к.т.н. А.Н. Мешков, М. П. Рыжов, В. А. Шмелев (ЗАО «МЦСТ»)

A. Meshkov, M. Ryzhov, V. Shmelyov

РАЗВИТИЕ СРЕДСТВ ВЕРИФИКАЦИИ МИКРОПРОЦЕССОРА «ЭЛЬБРУС-2S»

THE DEVELOPEMENT OF THE VERIFICATION TOOLS OF THE ELBRUS-2S MICROPROCESSOR

Рассматривается опыт верификации микропроцессора «Эльбрус-2S». Описываются различные подходы к его верификации, разработанные в рамках проекта. Анализируются и сравниваются примененные средства автономной верификации. Приводится описание процесса разработки программной модели и алгоритма генерации тестов с опорой на нее.

The experience of the verification of the Elbrus-2S microprocessor is considered. Different approaches having been developed in the project are described. Tools having been applied for stand-alone verification are analyzed and compared. Process of development of the reference model and an algorithm of reference model-based test generation is described.

Ключевые слова: Эльбрус-2S, функциональная верификация, тестирование на основе моделей, автономная верификация, эталонная модель.

Keywords: Elbrus-2S, functional verification, model-based testing, stand-alone verification, reference model.

Введение

Микропроцессор «Эльбрус-2S» продолжает линию развития вычислительных средств семейства «Эльбрус» как за счет перехода на передовой технологический процесс и связанное с этим усовершенствование частотных характеристик, так и путем введения новых архитектурных и микроархитектурных решений. Ряд ключевых устройств разрабатывался с нуля, значительным изменениям подверглись многие модули, была расширена система команд и реализован новый протокол когерентности. С этим связаны и новые

элементы в процессе логической верификации, которые рассматриваются в данной статье.

1. Развитие инструментария

Использование шаблонных генераторов

Проверка функционирования ряда устройств предполагает повторное использование базового кода с изменением ряда параметров. В предыдущих проектах оно обеспечивалось препроцессором языка C (cpp), возможностей которого недостаточно для создания сложных шаблонов, свойственных данному проекту. В связи с этим в работу был введен шаблонизатор Genshi [1] – библиотека, изначально предназначенная для веб-приложений, которая, тем не менее, может быть встроена в любой инструмент. Поддержка использования широких возможностей языка python, а также наличие стандартных управляющих конструкций (условных операторов, циклов, операторов выбора, включения файлов) позволили создавать шаблоны любой сложности. Данное средство было успешно использовано при разработке ряда тестов и генераторов. Однако в связи с тем, что в определенных случаях его производительность оказалась недостаточной, был создан собственный препроцессор, учитывающий особенности разработки ассемблерных тестов.

Особое внимание в новой разработке было уделено поддержке адекватных задачам типов данных, обеспечению возможности полного перебора пространства параметров шаблона в псевдослучайном порядке и обеспечению высокой производительности. Разработанный препроцессор использован при верификации ряда расширений в системе команд микропроцессора «Эльбрус-2S» [2]. Его дальнейшее развитие позволит существенно увеличить эффективность создания шаблонных тестов.

Расширение базы регрессионных тестов

К началу проекта «Эльбрус-2S» была накоплена база тестов для регрессионного тестирования, включающая следующие типы:

- тесты проверки базовой функциональности команд, предназначенные для быстро-

го обнаружения грубых ошибок и упрощения их разбора;

- тесты на краевые случаи и редкие динамические ситуации;
- псевдослучайные тесты, на которых ранее были обнаружены ошибки.

В новом проекте ряд тестов потребовал переработки в соответствии с изменениями в системе команд. Кроме того, была проведена работа по выделению групп тестов, функциональность которых может быть обеспечена кодом, сгенерированным на основе параметризованных шаблонов. Типичной функцией тестов этого вида является проверка формата конфигурационных регистров, а также корректности статического функционирования арифметико-логических операций на фиксированном наборе значений. Заменяющие тесты-шаблоны были разработаны с использованием описанных выше генераторов. Данная работа позволила упростить поддержку тестовой базы и проверку ее полноты.

