УДК 681.518

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ ПОСТРОЕНИЯ ИДЕНТИФИЦИРУЩИХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ДЛЯ ЦИФРОВЫХ СХЕМ С ПАМЯТЬЮ

Иванов Д.Е., к.т.н., с.н.с., доцент
Институт Прикладной Математики и Механики НАН Украины
E-mail: *ivanov@iamm.ac.donetsk.ua*

Abstract

In the life cycle of development of digital circuits developer need to solve the several problems of building of input sequences: test, initializing and verifying of equivalence. In this paper a generalized approach to solving this problem is presented. It based on the genetic algorithm and avoids some problems that are intrinsic to the traditional deterministic methods.

1. Введение.

В настоящее время средства автоматизированного проектирования позволяют эффективно разрабатывать цифровые схемы, содержащие десятки и сотни тысяч логических вентилей, а также тысячи элементов состояний. Перед разработчиком кроме непосредственно задачи синтеза структуры схемы стоят проблемы построения целого ряда входных последовательностей: тестовой последовательности для определения исправности схемы; инициализирующей последовательности, необходимой для перевода схемы в начальное состояние; последовательностей верифицирующих эквивалентность двух заданных схем с целью определения корректности процесса их оптимизации. Авторами уже предложены генетические алгоритмы построения данных последовательностей. Целью данной работы является обобщение этих алгоритмов в единый подход, позволяющий решать класс подобных задач.

Для всех указанных выше задач построения входных последовательностей в своё время зарубежными и отечественными авторами были предложены детерминированные и псевдослучайные алгоритмы их решения. Для задачи построения тестовых последовательностей разработаны структурные методы, являющиеся расширением для последовательностных схем метода ветвей и границ [1], методы, основанные на символьном моделировании [2], а также псевдослучайные методы [3].

Аналогично, для задачи построения инициализирующих последовательностей предложен точный метод, основанный на построении деревьев решений и технике символьного преобразования [4-5]. Ряд методов построения инициализирующих последовательностей были внедрены непосредственно в алгоритмы генерации тестов и отдельно не описывались.

Для задачи верификации эквивалентности двух схем также разработаны алгоритмы, основанные на преобразованиях функционального описания схемы [6], символьном поиске в ширину на деревьях решений [7].

Для всех указанных выше алгоритмов можно отметить тот факт, что они либо имеют низкую эффективность (псевдослучайные методы), либо эффективно применяются для схем малой и средней размерности (до нескольких тысяч логических вентилей). Использование данных алгоритмов для схем большой размерности затруднено из-за проблем переполнения памяти и стеков возвратов.

Ранее авторы разработали ряд алгоритмов для решения указанных задач, которые основаны на генетическом подходе [8-10]. В данной статье предлагается обобщить разработанные ранее методы конструирования разного типа входных последовательностей с

целью построения унифицированного подхода, который основан на генетическом алгоритме. Решение задачи в такой постановке в литературе не имело места.

Данная статья организована следующим образом. Во введении обосновывается актуальность проводимых исследований. Второй раздел посвящён описанию модели, используемой при исследовании и постановке задач. В третьем разделе описывается общая структура генетического алгоритма построения входных последовательностей, а также применяемое кодирование особей и популяций, схемы отбора и т.д. В четвёртом разделе приведены примеры конструирования фитнесс-функций для построения входных последовательностей с разными заданными свойствами: тестовых, инициализирующих и верифицирующих эквивалентность. В пятом разделе рассматривается практическая программная реализация классов предложенных генетических алгоритмов. В заключении делаются выводы и указываются направления дальнейших исследований.

2. Формальная модель и постановка задач.

В качестве модели в данной работе используются синхронные последовательностные схемы (рис.1). В данной модели схема представляется в виде комбинационного блока (в свою очередь состоящего из нескольких функциональных комбинационных блоков, КФБ) и блока памяти, который состоит из D-триггеров. Далее также будем использовать следующие обозначения: *#Bx* – число внешних входов схемы, размерность вектора v; *#Вых* – число внешних выходов схемы, размерность вектора Y; *#Tp* – число элементов состояния (D-триггеров) схемы, размерность вектора T.

