i] - 1; j++)
                m11[i1] += Pow2(INT_SIZE - 1 - j);
            m12[i1] = 0;
            m2[i1] = HIGH_BIT;
            sm[i] = i1;
            mh[i] = 1;
            mn[i] = 1;
            i1++;
            break;
    case 2: ml[i] = puzl[i + 1] - puzl[i] - 1;
            sm[i] = i1;
            for (j = 0; j <= ml[i] - 1; j++)
            {
                m11[i1] = Pow2(INT_SIZE - 1 - j);
                m2[i1] = HIGH_BIT;
                m12[i1] = 0;
                i1++;
            }
            mn[i] = 1;
            mh[i] = ml[i];
            break;
    case 3:
    case 9:
    case 10:
    case 11:
    case 12:
    case 13:
    case 14:
            ml[i] = 1;
            sm[i] = i1;
            m11[i1] = 0;
            m12[i1] = HIGH_BIT;
            m2[i1] = HIGH_BIT;
            mn[i] = 1;
            mh[i] = 1;
            i1++;
            break;
    case 4:
    case 6:
    case 7:
    case 8:
    case 16:
    case 17:
    case 25:
    case 26:
            ml[i] = puzl[i + 1] - puzl[i] - 1;

```

```

sm[i] = i1;
for (j = 0; j <= ml[i] - 1; j++)
{
    m11[i1] = 0;
    m12[i1] = Pow2(INT_SIZE - 1 - j);
    m2[i1] = HIGH_BIT;
    i1++;
}
mn[i] = 1;
mh[i] = ml[i];
break;
case 5:
case 18:
case 27:
case 28:
case 30:
    m12[i1] = 0;
    ml[i] = puzl[i + 1] - puzl[i] - 1;
    sm[i] = i1;
    for (j = 0; j < ml[i]; j++)
        m12[i1] += Pow2(INT_SIZE - 1 - j);
    m11[i1] = 0;
    m2[i1] = HIGH_BIT;
    mh[i] = 1;
    mn[i] = 1;
    i1++;
    break;
// "ЛЯ ЭЛЕМЕНТО" - "??- "??"'Л?ОТЕК?
default:
    t = sbib[tt[i]];
    sm[i] = i1;
    for (j = 0; j < mhbib[tt[i]]; j++)
    {
        m11[i1] = m11bib[t];
        m12[i1] = m12bib[t];
        m2[i1] = m2bib[t];
        t++; i1++;
    }
    ml[i] = mlbib[tt[i]];
    mh[i] = mhbib[tt[i]];
    mn[i] = mnbib[tt[i]];
} // switch
}; // for
// Резервирование памяти nvx,nvix,svx,svix...
for (i = 0; i < ns; i++) sum += mn[i];
for (i = 0; i < ns; i++) sum1 += ml[i];
if ((nvix = new int[sum]) == NULL) return 4;
if ((nvx = new int[sum1]) == NULL) return 4;
if ((svx = new int[ns]) == NULL) return 4;
if ((svix = new int[ns]) == NULL) return 4;
// Обнуление элементов массива
for (i = 0; i < sum; i++) nvix[i] = 0;
for (i = 0; i < sum1; i++) nvx[i] = 0;
for (i = 0; i < ns; i++) svix[i] = svx[i] = 0;
// Формирование svx- секционир. для nvx
svx[0] = 0;
for (i = 1; i < ns; i++) svx[i] = svx[i - 1] + ml[i - 1];
// Формирование массива svix- секционир. для nvix
i1 = 0;
for (i = 0; i < ns; i++)
{
    svix[i] = i1;
    i1 += mn[i]; // для многовыданных элементов

```



