В статті розглянуто застосування еволюційного підходу до генерації тестів для схеми вбудованого функціонального самотестування. Головна мета пропонованого підходу – зменшення розміру тестового словника за рахунок тестування внутрішньої логіки структурного рівня мікропрограмою пристрою. При цьому генерація даних та спостереження виконуються на функціональному рівні за допомогою генетичного алгоритму. Експериментальні дані показують високу ефективність підходу, що запропоновано

Ключові слова: цифровий пристрій, система на чіпі, вбудоване самотестування, функціональний тест, генетичний алгоритм

В статье рассматривается эволюционный подход к генерации тестов для схемы встроенного функционального самотестирования. Главная цель предлагаемого подхода – уменьшение размера тестового словаря за счёт тестирования внутренней логики структурного уровня микропрограммой устройства. При этом генерация тестовых данных и наблюдение происходят на функциональном уровне с помощью генетического алгоритма. Экспериментальные данные показывают высокую эффективность предложенного подхода

Ключевые слова: цифровое устройств, система на чипе, встроенное самотестирование, функциональный тест, генетический алгоритм

УДК 004.3:681.518

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ПОСТРОЕНИЯ ФУНКЦИОНАЛЬНЫХ ТЕСТОВ АРИФМЕТИКО-ЛОГИЧЕСКИХ УСТРОЙСТВ

Ю. А. Скобцов
Доктор технических наук, профессор,
заведующий кафедрой
Кафедра автоматизированных систем управления
Донецкий национальный технический университет
ул. Артема, 58, г. Донецк, Украина, 83001
E-mail: skobtsov@kita.dgtu.donetsk.ua

Д. Е. Иванов
Доктор технических наук, доцент
Старший научный сотрудник
Отдел теории управляющих систем*
E-mail: ivanov@iamm.ac.donetsk.ua

В. Ю. Скобцов
Кандидат технических наук, доцент,
заведующий лабораторией
Лаборатория дискретной математики
и прикладной алгебры*
E-mail: skobtsov@iamm.ac.donetsk.ua

*Институт прикладной математики и механики
НАН Украины
ул. Розы Люксембург, 74, г. Донецк, Украина, 83114

1. Введение

Рост сложности проектируемых цифровых устройств (ЦУ) не прекращается и в настоящее время. При этом по оценкам ITRS [1] длина строящихся тестовых наборов возрастает настолько быстро, что существенно влияет на время их приложения, а, следовательно, и на стоимость тестирования.

Одним из направлений в попытке преодолеть указанный недостаток является применение схем встроенного самотестирования (BIST). Подход был предложен для эффективного решения данной задачи именно в современных высокоинтегрированных решениях, в частности в системах-на-чипе (SOC) [2]. Функциональный или арифметический подход к построению BIST позволяет сбалансировано разделять процедуру тестирования на внешнюю и встроенную.

С другой стороны, для решения основных задача диагностики ЦУ широко применяется эволюционный подход [3]. На его основе разработаны эффективные методы построения и оптимизации тестов ЦУ.

Цель разрабатываемого метода – максимальное сжатие объёма тестового словаря в схеме BIST. Это достигается за счёт объединения двух указанных направлений:

– совмещения функционального и структурного уровней тестирования в схеме BIST путём вовлечения в него микропрограммной логики подачи и контроля тестовых наборов;

– применение эволюционного подхода к генерации тестовых наборов встроенного словаря тестирования, учитывающего двухуровневую схему организации тестирования.

2. Анализ литературных данных и постановка задачи исследования

Задача построения тестов минимальной длины формально сводится к задаче минимального покрытия заданного множества неисправностей. Первоначально задача сжатия тестовых наборов решалась путём

статической оптимизации уже построенных тестовых последовательностей, поскольку применение оптимизации непосредственно во время построения тестов – динамическая оптимизация – существенно усложняет задачу. В этом направлении можно отметить работу [4], в которой предложены алгоритмы сжатия тестов путём удаления несущественных векторов и уменьшения множества целевых неисправностей без потери качества тестирования.

