Моделирование протоколов семейства AXI в симуляторах производительности вычислительных систем

П. И. Крюков, К. А. Королев, О. И. Ладин

***Аннотация* — Протоколы семейства AXI являются распространённым способом интеграции различных устройств в современных вычислительных системах. Анализ производительности таких систем возможен с использованием программных моделей -- симуляторов производительности. В статье приводится анализ основных принципов разработки таких симуляторов, на основе которого предлагается способ моделирования протоколов семейства AXI. Рассмотренный способ может быть внедрён непосредственно в окружение потактового моделирования, что позволяет обеспечить высокую масштабируемость моделей вычислительных систем. Для описанного решения приведено качественное и количественное исследования влияния на быстродействие моделирования.**

***Ключевые слова* — Asim, моделирование производительности, AXI, симуляторы**

I. Введение

Настоящее время описывается как «бум архитектур»: на смену вычислительным устройствам общего назначения приходят устройства, специализированные под решение конкретных задач [1]. Успех внедрения таких устройств зависит не только от качества их аппаратной архитектуры, но и от качества нового промежуточного программного обеспечения. Задачами такого ПО являются не только поддержка пользовательских задач, но и оптимизация их производительности с учётом специализации архитектуры.

Увеличение производительности вычислительных систем требует решения широкого круга задач, начиная от анализа алгоритмов работы приложений и заканчивая учётом происходящих в системе физических процессов. Для декомпозиции задач между коллективами инженеров общепринято деление изучающих вычислительную технику дисциплин на компьютерные науки и физические науки, такие как радиотехника, схемотехника, микроэлектроника и т. д. Это разделение, в свою очередь, требует общей методологической базы, позволяющей специалистам в разных дисциплинах использовать одни термины, метрики и инструменты. Для этих целей широкое распространение получил подход создания программных симуляторов производительности вычислительных систем [2].

Специализация вычислительных устройств приводит к их объединению в сложно организованные системы. Соответственно, растёт сложность и моделей [3]: помимо установления соответствия в производительности между каждым устройством и его симулятором, важной задачей становится создание общего окружения для интеграции симуляторов разных устройств и моделирование накладных расходов, возникающих при их взаимодействия между ними.

В целях унификации интеграции устройств в вычислительные системы предложено несколько стандартных способов передачи данных, одними из наиболее распространённых являются протоколы семейства Advanced eXtensible Interface (AXI) [4]. Таким образом, задача моделирования может быть дополнена задачей моделирования таких протоколов посредством общего для разных симуляторов окружения, как на функциональном уровне, так и для оценки производительности передачи данных.

Для решения этой задачи в этой статье проводится анализ основных принципов построения окружения моделирования производительности, рассматриваются существующие возможности, которые могут быть использованы для решения задачи, а также рассматривается их влияние на быстродействие и масштабируемость. По результатам анализа предлагается способ дополнения существующих окружений моделирования производительности функциональностью AXI-протоколов и проводится исследование его быстродействия.

II. Основные принципы моделирования производительности

На сегодняшний день создано несколько десятков программных моделей (симуляторов), охватывающих как широкий спектр вычислительных систем, так и разнообразие решаемых задач [2]. Многие модели могут быть использованы для оценки производительности моделируемых систем, в том числе состоящих из большого количества разнообразных устройств.

Успех программных моделей как общей методологической базы основан на нескольких их свойствах. Изначально они абстрагируют большинство деталей физической реализации вычислительной системы от специалистов в области ПО. Далее, ими моделируются свойства системы, которые необходимы для измерения, исследования, анализа и оптимизации производительности. Наконец, такие модели

Статья получена 9 февраля 2025
П. И. Крюков -- ПАО Сбербанк, МФТИ (e-mail: pavel.kryukov@phystech.edu)
К. А. Королёв -- ПАО Сбербанк (e-mail: kirill.korolef@gmail.com)
О. И. Ладин -- ПАО Сбербанк, МФТИ (e-mail: ladin@phystech.edu)

разрабатываются с использованием языков программирования (ЯП), преимущественно объектно-ориентированных, что уменьшает порог вхождения для их использования экспертами в программном обеспечении.

