Для цитирования: Порошин П. А., Мешков А. Н., Черных С. В. Разработка симулятора, поддерживающего потактовый режим работы, на основе текущей версии функционального симулятора архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2018. № 2. С. 69–75.
УДК 004.414.23

П. А. Порошин[1, 2], А. Н. Мешков[1, 3], С. В. Черных[3]

[1] ПАО «ИНЭУМ им. И. С. Брука», [2] МТУ (МИРЭА), [3] АО «МЦСТ»

РАЗРАБОТКА СИМУЛЯТОРА, ПОДДЕРЖИВАЮЩЕГО ПОТАКТОВЫЙ РЕЖИМ РАБОТЫ, НА ОСНОВЕ ТЕКУЩЕЙ ВЕРСИИ ФУНКЦИОНАЛЬНОГО СИМУЛЯТОРА АРХИТЕКТУРЫ «ЭЛЬБРУС»

В процессе разработки вычислительных систем и сопутствующего им программного обеспечения возникают потребности в программных моделях процессоров разной степени детализации и быстродействия. Быстроту моделирования обеспечивают функциональные модели, высокую детализацию – потактовые. Одновременная разработка функционального и потактового симуляторов является трудоемкой задачей. В данной работе рассматривается задача снижения сложности разработки и поддержки программных моделей разного уровня точности для микропроцессоров архитектур широкого командного слова. В качестве решения предлагается построение одной программной модели, способной работать в качестве как функционального, так и потактового симулятора. На примере архитектуры «Эльбрус» описывается способ построения такой модели, основанный на постепенном расширении и уточнении функционального симулятора до возможности потактового моделирования и на полуавтоматическом упрощении получившейся потактовой модели до функциональной. Производительность получившейся функциональной модели не уступает оригинальной, замедление потактовой модели относительно функциональной оценивается в пределах 10–15 раз.

Ключевые слова: *архитектура «Эльбрус», программное моделирование, функциональный симулятор, потактовый симулятор.*

Введение

На разных этапах проектирования устройства или системы ставятся разные требования к технологии их моделирования, основными из которых является высокая производительность, достигаемая в функциональных симуляторах, и высокая точность с возможностью доступа к информации о времени исполнения операций, свойственная потактовым симуляторам. Обычно такие цели достигаются применением симуляторов двух типов, хотя разработка и поддержка каждого из них очень трудоемки. В связи с этим в рамках общей работы по верификации микропроцессоров с архитектурой «Эльбрус» в АО «МЦСТ» был предложен и экспериментально опробован изложенный в данной статье метод построения потактового симулятора на основе функционального с сохранением возможности функционального моделирования.

Моделирование исполнения широкой команды в оригинальном функциональном симуляторе архитектуры «Эльбрус»

Относящаяся к классу VLIW архитектура «Эльбрус» [1] предполагает параллельное исполнение совокупности операций, закодированных в пределах одной широкой команды (ШК). Это свойство в определенной степени усложняет моделирование и требует специальных механизмов для корректной реализации данного принципа даже в пределах функционального симулятора.

В частности, операции в пределах одной широкой команды не должны видеть вносимые друг другом изменения в текущее состояние процессора. Усложняется учет особых ситуаций и прерываний – например, при возникновении точного прерывания во время исполнения какой-либо из составляющих операций вся широкая команда не должна вносить изменений в текущее состояние. По определенным правилам должны разрешаться и WAW-конфликты (Write-after-Write, конфликт «запись-запись»), возникающие в пределах одной инструкции.

Для корректного учета таких особенностей в процесс, реализуемый оригинальным симулятором, помимо понятия «стадии», характеризующего заложенные в проекте рабочие состояния устройства (и, соответственно, модели), вводятся понятия «псевдостадии чтения» и «псевдостадии записи». В первом случае имеется в виду считывание состояния

процессора, необходимого для выполнения операции, без внесения в него модификаций, во втором – модификация текущего состояния. Алгоритмические действия, соответствующие моделируемой операции, могут быть реализованы как в псевдостадии чтения (после считывания состояния), так и в псевдостадии записи (до записи результата). Моделирование всей широкой команды сводится к последовательному исполнению псевдостадии чтения всеми операциями широкой команды, проверке наличия особых ситуаций и последующему исполнению псевдостадии записи (рис. 1). Порядок выполнения каждого этапа моделирования, в особенности псевдостадии записи, также не случаен, ибо от него зависит итоговое состояние процессора при наличии каких-либо конфликтов между операциями.