Однако с ростом сложности ЦУ эффективность такого похода снижалась. Поэтому для систем-на-чипе был предложен иной подход – встроенное самотестирование [2, 5]. Основная идея подхода – использовать ресурсы системы по хранению и подаче тестовых наборов на тестируемое вычислительное ядро. Работы, применявшие данные подход, делали упор на обработку предварительно построенных тестовых наборов. Хранимые тестовые наборы обычно являются частично определёнными, т.е. содержащими неопределенные биты. При извлечении наборов из словаря происходит их декомпрессия, т.е. доопределение. В частности, в [6 – 7] изучается вопрос эффективности декомпрессии хранимых сжатых частично определённых тестовых наборов и применения для этой задачи подхода сканирующих регистров.

Одним из бурно развивающихся направлений в технической диагностике ЦУ являются эволюционные вычисления. Наиболее популярными у исследователей являются генетические алгоритмы (ГА) [8]. На основании ГА разработаны как методы построения тестов, так и их оптимизации. Обобщение подходов построения различных классов идентифицирующих последовательностей ЦУ дано в монографии [3]. В данных методах происходит эволюция популяции потенциальных решений, которые являются входными двоичными последовательностями. Их качество проверяется путём явного моделирования поведения ЦУ на заданной входной последовательности. Это позволяет обрабатывать ЦУ большой размерности.

Также ГА применялись к решению задач минимизации тестовых последовательностей [9].

В данной работе предлагается метод автоматизированной генерации тестов для SOC, который объединяет два вышерассмотренных подхода. Система рассматривается как содержащая набор регистров и комбинационное функциональное устройство. Встроенная в функциональную часть микропрограмма при приложении тестового набора производит тестирование внутренней логики структурного уровня, которая реализует заданную/заданные арифметические операции. При этом подсистема подачи входных наборов и наблюдения результатов не имеет доступа к структурному уровню, а рассматривает устройство только с функциональной точки зрения.

В статье разрабатывается генетический алгоритм построения компактных тестовых наборов функционального уровня для такой схемы самотестирования. Поскольку в данной схеме BIST фактически происходит объединение функционального и структурного уровней, то метод использует как двоичное, так и арифметическое кодирование, а также разработанные для них эволюционные операции. Также в такой схеме тестовые наборы принадлежат верхнему функциональному уровню, тогда как применяемая для их оценки фитнесс-функция отображает тестовые свойства структурного уровня.

3. Функциональная схем встроенного самотестирования

Суть функционального самотестирования заключается в следующем. Рассмотрим схему, изображённую на рис. 1. Центральным элементом, непосредственно выполняющим все реализуемые операции с данными, является арифметико-логическое устройство (АЛУ). Именно оно является объектом тестирования структурного уровня. АЛУ содержит входные и выходные линии данных, которые посредством шин соединены с блоком регистров (БР). Процесс контролируется сигналами от устройства управления, которые не изображены на рисунке, поскольку носят вспомогательный характер.

Рассмотрим теперь прохождение данных в такой схеме. Если рассматривать АЛУ A_0 как обычное функциональное устройство, то для K входных операндов X и Y мы получаем K результатов: $Z = X / Y$ (для примера и без потери общности будем считать, что в текущий момент реализуется операция деления в АЛУ). Тогда в традиционной схеме данные K результатов упаковываются в сигнатурный анализатор и будут использованы при встроенном самотестировании для верификации корректности работы операции деления в АЛУ.