Рис.1. Модель синхронной последовательностной схемы.

Вектор v – упорядоченное множество двоичных значений, которое подаётся на вход цифровой схемы в определённый такт времени. Последовательность s_i заданной длины l_i - упорядоченное множество из l_i векторов, которые подаются на вход схемы в последовательные такты времени. Обозначение v_{ij} говорит, что мы рассматриваем в последовательности s_i вектор с номером j ($j = 0,...,l_i - 1$).

При моделировании используется преобразование синхронной последовательностной схемы в псевдокомбинационный эквивалент с дальнейшим его итеративным моделированием в последовательные такты времени. Для такого преобразования удаляют элементы состояний. Входы элементов состояний (вектор T_i на рис.1) при этом называются псевдовыходами, а их выходы (вектор T_{i-1} на рис.1) – псевдовыходами.

При таком преобразовании работа блока памяти представляется следующим образом. В момент смены тактового импульса в устройстве синхронизации (для упрощения на рисунке не представлен) происходит подача на вход схемы новых входных значений v_i момента времени i, на псевдовходы подаются значения псевдовыходов схемы в предыдущий такт времени T_{i-1}, после чего путём моделирования комбинационной части схемы для такта времени i формируются выходные сигналы схемы Y_i и сигналы псевдовыходов T_i.

Рассмотрим постановку трёх проблем построения различных типов входных последовательностей.

1) Построение тестовых последовательностей.

Пусть задано исправное устройство A_0 и конечное множество его неисправных модификаций $A = \{A_1, A_2, ..., A_n\}$, и $A_0 \neq A_i$ для $i \in \overline{1,n}$. Тестом, проверяющим заданную неисправность, назовём такую входную последовательность s, выходные реакции на которую устройств A_0 и A_i различны. Тестом, проверяющим все неисправности множества A, называется такая входная последовательность, которая является проверяющей для всех $A_i \in A$. Отметим, что для проверки того факта, что последовательность s является проверяющей, необходимо моделирование работы $n+1$-й схемы: одна исправная схема и n схем с неисправностями. Для моделирования работы неисправных схем из множества A необходимо выбрать класс модельных неисправностей. Традиционно мы используем класс одиночных константных F_{const} ($const0$ и $const1$) неисправностей для всех линий схемы. Также для уменьшения времени работы алгоритмов класс неисправностей сжимается с учётом их эквивалентности [11] с получением сжатого класса неисправностей $\tilde{F}_{const}$. Тогда $|A| = |\tilde{F}_{const}|$. Отметим здесь также, что в рассматриваемой постановке внутреннее представление устройства A_0 и всех элементов множества A в виде логической сети одинаково.

В качестве меры качества тестовой последовательности выбирается полнота теста: отношение числа проверенных неисправностей m к общему числу неисправностей $n = |A|$:

$$P = \frac{m}{n} \cdot 100\% \qquad (1)$$

2) Построение инициализирующих последовательностей

В данной работе мы будем рассматривать задачу логической инициализируемости схемы. Задача состоит в нахождении одиночной последовательности s, которая переводит схему из неопределённого состояния в некоторое известное состояние.

Дадим более формальное определение. Пусть Q - множество всех состояний последовательностной схемы, тогда $Q = \{0,1,X\}^{\#Tp}$ (0,1,X – элементы трёхзначного алфавита моделирования схемы). Пусть S - множество всех возможных определённых состояний схемы, состоящей из $\#Tp$ триггеров: $S = \{0,1\}^{\#Tp}$. Пусть также Σ - множество всех возможных входных последовательностей s_i. Очевидно также, что для выбранного трёхзначного моделирования $S \subset \Sigma$. Пусть $x \in Q$ начальное неопределённое состояние. Функция $F : Q \times \Sigma \to Q$ обозначает все состояния, достижимые схемой при поступлении на её вход последовательностей из Σ при использовании трёхзначного алфавита моделирования. Тогда, некоторая последовательность $s \in \Sigma$ называется инициализирующей для заданной схемы, если в финальном состоянии $q = F(x,s)$ все состояния определены, т.е. $q \in S$.

