К.ф.-м.н. А.С. Камкин (ИСП РАН)**, М.В. Петроченков** (ЗАО «МЦСТ»)

A. Kamkin, M. Petrochenkov

СИСТЕМА ПОДДЕРЖКИ ВЕРИФИКАЦИИ РЕАЛИЗАЦИЙ ПРОТОКОЛОВ КОГЕРЕНТНОСТИ С ИСПОЛЬЗОВАНИЕМ ФОРМАЛЬНЫХ МЕТОДОВ

A SYSTEM TO SUPPORT FORMAL METHODS-BASED VERIFICATION OF COHERENCE PROTOCOL IMPLEMENTATIONS

Описывается подход к разработке тестовых систем, предназначенных для верификации RTL-реализаций протоколов когерентности памяти. На основе анализа проблем, возникающих при верификации механизмов работы с памятью многоядерных микропроцессоров, предлагается архитектура тестовой системы, рассчитанной на эту специфику. Существенное внимание уделяется проблеме детерминированного воспроизведения тестов, построенных с помощью формальных методов. Рассматривается опыт применения предложенного подхода для микропроцессора «Эльбрус-2S».

The paper describes an approach to develop test systems aimed at verification of RTL implementations of memory coherence protocols. Tasks for verification of memory management mechanisms of multi-core microprocessors are analyzed; basing on the analysis, a test system architecture is suggested. The emphasis is put on such a task as deterministic replay of tests constructed with the help of formal methods. An experience of applying the suggested approach to the Elbrus-2S microprocessor is considered.

Ключевые слова: консистентность памяти, протоколы когерентности, верификация, тестирование, проверка моделей, детерминированное воспроизведение.

Keywords: memory consistency, coherence protocols, verification, testing, model checking, deterministic replay.

Введение

Одна из основных проблем, возникающих при создании многоядерных микропроцессоров, типична для всех видов распределенных систем с общей памятью – это обеспечение когерентности памяти. Каждый узел системы (ядро) имеет в своем составе локальную память (кэш), из-за чего в системе могут существовать несколько копий одних данных – одна копия в основной памяти и несколько копий в локальной памяти узлов. При изменении копии данных в одном из узлов другие совладельцы данных должны согласованным образом изменить свои копии (или удалить их). Взаимодействие между узлами системы осуществляется согласно некоторому протоколу когерентности, за реализацию которого отвечают соответствующие механизмы подсистемы управления памятью.

Разработка механизмов обеспечения когерентности памяти осуществляется в два этапа: первый – проектирование протокола когерентности, второй – реализация протокола в аппаратуре. Ввиду сложности современных протоколов на обоих этапах возможны ошибки (пример – erratum #298 в AMD Phenom [2]), для обнаружения которых применяются методы верификации спецификаций протоколов и методы верификации реализаций протоколов соответственно. Первой теме посвящено много исследований, сконцентрированных, в основном, вокруг формальных методов [3]. Работы по второй теме носят технический характер и базируются, как правило, на случайной генерации тестов [4]. Цель статьи – предложить подход к верификации реализаций протоколов когерентности, позволяющий использовать имеющиеся наработки в области формальных методов.

1. Краткий обзор работ

Формальные методы неприменимы напрямую к реализациям протоколов когерентности (RTL-моделям подсистем управления памятью), т.к. пространство состояний слишком велико, а логика протокола распределена по множеству устройств микропроцессора. Для верификации подобных систем обычно используют динамическую верификацию –

тестирование, эффективность которого определяется способом генерации тестовых воздействий; проверкой корректности и оценкой адекватности тестирования.

Самым распространенным способом построения тестовых воздействий (цепочек из инструкций обращения к памяти) является случайная генерация [4, 5]. Признаком ошибки является считывание из памяти некорректных данных (из-за интерливинга запросов возможны несколько допустимых значений). Критерием завершения верификации является отсутствие проявлений ошибок в течение длительного времени. Недостатком случайной генерации является несистематичность – некоторые маловероятные состояния реализации могут остаться неохваченными.

Использование формальных моделей позволяет устранить этот недостаток. На их основе можно генерировать модельные трассы, которые исчерпывающе покрывают ситуации, возможные в работе протокола [3, 6]. Трассы применяются и для управления тестированием (воспроизведения событий реализации в требуемом порядке), и для проверки корректности поведения (сопоставления реакций реализации с событиями модели) [7, 8]. Полнота верификации оценивается покрытием пространства состояний модели.

