МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”
(ФГБОУ ВО «ВГУ»)

Физический факультет
Кафедра физики полупроводников и микроэлектроники

ЭВОЛЮЦИОННЫЕ ВЫЧИСЛЕНИЯ ДЛЯ ОПТИМИЗАЦИИ СХЕМ КОМБИНАЦИОННОЙ ЛОГИКИ

Выпускная квалификационная работа
03.03.03 – радиофизика
Микроэлектроника и полупроводниковые приборы

Зав. кафедрой _____ д.ф.-м.н., проф., Бормонтов Е.Н.
Обучающийся _____ студентка Волкова Ж.В.
Руководитель _____ к.т.н., доцент Николаенков Ю.К.

Воронеж 2020

Реферат

УДК 004.8

Волкова Ж.В.

Эволюционные вычисления для оптимизации схем комбинационной логики

Бакалаврская работа, Воронеж, ВГУ, 2020г. – 55 страниц, 46 иллюстраций, 20 источников.

Ключевые слова: комбинационная логика, генетические алгоритмы, эволюционная электроника, последовательностные логические схемы.

В данной бакалаврской работе представлены методы и технологии эволюционной логики, в среде программирования MATLAB продемонстрированы возможности эволюционной электроники в области автоматической многопараметрической оптимизации комбинаторных логических схем, проведённого сравнительный анализ результатов работы ручного метода оптимизации и автоматического.

Содержание

Введение

Системой комбинационной логики принято называть систему логических устройств, выходные функции которых однозначно определяются входными логическими функциями в тот же момент времени - имея на входе двоичные сигналы, получаем на выходе логические функции, не зависящие от сигналов, прошедших через систему в предыдущие моменты времени.

Комбинаторная логика включает в себя главным образом объекты, а также способы комбинирования объектов.

Комбинирование одних объектов с другими выполняется с помощью изначально выделенных объектов-констант, которые называются комбинаторами.

На данный момент комбинаторная логика является одним из основных математических аппаратов компьютерной науки, однако - она не совершенна.

Во-первых, исходный набор комбинаторов одинаков для каждой новой задачи. Иначе говоря, если мы попробуем решить сложную задачу путем комбинаторной логики, системе придется разложить ее на несколько более простых задач, а каждую из них, в свою очередь, на несколько очень легких, и так может продолжаться большое количество циклов, а значит, решение займет значительное время. Во-вторых, всегда возможно, что результат все равно не будет представлен в максимально рациональной - оптимизированной - форме. Человеческий разум все еще намного более гибок, и часто даже обыватель может сказать, что видит более рациональные методы решения той или иной задачи.

Наиболее распространены к этому времени следующие методы оптимизации цифровых схем: метод карт Карно, в качестве основных объектов использующий минтермы и макстермы, и метод Мак-Класки, но оба они не совершенны и имеют ограничения уже по объёму входящих данных.

Перед нами стоит задача: научить программу использовать то, что до поры до времени представляется исключительно прерогативой человека - опыт. Эволюционные генетические алгоритмы в основе своей используют именно идеи комбинирования.

Подобные программы лишь недавно достигли достаточно высокого уровня развития и в настоящее время именно эта отрасль электроники имеет высокий приоритет развития. Как использовать эволюционные вычисления для повышения интеллектуальности систем проектирования схем логики? Это и будет целью данной работы. Ознакомимся также с методами и технологиями эволюционной электроники, изучим генетические алгоритмы для автоматической многопараметрической оптимизации комбинационных логических схем.

Для демонстрации возможностей эволюционной электроники будет использован пакет расширений Genetic Algorithm and Direct Search Toolbox среды программирования MATLAB. Также ручные методы для сравнительного анализа реализованы в среде MATLAB.

1. Общий обзор схем комбинационной логики. Достоинства и недостатки существующих подходов

В этой главе основное внимание мы уделим основополагающим моментам в построении комбинаторных схем, в основном тем, с которыми возникают проблемы при оптимизации. Рассмотрим уже существующие и широко используемые в наше время подходы и разберемся в их достоинствах и недостатках.

1.1. Основные препятствия в построении и оптимизации схем комбинаторной логики

В электронике *схемой* называют цепь, способную обрабатывать дискретные сигналы. Еще ее можно рассматривать как «черный ящик»: входы — функциональные спецификации — выходы.

Комбинационная логическая цепь — цепь, способная обрабатывать бинарные электрические сигналы, выходной сигнал которой в каждый момент времени зависит только от входного сигнала. *Последовательностные* схемы, в отличие от комбинационных, имеют память, а значит, выходные значения зависят не только от текущих входных, но и от предшествующей последовательности сигналов, другими словами, последовательностная схема является сочетанием комбинационной схемы и элементов памяти. Память строится на основе триггеров.

Рассмотрим для примера простую задачу построения комбинационной логической схемы по аналитически заданной функции. Имея логическую функцию в виде y=f(x1,x2,x3,...xn), мы можем перейти к созданию схемы цифрового логического устройства, которое будет обрабатывать поступающие логические сигналы x1,x2,x3,...xn по заданным требованиям. Рассмотрим конкретную функцию:

$$Y = \overline{X_1} \cdot X_2 \cdot X_3 + X_1 \cdot \overline{X_2} \cdot X_3 + X_1 \cdot X_2 \cdot \overline{X_3} + X_1 \cdot X_2 \cdot X_3$$

Рис.1.1.1

Набор необходимых простейших логических элементов включает в себя: 3 инвертора, 4 трехвходовых конъюнктора, 1 четырехвходовый дизъюнктор (8 логических элементов). Составленная схема изображена на рис.1.1.2.

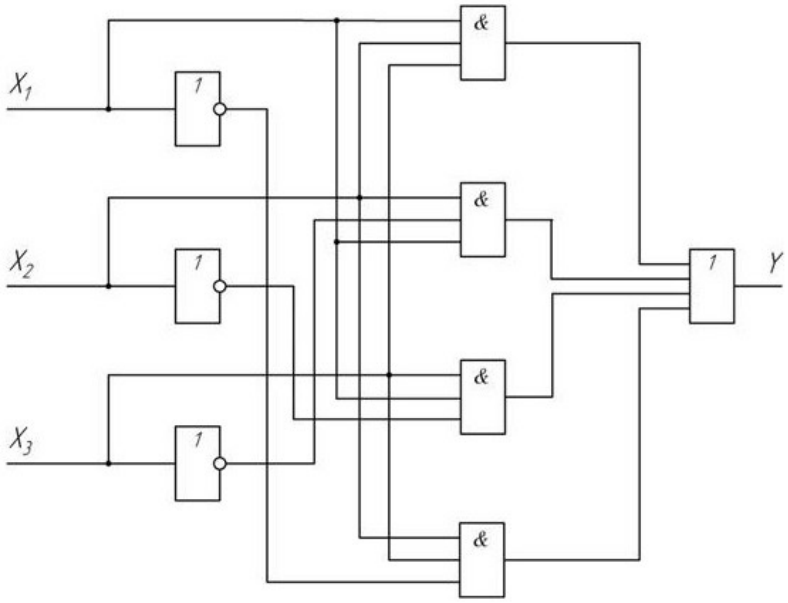


Рис.1.1.2.

Мы столкнулись с задачей оптимизации. Математически задача оптимизации звучит следующим образом:

Необходимо минимизировать (в частном случае задача может заключаться в максимизации) логическую схему, учитывая ограничения на управляемые переменные.

Под минимизацией или максимизацией функции n переменных $f(x)=f(x_1,x_2,\ldots x_n)$ на заданном множестве следует понимать определение хотя бы одной точки экстремума этой функции, а также при необходимости ее значения в этой точке. На практике это означает минимальное количество логических элементов. Логическим элементом мы называем простейшее устройство, выполняющие только одну определённую логическую операцию над выходным сигналом согласно правилам булевой алгебры.

Основными параметрами оптимизации можно считать быстродействие схемы, потребляемых ею мощность, а также площадь, занимаемую схемой на кристалле, и стоимость производства.

Вернёмся к полученной на простейших элементах схеме. При практической реализации уже из набора элементов очевидно, что такая схема вызовет перерасход микросхем, несмотря на кажущуюся простоту аналитического уравнения. Попробуем заменить простейшие логические элементы на более сложные и незаменимые: И-НЕ и ИЛИ-НЕ, изображение на рисунке 1.1.3.

$$Y = \overline{X_1 + X_2}$$

$$Y = \overline{X_1 \cdot X_2}$$

Рис.1.1.3

Эти элементы являются универсальным логическим базисом. Аналитический вид функции для преобразования в схему из базисных элементов, а также разбор функции в простых логических обозначения, приведены на рис.1.1.4.

Для выполнения схемы в элементах И-НЕ нам потребуется: 1 трехвходовый И-НЕ, 3 двухвходовых И-НЕ. Для выполнения схемы в элементах ИЛИ-НЕ необходимы: 3 двухвходовых ИЛИ-НЕ, 1 трехвходовый ИЛИ-НЕ. В каждой схеме 4 элемента.

$$y = x_1 x_0 \vee x_2 x_0 \vee x_2 x_1.$$

$$Y = (X_2 \vee X_1) \cdot (X_1 \vee X_0) \cdot (X_2 \vee X_0),$$

И–НЕ

$$y = x_1 x_0 \vee x_2 x_0 \vee x_2 x_1 = \overline{\overline{x_1 x_0 \vee x_2 x_0 \vee x_2 x_1}} = \overline{\overline{x_1 x_0} \cdot \overline{x_2 x_0} \cdot \overline{x_2 x_1}}.$$

ИЛИ–НЕ.

$$Y = (X_2 \vee X_1) \cdot (X_1 \vee X_0) \cdot (X_2 \vee X_0) = \overline{\overline{(X_2 \vee X_1) \cdot (X_1 \vee X_0) \cdot (X_2 \vee X_0)}} =$$
$$= \overline{\overline{(X_2 \vee X_1)} \vee \overline{(X_1 \vee X_0)} \vee \overline{(X_2 \vee X_0)}}.$$

Рис. 1.1.4

Итак, имеем как минимум три возможных варианта построения схемы: в базисе простейших элементов, базисе И-НЕ, ИЛИ-НЕ или смешанном. При оптимизации нужно учесть не только сложность построения — например, на простом базисе большинство существующих программ способны реализовать схему автоматически, в то время как для включения в схему более сложных элементов, таких как сумматоры, шифраторы, мультиплексоры и другие, требуется вмешательство человека, — но и стоимость, которую потребуется затратить на реализацию партии таких схем, а также быстродействие по заданным параметрам, которые могут очень разниться для выполнения различных задач. Нельзя упускать из виду и помехи, обязательно возникающие в разных областях и процессах создания и работы будущей схемы.

Рассмотрим для примера импульсные помехи — одиночное изменение сигнала на входе вызывает несколько выходных изменений. На рисунке 1.1.5 показана схема, подверженная таким паразитным импульсом, и карта Карно для нее.

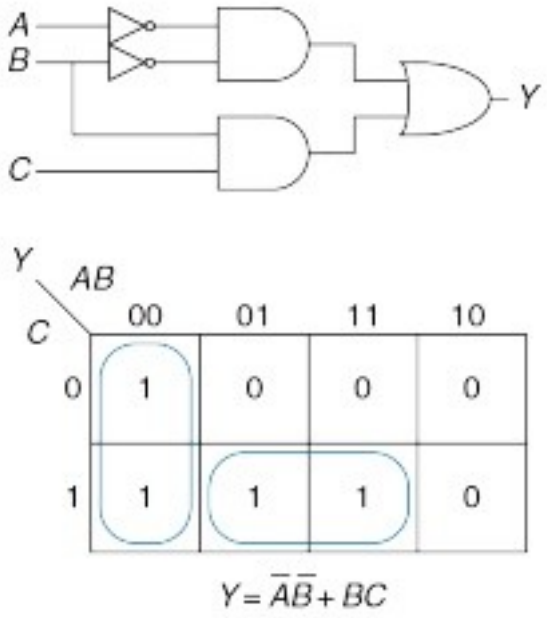


Рис 1.1.5.

Мы можем исходя из вышеизложенного определить, что логическое уравнение минимизировано правильно, но мы также видим, что произойдёт, если A=0, C=1 и B меняется из 1 в 0. На рисунке 1.1.6 наглядно показана эта ситуация. Серым обозначен короткий путь через два элемента— И и ИЛИ, а синим показан критический путь через инвертор и элементы И и ИЛИ.

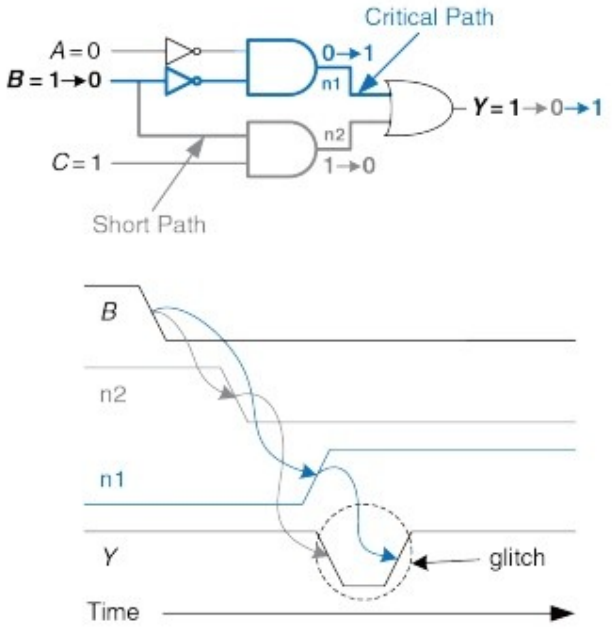


Рис.1.1.6

Как только B переключается из 1 в 0, n2 в коротком пути опускается в 0 до того, как в критическом пути n1 приобретает значение 1. До подъёма n1 оба входа на элементе ИЛИ имеют значение 0, выход, соответственно, тоже сбрасывается в 0. Когда n1 поднимается, Y тоже возвращается в 1, как показано на рисунке. То есть, Y начинается с 1 и заканчивается 1, но на короткое время принимает значение 0. Пока интервал, равный времени задержки распространения, выдержан верно, такая импульсная помеха не является проблемой. Этого импульса, однако, можно избежать добавлением дополнительного элемента в схему. На рисунке 1.1.7 карта Карно имеет другой вид для той же функции. Показано, как изменение при переходе из ABC=001 в ABC=011 приводит к переходу от одной первичной имликанты к другой, на это и нужно обращать внимание при выявлении возможности появления импульсов помехи.