Рисунок 1. Алгоритм моделирования широкой команды

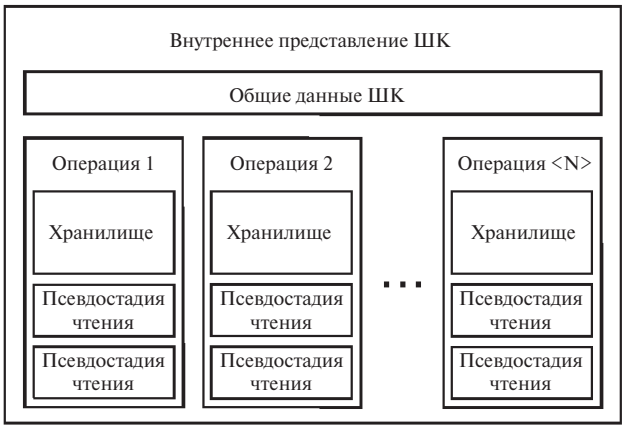

Рисунок 2. Внутреннее представление широкой команды в функциональном симуляторе

В процессе моделирования симулятор оперирует внутренними представлениями широкой команды (рис. 2), которые формируются на этапе декодирования. Внутреннее представление состоит из некоторых общих для всей широкой команды данных и массива структур, описывающих индивидуальные операции. Структуры состоят из указателя на псевдостадию чтения, указателя на псевдостадию записи и хранилища, через которое передаются данные между ними.

Принципиальные варианты совмещения функционального и потактового режимов работы в одном симуляторе

В контексте изложенной во введении проблемы целесообразно охарактеризовать возможные методы, используя которые, можно сократить большой объем работы, неизбежный при одновременном проектировании нескольких симуляторов.

Прежде всего, есть предпосылки для использования высокоуровневых языков описания микропроцессорных архитектур. Некоторые из них, например [2–4], способны по единому описанию микропроцессора сгенерировать как функциональный, так и потактовый симулятор. К сожалению, выразительные способности таких языков ограничены, в силу чего описание на них комплексных и не вполне традиционных архитектур затруднено. Так, например, не видится возможным на них описать асинхронный относительно основного поток команд. Кроме того, в данном случае непросто использовать наработки, накопившиеся при разработке существующего функционального симулятора.

Другой подход основан на построении не полноценного самодостаточного потактового симулятора, а некоторой надстройки над функциональным симулятором, которая ответственна за расчет времен исполнения операций и задержек [5, 6]. В этом случае происходит разграничение зон ответственности: функциональный симулятор отвечает за корректность исполнения инструкций, надстройка – за моделирование временных характеристик исполняемого кода. Недостатками этого решения являются ограниченность эффектов, которые способна моделировать надстройка, и неизбежные сложности при реализации влияния надстройки на поведение функционального симулятора (что,

например, может быть необходимо при моделировании многоядерных систем). Следует отметить и то обстоятельство, что применительно к архитектуре «Эльбрус» выделение полученной функциональным симулятором информации, которая необходима в надстройке для расчета задержек, достаточно сложно.

В силу этих причин авторами был предложен и проанализирован описанный далее подход, основанный на построении одного симулятора, который поддерживает как функциональный, так и потактовый режим моделирования. В функциональном режиме симулятор должен обеспечивать высокую скорость исполнения инструкций, в потактовом – высокую точность и возможность получения информации о задержках и блокировках. Основная идея заключается в полуавтоматической реализации менее детального режима моделирования (функционального) на основе более детального (потактового). Надо особо отметить, что в данном случае сокращаются усилия по одновременной поддержке обеих моделей, т.к. по существу необходима поддержка только более детализированного (потактового) режима моделирования.

Расширение функционального симулятора до потактового

Принципиальные решения

Естественной оптимизацией выбранного подхода является сокращение затрат на разработку потактового симулятора на начальном этапе за счет использования кода уже существующего функционального симулятора. Более конкретно – потактовую модель можно реализовать путем постепенного уточнения функциональной модели, добавления в нее конвейерных структур и других элементов. Часть действующих модулей позволительно использовать без модификаций (например, регистровый файл), другую часть с небольшими изменениями можно временно внедрить в качестве упрощенной модели и уточнить позже. По ходу этого процесса важно сохранять работоспособность функционального симулятора (что достигается без затруднений при его полуавтоматической генерации из потактовой модели), механизм которого подробно описывается ниже.

