УДК 004.05

Метод извлечения EFSM-моделей из HDL-описаний: применение к функциональной верификации

А.С. Камкин, С.А. Смолов

Институт системного программирования РАН, ssedai@ispras.ru

***Аннотация* — Сложность цифровой микроэлектронной аппаратуры неуклонно возрастает, что существенно затрудняет ее верификацию — проверку корректности. Автоматизированные методы верификации аппаратуры основаны на использовании моделей, удобных для генерации тестов и формальной проверки свойств. В данной статье описывается новый подход к извлечению моделей в форме расширенных конечных автоматов из исходного кода HDL-описаний, а также обсуждаются возможности использования таких моделей в процессе верификации.**

***Ключевые слова* — проектирование аппаратуры, функциональная верификация, статический анализ, генерация тестов, синтез автоматов, расширенные конечные автоматы, охраняемые действия.**

I. Введение

Функциональная верификация является одним из наиболее трудоемких и дорогостоящих этапов в процессе проектирования цифровой микроэлектронной аппаратуры [1]. Для автоматизации верификации широко используются модели — математические абстракции, описывающие структуру и/или поведение разрабатываемой системы. Примерами типов моделей, применяемых для проектирования аппаратуры, являются конечные автоматы и сети Петри [2]. Модели могут строиться на основе анализа требований (технического задания, внутренней документации и т.п.) либо извлекаться из описаний, выполненных на специализированных языках (Hardware Description Language, HDL), например, VHDL или Verilog [3]. В статье рассматривается метод второго типа и его применение к верификации. Для моделирования аппаратуры используется формализм расширенных конечных автоматов (Extended Finite State Machine, EFSM).

Расширенные конечные автоматы (EFSM-модели) устроены следующим образом. Во-первых, в дополнение к конечному множеству состояний, имеющемуся в классических автоматах (Finite State Machine, FSM), они содержат множество переменных (входных, внутренних и выходных). Во-вторых, в EFSM-модели переходы между состояниями снабжены охранными условиями (guards) на значения переменных (входных и внутренних) и действиями (actions) по изменению значений переменных (внутренних и выходных). Переход автомата может сработать, только если выполнено его охранное условие; при срабатывании перехода выполняется соответствующее действие. В EFSM-моделях управляющая логика (control) естественным образом отделяется от функций преобразования данных (datapath), как это принято при проектировании цифровой аппаратуры [4]. Являясь адекватным формализмом для моделирования широкого класса систем, расширенные конечные автоматы активно используются в верификации: для построения тестовых наборов, проверяющих соответствие системы требованиям [5]; для генерации тестовых последовательностей, нацеленных на маловероятные ситуации в работе системы [6] (в таких ситуациях могут проявляться трудно обнаруживаемые ошибки [7]); для формальной проверки свойств системы [8].

В работе предлагается метод извлечения EFSM-моделей из исходного кода HDL-описаний, ориентированный на решение задач верификации. Разработка подхода мотивировалась следующими соображениями. Во-первых, автоматическое построение модели по исходному коду позволяет избежать ошибок, имеющих место при ручном моделировании; упрощает поддержку систем верификации (при изменении кода модель и некоторые части тестового окружения могут быть перестроены автоматически); повышает точность и нацеленность верификации. Во-вторых, расширенные конечные автоматы являются хорошо изученными математическими объектами, для которых разработаны эффективные методы анализа — представляется перспективным адаптировать имеющийся арсенал методов и инструментов для HDL-описаний. В-третьих, EFSM-модели представляются удобным средством для интеграции различных техник верификации аппаратуры, как имитационных, так и формальных.

Статья организована следующим образом. В разделе II делается обзор работ по извлечению EFSM-моделей из HDL-описаний и их применению к верификации. В разделе III определяются основные понятия, используемые в работе (охраняемое действие, расширенный конечный автомат и др.). В разделе IV описывается предлагаемый метод построения моделей; раздел разбит на два подраздела, посвященных построению системы охраняемых действий по HDL-описанию и структурированию этих действий в EFSM-модели. В разделе V приводятся результаты экспери-

ментов. В разделе VI делается заключение и обсуждаются направления дальнейших исследований.