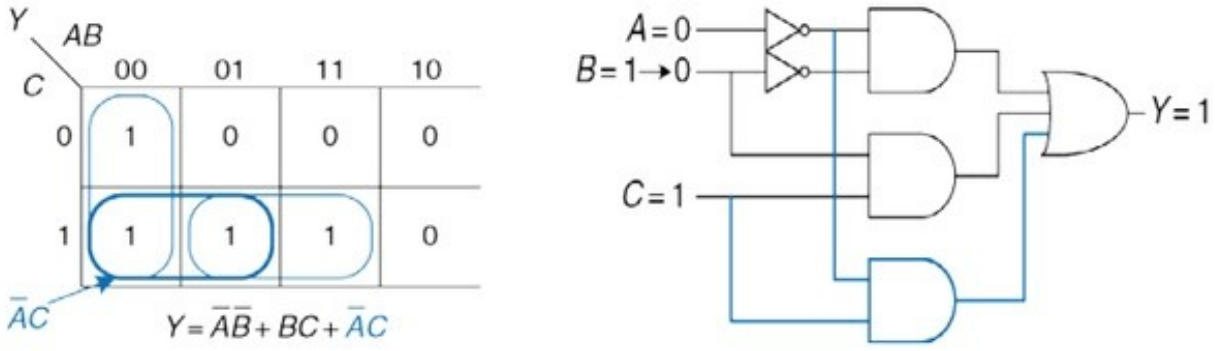


Рис.1.1.7.

Новая цепь в схеме охватывает границу первичных имликант, выделенную дополнительно синим цветом на карте Карно. Добавленные терм является согласованным или *избыточным.* В итоге схема, представленная на рисунке 1.1.7, является устойчивой к паразитным импульсам.

Можно видеть, что способ добавления избыточным импликант к картам Карно достаточно затратный, а большинство интересующих нас систем имеют одновременное переключения множества входов, в следствие чего возникновение паразитных импульсов в них неизбежно. Для подавления одного импульса для небольшой схемы мы добавили один элемент, но в больших масштабах такая перспектива выглядит пугающей, и следует найти более высокоуровневые методы оптимизации или же заняться этим вручную.

Комбинационная логика, основанная на дизъюнкции конъюнкций, то есть сумме произведений, считается двухуровневой. Первый уровень составляют элементы И, второй — элементы ИЛИ. Однако при построении многоуровневых схем можно использовать меньшее количество логических элементов, чем в двухуровневой реализации той же схемы. Самым простым примером будет *исключающее ИЛИ* — XOR для нескольких переменных. Рассмотрим построение трехвходового элемента XOR с применением двухуровневой схемы. N-входовый элемент XOR выдаёт на выход значение истины, если несметное число входных операндов имеют значение *истина.* Аналитическая функция будет иметь как на рисунке 1.1.8.

$$Y = \overline{A}B\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

Рис. 1.1.8.

Трехвходовый элемент XOR можно реализовать на двухуровневой схеме, как на рисунке 1.1.9.

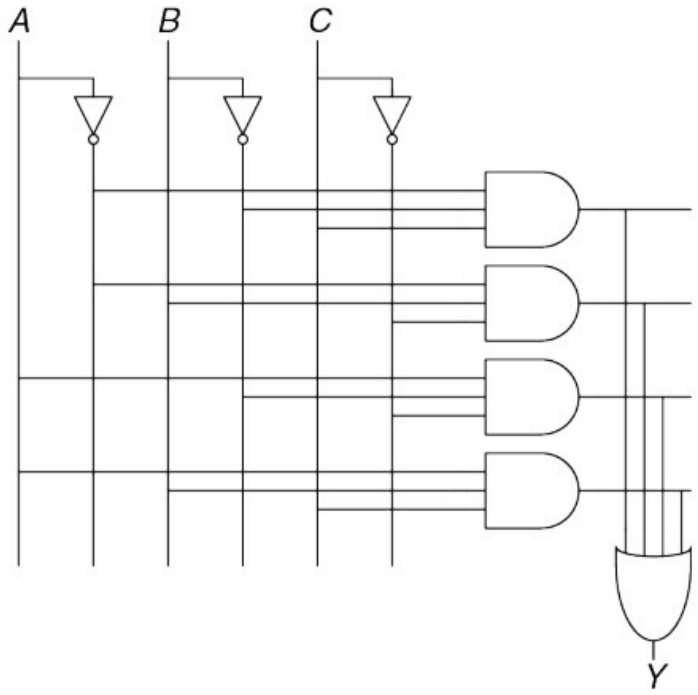


Рис.1.1.9.

Для реализации потребовалось: 3 инвертора, 4 трехвходовых конъюнктуры, 1 четырехвходовый дизъюнктор. И это один элемент, логика которого в человеческом понимании может считаться весьма примитивной. Можно представить и реализацию пятивходового XOR для двухуровневой реализации. Дерево двухуровневых элементов представляется гораздо лучшей альтернативой. Например, для реализации восьмивходового XOR потребуется семь двухвходовых. Тем не менее, выбор наилучшего варианта многовходовой реализации — это сложный процесс. Кроме того, нельзя забывать про то, что под «наилучшим» могут подразумеваться различные моменты: как наименьшее количество элементов, так и быстродействие, быстрая разработка, дешевизна элементов или минимальное энергопотребление. Также для некоторых технологий подходят определённые элементы, на которых реализация будет более эффективной.

Исходя из этого, можно сказать, что несмотря на уже достойный уровень развития методов логического упрощения, все еще существуют задачи, с которыми человек с достаточным опытом может справиться лучше, чем предназначенная для этого программа. Оптимизация большинством существующих методов является процедурой сложной, затратной и зависящее

от большого количества параметров, в числе которых и цели создания данной конкретной схемы.

В конечном итоге, эволюционные алгоритмы используются для автоматического проектирования синхронных последовательных логических схем с минимальным количеством логических элементов.

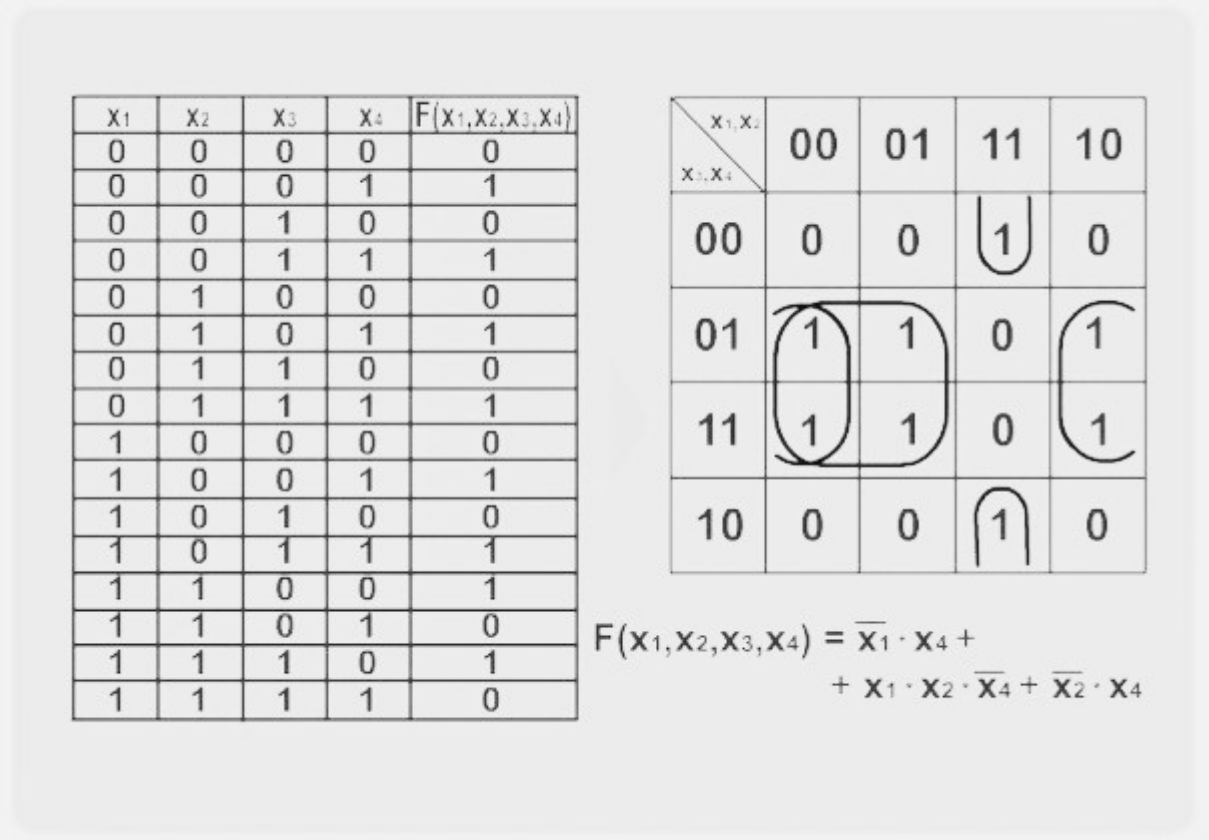
1.2. Достоинства и недостатки существующих подходов

В обсуждении существующих подходов интересно обратить внимание, что генетические алгоритмы как процедуры, основанные на механизмах естественного отбора и наследования, также подчиняются эволюционному принципу выживания наиболее приспособленных особей. На определенном уровне развития, имея выбор между различными подходами, люди навсегда оставят в прошлом некоторые из них.

Рассмотрим существующие и широко использующиеся в наше время подходы. Самые распространённые — карты Карно и метод МакКласки, также используются методы испытания импликант, импликантных матриц и метод Квайна. Рассмотрим первые два.

Метод карт Карно прост и понятен в использовании. Он позволяет минимизировать логические функции с относительно небольшим числом переменных – не более шести. Это графический способ минимализации булевых функций. Карты Карно можно рассматривать как перестроенную таблицу истинности функции или как развертку n-мерного булевого куба.

На рис.1.2.1. представлена функция в виде таблицы истинности и ее минимализация с помощью метода карт Карно.



x_1	x_2	x_3	x_4	$F(x_1,x_2,x_3,x_4)$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

x_3,x_4 \ x_1,x_2	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	1	1	0	1
10	0	0	1	0

$$F(x_1,x_2,x_3,x_4) = \overline{x_1} \cdot x_4 + x_1 \cdot x_2 \cdot \overline{x_4} + \overline{x_2} \cdot x_4$$

Рис.1.2.1.

Следует обратить внимание на то, что каждая клетка или *минтерм* отличается от соседней клетки изменением лишь одной переменной. Также комбинируется и переменные X1 и X2, X3 и X4 согласно коду Грея: 00, 01, 11, 10, где соседние записи отличаются только на один разряд. Важно, что карты Карно «закольцованы», то есть крайняя правая клетка является соседней для крайней левой, самая верхняя будет соседствовать с самой нижней.

Из плюсов следует отметить простоту и интуитивную доступность метода и его наглядность. Из минусов – ограничение шестью аргументами, в случае перехода это границы метод утрачивает очевидные преимущества, а также важно, что выбор производится в значительной степени интуитивно, а значит, конечный результат во многом зависит от индивидуального опыта разработчика, то есть, увеличение числа переменных препятствует применению метода для минимизации на ЭВМ.

Метод Квайна-МакКласки является усовершенствованным методом Квайна и обычно называется просто методом МакКласки. Алгоритм применяется для функций, заданных в сокращенной дизъюнктивной нормальной форме — СДНФ. Это один из легко реализуемых для любого количества входных аргументов методов. Для метода МакКласки можно выделить два основных этапа:

1. Нахождение всех простых терм логической функции по законам склеивания, приведённым на рис.1.2.2.
2. Минимализация количества простых терм, полученных на первом пункте, с наиболее оптимальным результатом.

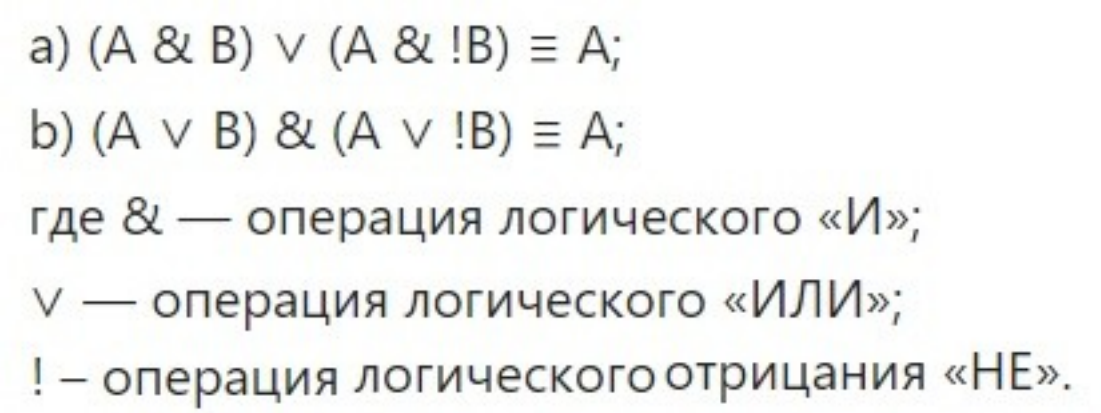


Рис.1.2.2.

Вначале происходит многократный перебор терм, чтобы выбрать подходящие для попарной склейки. Две термы считаются соседними и подходящими для склейки, если различны только в одной из позиций: у одной значение «0», а у другой — «1». Склеенные термы называются родительскими и исключаются из дальнейшей обработки, вместо них далее рассматривается дочерняя терма, у которой позиции различающихся переменных родительских терм заменены на знак «*». Этот приём расширяет двоичный алгоритм внутри термы до троичного, состоящего из «0», «1» и «*». В случае, если во время переборки в пункте 1 терма не находит отличной только на одну позицию термы, то она считается одним из результатов работы перебора и входит в следующий пункт алгоритма.

Покрытие, наиболее приближенное к кратчайшему, даёт алгоритм преобразования таблицы покрытий, основанный на методе “минимальный столбец – максимальная строка”. В общих чертах его логика состоит в следующем:

1. Исходная таблица является текущей преобразуемой таблицей покрытий, множество строк покрытий – пусто;
2. В текущей таблице выделяется столбец с наименьшим числом единиц. Среди строк, содержащих единицы в этом столбце, выделяется одна с наибольшим числом единиц. Эта строка включается в покрытие, текущая таблица сокращается вычеркиванием всех столбцов, в которых выбранная строка имеет единицу;
3. Если в таблице есть не вычеркнутые столбцы, то выполняется пункт 2, иначе – покрытие построено.

При подсчёте числа единиц в строке учитываются единицы в невычеркнуых столбцах.

Из плюсов этого метода очевиден в первую очередь тот факт, что число переменных на входе в теории любое, а на практике ограничено достаточно большими цифрами — некоторые алгоритмы работают при количестве входных переменных логических функций до 40, а некоторые включаются и при больших значениях, но их быстродействие обычно ниже.

Минусом являются значительные затраты памяти при обработке большого числа переменных. Время работы метода МакКласки экспоненциально растёт с увеличением числа входных аргументов. При большом количестве переменных во выходной функции используется эвристический алгоритм – алгоритм, включающий практический метод, не являющийся гарантированно точным и оптимальным. Такого алгоритма обычно достаточно для решения данной конкретной задачи, он используется в основном для ускорения поиска решения сложной задачи.