Исполнение операций в потактовом режиме было решено реализовывать по принципу непосредственного моделирования их конвейерных стадий в виде набора соответствующих функций (далее называемых стадийными функциями), что до некоторой степени развивает представленную выше идею псевдостадий функционального симулятора. Сами стадийные функции реализуются путем «дробления» псевдостадий функционального симулятора и их последующего уточнения.

Внутреннее представление широкой команды также подверглось изменениям (рис. 3). Вместо массива структур, соответствующих операциям, оно содержит совокупность массивов, включающую один массив хранилищ и по массиву указателей на стадийные функции для каждой из моделируемых стадий. Каждый массив заполняется на основе всех, составляющих широкую команду операций. Причем в качестве оптимизации хранятся указатели только на нетривиальные стадийные функции (тривиальной считается стадийная функция, которая ничего не делает).

Модель конвейера команд

В основе потактового симулятора лежит программная модель конвейера команд. В ней учитываются только стадии, существенно различные для разных операций. Более ранние стадии конвейера, поведение которых одинаково для всех операций, на данный момент обрабатываются упрощенно. Это, в частности, касается стадий, ответственных за подкачку кода и декодирование инструкций.

Программная модель конвейера реализована с помощью кольцевого буфера, в котором хранятся промежуточные представления широких команд вместе с некоторой дополнительной информацией, включающей номер текущей стадии команды и, в случае ее блокировки, номер такта ее снятия.

Моделирование работы конвейера сводится к обходу лежащих в буфере широких команд и вызову для каждой широкой команды стадийных функций ее подопераций. Обход проводится в программном порядке (от старших команд к младшим), что, в частности, необходимо для корректности информационных передач между командами,

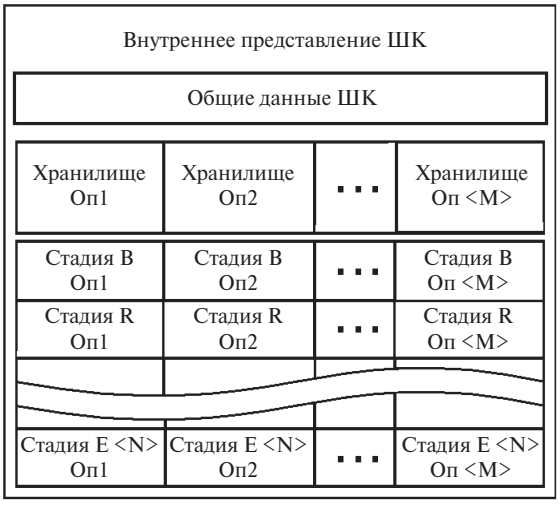

Рисунок 3. Внутреннее представление ШК потактового симулятора: B, R, E<N> – названия моделируемых стадий конвейера; Оп<M> – сокращение от «Операция номер <M>»

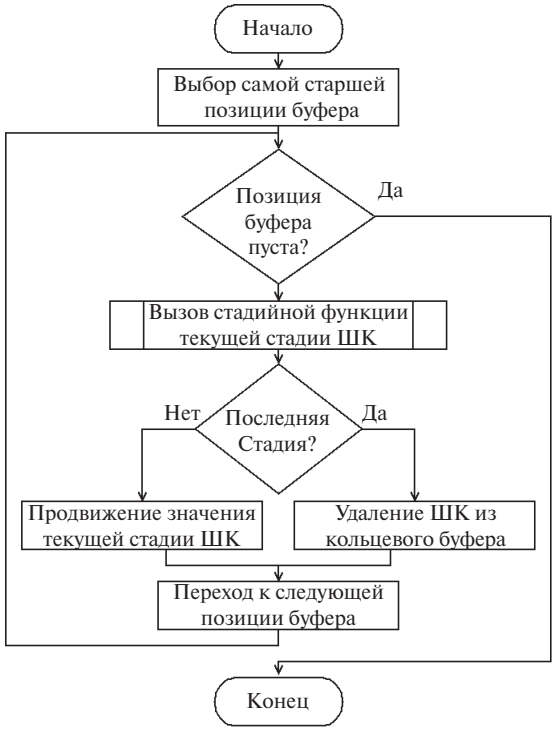

Рисунок 4. Упрощенный алгоритм моделирования конвейера

выставления состояний байпасов и контроля блокировок (рис. 4).