Формальная верификация протокола когерентности

Для верификации протокола когерентности в соответствии с подходом Model Checking [3] на языке promela была разработана формальная модель системы поддержки когерентности, а также математически сформулированы требования корректности ее функционирования для системы SPIN. На базе доказательства корректности модели были получены последовательности воздействий, преобразованные в тесты для специализированной гетерогенной сборки, которая сочетает RTL-модель процессора и функциональные модели подсистем памяти, исполняющие псевдокод E2S_MU. В составе данной системы тесты не имитируют стандартный процесс исполнения инструкций, но детально моделируют последовательность исполнения запросов к общей памяти с учетом всех вспомогательных обращений, таких как запросы поддержки когерентности. Использование описанной системы позволило обнаружить ряд ошибок протокола когерентности, не выявленные другими средствами [3].

Исполнение полученных тестов на прототипе вычислительного комплекса, разрабатываемого при проектировании микропроцессора «Эльбрус-2S», позволило обнаружить

ошибки в реализации подсистемы памяти.

Окружение для переносимых тестов

Для верификации устройств APIC (подсистема прерываний), RDMA (межмашинное соединение) и IOMMU (система трансляции адресов операций ввода-вывода) было использовано окружение, спроектированное для разработки переносимых тестов на языке C++. Это позволило отразить в существующих тестах изменения функциональности периферийных устройств, не требуя учета изменения системы команд. Тесты на языке C++ также использовались для верификации контроллера памяти.

Кроме того, окружение позволило упростить адаптацию программ, запускаемых под операционной системой и инженерной программой начальной загрузки. В частности, было обеспечено функционирование псевдослучайных тестов, предназначенных для проверки компилятора, на прототипе вычислительного комплекса.

2. Автономная верификация

При верификации предыдущих проектов ЗАО «МЦСТ», использующей в основном системные тесты полной HDL-модели микропроцессора и ее аппаратного прототипа, периодически возникали проблемы, преодоление которых требовало существенных усилий:

- большие затраты вычислительных ресурсов и времени на моделирование полного описания микропроцессора;
- сложность реализации тестов, направленных на верификацию конкретного устройства, конкретной функциональности устройства или модуля;
- сложность оценки качества тестирования;
- сложность разбора результатов запуска тестов и локализации ошибки.

В силу этих причин в данном проекте наряду с использованием системных тестов был применен метод модульной, или автономной верификации, объектом которой является HDL-описание отдельного устройства (модуля) или подсистемы микропроцессора.

Тестовая система этого типа в общем виде состоит из следующих компонентов (рис. 1):

• тестовая последовательность воздействий, которые необходимо применить к верифицируемому устройству. Они могут быть заданы в виде битового вектора, подаваемого на входы модуля, либо в виде транзакций более высокого уровня. Тестовая последовательность может быть заранее заданной (направленный тест) или генерироваться в ходе моделирования (случайный тест);

• генератор тестовых воздействий, подающий входные сигналы на верифицируемый модуль в соответствии с тестовой последовательностью; если она представляет собой набор высокоуровневых транзакций, генератор преобразовывает их в битовые;

• монитор реакций, предназначенный для сбора данных с выходов модуля реакций и передачи их в модуль проверки;

• модуль проверки (чекер), контролирующий правильность функционирования устройства. С этой целью могут быть использованы эталонный набор реакций на входные воздействия, потактовая или функциональная модель устройства на языке высокого уровня, а также некий набор правил функционирования модуля, не выделенный в виде модели; зачастую в реализации чекера применяются несколько таких методов.

