УДК 004.315 DOI 10.31114/2078-7707-2018-2-86-91

Верификация алгоритма арбитража потоков запросов к памяти

М.Е. Барских, О.И. Эсула

Научно-исследовательский институт системных исследований РАН, г. Москва,
barskikh@cs.niisi.ras.ru, olgaesula@rambler.ru

***Аннотация* — Рассмотрен опыт верификации алгоритма арбитража потоков запросов к памяти от нескольких устройств, проведённой с помощью UVM (Universal Verification Metodology). Описана организация используемого тестового окружения. Представлены преимущества выбранного способа верификации. Обосновывается возможность повторного использования созданного тестового окружения.**

***Ключевые слова* — верификация, арбитраж, QoS, UVM.**

I. Введение

Основной функцией контроллера памяти микропроцессора является обеспечение доступа различных устройств к внешней памяти. Одновременно в память может быть направлено несколько запросов, но исполняться они будут по порядку, определяемому арбитром. В соответствии с алгоритмом арбитража выставленным на входы арбитра запросам назначаются приоритеты, что обеспечивает доступ к памяти только одному запросу. При выборе алгоритма арбитража запросов в память учитываются требования технического задания, а именно, пропускная способность подсистемы динамической памяти, тактовая частота работы контроллера памяти, перечень устройств, обращающихся к памяти, и их требования к ширине пропускания канала памяти.

При разработке микропроцессора 1890ВМ9 (далее упоминается только номер процессора, без серии) с архитектурой КОМДИВ64 оказалось неприемлемым использовать алгоритм арбитража потоков запросов, реализованный для микропроцессора предыдущей версии ВМ8. Так, на ПЛИС-прототипе при разрешении 1024x168 и частоте памяти DDR3 75МГц при интенсивном обращении нескольких устройств к памяти на экране наблюдалось непериодическое дрожание изображения. Пропускная способность канала памяти была повышена за счёт изменения алгоритма арбитража, а для операций вывода на экран установлен более высокий приоритет. По этой причине возникла задача тщательной проверки работоспособности алгоритма арбитража, разработанного для микропроцессора ВМ9.

Тесты основного маршрута верификации, применяемых для проектов НИИСИ РАН, являются программами, выполняемыми процессором. В случае проверки алгоритма арбитража запросов в память этот подход является сложным и медленным в силу следующего ряда причин:

- тест должен запрограммировать контроллеры всех устройств, обращающихся к памяти, подготовить дескрипторы и данные в память;
- в проекте должна быть модель этих устройств, имитирующая их поведение;
- тест должен идти достаточное время, чтобы все устройства выполнили требуемые обращения в память.

На время тестирования, кроме длительности самого теста, влияет скорость моделирования, которая будет невысокой из-за объема и сложности моделируемого проекта. Поэтому было принято решение в качестве мастеров на шине контроллера памяти использовать UVM-агенты (рис. 1), а их поведение описывать используя библиотеку последовательностей настраиваемых запросов. В качестве языка реализации выбран Specnam *e*, который, с одной стороны, позволяет использовать методологию UVM, а с другой стороны, объем исходного кода агентов и всего тестового окружения по сравнению с UVM-SV (Universal Verification Metodology SystemVerilog) у него меньше.

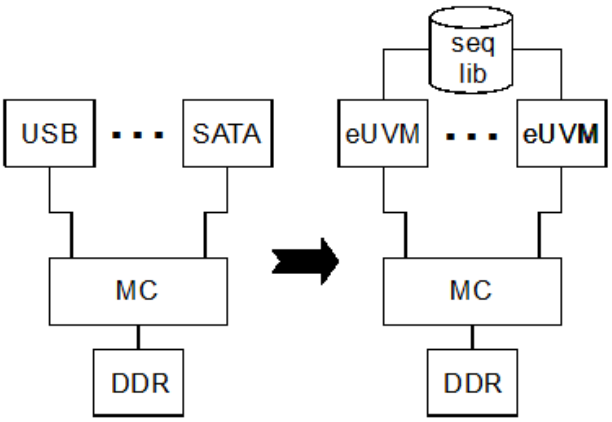

Рис. 1. Подключение мастеров к шине контроллера памяти (MC – контроллер памяти; DDR – модель внешней памяти; USB, SATA – контроллеры USB, SATA и их модели; eUVM – UVM-агенты; seq lib – библиотека последовательностей настраиваемых запросов)