Контроль блокировок

Архитектура «Эльбрус» предполагает статическое планирование выполнения операций и обязывает программиста соблюдать определенные соотношения между временами запуска операций на исполнение. Несоблюдение этих соотношений может приводить к нарушению логики работы программы или к возникновению блокировок конвейера. И если нарушение логики отслеживается еще на уровне функциональной симуляции, то блокировки конвейера игнорируются в силу их значимости только для времени исполнения программы функциональной моделью – они становятся зоной ответственности потактового симулятора.

Если во время моделирования операции потактовым симулятором обнаруживается ситуация, вызывающая блокировку, то информация о ней передается в модель конвейера. Модель конвейера, в свою очередь, останавливает исполнение и продвижение по стадиям команды, вызвавшей блокировку, и всех следующих за ней широких команд. В момент снятия блокировки исполнение всех заблокированных команд возобновляется.

Одной из причин блокировки является неготовность операндов, подлежащих передаче между операциями. При вычислении операцией результата его значение не сразу попадает в регистровый файл и для последующих операций какое-то время доступно через систему байпасов. Разные операции имеют отличающиеся времена вычисления результата, используют разные байпасы при чтении операндов и при записи результатов. Помимо этого, есть ограничения на типы и размеры значений, передаваемых по байпасам. С целью учета этих особенностей в потактовом симуляторе реализована модель байпасирования.

Для каждой ячейки регистрового файла модель хранит информацию о том, лежит ли актуальное значение в регистровом файле, с каких байпасов оно доступно (в данный момент и в ближайшее время), а также тип записанного результата. Помимо этого, в рамках модели описаны разрешенные передачи данных из байпасов в операции в качестве считываемых операндов. Этой информации достаточно для определения готовности операндов и обнаружения связанных с этим блокировок.

Модель байпасов обновляется непосредственно в стадийных функциях операций. Корректность информационных передач и состояний байпасов также во многом обеспечивается порядком исполнения стадийных функций. Исполнение стадийных функций от старших команд к младшим обеспечивает корректность передач между разными широкими командами, порядок исполнения стадийных функций в пределах одной из них, разрешение конфликта «запись–запись» и корректную установку состояния байпасов.

Выбор режима моделирования и построение функционального симулятора

То, какой именно симулятор – потактовый или функциональный – будет собран из исходных кодов, контролируется специальным флагом препроцессора, выбираемым при компиляции. Компиляция функциональной версии опирается на возможность полуавтоматической генерации исходного кода функционального симулятора на основе кода потактового. Процесс генерации сводится к следующим преобразованиям кода:

1. Из кода удаляются ненужные функциональному моделированию структуры (например, модель байпасов) и/или вместо них устанавливаются тривиальные заглушки с совпадающим интерфейсом.
2. Некоторые уточненные для потактовой модели структуры заменяются на упрощенные функциональные. Например, более сложное внутреннее представление широкой команды заменяется на оригинальное функциональное, модель конвейера заменяется на интерпретационный цикл.
3. На основе стадийных функций формируются близкие к оригинальным псевдостадии чтения и записи, что достигается путем «склейки»

стадийных функций (рис. 5). Это, в свою очередь, возможно в силу того, что сами стадийные функции были получены из псевдостадий путем их «дробления».

Все преобразования реализованы с помощью стандартных средств препроцессора (CPP) и метапрограммирования, доступных в языке C++.

Генерацию функционального симулятора можно производить и без полностью рабочего потактового. Это позволяет в процессе разработки и уточнения потактовой модели в новом симуляторе иметь доступ к его функциональному режиму работы, а также снимает необходимость в поддержке оригинального симулятора.

Эффективность всего преобразования во многом опирается на оптимизационные возможности используемого компилятора. Полученный опыт показывает, что их достаточно для получения функционального симулятора, сопоставимого по производительности с оригинальным.

Оценка производительности

Как было отмечено, определяющей характеристикой функционального симулятора является его быстродействие. Поэтому одним из требований к функциональному режиму нового симулятора было сохранение им производительности оригинальной функциональной модели. С целью проверки этого условия проводилось сравнительное тестирование производительности двух симуляторов (оригинального функционального и нового в функциональном режиме). В качестве тестовых задач использовались: программа начальной загрузки (boot), загрузка операционной системы Linux (linux), работа системного двоичного компилятора (lintel), работа пользовательского двоичного компилятора (user lintel). Результаты представлены в табл. 1, в качестве измерений приведены данные о среднем числе моделируемых широких команд в секунду (ШК/с).

