М.Л. Громов, С.А. Прокопенко, А.В. Лапутенко

СИНТЕЗ И ОПТИМИЗАЦИЯ ЦИФРОВЫХ СХЕМ

Часть 2. Работа с системой ABC

Учебно-методическое пособие

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

М.Л. Громов, С.А. Прокопенко, А.В. Лапутенко

СИНТЕЗ И ОПТИМИЗАЦИЯ ЦИФРОВЫХ СХЕМ

Часть 2. Работа с системой ABC

Учебно-методическое пособие
по курсу «Дискретная математика»
для студентов радиофизического факультета
направления подкотовки 03.03.03 – Радиофизика

Томск
Издательский Дом Томского государственного университета
2018

РАССМОТРЕНО И УТВЕРЖДЕНО методической комиссией радиофизического факультета
Протокол № 3/18 от «20» марта 2018 г.
Председатель МК РФФ А.П. Коханенко

УДК 519.1 (075.8)
ББК 22.174я73
Г87

Громов М.Л., Прокопенко С.А., Лапутенко А.В.
Г87 Синтез и оптимизация цифровых схем. Часть 2. Работа с системой ABC : учебно-методическое пособие. – Томск : Издательский Дом Томского государственного университета, 2018. – 18 с.

В настоящем учебно-методическом пособии даются базовые понятия теории цифровых схем, описываются две основные модели цифровых схем (логическая схема и система булевых функций), описываются форматы представления цифровых схем. Описывается работа с системой ABC, которая позволяет делать основные манимпуляции со схемами, в том числе и оптимизировать их, и проверять на эквивалентность. Каждый раздел пособия содержит задания для самостоятельного решения.

Для студентов кафедры информационных технологий в исследовании дискретных структур (ИТИДиС) РФФ ТГУ, изучающих курс «Дискретная математика», а также для всех интересующихся данной темой.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цифровое устройство – это устройство, которое обрабатывает (принимает и выдаёт) цифровые сигналы. Цифровым сигналом принято называть сигнал, уровень которого может принимать некоторые фиксированные значения (например, 0 В и 3,3 В). Число таких уровней также фиксировано. Чаще всего число уровней выбирается равным двум (как в примере выше) и тогда речь идёт о двоичных цифровых сигналах и устройствах. Далее мы будем рассматривать только двоичные цифровые сигналы и устройства.

Как и любые другие системы, цифровые устройства можно рассматривать на различных уровнях абстракции. Например, на физическом уровне (и тогда речь идёт о полупроводниках, проводниках и электрических полях), на компонентном уровне (транзисторы, токи, напряжение), на уровне цифровой схемы (цифровые компоненты, логические сигналы, временные диаграммы), на уровне логики (логические элементы, элементы памяти, булевы векторы) и т.д. На каждом из уровней, во-первых, используется свой формализм описания (например, на физическом уровне удобным формализмом являются системы дифференциальных уравнений), а во-вторых, можно проверять свои, специфические, параметры устройства (например, на компонентном уровне можно проверить, не превышает ли напряжение в некоторой точке допустимого значения). В данном методическом пособии мы будем рассматривать только уровень цифровой схемы и в большей степени логический уровень.

Для описания цифровой схемы совершенно не важно, какое конкретное физическое значение (0 В, 3,3 В или какое-то другое фиксированное значение) имеет тот или иной цифровой сигнал, важно только то, что уровни отличаются друг от друга и должны обрабатываться своим определённым образом. Поэтому каждому из уровней приписывают некоторое удобное для обращения значение и говорят о логическом уровне сигнала. Поскольку мы рассматриваем устройства только с двумя фиксированными значениями, то оказывается удобным приписать одному уровню значение 0, а другому – 1 и говорить о логическом уровне сигнала 0 и логическом уровне сигнала 1. Причём совершенно не обязательно, что 1 соответствует большему напряжению сигнала. Например, можно уровню 0 В приписать логическую 1, а уровню сигнала 3,3 В – логический 0. Тогда получается, что естественным форма-

лизмом на уровне цифровых схем является булева алгебра. Кроме того, широко используются графы.

1. ЦИФРОВАЯ СХЕМА. МОДЕЛИ ЦИФРОВЫХ СХЕМ

1.1. Классы цифровых схем

Множество всех цифровых схем принято делить на два больших класса: комбинационные схемы и последовательностные схемы [1, 2]. Значение выходного сигнала для комбинационной схемы зависит только от значения сигналов на её входах в данный момент времени. Говорят, что комбинационная схема не обладает памятью.