II. Обзор работ

Несмотря на то, что имеется большое число работ, посвященных использованию EFSM-моделей для верификации программных и аппаратных систем, существует не так много подходов, в которых такие модели извлекаются непосредственно из исходного кода HDL-описаний. Алгоритмы построения EFSM-моделей по исходному коду известны и широко используются в современных САПР [9] (на них базируются методы логического синтеза), однако получаемые при их использовании модели не всегда удобны для верификации [6] (подробнее об этом будет рассказано ниже).

В работе [10] рассматривается среда мутационного тестирования (mutation testing) FAST. Мутационное тестирование — это метод оценки адекватности тестовых наборов, основанный на внесении небольших изменений (мутаций) в исходный код: если тесты не в состоянии обнаружить такие изменения, они считаются неполными. HDL-описания с внедренными в них ошибками автоматически транслируются средой FAST в событийно эквивалентные модели уровня транзакций (Transaction Level Model, TLM). Цель этого преобразования состоит в ускорении прогона тестов для измененных описаний. Ключевой частью работы является метод абстракции — преобразования HDL-описаний уровня регистровых передач (Register Transfer Level, RTL) в TLM-представления, — в основе которого используются EFSM-модели. Анализируя извлеченный из HDL-описания автомат среда FAST идентифицирует операции над данными (computational phases) — пути в графе состояний, включающие действия по получению входной информации, ее обработке и вычислению выходного результата. Абстракция осуществляется путем объединения состояний автомата, относящихся к одному этапу одной операции.

Ряд работ (например, [5], [6], [7], [11]) посвящен проблеме генерации тестов на основе EFSM-моделей. Если модель извлекается из исходного кода HDL-описания, сгенерированные тесты обеспечивают высокий уровень покрытия кода (code coverage). Основным подходом к построению тестов на основе автоматных моделей является обход графа состояний — построение пути (или набора путей), содержащего все состояния и переходы автомата [12]. В отличие от классических конечных автоматов при обходе EFSM-моделей имеется сложность, связанная с наличием у переходов охранных условий. Если условия зависят не только от входных переменных, но и от внутренних, определение достижимости состояний становится вычислительно трудной задачей. Существуют техники преобразования EFSM-моделей, которые позволяют устранять (или минимизировать) такие зависимости ([5], [11], [13]), но они, вообще говоря, приводят к комбинаторному взрыву числа состояний. Альтернативу им составляют подходы на основе поиска с возвратом (backtracking, backjumping) [6].

В работе [6] описывается метод извлечения «простых для обхода» (easy-to-traverse) EFSM-моделей из HDL-описаний. Метод состоит из четырех основных этапов. На первом этапе (для каждого процесса, заданного в описании), используя известный алгоритм [9], строится начальная EFSM-модель (Reference EFSM, REFSM). В общем случае полученная модель «трудна для обхода» (hard-to-traverse), поскольку 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 такт работы REFSM может соответствовать нескольким тактам в LEFSM. Для обеспечения временной эквивалентности для последней модели выполняется расщепление промежуточных состояний и объединение совместимых переходов. В результате образуется SEFSM-модель (Smallest EFSM). На завершающем этапе, используя метод [11], выполняется частичная стабилизация SEFSM-модели, нацеленная на устранение зависимостей охранных условий переходов от переменных, кодирующих состояния. Результатом является S^2EFSM-модель (Semi-Stabilized EFSM), которая эквивалентна исходной модели и, по утверждению авторов, является «простой для обхода».

Результаты экспериментов, представленные в работе [6], демонстрируют эффективность подхода для решения задач нацеленной генерации тестов, однако процедура построения EFSM-модели по исходному коду HDL-описания вызывает вопросы. Во-первых, представляется слишком жестким ограничение, согласно которому для одного процесса HDL-описания строится одна EFSM-модель: с одной стороны, один логический блок аппаратуры (по сути, автомат) может быть определен с помощью нескольких процессов (такие процессы используют общие переменные и работают в режиме взаимного исключения); с другой стороны, в рамках одного процесса могут быть описаны действия, относящиеся к разным логическим блокам (возможно, это не самый хороший стиль кодирования, но он допускается HDL-языками). 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.

III. Основные понятия

Пусть V — множество переменных. Функция, которая каждой переменной ставит в соответствие значение соответствующего типа, называется *означиванием*. Пусть Dom_V. — множество всех означиваний на множестве V. *Охранным условием* называется булева функция, определенная на множестве означиваний ($Dom_V \to \{true, false\}$); *действием* — преобразование означиваний ($Dom_V \to Dom_V$). Пара $\gamma \to \delta$, где γ — охранное условие, а δ — действие, называется *охраняемым действием* (*Guarded Action*, *GA*). В дальнейшем

будем считать, что помимо семантики охранных условий и действий известна их синтаксическая структура (что позволяет совершать над ними символические манипуляции).