3) Построение последовательностей, верифицирующих эквивалентность схем.

При решении данной задачи с помощью генетического алгоритма она, по существу, сводится к доказательству несуществования входной последовательности, различающей две заданные схемы. Фактически ищется контрпример, показывающий, что схемы различны.

Формально постановка задачи выглядит следующим образом. Заданы две цифровые схемы A_0 и A_1. Необходимо построить такую входную последовательность $s \in \Sigma$, выходные реакции на которую схем A_0 и A_1 различны.

Отметим разницу с постановкой задачи в случае 1). При построении тестовых последовательностей предполагается, что логическая сеть, представляющая внутреннюю

структуру A_0 и элементов из A идентичны. В данной же постановке задачи схема A_1 получена результате некоторой оптимизации внутренней структуры схемы A_0, т.е. вообще говоря структура их элементов состояний одинакова, тогда как логические сети, описывающие комбинационные блоки, различны.

3. Общая схема генетического алгоритма.

В общем случае генетический алгоритм представляет собой итерационный процесс построения текущей популяции особей из некоторой начальной [12]. Для того, чтобы полностью задать (построить) генетический алгоритм, необходимо определить следующие его элементы: кодирование особей и популяций, вид фитнесс-функции и способ её вычислений, генетические операции выбора, скрещивания и мутации. Для всех трёх поставленных выше задач авторы ранее разрабатывали генетические алгоритмы [8-10]. Общими для них элементами являются: кодирование особей и популяций, а также генетические операторы. Напомним кратко их сущность.

Общим для всех трёх задач является поиск некоторой входной последовательности, обладающей заданным свойством. Исходя из этого, выбирается следующее кодирование особей и популяций (рис.2). Число бит в каждом двоичном наборе (ширина последовательности) соответствует числу входов схемы ($\#Bx$) и является неизменной величиной в алгоритме. Каждый входной набор соответствует одному такту работы схемы, а число таких наборов, входящих в последовательность, определяет длину особи. Эта величина может изменяться в процессе работы алгоритма под воздействием генетических операторов. В качестве популяции выступает набор особей (входных последовательностей). В нашем алгоритме число особей в популяции (её размер) является величиной неизменной.

Исходя из выбранного кодирования, используются следующие операции скрещивания: вертикальное и горизонтальное (рис.3). Операции скрещивания применяются к выбранным особям с некоторой вероятностью P_c. При вертикальном скрещивании разрезание особей идёт по входам схемы. Для каждого столбца входной последовательности особи-потомка случайным образом определяется столбец, какого из родителей необходимо скопировать. При горизонтальном скрещивании, которое выполняется по входным наборам (тактам времени) в каждой особи-родителе выбирается одна точка скрещивания. Особь-потомок формируется из начала последовательности одного родителя и конца последовательности второго родителя.