Основной единицей симулятора является модуль — модель какого-либо аппаратного блока вычислительной системы. Использование средств ЯП позволяет применять гибкие настройки иерархии модулей для исследования различных архитектур. Подобное свойство несёт несколько преимуществ. Во-первых, за счёт декомпозиции упрощается разработка, поддержка и валидация исходного кода, открываются возможности использования модульного тестирования [5]. Во-вторых, практичным подходом становится создание семейства моделей, используемых для исследования целого класса схожих вычислительных систем; например, нескольких инкрементально развивающихся поколений.

Каждому блоку вычислительной системы может быть поставлен в соответствие конечный автомат; таким образом, программный код модуля может быть рассмотрен как описание этого конечного автомата [6]. При объектно-ориентированном подходе к программированию переменные объекта описывают состояния конечного автомата, а методы класса -- переходы между состояниями. В зависимости от внутреннего состояния, а также входных воздействий, модуль осуществляет переход в новое состояние и порождает выходные сигналы, являющиеся, в свою очередь, воздействиями на другие модули.

Отметим, что с точки зрения моделируемой системы указанные операции осуществляются мгновенно. Для подсчёта времени необходима дополнительная стадия нотации, в которой фиксируется изменение времени. Если величина изменения времени постоянна, говорят о потактовом моделировании; в событийном моделировании величина изменения времени зависит от состояния модуля. На рис. 1 приведён цикл моделирования со всеми указанными стадиями.

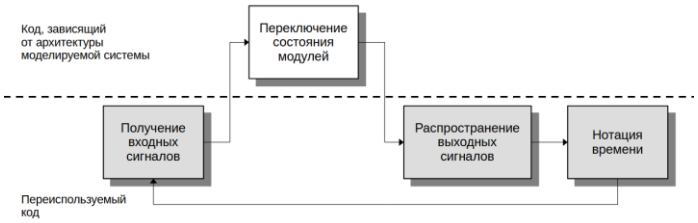

Рис. 1. Стадии моделирования производительности

Поведение стадий, отмеченных на рис. 1 серым, зависит от поведения модуля только параметрически, и их реализация может быть переиспользована. Как правило, общий программный код поставляется в рамках окружения (англ. framework) моделирования, например Sparta [7], Gem5 [8], Asim [9], а код модели, зависящий от архитектуры системы, является по отношению к окружению пользовательским.

Для рассмотрения задачи моделирования протоколов передач данных необходимо рассмотреть моделирование как распространение сигналов, так и нотацию времени их распространения. Окружение предоставляет этот функционал посредством классов-портов. Порт представляет собой абстракцию, позволяющую перейти от низкоуровневого описания регистровых передач через схемотехнические триггеры к исследованиям характеристик производительности системы, таких как латентность и пропускная способность. Рассмотрим функционал портов на примере Asim [9], определим основные свойства и проведём качественный анализ скорости их работы.

Поведение порта параметризуется значениями задержки распространения сигнала (латентности) и количества элементов, записываемых за такт (пропускной способности). Порты имеют два асимметричных интерфейса: передатчик может отправлять сигналы через объект WritePort, а приёмник -- опрашивать о наличии сигналов (IsValid) и считывать их значения через объект ReadPort.

Инициализация каждого модуля содержит описание портов, позволяющего правильно соединить модуль с другими. Можно говорить о том, что модули являются вершинами взвешенного мультиграфа, а порты - его рёбрами с весами, обусловленными значениями латентности и пропускной способности [6]. Необходимо отметить, что граф является ориентированным, передача информации по каждому ребру всегда направлена в одну сторону, от передатчика к приёмнику. Как правило, конвейерные архитектуры предполагают наличие сигналов остановки [10], распространяемых обратно тракту данных; они должны реализовываться, как отдельный порт. На рис. 2 приведён пример представления части модели простого вычислительного ядра в виде графа.