II. Алгоритм арбитража запросов в память

Задача распределения приоритетов между несколькими одновременно выставленными запросами

к одному ресурсу имеет известные решения [1]. Реализация арбитража с помощью фиксированных приоритетов требует минимум ресурсов, но гарантирует доступ запросу только с самым высоким приоритетом. Алгоритм арбитража Round-Robin [2] обеспечивает равные условия доступа всем запросам, таким образом вообще не учитывается их приоритетность. В микропроцессоре ВМ8 из-за большого количества устройств, обращающихся в память, реализовано 2 уровня арбитража [3]. На первом уровне вне контроллера памяти используется алгоритм Round-Robin, а на втором для арбитража запросов внутри контроллера памяти реализован алгоритм распределения приоритетов Least Recently Used (LRU) [4]. Реализация LRU не потребовала сложной логики с большим количеством элементов. Этот алгоритм отлаживался с помощью тестов основного маршрута верификации, для прохождения которых оказалось достаточным нескольких часов.

Перечисленные алгоритмы арбитража запросов к памяти не позволяли обеспечить необходимый уровень качества обслуживания устройств (Quality of Service, QoS) во всех режимах, требуемых в техническом задании для микропроцессора ВМ9. По этой причине был реализован усложнённый механизм LRU в области предоставляемой устройству ширины полосы пропускания [3], кратко описанный ниже.

Контроллер памяти микропроцессора ВМ9 имеет 7 каналов, работающих по протоколу AXI3 (Advanced eXtensible Interface). Каждому каналу соответствует определенный приоритет от 1 до 7, меняющийся по алгоритму LRU. В любой момент времени все приоритеты разные. Дополнением к алгоритму LRU является присвоение каждому каналу контроллера памяти некоторого числа (обозначается далее n_i, где i – номер канала). n_i определяет количество запросов, передаваемых по каналу i и имеющих неизменный приоритет. Ниже описан механизм изменения n_i для канала i и его влияние на значение приоритета. При одновременно выставленных запросах по каналу i и какому-то другому каналу сравниваются приоритеты каналов. Если приоритет канала i ниже, то значение приоритета увеличивается. Иначе запросу предоставляется доступ в память, при этом приоритет остаётся неизменным, а $n_i = n_i - 1$. Когда n_i достигает 0 приоритет становится минимальным, а n_i принимает начальное значение, которое является настраиваемой величиной и задаётся программистом. Реализованный алгоритм арбитража позволяет изменять ширину полосы конкретного канала в зависимости от выполняемой вычислительной задачи в процессе работы.

III. Повышение эффективности верификации

Верификация правильности RTL-описания реализованного алгоритма арбитража запросов в память проводилась на основе плана тестирования и управлялась тестовым покрытием. План тестирования содержал 3 раздела:

1) Интерфейсная часть: контроль корректности реализации входных и выходных интерфейсов контроллера памяти. Так как использовался стандартный интерфейс AXI3, который реализован в большом количестве блоков и проектов, и его верификация не являлась основной частью описанной в статье работы, то данный раздел содержал в основном ссылки на формальные утверждения (*assertion*). Для контроля корректности использовалась библиотека ARM [5].

2) Функциональное ядро: описание тестирования именно механизма арбитража, учитывающее особенности его реализации (т.е. блок рассматривался как «белый ящик» – *White Box*). Это позволяло перечислить состояния RTL-модели, которые должны быть достигнуты во время тестов.

3) Специальные случаи: здесь описывались так называемые «*corner case*» и критические ситуации, которые должны быть покрыты тестами. Часть из этих случаев может быть описана во втором разделе, но здесь они перечисляются именно с целью выделить те особые ситуации (в запросах и/или поведении контроллера), которые необходимо проконтролировать «вручную».

Для описания формальных утверждений (*assertion*) и функционального покрытия (*functional coverage*) выбран язык SystemVerilog, так как его использование упрощает проверку при выполнении тестовых программ в рамках основного маршрута верификации.

План верификации организован как древовидная структура записей, уточняющихся по мере углубления уровня вложенности, и завершающаяся записью, которая имеет ссылку на пункт технического задания.

