Обзорная глава, постановка задачи

Содержание

1 Обзор методов генерации тестовых программ

Тестирование микропроцессоров является важной составляющей частью процесса их разработки. Тестированию может подвергаться как готовый чип, так и модель. Тестирование может проводиться как на модульном, так и на системном уровне. В данной работе речь идет о системном функциональном тестировании. Иными словами, целью тестирования является проверка правильности функционирования микропроцессора целиком. Эта проверка выполняется путем запуска на микропроцессоре специальных машинных программ (далее такие программы будут называться *тестовыми*).

Системное функциональное тестирование включает в себя следующие этапы [1]:

1. определение целей тестирования, тестового покрытия и тестовых ситуаций (структурные – какие инструкции включать в тестирование – и функциональные – как инструкции должны быть исполнены);

2. генерация тестовых программ для тестовых ситуаций;

3. исполнение тестовых программ на микропроцессоре, получение выходных данных (трасса исполнения, финальные значения регистров);

4. вынесение вердикта на основе анализа выходных данных.

Данная работа посвящена этапу генерации тестовых программ. В настоящее время в практике системного функционального тестирования микропроцессоров можно выделить следующие подходы к построению тестовых программ:

- *ручная разработка тестовых программ* хоть и практически неприменима для полного тестирования микропроцессора, всё же может применяться для тестирования особых, крайних случаев;
- *тестирование с использованием кросс-компиляции* применяется часто из-за невысокой сложности его проведения: после согласования спецификации микропроцессора можно начинать делать кросс-компилятор, а код, предназначенный для кросс-компиляции, уже готов. Однако гарантировать полноту такое тестирование не может;
- *случайная генерация тестовых программ* применяется так же часто в силу простоты автоматизации. Сгенерированные таким образом тестовые программы позволяют быстро обнаружить простые ошибки, однако не гарантируют полноты тестирования. Разрабатываются и более сложные варианты случайной генерации [6];
- *генерация тестовых программ на основе тестовых шаблонов* предполагает разделение процесса генерации тестовой программы на два этапа: на первом на основе тестовых ситуаций подготавливаются тестовые шаблоны – абстрактные представления тестовых программ – а на втором этапе по тестовым шаблонам генерируются тестовые программы.

Тестовые шаблоны могут описывать следующие свойства тестовых программ:

- заданная последовательность инструкций (только коды операций или коды операций с аргументами);
- заданная последовательность типов инструкций;
- выборка инструкций заданных типов;
- аргументы инструкций (регистры, непосредственные значения, переменные величины);
- дополнительные ограничения на инструкции;
- дополнительные ограничения на отдельные аргументы инструкций, аргументы разных инструкций;
- дополнительные функциональные ограничения на инструкции (при исполнении должны произойти некоторые заданные события).

Исследователями предложены следующие методы генерации тестовых программ на основе тестовых шаблонов:

1. ручная генерация тестовых программ;
2. комбинаторные методы;
3. использование методов генерации входных векторов (ATPG [14]);
4. использование методов разрешения ограничений.

1.1 Ручная генерация тестовых программ

Александром Камкиным разработана технология системного функционального тестирования микропроцессоров с использованием тестовых шаблонов [1]. Построение тестовых шаблонов осуществляется полуавтоматически на основе тестового покрытия по модели системы инструкций микропроцессора. Тестовые шаблоны представляют из себя последовательность инструкций с зависимостями между аргументами (например, «запись-чтение») и тестовыми ситуациями для инструкций.

Для получения тестовых программ по сгенерированным тестовым шаблонам следует реализовать на языке Java *конструкторы тестовых данных*. Под «тестовыми данными» понимаются значения регистров, аргументы инструкций обращения к памяти для инициализации состояния кэш-памяти и ячеек оперативной памяти, если это требуется. Все зависимости в тестовом шаблоне обладают направлением, конструирование аргументов инструкций производится итеративно от инструкций, которые не зависят от остальных инструкций, к инструкциям, которые зависят от уже сгенерированных инструкций. Для выбора независимых значений используется случайная генерация.