Основная проблема этого подхода связана с тем, что тестирование реализации требует значительных вычислительных ресурсов. Существует несколько общих способов сокращения размера трасс: редукция частичных порядков при обходе состояний модели; ограничение глубины поиска в графе состояний; факторизация пространства состояний путем абстракции модели. Кроме того, специально для протоколов когерентности предложены эвристические методы построения тестов, эффективно обнаруживающих ошибки – они позволяют строить более короткие (чем при традиционном поиске в глубину [9]) тестовые последовательности, которые полностью покрывают пространство состояний протокола.

Другой подход к уменьшению вычислительных затрат описан в [5]. Он основан на

гибридных кластерах, в которых вместо RTL-реализаций некоторых узлов используются упрощенные эталонные модели, что позволяет более точно диагностировать ошибки в реализации.

Универсальным способом ускорения верификации является распараллеливание, использующее возможности современных архитектур. Так, в [10] отмечается пятикратное ускорение верификации RTL-реализации протокола MOESI в четырехъядерном микропроцессоре на 16-ядерной инструментальной машине.

2. Постановка задачи

В настоящее время тестирование является безальтернативным методом верификации реализаций протоколов когерентности [4, 5]. В то же время, для проверки спецификаций протоколов все чаще используются формальные методы, основанные на статическом анализе обобщенных моделей [3, 6]. На базе формальных методов в результате исследования пространства состояний модели могут быть построены модельные трассы событий протокола, которые после обработки будут использованы при построении тестов реализации протокола. В простейшем случае такая обработка сводится к удалению из трасс внутренних событий протокола (снуп-запросов) и преобразованию оставшихся внешних событий (первичных запросов) в формат, применяемый в тестовой системе (как правило, тест – это набор программ, предназначенных для выполнения на отдельных узлах). Недостатками подобного проецирования является существенная вероятность того, что будет сформировано большое число одинаковых тестов, а их выполнение охватит лишь небольшую часть возможных трасс (по сути, такие тесты мало чем отличаются от тестов, сгенерированных случайно).

Полноценное использование формальных методов возможно только в том случае, когда задействуется информация о порядке внутренних событий протокола [7]. Возможны

два способа использования такой информации: управление (воспроизведение событий в требуемом порядке [7]) и мониторинг (проверка корректности поведения и отслеживание тестового покрытия [8]). Следует обратить внимание на два обстоятельства: в абстрактной модели возможны трассы, которые неосуществимы в конкретной реализации (лишнее поведение); трассы имеют вид последовательностей событий, однако порядок некоторых событий несущественен (переопределенность поведения).

Целью работы является поддержка тестирования реализаций протоколов когерентности на основе модельных трасс. К средствам поддержки предъявляются следующие требования:

– они должны воспроизводить трассы на реализации протокола, проверять корректность поведения и измерять уровень достигнутого тестового покрытия;

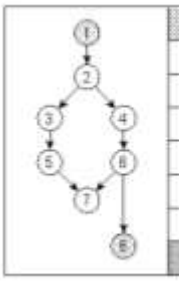
– в них должны присутствовать механизмы для ослабления или конкретизации порядка между событиями, действующие на основе предоставляемой информации о реализации;

– они должны быть применимы к системам, основанным на разных архитектурах и протоколах когерентности.

3. Архитектура тестовой системы

Предлагаемый способ верификации в применении к реализации протоколов когерентности памяти базируется на архитектурно-независимом формате модельной трассы событий. Каждое событие трассы имеет уникальный идентификатор (натуральное число), множество зависимостей (совокупность идентификаторов событий, предшествующих данному) и множество атрибутов (набор пар вида ключ-значение). Таким образом, трасса описывает частично упорядоченное множество атрибутированных событий (рис. 1, слева – диаграмма Хассе, справа – текстовое представление, серым цветом выделены первичные

запросы). Отметим, что представление модельных и реализационных событий в тестовой системе унифицировано, – при получении пакета от реализации система преобразует его во внутреннее представление, которое может быть сопоставлено с одним из событий трассы. Сопоставление событий осуществляется путем сравнения значений их общих атрибутов (реализационные события могут содержать дополнительные атрибуты, которые не отражены в трассе модели: номер ячейки в буфере запросов, данные кэш-строки и т.п.).