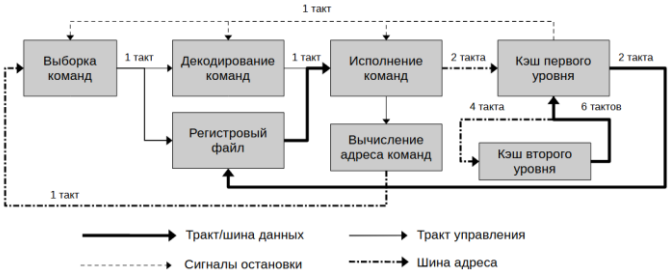

Рис. 2. Пример части модели, представленной в виде ориентированного графа.

Каждая связанная пара портов порождает модель регистровых передач через flip-flop в конвейерном режиме. На уровне программного кода модель может быть реализована посредством общей структуры данных «очередь» с автоматическим использованием стандартных контейнеров языка C++. Запись в порт дополняет данные информацией о номере такта доступности -- момента времени, когда данные могут быть прочитаны, высчитываемого как такт записи плюс задержка порта.

Работа с портом на стороне приёмника осуществляется вызовом метода `bool ReadPort::Read(T& data, UINT64 cycle)`, который проверяет элемент в голове очереди и сравнивает

текущий номер такта с записанным. Если очередь не пуста и записанный номер такта меньше либо равен номеру текущего, то метод перемещает данные из очереди наружу и возвращает истину: таким образом, порт передаёт вместе с данными бит валидности. Данные в этом случае удаляются из очереди. Дополнительно, порт Asim расширяется функционалом останова (stall): при вызове функции ‘ReadStallPort::SetStalled(true);` все номера тактов чтения увеличиваются на единицу, а функционал WritePort блокируется. На рис. 3 приведён пример состояния очереди, запись в которую производилась не каждый такт работы, т. е. с «пузырями» конвейера:

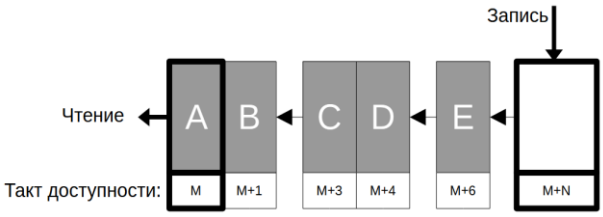

Рис. 3. Работа очереди с порта латентности N в такте M. На тактах M+2 и M+5 данные прочитаны не будут.

Из закона Литлла [11] следует, что при передаче данных с латентностью в N тактов и пропускной способностью B максимальный размер очереди равен N * B элементов. При этом необходимо учесть, что при записи данных ранее их чтения размер очереди увеличивается на B, таким образом максимальный размер очереди составит (N + 1) * B. Верно и обратное: при передаче без простоев данных по очереди из N элементов задержка будет равна N - 1 тактов.

Следует рассмотреть особый случай портов с нулевой латентностью: его работа возможна с очередью размером 1 элемент, однако требуется обеспечить моделирование работы приёмника данных после передатчика. При невозможности такого обеспечения пара модулей должна быть связана прямым переключением состояния конечного автомата одного модуля в процессе моделирования другого. Так как обычно это осуществляется посредством вызова функций одного класса модуля из функций другого, будем называть это способ функциональной передачей данных. Подобная передача нарушает принцип декомпозиции, что снижает масштабируемость, предсказуемость и другие показатели качества модели для конечного пользователя.