Последовательностные схемы наоборот частично помнят свою историю, и значение выходного сигнала для одного и того же входного сигнала может оказаться разным в зависимости от времени подачи этого входного сигнала.

В данном пособии мы рассматриваем только комбинационные схемы.

1.2. Логическая схема

Логическая схема – это ориентированный граф $G = <V, E>$ без циклов. Все вершины множества V могут быть одного из трёх видов: входными полюсами, выходными полюсами или логическими элементами. Дуги могут быть кратными, то есть между двумя вершинами может существовать более одной дуги.

Каждому логическому элементу соответствует некоторая булева функция арности (степени) k.

Входные полюса не имеют входящих дуг (полустепень входа для них равна 0), и имеют как минимум одну исходящую дугу (полустепень исхода как минимум равна 1).

Выходные полюса не имеют исходящих дуг (полустепень исхода равна 0) и ровно одну входящую дугу (полустепень входа равна 1).

Логические элементы имеют ровно k входящих дуг и как минимум одну исходящую. Здесь k – это арность булевой функции $f(x_1, \ldots, x_k)$, которая соответствует данному логическому элементу. Каждой входя-

щей дуге i логического элемента соответствует ровно один аргумент x_i функции f.

Логическую схему с n входными полюсами и m выходными полюсами будем называть (n, m)-полюсной схемой или просто (n, m)-схемой.

Логическая (n, m)-полюсная схема преобразует булевы вектора длины n в булевы вектора длины m. Для описания процесса преобразования введём правила сопоставления вершинам и дугам логической схемы булевых констант.

Если некоторой вершине логической схемы поставлена в соответствие булева константа r, то всем исходящим дугам этой вершины тоже сопоставляется данная булева константа.

Если всем k входящим дугам некоторой вершины сопоставлены булевы константы $r_1, r_2, \ldots, r_k$, то данной вершине необходимо поставить в соответствие булеву константу $y = f(r_1, r_2, \ldots, r_k)$, где f – функция, реализуемая логическим элементом, а r_i – константа, сопоставленная входящей дуге i вершины схемы.

Пронумеруем входные полюса заданной логической (n, m)-схемы целыми числами от 1 до n, а выходные полюса – целыми числами от 1 до m, и будем считать эту нумерацию фиксированной.

Пусть задан булев вектор $(r_1\, r_2 \ldots r_n)$ длины n, который подается на схему (входной вектор). Каждому входному полюсу i схемы сопоставим в соответствие булеву константу r_i. Будем применять вышеописанные правила до тех пор, пока всем выходным полюсам не будет приписана некоторая булева константа. Благодаря данному нами определению логической схемы, через конечное число шагов каждый выходной полюс получит определенное значение, и оно будет однозначным.

Булев вектор $(p_1 \ldots p_m)$, где константа p_j сопоставлена выходному полюсу j, является выходным вектором схемы, на которую подали входной вектор $(r_1\, r_2 \ldots r_n)$.

1.3. Система булевых функций

Если перебрать все вектора длины n и, воспользовавшись процедурой, описанной в предыдущем разделе, для каждого из них найти выходной вектор некоторой (n, m)-схемы G, то получим таблицу истинности некоторой функции $F : B^n \to B^m$. Эту функцию можно предста-

вить как систему булевых функций $f_1(x_1, \ldots, x_n)$, …, $f_m(x_1, \ldots, x_n)$. Будем говорить, что заданная схема G реализует функцию F (реализует систему булевых функций $f_1, \ldots, f_n$).

Очевидно, что одну и ту же систему булевых функций могут реализовывать различные логические схемы. В связи с этим возникают два вопроса. Реализует ли заданная логическая схема заданную систему булевых функций? Реализуют ли две заданные логические схему одну и ту же систему булевых функций? Оба этих вопроса являются предметами изучения теории тестирования и верификации цифровых схем.

1.4. Модели цифровых схем

Логическая схема и система булевых функций, описанные в предыдущих разделах, являются моделями цифровых схем. Обе эти модели задают функциональное описание схемы на уровне логики, поскольку однозначно определяют, как должна вести себя схема (как реагировать на входные воздействия), а сигналы заданы логическими уровнями (булевыми векторами). Кроме того, если задать множество булевых функций, которые можно приписывать логическим элементам, множество функций, которое реализуется элементной базой на уровне цифровой схемы, а так же задать временные характеристики элементов (времена задержек сигналов), то логическую схему можно рассматривать как цифровую схему. Таким образом, при определённых условиях логические схемы и системы булевых функций затрагивают сразу два уровня абстракции цифровых устройств: уровень цифровой схемы и логический уровень.