1.2 Комбинаторные методы генерации тестовых программ

Тестовый шаблон состоит из заданной последовательности инструкций, аргументами которых являются переменные величины. Кроме того для каждой переменной величины указывается конечная область значений. Все значения в области равноправны. Тестовая программа содержит ту же последовательность инструкций, а для каждого аргумента выбрано значение из области значений этого аргумента. В комбинаторных методах инструкции воспринимаются лишь как синтаксические объекты (термы) – у них есть лишь имя и аргументы (возможно типизированные).

Последовательность инструкций может быть задана неявно, но у каждой инструкции всё же будут переменные величины в качестве аргументов и для каждой переменной величины задана область значений. Исследователи из Fujitsu Lab. [7] предлагается описать последовательность инструкций в виде выражений (Test Specification Expressions, TSE), а семантику инструкций – на языке ISDL [8]. Отдаленно TSE могут напоминать регулярные выражения, где бесконечнозначные операции заменены конечными аналогами. ISDL-описание может включать в том числе и параметры исполнения инструкции на конвейере, которые могут быть использованы в TSE. Авторы исследования реализовали специальный генератор, который строит тестовые программы, удовлетворяющие данному TSE.

Kohno и Matsumoto [11] рассматривают задачу верификации конвейерных микропроцессоров, используя для этого генерацию тестовых программ с помощью тестовых шаблонов. Тестовый шаблон явно содержит последовательность типов инструкций, возможно, с использованием конструкций итерирования блоков инструкций. Использование разными инструкциями в шаблоне одной и той же переменной величины должно приводить в тестовой программе к использованию одного и того же значения для этой переменной величины. Областями значений являются заданное в архитектуре множество регистров (GPR – множество регистров общего назначения, CPR – множество регистров сопроцессора).

1.3 Генерация тестовых программ с использованием методов решения задачи ATPG

Задача ATPG (Automatic Test Pattern Generation) [14] относится к вопросам модульного тестирования микропроцессоров. Модульное тестирование осуществляется подачей определенных сигналов

(возможно, многотактовых) на входы модуля (схемы) и снятие значения выходных сигналов (возможно, также многотактовых). Принятие вердикта осуществляется на основе сравнения ожидаемого выходного сигнала и снимаемого с данной схемы. Тестовым воздействием является сигнал, поданный на входные порты схемы. Моделью ошибки является смена функции некоторых элементов схемы (например, в результате пробоя или замыкания элемент может сменить функцию, которую он реализует, на тождественную константу). ATPG – это задача построения тестовых воздействий для схем, нацеленных на данную модель ошибки. Аргументы инструкций являются входными сигналами некоторых модулей микропроцессора, поэтому решая задачу генерации входных сигналов, можно решать и задачу генерации тестовых программ.

Эту идею использовали исследователи из Politecnico di Milano [12]. Тестовым шаблоном выступает препроцессированный модель фазы декодирования инструкции на языке VHDL [4]. Специальный генератор подставляет на место кода инструкции заданные значения кодов операций и передает получившуюся модель стороннему (коммерческому) ATPG-инструменту. Тот в свою очередь возвращает остальные значения, которые надо передать в модуль декодирования инструкции, т.е. значения аргументов инструкции. Метод был применен к тестированию АЛУ VLIW-микропроцессора.

1.4 Генерация тестовых программ с использованием методов разрешения ограничений

Под *ограничением* будет пониматься предикат, в котором переменные принимают значения из конечной области. Например, $x > 0$, если $x \in \{0, 10, 100\}$. Задачей разрешения ограничений (constraint satisfaction problem) является задача поиска значений для переменных из их областей значений, при которых все ограничения выполнены [15]. Для областей значений небольшого размера достаточно перебрать все комбинации значений переменных, пока не встретится комбинация, на которой выполнены все ограничения. В общем случае применяются более сложные алгоритмы (зачастую с привлечением эвристик), сочетающие перебор с возвратом и распространение ограничений (т.е. автоматический вывод ограничений-следствий по данной системе ограничений).

Представление в виде CSP удобно для задач, сформулированных в виде задачи выполнимости некоторого набора условий. Задача генерации тестовых программ по тестовым шаблонам тоже может быть сформулирована в таком виде, поскольку есть связанный набор переменных (инструкций, аргументов инструкций, элементов состояния микропроцессора), причем связи выражаются в виде утверждений, зависимостей. Сама идея построения тестовой программы через формулирование тестового шаблона близка решению задачи с использованием CSP, поскольку этап построения тестового шаблона (формализации требований к тестовому воздействию) по сути является этапом формулирования задачи построения тестового шаблона в виде утверждений, в виде задачи выполнимости. Остается только перевести эту формулировку к виду, используемому в инструментах для решения CSP. Выбор инструментов, метод их решения, а также вида самих ограничений, зависит от того, какие применяются тестовые шаблоны и как описывается семантика инструкций.