1			*cpu_dst*=3	*core*=2	*cop*=ld_wb	*addr*=0xf0	
2	(1)	*cpu_src*=3	*cpu_dst*=0		*cop*=R64	*addr*=0xf0	*label_cpu*=3
3	(2)	*cpu_src*=0	*cpu_dst*=2		*cop*=snR64	*addr*=0xf0	*label_cpu*=3
4	(2)	*cpu_src*=0	*cpu_dst*=1		*cop*=snR64	*addr*=0xf0	*label_cpu*=3
5	(3)	*cpu_src*=2	*cpu_dst*=3		*cop*=R64_DATA		*label_cpu*=3
6	(4)	*cpu_src*=2	*cpu_dst*=3		*cop*=COH_ACK		*label_cpu*=3
7	(5, 6)	*cpu_src*=3	*cpu_dst*=0		*cop*=READ_END		*label_cpu*=3
8	(6)		*cpu_dst*=3	*core*=0	*cop*=ld_wb	*addr*=0xf0	

Рис. 1

Пример трассы

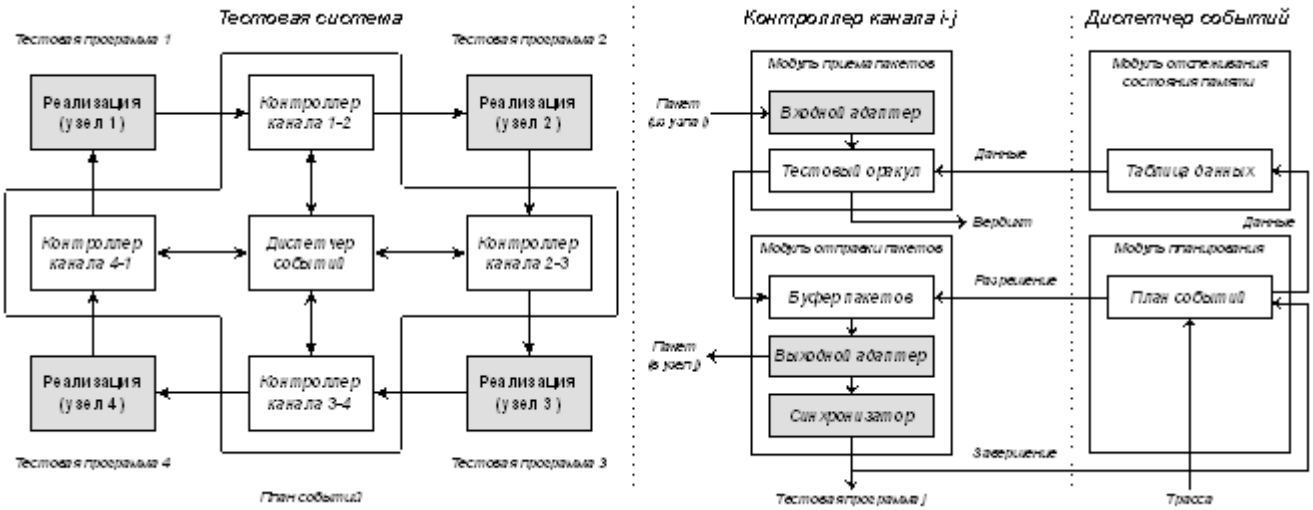
По трассе строятся набор тестовых программ (по одной для каждого узла) и план событий – внутреннее представление трассы, используемое тестовой системой для обеспечения требуемого порядка взаимодействий между узлами. Приведенный на рис. 1 фрагмент трассы соответствует следующей программе для узла 3 (программы для других узлов являются пустыми, поскольку трасса не содержит связанных с ними первичных запросов):

```
{ ldw,2 [0xf0] %dr20 } WAIT_SYNC(6) { ldw,0 [0xf0] %dr14 }
```

Следует обратить внимание на макрос `WAIT_SYNC`, используемый для приостановки процесса выполнения программы (параметрами макроса являются идентификаторы событий, которых следует дождаться). Регистры, используемые в командах программы, выбираются случайным образом.