Как видно из таблицы, о существенной потере производительности новой функциональной моделью по сравнению с оригинальной говорить не приходится. Ввиду того что потактовый режим работы симулятора все еще находится в разработке, нет возможности полноценно измерить его производительность. С целью приблизительной оценки его замедления относительно функционального режима было проведено тестирование на простых, вручную написанных тестах, не затрагивающих еще не поддержанных операций. Результаты представлены в табл. 2.

Из таблицы следует, что замедление находится в пределах 10–15 раз. Стоит заметить, что это значение является лишь грубой оценкой, оно может измениться по мере уточнения модели. Также в дальнейшие планы входит исследование возможных оптимизаций симулятора.

Заключение

В статье описан метод, позволивший упростить задачу реализации и поддержки одновременно двух типов симулятора – функционального и потактового. В основе метода лежит построение на базе существующего функционального симулятора нового, способного работать как в функциональном, так и в потактовом режиме. Приведен пример

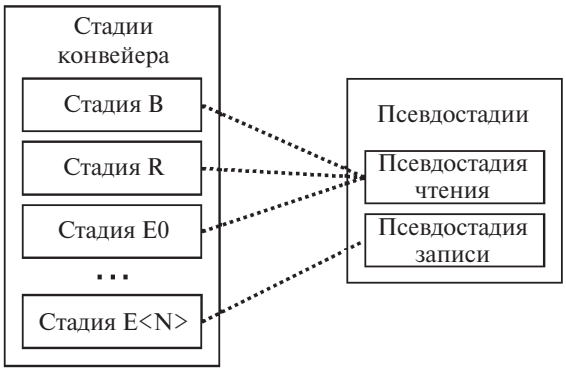

Рисунок 5. Формирование псевдостадий на основе стадийных функций

Таблица 1. Результаты оценки производительности функционального режима работы нового симулятора

Тест	Оригинальный симулятор, 10^6 ШК/с	Функциональный режим нового симулятора, 10^6 ШК/с
boot	2,74	2,80
linux	3,44	3,42
lintel	2,21	2,19
user lintel	3,92	3,82

Таблица 2. Результаты оценки производительности потактового режима работы нового симулятора

Тест	Потактовый режим, 10^6 ШК/с	Функциональный режим, 10^6 ШК/с
add1	0,80	10,53
add3	0,76	7,87
add6	0,63	5,71
ibranch	0,92	14,09

применения данного подхода в процессе разработки нового симулятора микропроцессоров архитектуры «Эльбрус». Полученные экспериментальные данные свидетельствуют о целесообразности выбранного решения.

Дальнейшие работы предполагают уточнение опробованных решений, поддержку большего числа операций, реализацию потактовых моделей устройств. В числе запланированных работ – исследование и реализация возможных оптимизаций.

СПИСОК ЛИТЕРАТУРЫ

1. Архитектура, программное обеспечение и применение компьютеров серии «Эльбрус» / А.К. Ким, В.Ю. Волконский, Ф.А. Груздов и др. // Сборник трудов IV Международной научно-практической конференции «Современные информационные технологии и ИТ-образование», Москва, 2009. С. 53–72.
2. Kassem R., Briday M., Brechennec J.-L., Savaton G., Trinquet Y. Harmless, a hardware architecture description language dedicated to real-time embedded system simulation. Journal of Systems Architecture. North-Holland, 2012, vol. 58 (8), pp. 318–337.
3. Pees S., Hoffmann A., Zivojnovic V., Meyr H. Lisa – machine description language for cycle-accurate models of programmable dsp architectures. In: DAC'99: Proceedings of the 36th ACM/IEEE conference on Design automation. ACM, New York, NY, USA, 1999, pp. 933–938.
4. Qin W. Modeling and description of embedded processors for the development of software tools. PhD thesis, Princeton University, Princeton, NJ, 2004.
5. Strazdins P., Clarke B., Over A. Efficient Cycle-Accurate Simulation of the UltraSPARC III CPU, in CRPITS'07: Proceedings of the Thirtieth Australasian Conference on Computer Science, Jan. 2007, to appear.
6. Nussbaum F.D., Fedorova A., Small C. An overview of the Sam CMT simulator kit, Technical Report TR-2004–133, Sun Microsystems Research Labs, February 2004.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Порошин Павел Алексеевич, аспирант МТУ (МИРЭА); инженер-программист 2-й категории, ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д.24, тел.: 8 (499) 135-70-79, e-mail: poroshin_p@mcst.ru.
Мешков Алексей Николаевич, к.т.н., начальник отдела, АО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д.24, тел: 8 (499) 135-70-79, e-mail: alex@mcst.ru.
Черных Сергей Витальевич, начальник сектора, АО «МЦСТ», 119334, Москва, ул. Вавилова, д.24, тел: 8 (499) 135-70-79, e-mail: chernyh_s@mcst.ru.