В дальнейшем, в данном пособии, мы будем использовать термины «схема», «логическая схема» и «цифровая схема» как равнозначные, если иное не будет оговорено отдельно.

2. ФОРМАТЫ ПРЕДСТАВЛЕНИЯ СХЕМ

В данном пособии мы познакомим читателя с работой системы ABC [3, 4], представим основные команды этой системы и дадим самые простые алгоритмы решения некоторых задач с помощью системы ABC. Но чтобы сделать это, сначала нужно описать, в каком виде система ABC принимает на вход схемы. Конечно же, описания схем хранятся в файлах и файлы должны быть в определённых форматах. Сре-

ди всех форматов описания мы выделим три: ISCAS'89 (он же .bench), .pla и .blif.

Специально отметим, что мы рассматриваем только комбинационные схемы, поэтому возможности этих форматов, связанные с последовательностями схемами, мы освещать не будем.

2.1. Формат .bench

Это очень простой формат [2], он практически один в один соответствует логической схеме (графу).

В файле этого формата одна за другой построчно перечислены директивы. Одной строке соответствует ровно одна директива. Часть строки, начинающаяся с символа #, и до конца строки считается комментарием и не учитывается при чтении файла.

Каждая директива описывает либо входной полюс, либо выходной полюс, либо логический элемент.

Входной полюс описывается служебным словом INPUT и в скобках задаётся имя полюса (например, INPUT(x0)). Выходной полюс описывается служебным словом OUTPUT, а в скобках задаётся имя элемента, с которого берётся сигнал для выходного полюса (например, OUTPUT(y0)). То есть, выходные полюса оказываются безымянными. Чтобы избежать громоздких текстовых конструкций, будем вместо «выходной полюс, сигнал для которого берётся с элемента y» говорить просто: «выходной полюс (выход) y».

Логические элементы задаются в формате имя_эл = имя_функции (имя_1, имя_2, …, имя_k), где в качестве имени функции могут выступать следующие: AND, OR, XOR, NAND, NOR, XNOR, NOT, BUF, VDD, GND, *k*–арность функции, а имя_1, имя_2, …, имя_k – имена элементов или входных полюсов, выходы которых (исходящие дуги) соединены со входами (являются входящими дугами) рассматриваемого элемента. Первые шесть из перечисленных функций бинарные, функции NOT и BUF – унарные, а функции VDD и GND — ноль-арные. NOT – это отрицание, BUF – тождественная функция, VDD – константа 1, GND – константа 0.

В качестве примера рассмотрим логическую схему, изображенную на рисунке 1. На листинге 1 представлено описание рассматриваемой схемы в формате .bench.

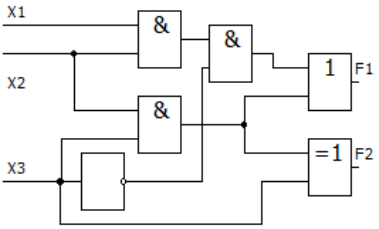

Рис. 1. Рабочий пример логической схемы

```
INPUT(X1)
INPUT(X2)
INPUT(X3)
OUTPUT(F1)
OUTPUT(F2)
AND1   = AND(X1, X2)
AND2   = AND(X2, X3)
AND3   = AND(AND1,NX3)
NX3    = NOT(X3)
F1     = OR(AND2, AND3)
F2     = XOR(AND2, X3)
```

Лист. 1. Описание рабочего примера в формате .bench

2.2. Формат .blif

Ещё один достаточно распространённый формат описания схем – это формат .blif. Он также тесно связан со структурой схемы, но в отличие от .bench позволяет задавать для элементов схемы произвольные функции произвольной арности.

Структура файла в формате .blif для (*n*, *m*)-схемы выглядит следующим образом:

```
.model имя_схемы
.inputs вход_1 вход_2 … вход_n
.outputs выход_1 выход_2 … выход_m
<ТЕЛО>
.end
```

Здесь имя_схемы – это имя схемы (модуля), вход_1, вход_2, …, вход_n – имена входных полюсов, выход_1, выход_2, …, выход_m – имена выходных полюсов. Ключевое слово .end завершает описание схемы.