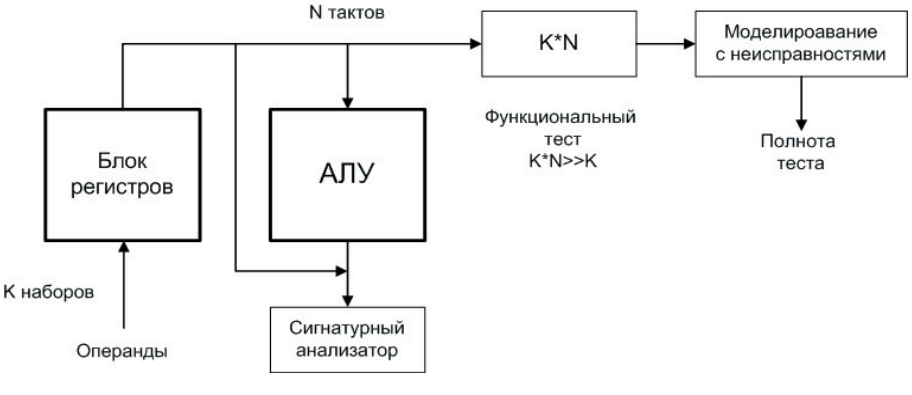

Рис. 1. Структурная схема функционального самотестирования

Если рассматривать АЛУ A_0 на структурном уровне, то каждая операция выполняется за некоторое число машинных тактов N . В функциональном встроенном самотестировании предполагается, что для наблюдения доступны результаты работы на каждом из тактов работы A_0 при реализации выбранной операции деления [2, 10]. Таким образом, в примере с K входными операндами наблюдается K*N выходных реакций, производимых АЛУ за K*N тактов работы. Следовательно, при перемещении с функционального уровня (уровня операндов) на уровень микроинструк-

ций (логический уровень представления ЦУ) размерность выходных реакций, а, следовательно, сигнатур для анализа, увеличивается в N раз. Т. е. именно в N раз можно уменьшить число входных данных на функциональном уровне, оставив размер словаря сигнатур структурного уровня в прежнем размере.

Обозначим через L разрядность данных (операндов функционального уровня) и l - разрядность входа АЛУ. Уменьшение размерности тестовых данных при такой «компрессии» можно выразить:

$$R = \frac{N*l}{2*L}. \qquad (1)$$

Например, для случая с шириной машинного слова 32 бита, для делителя со 105 входами и 120 тактами работы операции уменьшение объёма тестовых данных в соответствии с (1) составит $120*105/64 = 197$.

В рассматриваемой схеме BIST входные наборы каждого такта работы АЛУ A_0 выполняют роль псевдослучайных тестовых наборов в классической схеме самотестирования и в дальнейшем используются совместно с процедурами моделирования неисправностей для оценки качества тестирования для традиционных типов неисправностей, например одиночных константных.

Таким образом, возникает задача автоматизированной генерации входных наборов для рассматриваемой схемы функционального самотестирования, которые достигают максимального покрытия за очень короткое время. Покажем, что данная задача может быть эффективно решена с помощью ГА.

Любое АЛУ, применяемое на практике, реализует некоторый базовый набор арифметических операций: $Op = \{op_1,....,op_m\}$. Тогда задача заключается в итеративной реализации тестирования каждой из операций $op_i \in Op$. Каждая такая итерация распадается на два этапа. Первый этап заключается в выборе текущей тестируемой операции $op_i \in Op$. На втором этапе ГА должен породить такой набор операндов, при подаче которых на вход при рассматриваемой схеме тестирования будет получено максимальное покрытие множества рассматриваемых неисправностей структурного уровня. После чего этапы повторяются до завершения тестирования всех операций, реализуемых АЛУ. Итоговый тест строится как набор тестов для всех операций $op_i \in Op, i = \overline{1,m}$. Мы не будем приводить алгоритмическую реализацию данного метода, поскольку она является тривиальной.

4. Генетический алгоритм построения функциональных тестов

Рассмотрим теперь второй этап, заключающийся непосредственно в тестировании операции $op_i \in Op$. Решение данной задачи будем строить с помощью ГА.

ГА [8] представляют собой алгоритмы поиска, которые основаны на принципах естественного природного отбора. Каждое потенциальное решение задачи называется особью и кодируется с помощью некоторых генов. В простейших реализациях ГА представляется обычной двоичной строкой, что показывает его применимость к решению задачи построения тестов ЦУ. Набор особей называется популяцией и является подмножеством точек в пространстве поиска. В целом генетический поиск заключается в моделировании по поколениям эволюции такой искусственной популяции. Для задач генерации тестов ЦУ он подробно описан в [3].

С каждой особью сопоставляется оценочная функция, которая показывает, насколько успешной является особь в смысле решения задачи и, следовательно, задаёт её возможность перейти в следующее поколение. Построение новых особей в процессе эволюции определяется механизмом скрещивания. В нём выбираются две особи, называемые родителями. После применения операции скрещивания порождаются особи, называемые потомками. К ним дополнительно применяется операция мутации, целью которой является разнообразить поиск. Новое поколение популяции представляет собой лучших особей из предыдущего поколения и модифицированных особей. Таким образом, в процессе эволюции происходит накопление некоторой информации, аккумулирующей свойство приспособленности.