Для дальнейшего развития такой сферы технологий, как автоматическая оптимизация схем, требуется поиск новых, актуальных и экономных подходов. Одним из наиболее перспективных и подходящих считаются эволюционные вычисления, к которым относится метод генетических алгоритмов.

2. Генетические алгоритмы — история, понятия, терминология и основы работы

2.1. Основы работы генетического алгоритма и терминология

На сегодняшний день теоретические разработки в области искусственного интеллекта успешно применяются на практике. Благодаря перспективности этого раздела, большое внимание уделяется развитию направления, и уже в ряде задач решения, полученные с помощью искинов, не уступают разработанным людьми, а порой даже превосходят их. Одной из задач искусственного интеллекта и той самой, о которой идет речь в данной работе, является автоматическая разработка программ. Как мы уже выяснили, создание алгоритмов вручную является трудоёмким, долгим и порой чрезвычайно ресурсозатратным процессом. Логично поручить эту задачу компьютеру.

Эволюционные вычисления — широко используемое направление искусственного интеллекта, использующееся для автоматического создания программ. Сюда включают генетические алгоритмы, генетическое программирование и программирование с экспрессией генов.

Конечные программы, разработанные с помощью генетических алгоритмов, на данный момент широко применяются в задачах распознавания регулярных языков. Существует иное применение эволюционных вычислений — это проектирование клеточных автоматов. К сожалению, этот тип задач имеет пока что мало общего с практическими задачами автоматного программирования.

Подробно в работе рассмотрены генетические алгоритмы. Такой метод был предложен еще в середине восьмидесятых годов двадцатого века Джоном Холландом, что делает его первым из эволюционных алгоритмов. Особи генетического алгоритма — представленная в виде строк фиксированной длины информация. Обычно имеются в виду битовые строки.

Сама идея генетических алгоритмов появилась благодаря стремлению человека скопировать естественные процессы из мира живых организмов. ГА создали на примере эволюции и естественного отбора популяций живых существ. Организация эволюционного процесса, целью которого становится получение решения сложной задачи, заимствована у эволюционного процесса.

Удивительно сложилась история в самом начале существования идеи генетических алгоритмов: впервые она была применена на практике за десять лет до теоретического обоснования подхода.

Генетический алгоритм является комбинацией двух методов: переборного и градиентного. Механизмы скрещивания и мутации реализуют ту часть, которая

относится к переборному, а отбор лучших решений является градиентным спуском.

Метод перебора или метод равномерного поиска является методом нулевого порядка (или прямой, для реализации не требуется использовать производные целевой функции, иными словами, целевую функцию при использовании таких методов необязательно задавать в аналитической форме, достаточно даже табличной). Суть метода в следующем: интервал поиска *[a, b]* разбивается на *n* равных частей, *n* определяется исходя из заданной погрешности $E_{зад}$ по формуле $n=(b-a)/E_{зад}$. Далее рассматривается множество точек X_i, где $i=0,1,...n$. Численные значения вычисляются по формуле: $X_i=a+i(b-a)/n$. Сравнивания полученные значения, находим экстремум функции.

Градиентный метод является методом первого порядка, то есть для его использования требуется знание первой производной целевой функции. По этой причине целевую функцию нужно задать аналитически.

Направлением спуска удобно выбрать антиградиент целевой функции F(x) - вектор, направленный в сторону, противоположную градиенту. Итерационный вид процесса в таком случае будет: $X_i+1=X_k-a_k*F'(X_k)>0, k=0,1,2...$

Алгоритм является подходящим для оптимизации многомерной функции, и это его свойство, разумеется, разделяет и генетический алгоритм. Комбинация этих двух алгоритмов обеспечивает хорошую эффективность генетического поиска для любых типов задач.

ГА применяются для разработки программного обеспечения, также в системах искусственного интеллекта, для оптимизации логических схем, в искусственных нейронных сетях, а также в других областях. Они созданы для решения задач, которые раньше решались только с помощью нейронных сетей, являясь альтернативным им методом, но часто используются в связке с нейронными сетями или нечеткими системами.

Генетические алгоритмы, как было упомянуто выше, созданы по образу биологических процессов, и терминология сохранила такие слова, как гены, хромосомы, популяция, особь, а еще генотип и фенотип и, также, аллель. Из области технологий для понимания важны такие термины, как цепь, структура и двоичная последовательность.

Популяция - это конечное множество особей. В генетическом алгоритме особи представляют из себя хромосомы, содержащие закодированную информацию о множествах параметров задачи. Также особи можно называть организмами или индивидами. Хромосомы, цепочки или кодовые последовательности —

обозначения одного термина, это упорядоченные последовательности генов - элементов генотипа, в частности, элементов хромосомы.

С термином "генотип" в техническом лексиконе можно сопоставить термин "структура", подразумевая набор хромосом данной особи. Для данного индивида генотипом может являться как набор хромосом, так и одна-единственная.

Под словом "фенотип" подразумевается набор значений, соответствующих конкретному генотипу, но уже декодированных, то есть, множество параметров задачи, являющихся точкой пространственного поиска или решением.

Аллелью является значение одного гена, оно же значение свойства или вариант свойства.

Локусом или, проще говоря, позицией называют место размещения гена в хромосоме. Лот - место размещения множества позиций генов.

Функция приспособленности или функция оценки (fitness function) представляет собой меру приспособленности конкретной особи в популяции. Это очень важное понятие. С помощью функции оценки можно оценить степень приспособленности конкретных особей и выбрать лучшие в соответствии в заданными критериями "естественного отбора". Название функции тоже происходит из генетики. В связи с ее важностью и в области генетических алгоритмов, она должна иметь четкое и корректное определение, а в задачах оптимизации функция оценки, наоборот, максимизируется и называется целевой функцией. Для задач, чьей целью является минимизация, функция приспособленности преобразуется и проблема сводится к максимизации.

С помощью функции приспособленности оцениваются особо генетического алгоритма на каждой итерации, только после этого создается новое поколение особей, имеющее, что логично, более высокие значения для функции приспосабливаемости.

Пространством поиска называется множество потенциальных решений задачи. Точка пространства поиска, также фенотип или набор значений параметров является решением, принадлежащим к этому множеству. Наилучшим или оптимальным будет решение, оптимизирующее функцию.

2.2. Операторы генетических алгоритмов

Эволюционная электроника применяет концепции генетических алгоритмов к эволюционным схемам. Основная идея этой области исследований заключается

в том, что каждая электронная схема может быть представлена как индивид или хромосома эволюционного процесса, который выполняет стандартные генетические операции по цепям. Из-за широкого охвата области исследователи сосредоточены на различных проблемах, такие как размещение, отображение полевых программируемых вентильных массивов (FPGA), оптимизация комбинационных и последовательных цифровых схем, синтез цифровых схем, синтез пассивных и активных аналоговых схем, синтез операционных усилителей и оптимизация размеров транзисторов. Большое значение имеют работы, сосредоточившие внимание на «внутренней» эволюции аппаратного обеспечения.

Схему автоматического проектирования на основе генетических алгоритмов можно представить в виде дерева, где: входы схемы будут соответствовать листьям, операторы будут ветками, а выход - корневым узлом.

Под генетическими операторами будут подразумеваться следующие операторы:

Оператор кроссовера или скрещивания - является основным методом генетических алгоритмов для генерации нового поколения индивидов. В нашем алгоритме во время выполнения оператора кроссовера, сначала выбираются пары хромосом из родительской популяции. Совокупность хромосом, которые должны стать родителями нового поколения, можно считать временной популяцией, отобранной в результате селекции и специально предназначенной для дальнейших преобразований. С помощью операторов кроссовера и мутации они объединяются в пары случайным способом с вероятностью скрещивания, которую можно определить и обозначить *p*. Для каждой пары отобранных родительских хромосом разыгрывается позиция гена (*локус*) в хромосоме. Локус определяет то, что мы назовём *точкой кроссовера*. Если длина каждой из родительских хромосом состоит из определённого числа генов, то отсюда легко понять, что точкой кроссовера будет натуральное число, имеющее значение меньше максимального числа генов родительской хромосомы. Создаются новые индивидуумы, очень похожие на родителей, это помогает сохранить хорошие гены, так что потомок является более оптимизированным, то есть алгоритм приобретает больше необходимых черт. На рисунке 2.2.1. показан процесс кроссовера.

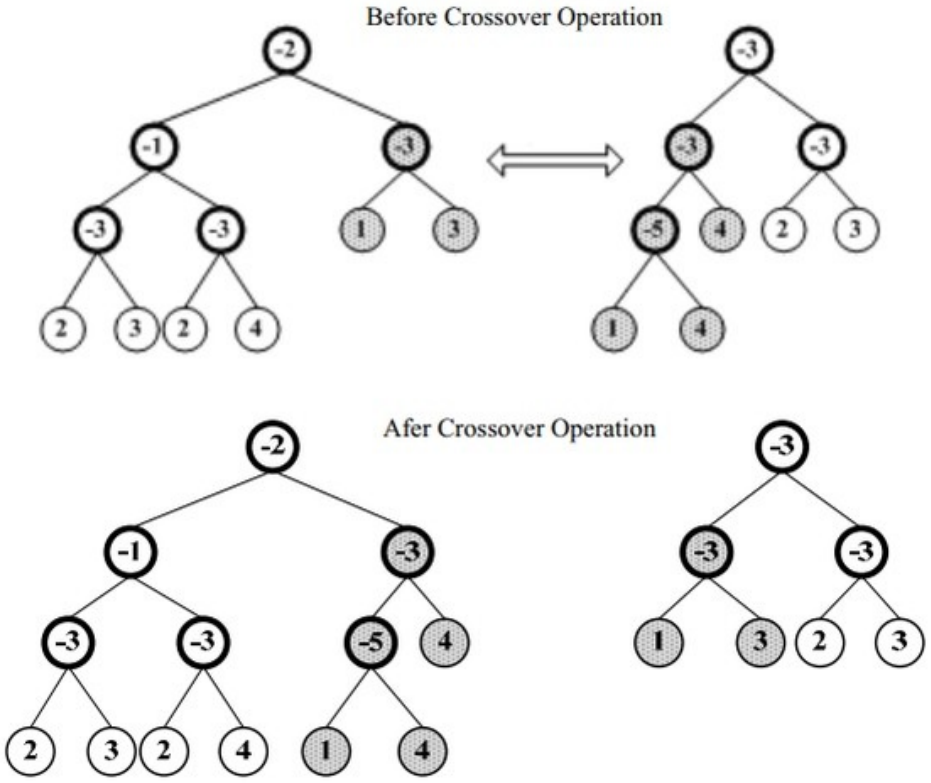


Рис.2.2.1

Есть много разновидностей оператора кроссовера. Самой простой является *одноточечный кроссовер*. Первым дело случайным образом выбирается одна из возможных точек разрыва — участков между соседними битами в строке. По выбранной точке обе родительских хромосомы разрываются на два сегмента, в итоге получится два потомка, склеенных из двух сегментов – по одному из кожа каждого родителя. На рисунке 2.2.2 изображение просто пример работы одноточечного кроссовера.

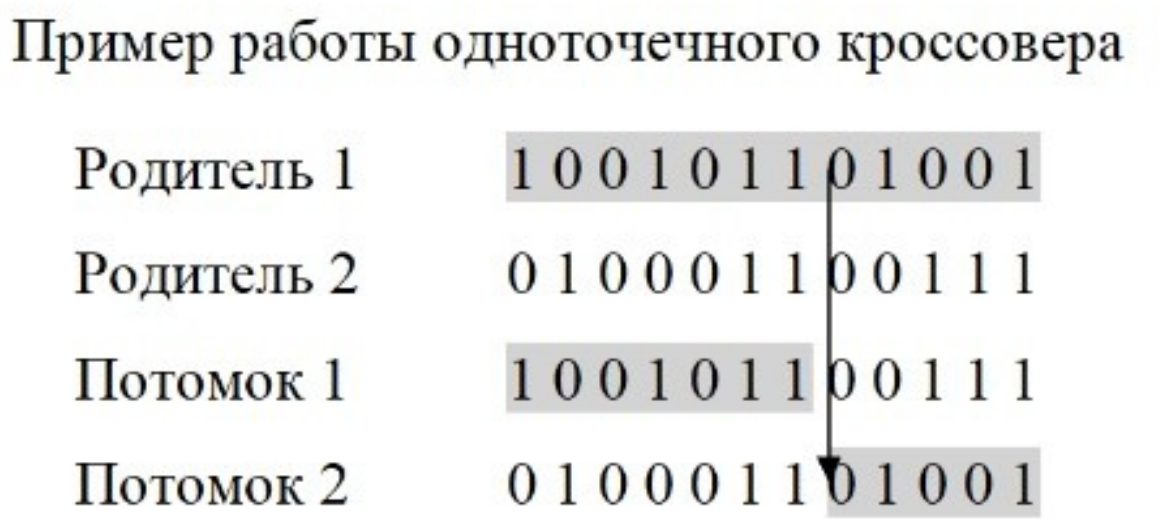


Рис.2.2.2.

Существуют и другие операторы кроссовера. Более развитыми альтернативными одноточечному кроссовера являются *двухточечный* и *равномерный* кроссоверы. В двухточечном имеется, очевидно, две точки разрыва у каждой из родительских хромосом, обмен происходит участками генов, находящимися между точками разрыва. В равномерном кроссовере каждый новый бит потомка случайным образом наследуется от одного из родителей, при этом второй потомок получает незадействованный бит другого родителя.

Оператор мутации в простейшей свое проявление меняет значение гена в хромосоме на противоположное. Это *одноточечная* мутация. Например, если мы имеем ряд [10010011], то при мутации на позиции 3 получим: [10110011]. Вероятность такой мутации выражается очень малыми значениями. Существуют и более сложные операторы мутации.

Во-первых, это мутация декремента: выбирается поддерево любой схемы и трансформируется в листовой узел вместо промежуточного - ветки. В корневом узле эта операция будет определена как недопустимая, потому что после мутации в цепи остался только один узел.

Вторая мутация инкрементная. Выбирается узел листа из любой схемы и трансформируется в поддерево.

Третий - это прямая мутация: выбирается узел из любой отдельной схемы и трансформируется в новый узел. Если это листовой узел, затем он преобразуется в неконечный узел, если это неконечный узел, то мутирует в листовой узел. Для этого типа мутации необходимо, чтобы схема была составлена корректно, потому что в этом процессе может быть сгенерирован неучтенный операнд.

Примеры всех трех операторов мутации в том же порядке (декремент, инкремент и прямая мутация) изображены в виде деревьев на рис.2.2.3.

