ЭВОЛЮЦИОННЫЙ ПОДХОД К ТЕСТИРОВАНИЮ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Скобцов Ю.А.,Ермоленко М.Л.
Донецкий национальный технический университет,
кафедра автоматизированных систем управления
E-mail: skobtsov@kita.dgtu.donetsk.ua

Abstract. ***Skobtsov Y.A., Ermolenko M.L. Evolutionary approach to testing of microprocessor systems.*** *Testing is a crucial issue in microprocessor systems development and production process. This paper describes a method for the generation of effective programs for self-test of a processor. The method is based on the two-level genetic algorithm and combines ideas from traditional functional approaches from digital testing.*

Актуальность и состояние проблемы. Современная технология проектирования и производства компьютерных систем позволяет реализовать в одном кристалле миллионы транзисторов. В настоящее на одном чипе достаточно сложные микропроцессорные системы. Их тестирование представляет очень серьезную проблему. Особенно сложной является задача построения тестовой последовательности. Традиционные структурные методы генерации тестов, требующие, как правило, описания структуры логической схемы на вентильном уровне [1], для подобных систем не применимы в силу очень высокой размерности задачи. Построение тестовых программ микропроцессоров (МП) выполнялось обычно на функциональном уровне [2] практически "вручную". При этом тест представляет собой программу на ассемблере в отличие от двоичных последовательностей для логических схем. Недавно в [3] предложен детерминированный метод генерации тестовых программ микропроцессоров. Этот метод строит тестовые программы, которые дают хорошую полноту (процент проверяемых неисправностей) , в основном, для комбинационных блоков МП, таких как арифметико-логическое устройство (АЛУ). Но для управляющих автоматов МП, имеющих существенно последовательностные модули, этот подход дает плохие результаты. В настоящей работе рассматривается эволюционный подход к построению тестовых последовательностей МП на функциональном уровне, который использует методы эволюционных вычислений.

Эволюционный подход. Особенности идей теории эволюции и самоорганизации заключается в том, что они находят свое подтверждение не только для биологических систем, успешно развивающихся многие миллиарды лет. В настоящее время бурно развивается новое направление в теории и практике искусственного интеллекта эволюционные вычисления — термин, обычно используемый для общего описания алгоритмов поиска, оптимизации или обучения, основанных на некоторых формализованных принципах естественного эволюционного отбора. Эволюционные вычисления используют различные модели эволюционного процесса. Среди них можно выделить следующие основные парадигмы:

1. Генетические алгоритмы;
2. Эволюционные стратегии;
3. Эволюционное программирование;
4. Генетическое программирование.

Отличаются они, в основном, способом представления искомых решений и различным набором используемых в процессе моделирования эволюции операторов.

Генетические алгоритмы (ГА), являясь одной из парадигм эволюционных вычислений, представляют собой метод поиска решения проблемы , построенный на принципах, сходных с принципами естественного отбора. Решение данной проблемы соответствует особи, которая в ГА представляется в виде специальной структуры — хромосомы. В простейшем случае это двоичная последовательность. Это делает привлекательным применение этого математического аппарата к решению задачи построения проверяющих тестов [4]. Множество особей (решений проблемы) составляет популяцию.

ГА использует три основных генетических оператора: репродукция, кроссинговер, мутация. При репродукции особи копируются согласно их значениям ЦФ. Копирование лучших особи с большими значениями ЦФ определяет большую вероятность их попадания в следующую генерацию. Оператор репродукции реализует принцип “выживания сильнейших” по Дарвину.

Оператор кроссинговера (ОК) обычно выполняется в три этапа:

1) Два стринга (хромосомы, особи) $A = a_1 a_2 \ldots a_n$ и $B = b_1 b_2 \ldots b_n$ выбираются случайно из текущей популяции после репродукции;

2) Выбирается также случайно точка кроссинговера k $(1 \le k < n)$;

3) Хромосомы A и B обмениваются частями после k-й позиции и производят два новых стринга $A' = a_1 a_2 \ldots a_k b_{k+1} \ldots b_n$ и $B' = b_1 b_2 \ldots b_k a_{k+1} \ldots a_n$.