Для построения ГА генерации тестов необходимо задать понятия особи, популяции, а также применяемые эволюционные операции. На этапе экспериментов определяются численные значения эвристических констант: вероятности применения эволюционных операций, число особей в популяции, схема построения новой популяции и максимальное число итераций.

Особью в методе служит набор операндов (X,Y), которые для устройства A_0 представляют входной двоичный вектор ширины $2l$. Тестом для выбранной операции $op_i \in Op$ является набор операндов $S = \{(X_1,Y_1),...,(X_k,Y_k)\}$.

Прежде всего, следует выбрать модель неисправности функционального уровня, что, в свою очередь, определит построение оценочной функции разрабатываемого ГА-метода. На функциональном уровне устройство A_0 рассматривается как чёрный ящик, реализующий некоторую операцию $op_i \in Op, i = \overline{1,m}$. При этом разработчику доступны для наблюдения только его входные и выходные линии. Поэтому традиционные константные неисправности не могут рассматриваться в данном контексте. На функциональном уровне наиболее близкими по физическому смыслу являются неисправности инвертирования входных/выходных линий АЛУ. Т. е. всякая неисправность инверсии линии входа, или что то же самое инверсия в битовом представлении операндов X или Y, должна наблюдаться на выходных линиях устройства A_0, также выраженная в виде инверсии бита результата операции $op_i(X,Y)$. Причём, чем больше различий (инверсий линий выходов) в сравнении с истинным результатом операции, тем выше качество набора в смысле теста.

Определим матрицу P размерности $2l*l$. В ней элемент p_{ij} равен единице, если инверсия входа A_0 с номером i породила инверсию выхода с номером j. В начальный момент тестирования все элементы матрицы P равны 0.

Тогда оценочная функция набора операндов

$$S = \{(X_1,Y_1),...,(X_k,Y_k)\},$$

выражается в виде:

$$O(op_i,S)=\frac{\sum_{i=1}^{2l}\sum_{j=1}^{l}p_{ij}}{2l*l}. \quad (2)$$

При этом для отдельного набора (X,Y) на некоторой итерации ГА выражение (2) следует вычислять с учётом того, что матрица P заполнена по результатам моделирования предыдущих наборов.

В данном подходе объект тестирования рассматривается как чёрный ящик, входными наборами которого являются операнды. С одной стороны, они являются двоичными наборами, с другой – числами. Поэтому будем применять два типа эволюционных операций [11].

Операции скрещивания включают арифметическое скрещивание (применяемое с вероятностью $p_{скр}^{ар}$) и двоичное скрещивание (вероятность применения $p_{скр}^{бит}$), $p_{скр}^{ар}+p_{скр}^{бит}=p_{скр}<1$.

Арифметическое скрещивание выполняется для двух особей $A=(X_a,Y_a)$ и $B=(X_b,Y_b)$. Особь-потомок $\tilde{A}=(\tilde{X},\tilde{Y})$ определяются как:

$$\tilde{X}=(1-\alpha)\cdot X_a+\alpha\cdot X_b,$$

$$\tilde{Y}=(1-\alpha)\cdot Y_a+\alpha\cdot Y_b, \quad (3)$$

где $\alpha\in[0;1]$.

Операции мутации также включают арифметическую мутацию (вероятность $p_{мут}^{ар}$) и двоичную (вероятность применения $p_{мут}^{бит}$), $p_{мут}^{ар}+p_{мут}^{бит}=p_{мут}<<1$.

Арифметическая мутация выполняется над одной особью-родителем $A=(X_a,Y_a)$ и порождает одну особь-потомка $\tilde{A}=(\tilde{X},\tilde{Y})$:

$$\tilde{X}=X_a\pm\Delta\cdot X_a,$$

$$\tilde{Y}=Y_a\pm\Delta\cdot Y_a, \quad (4)$$

где Δ – некоторое небольшое число.

В том случае, если АЛУ реализует целочисленные операции, то после выполнения (3) и (4) необходимо выполнить округление результата.