```

}
// формирование массива nvx-входы элементов nvix-выходы элементов
j = t = i1 = vs = 0;
for (i = 0; i < ns; i++)
{
    for (t = 0; t < ml[i]; t++)    nvx[i1++] = uzlel[j++];
    for (t = 0; t < mn[i]; t++)    nvix[vs++] = uzlel[j++];
}
// ?'ыполнение правильной нумерации элементов схемы
sum = 0;
for (i = 0; i < ns; i++) sum += mn[i];
if ((nct = new int[sum + 1 + 1]) == NULL) return 4;
if ((ctn = new int[sum + 1 + 1]) == NULL) return 4;
if ((ncte = new int[ns + 1]) == NULL)    return 4;
if ((ctne = new int[ns + 1]) == NULL)    return 4;
//Обнуление элементов массива
for (i = 0; i <= sum + 1; i++) nct[i] = ctn[i] = 0;
for (i = 0; i <= ns; i++) ctne[i] = ncte[i] = 0;
for (i = 1; i <= 1; i++) nct[i] = ctn[i] = i;
for (i = 1 + 1; i <= (1 + sum); i++) ctn[i] = 0;
i1 = 0; i2 = 1;
while (i1 < ns)
{
    for (i = 0; i < ns; i++)
    {
        t = 0;
        if (ctne[i + 1] == 0)
        {
            t = 0;
            for (k = 0; k < ml[i]; k++)
            {
                if (ctn[nvx[svx[i] + k]] == 0)
                {
                    t = 1;
                    break;
                }
            }
            if (t == 0)
            {
                i1++;
                ctne[i + 1] = i1;
                ncte[i1] = i + 1;
                for (j = 0; j < mn[i]; j++)
                {
                    if (ctn[nvix[svix[i] + j]] == 0)
                    {
                        i2++;
                        ctn[nvix[svix[i] + j]] =
i2;
                        nct[i2] = nvix[svix[i] +
j];
                    }
                }
            }
        }
    }
}
// формирование массива неисправностей cf
if ((cf = new int[1 + ns * 4]) == NULL)    return 4;
nhcf = 0;

for (i = 0; i < 1; i++) cf[i] = c11[1];
nhcf = 2 * 1; //для подсчета к-ва неисправностей

```

```

for (i = 0; i < ns; i++)
{
    cf[l + i * 4] = cf[l + i * 4 + 1] = c11[ml[ncte[i + 1] - 1] -
1];

    //                cf[l+i*4]=cf[l+i*4+1]=0;
    cf[l + i * 4 + 2] = cf[l + i * 4 + 3] = c11[mn[ncte[i + 1] -
1] - 1];

    nhcf += mn[ncte[i + 1] - 1] * 2; // подсчет к-ва неисправностей
    nhcf += ml[ncte[i + 1] - 1] * 2;
}
// запись выходных данных в файл
strcat(LpName, ".inf");
if ((fptr = fopen(LpName, "w")) == NULL) return 25;
LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
fprintf(fptr, "%3d\n", 1);
fprintf(fptr, "%5d%5d\n", n, ns);
i1 = 0;
for (i = 0; i < ns; i++)
{
    fprintf(fptr, "%4d\n", i1);
    i1 += mh[ncte[i + 1] - 1];
}
for (i = 0; i < ns; i++) fprintf(fptr, "%2d\n", ml[ncte[i + 1] - 1]);
for (i = 0; i < ns; i++) fprintf(fptr, "%2d\n", mn[ncte[i + 1] - 1]);
for (i = 0; i < ns; i++) fprintf(fptr, "%2d\n", mh[ncte[i + 1] - 1]);
for (i = 0; i < ns; i++)
{
    i1 = sm[ncte[i + 1] - 1];
    for (j = 0; j < mh[ncte[i + 1] - 1]; j++)
        fprintf(fptr, "%8x\n", m1[i1++]);
}
for (i = 0; i < ns; i++)
{
    i1 = sm[ncte[i + 1] - 1];
    for (j = 0; j < mh[ncte[i + 1] - 1]; j++)
        fprintf(fptr, "%8x\n", m12[i1++]);
}
for (i = 0; i < ns; i++)
{
    i1 = sm[ncte[i + 1] - 1];
    for (j = 0; j < mh[ncte[i + 1] - 1]; j++)
        fprintf(fptr, "%8x\n", m2[i1++]);
}
for (i = 0; i < ns; i++)
{
    i1 = svx[ncte[i + 1] - 1];
    for (j = 0; j < ml[ncte[i + 1] - 1]; j++)
        fprintf(fptr, "%5d\n", ctn[nvx[i1++]]);
}
i1 = 0;

for (i = 0; i < ns; i++)
{
    fprintf(fptr, "%4d\n", i1);
    i1 += ml[ncte[i + 1] - 1];
}
for (i = 0; i < ns; i++)
{
    i1 = svix[ncte[i + 1] - 1];
    for (j = 0; j < mn[ncte[i + 1] - 1]; j++)
        fprintf(fptr, "%5d\n", ctn[nvix[i1++]]);
}

```