Тестовая система состоит из следующих модулей: конфигурация, диспетчер событий и множество контроллеров каналов (по одному на каждый канал связи между узлами). Конфигурация задает тип каждого узла (RTL-реализация или эталонная модель [5]) и то-

пологию связей между узлами. Диспетчер отвечает за воспроизведение событий протокола когерентности в порядке, заданном в плане. Контроллеры каналов перехватывают передаваемые пакеты и по указанию диспетчера меняют их порядок (приостанавливают пе-



ресылку пакетов, для которых не удовлетворены зависимости, определенные в плане). На рис. 2 показана организация тестовой системы для конфигурации из четырех узлов, соединенных в однонаправленное кольцо (серым цветом выделены узлы реализации и архитектурно-зависимые компоненты тестовой системы).

Рис. 2

Архитектура тестовой системы, организация контроллера канала и диспетчера событий

Контроллер канала делится на модуль приема и модуль отправки пакетов. Первый из них отвечает за перехват пакетов реализации и их преобразование к формату, используемому в тестовой системе. Оно осуществляется с помощью входного адаптера, устройство которого определяется типом узла реализации: для RTL-моделей используются средства, предоставляемые HDL-симуляторами (интерфейсы типа VPI и DPI); для эталонных моделей преобразование, как правило, тривиально. Еще одна функция, выполняемая модулем, – проверка передаваемых в пакетах данных (результат чтения должен совпадать с данными последней записи). Проверка осуществляется тестовым оракулом при содействии диспетчера, отслеживающего порядок выполнения операций и хранящего актуальные данные

для каждого адреса.

Модуль отправки пакетов имеет буфер, в котором пакеты, полученные от реализации, ожидают своей очереди на отправку (решение об отправке пакета принимает диспетчер). Для преобразования пакета из внутреннего представления тестовой системы в реализационное используется выходной адаптер, который, помимо прочего, информирует об отправке пакета синхронизатор. К функциям последнего относятся передача информации о каждом возникающем событии к тестовой программе и уведомление диспетчера о завершении передачи пакета.

Взаимодействие синхронизатора с тестовой программой, выполняемой на RTL-реализации, может осуществляться через выделенный регистр микропроцессора: каждый раз, когда осуществляется отправка пакета, синхронизатор записывает в регистр соответствующий идентификатор; в тестовой программе (макрос `WAIT_SYNC`) в цикле сравнивается значение регистра с числом, указанным в качестве параметра макроса, и при их совпадении осуществляется выход из цикла (в случае зависимости от нескольких событий схема проверок немного усложняется).

Диспетчер событий, являющийся ядром тестовой системы, состоит из модуля отслеживания состояния памяти и модуля планирования. Первый модуль содержит таблицу данных, моделирующую память микропроцессора (или многопроцессорного комплекса). Для каждого адреса, использующегося при верификации, таблица содержит последние записанные по нему данные. Модуль взаимодействует с тестовыми оракулами контроллеров каналов, передавая по их запросу данные, содержащиеся в таблице (эталонные значения).

Модуль планирования отвечает за воспроизведение событий протокола когерентности в порядке, указанном в трассе. Для этого используется план событий (расписание) – частично упорядоченное множество, которое вначале совпадает с исходной трассой, но из

которого в процессе верификации удаляются осуществившиеся события. Модуль планирования отвечает на запросы со стороны буферов пакетов. Для пакета, находящегося в буфере, дается разрешение на отправку, когда он сопоставляется с одним из минимальных по отношению порядка событий плана. После завершения отправки пакета (по сигналу от синхронизатора) сопоставленное ему событие удаляется из плана. Если удаляемое событие соответствует операции записи, его данные заносятся в таблицу данных. План событий представляется в виде ориентированного ациклического графа, дуги которого задают отношение непосредственного следования событий. Разрешенным событиям (минимальным элементам частичного порядка) соответствуют истоки графа – вершины, у которых нет входящих дуг.

4. Опыт практического применения

Для верификации протокола когерентности памяти микропроцессора «Эльбрус-2S» (вариант протокола MOSI) была разработана его модель на языке Promela [6]. При этом возникла задача использования трасс, полученных при формальной верификации модели с помощью инструмента Spin, для проверки реализации протокола в RTL-модели микропроцессора. С этой целью потребовалось модифицировать разработанный ранее гибридный кластер, основанный на эталонной модели E2S_MU [5]. Полученная тестовая система является частным случаем систем, описанных в этой статье, т.к.:

– используются только линейно упорядоченные трассы;

– критерием ошибки является истечение времени ожидания события протокола;

– синхронизация осуществляется посредством регистров микропроцессора.