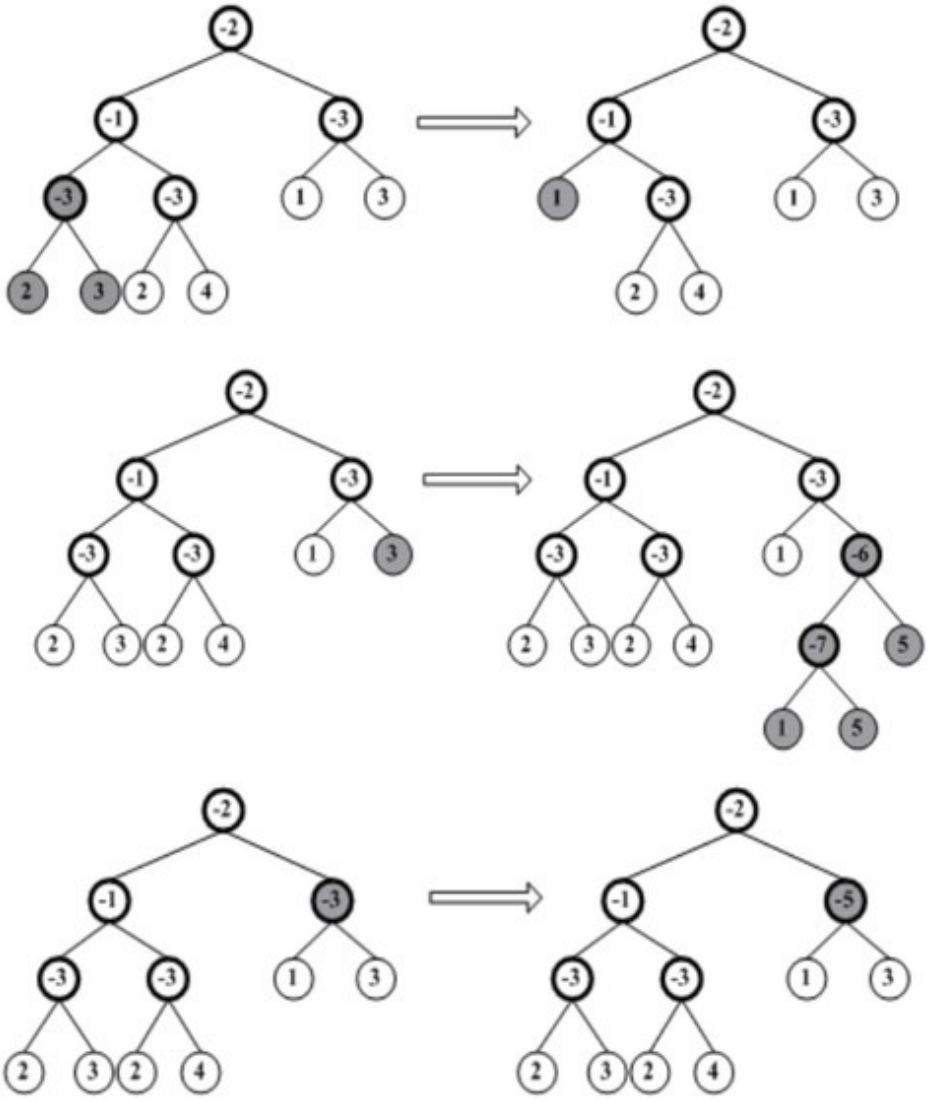


Рис.2.2.3.

Операторы *инверсии* и *транслокации* тоже являются операторами мутации. Инверсия — это перестановка генов в обратном порядке внутри выбранного

конкретного участка хромосомы. Например, в помощью оператора мутации инверсия из ряда [11*00111*0] можно получить ряд [11*11100*0].

Транслокация — это перенос участка хромосомы в другой сегмент той же хромосомы. Воздействуя этим оператором на оригинальный ряд из предыдущего примера, получим: [*00111*110].

С целью повышения приспособленности генов осуществляют *направленные мутации*, для чего после каждого изменения хромосомы проверяют, улучшилась ли в результате применения оператора мутации приспособленность, и если нет, действие откатывают — возвращают хромосому в исходное состояние.

Мера пригодности или функция приспособленности тоже может считаться одним из операторов генетического алгоритма. Фактически, это проверка того, выполняют ли эволюционные схемы желаемую логическую трансляцию входов в выходы. Проверка происходит с помощью запуска во все тестовые входы информации, затем сравнивают результаты с желаемыми функциональными возможностями побитово. Таблица истинности содержит целевую функцию и используется в качестве основы для сравнения. Процент всех правильных выходных данных в ответ на соответствующие входные данные затем используется в качестве критерия пригодности для генетического алгоритма, используемого для решения конкретной задачи. Другими словами, чем ближе эволюционная схема подходит к выполнению задачи с желаемой функциональностью, тем более развитой ее можно считать.

В наборе входных данных в таблице истинности есть один бит, который считается побочным и бесполезным для этого входа и отличается от желаемого значения. Когда суммарные выходные данные схемы равны ожидаемым выходным данным, тогда значение пригодности прибавляется 1.

В дополнение к таблице истинности, мы также должны учитывать количество ресурсов, используемых в схеме (в данном случае ссылаются на количество используемых логических элементов), поэтому конечная функция пригодности выглядит следующим образом:

$F = fit\text{-}N*к$, где:

F - коэффициент пригодности схемы,

N - номер логического элемента,

k - параметр, значение которого в данном случае 0,5,

fit - значение, которому соответствует схема в таблице истинности.

2.3. Особенности и достоинства генетических алгоритмов

Формирование популяции подразумевает исходное семейное сходство — в начальной популяции каждая из хромосом должна являться потенциальным решением задачи. При формировании нового поколения хромосом после применения операторов кроссовера и мутации, а также проверки приспосабливаемости нового поколения, необходимо решить следующую проблему: какие из новых особей станут частью следующего поколения, а также что делать с их родителями. Весьма животрепещущая проблема во многих областях.

Для ее решения существует два особенно распространённых способа.

Новые особи могут занимать места своих родителей. Затем круг повторяется: на новом этапе потомки будут оцениваться, отбираться, дадут потомство и уступят место своим «детям».

Второй способ заключается в том, что следующая популяция будет включать в себя как потомков, так и их родителей. При его применении нужно определить, какие из особей родителей и какие особи потомков попадают в новое поколение. Наиболее эффективным считается механизм *вынесения,* в ходе которого предполагается удалять из популяции похожие хромосомы и оставлять те, что сильнее отличаются.

Методы, упомянутые при описании операции отбора, также используются на каждом новом этапе работы алгоритма. Более подробно они описаны в третьей главе.

Остановка работы генетического алгоритма означает прекращение операционного процесса по некоторым признакам. Самый используемый вариант — пройдено заданное число итераций (циклов). Также критерием может являться длительное отсутствие улучшений значения лучшей приспособленности популяции или сближение худшего и лучшего значений в текущей популяции.

Одним из основных плюсов метода генетических алгоритмов является универсальность. Метод является лучшим, возможно, не во всех, но в подавляющем большинстве случаев поиска решений практических задач.

Существует несколько базовых элементов, которые вносят значимые различия между методом генетических алгоритмов и традиционными методами

оптимизации. Во-первых, при методе ГА обрабатываются не значения параметров задачи, а их закодированная форма. Во-вторых, генетические алгоритмы применяют вероятностей, а не детерминированные правила выбора. Далее, поиск решения осуществляется исходя из целой популяции, а не из единственной точки. И, конечно, нужно упомянуть, что для работы генетического алгоритма требуется знание только целевой функции, ее производные при этом не используются.

Исследуя современные работы и анализируя особенности генетических алгоритмов, можно сделать вывод, что к сегодняшнему дню этот метод доказал свою полезность и конкурентоспособность при решении сложных задач ч практических приложениях. Генетические алгоритмы способны решать задачи, формирующие сложную математическую структуру, совмещая при этом стохастические методы (методы нулевого порядка, «прямые») и градиентные (в которых используются производные, методы первого и второго порядков), а также динамическое и линейное программирование.

3. Сравнение результатов работы автоматического и ручного методов

3.1. Методы отбора в генетических алгоритмах

Рассмотрим в общих понятиях процесс работы алгоритма. В начале работы создается случайная начальная популяция. Популяция в генетическом алгоритме является набором хромосом, каждая из которых представляет из себя числовую строку фиксированной длины. Способ создания популяции может быть специфическим для каждой конкретной задачи, однако самым распространённым способом является случайная генерация строк.

Далее следует процесс отбора. Наиболее используемыми методами отбора являются метод рулетки, ранговый метод, элитизм и турнирный метод.

Метод *рулетки* заключается в том, что шанс выжить для каждой отдельной особи прямо пропорционален значению ее функции приспособленности.

При использовании *рангового* метода, шанс выжить прямо пропорционален рангу особи, под *рангом* подразумевается ее порядок в отсортированном по значению функции приспособленности списку хромосом.

Метод *элитизма* состоит в следующем: отбрасываются все особи, кроме заданной доли наиболее приспособленных.

При использовании *турнирного* метода выбираются две случайных особи. С вероятностью p выживает более приспособленная особь, соответственно, с вероятностью $1-p$ — менее приспособленная. Процесс повторяется до тех пор, пока не останется заданное количество особей.

3.2. Задачи и операторы генетического алгоритма

С полученными результатами отбора можно работать над дальнейшим решением задачи.

Объективно, по сравнению с генетическим программированием генетические алгоритмы крайне просты в реализации. Поддержка корректности выражений здесь имеет меньшее значение. При использовании генетического алгоритма также возможна поддержка типов, ведь в отличие от программирования с

экспрессией генов, в алгоритмах нет требования корректности, выражающегося в том, чтобы в строке с определённого места были расположены только терминалы — «листья» дерева, конечные узлы.

Эволюционное программирование (далее EHW – Evolvable Hardware) — это область использования эволюционных алгоритмов для автоматизированного проектирования специализированной электроники. Помимо эволюционных вычислений, под этими словами подразумеваются также реконфигурируемое аппаратное обеспечение, отказоустойчивость и автономные системы. Этот метод помогает получить желаемый результат быстро, дёшево и точно, при этом максимально удобно для выполнения данной задачи схемы, но при этом сам метод представляет собой нетривиальную задачу, требующую обширного набора специфических правил и ограничений.

Процесс воплощения идеи метода в реальность обычно включает следующие этапы:

1. Преобразование исходной логической спецификации в форму, подходящую для целевой технологии;

2. Минимизация и оптимизация представления с учетом пользовательских ограничений;

3. Проведение отображения технологии на целевое устройство;

4. Размещение и маршрутизация шлюзов компонентов, которые составляют конечный продукт.

Следует подчеркнуть, что на всех этих этапах необходимо соблюдать большую осторожность для поддержания логической функциональности исходной спецификации схемы. Следовательно, потребность в эффективных инструментах, которые выполняют некоторые из задач проектирования и позволяют разработчику сконцентрироваться на вопросах оптимизации производительности, имеет высокий приоритет реализации. Эволюционный электронный дизайн (ЭЭД) продемонстрировал высокую степень гибкости при решении сложных и сложных с вычислительной точки зрения задач.

Автоматизированное создание схем цифровой логики для удовлетворения спецификации функций является хорошо изученной областью. Синтезирование схемы с использованием спецификации функции является относительно простым процессом, однако оптимизация размера или производительности такой схемы представляет собой значительно более сложную проблему.

Существует очень мало исследований, которые фактически развивают функциональность последовательных логических схем. Эволюция аппаратного

обеспечения осуществляется путем внесения изменений, настройки функций элементарной ячейки и взаимосвязи между ячейками, а также выбора конфигураций элементов до тех пор, пока не будет достигнута целевая функциональность. Последовательные схемы делятся на чисто комбинационные блоки и регистры. Большие последовательные схемы обычно моделируются менее габаритными взаимодействующими конечными автоматами - finite state machine или FSM, которые определяются как математическая модель системы с дискретными входами, дискретными выходами и конечным числом внутренних конфигураций или состояний. Состояния системы полностью суммируют информацию, касающуюся прошлых входов в систему, которая необходима для определения ее поведения на последующих входах.

Генетический алгоритм (ГА) предназначен для имитации дарвиновской эволюции. Популяция кандидатов сохраняется и проходит через серию поколений. Для каждого нового поколения некоторые из существующих кандидатов дублируются, в то время как другие создаются по типу воспроизводства и мутации от набора генов родителей. EHW объединяет знания как ГА, так и логического дизайна эволюционных схем. С этой точки зрения, проблема реализации схем эквивалентна созданию черного ящика с «входами» и «выходами». Этот черный ящик должен быть таким, чтобы при исходном наборе входных сигналов были активированы желаемые выходы. Важной новой особенностью этой техники является то, что детали внутри черного ящика кодируются в хромосомы. Хромосома, представляющая цепь, подчиняется обычным процессам эволюционных алгоритмов.

Преимущество эволюционного программирования по сравнению с традиционным подходом к проектированию схем заключается в его способности к динамической и автономной адаптации. С помощью этого параметра можно изменять его структуру динамически (в режиме онлайн) и автономно в соответствии с изменениями в задании или в среде программирования. В последнее время достигнут значительный прогресс в развитии цифровых комбинационных логических схем. Эволюция последовательных логических схем значительно медленнее. Сложность соединений схем и кодирования хромосом для развития последовательной логической схемы может быть одной из причин того, что в этой области нет большого количества работ.

EHW только недавно был применен для синтеза последовательных логических схем. Несмотря на то, что ряд авторов внесли полезный вклад, данная область все еще находится на ранней стадии разработки. В таблице на рис.3.2.1 приведены последние работы, касающиеся эволюции последовательных логических схем. Анализируя таблицу, можно заметить, что EHW использовался

в основном для синтеза относительно небольших последовательных цепей. Аппаратное развитие использует примитивелогические шлюзы, которые недостаточно мощны для промышленного применения. Разработка больших схем остается важной задачей для исследователей EHW. Исследования в области EHW можно подразделить на две основные категории: внутренняя эволюция и внешняя эволюция.

Поэтому схемы, генерируемые EHW, оцениваются либо внешне (программная симуляция), либо внутренне. Внутренняя эволюция подразумевает, что схема загружается в реконфигурируемые аппаратные устройства и затем оценивается.

Проектирование синхронной последовательной схемы начинается с набора спецификаций и заканчивается логической диаграммой или списком булевых функций, из которых можно получить логическую диаграмму. В отличие от комбинационной логики, которая полностью определяется таблицей истинности, последовательная схема требует таблицы состояний для ее спецификации. Первым шагом в проектировании последовательностных схем является получение таблицы состояний или эквивалентного представления, такого как диаграмма состояний.