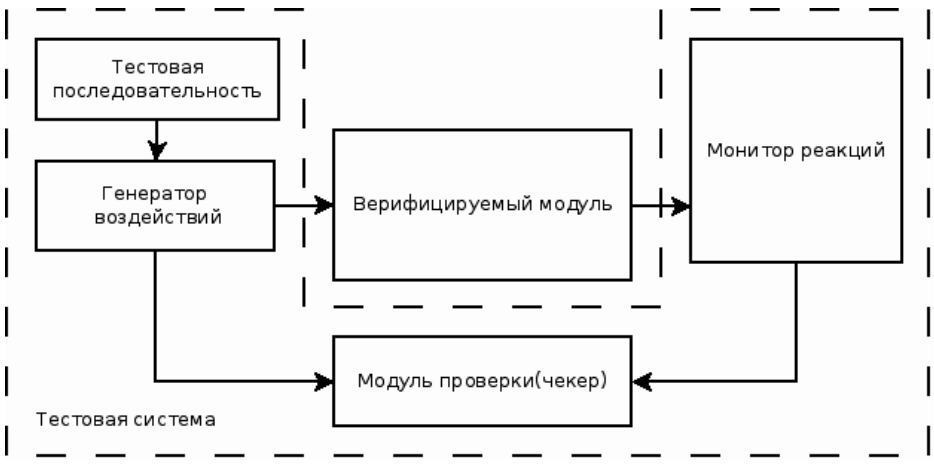

Рис. 1

Тестовая система для автономной верификации

Сводная информация об инструментах, использованных в проекте для построения тестовых систем, представлена в табл. 1. Помимо двух стандартных продуктов – широкого применяемого OVM и менее известного C++NESKHW – в ней представлен инструмент собственной разработки Alone-env.

Таблица 1

Сравнение инструментов автономной верификации, используемых в проекте «Эльбрус-2S»

	Alone-env	**C++TESKHW**	**OVM**
Язык тестовой системы	C++	C++	SystemVerilog
Язык теста	C++	C++	SystemVerilog
Тип тестовых воздействий	Направленные	Направленные, случайные, с применением обходчика конечного автомата	Направленные, случайные с ограничениями (constrained random)
Средства сбора функционального покрытия	Отсутствуют	Встроенные (C++)	Встроенные (SystemVerilog)
Средства автоматизации разработки	Входят в комплект инструмента	Входят в комплект инструмента	Собственной разработки
Сложность освоения	Низкая	Высокая	Высокая
Длительность разработки первых тестов	Низкая	Высокая	Низкая
Требования к точности эталонной модели	Высокие, предпочтительна потактовая модель	Высокие, предпочтительна потактовая модель	Низкие
Возможность переиспользования компонентов тестовой системы	Отсутствует	Отсутствует	Есть возможность переиспользования
Число модулей, проверенных в ходе проекта «Эльбрус-2S»	3	5	26

Инструмент Alone-env был разработан в ЗАО «МЦСТ» в 2009 г., чтобы снизить трудоёмкость реализации автономных тестов на языке Verilog, путем создания тестовых последовательностей на языке C++. Библиотека Alone-env предоставляет разработчику теста C++ класс-обёртку над Verilog-описанием верифицируемого модуля. В состав методов класса входят функции, позволяющие считывать значения с выходов устройства и уста-

навливать значения на входы, а также метод clk(), предназначенный для передачи управления симулятору HDL при моделировании одного такта работы устройства. Передача значений к устройству и от устройства реализована через интерфейс DPI. Библиотека обеспечивает прозрачное для пользователя преобразование форматов данных, применяемых в тесте, в формат, пригодный для передачи через DPI, а также контроль значений на выходах (проверка на неопределённое состояние). Кроме библиотеки в состав инструмента входит набор скриптов, генерирующих структуру тестовой системы и файлы интерфейса с тестируемым устройством. С помощью Alone-env была проведена верификация модулей контроллера памяти, MAU и ряда других устройств. По опыту применения библиотеки в проекте «Эльбрус-2S» следует отметить, что в целом задачи, поставленные при разработке Alone-env, были решены, хотя имеется ряд проблем, решение которых позволит расширить применение этого инструмента.

Инструмент C++TESKHW, использованный для верификации модулей «Эльбрус-2S», разработан в ИСП РАН. Одной из его особенностей является наличие генератора тестовых последовательностей на основе обхода графа состояний устройства [4, 5]. Применение обходчика позволяет получать библиотечные тестовые последовательности, дающие более полное покрытие функциональности устройства тестами. Взаимодействие с верифицируемым устройством осуществляется через функции VPI, но возможна и передача значений через DPI. К достоинствам библиотеки следует отнести наличие встроенных средств отладки тестов и оценки качества тестирования (сбор функционального покрытия). Отмеченные в [5] недостатки инструмента состоят в сложности его освоения, высоких требованиях к точности документации и эталонной модели, применяемой для верификации, долгом промежутке времени между началом и завершением разработки теста.

