МЕТОД ИЗВЛЕЧЕНИЯ РАСШИРЕННЫХ КОНЕЧНЫХ АВТОМАТОВ ИЗ HDL-ОПИСАНИЙ

С.А. Смолов

Институт системного программирования РАН

Аннотация

В статье предлагается новый метод извлечения расширенных конечных автоматов из HDL-описаний цифровой аппаратуры, а также представлены результаты апробации метода на модулях небольшой сложности.

Введение

В настоящее время сложность цифровой микроэлектронной аппаратуры неуклонно возрастает. Рост обусловлен, в том числе, активным повторным использованием модулей, спроектированных на специализированных языках описания аппаратуры (Hardware Description Language, HDL), таких как VHDL и Verilog (в дальнейшем представления модулей на языках семейства HDL будем называть HDL-описаниями). С учетом увеличения объема кода аппаратных систем, актуальной является задача разработки методов и инструментальных средств автоматизированной верификации (проверки функциональной корректности) HDL-описаний.

Распространенный подход к верификации модулей аппаратуры основан на использовании формальных моделей – математических абстракций, описывающих структуру и\или поведение проверяемой системы. Примерами таких моделей являются сети Петри и конечные автоматы. Модели могут разрабатываться вручную (на основе анализа требований) или извлекаться из HDL-описаний автоматически. В последнем случае снижается риск внесения дополнительных ошибок, а для проверки корректности получаемых моделей достаточно доказать корректность используемого алгоритма извлечения.

В статье рассматривается метод второго типа, реализующий автоматическое извлечение моделей типа расширенных конечных автоматов (Extended Finite State Machine, EFSM)[1] из HDL-описаний. По сравнению с классическим конечным автоматом (Finite State Machine, FSM), расширенный конечный автомат (далее EFSM-модель) обладает двумя особенностями. Во-первых, в дополнение к множеству состояний, EFSM-модель содержит множество переменных (входных, внутренних, выходных). Во-вторых, все переходы EFSM-модели снабжены охранными условиями (guards) на значения входных и внутренних переменных, а также действиями (actions) по изменению значений внутренних и выходных переменных. Переход EFSM-модели может сработать, только если выполнено его охранное условие; при срабатывании перехода выполняется соответствующее действие.

EFSM-модели являются удобным формализмом для описания структуры и поведения цифровой аппаратуры, поскольку в них управляющая логика (control) естественным образом отделена от функций преобразования данных (datapath). Этим обусловлено активное использование EFSM-моделей для решения различных задач, связанных с верификацией цифровой аппаратуры: для построения тестовых наборов, проверяющих соответствие системы требованиям; для генерации тестовых последовательностей, нацеленных на маловероятные ситуации в поведении системы; для формальной проверки свойств.

Статья организована следующим образом. В разделе «Обзор работ» описываются существующие методы извлечения EFSM-моделей из HDL-описаний. В следующем разделе описывается предлагаемый метод построения моделей. Заключение завершает статью.

Обзор работ

Исторически одними из первых были разработаны алгоритмы построения FSM-моделей по исходному коду (на них, в частности, базируются методы логического синтеза [2]), однако получаемые при их использовании модели не всегда адекватны для целей верификации [3]. Состояниям в таких моделях соответствуют операторы (точки) в исходном коде, в которых выполняется ожидание входных событий; при выделении состояний никак не учитываются заданные в коде соотношения между переменными (условия ветвления, выражения в присваиваниях и т.п.).

В работе [3] описывается метод извлечения «простых для обхода» (easy-to-traverse) EFSM-моделей из HDL-описаний. Метод состоит из четырех этапов. На первом этапе, используя известный алгоритм [2], строится начальная EFSM-модель (Reference EFSM, REFSM). В общем случае полученная модель «трудна для обхода», в частности, из-за того, что содержит условные операторы в действиях переходов. На втором этапе в REFSM-модель добавляются промежуточные состояния, а переходы декомпозируются таким образом, чтобы их действия не содержали ветвлений. Полученная модель (Largest EFSM, LEFSM), строго говоря, не эквивалентна исходной модели — один шаг работы REFSM может соответствовать нескольким шагам в LEFSM. Для обеспечения временно*й* эквивалентности с REFSM в LEFSM выполняется расщепление промежуточных состояний и объединение совместимых переходов. В результате образуется SEFSM-модель (Smallest EFSM). На завершающем этапе выполняется частичная стабилизация SEFSM-модели, нацеленная на устранение зависимостей охранных условий переходов от переменных, кодирующих состояния. Результаты экспериментов демонстрируют эффективность подхода, однако процедура построения EFSM-модели вызывает вопросы. Во-первых, представляется слишком жестким то ограничение, что для одного процесса HDL-описания строится одна EFSM-модель. Во-вторых, процесс построения модели представляется чрезмерно усложненным: аналогичных результатов можно добиться более простыми средствами, если с самого начала определить, какие внутренние переменные описывают состояние автомата.

Предлагаемый метод

Предлагаемый метод извлечения EFSM-моделей из HDL-описаний содержит следующие основные шаги:

1) Построение внутреннего представления;
2) Анализ внутреннего представления:
   a. Идентификация синхросигналов;
3) Трансформация в систему охраняемых действий;
4) Идентификация переменных состояния;
5) Построение пространства состояний EFSM-модели;
6) Построение отношения переходов EFSM-модели.