Оператор мутации (ОМ) случайным образом (с небольшой вероятностью) изменяет ген (элемент) стринга.

Используя эти три основные оператора, популяция (множество решений данной проблемы) эволюционирует от поколения к поколению Эволюция такой искусственной популяции представлена на рис.1.

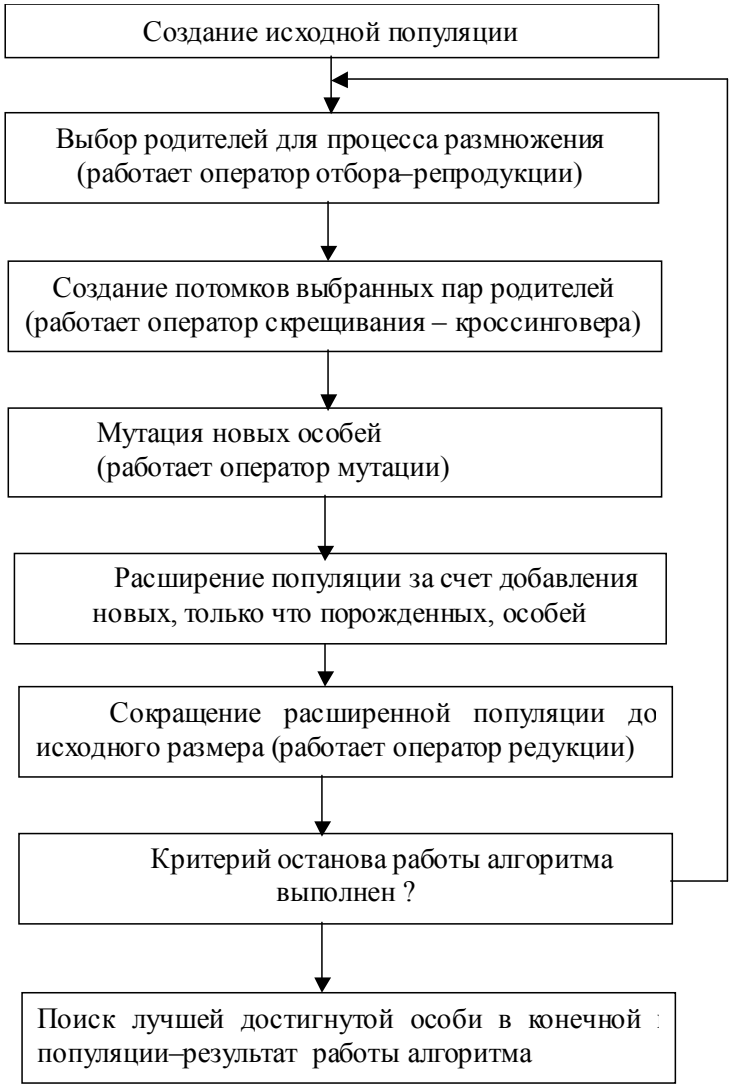

Рисунок 1

Постановка проблемы тестирования микропроцессорных систем. Таким образом, чтобы применить ГА к решению данной проблемы, необходимо определить понятия особи, популяции, операции скрещивания и мутации, задать оценочную функцию. Очевидно также, что эффективность применения ГП зависит от целого ряда параметров: размера популяции, метода выбора особей из предыдущей популяции, скрещивания и мутации, а также вида оценочной функции.

При построении теста МП систем необходимо иметь описание ее функционирования (включая систему команд, подробное описание каждой инструкции, способы адресации и т.п.). В настоящее время для этого используются специальные языки описания аппаратуры, например, VHDL, позволяющие получить достаточно точные модели МП систем. Рассматриваемый подход также предполагает использование описания МП системы на языке VHDL.