OVM (Open Verification Methodology) [6] – разработанная компаниями Mentor Graphics и Cadence методология, согласно которой тест представляет собой пакет случайных или заранее заданных транзакций, подлежащих последовательной передаче в вери-

фицируемое устройство, где специальный драйвер преобразовывает их в битовые векторы. В комплект поставки OVM входит библиотека классов на языке SystemVerilog, которые реализуют основные элементы тестовой системы и ряд дополнительных возможностей по управлению, конфигурированию и отладке тестов. Библиотека позволяет разрабатывать сложные многоуровневые тесты и использовать возможности SystemVerilog для генерации случайных тестов (с ограничениями) и сбора функционального покрытия. Методология OVM хорошо зарекомендовала себя при верификации достаточно крупных модулей проекта «Эльбрус-2S», таких как хост-контроллер, системный коммутатор, контроллеры каналов ввода-вывода и межпроцессорного обмена. Однако в случае относительно небольших устройств со сложным конвейеризированным интерфейсом возникают значительные сложности [7]. Кроме того, отсутствие средств автоматизации разработки в комплекте поставки библиотеки также увеличивает трудоёмкость проектирования и время получения первого теста.

В ходе работ по верификации проекта «Эльбрус-2S» был выявлены основные факторы, создающие определенные проблемы при использовании автономной верификации:

- высокие требования к точности и полноте документации на модули. Зачастую в ходе процесса верификации объём документации на устройство увеличивался в 2-3 раза;
- необходимость представлять устройство в виде функционально дополняющих друг друга частей из-за наличия сложных связей с другими модулями. Например, управляющая логика банка кэш-памяти L2 верифицировалась как два отдельных модуля, в связи с чем в каждой из тестовых систем приходилось имитировать работу недостающего модуля;
- высокая чувствительность к изменениям интерфейса верифицируемого устройства;
- невозможность обнаружения ошибок при объединении устройств на уровне подсистемы и системы.

Однако, несмотря на приведенные особенности автономной верификации с использованием HDL-моделей устройства, она показала себя эффективным инструментом выяв-

ления ошибок, которые не могли быть выявлены ни на аппаратном прототипе, ни при моделировании полного HDL-описания микропроцессора.

3. Верификация с использованием функциональных программных моделей

Системная верификация

Относительные простота и сокращение времени разработки программной модели устройства на языке высокого уровня по сравнению с его verilog-описанием делают ее основой верификации устройств с уровнем сложности микропроцессора «Эльбрус-2S». Функциональные (эталонные) программные модели применяются для автоматизации процесса верификации; они не только моделируют аппаратуру с различной степенью точности, но могут использоваться для поиска оптимальных архитектурных решений, разработки архитектурно-зависимой части операционных систем, отладки программного обеспечения в условиях неготовности аппаратуры и других целей.

Обобщенная структурная схема программной модели на языке C++, использующейся в предыдущих разработках микропроцессоров с архитектурой Эльбрус, представлена на рис. 2.

К ее характерным особенностям можно отнести поддержку работы с неоднородным доступом к памяти (NUMA) и разнообразной периферией, управление трассировкой устройств процессора (такты, инструкции), поддержку контрольных точек, возможность настройки параметров – объема памяти, типов периферийных контроллеров, образов дисков и других. При разработке микропроцессора «Эльбрус-2S» было принято решение, оставляя старую функциональность полностью работоспособной (т.е. поддерживая предыдущие версии архитектуры «Эльбрус»), модернизировать программу в соответствии с нововведениями нового проекта [8].

Все проведенные доработки можно условно разделить на две категории:

- развитие уже существующей функциональности, например, новые операции, рас-

ширяющие возможности существующих;

• существенные доработки структуры модели, наиболее сложными из которых стали поддержка градации обращений в память по поколениям, измененный вариант буфера инструкций и поддержка 64-битовых обращений во вторичное пространство, потребовавшая и введения 64-битовой таблицы страниц.