Author	Year	Type of sequential circuit	Evolving platform	Target application	Type of EHW
T Higuchi	1993	State transition graph		Digital logic circuit	Extrinsic
H Hemmi	1994	Digital sequential adder	AdAM system	Serial Adder	Extrinsic
A Thomson	1995	Dynamic state machine (DSM)	Simulator "Mr. Chip robot"	Robotics	Intrinsic
C. Manovit [10] and P. Chongstitvatana	1998 1999	Synchronous sequential logic circuits partial input/output sequence [10] and finite-state machine synthesis from multiple partial input/output sequences [3]	PLD, GAL	Frequency detector, Odd Parity Detector, Module-5 counter, Serial Adder	Intrinsic
C. Aporntewan et al.	2000	Learning finite state machine synthesis from partial input/ output sequences	PLD, FPGA	Serial Adder, 0101 Detector, Module-4 counter, Reversible 8-counter	Intrinsic

Рис.3.2.1.

Сложность логических схем может быть определена как функция от числа затворов в схеме. Основная идея этого подхода состоит в том, чтобы представить схемы достаточно совершенными, чтобы генетические операции могли быть выполнены. Архитектура генетического синтеза последовательных логических схем приведена на рис. 3.2.2. Этап 1 представляет собой спецификацию целевой схемы с использованием таблицы символьных переходов состояний. Минимизация состояния, если требуется, может быть выполнена с использованием уже существующих инструментов. На следующем этапе генетический алгоритм использует эту таблицу перехода состояний (state transition table - STT) для генерации оптимального назначения состояний для назначения двоичного кода каждого состояния. Поэтому STT последовательной схемы выглядит как двухуровневый логический файл. ГА используется для генерации назначения состояний с целью уменьшения площади цепи. Целевая функция ГА приводит к более простым уравнениям и, следовательно, к меньшей площади. Наконец, обработка генетического алгоритма для назначения состояния и EHW, используемое для разработки желаемой схемы, объединяются для получения оптимальной логической схемы. Этот комбинированный процесс приводит к четкому взаимодействию между компонентами. Внешнее EHW предполагается использовать для генерации комбинационной части последовательной логической схемы. Внешняя EHW использует программные модели для оценки функции устойчивости результирующей цепи. Генетический синтез создает схемы на простом уровне, используя набор функций логического устройства, такого как И, ИЛИ, НЕ и D-флип-флоп триггер.

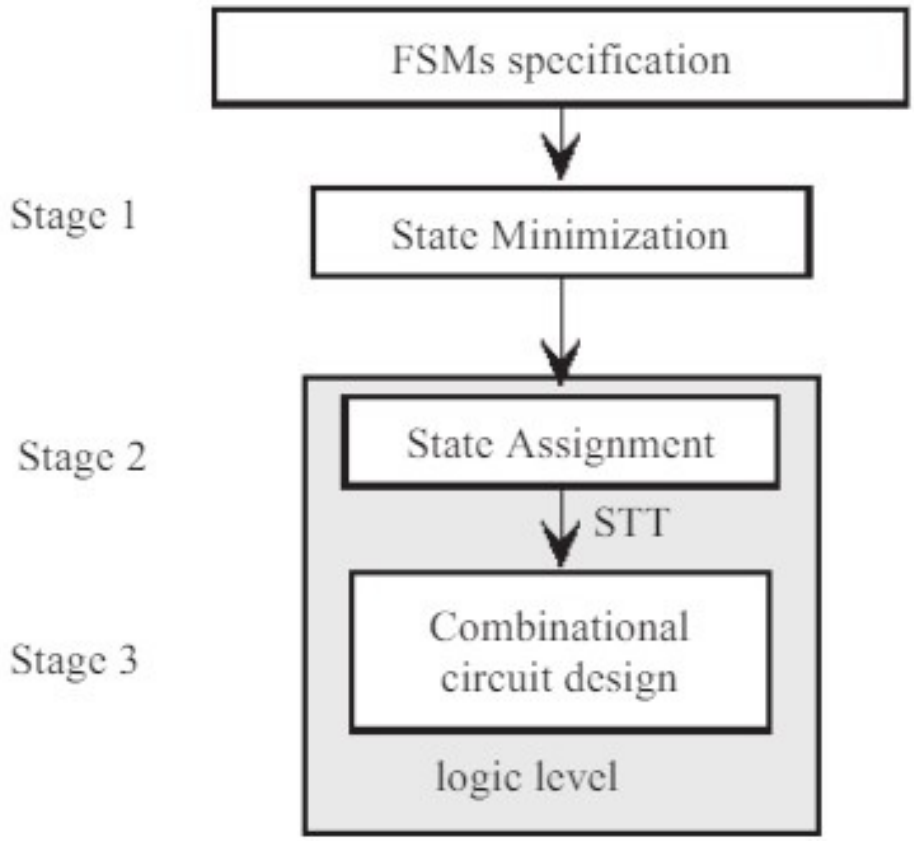


Рис. 3.2.2.

Поговорим о генерации таблицы переходов состояний STT. Первым этапом создания последовательной схемы является поиск оптимального состояния назначения, которое минимизирует количество компонентов. Единственное ограничение для допустимого назначения состояния минимального размера состоит в том, что каждому состоянию должно быть присвоено уникальное двоичное значение.

Общее количество возможных уникальных назначений для FSM определяется как A (n, b) = (2^b-1)!/(b!(2^b-n)!), где n - количество состояний, b - наименьшее целое число, которое равно или больше, чем $\log_2 n$. Количество отдельных назначений достаточно велико, чтобы препятствовать любым попыткам получить решение. В таблице на рисунке 3.2.3. представлены различные доступные подходы к проблеме назначения состояний. Кроме того, существует множество алгоритмов, реализованных с помощью SIS - «Системы для синтеза

последовательных цепей», таких как NOVA, которая нацелена на двухуровневую реализацию, и JEDI, которая нацелена на многоуровневую реализацию. Входными данными для системного синтеза FSM обычно является файл формата KISS (Keep Internal State Simple).

Summary of available approaches to find optimal state assignment

Approach	Theory	Merits	Comments
Partition Theory Hartmans and Stearns 1966 [19].	Algebraic techniques are used to decompose the state machine.	The state assignment is based on closed partitions resulting in reduced dependency between the state variables.	Not all machines have close partitions.
Column evaluation approach. Dolota and McCuskey 1964 [20].	The column of state table are scored with respect to various criteria to influence quality of assignment	This approach gives very good realisation, the result even better than approach [19].	It is intractable for machine of more than 12 states.
Enumerative approach of Story 1972 [12].	All possible partitions are evaluated as candidates for assignment separately by calculating the complexities of the corresponding Boolean function.	All partitions are optimally selected and the subset of best partition equally matching is selected for assignment.	The implementation of approach is slow.
Approach of Moroz 1970 [21].	The graph embedding created directly from the state graph whose edges corresponding to oriented transition between states.	The approach can be treated as approximate solution to quadratic assignment.	It doesn't solve the quadratic assignment but simply edge embedding problem.
Quadratic assignment approach. DeMicheli 1984 [22].	This approach is based on embedding some graphs created from FSM table to hypercube graphs.	This approach permits for realisation of machine up to 100 states.	The approach still can't be applied for machine with more than 100 states, and it doesn't take in to count output states for assignment.
Devadas "Mustang 1988" [23].	Commercial ECAD tools look for optimal state assignment.	Available	For multilevel implementation.
Approach of Villa "NOVA 1990" [15].	Commercial ECAD tools look for optimal state assignment.	Available	For two level implementation only.
"Genetic algorithm approach" Almaini and Amaral, 1995 [17, 18].	Using GA to generate optimal state assignment.	All possible assignment for a given problem maybe considered as search space of potential solution.	GA approach requires a set of operators for its implementation. The selection of these operators is crucial for the efficiency of the algorithm.
Proposed approach 2003	Using GA to generate optimal state assignment and conventional part of the circuit.	Combination of both GA and EHW for sequential logic circuit.	

Рис.3.2.3.

В последние годы несколько исследователей применяли метод генетических алгоритмов для решения проблемы назначения состояний. Не существует полностью удовлетворительной ручной методики для определения оптимального состояния. ГА находит хорошие оптимальные решения, если не считать полного перечисления и оценки всех возможных состояний.

3.3. Хромосомное представление для задачи назначения состояний ГА

Конечный автомат может быть описан с помощью STT или графа перехода состояний (STG). При реализации конечных автоматов они обычно представляются в виде двумерных таблиц с числом строк, равным количеству состояний, и количеством столбцов, равным размер входного алфавита. Пересечение между состоянием и входным алфавитом содержит выходной символ и переход следующего состояния. Хромосома представляет автоматы в

виде списка состояний. Длина хромосомы равна числу состояний, используемых последовательностной схемой. Начальная популяция генерируется случайным образом. Каждая хромосома представляет собой одно из решений проблемы. Дублированные хромосомы отбрасываются. Чтобы закодировать актуальную информацию, FSM представляется в виде списка из n состояний; i-й элемент списка - это число в диапазоне от 1 до (2^b-i+1).

Рассмотрим пример, показанный на рисунке 3.3.1, где генотип хромосомы был сгенерирован случайным образом. Генотип проблемы представлен массивом целых чисел. На рисунке показано, как кодируется машина из шести состояний. Согласно рисунку 2, метод может быть обобщен следующим образом:

а) Случайная функция используется для генерации шести целых чисел (2, 4, 5, 2, 4, 2).

б) Порядковые целые числа представляют состояния в виде списка (0, 1, 2, 3, 4, 5, 6, 7); список начинается с нуля и содержит все возможные назначения состояний FSM с использованием кода минимальной длины, то есть $b=[\log_2 n]$ битов для кодирования набора из n состояний. Алгоритм работает через таблицу определения статуса выживаемости, где единица - положительный результат. Числа отсчитываются слева направо.

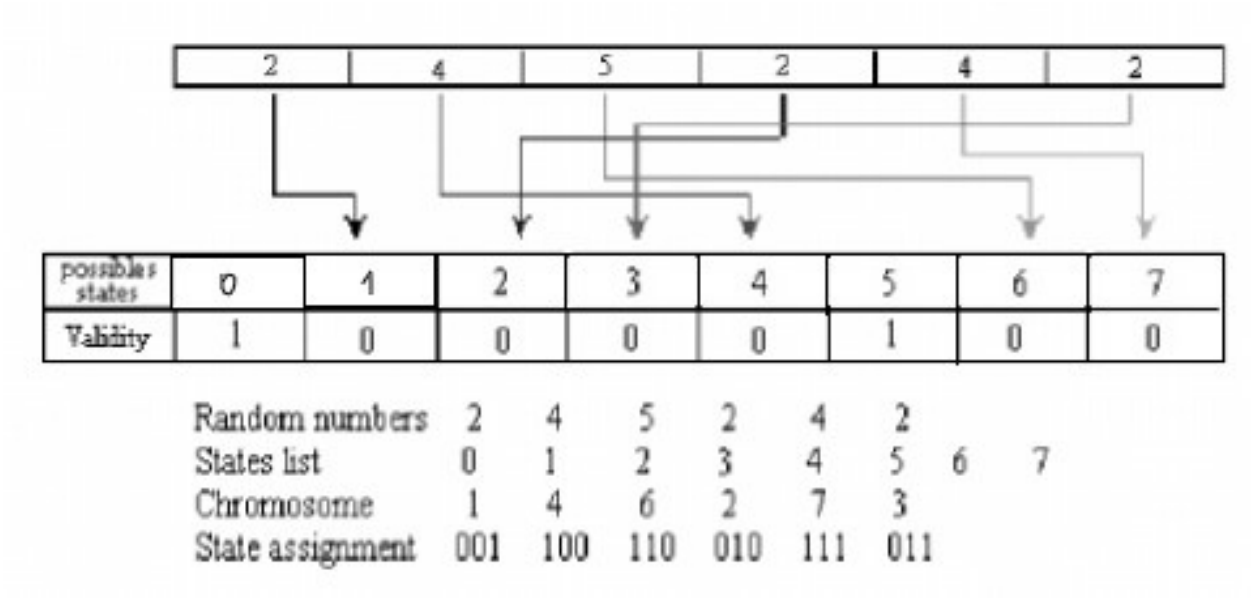


Рис. 3.3.1.

Процедура интерпретируется следующим образом:

1) Первое случайное число равно 2, возьмем второе число из списка возможных состояний 1 в качестве первого кода для начального состояния и установим достоверность 0, чтобы оно не использовалось для последующего отбора;

2) Следующее случайное число 4, возьмем четвертое число из списка возможных состояний и удалим его из списка, установив достоверность 0;

3) Следующим случайным числом будет 5, возьмем пятое число из списка возможных состояний и установим достоверность 0. Процедура продолжается аналогичным образом для остальных чисел в списке. Из рисунка видно, что случайные числа 2, 4, 5, 2, 4, 2 сопоставили бы состояния 0, 1, 2, 3, 4, 5, 6, 7 с назначением 1, 4, 6, 2, 7 и 3 соответственно, присвоив уникальный код каждому состоянию.

Целью ГА является извлечение оптимального назначения состояний для конечного автомата, для которого требуется наименьшее количество логических элементов. Поэтому число логических вентилей с двумя входами И, ИЛИ используется для определения функции приспособленности.

3.4. Внешний эволюционный аппаратный подход для комбинационного логического проектирования

Внешние EHW были использованы для разработки комбинационной части схемы. Подход EHW - это недавно разработанная методика, используемая для синтеза комбинационных и последовательностных схем. Автоматизированное проектирование цифровых систем использует как программную симуляцию, так и программируемые аппаратные методы. Следовательно, подход EHW основан на идее объединения реконфигурируемых аппаратных устройств с ГА для автономного выполнения реконфигурации. Таблица перехода состояний (STT) была выбрана для описания поведения синхронной последовательной логической схемы. Структура последовательностных логических схем, показанных на рисунке 3.4.1, включает в себя набор из двух секций комбинационной логической схемы и D-флип-флоп-операций (DFF).

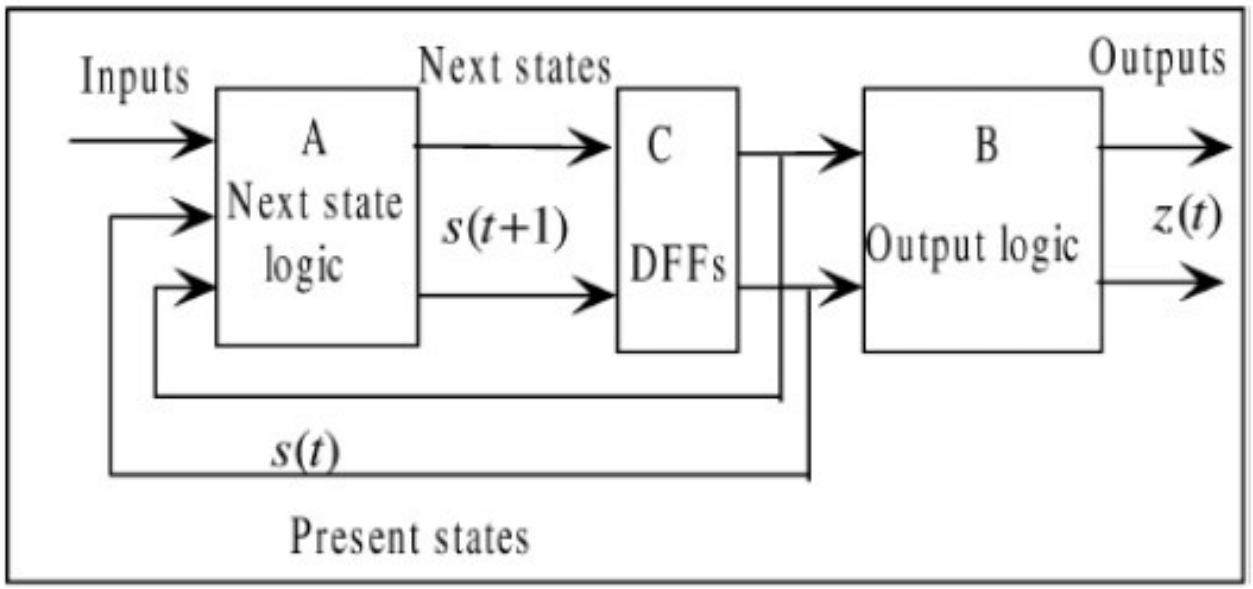


Рис.3.4.1.