Тело схемы представляет собой последовательность описаний элементов, каждый элемент описывается блоком вида

```
.names имя_1 имя_2 … имя_k имя
интервал_1 константа
интервал_2 константа
…
интервал_j константа
```

Здесь имя_1, имя_2, …, имя_k – имена вершин схемы (полюсов или логических элементов), с выходов которых берутся сигналы для данного элемента, имя – имя данного элемента, интервал_i для всех *i* от 1 до *j* – это булевы интервалы длины *k*, а константа – это булева константа, причём она должна быть одинаковой для всех *j* строк. Таким образом, булева функция, соответствующая данному логическому элементу, описывается либо характеристическим множеством M_0 (если константа равна 0), либо характеристическим множеством M_1 (если константа равна 1). Характеристические множества представлены интервалами.

Два разных блока (элемента) не могут иметь одно и то же имя. Так же как и для формата .bench выходные элементы «безымянны», они ассоциированы с элементами, с которых берётся выходной сигнал.

На листинге 2 представлено описание схемы из рабочего примера в формате .blif.

```
.model example.blif
.inputs X1 X2 X3
.outputs F1 F2
.names X1 X2 AND1
11 1
.names X3 NX3
0 1
.names X2 X3 AND2
11 1
.names AND1 NX3 AND3
11 1
.names AND2 AND3 F1
-1 1
1- 1
.names X3 AND2 F2
01 1
10 1
.end
```

Лист. 2. Описание рабочего примера в формате .blif

2.3. Другие форматы

Существуют и другие форматы описания схем. Например, иерархический .blif, .pla, VHDL, Verilog и т.д. Мы предполагаем, что иерархический .blif и .pla читатель изучит сам. Остальных форматов в данном пособии мы касаться не будем.

Задания для самостоятельной работы

1. Нарисовать логические схемы, реализующие сумматор, сумматор по модулю 2, мультиплексор, кодер.
2. Описать данные схемы в форматах .bench, .pla и .blif.

3. СИСТЕМА ABC

Система ABC [3], разработанная в Калифорнийском университете Беркли, предназначена для синтеза и верификации цифровых схем. Данная система позволяет осуществлять преобразование файлов из одного формата в другой, осуществлять оптимизацию схем, представленных в формате .bench, а так же осуществлять проверку эквивалентности двух схем, т.е. определять, производит ли схема одинаковые выходные последовательности на одни и те же входные последовательности.

Система ABC работает не непосредственно со схемами, представленными в виде системы булевых функций или в виде графа, а со схемами, представленными в форматах .blif, .pla и .bench. Прежде чем выполнять какие-либо действия со схемами, необходимо установить, настроить и запустить систему ABC.

3.1. Установка, настройка и запуск системы ABC

Работа в системе ABC происходит через интерфейс командной строки. Запуск системы на компьютере с операционной системой Windows возможен с помощью исполняемого файла доступного для скачивания на сайте разработчиков по ссылке [3] в разделе «Getting ABC». Система ABC постоянно дорабатывается и обновляется авторами. Текущая версия системы, представленная в виде исходных кодов на языке программирования C доступна в онлайн репозитории по ссылке [4]. Система ABC предоставляется на условиях свободной лицензии, дающей право на свободное использование, изменение и распространение данного программного обеспечения в любых целях.

3.2. Работа с файлами в системе ABC

Иногда возникают задачи, требующие преобразования схемы, представленной в одном формате, в какой-либо другой формат. Например, требуется оптимизировать схему, представленную в .pla формате. Поскольку в данном формате с помощью ABC оптимизировать схему не представляется возможным, то сначала требуется преобразовать формат .pla в .bench.

В данном разделе описаны команды, позволяющие преобразовывать различные форматы.

После запуска системы, в командной строке отображается приглашение в следующем виде:

```
abc 01>
```

Список всех доступных команд в текущей версии системы можно получить, вызвав команду help. Для считывания описания логической схемы из файла в любом из доступных форматов используется команда read. Пример команды считывания описания схемы из файла с именем circuit.blif, находящегося на диске C приведен ниже.

```
abc 01>read C:\circuit.blif
```

Также, в системе доступны команды, обеспечивающие считывание только определенных форматов файлов. Например, для формата .blif это команда read_blif, для формата .bench – команда read_bench и т.д. Полный список команд можно получить, вызвав команду help.

После вызова команды считывания файла с описанием логической схемы, становится доступным все множество команд, реализующих поддерживаемые алгоритмы логического анализа и синтеза.