Расширенным конечным автоматом называется тройка $\langle S, V, T\rangle$, где S — конечное множество состояний; $V = I \cup O \cup R$ — конечное множество переменных, состоящее из *входных сигналов* (I), *выходных сигналов* (O) и *внутренних регистров* (R); T — конечное множество *переходов*: каждый переход $t \in T$ — это кортеж $(s_t, \gamma_t \to \delta_t, s'_t)$, где s_t и s'_t — соответственно *начальное* и *конечное состояние* перехода t, а γ_t и δ_t — соответственно охранное условие и действие перехода t. Означивание $\nu \in Dom_V$ называется *контекстом* автомата, а пара $(s, \nu) \in S \times Dom_V$ — его *конфигурацией*. Переход t называется *разрешенным* в конфигурации (s, ν), если $s_t = s$ и $\gamma_t(\nu) = true$.

Для заданных *часов* C и *начальной конфигурации* (s_0, ν_0) расширенный конечный автомат функционирует следующим образом. Вначале выполняется сброс — устанавливается начальная конфигурация автомата: $(s, \nu) \leftarrow (s_0, \nu_0)$. На каждом такте (тике часов C) определяется множество разрешенных в текущей конфигурации переходов: $E \leftarrow \{t \in T \mid s_t = s \wedge \gamma_t(\nu) = true\}$. Если множество E не пусто, срабатывает переход $t \in E$, выбранный недетерминированным образом. При выполнении перехода конфигурация обновляется соответствующим образом: $(s, \nu) \leftarrow (s_t, \delta_t(\nu))$.

IV. Извлечение EFSM-модели

Предлагаемый метод извлечения EFSM-модели (точнее, системы EFSM-моделей) из исходного кода HDL-описания состоит из следующих шагов:

1) синтаксический анализ HDL-описания и построение *дерева абстрактного синтаксиса*;

2) обход дерева абстрактного синтаксиса и построение *внутреннего представления*:

   a. идентификация *синхросигналов*;

   b. выявление *неявных переменных состояния*;

3) трансформация внутреннего представления в систему охраняемых действий;

4) анализ зависимостей между охраняемыми действиями и определение *переменных состояния*;

5) анализ условий на переменные состояния и построение *пространства состояний* EFSM-модели;

6) построение *отношения переходов* EFSM-модели.

Подраздел A данного раздела описывает *предварительную обработку* HDL-описания (шаги 1, 2 и 3). Подраздел B содержит описание *основных действий* по построению EFSM-модели (шаги 4, 5 и 6).

A. Построение системы охраняемых действий

Результатом предварительной обработки является система охраняемых действий, с каждым из которых связаны часы $(\{\langle C^{(i)}, \gamma^{(i)} \to \delta^{(i)}\rangle\})$ — такие действия называются *синхронизированными* (*clocked guarded action*) [14]. Под *часами* ($C^{(i)}$) в данной работе понимается множество *элементарных входных событий*, где каждое событие — это однобитный сигнал (синхросигнал) и тип его регистрации (передний или задний фронт). Тик часов происходит при возникновении хотя бы одного события из заданного множества.

Способы реализации **шага 1** широко известны, поэтому сразу перейдем к **шагу 2**. Одной из его целей является идентификация синхросигналов. Для решения этой задачи предлагается следующая эвристика. Считается, что переменная v является *синхросигналом*, если выполнены следующие условия:

1) v является входным однобитным сигналом;

2) v присутствует в списке чувствительности хотя бы одного из процессов (или в операторе *wait*);

3) v не используется в присваиваниях (ни в левых, ни в правых частях).