```

i1 = 0;

for (i = 0; i < ns; i++)
{
    fprintf(fp, "%4d\n", i1);
    i1 += mn[ncte[i + 1] - 1];
}
for (i = 0; i < n; i++)    fprintf(fp, "%5d\n", ctn[nvixc[i]]);
for (i = 0; i < (l + sum); i++) fprintf(fp, "%5d\n", nct[i + 1]);
for (i = 0; i < (l + sum); i++) fprintf(fp, "%5d\n", ctn[i + 1]);
for (i = 0; i < ns; i++)    fprintf(fp, "%5d\n", ncte[i + 1]);
for (i = 0; i < ns; i++)    fprintf(fp, "%5d\n", ctne[i + 1]);
fclose(fp);
//запись файла неисправностей
strcat(LpName, ".cff");
if ((fcf = fopen(LpName, "wb")) == NULL) return 26;
LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
fwrite(&nhcf, sizeof(int), 1, fcf);
fwrite(cf, sizeof(int), l + ns * 4, fcf);
fclose(fcf);
delete[] puzl;
delete[] tt;
delete[] pel;
delete[] uzlel;
delete[] m1lbib;
delete[] m12bib;
delete[] m2bib;
delete[] sbib;
delete[] mhbib;
delete[] mnbib;
delete[] mlbib;
return 0;
}

////////////////////////////////////

int Pow2(int st)
{
    return (st) ? 2 * Pow2(st - 1) : 1;
}

////////////////////////////////////

void ReadElement(FILE* in_file)
{
    int i, j;
    char aCh[33];
    int n_el, n_in, n_out, n_h;
    int c12[32] = { 0x80000000, 0x40000000, 0x20000000, 0x10000000,
                    0x8000000, 0x4000000, 0x2000000, 0x1000000,
                    0x800000, 0x400000, 0x200000, 0x100000,
                    0x80000, 0x40000, 0x20000, 0x10000,
                    0x8000, 0x4000, 0x2000, 0x1000,
                    0x800, 0x400, 0x200, 0x100,
                    0x80, 0x40, 0x20, 0x10,
                    0x8, 0x4, 0x2, 0x1 };
    if (fscanf(in_file, "%d%d%d", &n_el, &n_in, &n_out) == EOF) return;
    if (fscanf(in_file, "%d", &n_h) == EOF) return;
    sbib[n_el] = mt_n;
    mlbib[n_el] = n_in;
    mnbib[n_el] = n_out;
    mhbib[n_el] = n_h;
}

```

```

n_el = mt_n;
////////////////////////////////////
for (i = 0; i < n_h; i++)
{
    if (fscanf(in_file, "%s", aCh) == EOF) return;
    for (j = 0; j < n_in; j++)
    {
        switch (aCh[j])
        {
            case '1': m11bib[mt_n] |= c12[j];
                      break;
            case '0': m12bib[mt_n] |= c12[j];
                      break;
        }
    }
    mt_n++;
}
////////////////////////////////////
for (i = 0; i < n_h; i++)
{
    if (fscanf(in_file, "%s", aCh) == EOF) return;
    for (j = 0; j < n_out; j++)
    {
        if (aCh[j] == '1')
            m2bib[n_el] |= c12[j];
    }
    n_el++;
}
}

void ClearArrays() {
    extern int nn, nh;
    delete[] sm;
    delete[] mn;
    delete[] mh;
    delete[] ml;
    delete[] nvx;
    delete[] svx;
    delete[] svix;
    delete[] nvixc;
    delete[] cf;
    delete[] m11;
    delete[] m12;
    delete[] m2;
    delete[] ctn;
    delete[] nct;
    delete[] ncte;
    delete[] ctne;
    delete[] tst;
    nhcf = 0;
    nhcf1 = 0;
    nn = 0;
    nh = 0;
}

int ReadData()
{
    int alt;
    int j, i;
    int il;
    FILE* fptr;

```

```

strcat(LpName, ".inf");
if ((fptr = fopen(LpName, "r")) == NULL) return 1;
LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
fscanf(fptr, "%3d", &l);
fscanf(fptr, "%5d%5d", &n, &ns);

for (i = 0; i < ns; i++)
{
    fscanf(fptr, "%4d", &sm[i]);
}

for (i = 0; i < ns; i++)
{
    fscanf(fptr, "%2d", &ml[i]);
}

for (i = 0; i < ns; i++)
{
    fscanf(fptr, "%2d", &mn[i]);
}

if ((tst = new int[6000]) == NULL) return 1;
for (i = 0; i < 6000; i++) tst[i] = 0;
for (i = 0; i < ns; i++)
{
    fscanf(fptr, "%2d", &mh[i]);
}