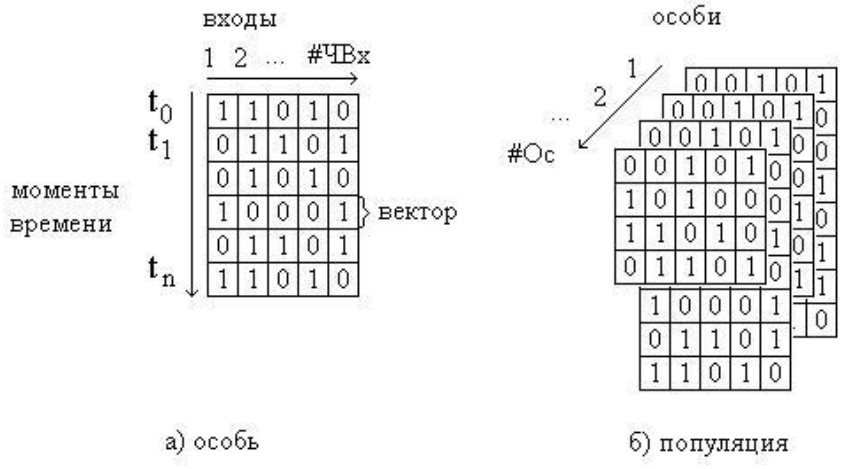

Рис.2. Кодирование особей и популяций.

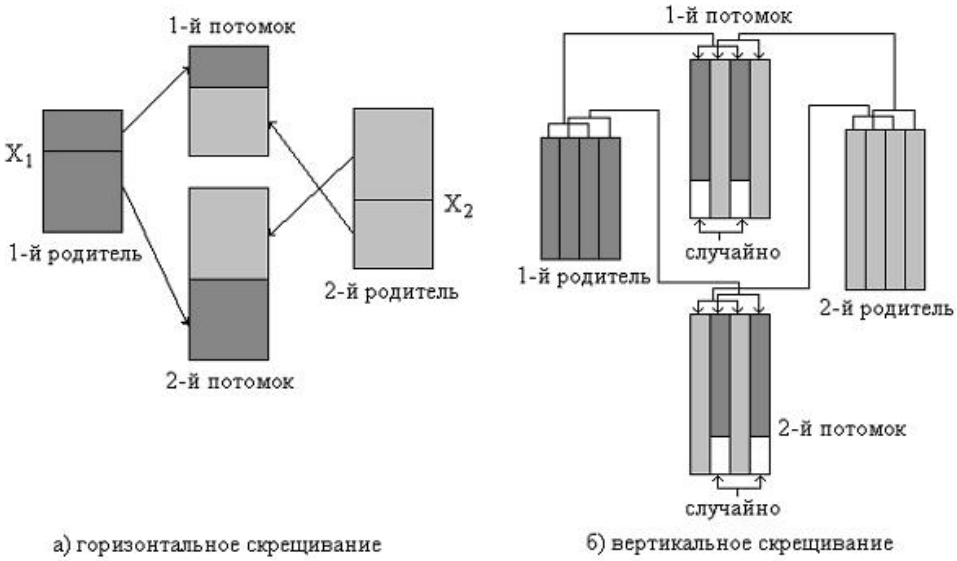

Рис.3. Операции скрещивания над двоичными последовательностями.

Механизм мутации обеспечивает появление новых генов в особях. Он применяется с заранее заданной вероятностью P_m к выходу операции скрещивания. В алгоритме применяется три стандартных для данного типа алгоритмов вида мутации, выбор между которыми происходит случайно: а) мутация одиночного бита (каждый бит в последовательностях-потомках может изменить своё значение с вероятностью 0.5%); б) добавление вектора (в случайную позицию в последовательности вставляется случайным образом сгенерированный вектор, что позволяет вводить и рассматривать тестовые последовательности с большей длиной); в) удаление вектора (из случайно выбранной позиции удаляется вектор, если он был несущественным, то оценочная функция всей последовательности не должна уменьшится).

Общая структура генетических алгоритмов, применявшихся авторами ранее также практически идентична за исключением некоторых моментов. Псевдокод основного цикла унифицированного генетического алгоритма приведён на рис.4. Различными являются эвристики построения начальной популяции и вычисление фитнесс-функции. В общем случае формирование начальной популяции сводится к псевдослучайной генерации тестовых последовательностей. Отметим также, что задачи построения тестов приведённый алгоритм ищет тестовую последовательность для одной неисправности и, поэтому, он будёт обрамлён ещё одним циклом, в котором производится перебор неисправностей из списка.

4. Построение фитнесс-функций.