Поскольку применяется модель МП функционального уровня, то используются функциональные неисправности этого же уровня. Мы будем использовать модель одиночных константных неисправностей отдельных битов на уровне языков регистровых передач [5] (RT-level assignment single-bit stuck-at fault). Эта модель определяется как установка постоянного значения 0 или 1 в одном бите при выполнении оператора присваивания языка регистровых передач. При наличии этой неисправности оператор загружает правильное значение переменной за исключением одного бита, где значение соответствующего разряда насильственно устанавливается в 0 или 1.

Например, рассмотрим выполнение оператора addr <- (taill + reg1) mod 2**8 при значениях переменных taile='00000100' и reg1='0001001'. При неисправности константа 1 в третьем слева бите в результат сложения '00001101' искусственно будет в этот разряд установлена 1. Тогда после пересылки получим следующее значение переменной addr ='00101101'. Далее сигнал, соответствующий этой переменной, будет обрабатываться и распространяться как обычно, но с неправильным значением одного бита.

Данные моделирования показывают, что эта модель, несмотря на свою простоту, позволяет достаточно точно оценить полноту тестовой последовательности. Полнота теста, полученная для такой модели, хорошо коррелирует с полнотой, которая оценивалась по данным логического моделирования с одиночными константыми неисправностями на вентильном уровне.

Моделирование неисправностей при таком подходе выполняется с помощью программ, которые в процессе моделирования изменяют значения соответствующих разрядов переменных. Такой подход позволяет выполнять моделирование неисправностей с оценкой полноты тестовой последовательности на уровне ЯРП с минимальными дополнительными затратами, так как VHDL модель неисправных схем работает практически также быстро как и для исправных устройств. Незначительное за-

медление имеет место при активации неисправностей, для которых устанавливаются некоторые точки останова, где вносятся значения неисправных битов.

Генетический алгоритм построения тестовой последовательности. Для построения тестовой последовательности МП систем на функциональном уровне используется двухуровневый ГА. Суть его заключается в следующем.

Сначала выполняется анализ системы команд данной МП системы. Далее для каждой команды МП строится макрос, который представляет короткую последовательность команд, активизирующий эту команду и проверяющий правильность ее выполнения путем распространения на выходы МП одного или более результирующих машинных слов. Например, макрос для "арифметической" команды (типа сложения) организуется в три этапа:

1) загрузка значений двух операндов команды с использованием различных способов адресации (отметим, что фактические значения операндов генерируются на фазе оптимизации с использованием ГА нижнего уровня);

2) выполнение проверяемой команды;

3) распространение полученных результатов путем их записи прямо или косвенно в одно или несколько результирующих машинных слов.

Очевидно, целью макроса является получение наблюдаемых результатов для каждой команды с целью проверки правильности ее выполнения. Например, для проверки команды сложения может быть использован следующий код.

```
MOV AX, K1          ;  загрузка регистра AX константой K1
MOV AX, K2          ;  загрузка регистра BX константой K2
ADD AX, BX          ;  сложение AX и BX
MOV RW, AX          ; запись AX  в RW
MOV RW2,  PSW       ;   запись регистра состояния в RW2
```

Здесь K1 и K2 являются параметрами, чьи значения определяют собственно проверяющие свойства макроса (полноту теста), RW1 и RW2 — ячейки памяти, содержащие результаты выполнения команды.

Для проверки команд условного и безусловного переходов разрабатывается свой класс макросов. В этом случае различные значения пишутся в результирующее машинное слово для двух альтернатив. Для этого класса команд макрос также выполняется в три этапа:

1) формирование условия перехода;

2) проверка условия и возможно переход;

2) запись различных результирующих значений в зависимости от результатов проверки условия.