Двоичное скрещивание и двоичная мутация выполняются над двоичным представлением особей и соответствуют операциям простого ГА [12].

Схема построения новой популяции выбирается на этапе машинных экспериментов.

5. Исследование эффективности разработанного алгоритма

Предложенный метод генерации тестов реализован программно на языке программирования C++ в среде CodeGear.

Для оценки эффективности предложенного подхода по проверке константных неисправностей структурного уровня проводились эксперименты с АЛУ, реализующим функцию деления. Комбинационный блок устройства содержал: регистровый файл ёмкостью три операнда (делимое, делитель, частное), 5-битовый счётчик и АЛУ со следующими структурными характеристиками: 105 входов, 71 выход, 513 логических вентилей, 2383 константные неисправности.

Экспериментально выбраны следующие значения эвристических констант: $p_{скр}=0.8$; $p_{мут}=0.01$; $\alpha=0.5$; $\Delta=0.5$; число особей в популяции = 100. При построении новой популяции применяется схема элитизма с коэффициентом 0.8. Пример роста оценочной функции (2) в зависимости от числа поколений приведён на рис. 2. Видно, что данное значение стабилизируется достаточно быстро. Это позволило выбрать предельное число поколений =40.

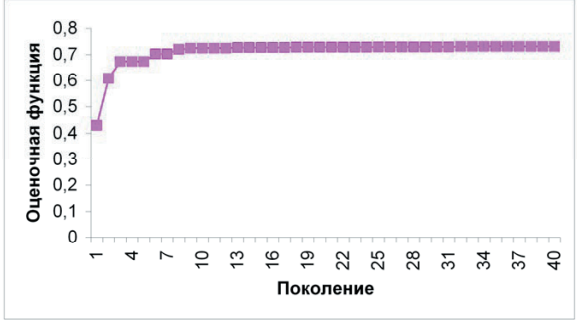

Рис. 2. Рост оценочной функции в зависимости от числа поколений

В табл. 1 показаны результаты экспериментов построения тестов на основе предложенного подхода. Приведённый входной набор из 7 тестов функционального уровня порождал 759 входных наборов структурного уровня. При этом полнота приведённого теста для инверсных неисправностей в виде (2) была около 70 %. Для константных неисправностей структурного уровня полнота составила 89.8 %.

Таблица 1

Покрытие константных неисправностей функциональным тестом

№ набора	операнд 1	операнд 2	результат	N_k	N	полнота теста, %
1	0.7345	0.7659	0.9590	108	108	66.8
2	0.6943	0.7234	0.9598	105	213	76.7
3	0.4320	0.8569	0.5041	113	326	83.3
4	0.1964	0.2098	0.9361	108	434	85.5
5	0.4679	0.4987	0.9382	110	544	88.5
6	0.4567	0.4678	0.9763	104	648	88.9
7	0.9234	0.9546	0.9673	111	759	89.8
...	...	...	...	...	...	89.8

Таким образом, для получения покрытия 89.8 % константных неисправностей в АЛУ, реализующем операцию деления, в сигнатурном анализаторе требуется хранить только 7 входных наборов. Видно, что степень сжатия теста при таком подходе 759/7>100.

Также были проведены эксперименты по изучению зависимости полноты теста в виде (2) от размерности операндов. Длина битового представления изменялась от 8 до 32 бит дискретно с шагом 8. График данной зависимости показан на рис. 3. График зависимости дли-

ны функционального теста в зависимости от ширины операндов приведён на рис. 4.

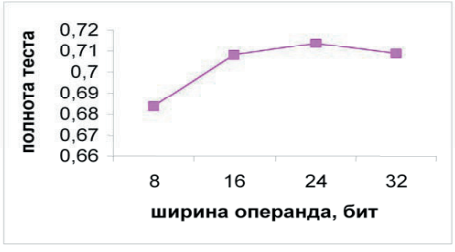

Рис. 3. Достигнутая полнота теста в зависимости от ширины операндов

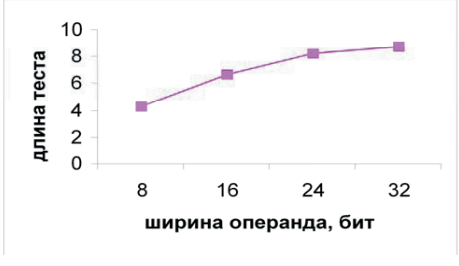

Рис. 4. Длина построенного теста в зависимости от ширины операнда

Данные двух графиков говорят о том, что при изменении ширины операндов АЛУ в рассматриваемом диапазоне эффективность построенных тестов в терминах полноты и длины последовательностей будет близка к приведённой выше.