alt = 0;
for (i = 0; i < ns; i++)
    alt += mh[i];
for (i1 = 0, i = 0; i < ns; i++)
    for (j = 0; j < mh[i]; j++)
    {
        fscanf(fptr, "%8x", &m11[i1]);
        i1++;
    }

for (i1 = 0, i = 0; i < ns; i++)
    for (j = 0; j < mh[i]; j++)
    {
        fscanf(fptr, "%8x", &m12[i1]);
        i1++;
    }

for (i1 = 0, i = 0; i < ns; i++)
    for (j = 0; j < mh[i]; j++, i1++)
    {
        fscanf(fptr, "%8x", &m2[i1]);
    }

i1 = 0;
for (i = 0; i < ns; i++)
    i1 += ml[i];

for (i1 = 0, i = 0; i < ns; i++)
    for (j = 0; j < ml[i]; j++, i1++)
    {
        fscanf(fptr, "%5d", &nvx[i1]);
    }

for (i = 0; i < ns; i++)
{

```

```

        fscanf(fptr, "%4d", &svx[i]);
    }

    i1 = 0;
    for (i = 0; i < ns; i++)
        i1 += mn[i];
    for (i = i1 = 0; i < ns; i++)
        for (j = 0; j < mn[i]; j++, i1++)
        {
            fscanf(fptr, "%5d", &nvix[i1]);
        }

    for (i = 0; i < ns; i++)
    {
        fscanf(fptr, "%4d", &svix[i]);
    }

    for (i = 0; i < n; i++)
    {
        fscanf(fptr, "%5d", &nvixc[i]);
    }

    i1 = l + 1;
    for (i = 0; i < ns; i++)
        i1 += mn[i];
    for (i = 0; i < i1; i++)
        nct[i] = 0;
    for (i = 0; i < (i1 - 1); i++)
        fscanf(fptr, "%5d", &nct[i + 1]);

    for (i = 0; i < (i1 - 1); i++)
        fscanf(fptr, "%5d", &ctn[i + 1]);

    for (i = 0; i < ns; i++)
        fscanf(fptr, "%5d", &ncte[i + 1]);

    for (i = 0; i < ns; i++)
        fscanf(fptr, "%5d", &ctne[i + 1]);

    fclose(fptr);

    for (i = 0; i < 6000; i++)
        tst[i] = 0;

    strcat(LpName, ".tst");
    if ((fptr = fopen(LpName, "rb")) == NULL)
    {
        tst[0] = nt = 0;
    }
    else
    {
        i1 = l / 32 + n / 32;
        if ((j = l % 32) != 0) i1++;
        if ((j = n % 32) != 0) i1++;
        fread(tst, sizeof(int), 1, fptr);
        nt = tst[0];
        fread(&tst[1], sizeof(int), nt * i1, fptr);
        fclose(fptr);
    }

    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    // 'вод списка неисправностей cf
    strcat(LpName, ".cff");

```

```

        if ((fptr = fopen(LpName, "rb")) == NULL) return 1; //Невозможно от-
крыть файл со списком неисправностей
        fread(&nhcf, sizeof(int), 1, fptr);
        fread(cf, sizeof(int), 1 + ns * 4, fptr);
        fclose(fptr);
        LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
        nhcf1 = nhcf;
        // 'вод идентификаторов линий схемы
        strcat(LpName, ".pnt");
        if ((fptr = fopen(LpName, "rb")) == NULL) return 1; //Невозможно от-
крыть файл типа .pnt
        i1 = 1 + 3;
        for (i = 0; i < ns; i++)
            i1 += mn[i];
        fread(PointIdent, 4, i1, fptr);
        fclose(fptr);
        LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
        strcat(LpName, ".idn");
        if ((fptr = fopen(LpName, "rb")) == NULL) return 1; //Невозможно
открыть файл типа .idn
        fread(Ident, sizeof(char), PointIdent[i1 - 1], fptr);
        fclose(fptr);
        LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
        return 0;
    }