На шаге 2 также определяются неявные переменные состояния и добавляются во внутреннее представление. Под *неявными переменными состояния* понимаются регистры, явно не присутствующие в коде, но необходимые для корректного представления автомата, специфицированного в HDL-описании (обычно такие переменные искусственно вводятся инструментами логического синтеза [13]). Цель выявления и добавления неявных переменных состояния состоит в декомпозиции сложных, многотактных процессов на однотактные микрооперации. С каждым процессом p связана неявная переменная состояния (обозначим ее r_p), а с каждой операцией w типа *wait* внутри процесса p (включая операцию активации процесса) — определенное значение этого регистра (обозначим его v_w). В графе потока управления процесса p анализируются пути между парами операций *wait*, при этом проводится ряд преобразований. Процесс p удаляется из внутреннего представления; вместо него для каждого пути (точнее, ациклического подграфа с одним истоком и одним стоком) π (пусть это будет путь между w_i и w_j) строится новый процесс p_π. Условие активации p_π совпадает с условием операции w_i, а тело имеет следующий вид: **if** $r_p = v_{wi}$ **then** π; $r_p := v_{wj}$ **end if**.

На **шаге 3** для каждого элементарного процесса (микрооперации) p, построенного на шаге 2, строится множество синхронизированных охраняемых действий $\{\langle C_p^{(i)}, \gamma_p^{(i)} \to \delta_p^{(i)}\rangle\}_{i=1,n}$ таким образом, что:

1) $C_p^{(i)}$ содержит все синхроимпульсы, задействованные в процессе p;

2) $\gamma_p^{(i)}$ определяет условие $i^{\text{й}}$ ветви условного оператора верхнего уровня (если такого оператора нет, то $n = 1$ и $\gamma_p^{(1)} \equiv true$);

3) $\delta_p^{(i)}$ содержит все действия $i^{\text{й}}$ ветви (или все тело процесса p, если $n = 1$).

Отметим, что после выполнения шага 2 возможны ситуации, когда некоторые охраняемые действия содержат вложенные условные операторы. Для упрощения последующего анализа эти операторы «поднима-

ются» на уровень охранных условий (с учетом зависимостей от предшествовавших им операторов). Для этого используются техники символического выполнения, включая классический метод *обратных подстановок* [15], позволяющий вычислить *слабейшее предусловие*. Это приводит к дополнительному расщеплению охраняемых действий — каждому пути в графе потока управления процесса соответствует свое охраняемое действие (циклы с переменным числом итераций не допускаются).

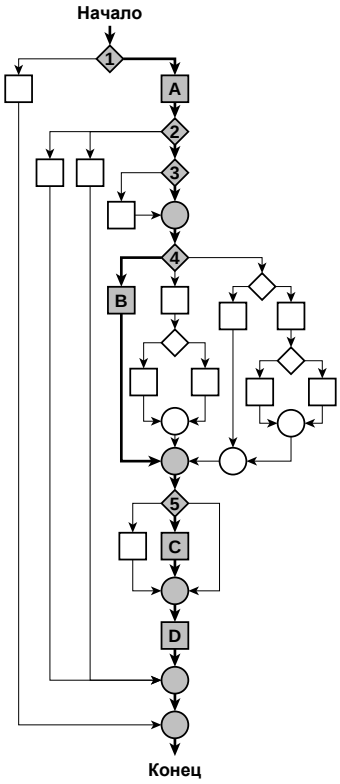

Рис. 1. Пример графа потока управления

Для того чтобы проиллюстрировать построение системы охраняемых действий, рассмотрим HDL-описание b04 из пакета тестов ITC'99 [16]. Это арифметико-логическое устройство, позволяющее вычислять сумму, среднее арифметическое, минимум и максимум набора целых чисел. На рис. 1 изображен граф потока управления описания b04. Квадратные вершины соответствуют *базовым блокам*, ромбические — *ветвлениям*, круглые — *соединениям*. У базовых блоков одна входящая дуга и одна выходящая; у ветвлений одна входящая дуга и не менее двух выходящих; у соединений не менее двух входящих и одна выходящая. Каждая дуга, выходящая из ветвления, помечена условием.

В показанном на рис. 1 графе выделен путь из начальной вершины в конечную, для которого в данном примере построено охраняемое действие. В левой колонке табл. 1 представлен код на языке VHDL, соответствующий выделенным вершинам графа (латинскими буквами нумеруются базовые блоки, цифрами — ветвления). Правая колонка табл. 1 содержит охраняемое действие, полученное в результате анализа данного пути. Отметим также, что передний фронт сигнала *CLOCK* и задний фронт сигнала *RESET* HDL-описания b04 были определены как синхроимпульсы для данного охраняемого действия.