Как правило, после необходимых преобразований текущей схемы, необходимо сохранить результат преобразований в виде нового файла, содержащего описание логической схемы. Для этого используется команда write. Например, для записи в файл с именем new_circuit.blif текущей схемы, можно выполнить следующую команду

```
abc 01>write C:\new_circuit.blif
```

При использовании этой команды, формат описания схемы будет выбран на основе указанного расширения файла, в текущем примере

это формат .blif. Для явного задания формата файла с сохраняемой схемой в независимости от указанного расширения можно использовать команды write_blif, write_bench, write_eqn и т.д., по аналогии с набором команд для считывания схем из файлов.

3.3. Оптимизация схем с помощью ABC

При считывании схемы из файла, она преобразуется в формат списка соединений (netlist), который затем автоматически преобразуется в формат логической сети (logic network). Для схемы, представленной в этом формате доступны некоторые операции, например, минимизация. Минимизацию логической схемы можно осуществить с помощью команды espresso.

```
abc 01>read C:\circuit.blif
abc 02>espresso
abc 03>write_blif C:\circuit_min.blif
```

Основной формой внутреннего представления логической схемы в системе ABC является И-НЕ граф (And-Inverter Graph, AIG). Данный формат представляет собой граф, вершинами которого являются двухвходовые логические элементы И, а дуги представляют соединения (возможно, с инверсией) данных элементов. Для преобразования схемы в данный формат используется команда strash. Для минимизации схемы, представленной в виде И-НЕ графа можно воспользоваться командой fraig, которая позволяет обнаружить и объединить две и более вершины И-НЕ графа, представляющие одинаковую булеву функцию.

3.4. Проверка эквивалентности двух схем

Поскольку в ABC для оптимизации используются эвристики, то оптимизированная схема, вполне возможно, окажется не эквивалентной исходной. Поэтому требуется проверить, являются ли две схемы эквивалентными. Данная задача решается посредством выполнения следующих шагов.

Для проверки эквивалентности двух комбинационных схем в ABC применяется команда cec. Данная команда основана на решении задачи

выполнимости для булевой функции, представляемой майтером двух проверяемых схем. Майтером называется схема, получаемая отождествлением одноименных входов схем и подачей одноименных выходов на элементы XOR. Таким образом, если один из выходов майтера может принять значение логической единицы, то это значит, что исходные схемы производят различные выходные последовательности на одинаковые входные последовательности, то есть не являются эквивалентными.

На рисунке 2 и листинге 3 представлена оптимизированная версия схемы из рабочего примера и ее описание в формате .bench.

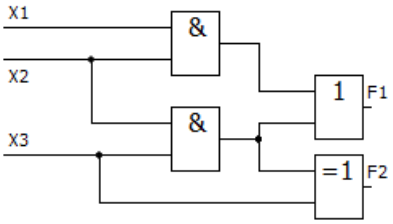

Рис. 2 – Оптимизированная схема рабочего примера

```
INPUT(X1)
INPUT(X2)
INPUT(X3)
OUTPUT(F1)
OUTPUT(F2)
AND1    = AND(X1,X2)
AND2    = AND(X2,X3)
F1      = OR(AND1,AND2)
F2      = XOR(AND2,X3)
```

Лист. 3 – Оптимизированная схема в формате .bench

Для проверки эквивалентности исходной и оптимизированной схем, воспользуемся командой cec. Для этого сначала сохраним исходную схему в файле с названием example.bench, а оптимизированную схему – в файле example_opt.bench. Далее в командной строке ABC введем:

```
abc 01>cec C:\example.bench C:\example_opt.bench
```

Вывод Networks are equivalent команды cec говорит о том, что схемы эквивалентны, следовательно, оптимизация проведена корректно.

3.5. Построение тестов

При построении тестовых наборов для логических схем часто пользуются подходом, основанным на проверке отсутствия в схеме неисправностей определенного типа. Наиболее распространенным типом неисправностей, рассматриваемых для логических схем, является одиночная константная неисправность. Такая неисправность возникает в

схеме, когда выход одного из элементов схемы принимает значение только логического нуля или только логической единицы (как бы «залипая» на 1 или 0) вне зависимости от поступающих входных сигналов. Для моделирования такой неисправности при описании схемы в формате .blif необходимо для одного элемента заменить значения всех входных сигналов на символ «–», а выходное значение – на 0 либо на 1.

На листинге 4 представлено .blif описание схемы рабочего примера, в котором присутствует одиночная константная неисправность, связанная с тем, что выход элемента AND2 равен 0.