void SaveData()
{
    int i1, j;
    strcat(LpName, ".tst");
    if ((in_file = fopen(LpName, "wb")) == NULL)
        //MessageBox(NULL, "Невозможно открыть файл с тестом", "Ошибка
", MB_OK | MB_ICONERROR); //файл с тестом
        myAlert("невозможно открыть файл с тестом");
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    i1 = 1 / 32 + n / 32;
    if ((j = 1 % 32) != 0) i1++;
    if ((j = n % 32) != 0) i1++;
    fwrite(tst, sizeof(int), tst[0] * i1 + 1, in_file);
    fclose(in_file);
    strcat(LpName, ".cff");
    if ((in_file = fopen(LpName, "wb")) == NULL)
        //MessageBox(NULL, "Невозможно открыть файл .cff", "Ошибка ",
MB_OK | MB_ICONERROR); //файл с неисправностями
        myAlert("невозможно открыть файл .cff");
    LpName[strlen(LpName) - 4] = 0x00; // Удаление расширения
    fwrite(&nhcf1, sizeof(int), 2, in_file);
    fwrite(cf, sizeof(int), 1 + ns * 4, in_file);
    printf();
    printtst();
}

```

Header.cpp

```
#include "pch.h"
```

```

System::String^ charToString(char* str) {
    const char* temp = str;
    System::String^ s = gcnew System::String(temp);
    return s;
}

```

```

}

char* stringToChar(System::String^ s) {
    return (char*) (void*) System::Runtime::InteropServices::Mar-
shal::StringToHGlobalAnsi(s);
}

char* myCaption = "Внимание!";

void myAlert(char* msg)
{
    System::String^ message = charToString(msg);
    System::Windows::Forms::MessageBox::Show(message, charToString(myCap-
tion));
}

void myAlert(const char* msg)
{
    myAlert((char*)msg);
}

/*
void myAlert(CStringA msg)
{
    System::String^ message = charToString((char*)(const char*)msg);
    System::Windows::Forms::MessageBox::Show(message, charToString(myCap-
tion));
}
*/

void myAlert(System::String^ msg)
{
    System::String^ message = gcnew System::String(msg);
    System::Windows::Forms::MessageBox::Show(message, charToString(myCap-
tion));
}

```

log_schem_test.cpp

```

#include "pch.h"

using namespace System;
using namespace System::Windows::Forms;

int main(array<System::String ^> ^args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    logschemtest::form_start start;
    Application::Run(% start);

    return 0;
}

```



```
pch.cpp
// pch.cpp: source file corresponding to the pre-compiled header

#include "pch.h"