Схема генерируется с использованием заданного набора доступных логических элементов. Комбинационные части последовательностных логических схем генерируются с использованием внешнего подхода EHW. Требуемая функциональность для комбинационных частей логической схемы описана с использованием STT. Пространство поиска определяется рядом различных факторов:

1) Тип строительных блоков, представленных в рамках;

2) Количество логических элементов, используемых для генерации схемы;

3) Приложение, для которого развивается схема.

В процессе проектирования давно признано, что лучший способ решить проблему - разложить проблему на несколько подзадач. Структура последовательностной логической схемы в предлагаемом подходе содержит 3 подсхемы, как показано на рисунке 4. Каждая подсхема развивается отдельно. После того как подсхемы были спроектированы, последовательная схема можно объединять. Этот подход был описан в данной работе в обобщенном виде здесь для удобства.

Каждая комбинационная схема представлена в виде прямоугольного массива логических элементов. Каждый логический элемент в этом массиве является незафиксированным и может быть удален из сети, если он окажется избыточным. Генотип характеризуется следующими параметрами:

1) Количеством столбцов и числом строк,

2) Параметром обратного уровня,

3) Списком связности прямоугольного массива логических элементов.

Пояснения параметров изображено на рисунке 3.4.2.

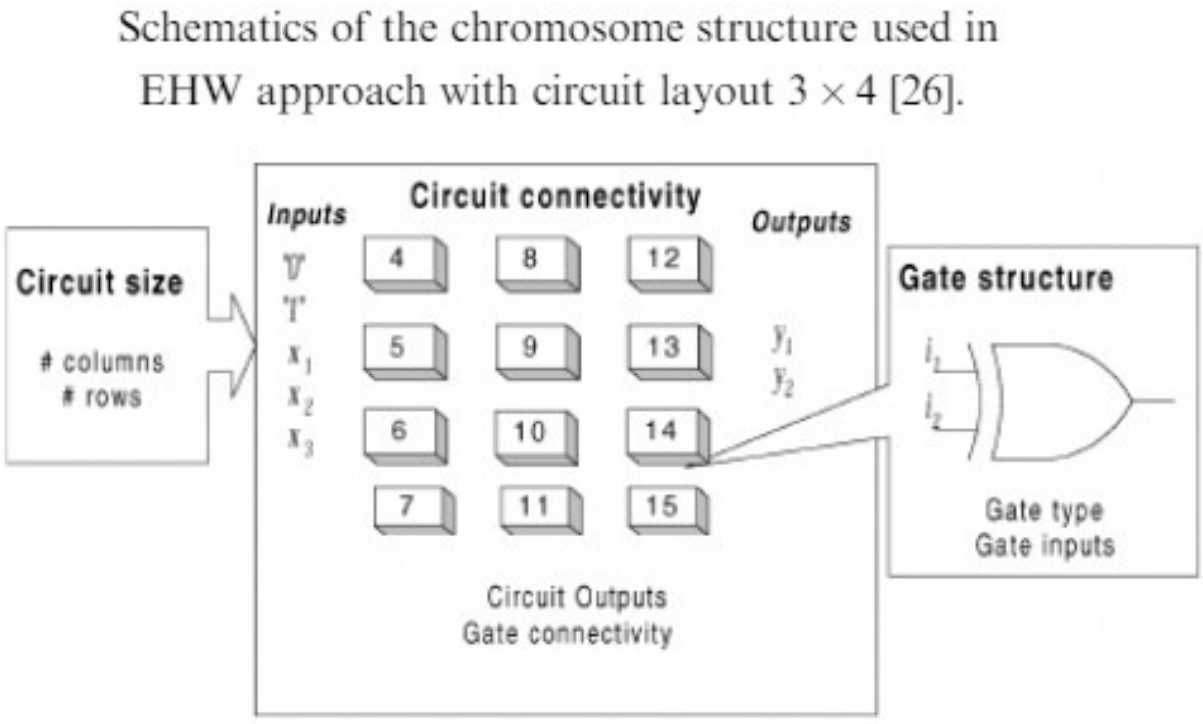


Рис.3.4.2.

Первые два параметра - это всего лишь размерность прямоугольного массива, а третий - параметр, который управляет внутренней связностью логической схемы. Максимальное соединение с ячейкой может быть достигнуто, если число строк равно единице, а параметр обратного уровня равен числу столбцов. В то же время, если число строк равно единице, а параметр обратного уровня равен единице, то каждая ячейка должна быть подключена к своему непосредственному соседу слева. Кроме того, ячейки в любом конкретном столбце не могут быть соединены друг с другом, и каждый логический элемент имеет два входа и один выход.

Хромосома определяет соединение в сети между первичными входами и первичными выходами. На рисунке 3.4.2 показано представление хромосомной связи между тремя первичными входами и двумя первичными выходами. Сеть спроектирована с использованием логических вентилей с двумя входами и

одним выходом. Расположение хромосом - 3x4 ($n_{столбцов}$ x $m_{строк}$). Это геометрия незанятых логических ячеек, а также нумерация списка соединений. Каждая логическая клетка представлена тройкой целых чисел [$c^1c^2c^3$], где c^1 определяет функциональный ген, а c^2, c^3 определяют входные данные.

Функцией приспосабливаемости в данном случае будет функция динамической устойчивости *(F1 + F2)*. Она используется для оценки схемы. *F1* использует расстояние Хэмминга для измерения функциональности схемы между заданным набором входов и выходов. *F2* определяет количество примитивных логических ячеек, которые используются в схеме. F2 активируется, когда F1 достигает 100% функциональности. Первая функция сравнивает соответствующий выходной сигнал подсхемы A и подсхемы B (рисунок 3.4.1) с заданными столбцами следующего состояния STT и выхода. Процент правильных битов следующего состояния, соответствующих *j*-му выходу, вычисляется следующим образом:

$$Fy_j = \left(\sum_{j=1}^{p} |y_j - d_j|/p\right) * 100$$

Рис. 3.4.3.

Здесь | y_j- d_j | представляет собой абсолютную разницу между фактическим выходом следующего состояния и требуемым выходом d_j,

y_j - вектор выходов *j*-й схемы,

p - количество комбинаций входов в данной логической функции.

Схема полностью реализует выход y_j и достигает 100% функциональности. Как только решение было разработано, активируются критерии оптимизации цепи. Вторая функция F2 минимизирует количество логических элементов, определяя в эти схемы наименьшее количество активных логических элементов. Процедура, описанная выше, применяется как к подсхеме A, так и к подсхеме B.

Попробуем рассмотреть процесс проектирования последовательностной схемы, исходя из синтеза таблицы символических переходов, приведенной на рисунке 3.4.4.

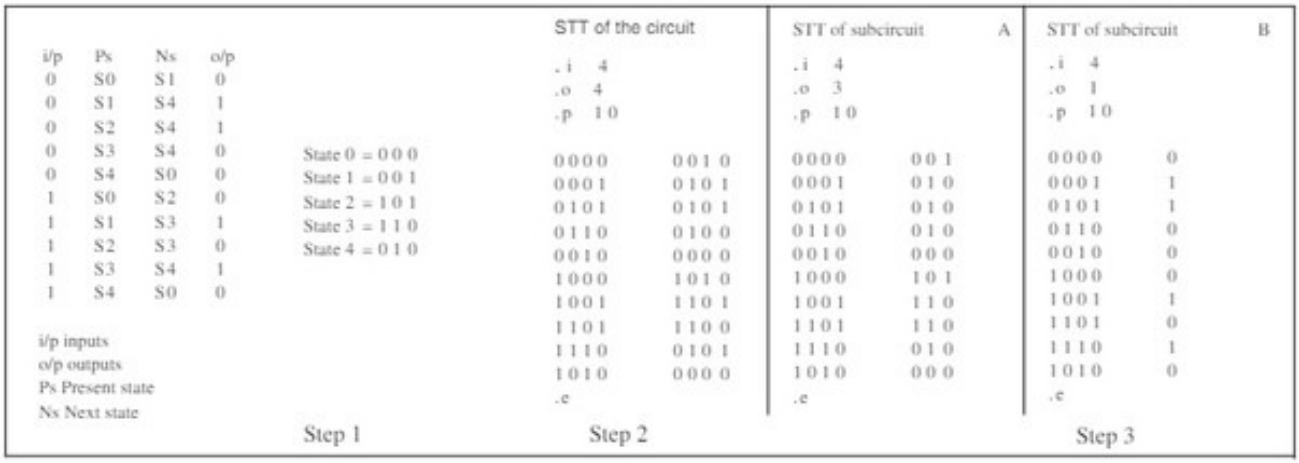


Рис. 3.4.4.

FSM в данном случае имеет пять состояний, один вход и один выход. На рисунке 3.4.4 шаг 1 показывает таблицу символических состояний FSM, и назначение состояния, сгенерированное генетическим алгоритмом, присвоено каждому состоянию. На втором этапе кодирование используется для получения стандартного двухуровневого формата PLA. На третьем шаге схема делится на входную комбинационную логическую схему A и выходную комбинационную логическую схему B. Как только разложение завершено, получаем возможность сгенерировать полнофункциональные схемы с использованием EHW. Исходные данные параметров эволюционных алгоритмов приведены на рисунке 3.4.5.

Functional set of logic gates used in EHW

Gene	Function gene	Gene	Function gene	Gene	Function gene	Gene	Function gene
0	"0"	4	!a NOT (a)	8	!ab AND (!a, b)	12	!a|b OR (!a, b)
1	"1"	5	!b NOT (b)	9	!a!b AND (!a, !b)	13	!a|!b OR (!a, !b)
2	"a" wire	6	ab AND (a, b)	10	a|b OR (a, b)	14	a ^ b XOR (a, b)
3	"b" wire	7	a!b AND (a, !b)	11	a| !b OR (a, !b)	15	!a ^ !b XOR (!a, !b)

Рис.3.4.5.

Полученная в этом примере эволюционная схема показана на рисунке 3.4.6.

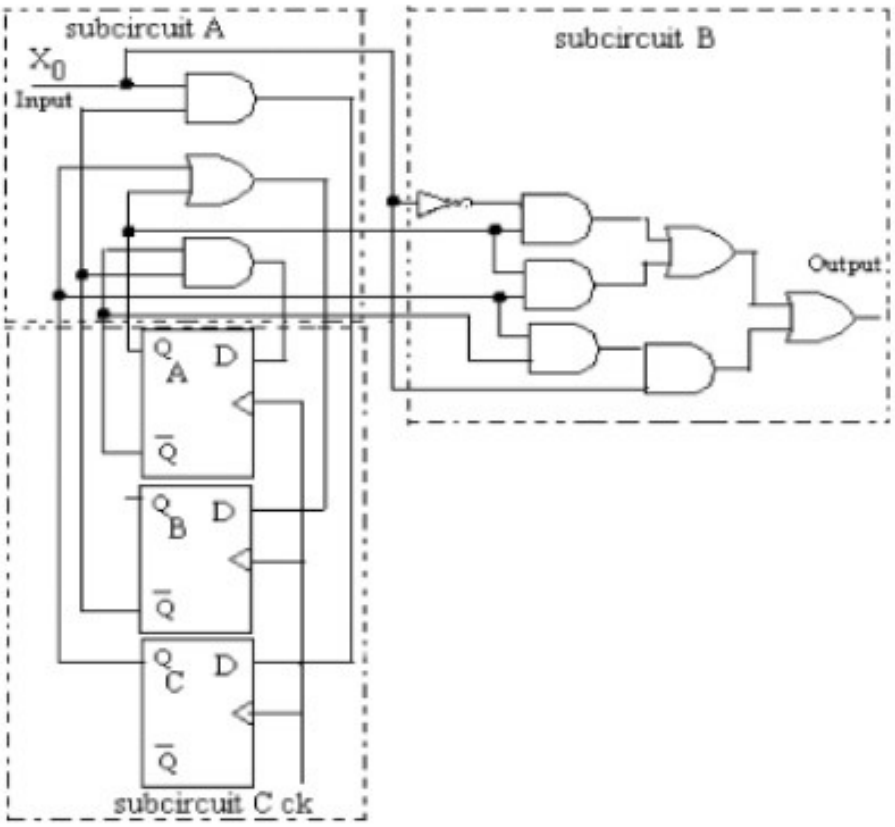


Рис.3.4.6.

Общее количество логических элементов в собранной схеме - десять (6 двухвходовых И, 3 двухвходовых ИЛИ, 1 НЕ). Наиболее эффективная и развитая подсхема состоит из 3 логических элементов в схеме A, 7 элементов в схеме B и 3 D-флип-флоп-операций. Начальные параметры ГА для генерации оптимального назначения состояния (SA) для построения STT и внешнего EHW для проектирования требуемых логических схем приведены в таблице на рис.

Initial parameters used to evolve the circuit

Parameters	SA	EHW
Population size	20	10
Number of generations	100	50000
Number of GA runs	10	100
Crossover rate	0.25	0.6
Crossover type	Two-point	Uniform
Mutation rate	0.015	0.05
Circuit layout	—	3×4

3.4.7.