Для моделирования случая, когда «залипает» один из главных входов схемы, необходимо ввести в схему дополнительный элемент, входом которого будет главный вход схемы. На листинге 5 приведен пример, на котором вход X2 «залип» на 1.

```
.model example.blif_AND2_S0
.inputs X1 X2 X3
.outputs F1 F2
.names X1 X2 AND1
11 1
.names X3 NX3
0 1
.names X2 X3 AND2
-- 0
.names AND1 NX3 AND3
11 1
.names AND2 AND3 F1
-1 1
1- 1
.names X3 AND2 F2
01 1
10 1
.end
```

Лист. 4. Одиночная константная неисправность (элемент AND2)

```
.model example.blif_X2_S1
.inputs X1 X2 X3
.outputs F1 F2
.names X2 X2_S1
- 1
.names X1 X2_S1 AND1
11 1
.names X3 NX3
0 1
.names X2_S1 X3 AND2
11 1
.names AND1 NX3 AND3
11 1
.names AND2 AND3 F1
-1 1
1- 1
.names X3 AND2 F2
01 1
10 1
.end
```

Лист. 5. Одиночная константная неисправность (вход X2)

Для комбинационной схемы, имеющей k элементов и n входов, число всевозможных одиночных константных неисправностей составляет $2(k + n)$.

Для построения полного тестового набора необходимо воспользоваться понятием модели неисправности. Под моделью неисправности будет понимать набор $\langle S, \approx, FD \rangle$, где S – спецификация логической схемы, т.е. схема, задающая эталонное поведение; FD – множество неисправностей, включающее в себя все схемы с неисправностями рассматриваемого типа (например, одиночные константные неисправности); $\approx$ – отношение эквивалентности. Две комбинационные схемы находятся в данном отношении, если производят одинаковые выходные последовательности на всевозможные входные последовательности.

Тест называется полным, если для каждой неисправной схемы в нем существует последовательность, реакция спецификации на которую отличается от реакции неисправной схемы.

Для получения полного тестового набора относительно рассматриваемого типа ошибок нужно получить различающую последовательность для каждой пары $\langle S, E \rangle$, где S – эталонная схема (спецификация), E – неисправная схема из множества FD. Для получения различающей последовательности для двух схем нужно построить майтер этих схем и решить для него задачу выполнимости. Для построения майтера двух схем, находящихся в файлах с именами C:\circuit_1.blif и C:\circuit_2.blif, нужно выполнить команду:

```
abc 01>miter C:\circuit_1.blif C:\circuit_2.blif
```

Для решения задачи выполнимости в системе ABC используется инструмент MiniSAT. Для его запуска используется команда sat.

```
abc 02>sat
```

Результатом выполнения команды sat является вывод на экран строки “SATISFIABLE” в случае, если для майтера существует выполняющий набор (различающая последовательность), либо “UNSATISFIABLE”, если для майтера не существует выполняющего набора, что означает, что проверяемые схемы эквиваленты. Если различающая последовательность существует, то ее можно записать в файл с помощью команды write_counter.

```
abc 03>write_counter C:\test_sequence.txt
```

Таким образом, объединение всех тестовых последовательностей составляет полный тестовый набор относительно неисправностей заданного типа.

Задания для самостоятельной работы

1. Оптимизировать схемы, построенные в предыдущем разделе.
2. Убедиться, что оптимизированные схемы эквивалентны исходным схемам.
3. Внести в каждую схему по одной константной неисправности и построить тест, обнаруживающий данную неисправность.

ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. *Харрис Д., Харрис С.* Цифровая схемотехника и архитектура компьютера. М.: ДМК Пресс, 2017. 772 с.
2. *Скобцов Ю.А., Скобцов В.Ю.* Логическое моделирование и тестирование цифровых устройств. Донецк: ИПММ НАН Украины, ДонНТУ, 2005. 436 с.
3. *Mishchenko A.* ABC: A System for Sequential Synthesis and Verification: Berkley University. URL: https://people.eecs.berkeley.edu/~alanmi/abc/ (дата обращения: 26.03.2018).
4. *Mishchenko A.* ABC: A System for Sequential Synthesis and Verification. URL: https://bitbucket.org/alanmi/abc (дата обращения: 26.03.2018).

Издание подготовлено в авторской редакции

Отпечатано на участке цифровой печати
Издательского Дома Томского государственного университета

Заказ № 3109 от «29» марта 2018 г. Тираж 50 экз.