Способы реализации Шага 1 широко известны [4]. Шаг 2 состоит в анализе потоков данных между инструкциями и выявлении т.н. *синхросигналов* – входных однобитных сигналов модуля, необходимых не для обмена данными между модулями, а для уведомления об истечении очередного периода («такта») времени. В предлагаемом методе синхросигналы выявляются с помощью эвристики. На Шаге 3 внутреннее представление преобразуется в систему т.н. *охраняемых действий*. Охраняемое действие – это пара «охранное условие - действие». Преобразование выполняется путем построения для каждого пути выполнения во внутреннем представлении соответствующего охраняемого действия. В простейшем случае охранное условие содержит условный оператор верхнего уровня, а действие – последовательность вложенных присваиваний. Если путь выполнения содержит вложенные условные операторы, то осуществляется их «подъем» методом обратных подстановок [5]. На Шаге 4 проводится анализ потоков данных в системе охраняемых действий с целью определения *переменных состояния* – внутренних переменных, используемых для организации потоков управления в процессах исходного HDL-описания. Переменные состояния, подобно синхросигналам, определяются с помощью соответствующей эвристики.

На Шаге 5 строится пространство состояний EFSM-модели. Для этого из системы охраняемых действий извлекается множество условий, зависящих от переменных состояния. Путем ортогонализации [6] множество разбивается на попарно несовместные условия. Противоречивые условия обнаруживаются с помощью средств проверки выполнимости ограничений и исключаются из дальнейшего рассмотрения. Все совместные условия трактуются как состояния EFSM-модели.

Финальный Шаг 6 метода заключается в построении отношения переходов EFSM-модели. Для каждой пары состояний и каждого охраняемого действия проверяется совместимость охранного условия с первым состоянием пары, а результата действия — со вторым; при положительных результатах проверок соответствующий переход добавляется в конструируемую модель.

Описанный метод извлечения EFSM-моделей из исходного кода HDL-описаний был реализован в прототипе инструмента HDL Retrascope [7]. Разработка выполнена на языке программирования Java с использованием средств Z3 (SMT-решатель) [8], zamiaCAD (платформа для разработки и анализа HDL-описаний) [9] и Fortress (библиотека работы с формулами) [10]. Прототип позволяет анализировать описания цифровой аппаратуры на синтезируемых подмножествах (synthesizable subsets) языков VHDL и Verilog, строить и визуализировать систему расширенных конечных автоматов, моделирующих процессы HDL-описаний (в текущей версии не поддерживается анализ описаний, имеющих итеративные циклы и\или иерархическую структуру).

В рамках работы были проанализированы HDL-описания, входящие в пакет тестов ITC’99 [11]. EFSM-модели были извлечены для 13 описаний из 22 — остальные используют конструкции языка VHDL, не поддержанные в прототипе на момент проведения экспериментов. Для всех успешно обработанных описаний были извлечены синхросигналы, и во всех случаях их множества включали *CLOCK* и *RESET*. Для всех описаний все переменные, в имени которых присутствует подстрока *STAT*, были идентифицированы как переменные состояния (такие имена позволяют предположить, что они используются инженерами-проектировщиками для кодирования состояний управляющих автоматов).

Отметим, что для всех проанализированных HDL-описаний число состояний в извлеченных автоматах относительно невелико и растет с ростом числа выявленных переменных состояния. Число переходов также растет с увеличением числа состояний и числа путей выполнения.

Заключение

Расширенные конечные автоматы активно используются в области имитационной и формальной верификации цифровой аппаратуры. В работе рассмотрен метод извлечения расширенных конечных автоматов из исходного кода HDL-описаний. Отличительной чертой метода является автоматическое выделение внутренних переменных, представляющих состояния устройства, а также использование техник символического выполнения для построения множества состояний и отношения переходов. Эксперименты показали применимость подхода к HDL-описаниям небольшой сложности (до 1000 строк кода). В ближайшем будущем планируется испытать созданный прототип для описаний средней и повышенной сложности (порядка 10 000 строк кода), а также провести сравнение метода с другими подходами.

Список литературы

[1] Holzmann G.J. Design and Validation of Computer Protocols. PrenticeHall, 1990.512 p.

[2] Giomi J.-C. Finite State Machine Extraction from Hardware Description Languages // ASIC Conference and Exhibition, 1995. P. 353-357.

[3] Guglielmo G., Guglielmo L., Fummi F., Pravadelli G. Efficient Generation of Stimuli for Functional Verification by Backjumping Across Extended FSMs // Journal of Electronic Testing, 2011. № 27(2). P. 37-162.

[4] Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д. Компиляторы: принципы, технологии и инструментарий. М.: ООО «И.Д. Вильямс», 2011. 1184 с.

[9] Floyd R.W. Assigning Meaning to Programs // Symposium on Applied Mathematics, 1967. P. 19-32.

[6] Закревский А.Д. Параллельные алгоритмы логического управления. Мн.: Институт технической кибернетики АН Беларуси, 1999. 202 с.

[7] Retrascope. http://forge.ispras.ru/projects/retrascope

[8] Z3. http://z3.codeplex.com

[9] zamiaCAD. http://zamiacad.sourceforge.net

[10] Fortress. http://forge.ispras.ru/projects/solver-api

[11] ITC’99. http://www.cad.polito.it/downloads/tools/itc99.html