Воспроизведение трасс, полученных при верификации модели протокола, позволило найти ошибку в эталонной модели E2S_MU – некорректное преобразование первичного запроса типа *I*64 (*Invalidate*) в запрос типа *RI*64 (*Read-Invalidate*). Данная ошибка не при-

водит к нарушению когерентности памяти, поэтому не была обнаружена ранее с помощью других методов верификации, основанных на проверке данных, однако она вызывает появление лишнего трафика – дополнительных пакетов с данными из кэш-памяти, что снижает производительность системы.

Учитывая положительные результаты, были созданы средства поддержки разработки тестовых систем подобного типа. Разработка осуществлялась в два этапа: на первом – были расширены возможности имеющейся тестовой системы, в частности, реализованы механизмы работы с частично упорядоченными трассами; на втором – из тестовой системы была выделена архитектурно-независимая часть, допускающая повторное использование в виде библиотеки. Ослабление порядка между событиями позволило сократить тестовый набор (некоторые трассы стали эквивалентными) и уменьшить время выполнения тестов (более слабому порядку событий соответствует больший уровень параллелизма). В настоящее время созданные средства используются для разработки тестовой системы для микропроцессора «Эльбрус-4С+».

Заключение

Формальная верификация протокола когерентности позволяет обнаружить концептуальные просчеты проектирования, но не гарантирует отсутствие ошибок в аппаратуре. Для проверки корректности реализации определенных механизмов протокола используются универсальные методы и инструменты, основанные, прежде всего, на случайной генерации тестов. В промышленной практике такие средства выявляют значительное число ошибок, однако с увеличением числа ядер и усложнением микроархитектуры будет возрастать потребность в более систематических подходах к верификации.

Представляется целесообразным использовать модели протоколов, применяемые в контексте формальных методов, для верификации RTL-реализаций. При этом возникают

задачи генерация тестов по модели и детерминированного (сохраняющего порядок событий) воспроизведения построенных тестов на реализации. Первая задача решается методами тестирования на основе моделей. Вторая задача может быть решена с помощью средств, описанных в этой статье. Мы надеемся, что предложенный подход поспособствует повышению надежности многоядерных микропроцессоров.

Литература

1. А.К. Ким, В.И. Перекатов, С.Г. Ермаков. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». – СПб.: Питер, 2013, 272 с.

2. Revision Guide for AMD Family 10h Processors. Revision 3.84, August 2011.

3. F. Pong, M. Dubois. Verification Techniques for Cache Coherence Protocols. ACM Computing Surveys (CSUR), 29(1), 1997. P. 82-126.

4. D.A. Wood, G.A. Gibson, R.H. Katz. Verifying a Multiprocessor Cache Controller Using Random Test Generation. IEEE Design & Test of Computers, 7(4), 1990. P. 13-25.

5. В.Н. Куцевол, А.Н. Мешков, М.В. Петроченков. Методология верификации протокола когерентности микропроцессора «Эльбрус-2S» – «Вопросы радиоэлектроники», сер. ЭВТ, 2013, вып. 3, с. 107-117.

6. В.С. Буренков. Анализ применимости инструмента Spin к верификации протоколов когерентности памяти – «Вопросы радиоэлектроники», сер. ЭВТ, 2013, вып. 3, с. 126-134.

7. S. Tasiran, Y. Yu, B. Batson. Using a Formal Specification and a Model Checker to Monitor and Direct Simulation. Design Automation Conference (DAC), 2003. P. 356-361.

8. В.П. Иванников, А.С. Камкин, М.М. Чупилко. Проверка корректности поведения HDL-моделей цифровой аппаратуры на основе динамического сопоставления трасс. – Материалы международной научно-практической конференции «Инструменты и методы

анализа программ» (TMPA), 2013. С. 94-105.

9. Y. Chen, D. Abts, D.J. Lilja. State Pruning for Test Vector Generation for a Multiprocessor Cache Coherence Protocol. Workshop on Rapid System Prototyping (RSP), 2004. P. 74-77.

10. Q. Xiong, J. Yi, T. Song, Z. Xie, D. Tong. VFCC: A Verification Framework of Cache Coherence using Parallel Simulation. Asia and South Pacific Design Automation Conference (ASP-DAC), 2013. P. 705-710.