С целью упрощения подготовки нужного представления семантики микропроцессора китайские исследователи в своем инструменте MAATG [16] предложили использовать хорошо известный язык описания архитектуры EXPRESSION [9]. Тестовые шаблоны позволяют явно задавать блоки инструкций, задавать ограничения на аргументы разных инструкций (одинаковые регистры, разные регистры, непосредственные значения из некоторого множества констант), а также указывать события, которые могут произойти при исполнении инструкции (например, целочисленное переполнение для инструкции ADD). Специальный генератор строит тестовую программу итеративно. Сначала он

упорядочивает инструкции так, чтобы переменные для очередной инструкции зависели только от переменных предыдущих инструкций. Это позволяет разбить задачу генерации тестовой программы на последовательность более простых задач генерации одной инструкции. Однако по доступным публикациям невозможно сделать вывод о том, какие ограничения генерирует MAATG и тем самым оценить эффективность работы этого инструмента.

Еще одно семейство инструментов генерации тестовых программ на основе тестовых шаблонов было разработано в IBM в течение последних 20 лет. Далее будет дано описание последнего на сегодняшний день инструмента в этой семействе – Genesys-Pro [13]. Тестовые шаблоны этого инструмента позволяют описывать как заданные последовательности инструкций, так и всевозможные их композиции. Разработчиками предложен несложный императивный язык, позволяющий задать эту последовательность инструкций. Для каждой инструкции могут быть указаны эвристики для выбора аргументов (обязательные и необязательные). Кроме того, тестовый шаблон может содержать параметры работы генератора тестовых программ (вероятности выбора тех или иных значений, параметры распределения адресов в памяти и др.). Вероятности выбора тех или иных значений играют важную роль при разрешении системы ограничений, поскольку они позволяют задать дополнительные требования на значение переменной в тех случаях, когда решатель CSP выделяет некое множество допустимых значений для переменной, в котором более чем одно значение. Эвристики для выбора аргументов, используемые в тестовых шаблонах, предопределены в Genesys-Pro, задавать пользовательские эвристики нельзя (это объясняется тем, что эвристики нетривиальным образом используются при разрешении ограничений, поэтому внесение новой эвристики не является простой задачей). Инструмент предлагает описать особенности системы команд микропроцессора с использованием ограничений, эти ограничения в дальнейшем будут использованы для генерации тестовых программ [2]. Для описания механизма трансляции адресов (получения физического адреса по виртуальному) предлагается использовать подход DeepTrans [3]. Кроме того, для генерации тестовой программы требуется законченная модель микропроцессора для запуска ее на симуляторе.

Рассмотрим теперь, как Genesys-Pro генерирует тестовые программы на основе тестовых шаблонов. Сначала строится явная последовательность инструкций на основе ее описания из тестового шаблона (получается т.н. «поток инструкций»). Далее итеративно для каждой следующей инструкции потока генерируются аргументы инструкций. Для этого строится задача разрешения ограничений (с учетом описания системы команд, тестового шаблона, эвристик и текущего модельного состояния микропроцессора), эта задача разрешается, в результате чего получаются значения аргументов. Готовая инструкция исполняется на модели микропроцессора (с использованием симулятора) с получением нового модельного состояния микропроцессора, что и завершает очередную итерацию. Ключевым моментом является эффективность работы решателя ограничений. Для этой цели разработчики инструмента самостоятельно написали свой решатель. Он базируется на хорошо известном семействе алгоритмов разрешения ограничений MAC (Maintaining Arc-Consistency) [15], но заточен под ограничения, генерируемые для тестовых программ [5]. Написание такого решателя является довольно нетривиальной задачей и предметом отдельного исследования.

1.5 Сравнение методов генерации тестовых программ

Сравнение проводилось по следующим критериям:

1. выразительная мощность тестовых шаблонов;

2. допустимые архитектурые механизмы;

3. сложность подготовки исходных данных;

4. переиспользуемость частей генератора тестовых программ;

5. вычислительная эффективность генератора тестовых программ.