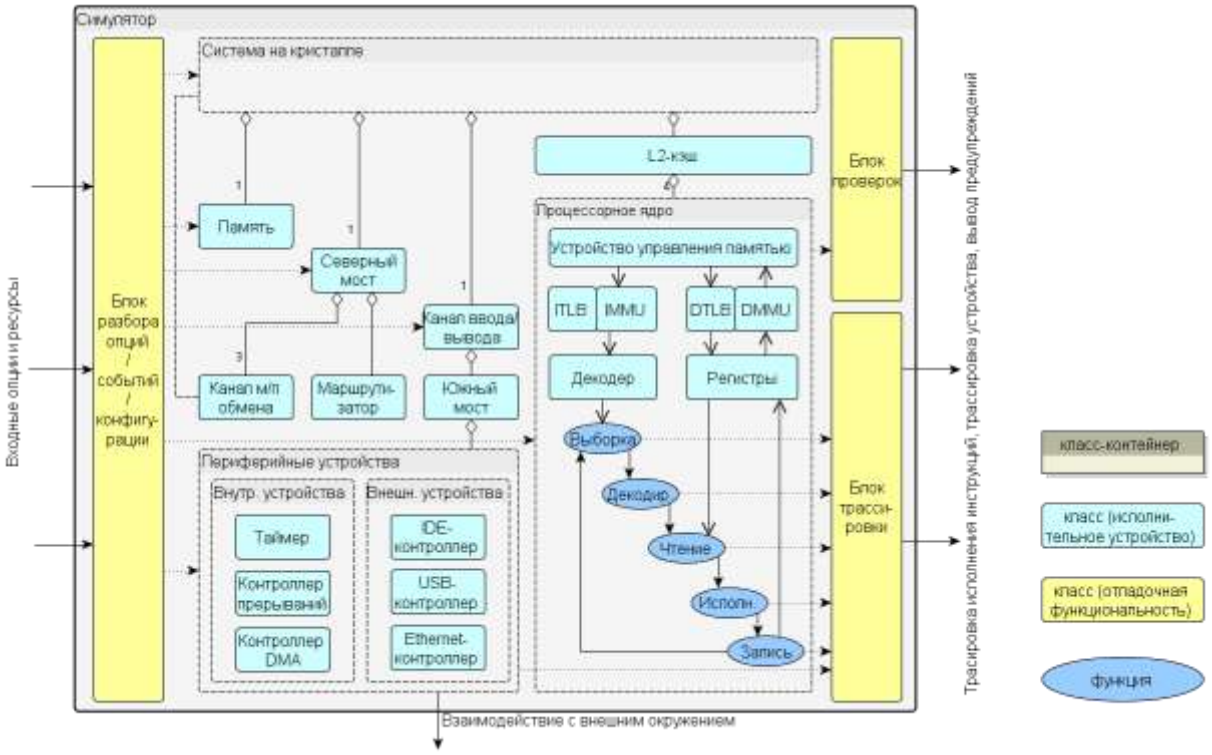

Рис. 2

Структурная схема программной модели

Автономная верификация

Наряду с разработкой функциональной модели проектируемого микропроцессора, модели этого класса использовались и в автономной верификации новых или существенно переработанных устройств: буфера инструкций [4], TLU, устройств, входящих в состав L2-кэша (контроллера банка, WBQ, STMB, устройства поддержки поколений), устройства доступа в память процессорного ядра (MAU), кэша директории в составе системного коммутатора, хост-контроллера, контроллера памяти.

Точность моделирования определялась особенностями конкретного устройства,

часть моделей была реализована как полностью функциональные (MAU, TLU), часть – как полностью потактовые (буфер инструкций, контроллер банка L2-кэша), часть – как гибридные, функциональные с некоторыми потактовыми свойствами (хост-контроллер).