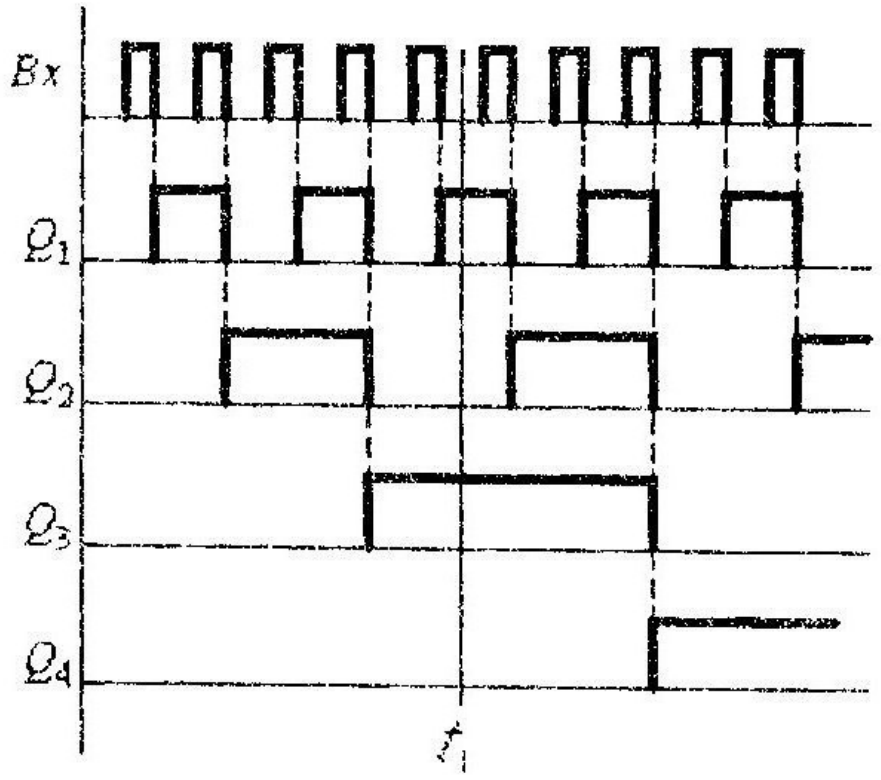
Рис. 3.4.7.

3.5. Результаты эксперимента и сравнительный анализ

В этом пункте главы рассмотрим структуру схемы, созданной с использованием вышеописанного подхода, а также сравним полученные результаты в другими проектами, созданными с помощью традиционных методов, то есть вручную. Для простоты понимания разложил проблемы проектирования цифровой логической схемы на несколько пунктов. Таблица, представленная на рисунке 3.4.7, где описаны параметры работы генетического алгоритма, используется для решения задачи назначений состояний, а выбор подходящих параметров EHW проводится экспериментально.

3.5.1. Четырехмодулевый счетчик

Самой простой цифровой системой можно представить двоичный счётчик. Таблицей истинности двоичного счётчика является последовательность двойных чисел *0—2^n-1, n* — разрядность счётчика, соответственно, состояние каждого разряда показывает определённый столбец таблицы. Модуль счета — это максимальное число единичных сигналов, которое может быть принято счётчиком. Пример обработки сигнала двоичным счетчиком показан на рисунке 3.5.1.



р

Рис.3.5.1.

Счетчик с числом модуля 4 имеет, что логично, четыре внутренних состояния. Каждое состояние соответствует числу в двойной системе. На рисунке 3.5.2 показаны состояния, а также идентифицировано оптимальное назначение с использованием метода генетических алгоритмов.

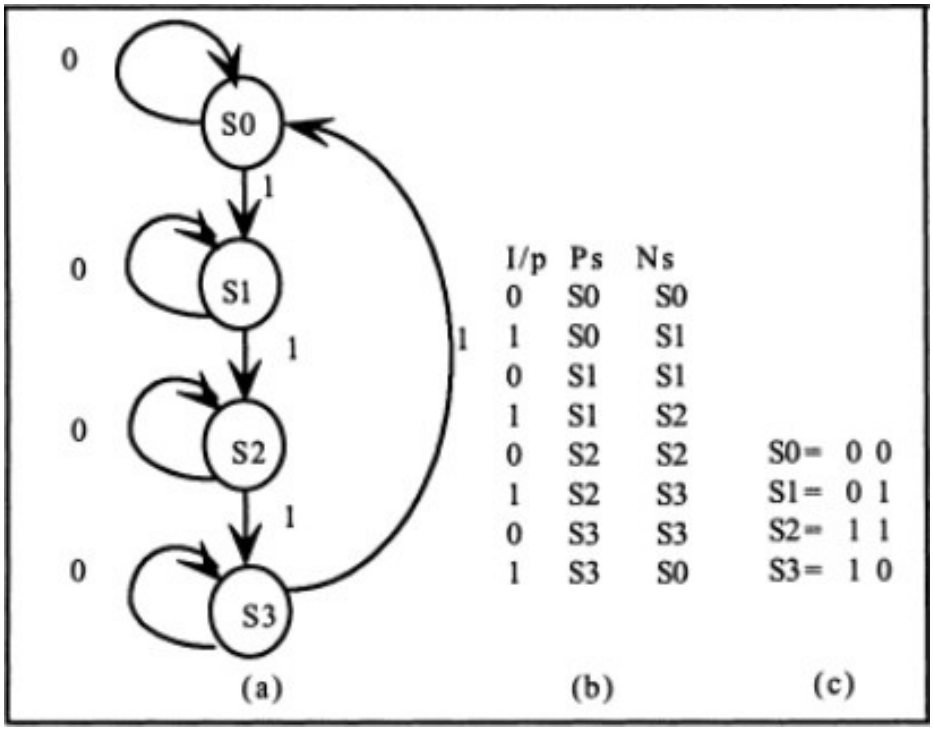


Рис. 3.5.2.

Вспомним параметры, необходимые для создания генетического алгоритма в данной задаче: размер популяции составляет 5 особей, максимальным числом генерации заявлено 50000, общее число протонов равняется 100, частота применения оператора кроссовера 0,6, частота применения оператора мутации – 0,05 (следует вспомнить, ранее говорилось, что вероятность мутации выражается малыми числами, следовательно, частота мутации может быть, как здесь, на порядок ниже частоты кроссовера), разметка будущей схемы разная для двух подходов: 1*10 в предлагаемом подходе A и 4*4 в предлагаемом подходе B. Анализируя эту информацию и вводные данные, можем составить уравнения. Полученная система показана в таблице на рисунке 3.5.3.

Solution obtained for Module-4 counter produced using proposed method and manual design

Proposed approach (a)	Proposed approach (b)	Manual method
$D_A = \bar{X}_0 A + X_0 B$	$D_A = \bar{X}_0 A + X_0 \bar{B}$	$D_A = \bar{X}_0 A + X_0 \bar{B}$
$D_B = X_0 \bar{A} + \bar{X} B$	$D_B = X_0 A + \bar{X}_0 B$	$D_B = X_0 A\bar{B} + \bar{X}_0 B + X_0 A$
Subcircuit A = 7 gates	Subcircuit A = 7 gates	*Subcircuit A = 9 gates*
Subcircuit C = 2-D flip-flops	*Subcircuit C = 2 D flip-flops*	*Subcircuit C = 2 D flip-flops*

Рис. 3.5.3.

Для вышеописанного примера взята совсем небольшая численность популяции, но используется большое количество поколений. С помощью двух различных разметок, но с одинаковыми наборами назначения состояний созданы схемы, представленные на рисунке 3.5.4.

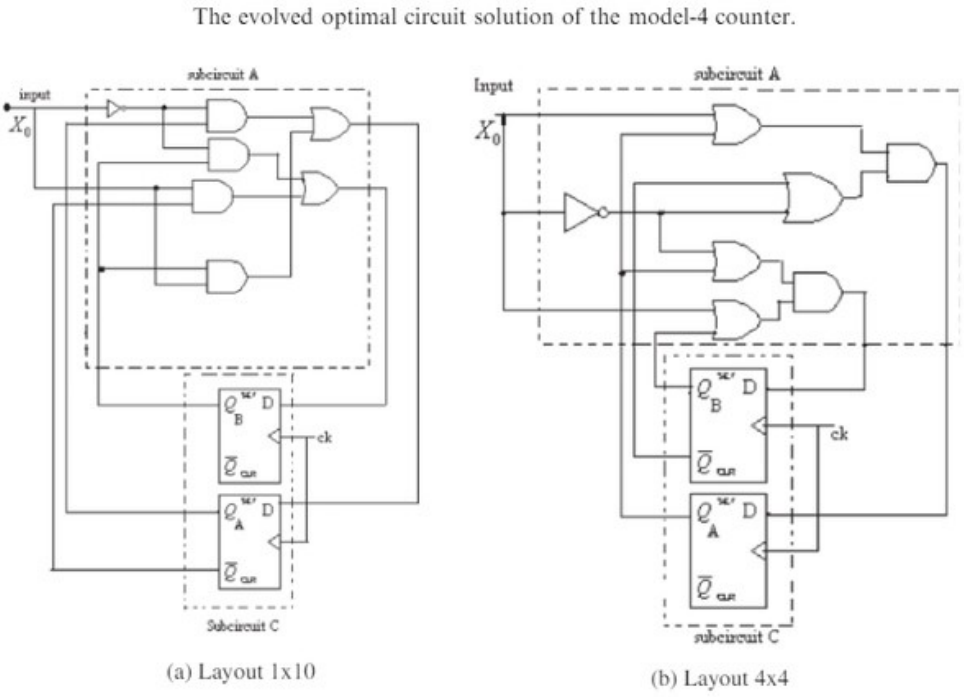


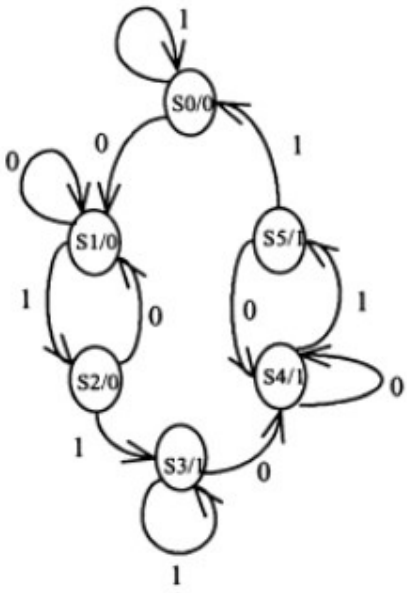
Рис.3.5.4.

В некоторых случаях выбор разметки может быть проблематичным: слишком маленькая разметка схемы даёт большую вероятность, что не будет достигнуто 100% функциональности, ведь в этом случае невозможно построить схему заданной функциональности, исходя из малого количества логических элементов; слишком большая разметка даст алгоритму множество избыточных возможностей работы, что увеличит время прогона и затраты мощности.

Исходя из рисунка 3.4.4 можно заметить, как изменение разметки влияет на состав и количество логических элементов в схеме. Нужно обратить внимание на то, что в части b для оптимального решения с заданной разметкой 4x4 потребовалось два двухвходовых И, 4 двухвходовых ИЛИ и 1 НЕ.

3.5.2. Детектор последовательности сигналов

В такой схеме уже 6 внутренних состояний, а ее поведение описано рисунков 3.5.5.



I/p	Ps	Ns	O/p
0	S0	S1	0
1	S0	S0	0
0	S1	S1	0
1	S1	S2	0
0	S2	S1	0
1	S2	S3	0
0	S3	S4	1
1	S3	S3	1
0	S4	S4	1
1	S4	S5	1
0	S5	S4	1
1	S5	S0	1

(b)

S0	000
S1	010
S2	001
S3	100
S4	110
S5	101

(c)

Рис. 3.5.5.

В данном случае детектор настроен на последовательность *011*. Когда на вход поступает эта последовательность, на каждом выходе мы имеем значение 1, которое не изменится до тех пор, пока на входы детектора вновь не поступит последовательность *011,* после чего все выходы складываются в 0, пока искомая последовательность не появится снова. Сравнивая результаты, полученные с помощью ручных расчётов и с помощью метода генетических алгоритмов, можно составить таблицу, представленную на рисунке 3.5.6.

Solutions obtained for sequential detector produced by proposed approach and manual method

Proposed approach	Manual method
$D_A = XB$	$D_A = A\bar{C} + A\bar{X} + BC\bar{X}$
$D_B = \bar{X}$	$D_B = BX + \bar{A}CX$
$D_C = XA\bar{C} + \bar{X}C + \bar{A}C$	$D_C = BX + \bar{A}\bar{C}\bar{X} + \bar{A}\bar{B}\bar{X} + A\bar{C}X$
$Z = C$	$Z = A + BC$
Subcircuit A = 8,	Subcircuit A = 17,
Subcircuit B = 1	Subcircuit B = 2
Subcircuit C = 3 D flip-flops	Subcircuit C = 3 D flip-flops

Рис. 3.5.6.

Видно, что подсхема A может быть сгенерирована с помощью восьми простых логических элементов: 5 И, 2 ИЛИ, 1 НЕ и подсхемы C, состоящей из 3 D-флип-флоп-триггеров. На рисунке 3.5.7 показана реализованная с помощью этих элементов схема.

Evolved optimal circuit solution of the sequence detector with circuit layout 4 × 5.

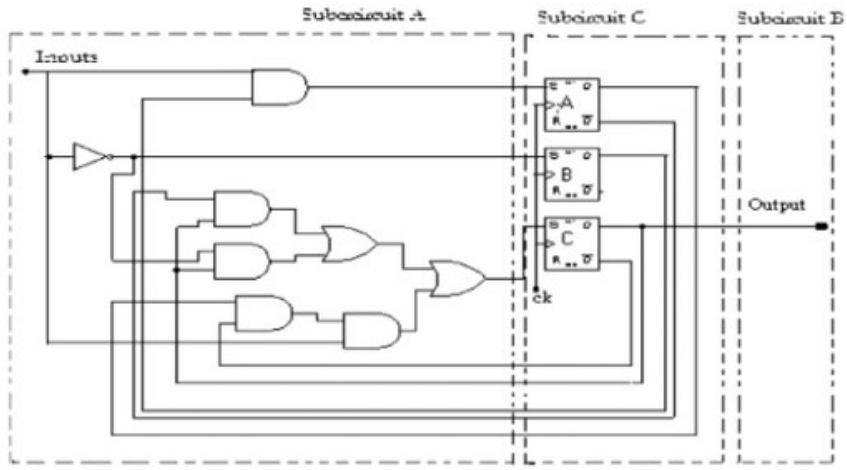


Рис. 3.5.7.

В качестве ручного метода здесь использовался метод Карт Карно и случайное назначение состояний. Для генетического алгоритма выбраны следующие параметры: частота мутации 0,05, численность популяции 20, число поколений не превосходит 50000. Разметкой схемы является структура из четырех строк и пяти столбцов, параметр обратного уровня 5 –

максимальный. Такое значение параметра увеличивает вероятность достижения правильного решения практически до 100%.

Сравнивая полученную методом генетических алгоритмов схему, можно понять, что она является более оптимизированной, чем схема, полученная при решении той же задачи ручным методом, то есть, имеет более эффективную структуру.

3.5.3. Детектор 1010

Схема такого детектора имеет 1 вход, 1 выход и 4 внутренних состояния. Она изображена на рисунке 3.5.8.