Ключевым моментом быстродействия генетических алгоритмов является время вычисления фитнесс-функций, поскольку оно производится для каждой особи популяции и физически скрыто в самом внутреннем цикле алгоритма. Время вычисления фитнесс-функций зависит от двух факторов: её вида и способа вычисления. В генетических алгоритмах построения входных последовательностей вычисление фитнесс-функции основано на моделировании работы схемы: исправной, с неисправностями или обеих сразу. Сам процесс моделирования является очень трудоёмким и его скорость в свою очередь сильно зависит от применяемых алгоритмов. Поэтому вторым ключевым фактором скорости вычисления фитнесс-функций являются применяемые алгоритмы моделирования работы цифровых схем. В данной работе мы остановимся только на первом факторе: виде фитнесс-

```
Генетический_алгоритм(Схема(ы),РазмерПопуляции,Число итераций)
{
  // подготовка начальной популяции
  ПостроитьНачальнуюПопуляцию();
  ОценитьОсобей(НачальнаяПопуляция);
  НомерПопуляции=0;
  Пока( не_достигнут_критерий_остановки )
  {
    НоваяПозиция=0;
    Для( i=0 ; i<РазмерПопуляции ; i++)
    {
      ОперацияВыбора(РодительА, РодительБ);
      Если( rand() < Pc )
        ОперацияСкрещивания(РодительА, РодительБ, Потомок);
      Если( rand() < Pm )
        ОперацияМутации(Потомок);
      ДобавитьВПромежуточнуюПопуляцию(Потомок, НоваяПозиция);
      НоваяПозиция++;
    }
    ОценитьОсобей(ПромежуточнаяПопуляция);
    ПостроитьНовуюПопуляцию(РазмерПопуляции);
    НомерПопуляции++;
  }
  СоздатьОтчёт();
}
```

Рис.4. Псевдокод основного цикл генетического алгоритма.

функции и способе её вычисления. В статье не рассматриваются алгоритмы моделирования работы цифровых схем, поскольку они являются отдельной большой отраслью научных исследования. Отметим, однако, что авторы ранее также проводили исследования в данном направлении [13].

Рассмотрим построение фитнесс-функций, основанных на моделировании для каждой постановки задачи.

1) Построение тестовых последовательностей.

Фитнесс-функция в данном случае имеет вид:

$$H(s, f) = \sum_{i=1}^{i=длина} H^i * h(v_i, f) \qquad (2)$$

где s-анализируемая последовательность; v_i - вектор из рассматриваемой последовательности, i – позиция вектора в последовательности, $f \in \tilde{F}_{const}$ - заданная неисправность, H – предварительно заданная константа в диапазоне $0 < H <= 1$, благодаря которой предпочтение отдаётся более коротким последовательностям.

Функция $h(v, f)$ в свою очередь определяет качество отдельного входного вектора v при моделировании в присутствии неисправности f:

$$h(v, f) = f_1(v, f) + c_1 * f_2(v, f) = n_1 + c_1 \cdot n_2 \qquad (3)$$

где v – текущий входной набор, c_1 - константа нормирования, равная отношению числа вентилей схемы к числу триггеров; $f_1(v, f)$ и $f_2(v, f)$ эвристические функции, определяющие:

$$n_1 = f_1(v, f) = \sum_{g \in G} вес(g) \cdot различие(g) \quad (4)$$

- взвешенное число линий с различными значениями сигналов в исправной и неисправной схемах, G - множество всех вентилей схемы;

$$n_2 = f_2(v, f) = \sum_{g \in T} вес(g) \cdot различие(g) \quad (5)$$

- взвешенное число триггеров с различными значениями сигналов в исправной и неисправной схемах, T - множество всех триггеров схемы.

Функция определяется следующим образом:

$$различие(g) = \begin{cases} 0, если\ выход\ вентиля\ g\ одинаков\ в\ исправной\ и\ неисправной\ схемах; \\ 1, в\ противном\ случае. \end{cases} \quad (6)$$

В качестве веса выбирается мера наблюдаемости элемента схемы, вычисляемая на этапе предварительной обработки схемы.