III. Моделирование стандартных протоколов передачи данных

Современные вычислительные системы представляют ансамбль множества устройств, таких как вычислительные ядра, устройства прямого обращения к памяти (DMA), контроллеры внешних интерфейсов. В целях унификации процесса интеграции компонент предложено несколько стандартных протоколов передачи данных, одним из наиболее популярных является семейство протоколов Advanced eXtensible Interface (AXI), например, AXI4, AXIStream и AXILite [4]. Рассмотрим подходы к их моделированию с использованием средств, описанных в предыдущей главе.

Протоколы двунаправлены: передатчик, помимо отправки данных и сигнала валидности (Valid), получает от приёмника сигнал готовности (Ready) [4]. Пересылка данных осуществляется только при возникновении «рукопожатия», то есть одновременной активности сигналов Valid и Ready. Для передатчика это означает приём сигнала Ready и валидность собственных данных, а для приёмника - получение сигнала Valid и возможность эти данные обработать.

Рассмотрим модель передачи данных через подобный протокол с 1 буфером: запись данных в начале такта, их чтение в конце текущего и обработка в начале следующего. Как было указано выше, буфер размера 1 возможно моделировать посредством порта нулевой латентности и расположения вызова передатчика ранее приёмника. Однако, это невозможно сделать, так как для передачи симметричного Ready также необходим порт нулевой задержки, требующий другого порядка моделирования приёмника и передатчика.

Физическое расположение реальных устройств может требовать наличия большего количества буферов, что увеличивает латентность, и как следствие, нуждается в моделировании такой функциональности. Однотактовая ретрансляция данных и контрольных сигналов протокола семейства AXI осуществляется посредством регистрового слоя (англ. register slice). Логически регистровый слой может быть представлен как очередь на 2 элемента. Передача данных снаружи в первый элемент, между элементами и из второго элемента наружу осуществляется при одновременной активности соответствующих им сигналов Valid и Ready.

При моделировании передачи данных через последовательность регистровых слоёв необходимо учитывать сигналы Valid и Ready для каждого элемента последовательности, то есть рассматривать их как отдельные конечные автоматы, реализованные как отдельные модули. Так как каждый регистровый слой представляет очередь на 2 элемента, он может содержать порты латентностью в 1 такт. Примем для определённости, что через такие порты осуществляется запись данных с сигналов Valid/Ready наружу, а их чтения -- посредством функционального вызова.

Таким образом, внедрение каждого регистрового слоя представляет собой добавление узла в ориентированном графе модели, и связывающих его 4 ребра (Read Data/Valid, Read Ready, Write Data/Valid, Write Ready), из которых 2 представляют порт. Построение таких модулей может быть автоматизировано путём добавления соответствующих конструкторов в окружение. Соответствующий граф для латентности 4 представлен на рис. 4.

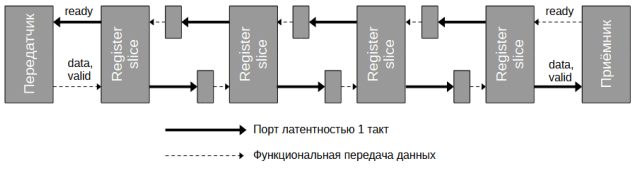

Рис. 4. Модель передачи данных через 4 регистровых

слоя, представленная в виде ориентированного графа.

Представленная реализация имеет низкую масштабируемость из-за наличия функциональных вызовов, требующих упорядочивание моделирования приёмника, передатчика и регистровых слоёв. Для решения этой проблемы нами предлагается интеграция упрощённой версии протокола AXI непосредственно в интерфейсы и реализацию Asim-подобных портов, то есть, внесение в окружение не только инициализации регистровых слоёв, но и модели их поведения. Назовём такой модифицированный класс AXI-портом.

AXI-порт дополняет интерфейсы обычных портов методами `WriteAXIPort::IsReady` и `ReadAXIPort::ResetReady`, использующимися для передачи сигнала Ready. Если метод ResetReady никогда не вызывается, то IsReady всегда возвращает истину, что обеспечивает обратную совместимость.