Всего для HDL-описания b04 с помощью предложенного метода строится 27 охраняемых действий. Это число меньше общего количества путей между начальной и конечной вершинами графа потока управления, так как для ряда путей получаемые охранные условия являются невыполнимыми.

Таблица 1

Построение охраняемого действия

	HDL-описание	**Охраняемое действие**
1	**not** (RESET = '1') **and** (CLOCK'**event and** CLOCK = '1')	**Охранное условие**: (stato == sC) **and** (DATA_IN < RMIN) **and not** (DATA_IN > RMAX) **and not** (RESTART = '1') **and not** (ENABLE = '1') **Действие**: RES := RESTART; ENA := ENABLE; AVE := AVERAGE; DATA_OUT <= RLAST; RMIN := DATA_IN; REG4 := REG3; REG3 := REG2; REG2 := REG1; REG1 := DATA_IN; stato := sC; **Часы**: CLOCK (передний фронт), RESET (задний фронт)
A	RES := RESTART; ENA := ENABLE; AVE := AVERAGE;	
2	stato <= sC	
3	**not** (ENA = '1')	
4	**not** (RES = '1') **and not** (ENA = '1')	
B	DATA_OUT <= RLAST;	
5	**not** (DATA_IN > RMAX) **and** (DATA_IN < RMIN)	
C	RMIN := DATA_IN;	
D	REG4 := REG3; REG3 := REG2; REG2 := REG1; REG1 := DATA_IN; stato := sC;	

B. Построение EFSM-модели

На **шаге 4** осуществляется анализ зависимостей между охраняемыми действиями. Пусть x и y — охраняемые действия, а v — переменная. Говорят, что переменная v *определяется* в охраняемом действии x (и обозначают это как $v \in Def_x$), если действие x содержит присваивание переменной v. Говорят, что переменная v *используется* в охраняемом действии y (и обозначают это как $v \in Use_y$), если v присутствует в охранном условии y или в правой части некоторого присваивания его действия. Говорят, что охраняемое действие y *зависит* от охраняемого действия x, если имеет место неравенство $Def_x \cap Use_y \neq \varnothing$. В зависимости от того, как именно используется переменная, в охранном условии или в действии, различают *зависимости по управлению* и *по данным*. Допускаются зависимости охраняемого действия от самого себя. *Графом зависимостей* называется ориентированный граф, вершинами которого являются охраняемые действия, а дугами — зависимости.

Построенный для HDL-описания b04 граф зависимостей обладает следующими особенностями. Из 27

его вершин 26 образуют компоненту сильной связности. Происходит это потому, что соответствующие пути в графе потока управления проходят через ветвление 2 (см. табл. 1), в котором используется значение внутренней переменной *stato* (это ветвление образовано оператором *case*), и через базовый блок *D*, где эта переменная определяется. Существует лишь один путь, не проходящий ни через одну из указанных вершин — соответствующее ему охраняемое действие описывает сброс состояния.

На основе анализа зависимостей между охраняемыми действиями осуществляется идентификация переменных состояния. Под *переменными состояния* (на этом шаге все переменные являются явными) понимаются регистры, используемые для организации потоков управления процессов. Для определения таких переменных используется следующая эвристика. Переменная *v* является *переменной состояния*, если выполнены следующие условия:

1) *v* не является входным сигналом;

2) существует охраняемое действие, которое через переменную *v* зависит по управлению от самого себя;

3) все охраняемые действия, которые используют *v*, синхронизируются одинаковыми часами.

Дополнительно можно потребовать, чтобы условия на переменную *v* имели определенный вид, например, чтобы они были эквивалентны ограничению $[v] \in X$, где X — небольшое множество констант.

На **шаге 5** строится пространство состояний EFSM-модели. В качестве предварительного шага выполняется факторизация множества переменных состояния по отношению зависимости. Переменные *u* и *v* называются *зависимыми*, если найдется пара охраняемых действий, *x* и *y* (возможно совпадающих) таких, что $u \in Use_x \cup Def_x$ и $v \in Use_y \cup Def_y$ и существует путь в графе зависимостей между *x* и *y*. Каждый класс эквивалентности *R* на множестве переменных состояния обрабатывается отдельно и соответствует самостоятельной EFSM-модели. В качестве состояний рассматриваются ограничения (условия) на переменные состояния. Их построение осуществляется следующим образом:

1) для каждого охраняемого действия *x*, использующего или определяющего переменные *R*, путем символического выполнения строится *сильнейшее постусловие* (обозначим его символом ψ_x);

2) все охранные условия и постусловия, использующие переменные *R*, объединяются во множество $\Phi(R)$;

3) осуществляется *ортогонализация* условий из $\Phi(R)$: путем уточнения условий они делаются попарно несовместными (обозначим результат символом $\Phi^*(R)$);

4) каждому ограничению $\varphi_s \in \Phi^*(R)$ сопоставляется состояние *s*.