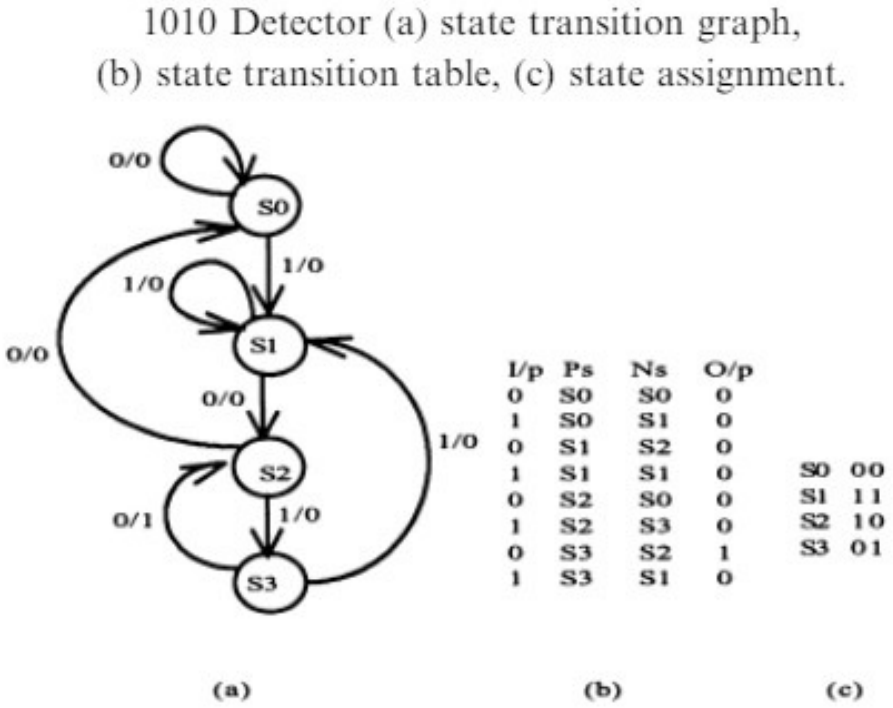


Рис. 3.5.8.

Результаты, полученные предлагаемым способом, сравниваются с результатами работы ручного метода, как показано в таблице на рисунке 3.5.9.

Solution obtained for 1010 detector produced using proposed method and manual design

Proposal approach	Almaini, 1994 [1]
$D_A = X\bar{B} + A$	$D_A = \bar{X}\bar{A}B + \bar{X}A\bar{B} + XAB$
$D_B = X$	$D_B = \bar{A}B + A\bar{B} + X\bar{B}$
$Z = \bar{X}\bar{A}B$	$Z = \bar{X}A\bar{B}$
Subcircuit A = 2	Subcircuit A = 12
Subcircuit B = 3	Subcircuit B = 2
Subcircuit C = 2 D flip-flops	Subcircuit C = 2 D flip-flops

Рис. 3.5.9.

Решение, которые мы здесь выбрали в качестве ручного метода, использует почти в 3 раза больше вентилей, чем схема, произведенная подходом генетического алгоритма. Стоит обратить внимание: выходы реализуются одинаково для обеих цепей. Схема фактически развивалась путем разделения каждой подсхемы и развития каждой подсхемы отдельно для получения конечного результата. Вероятность мутации клеток равна 0,05, размер популяции составляет 20, а число поколений – 50000 при разметке схемы 1×10. Полученная в разметке 4×5 приведена на рисунке 3.5.10.

Evolved optimal circuit solution for 1010 detector using 4 × 5-circuit layout.

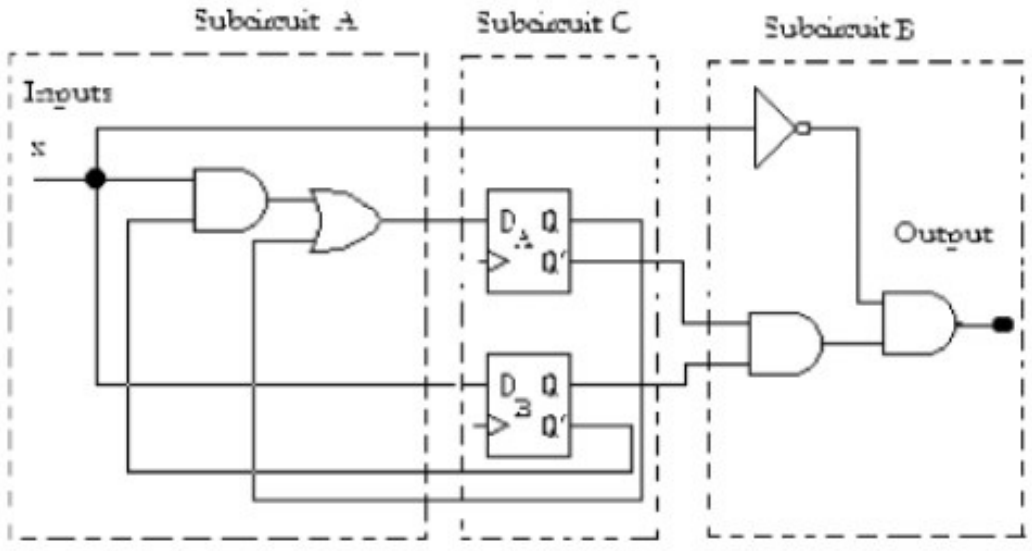


Рис. 3.5.10.

3.5.4. Эталонные тесты MCNS

Рассмотрим результаты эксперимента в другой точки зрения: применим их к набору схем, выбранных из эталонных тестов MCNS (multimedia cable network system, стандарт MCNS). В таблице на рисунке 3.5.11 прописаны назначения состояний, сгенерированные с помощью метода генетических алгоритмов.

State assignments generated by GA

FSM	#State	State assignment
bbara	10	2,3,5,4,7,8,9,0,1,11
bbtas	6	6,1,5,4,3,0
dk15	4	0,2,1,3
dk16	27	12,8,1,27,13,28,14,29,0,16,26,9,2,4, 3,10,11,17,24,5,18,7,21,25,6,20,19
dk27	7	6,1,5,7,4,3,0
dk512	14	4,3,14,9,12,7,2,1,0,10,13,8,5,6
lion9	9	1,0,4,6,7,53,1,11
shiftreg	8	6,2,4,0,7,3,5,1
tav	4	4,0,2,1

Рис. 3.5.11.

Процесс EHW берет старт со случайным образом выбранных логических элементов, постепенно все больше развивая целевую функциональность. В каждым новым поколением индивидов увеличивается значение целевой функции. Так как даже выбирая самые универсальные алгоритмы для достижения высокой функциональности, мы не можем гарантировать 100% выполнение схемой заданий конкретной задачи, для уверенности возьмём не конкретное значение, а среднее после 100 прогонов.

В таблице, представленной на рисунке 3.5.12, описаны результаты эксперимента, полученные для общих FSM (напомним: машина конечных

состояний или конечный автомат) из MCNC-тестов. В таблице показано число элементов, которые использовались для улучшения каждой подсхемы после 100 прогонов программы. Окончательный набор логических элементов определения заранее, но каждый может как использоваться один раз или несколько, так и остаться неиспользованные в данной подсхеме. Отсюда можно сделать вывод об очевидном преимуществе такого подхода: логические элементы схемы будут выбраны из любого, заранее определённого набора.

Experimental results of extrinsic EHW approach. 100 cases is the number of fully functional solutions obtained after 100 runs of GA

	Specification				Estimation of the best obtained solution					SIS [5]	CPU Time (S)
Benchmark. Kiss	#in	#out	#stat	Functional set	Subcircuit A	Subcircuit B	Subcircuit C	Total	#100 cases	Total	t_{EHW}
Bbara	4	2	10	0–5,6,10,15	32	28	3	60	7	79	660
Bbtas	2	2	6	2–7,10,11,15	15	4	3	19	24	28	480
dk15	3	5	4	0,1,6,7,10,11–15	20	33	2	53	11	66	300
dk16	2	3	27	0,6,8,10–15	265	40	5	305	1	285	3180
dk27	1	2	7	0–6,10,15	11	5	3	16	28	20	420
dk512	1	3	14	2–6,9–14,15	25	22	4	47	31	58	484
Lion9	2	1	9	0–6,10,15	29	21	4	50	7	35	410
Shiftreg	1	1	8	0–13,15	13	5	3	18	21	12	100
Tav	4	4	4	0–6,10,15	3	23	2	26	9	29	360

Рис. 3.5.12.

С помощью подхода генетических алгоритмов можно синтезировать компактные и необычные структуры схем. Качество эволюционной схемы определяется в самом простом случае количеством элементов в ней. Из приведённой выше таблицы можно видеть, что для больших тестов FSM (это dk16) довольно сложно получить за сто прогонов одно действительное решение. Полученные результаты мы сравним с результатами SIS (SIS, system for sequential circuit synthesis, инструмент для синтеза и оптимизации последовательностных цепей), чтобы получить возможность дальнейшего логического синтеза и оптимизации. Входные данные для SIS предоставляются в формате таблицы состояний. В результате работы программы получим список элементов, которые следует использовать для получения оптимизированной конечной схемы.

Крайний правый столбец таблицы на рисунке 3.5.12 представляет показатель времени вычислений t_{EHW}, при этом он записан в результате работы компьютера с частотой 450 МГц, имеющего 128 Мб оперативной памяти. Анализируя эти результаты, можно заметить, что тест EHW с малым количеством состояний требовал меньших временных затрат на работу процессора.

Кроме этого, видно, что результаты работы этого метода не хуже, а в некоторых случаях превосходят результаты, полученные с помощью ручных методов обработки.

Заключение

Основной темой работы является автоматическая оптимизация схем комбинаторной логики. Схемы делятся на комбинационные (выход зависит от входных данных на текущий момент) и последовательностные (имеющие память). Можно сделать следующие выводы касательно реализации в настоящее время и перспектив в области генетического программирования и генетических алгоритмов в частности:

1. Сложность схемотехнических решений современных аналоговых и цифровых систем требует применения алгоритмов оптимизации при их проектировании с целью минимизации количества элементов, временных затрат и подбора их параметров.

2. Существующие алгоритмы оптимизации, например, карты Карно и метод МакКласски, либо ограничены количеством входов-выходов, либо не обладают свойствами минимизации.

3. Тщательная подборка параметров ГА (функций отбора, кроссовера, мутаций и способа отбора особей в каждую следующую популяцию) позволяет получить автоматический метод оптимизации, работающий за короткие временные сроки с разнотипными критериями оптимизации одновременно, то есть метод многопараметрической оптимизации.

4. В практической части работы были интегрированы схемы делителя частоты и детектора последовательности в двух вариантах: ручным методом карт Карно и методом генетического алгоритма, исходя из подбора параметров для алгоритма.

5. Результаты работы метода рассмотрены в сравнении, сделан вывод о целесообразности использования и дальнейшего развития метода генетических алгоритмов в связи с очевидными плюсами его использования: оптимизация за малое время работы программы, возможность изменения параметров, таких как максимальное число прогонов, численность начальной популяции и минимальное значение результирующей функции приспособленности, а также многопараметрическая оптимизация.

Список литературы

1. B. Ali, Evolutionary Algorithms and Their Use in the Design of Sequetial Logic Curcuit / A. E. A. Almaini, T. Kalganova / Genetic programming and Evolvable Machines / Kluwer Academic Publishers, 2004. – 25 p.
2. Д. М. Харрис, Цифровая схемотехника и архитектура компьютера / С. Л. Харрис, - Morgan Kaufman, 2013. – 1662 с.
3. Забродним Ю.С., Промышленная электроника: Учебник для ВУЗов. - М.: "Высшая школа", 1982. – 496 с.
4. T. Kalganova, Circuit layout evolution: an evolvable hardware approach / R. Miller / Colloquiumon ‘‘Evolutionary hardware systems,’’ IEE Colloquium Digest: London, UK, 1999. – 46 p.
5. T. Villa, NOVA: state assignment of finite state machines for optimal two level logic implementation / A. Sangiovanni-Vincentelli / IEEE Trans., C-9, pp. 905–924, 1990.
6. Zebulum R. S., Evolutionary Electronics: Automatic Design of Electric Circuits and Systems by Genetic Algorithms/ M. A. Pacheco, M. M. Vellasco. / CRC Press, 2002. – 302 p.
7. С. Осовский, Нейронные сети для обработки информации / пер. И. Д. Рудинского, Москва: «Финансы и статистика», 2004. – 344 с.
8. Рутковская Д., Нейронные сети, генетические алгоритмы м нечеткие системы / М. Пилиньский, Л. Рутковский / пер. И. Д. Рудинского, Москва: Горячая линия-Телеком, 2006. – 452 с.
9. З. Мишалевич, Генетические алгоритмы + Структуры данных = Эволюционная программа / Speinger-Verlag, 1992.
10. J. Amaral, K. Tumer, and J. Ghosh, Design genetic algorithm for the state assignment problem / IEEE Trans, vol. no. SMC-25, 4, 1995. – 730 p.
11. Дьяконов В. П. MATLAB 6.5. SP1/7/7 SP1/7 SP2 + Simulink 5/6. Инструменты искусственного интеллекта и биоинформатики / В. П. Дьяконов, В. В. Круглов, — М.: СОЛОН-ПРЕСС, 2006. – 456 с.
12. Cohoon J. P., Genetic Algorithms in Engineering Systems – The Institute of Electrical Engineers / Paris W. D.. – London, 1997. – 276 p.

13. Каширина И. Л., Введение в эволюционное моделирование: учебное пособие. – Воронеж: Издательский дом «Воронежский государственный университет», 2007. – 40 с.
14. S. Louis, Genetic algorithm as computational tool for design / PhD Dissertation, Department of Computer Science, Indiana University, 1993.
15. J. Hartmanis and E. Stearns, Algebraic Structure Theory of Sequential Machines / UK, Prentice Hall, 1996.
16. T. Dolotta and E. McCluskey, ‘‘The Coding of Internal States of Sequential Machines,’’ IEEE Transaction on Electron. Comput, vol. EC-13, 1964. – 767 p.
17. C. Aporntewan, An on-line evolvable hardware for learning finite-state machine / P. Chongstitvatana / in Proceeding of Int. Conf. on Intelligent Technologies, Bangkok, December 13–15V. Kreinovich and J. Daengdej (eds.), 2000. – 38 p.
18. S. Yang, Logic synthesis and optimisation benchmark user guide version 3.0, MCNC, 1991. – 45 p.
19. E. M. Sentovich, «SIS. A system for sequential circuit synthesis» / Tech. Rep. UCB/ERLM92/41 Electronics Research Lab. University of California: Berkeley, CA 94720, 1992. – 153 p.
20. A. Thompson, ‘‘An evolved circuit, intrinsic in silicon, entwined with physics,’’ in Proceedings of the1st International Conference on Evolvable Systems: From Biology to Hardware (ICES96), Lecture notes in Computer Science, T. Higuchi, et al. (eds.) / Springer-Verlag, vol. 1259, 1997. – 502 p.