6. Выводы

В работе предложен генетический подход построения функциональных тестов для двухуровневой схемы встроенного самотестирования. Для решения задачи минимизации тестового словаря разработан генетический алгоритм. Поскольку в задаче речь идет о двух уровня представления цифрового устройства, то и входные тесты рассматриваются двойственным образом, а в процессе их эволюции используются два типа генетических операций: двоичные и арифметические.

Экспериментальные данные показали, что для построенных тестов с функциональной полнотой около 70 % полнота тестов структурного уровня для константных неисправностей составила около 90 %. При этом степень сжатия тестов составила около двух порядков (>100). Эксперименты проводились блоком, реализующим делитель, однако их можно распространить на АЛУ, реализующие набор арифметических операций.

В дальнейшем довести полноту тестов до требуемого уровня можно традиционными техниками, например, построением сканирующего регистра, либо вводом контрольных точек [13].

Литература

1. ITRS 2010 technology roadmap [Electronic resource]. – available at: –http://www.itrs.net/Links/2010ITRS/Home2010.htm/. – Загл. с экрана. – (1.06.2013).
2. Rajski, J. Arithmetic Built-in Self-test for Embedded Systems [Text] / J. Rajski, J. Tyszer. – Prentice Hall:Pearson Professional Education, 1997. – 256 p.
3. Иванов, Д. Е. Генетические алгоритмы построения входных идентифицирующих последовательностей цифровых устройств [Text] / Д. Е. Иванов. – Донецк, 2012. – 240 с.
4. Hamzaoglu, I. Test Set Compaction Algorithms for Combinational Circuits [Текст] / I. Hamzaoglu, H. Patel // Proceedings of the International Conference on CAD (ICCAD), 1998. – P. 283–289.
5. Kruus, H. Defect-oriented BIST quality analysis [Text] / H. Kruus, R. Ubar, J. Raik // 12th Biennial Baltic Electronics Conference (BEC), 2010. – P. 153–156.
6. Dorsch, R. Reusing scan chains for test pattern decompression [Text] / R. Dorsch, H.-J. Wunderlich // Proc. of Int. Test Conf., May 29 - Jun. 1, 2001. – P. 124–132.
7. Novak, O. Test pattern decompression using a scan chain [Text] / O. Novak, J. Nosek // Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 24-26 Oct. 2001. – P. 110–115.
8. Goldberg, D. E. Genetic Algorithm in Search, Optimization, and Machine Learning [Text] / D. E. Goldberg. – Boston, MA:Addison-Wesley Longman Publishing Co. – 1989. – 412 p.
9. Logofătu, D. Efficient Evolutionary Approach for the Test Compaction Problem [Text] / D. Logofătu // 9th International Conference on development and application systems, 2008. – P. 144–148.
10. Ubar, R. HyFBIST: Hybrid Functional Built-In Self-Test in Microprogrammed Data-Paths of Digital Systems [Text] / R. Ubar, N. Mazurova, J. Smahtina, E. Orasson, J. Raik // Int. Conference MIXDES. – 2004. – P. 497–502.
11. Skobtsov, Y. A. Evolutionary approach to the functional test generation for digital circuits [Text] / Y. A. Skobtsov, D. E. Ivanov, V. Y. Skobtsov, R. Ubar // In Proc. of 9th Biennial Baltic Electronics Conf., BEC 2004 (Tallinn, Oct. 2004).- Tallinn Univ. of Techn., 2004. – P. 229–232.
12. Whitley, D. A Genetic Algorithm Tutorial [Text] / Darrell Whitley // Statistics and Computting. – 1994. – №4. – P. 65–85.
13. Touba, N. A. Test point insertion based on path tracing [Text] / N. A. Touba, E. J. McCluskey // In Proc. of IEEE VLSI Test Symposium, 1996. – P. 2–8.