For citation: Poroshin P.A., Meshkov A.N., Chernyh S.V. Development of simulator with support of cycle-accurate simulation mode on base of the existing instruction set simulator of the Elbrus architecture. Voprosy radioelektroniki, 2018, no. 2, pp. 69–75.

P.A. Poroshin, A.N. Meshkov, S.V. Chernyh

DEVELOPMENT OF SIMULATOR WITH SUPPORT OF CYCLE-ACCURATE SIMULATION MODE ON BASE OF THE EXISTING INSTRUCTION SET SIMULATOR OF THE ELBRUS ARCHITECTURE

During development of computer systems and accompanying software there is a need in software models of processors (simulators) of various simulation detail and speed. High speed simulation is provided by instruction set simulators (ISS) and high detail simulation is provided by cycle-accurate simulators (CAS). Simultaneous development of ISS and CAS is a time-consuming task. In this paper, we address the problem of reducing the complexity of development and support of simulators of various levels of accuracy for VLIW microprocessors. As a solution we propose development of single simulator that can function both as ISS and as CAS. Using the example of Elbrus simulator development, we describe a method for constructing such multipurpose simulator. This method is mostly based on the gradual extension and refinement of the ISS to the capabilities of CAS and on the semi-automatic simplification of the resulting CAS to the ISS. Performance of the resulting ISS is on par with performance of the original ISS, slowdown of the CAS relative to the ISS is estimated at 10x-15x.

Keywords: Elbrus processor architecture, software emulation, instruction set simulator (ISS), cycle-accurate simulator (CAS).

REFERENCES

1. Kim A.K., Volkonskiy V. Yu., Gruzdov F.A. et al. Architecture, software and application of Elbrus series computers. *Sbornik trudov IV Mezhdunarodnoy nauchno-prakticheskoy konferentsii «Sovremennye informatsionnye tekhnologii i IT-obrazovanie»,* Moscow, 2009. P. 53–72 (In Rissian).
2. Kassem R., Briday M., Brechennec J.-L., Savaton G., Trinquet Y. Harmless, a hardware architecture description language dedicated to real-time embedded system simulation. *Journal of Systems Architecture.* North-Holland, 2012, vol. 58 (8), pp. 318–337.
3. Pees S., Hoffmann A., Zivojnovic V., Meyr H. Lisa – machine description language for cycle-accurate models of programmable dsp architectures. *In: DAC'99: Proceedings of the 36th ACM/IEEE conference on Design automation*. ACM, New York, USA, 1999, pp. 933–938.

4. Qin W. Modeling and description of embedded processors for the development of software tools. *Ph D. thesis,* Princeton University, Princeton, 2004.
5. Strazdins P., Clarke B., Over A. Efficient Cycle-Accurate Simulation of the UltraSPARC III CPU. *In: CRPITS'07: Proceedings of the Thirtieth Australasian Conference on Computer Science,* Jan. 2007.
6. Nussbaum F. D., Fedorova A., Small C. An overview of the Sam CMT simulator kit. *Technical Report TR-2004–133, Sun Microsystems Research Labs,* February 2004.

AUTHORS

Poroshin Pavel, graduate student, MTU (MIREA); software engineer of the 2st category, PJSC Brook INEUM, 24, ulitsa Vavilova, Moscow, 119334, Russian Federation, tel.: +7 (499) 135-70-79, e-mail: poroshin_p@mcst.ru.
Meshkov Aleksey, PhD, head of Department, JSC MCST, PJSC Brook INEUM, 24, ulitsa Vavilova, Moscow, 119334, Russian Federation, tel.: +7 (499) 135-70-79, e-mail: alex@mcst.ru.
Chernyh Sergey, head of Sector, JSC MCST, 24, ulitsa Vavilova, Moscow, 119334, Russian Federation, tel.: +7 (499) 135-70-79, e-mail: chernyh_s@mcst.ru.