Завершающий **шаг 6** заключается в построении отношения переходов EFSM-модели:

1) для каждого охраняемого действия *x*, если γ_x совместим с ограничением φ_s, создается протопереход $t = (s, \gamma_x \to \delta_x, -)$, чье конечное состояние еще не определено;

2) если ψ_x в точности совпадает с некоторым ограничением $\varphi_{s'}$, тогда $(s, \gamma_x \to \delta_x, s')$ добавляется во множество переходов;

3) иначе для каждого s', такого что ψ_x и $\varphi_{s'}$ совместимы, по постусловию $\varphi_{s'}$ строится слабейшее предусловие действия δ_x (обозначим его символом $\gamma^*_{xs'}$), переход $(s, (\gamma_x \wedge \gamma^*_{xs'}) \to \delta_x, s')$ добавляется во множество переходов;

4) переходы с одинаковыми начальными состояниями и охранными условиями объединяются.

На рис. 2 схематично изображена EFSM-модель для HDL-описания b04, построенная с помощью описанного метода. В качестве переменной состояния была выделена переменная *stato*, а в качестве состояний — ограничения вида *stato* == *c*, где *c* — константа. Состояния EFSM-модели помечены значениями переменной *stato*, а переходы — ограничениями на переменные состояния, входящими в состав охранных условий. Два перехода ($sB \to sA$ и $sC \to sA$) соответствуют сбросу состояния. Еще в двух переходах происходит изменение состояния ($sA \to sB$ и $sB \to sC$). Петля в вершине *sC* соответствует 23 переходам, реализующим вычисления.

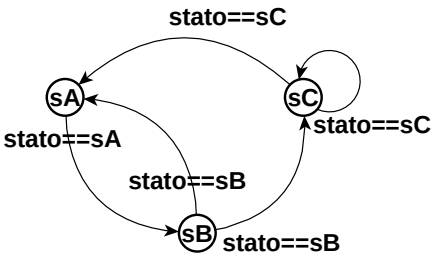

Рис. 2. Пример EFSM-модели

V. Результаты экспериментов

Описанный метод извлечения EFSM-моделей из исходного кода HDL-описаний был реализован программно в прототипе инструмента HDL Retrascope. Разработка выполнена на языке программирования Java с использованием библиотек JUNG [17], zamiaCAD [18] и Fortress [19]. Прототип позволяет анализировать описания цифровой аппаратуры на синтезируемом подмножестве языка VHDL и строить систему EFSM-моделей (иерархическая структура проекта при этом не учитывается).

В рамках работы были проанализированы четыре проекта с открытым исходным кодом: PLASMA, DLX, HC11 и UART (всего 42 модуля, содержащих 14 000 строк кода). Для каждого модуля извлекались часы, переменные состояния и ограничения на переменные, соответствующие состояниям EFSM-модели. Часы были автоматически выделены для 70% модулей (ос-

тавшиеся модули содержали исключительно комбинационную логику). Сигналы, фигурирующие в выделенных часах, можно классифицировать как синхросигналы, сигналы сброса, а также сигналы прерываний. Переменные состояния были извлечены для 25% модулей. В 30% случаев их имена содержали подстроку «state» (стоит отметить, что все переменные с подстрокой «state» в имени были определены инструментом как переменные состояния). В других случаях результаты также можно считать адекватными (например, в качестве переменных состояния были извлечены регистры, хранящие наполненность очередей FIFO).

VI. Заключение

В настоящее время EFSM-модели активно используются в области формальной и имитационной верификации аппаратуры. Предложен ряд подходов к генерации тестов и формальной проверке свойств на основе анализа EFSM-моделей [5]-[10], [11], [13]. В работе описан новый метод извлечения EFSM-моделей из исходного кода HDL-описаний. Отличительной особенностью предложенного метода является автоматическое выделение внутренних переменных, описывающих состояние, а также извлечение состояний и переходов EFSM-модели на основе символического анализа условий на переменные состояния (в работе [6] информация о переменных состояния используется только на поздних фазах алгоритма). Эксперименты доказали применимость и эффективность метода для аппаратуры средней сложности. В ближайшем будущем мы планируем провести детальное сравнение нашего подхода с аналогичными методами.