Генерация тестов с опорой на функциональную модель

Методология tgen была использована для обеспечения разработки генераторов, которые, анализируя исполнение предыдущих инструкций, динамически создают проверяющий код. Процесс содержит следующие этапы (рис. 3):

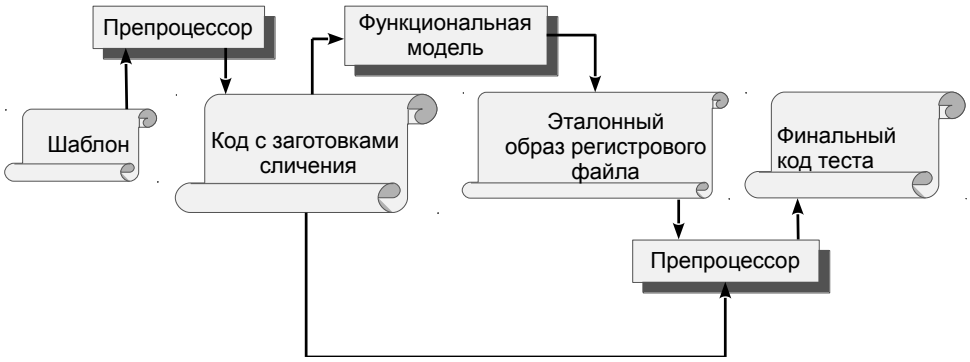

Рис. 3

Генерация тестов с опорой на программную модель

1. Препроцессор на языке python путем упрощенного взаимодействия с инструментом genshi, базирующимся на том же языке, используя полученный от него шаблон, создает тест. В его тексте оставляются пробелы для включения сличающих конструкций. Это позволяет предсказать адреса команд и данных в тестах до того, как сформирован код сличения.

2. Предварительный образ теста исполняется на функциональной модели микропроцессора, модифицированной применительно к этому процессу (из модели исключены модули внешних по отношению к процессору устройств, а все обращения к памяти и периферии обрабатываются в подключающей ее программе, куда управление передается с помощью функций обратного вызова callback). В результате исполнения функциональной модели формируется эталонное состояние регистрового файла.

3. Пробелы в образе теста заполняются кодом, сличающим содержимое регистрового

файла с эталонными значениями. С помощью препроцессора формируется финальный код теста.

Эта схема обладает следующими преимуществами:

- простой доступ к использованию данных эталонной модели;
- автоматическая генерация кода, проверяющего корректность исполнения тестов;
- возможность применения мощного инструмента шаблонизации.

На ее основе было разработано несколько семейств шаблонов, используемых для проверки механизмов обеспечения когерентности памяти и различных режимов доступа в память.

Заключение

В рамках проектирования микропроцессора «Эльбрус-2S» наряду с развитием ранее применявшихся подходов и средств верификации были внедрены новые методы и созданы новые инструменты, которые позволяют расширить возможности и улучшить качество контроля корректности новых разработок семейства «Эльбрус». Тем не менее, значительное количество ошибок, возникающих при функционировании прототипа вычислительного комплекса под операционной системой, показывает необходимость дальнейшей работы в этом направлении.

Литература

1. http://genshi.edgewall.org/

2. Рыжов М.П. Система шаблонной генерации тестов – «Вопросы радиоэлектроники», сер. ЭВТ, 2013, вып. 3, с. 117-125.

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

4. Баратов Р.А., Камкин А.С., Майорова В.М., Мешков А.Н., Сортов А.А., Якушева М.А. Трудности модульной верификации аппаратуры на примере буфера команд микропроцессора «Эльбрус-2S». – «Вопросы радиоэлектроники», сер. ЭВТ, 2013, вып. 3, с. 84-96.

5. http://forge.ispras.ru/projects/cpptesk-toolkit

6. https://verificationacademy.com/courses/basic-ovm

7. Шмелёв В.А., Стотланд И.А. Автономная верификация микропроцессоров на основе эталонных моделей разного уровня абстракции. – В сб.: Научные труды Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС)», 2012, № 1, с. 435-440.

8. Гурин К.Л., Мешков А.Н., Сергин А.В., Якушева М.А. Развитие модели подсистемы памяти вычислительных комплексов серии «Эльбрус». – «Вопросы радиоэлектроники», сер. ЭВТ, 2010, вып. 3.