// When you are using pre-compiled headers, this source file is necessary for
// compilation to succeed.
```

ПРИЛОЖЕНИЕ Б

Выполнение тестирования на контрольном примере
схема имеет следующий вид:

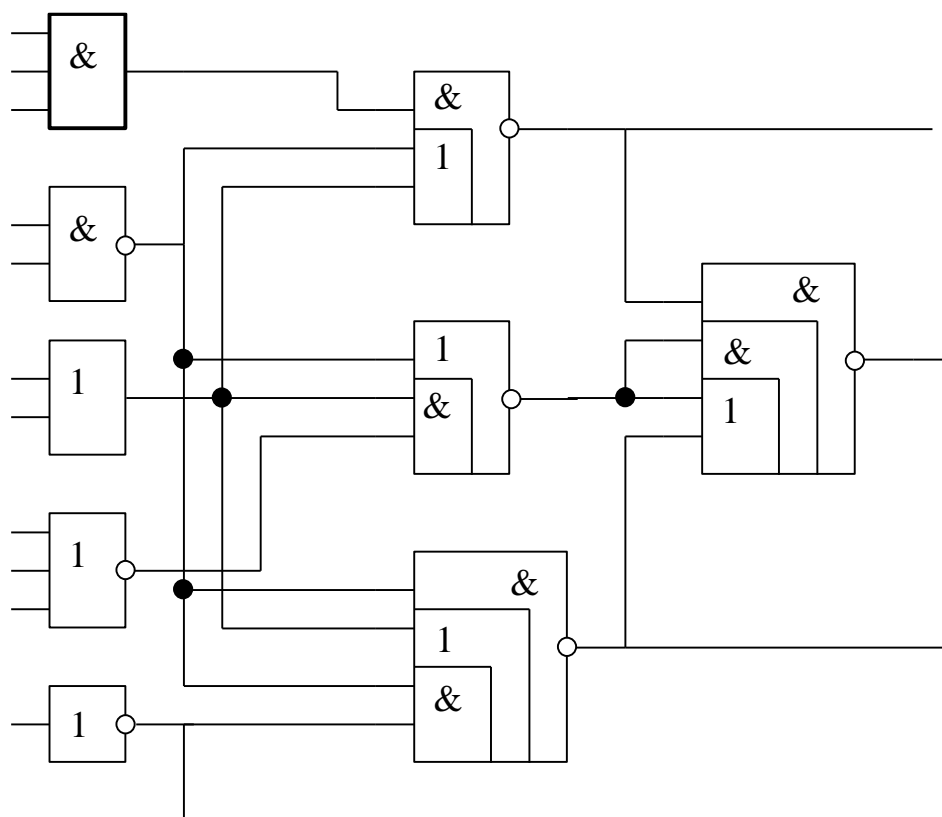
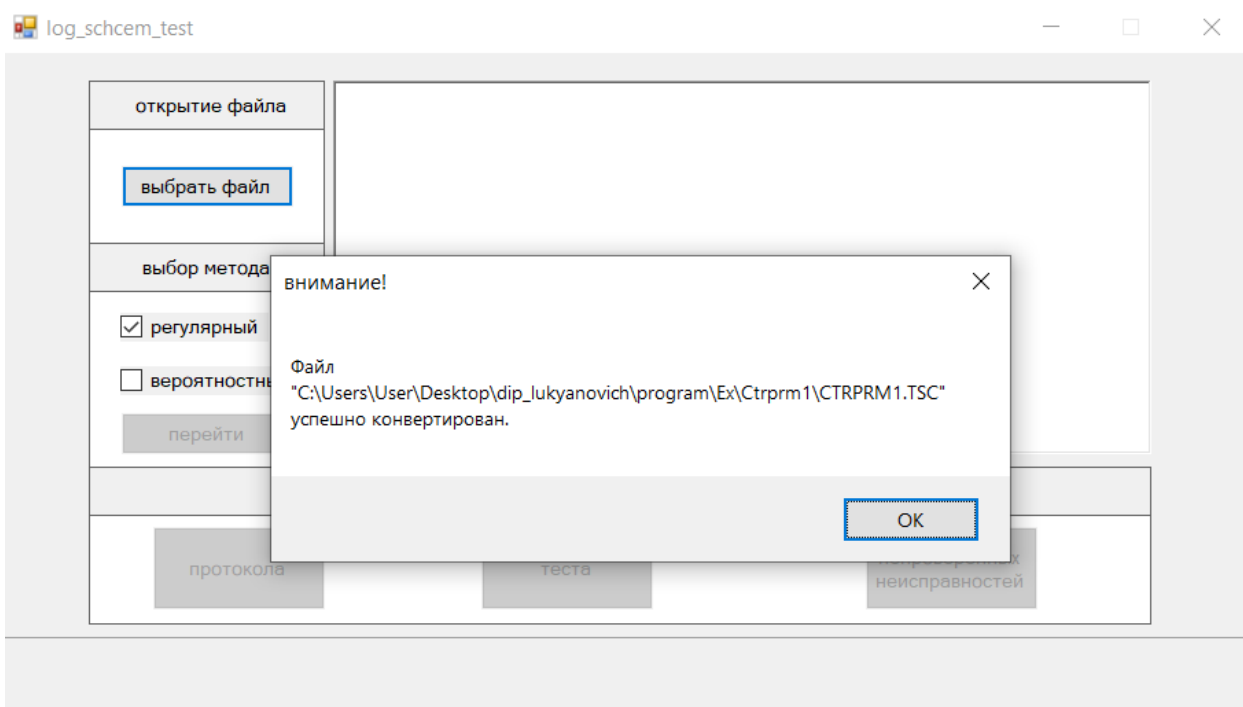


Схема ctrprm1




Генерация теста регулярным методом

M1:

M2:

M3:

Полнота теста 84%, время 0.039 сек

0%  100%

Входная последовательность обнаруживает 76 неисправностей
В тест включено 10 наборов

полученные результаты

Протокол:

РЕЗУЛЬТАТЫ ПОСТРОЕНИЯ ТЕСТА ДЛЯ СХЕМЫ
РЕГУЛЯРНЫМ МЕТОДОМ

I= 11, n= 4, ns= 9

Неисправность на 1 входе

Тест

входной набор: 11100000000

выходная реакция: 0101

входной набор обн. 19 неисправностей

входная последовательность обн. 19 неисправностей

входной набор: 01101111111

выходная реакция: 1100

входной набор обн. 14 неисправностей

входная последовательность обн. 33 неисправностей

Неисправность на 2 входе

Тест

входной набор: 11100000000

выходная реакция: 0101

входной набор обн. 0 неисправностей

входная последовательность обн. 33 неисправностей

входной набор: 10101111111

выходная реакция: 1100

входной набор обн. 2 неисправностей

входная последовательность обн. 35 неисправностей

Неисправность на 3 входе

Тест

входной набор: 11100000000

выходная реакция: 0101

входной набор обн. 0 неисправностей

входная последовательность обн. 35 неисправностей

входной набор: 11001111111

выходная реакция: 1100

входной набор обн. 2 неисправностей
выходная последовательность обн. 37 неисправностей

Неисправность на 4 входе

Тест

входной набор: 11111000000

выходная реакция: 1011

входной набор обн. 19 неисправностей

выходная последовательность обн. 56 неисправностей

входной набор: 11101001111

выходная реакция: 0110

входной набор обн. 2 неисправностей

выходная последовательность обн. 58 неисправностей

Неисправность на 5 входе

Тест

входной набор: 11111000000

выходная реакция: 1011

входной набор обн. 0 неисправностей

выходная последовательность обн. 58 неисправностей

входной набор: 11110001111

выходная реакция: 0110

входной набор обн. 2 неисправностей

выходная последовательность обн. 60 неисправностей

Неисправность на 6 входе

Тест

входной набор: 11111100000

выходная реакция: 0111

входной набор обн. 4 неисправностей

выходная последовательность обн. 64 неисправностей

входной набор: 11111001111

выходная реакция: 1010

входной набор обн. 0 неисправностей

выходная последовательность обн. 64 неисправностей

Неисправность на 7 входе

Тест

входной набор: 11111010000

выходная реакция: 0111

входной набор обн. 2 неисправностей

выходная последовательность обн. 66 неисправностей

входной набор: 11111001111

выходная реакция: 1010

входной набор обн. 0 неисправностей

выходная последовательность обн. 66 неисправностей

Неисправность на 8 входе

Необнаружимая неисправность

Неисправность на 9 входе

Необнаружимая неисправность

Неисправность на 10 входе

Необнаружимая неисправность

Связь 17, неисправность 1;

Тест

входной набор: 00011100000

выходная реакция: 1111

входной набор обн. 10 неисправностей

выходная последовательность обн. 76 неисправностей

Связь 17, неисправность 2;
Необнаружимая неисправность
Связь 17, неисправность 2;
Необнаружимая неисправность