2) Построение инициализирующих последовательностей

Поскольку алгоритм строит инициализирующие последовательности, оценочную функцию необходимо задавать с помощью следующих параметров:

- отношение числа инициированных триггеров к их общему числу n_1; чем больше триггеров схемы перешло из неопределённого состояния в определённое, тем выше качество заданной последовательности;
- активность схемы или число событий моделирования n_2; чем выше число событий при моделировании схемы на заданной последовательности, тем выше вероятность, что ещё неустановившиеся триггеры перейдут в определённое состояние.
- длина последовательности n_3; из двух последовательностей при прочих равных условиях необходимо предпочесть более короткую;

Таким образом, оценочная функция имеет следующий вид:

$$f(s) = f(n_1, n_2, n_3) = (c_1 * n_1 + c_2 * n_2) * c_3^{n3} \quad (7)$$

где n_1, n_2, n_3 – описанные выше параметры, c_1, c_2, c_3 – нормализующие константы, которые имеют следующий смысл:

- c_1 и c_2 уравновешивают влияние на фитнесс-функцию числа триггеров, перешедших в известное состояние, и активность вентилей в схеме; очевидно, что для построения эффективной фитнесс-функции недостаточно одного из параметров n_1 или n_2, поскольку это может завести алгоритм в тупик, например в случае, когда число установленных триггеров не изменяется, а активность схемы близка к нулю;
- c_3 позволяет искать более короткие входные последовательности; если данную константу выбрать близкой к единице c_3<1, то из двух входных последовательностей при прочих равных условиях (число активизированных триггеров и активность схемы) большую оценочную функцию будет иметь та последовательность, длина которой меньше; заметим здесь также, что выбор константы c_3 даже очень близкой к единице существенно сказывается на временных затратах при вычислении оценочной функции, когда необходимо достичь её некоторого фиксированного значения; в нашем случае введение данного компонента фитнесс-функции оправдано, поскольку её вычисление относительно нетрудоёмкая задача, например в сравнении с вычислением фитнесс-функции при построении тестов: в нашем алгоритме используется моделирование работы исправной схемы, тогда как в упомянутом алгоритме используется моделирование с неисправностями.

3) В данной постановке для фитнесс-функции главным параметром является число выходов, на которых значения в двух семах различны. Далее по важности идут количество

активированных триггеров и активность вентилей схемы на данной последовательности. Таким образом, фитнесс-функция будет иметь следующий:

$$f(s) = f(n_1, n_2, n_3) = n_1 + c_1 \cdot n_2 + c_2 \cdot n_3 \quad (8)$$

где n_1, n_2, n_3 имеют следующее значение:

$$n_1 = \sum_{g \in Y} различие(g) \quad (9)$$

- число различных значений на внешних выходах двух анализируемых схем. Данный параметр является определяющим, поскольку наличие даже одного расхождения на внешних входах двух схем, говорит о том, что различающая последовательность построена. Поэтому данный параметр в фитнесс-функцию должен входить с таким коэффициентом, чтобы при наличии хотя бы одного различия, значение фитнесс-функции нивелировало другие параметры и говорило о решении задачи. Y – множество всех внешних выходов схемы.

$$n_2 = \sum_{g \in T} различие(g) \quad (10)$$

- число различных псевдовыходов схемы. В случае, когда различные значения в двух проверяемых схемах, ещё не достигли выходов, решающим является распространения таких отличий на псевдовыходы (триггеры) схемы. Наличие триггеров с различными значениями на определённом такте времени позволяет с большей вероятностью получить различия на внешних выходах схемы на следующем такте работы. T – множество псевдовыходов двух схем.

$$n_3 = \sum_{g \in G} различие(g) \quad (11)$$

- число вентилей двух схем с различными значениями сигналов. В случае, когда различие в поведении двух схем не достигло ни внешнего выхода, ни псевдовыхода схемы, необходимо строить последовательности, для которых является важным различие сигналов внутри комбинационных блоков. Этот параметр важен либо когда ещё ни одна из пар триггеров в исправной и неисправной схемах не получили различные значения, либо когда такие пары есть, но этого не достаточно для распространения различных значения до внешних выходов схемы. Данный параметр в фитнесс функции должен иметь наименьшее значение. G – множество всех вентилей схемы.