Ниже приведен возможный вариант подобного макроса:

```
MOV BX, 0          ;      очистка  BX
CMP AX, K1         ;      сравнение регистра AX с K1
JG      M1         ;      переход приAX > K1
MOV BX, 1          ;      запись 1 в K1
        M1:
MOV RW,  BX        ;      запись  BX в RW
```

Подобные макросы строятся для каждой команды МП системы и формируют специальную библиотеку, которая используется на следующем этапе генерации теста (для типовых МП систем библиотека содержит около 200 макросов).

Далее находятся лучшие значения параметров построенных макросов, которые обеспечивают лучшую проверку команд. Для этого используется стандартный ГА (приведенный на рис.1) со следующими параметрами. Хромосомы представляются обычными двоичными строками, длина которых определяется соответствующими макросами. При эволюции используются:1) стандартный одноточечный оператор кроссинговера; 2) стандартный оператор мутации, инвертирующий с малой вероятностью (обратно пропорциональной длине хромосомы) значение одного бита хромосомы; 3) оператор репродукции реализует пропорциональный отбор (" метод рулетки") особей для скрещивания и мутации. В качестве целевой функции используется число дефектов указанной модели, проверяемых данным макросом, которое определяется программой моделирования с неисправностями. Таким образом на первом нижнем уровне ГА строятся подпрограммы, которые проверяют неисправности отдельных команд. Далее они используются на втором верхнем уровне ГА для генерации полной тестовой программы.

На этом уровне используется следующий эволюционный алгоритм. Популяция из N последовательностей генерируется из предыдущего поколения путем мутации. При этом применяется (N + N) стратегия — N родителей генерируют N потомков. Каждая родительская последовательность производит одну дочернюю последовательность путем мутации. Отметим, что на втором верхнем уровне оператор кроссинговера не используется. Это обусловлено тем, что здесь особь является по-

следовательностью команд МП, имеющей достаточно сложную структуру. Поэтому применение традиционных операторов кроссинговера может привести к сильному разрушению этой структуры, тем самым снизив эффективность тестовой последовательности. Фактически применение оператора кроссинговера для сложной структуры эквивалентно использованию случайного оператора мутации на уровне фенотипа. В качестве целевой функции на втором уровне возможно использование: 1) число изменений битов в схеме, которое определяется программой моделирования исправного устройства; 2) число проверяемых дефектов рассматриваемого класса, вычисляемых программой моделирования неисправных схем.

Выводы. Эффективность описанного алгоритма подтверждается данными моделирования схем на уровне ЯРП, описанных на языке VHDL из международного каталога. Анализ и сравнение данных моделирования на функциональном и логическом уровнях показывает, что построенные тесты имеют высокую полноту. В тоже время генерация проверяющих тестов на функциональном уровне выполняется существенно быстрее. Представляет значительный интерес развитие рассмотренного подхода с применением на втором верхнем уровне генерации проверяющих тестов методов генетического программирования.

Литература

1. Барашко А.С., Скобцов Ю.А., Сперанский Д.В. Моделирование и тестирование дискретных устройств. — Киев: Наукова думка, 1992. — 286с.
2. S. Thatte, J.Abraham. Test generation for microprocessors // IEEE transactions on computers. — vol.29. — 1980.
3. L.Chen,S.Dey. Defuse: a deterministic functional self-test methodology for processors// Proceedings IEEE VLSI Test Symposium. — 2000. — P.255–262.
4. Иванов Д.Е., Скобцов Ю.А. Ускорение работы генетических алгоритмов при построении тестов // Искусственный интеллект. — 2001. — №1. — с.52–60.
5. F.Corno, G.Cumani, M.Sonca Reorda, G.Squillero. ARPIA: a high-level evolutionary test signal generation//3-rd Europen workshop on evolutionary competation applications to image analysis and signal processing,2001. — p.298–306.

Сдано в редакцию: .03.2003г.

Рекомендовано к печати: д.т.н., проф. Скобцов Ю.А.