Тест построен
входной набор: 10010100000
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 76 неисправностей

входной набор: 11000100000
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 76 неисправностей

входной набор: 01111101000
выходная реакция: 1011
входной набор обн. 4 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 00100001111
выходная реакция: 1110
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 10000111010
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 00110110101
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 01101011001
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 01110101010
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 01001010100
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 00001001111
выходная реакция: 1110
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 10010010011
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 10001111101
выходная реакция: 1100

входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

входной набор: 10001111010
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 80 неисправностей

Тест:

ПРОВЕРЯЮЩИЙ ТЕСТ
ВХОДЫ ВЫХОДЫ
1) 11100000000 0101
2) 01101111111 1100
3) 10101111111 1100
4) 11001111111 1100
5) 11111000000 1011
6) 11101001111 0110
7) 11110001111 0110
8) 11111100000 0111
9) 11111010000 0111
10) 00011100000 1111
11) 01111101000 1011

Список неисправностей:

СПИСОК НЕПРОВЕРЕННЫХ НЕИСПРАВНОСТЕЙ


В списке 10 неисправностей
A(9) 0 A(10) 0
A6:A(9) 0 A6:A(10) 0
A4:A8(1) 1
A1:A2(1) 1 A1:A3(1) 0 A1:A3(1) 1 A1:A4(1) 0 A1:A4(1) 1

Результаты тесты для вероятностного метода:

Генерация теста вероятностным мето...

M1: 15
M2: 20
M3: 22

Полнота теста 92%, время 0.043 сек

0%  100%

Входная последовательность обнаруживает 83 неисправностей
В тест включено 14 наборов
Сгенерировано 48 входных наборов

старт

Протокол:

входной набор: 10010100000
выходная реакция: 1101
входной набор обн. 13 неисправностей
входная последовательность обн. 13 неисправностей

входной набор: 11000100000
выходная реакция: 1101
входной набор обн. 2 неисправностей
входная последовательность обн. 15 неисправностей

входной набор: 01111101000
выходная реакция: 1011
входной набор обн. 18 неисправностей
входная последовательность обн. 33 неисправностей

входной набор: 00100001111
выходная реакция: 1110
входной набор обн. 12 неисправностей
входная последовательность обн. 45 неисправностей

входной набор: 10000111010
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 45 неисправностей

входной набор: 00110110101
выходная реакция: 1100
входной набор обн. 2 неисправностей
входная последовательность обн. 47 неисправностей

входной набор: 01101011001
выходная реакция: 1100
входной набор обн. 4 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 01110101010
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 01001010100
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 00001001111
выходная реакция: 1110
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 10010010011
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 10001111101
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 10001111010

выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 11011101100
выходная реакция: 1011