В соответствии с планом верификации написан список утверждений и критических ситуаций. Т.е. каждой завершающей записи плана верификации сопоставляется или assertion или пункт functional coverage. Достижение этих точек в процессе тестирования позволяет не только видеть прогресс тестирования, но и корректировать тестовые воздействия для скорейшего достижения нужного результата. Тест может корректироваться двумя способами: или за счет добавления новых тестовых последовательностей (например, пакетная передача данных с декрементом адреса) или за счет изменения ограничений (constrain) на генерацию параметров. Полное выполнение тестового плана служит объективным критерием окончания процесса тестирования.

Утверждения разделены на две группы – проверка корректности выдачи ответов на внешние воздействия по протоколу AXI3 и проверка внутренних критических ситуаций, связанных непосредственно с работой алгоритма арбитража запросов в память. Все необходимые для описания внутренних ситуаций сигналы находятся в одном модуле, поэтому решено разместить эти утверждения в нём же, не выделяя их в отдельный файл. Утверждения, проверяющие обмен

информацией по протоколу AXI3, размещены в тестовом окружении (рис. 2).

Для проверки корректности работы алгоритма арбитража применен уже известный набор утверждений [6]. Например, любой выставленный на вход контроллера памяти запрос должен быть обработан или нельзя выдавать разрешение на обработку более, чем одному запросу. Также описаны критические ситуации, соответствующие данной реализации. Например, пока число n_i не нулевое, соответствующий каналу i приоритет не меняется.

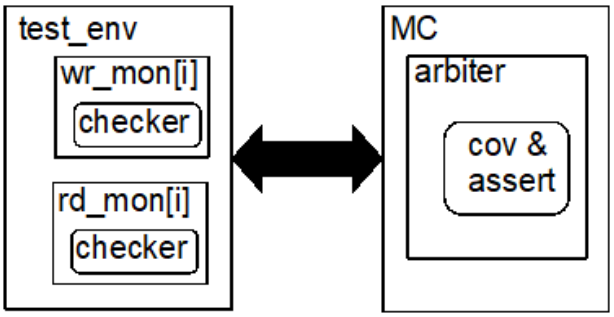

Рис. 2. Схема размещения проверочных утверждений. test_env – тестовое окружение; wr/rd_mon[i] – мониторы UVM-агентов канала записи/чтения, i = 0, …, 6; checker – проверочные утверждения; arbiter – модуль, описывающий алгоритм арбитража запросов в память; cov & assert – набор критических ситуаций и проверочных утверждений

IV. Организация тестов

Внешними воздействиями для контроллера памяти являются запросы, посылаемые 7 UVM-агентами по независимым каналам записи и чтения, работающим по протоколу AXI3. Все запросы формируются при помощи шаблона запроса, у которого есть набор необходимых параметров (табл. 1).

Таблица 1

Перечень параметров шаблона запроса

Название параметра	Описание параметра
	Параметры запроса
id	Идентификационный номер запроса
unalign	Признак, что адрес не выровнен (необходим для расчёта значения маски данных)
size	Число, определяющее размерность шины данных (= $2^{(size+3)}$)
length	Число, определяющее количество фаз данных (= length+1)
address	Адрес
mask	Маска данных
data	Данные
	Параметры, регулирующие плотность выдачи запросов
delay_before_address	Длительность паузы перед адресной фазой
delay_before_data	Длительность паузы перед началом передачи данных
delay_in_data	Длительность пауз между фазами данных

Проверка корректности реализации интерфейсов контроллера памяти осуществлялась с помощью изменения параметров запроса (длина, адрес, данные и др.). В шаблоне кроме логических параметров собственно запроса присутствуют параметры, отвечающие на задержки в выдаче запросов. Эти параметры учитываются драйвером шины и дают возможность в тесте регулировать загрузку шины, что делает тестовые сценарии более гибкими и разнообразными. На рис. 3 продемонстрирован пример части теста, состоящего из трёх запросов по записи со случайными параметрами плотности выдачи запросов.