Рассмотрим поведение при вызове функции ResetReady. В случае, если данные для чтения в текущем такте отсутствуют, состояние очереди не изменяется; это эквивалентно продвижению всех элементов по регистровым слоям на 1 позицию.

Если в голове очереди находится элемент, доступный для чтения, то такт доступности для него при вызове ResetReady увеличивается на единицу; при этом остальные элементы не изменяются. Таким образом, на следующем такте головной элемент может быть либо прочитан ещё раз, либо снова задержан вызовом ResetReady.

Так как каждая блокировка чтения головного элемента не изменяет номера такта доступности последующих элементов, к моменту их продвижения на головную позицию номер текущего такта станет равным или превысит номер такта доступности. В результате этого произойдёт «схлопывание» пузырей, образовавшихся при отсутствии записей в порт. Рассмотрим это на примере.

На рисунке 5 приведён результат однократного вызова функции ResetReady для состояния очереди, приведённого ранее на рисунке 3. Как видно, такт доступности элемента «A» теперь равен M+1. Предположим, что ResetReady приёмником более не вызывается; тогда в такте M+1 будет прочитан «A», в такте M+2 -- «B», так как его такт доступности M+1 > M+2. В последующих тактах будут доступны «C» и «D», таким образом, промежуток между «B» и «D» устраняется. Если предположить, что ResetReady всё же вызвался приёмником ещё раз, то будет устранён и промежуток между «D» и «E».

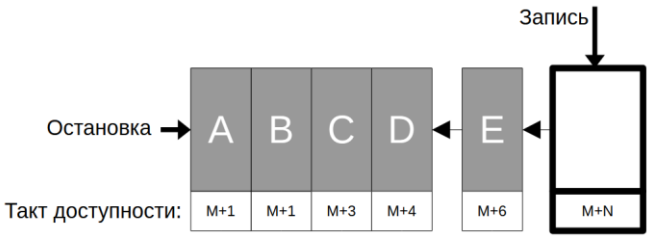

Рис. 5. Результат остановки порта латентности N в такте M. В следующие 4 такта возможно непрерывное чтение данных.

При вызове ResetReady в общем случае не производится остановка записи. Генерация сигнала IsReady для порта записи осуществляется путём измерения размера очереди. Допустим, что очередь содержит X элементов, с учётом элементов, записанных и прочитанных в этом такте. Тогда, если X меньше, чем общее количество данных, помещающихся в регистровые слои, то WriteAXIPort::IsReady возвращает истину. Так как каждый регистровый слой содержит 2 элемента, а общее их число равно N, то условие записи выглядит как X < 2*N.

IV. Анализ скорости моделирования

Практически важным свойством модели является скорость её работы [12], определяемая как отношение моделируемого времени (нотированного) к реальному времени исполнения. Исходя из разбиения процесса на стадии, указанные на рис. 1, выделяются три компоненты скорости моделирования: быстродействие окружения, количество вызовов окружения, т. е. объёма данных, пересекающих архитектурную границу, и быстродействие пользовательского кода, поведение которого в данной работе не рассматривается.

Важным свойством портов является минимальное влияние параметров латентности и пропускной способности на скорость моделирования. От этих параметров зависит только размер очереди, а процедуры записи и чтения имеют алгоритмическую сложность O(1). Далее, порт имеет минимальное взаимодействие с пользовательским кодом: необходимо записать данные в такте M и получить данные в такте N + M.

Реализация модели протокола AXI посредством новых регистровых слоёв уменьшает скорость моделирования по обоим факторам. Во-первых, для моделирования задержки N необходимо внедрение N новых модулей, то есть скорость моделирования зависит от латентности как O(N). Во-вторых, каждый модуль-регистровый слой добавляет собственные порты, увеличивая количество общее передач данных между пользовательским кодом и окружением.

Реализация передачи данных через AXI-порты сохраняет все преимущества портов. Меняет только операции ResetReady и IsReady, алгоритмическая сложность которых, как было показано выше, не зависит от количества регистровых слоёв.