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 10010111011
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 11010111100
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 11001101110
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 51 неисправностей

входной набор: 01100101011
выходная реакция: 1100
входной набор обн. 2 неисправностей
входная последовательность обн. 53 неисправностей

входной набор: 11011011010
выходная реакция: 1011
входной набор обн. 0 неисправностей
входная последовательность обн. 53 неисправностей

входной набор: 00111100101
выходная реакция: 1010
входной набор обн. 2 неисправностей
входная последовательность обн. 55 неисправностей

входной набор: 11100001011
выходная реакция: 0110
входной набор обн. 10 неисправностей
входная последовательность обн. 65 неисправностей

входной набор: 00110000100
выходная реакция: 1101
входной набор обн. 2 неисправностей
входная последовательность обн. 67 неисправностей

входной набор: 01111100111
выходная реакция: 1010
входной набор обн. 0 неисправностей
входная последовательность обн. 67 неисправностей

входной набор: 10011001000
выходная реакция: 1011
входной набор обн. 0 неисправностей
входная последовательность обн. 67 неисправностей

входной набор: 00100100100
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 67 неисправностей

входной набор обн. 0 неисправностей
входная последовательность обн. 78 неисправностей

входной набор: 11010001010
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 78 неисправностей

входной набор: 01100101011
выходная реакция: 1100
входной набор обн. 0 неисправностей
входная последовательность обн. 78 неисправностей

входной набор: 11111000000
выходная реакция: 1011
входной набор обн. 3 неисправностей
входная последовательность обн. 81 неисправностей

входной набор: 11000000101
выходная реакция: 1110
входной набор обн. 0 неисправностей
входная последовательность обн. 81 неисправностей

входной набор: 01011010010
выходная реакция: 1011
входной набор обн. 2 неисправностей
входная последовательность обн. 83 неисправностей

входной набор: 10111110011
выходная реакция: 1010
входной набор обн. 0 неисправностей
входная последовательность обн. 83 неисправностей

входной набор: 10010000101
выходная реакция: 1110
входной набор обн. 0 неисправностей
входная последовательность обн. 83 неисправностей

входной набор: 00011100010
выходная реакция: 1011
входной набор обн. 0 неисправностей
входная последовательность обн. 83 неисправностей

входной набор: 10100100110
выходная реакция: 1101
входной набор обн. 0 неисправностей
входная последовательность обн. 83 неисправностей

входной набор: 11111000011
выходная реакция: 1010
входной набор обн. 0 неисправностей
входная последовательность обн. 83 неисправностей

Тест:

ПРОВЕРЯЮЩИЙ ТЕСТ
ВХОДЫ ВЫХОДЫ

- 1) 10010100000 1101
- 2) 11000100000 1101
- 3) 01111101000 1011
- 4) 00100001111 1110
- 5) 00110110101 1100
- 6) 01101011001 1100

- 7) 01100101011 1100
- 8) 00111100101 1010
- 9) 11100001011 0110
- 10) 00110000100 1101
- 11) 01111010000 1111
- 12) 10110000001 1110
- 13) 11111000000 1011
- 14) 01011010010 1011

Непроверенные неисправности:

СПИСОК НЕПРОВЕРЕННЫХ НЕИСПРАВНОСТЕЙ

В списке 7 неисправностей

A2:A7(1) 0

A4:A8(1) 1

A1:A2(1) 1A1:A3(1) 0A1:A3(1) 1A1:A4(1) 0A1:A4(1) 1