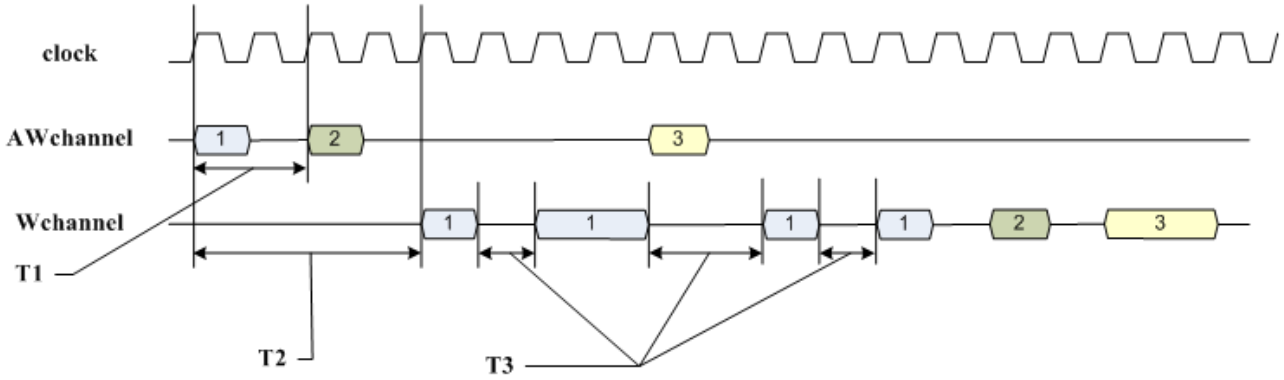

Рис. 3. Диаграмма нагрузки шины записи тремя запросами с разными параметрами плотности выдачи запросов. clock – сигнал тактовой частоты, AW channel – адресная фаза запроса по записи, W channel – фаза данных, T1 – временной интервал между запросами (между фазами передачи адреса двух разных запросов), T2 – временной интервал до записи данных (между фазами передачи адреса и данных одного запроса), T3 – временной интервал между фазами передачи данных одного запроса

Последовательности настраиваемых запросов определяют входные потоки запросов для тестируемого модуля. Сами последовательности в свою очередь реализованы в виде шаблонов в библиотеке, к которой обращается тест при формировании тестового сценария (рис. 4). В соответствии с методологией UVM шаблоны

реализованы иерархически, позволяя формировать несколько запросов в соответствии с заложенными в них алгоритмами. Для проверки работоспособности алгоритма арбитража запросов в память выбраны три типа последовательностей:

1) SINGLE – последовательность, состоящая из одного запроса;

2) MULTIPLE – последовательность, состоящая из нескольких запросов (по умолчанию 16), все запросы имеют разный размер и обращаются по разным адресам;

3) BURST - последовательность, состоящая из нескольких запросов (по умолчанию 16), имеющих одну длительность и идущие «подряд» адреса.

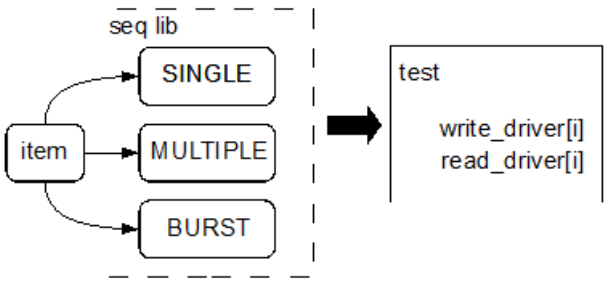

Рис. 4. Схема создания теста (item – шаблон настраиваемого запроса; seq lib – библиотека последовательностей запросов; test - тест; write/read_driver[i] – драйвера независимых каналов записи или чтения для каждого UVM-агента, i = 0, .., 6)

Тест считается успешно прошедшим, если данные по чтению совпали с ранее записанными (или теми, что инициализировали память), и при этом не было ошибок в ситуациях, описанных утверждениями. Механизм сравнения данных сделан универсальным для трёх значений параметра протокола AXI3 «size» = 4, 3, 2 [7], так как именно они используются при имитации входных запросов в память. Рассматриваемое в статье тестирование направлено не на проверку когерентности данных, а на проверку алгоритма арбитража запросов в память. Т.е. проверяется не арбитраж обращения нескольких устройств к одной области памяти, а только разделение доступа к ней. Поэтому для упрощения генерации тестов решено каждому UVM-агенту выделить свой диапазон адресов с учётом структуры DDR, размера страниц и банков памяти. Кроме того, принятые меры позволили упростить механизм сравнения данных по записи и чтению, что позволило ускорить процесс верификации.