Для количественной проверки утверждений нами было произведено исследование обеих реализаций посредством целевых бенчмарок. В качестве основного элемента тестового сценария использовалась пара «приёмник-передатчик», соединённая статически заданным количеством регистровых слоёв, при этом передатчик создаёт сигнал Valid каждый такт, а приёмник генерирует Ready случайным образом. Была введена параметризация по трём координатам: реализация модели протокола AXI, количество регистровых слоёв между приёмником и передатчиком (латентность), общее количество моделируемых пар одновременно пар. Измерения времени моделирования проводились с использованием окружения Google

Benchmark [13]. Для удобства анализа линейных зависимостей среднее время моделирования было поделено на количество промоделированных тактов, а также на количество пар «приёмник-передатчик».

На рисунке 6 приведено время моделирования одного такта для различного количества пар «приёмник-передатчик», моделируемых одновременно с латентностью 1. В этом случае модуль-регистровый слой является третьим элементом к паре, вследствие чего время моделирования растёт на 55%, от 75 нс до 116 нс, однако, при больших масштабах системы рост становится нелинейным и доходит до 78%.

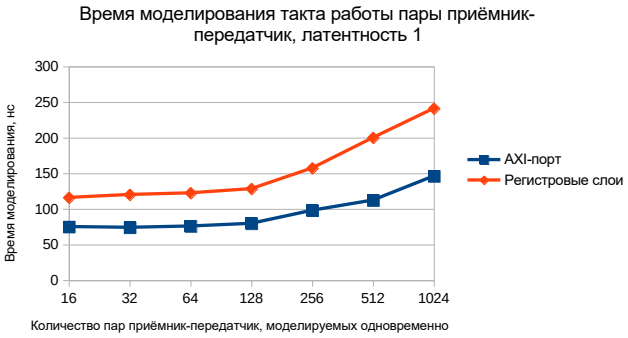

Рис. 6. Накладные расходы на добавление одного регистрового слоя в модель.

При увеличении латентности с использованием регистровых слоёв время моделирования растёт линейно, так как линейно увеличивается количество модулей, связей между ними и переключений контекста между моделью и окружением, что показано на рис. 7. Моделирование с использованием AXI-портов практически не показывает зависимости от латентности, что показано на рис. 8.

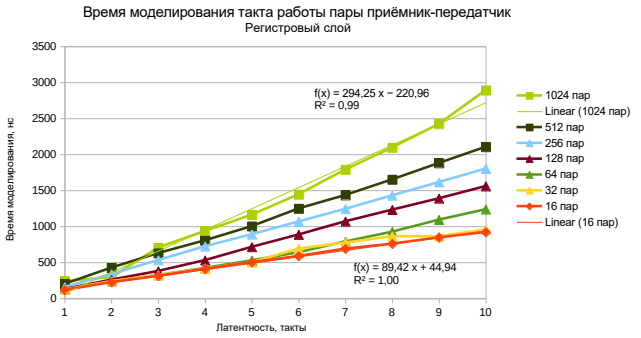

Рис. 7. Рост накладных расходов на моделирование регистровых слоёв при увеличении латентности.

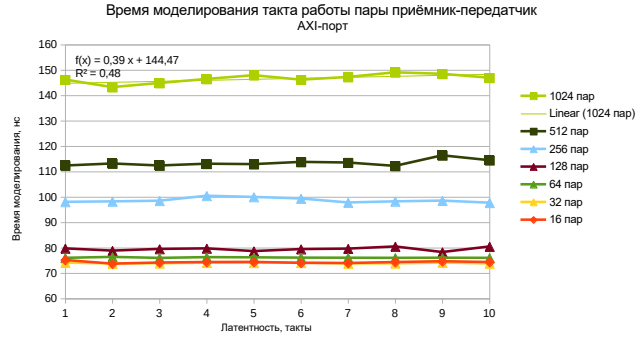

Рис. 8. Изменение накладных расходов на моделирование AXI-портов при увеличении латентности.