1.5.1 Выразительная мощность тестовых шаблонов

	ручная генерация	комбинаторные методы	использование ATPG	использование CSP
возможность задать последовательность инструкций	+	+	–	+
возможность задать тестовую ситуацию	+	–	–	+
возможность задействовать состояние микропроцессора	+	–	+	+

//нужен комментарий, почему заполнение такое?

1.5.2 Допустимые архитектурные механизмы

	ручная генерация	комбинаторные методы	использование ATPG	использование CSP
поддержка регистров общего назначения	+	+	–	+
поддержка кэш-памяти и трансляции адресов	+	–	–	+
поддержка механизмов параллелизма	+	+	–	+

//нужен комментарий, почему заполнение такое?

1.5.3 Сложность подготовки исходных данных

	ручная генерация	комбинаторные методы	использование ATPG	использование CSP
при смене тестового шаблона	сложно	сложно (нужна RTL-модель)	–	несложно
при смене микропроцессора	долго, но технологично	долго вплоть до невозможности	просто	небыстро (новые механизмы)

//нужен комментарий, почему заполнение такое?

1.5.4 Переиспользуемость частей генератора тестовых программ

	ручная генерация	комбинаторные методы	использование ATPG	использование CSP
при смене тестового шаблона	никакая	полная	полная	полная
при смене микропроцессора	никакая	полная	полная	только общие механизмы

//нужен комментарий, почему заполнение такое?

1.5.5 Вычислительная эффективность генератора тестовых программ

	ручная генерация	комбинаторные методы	использование ATPG	использование CSP
сложность достижения покрытия	сложно (можно достичь крайние случаи)	сложно (не задается семантика)	возможно покрытие структуры RTL-модели	несложно
среднее время генерации программ	долго	быстро	зависит от сложности RTL-модели	недолго
эффективность учёта начального состояния микропроцессора	сложно	не учитывается	не учитывается	возможно

//нужен комментарий, почему заполнение такое?

2 Постановка задачи

В диссертации решается задача построения тестовых программ по тестовым шаблонам, обладающим следующими свойствами. Тестовый шаблон представляется последовательностью троек (I_i, A_i, S_i),

где I_i – заданная инструкция, A_i – список аргументов инструкции, S_i – тестовая ситуация инструкции. Аргументами являются явно заданные регистры и переменные, не меняющие своего значения (в тестовой программе они станут непосредственными значениями). Тестовая ситуация инструкции – это ограничение на аргументы инструкции и текущее состояние микропроцессора. Примеры тестовых ситуаций: «при исполнении инструкции должно произойти целочисленное переполнение (это ограничение только на аргументы инструкции)», «при исполнении инструкции должно произойти кэш-попадание в кэш-памяти первого уровня» (это ограничение не только на аргументы инструкции, но и на состояние микропроцессора перед ее исполнением, поскольку кэш-память является подсистемой микропроцессора). Кроме тестового шаблона задано начальное состояние модели микропроцессора. Состояние модели микропроцессора включает в себя состояние всех его подсистем. Например, состоянием регистра является значение, которое в нем хранится. Состоянием кэш-памяти является его содержимое.

Требуется построить тестовую программу по тестовому шаблону, которая состоит из двух частей: инициализирующие инструкции и инструкции тестового воздействия (см.рис. 1). Инициализирующие инструкции переводят модель микропроцессора из заданного начального состояния в состояние, необходимое для тестового воздействия. Инструкции тестового воздействия в точности повторяют последовательность инструкций тестового шаблона с заменой переменных на непосредственные значения. На рисунке 2 приведен пример тестового шаблона и возможных инструкций тестового воздействия, построенных по этому шаблону.