Более полное покрытие RTL-модели тестами обеспечено использованием виртуальных последовательностей, позволяющих верификатору контролировать выбор типа последовательностей запросов и параметры запросов, а также другие настройки тестового сценария.

V. Процесс моделирования

Моделирование проводилось с помощью симулятора Cadence Incisive Enterprise Simulator. Запуск тестов осуществлялся с помощью программы Emanager, входящей в комплект поставки. В командном файле выбирались тесты UVM, требуемые для запуска, и параметры самого запуска (такие как количество итераций и значение ядра случайного генератора (seed) для синтезатора). В Emanager также велась статистика запусков. Помимо контроля ошибочных тестов и регрессионного тестирования это позволяло собрать и объединить покрытия (покрытие кода и функциональное) для различных запусков.

На основе анализа собранной информации менялись ограничения для запросов и сценарии тестов. Используемые средства создания тестовых сценариев и оценки функционального покрытия позволили добиться высокого качества отладки.

VI. Результаты

Описанный в работе пример верификации позволил обнаружить 7 ошибок в RTL-модели на языке Verilog алгоритма арбитража запросов в память для микропроцессора BM9. Также 5 ошибок в реализации механизма расщепления запросов при наличии перехода через ранки памяти. Кроме того обнаружены ещё 4 ошибки в контроллере памяти: первая – при работе в режиме включения кода Хемминга, вторая – в блоке, меняющем формат запроса с AXI3 на понятный контроллеру памяти, третья – при имитации запросов с параметром «size» = 3 и случайными масками. Последняя ошибка проявлялась при тестировании проекта на ПЛИС (Программируемая Логическая Интегральная Схема) фирмы Altera, но точного места ошибки не удавалось найти в течение нескольких дней. Благодаря коротким, управляемым и возобновляемым тестовым сценариям, описанным выше, по указанным разработчиком параметрам запросов на поиск ошибки понадобилось 4 часа. При этом большая часть времени ушла на воссоздание требуемой ситуации.

Реализованный алгоритм арбитража позволил решить поставленную задачу повышения пропускной способности канала памяти с приоритетом для операций вывода на экран. На ПЛИС-прототипе при разрешении 1024x168 и частоте памяти DDR3 75МГц при интенсивном обращении нескольких устройств к памяти на экране более не наблюдалось дрожание изображения [3].

VII. Преимущества

Можно выделить три основных преимущества представленного в статье способа верификации по сравнению с тестированием проекта на ПЛИС фирмы Altera:

1) Сокращение времени, затрачиваемого на поиск ошибки, с 13 часов до 4 часов, так как невыполнение хотя бы одного утверждения из всего набора сразу останавливает процесс тестирования;

2) Уменьшение объёма используемой памяти (рис. 5) за счёт того, что моделировалась не вся система целиком, а только её часть и тестовое окружение в виде UVM-агентов и библиотеки;

3) Сокращение времени моделирования благодаря использованию коротких направленных тестов и моделированию только интересующей части системы, как и для предыдущего пункта.

Рис. 5. Сравнение объёма используемой памяти. 1 - тестирование проекта на ПЛИС фирмы Altera, 2 - тестирование проекта описанным в статье способом

VIII. Заключение

Необходимость верификации аппаратной реализации нового алгоритма арбитража запросов в память для микропроцессора ВМ9 привела к созданию тестового окружения с несколькими агентами, работающими по протоколу AXI3, и с универсальным механизмом проверки данных по чтению. Использование проверочных утверждений позволило сделать проверку более эффективной. Гибкая система настройки тестовых сценариев дала возможность покрыть все перечисленные критические ситуации.

Проверка представленным способом позволила обнаружить ошибки в обособленных участках проекта за меньшее время в сравнении с основным методом верификации проекта.

Тестовая система организована так, что её использование стало возможным при проверке других частей проекта, и планируется её использование при верификации нового проекта.

Литература

[1] Mohan S., Joseph A. A Dynamic Priority Based Arbitration Algorithm // International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2013. V. 3, № 1. P. 232-233.

[2] Nosrati M., Karimi R., Hariri M.. Task Scheduling Algorithms Introduction // World Applied Programming. 2012. V. 2, № 6. P. 394-398.