В качестве другого направления для дальнейших исследований рассматривается анализ *зависаний* (*deadlock*) и *конфликтов доступа к данным*, которые могут возникать при наличии нескольких параллельно работающих EFSM-моделей над общим множеством переменных. Например, если автомат меняет значение некоторой переменной во время одного перехода, а во время другого перехода считывает ее значение, то не должно быть других автоматов, которые могли бы изменить значение этой переменной между операциями записи и чтения. Ещё одним многообещающим направлением является *конколическое тестирование* (*concolic* [*conc*rete & symb*olic*] *testing*), основанное на комбинировании статических и динамических техник верификации [20]. Извлеченную EFSM-модель можно использовать для получения сведений о порядке подачи воздействий и приеме реакций HDL-описания (т.е. о *протоколе взаимодействия* с устройством). Эти сведения могут быть использованы как для формальной верификации, так и для генерации шаблонов тестовых систем (в том числе систем, построенных с помощью методологии UVM на языке SystemVerilog [21]).

Предложенный метод извлечения EFSM-моделей может быть применен для анализа описаний аппаратуры на языке Verilog. Для этого необходимы синтаксический анализатор и средство построения внутреннего представления, аналогичное существующему для языка VHDL. Работы по их созданию уже ведутся.

Литература

[1] Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Springer, 2003. 478 p.

[2] Лазарев В.Г., Пийль Е.И. Синтез управляющих автоматов. М.: Энергоатомиздат, 1989. 328 с.

[3] Botros N.M. HDL Programming Fundamentals: VHDL and Verilog. Charles River Media, 2005. 506 p.

[4] Баранов С.И., Майоров С.А., Сахаров Ю.П., Селютин В.А. Автоматизация проектирования цифровых устройств. Л.: Судостроение, 1979. 264 с.

[5] Duale A.Y., Uyar M.U. A Method Enabling Feasible Conformance Functional Test Sequence Generation for EFSM Models // IEEE Transactions on Computers. 2004. № 53(5). P. 614-627.

[6] 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.

[7] Fummi F., Marconcini C., Pravadelli G. Functional Verification based on the EFSM Model // IEEE International High Level Design Validation and Test Workshop. 2004. P. 69-74.

[8] Guglielmo G., Fummi F., Pravadelli G., Soffia S., Roveri M. Semi-formal Functional Verification by EFSM Traversing via NuSMV // IEEE International High Level Design Validation and Test Workshop. 2010. P. 58-65.

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

[10] Bombieri N., Fummi F., Guarnieri V. FAST: An RTL Fault Simulation Framework based on RTL-to-TLM Abstraction // Journal of Electronic Testing. 2012. № 28(4). P. 495-510.

[11] Cheng K.-T., Krishnakumar A.S. Automatic Generation of Functional Vectors Using The Extended Finite State Machine Model // ACM Transactions on Design Automation of Electronic Systems. 1996. № 1(1). P. 57-79.

[12] Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай // Программирование. 2003. № 29(5). С. 11-30.

[13] Hierons R.M.,Kim T.-H., Ural H. Expanding an Extended Finite State Machine to Aid Testability // Computer Software and Applications Conference. 2002. P. 334-339.

[14] Brandt J., Gemünde M., Schneider K., Shukla S., Talpin J.-P. Integrating System Descriptions by Clocked Guarded Actions // Forum on Design Languages. 2011. P. 1-8.

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

[16] URL: http://www.cad.polito.it/downloads/tools/itc99.html (дата обращения: 20.04.2014).

[17] URL: http://jung.sourceforge.net (дата обращения: 20.04.2014).

[18] URL: http://zamiacad.sourceforge.net (дата обращения: 20.04.2014).

[19] URL: http://forge.ispras.ru/projects/solver-api (дата обращения: 20.04.2014).

[20] Sen K. Concolic Testing // IEEE/ACM International Conference on Automated Software Engineering. 2007. P. 571-572.

[21] URL: http://www.accellera.org/downloads/standards/uvm (дата обращения: 20.04.2014).