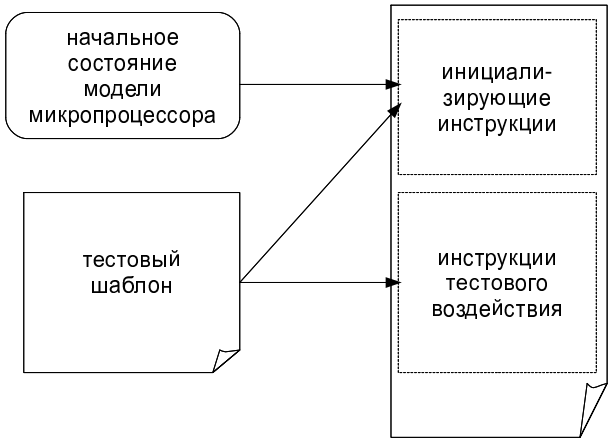

Рис. 1: Составление тестовой программы

```
AND r1, r2, r3 @ normal        AND r1, r2, r3
LD r4, r2, c1 @ l1Hit          LD r4, r2, 0x0FA2
SUB r3, r1, r5 @ overflow      SUB r3, r1, r5
```

Рис. 2: Тестовый шаблон и возможные соответствующие ему инструкции тестового воздействия

Уже сгенерированная программа может быть позднее дополнена инструкциями проверки состояния микропроцессора после исполнения инструкций тестового воздействия.

Инициализирующие инструкции призваны подготовить модель микропроцессора к исполнению

инструкций тестового воздействия. Без инициализирующих инструкций запуск инструкций тестового воздействия даже на корректной модели микропроцессора может приводить к ложным сообщениям об ошибках в модели. В работе рассматривается модель микропроцессора, включающая в себя регистры общего назначения, кэш-память (возможно многоуровневую) и буфер трансляции адресов (TLB, Translation Lookaside Buffer) [10]. Таким образом, инициализирующие инструкции могут включать инструкции изменения значений регистров и ячеек кэш-памяти и TLB.

В данной работе среди методов генерации тестовых программ выбран метод, использующий разрешение ограничений (CSP). Однако по сравнению с существующими аналогами в данной работе поставлена задача исследовать возможности снижения сложности подготовки генератора тестовых программ (по сравнению, например, с мощным Genesys-Pro), не проиграв сильно в масштабируемости генератора. При этом, возможно, придется выделить среди всевозможных архитектурных механизмов наиболее часто использующиеся и требующие тестирование в современных микропроцессорах.

3 Предварительные сведения и термины

3.1 Типы кэш-памяти

По организации кэш-память делят на *полностью ассоциативную*, *прямого доступа* и *наборно-ассоциативную*. Различие производится на основе двух параметров: количества секций W и количества наборов R. Кэш-память хранит некоторый набор данных. Каждому блоку данных соответствует некоторый адрес (физический или виртуальный). Блоки с адресами организованы в *секции* и *наборы* (см.рис. 3).

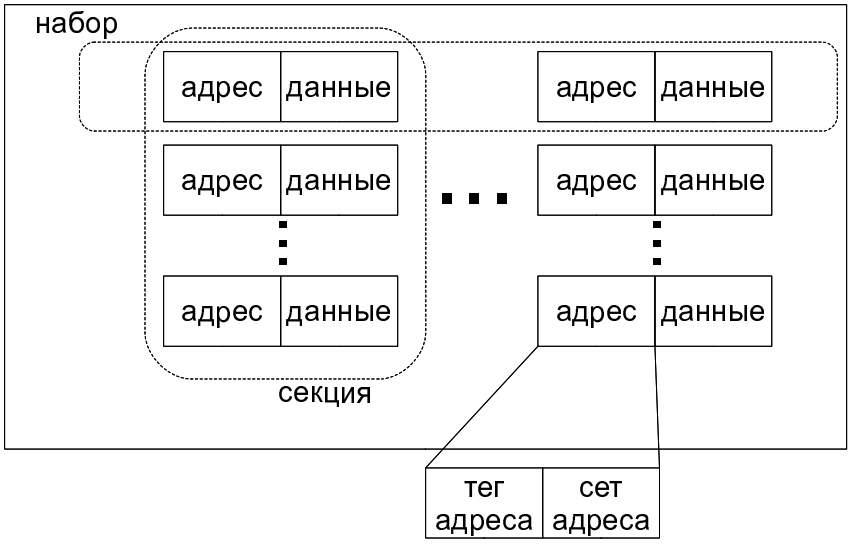

Рис. 3: Модель кэш-памяти и адреса данных

Каждый адрес может быть разделен на два битовых поля: поле *тега адреса* и поле *сет адреса*. Один набор составляют адреса с одинаковым сетом. Кэш-память организована таким образом, что

для каждого сета хранится всегда одно и то же количество адресов (равное количеству секций W). Адреса всех данных в кэш-памяти различные. Отсюда следует, что теги адресов одного набора разные. В кэш-памяти представлены все наборы, возможные в рамках битового поля сета адреса.