[3] Корниленко А.В., Эсула О.И. Оптимизация подсистемы памяти вычислительной системы с помощью предоставления гарантированной полосы пропускания канала памяти // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. №3. С. 136-140.

[4] Aravind A.A.. An arbitration algorithm for multiport memory systems. // IEICE Electronics Express. 2013. V. 2, № 19. P. 1-7.

[5] URL: https://silver.arm.com/download/download.tm?pv=1242915&p=1073545 (дата обращения 21.03.2018).

[6] Foster H., Krolnik A. Creating Assertion-Based IP // Springer. 2008. P. 318.

[7] URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022f.b/index.html (дата обращения 21.03.2018).

Verification of Memory Requests Arbitration Algorithm

M.E. Barskikh, O.I Esula

SRISA RAS, Moscow, barskikh@cs.niisi.ras.ru, olgaesula@rambler.ru

***Abstract* — Memory controller in microprocessor provides access to external memory for a number of system components. Some requests can be set in the same time. So there is a request arbiter in the memory controller that orders requests by priorities. Arbitration algorithm [1] depends on technical requirements. It was impossible to reuse in microprocessor 1890VM9 the arbitration algorithm that was designed for the previous version 1890VM8. The main verification path in SRISA RAS is a list of programs executed by the processor. This verification path is difficult and slow. The article considers arbitration algorithm verification that uses UVM. Masters on memory controller bus are UVM-agents, their behavior is described by using request sequence library. We used Specman *e* because agents' initial code and test environment are less in comparison with UVM-SV.**

Microprocessor 1890VM9 has 2 levels of arbitration. The first level uses Round-Robin [2] algorithm. The second level [3] uses Least Recently Used (LRU) [4] arbitration algorithm with some modification that allows to change the bandwidth of a particular memory controller channel depending on the computational task in progress [3].

Arbitration verification was based on a test plan and managed by test coverage. Test plan had references to technical documentation and assertion or functional coverage written in SystemVerilog. The ARM library [5] was used to check correctness. We used well-known set of assertions [6] and project specified assertions to check the correctness of the arbitration algorithm. Achieving of these points allows monitoring the progress of testing and adjusting the test effects to achieve the desired result as soon as possible. The test can be adjusted by adding new test sequences or by changing constraints on parameter generation. Full execution of the test plan serves as an objective criterion for the end of the testing process.

UVM agents sent read or write requests to the memory controller. All requests were generated using a request template that had a set of required parameters. A test is successful if the read data matches the previously written data (or initial memory data) and there are no errors in the situations described by the assertions.

Emanager conducted statistics runs. In addition to error test control and regression testing, this allowed us to collect and combine coverages (code coverage and functional) for different runs. Based on the analysis of the collected information, request and test scenario limits were changed.

Verification described in the article allowed to detect 7 errors in the RTL-model of the memory controller request arbitration algorithm. In addition, 9 errors were detected in the memory controller that were not related to arbitration algorithm.

The test system can be reused for verification other part of the project or in other project.

***Keywords* — verification, arbitration, QoS, UVM**

REFERENCES

[1] Mohan S., Joseph A. A Dynamic Priority Based Arbitration Algorithm // International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2013. V. 3, № 1. P. 232-233.

[2] Nosrati M., Karimi R., Hariri M.. Task Scheduling Algorithms Introduction // World Applied Programming. 2012. V. 2, № 6. P. 394-398.

[3] Kornilenko A.V., Esula O.I. Optimizatsiya podsistemy pamyati vychislitel'noj sistemy s pomosch'ju predostavlenija garantirovannoj polosy propuskanija kanala pamyati (Computer memory subsystem optimization by providing guaranteed memory bandwidth) // Problemy razrabotki perspestivnyh mikro- i nanojelectronnyh sistem (MES). 2016. №3. P. 136-140.

[4] Aravind A.A.. An arbitration algorithm for multiport memory systems. // IEICE Electronics Express. 2013. V. 2, № 19. P. 1-7.

[5] URL: https://silver.arm.com/download/download.tm?pv=1242915&p=1073545 (access date 21.03.2018).

[6] Foster H., Krolnik A. Creating Assertion-Based IP // Springer. 2008. P. 318.

[7] URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022f.b/index.html (access date 21.03.2018).