V. Заключение

Полученное решение с использованием AXI-портов имеет интерфейс, совместимый с интерфейсом обычных ASIM-портов, и вследствие этого, не требует использования функциональной передачи данных. Подобно тому, как обычные порты абстрагировали регистровые передачи, AXI-порты позволяют скрыть детали имплементации регистровых слоёв, что реализует архитектурный принцип сокрытия деталей и, как следствие, упрощает интеграцию подмоделей в модель системы [14].

Использование AXI-портов положительно повлияло на масштабируемость модели и предоставило новые возможности для её развития. В частности, наличие AXI-портов позволило создать модели генератора и приёмника AXI-сигналов, что позволило внедрить применить модульное тестирование для отдельных компонентов системы [15]. Дополнительно, полученная модель была проверифицирована относительно эталонных имплементаций AXI-протокола согласно методам, предложенным в [16].

Исследование влияния на скорость моделирования показало, что реализация AXI-портов непосредственно в окружении является существенно более экономичной с точки зрения вычислительных ресурсов и быстродействия в сравнении с реализациями, использующими только доступные возможности окружения потактового моделирования. Таким образом, можно говорить о решении задачи моделирования производительности AXI-подобных протоколов путём внедрения общего решения в окружение и увеличения его возможности за счёт покрытия систем, состоящих из большого количества специализированных устройств.

Библиография

[1] J. L. Hennessy and D. A. Patterson “A new golden age for computer architecture” in *Commun. ACM*, 2019, vol. 62, no. 2, pp. 48–60.

[2] A. Akram and L. Sawalha “A Survey of Computer Architecture Simulation Techniques and Tools” in *IEEE Access*, 2019, Vol. 7, pp. 78120-78145

[3] J. C. Hoe, D. Burger, J. Emer et al. “The future of architectural simulation” in *IEEE Micro*. 2010. Vol. 30, no. 3. pp. 8–18.

[4] AMBA AXI and ACE Protocol Specification, Feb. 2013, [online] Available: http://www.arm.com

[5] K. Beck “Test Driven Development: By Example” 1st ed., Addison-Wesley Professional, 2002.

[6] Ю. В. Байда *“Методы разработки и тестирования аппаратных потактовых процессоров на программируемых логических интегральных схемах“*. Диссертация на соискание учёной степени кандидата технических наук. М., 2013

[7] Sparta Modeling Architectural Platform // URL: https://github.com/sparcians/map

[8] J. Lowe-Power, A. M. Ahmad, A. Akram “The gem5 Simulator: Version 20.0+” in *ACM SIGARCH Computer Architecture News*, 2011. Vol. 39(2), pp. 1-7

[9] J. Emer, P. Ahuja, E. Borch et al. “Asim: A performance model framework.” in *IEEE Computer* 2002, vol. 35, no. 2, pp. 68-76

[10] J. L. Hennessy and D. A. Patterson *“Computer Architecture: A Quantative Approach”*. 6th edition. Morgan Kauffman. 2017.

[11] Т. П. Климов. *“Теория массового обслуживания“* М.: Московский государственный университет, 2011.

[12] I. Böhm, B. Franke, N. Tophman. “Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation

instruction set simulator" in *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, 2010. pp. 1-10

[13] Google Benchmark // URL: https://github.com/google/benchmark

[14] *Р. Мартин*. "Чистая архитектура. Искусство разработки программного обеспечения" СПб: Питер, 2024.

[15] П. И. Крюков, О. И., Ладин, К. А. Королев. *"Расширение моделей производительности средствами модульного тестирования"* Наноиндустрия. - 2024. - Т. 17. - №10. - С. 847-851.