5. Программная реализация объектно-ориентированной структуры классов.

В данном разделе мы опишем объектно-ориентированную структуру данных, которая использовалась при программной реализации предложенного алгоритма. Поскольку программная реализация проводилась в среде программирования C++ Builder, то структура классов и их наполнение направлены на максимальное использование библиотек данной среды.

Структура данных содержит определение нескольких классов и их методов.

Основным в программной реализации является класс TMyGA, непосредственно реализующий функцию генетического алгоритма. Данный класс является потоковым (производным от TThread), что позволяет избежать «замерзания» основной формы приложения при его выполнении. Он содержит следующие основные члены класса:

- IndividualsNumber – число особей в популяции, в данной реализации данный параметр не изменяется от поколения к поколению;
- PopulationNumber – номер текущего поколения;
- MaxPopulations – максимальное число поколений в генетическом алгоритме;
- CrossProbability – вероятность операции скрещивания, в данной реализации не изменяется от поколения к поколению;
- MutationProbability – вероятность операции мутации, также является постоянной

величиной;
- указатель нам массив популяции Population; структура отдельной популяции будет описана ниже, размер массива равен IndividualsNumber;
- указатель на массив фитнесс-функций Fitness, размер массива равен IndividualsNumber.

Класс TMyGA содержит также следующие основные методы:
- PrepareData() – выполняет все основные подготовительные действия: ввод описания схемы в основную структуру данных, выделение массивов памяти;
- GenerateStartPopulation() – строит начальную популяцию;
- ValuePopulation() – вычисляет оценочную функцию для всех особи в популяции;
- StopCriterionReached() – проверяет критерий остановки работы генетического алгоритма: достигну то ли максимальное число поколений либо инициализированы все триггеры;
- DoSelect() – выполняет выбор двух особей-родителей, к которым впоследствии будут применены генетические операции;
- DoCrossingover() – выполняет проверку необходимости проведения генетической операции скрещивания и выбор типа данной операции (см. ниже);
- DoVerticalCrossingover() – выполняет операцию вертикального скрещивания, результатом является одна особь-потомок;
- DoHorizontalCrossingover() - выполняет операцию горизонтального скрещивания, результатом является одна особь-потомок;
- DoMutation() – выполняет проверку необходимости проведения генетической операции мутации и выбирает тип данной операции;
- CutLineMutation() – выполняет операцию мутации путём удаления из входной последовательности случайного набора;
- AddLineMutation() – выполняет операцию мутации путём вставки одного набора во входную последовательность в случайную позицию;
- ChangeBitMutation() – выполняет операцию мутации – изменение случайных бит во входной последовательности;
- ConstructNewPopulation() – строит популяцию нового поколения из предыдущей промежуточной популяций;
- SortPopulationByFitness() – выполняет сортировку популяции по фитнесс-функции;

Отметим, что данный набор процедур является необходимым и является расширением списка процедур, использованных в псевдокоде алгоритма (см. выше).

Кроме данных процедур в программной реализации присутствует блок, отвечающий за моделирование работы цифровой схемы (вычисление фитнесс-функции особи). Он содержит следующие подпрограммы:
- FitnessSimulate() – вычисляет оценочную функцию особи путём моделирования; входными параметрами являются указатель на особь и длина особи, выходной параметр – оценочная функция;
- InputCircuit() – ввод описания схемы из файла в основную структуру данных;
- CutLoop() – преобразование последовательностной схемы в псевдо-комбинационный эквивалент путём обрыва линий обратных связей и введения псевдовходов и псевдовыходов;
- ChangeTestSet() – ввод на внешние входы схемы очередного входного набора;
- Kern4v() – моделирование работы схемы на заданном входном наборе в 3-значном алфавите;

Кроме описанных членов класса и его методов класс TMyGA содержит полное описание структуры данных представления внутреннего описания схемы [11,13].