Кэш-память является полностью ассоциативной, если $R = 1$. Кэш-память является кэш-памятью прямого доступа, если $W = 1$. И кэш-память является наборно-ассоциативной, если $R > 1$ и $W > 1$.

Инструкции обращения в память бывают двух видов: инструкции загрузки данных из памяти по данному адресу и инструкции сохранения данных в памяти по данному адресу. При выполнении этих инструкций может быть задействована кэш-память. Если данные по требуемому адресу присутствуют в кэш-памяти, операция проводится с нею. Такая ситуация называется *кэш-попаданием*. Если данные по требуемому адресу не присутствуют в кэш-памяти, осуществляется подгрузка данных в кэш-память и совершение операции. Такая ситуация называется *кэш-промахом*. В этом случае если кэш-память полностью заполнена, некоторые данные должны быть *вытеснены* из кэш-памяти и на их место будут загружены данные по требуемому адресу. *Стратегия вытеснения* (или *политика замещения*) – это правило, по которому определяются вытесняемые данные. Например, могут быть вытеснены данные, которые дольше всего не были нужны (такая стратегия называется LRU), или данные, которые были внесены в кэш-память раньше остальных (такая стратегия называется FIFO).

Список литературы

[1] А.С. Камкин. Комбинаторная генерация тестовых программ для микропроцессоров на основе моделей. *Препринт Института Системного Программирования РАН*, 21, 2008.

[2] A.Adir, E.Almog, L.Fournier, E.Marcus, M.Rimon, M.Vinov, and A.Ziv. Genesys-pro: Innovations in test program generation for functional processor verification. *IEEE Design and Test of Computers*, 21(2):84–93, Mar/Apr 2004.

[3] Allon Adir, Roy Emek, Yoav Katz, and Anatoly Koyfman. Deeptrans - a model-based approach to functional verification of address translation mechanisms. In *MTV*, pages 3–6, 2003.

[4] Peter J. Ashenden. *The Designer's Guide to VHDL*. Elsevier, 2008.

[5] Eyal Bin, Roy Emek, Gil Shurek, and Avi Ziv. Using a constraint satisfaction formulation and solution techniques for random test program generation. *IBM Systems Journal*, 41(3):386–402, 2002.

[6] F. Corno, E. Sanchez, M.S. Reorda, and G. Squillero. Automatic test program generation – a case study. *IEEE Design & Test, Special issue on Functional Verification and Testbench Generation*, 21(2):102–109, March-April 2001.

[7] F.Fallah and K.Takayama. A new functional test program generation methodology. *Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 76–81, 2001.

[8] S. Hanono G. Hadjiyiannis and S. Devadas. Isdl: An instruction set description language for retargetability. *Proceedings of the 34th Design Auiomation Conference*, pages 299–302, June 1997.

[9] Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt, and Alex Nicolau. Expression: A language for architecture exploration through compiler/simulator retargetability. In *In Proceedings of the European Conference on Design, Automation and Test*, pages 485–490, 1999.

[10] Andrea C. Arpaci-Dusseau John L. Hennessy, David A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 4 edition, 2007.

[11] K.Kohno and N.Matsumoto. A new verification methodology for complex pipeline behavior. *Proceedings of the 38st Design Automation Conference (DAC'01)*, 2001.

[12] M.Beardo, F.Bruschi, F.Ferrandi, and D.Sciuto. An approach to functional testing of vliw architectures. *Proceedings of the IEEE International High-Level Validation and Test Workshop (HLDVT'00)*, pages 29–33, 2000.

[13] M.Behm, J.Ludden, Y.Lichtenstein, M.Rimon, and M.Vinov. Industrial experience with test generation languages for processor verification. *Proceedings of the 41st Design Automation Conference (DAC'04)*, 2004.

[14] Alexander Miczo. *Digital logic testing and simulation*. Wiley-Interscience; 2 edition, 2003.

[15] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[16] T.Li, D.Zhu, Y.Guo, G.Liu, and S.Li. Maatg: A functional test program generator for microprocessor verification. *Proceedings of the 2005 8th Euromicro conference on Digital System Design (DSD'05)*, 2005.