[16] П. И. Крюков, А. В. Гаращенко, О. И. Ладин *"Интеграция программных моделей производительности с верификационными стендами"*. Российский форум «Микроэлектроника 2024». 10-я научная конференция «ЭКБ и микроэлектронные модули». Сборник тезисов. - 2024. - С. 121-122

Modeling for AXI protocol in performance simulation

P. I. Kryukov, K. A. Korolev, O. I.Ladin

***Abstract* - Data transferring protocols of AXI family became a common way to integrate various devices into a computing system. A widespread approach of performance analysis of such computing systems is performance modeling using cycle-accurate and event-driven simulators. This article provides a top-down overview of general architecture principles of performance modeling, followed by a proposal to extend existing simulation infrastructure with a novel approach to model protocols of AXI family. The approach can be integrated directly to the framework of cycle-accurate simulation similar to Asim, hiding all the microarchitecture details into a simple yet effective layer of abstraction. As a result, this solution allows to maintain scalability of target models, including application of unit testing, TDD and UVM methodology. Along with the description of the algorithm, the article discusses its impact on simulation speed providing theoretical complexity estimations and actual measurement utilizing a benchmark suite.**

***Keywords* — Asim, performance modeling, AXI, cycle-accurate simulation**

References

[1]J. L. Hennessy and D. A. Patterson “A new golden age for computer architecture” in Commun. ACM, 2019, vol. 62, no. 2, pp. 48–60.
[2]A. Akram and L. Sawalha “A Survey of Computer Architecture Simulation Techniques and Tools” in IEEE Access, 2019, Vol. 7, pp. 78120-78145
[3]J. C. Hoe, D. Burger, J. Emer et al. “The future of architectural simulation” in IEEE Micro. 2010. Vol. 30, no. 3. pp. 8–18.
[4]AMBA AXI and ACE Protocol Specification, Feb. 2013, [online] Available: http://www.arm.com
[5]K. Beck “Test Driven Development: By Example” 1st ed., Addison-Wesley Professional, 2002.
[6]Ju. V. Bajda “Metody razrabotki i testirovanija apparatnyh potaktovyh processorov na programmiruemyh logicheskih integral'nyh shemah“. Dissertacija na soiskanie uchjonoj stepeni kandidata tehnicheskih nauk. M., 2013
[7]Sparta Modeling Architectural Platform // URL: https://github.com/sparcians/map
[8]J. Lowe-Power, A. M. Ahmad, A. Akram “The gem5 Simulator: Version 20.0+” in ACM SIGARCH Computer Architecture News, 2011. Vol. 39(2), pp. 1-7
[9]J. Emer, P. Ahuja, E. Borch et al. “Asim: A performance model framework.” in IEEE Computer 2002, vol. 35, no. 2, pp. 68-76
[10] J. L. Hennessy and D. A. Patterson “Computer Architecture: A Quantative Approach”. 6th edition. Morgan Kauffman. 2017.
[11] T. P. Klimov. “Teorija massovogo obsluzhivanija“ M.: Moskovskij gosudarstvennyj universitet, 2011.
[12] I. Böhm, B. Franke, N. Tophman. “Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation instruction set simulator” in 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2010. pp. 1-10
[13] Google Benchmark // URL: https://github.com/google/benchmark
[14] R. Martin. “Chistaja arhitektura. Iskusstvo razrabotki programmnogo obespechenija” SPb: Piter, 2024.
[15] P. I. Krjukov, O. I., Ladin, K. A. Korolev. “Rasshirenie modelej proizvoditel'nosti sredstvami modul'nogo testirovanija“ Nanoindustrija. - 2024. - T. 17. - #10. - C. 847-851.
[16] P. I. Krjukov, A. V. Garashhenko, O. I. Ladin “Integracija programmnyh modelej proizvoditel'nosti c verifikacionnymi stendami“. Rossijskij forum «Mikrojelektronika 2024». 10-ja nauchnaja konferencija «JeKB i mikrojelektronnye moduli». Sbornik tezisov. - 2024. - S. 121-122