Апробация работы алгоритма проводилась на схемах из международного каталога ISCAS-89 [14]. Из-за ограниченности объёма статьи мы не приводим таблицу с экспериментальными данными. Для алгоритма генерации тестов они приведены в [8], для

задачи построения инициализирующих последовательностей – в [10]. Для третьей задачи построения последовательностей верифицирующих эквивалентность двух схем эксперименты авторами проводятся в настоящее время.

6. Заключение.

В статье предложено обобщённый подход построения входных тестовых последовательностей с заданными свойствами. Данный подход основа на генетических алгоритмах. Рассмотрены примеры формирования фитнесс-функций в зависимости от требуемых свойств входных последовательностей. Приведены экспериментальные данные, показывающие эффективность выбранного подхода. В качестве дальнейших исследований следует рассматривать распространение предложенного подхода на иные задачи построения входных последовательностей.

Литература

1. Niermann T., Patel J.H. HITEC: A Test Generation Package for Sequential Circuits // Proc. European Design Automation Conf.- 1991.- p.214-218.
2. Sellers F.F., Hsiao M.Y., Bearnson L.W. Analysing errors with the boolean difference // IEEE Transactions on Computers.- 1967.- №5.- p.675-680.
3. Lisanke R., Brgles F., Degens A.J., Gregory D. Testability-driven random test pattern generation // IEEE Trans. Computer-Aided Design.- 1987.- №6.- p.1082-1087.
4. Bryant R.E. Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Comput. Surv. 24, 3, 293-318
5. Rho J.-K., Somenzi F. and Pixley C. Minimul length synchronizing sequences of finite state mashine. In Proceedings of the ACM Design Automation Conference, 463-468.
6. A. Ghosh, S. Devadas and A.R. Newton, Sequential Logic Testing and Verification, Kluwer Academic Publishers, 1992.
7. O.Coudert, J.C.Madre: “A Unified Framework for Formal Verification of sequential Circuits,” ICCAD-90: IEEE Intl. Conf. on Computer Aided Design, Nov. 1990, pp. 134-137
8. Y.A.Skobtsov, D.E.Ivanov. Evolutionary approach to the test pattern generation for the sequential circuits // Радиоэлектроника и информатика.- 2003, №3.- с.46-51.
9. Skobtsov Y.A., El-Khatib, Ivanov D.E. Distributed Genetic Algorithm of Test Generation For Digital Circuits // Proceedings of the 10th Biennial Baltic Electronics Conference.-Tallinn Technical University,2006.-p.281-284. (0.4 д.а.)
10. Д.Е. Иванов, Ю.А. Скобцов, А.И. Эль-Хатиб Построение инициализирующих последовательностей синхронних цифрових схем с помощью генетических алгоритмов.- Проблеми інформаційних технологій.-2007.-№1.-с.158-164.
11. Барашко А.С., Скобцов Ю.А., Сперанский Д.В. Моделирование и тестирование дискретных устройств. – Киев:Наукова думка, 1992. – 288 с.
12. Goldberg D.E., Genetic Algorithm in Search, Optimization, and Machine Learning.- Addison-Wesley.- 1989.
13. Иванов Д.Е., Скобцов Ю.А. Параллельное моделирование неисправностей для последовательностных схем // Искусственный интеллект. – 1999. - №1. – С.44-50.
14. Brgles F., Bryan D., Kozminski K. Combinational profiles of sequential benchmark circuits // International symposium of circuits and systems, ISCAS-89.– 1989.– p.1929-1934.