УДК 004.4'22 DOI: 10.14529/cmse140403

ЭМУЛЯТОР PCI EXPRESS ДЛЯ HDL-МОДЕЛИРОВАНИЯ[1]

А.Б. Шворин

В данной работе описывается эмулятор PCI Express — инструмент, позволяющий упростить разработку и отладку некоторого класса аппаратных устройств, работающих по протоколу передачи данных PCI Express. Эмулятор позволяет промоделировать поведение разрабатываемого устройства на обычном компьютере, что значительно сокращает цикл отладки.

Ключевые слова: разработка аппаратного обеспечения, моделирование аппаратуры, эмуляция, PCI Express

Введение

Потребность в инструменте, позволяющем эмулировать работу PCI Express [1], была осознана автором и его коллегами в процессе работы над аппаратной реализацией интерконнекта в рамках проекта СКИФ-Аврора [2, 3] в 2010–2012 гг. Тогда же были сделаны первые наработки, которые в процессе развития выкристаллизовались в отдельный самостоятельный инструмент — эмулятор PCI Express. В дальнейшем в эмулятор была добавлена поддержка PCIe следующего поколения (Gen 3), что позволило с успехом применить его в следующем проекте Паутина [4].

В данной работе рассматривается задача разработки аппаратного дизайна для ПЛИС (программируемая логическая интегральная схема). Здесь ПЛИС может использоваться непосредственно в качестве целевой аппаратной платформы или же как средство быстрого прототипирования с целью дальнейшего переноса дизайна в специализированную микросхему.

Одним из факторов, негативно сказывающимся на продуктивности — общем времени разработки, — является длительный цикл отладки. Например, в проекте Паутина типичное время от внесения изменений в исходный текст дизайна до получения результатов теста составляет десятки минут, которые тратятся в основном на компиляцию и сборку дизайна. Разумным подходом является симуляция схемы в искусственной среде вместо ее загрузки в аппаратную среду (в ПЛИС). Такую возможность предоставляют современные САПР (системы автоматизированного проектирования, в данном случае специализированные для дизайна электронных устройств), в которых обычно ведется разработка, а также сторонние инструменты.

Основной сложностью такого подхода является то, что дизайн, как правило, не может работать сам по себе, он связан с какими-либо периферийными устройствами и требует от них выполнения определенного протокола. В случае загрузки дизайна в ПЛИС в роли периферии выступают устройства, электрически связанные с ножками ПЛИС. Если же происходит симуляция дизайна, то требуется каким-либо образом имитировать или, точнее, эмулировать поведение периферии. В простейших случаях (светодиодная индикация, кнопки, переключатели и др.) это можно делать средствами САПР, иногда вместо внешнего устройства достаточно реализовать простую заглушку и включить ее в основной дизайн, но бывают и гораздо более сложные устройства, так что их эмуляция представляет собой нетривиальную задачу.

[1]Статья рекомендована к публикации программным комитетом Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров — 2014».

В данной статье описывается способ реализации одного из ключевых компонентов модельной среды — виртуальной реализации интерфейса PCI Express. Этот эмулятор обладает достаточной гибкостью и поддерживает запуск стороннего программного обеспечения, работающего с устройством через PCI Express.

В разделе 1 кратко описана одна из реализаций интерфейса PCI Express — та, с которой работет эмулятор. Раздел 2 содержит общие идеи построения эмулятора в рамках клиент-серверной модели. Более подробное описание схемы работы серверной части эмулятора изложено в разделе 3. Там же показан механизм интеграции сервера с с пользовательским аппаратным дизайном. Наконец, в заключении статьи обозначена область применимости эмулятора и даны оценки его эффективности.

1. Интерфейс PCI Express

Интерфейс PCI Express (PCIe) является стандартом высокоскоростной передачи данных между различными устройствами компьютера, как правило, объединенными на одной плате. Для PCIe существуют также кабельные соединения. PCIe подразумевает наличие пакетного протокола адресной передачи данных, в котором имеется несколько типов пакетов, наиболее важные из которых перечислены ниже:

- запрос на запись по заданному адресу;
- запрос на чтение по заданному адресу;
- ответ на чтение.

Именно эти типы пакетов поддержаны в эмуляторе. Существуют также служебные с точки зрения протокола PCIe пакеты, но их поддержка в эмуляторе осталась невостребованной.

Поскольку проект Паутина подразумевал использование ПЛИС фирмы Altera, в качестве реализации интерфейса PCIe был взят предоставляемый этой фирмой протокол Avalon ST [5]. Таким образом, для эмулятора потребовалась программная реализация Avalon ST, что и было сделано. Разумеется, выбор конкретной реализации протокола сужает область применимости эмулятора, однако сравнительно легко добавляются новые протоколы на базе существующей библиотеки эмулятора.

На данный момент в эмуляторе реализованы следующие варианты протокола:

- 64-битный Avalon-ST;
- 128-битный Avalon-ST;
- 256-битный Avalon-ST без мультипакетного режима;
- 256-битный Avalon-ST в мультипакетном режиме.

Здесь разрядность означает количество данных, которое передается за один такт. Различные реализации PCIe могут иметь разную разрядность и работать и на разных частотах, что определяется поколением PCIe и количеством трансиверов. 256-битный вариант допускает так называемый мультипакетный режим (multiple packets per cycle), при котором на одном такте может передаваться хвост предыдущего пакета и начало следующего, что дает заметный выигрыш в пропускной способности по сравнению с обычным режимом.

Рассмотрим для примера 128-битный вариант Avalon-ST. Его HDL-интерфейс представлен на рис. 1.

Основными сигналами являются:

- rx_st_data — передаваемый поток данных, в данном случае разрядностью 128 бит;
- rx_st_sop, rx_st_eop — начало и конец пакета, соответственно;
- rx_st_valid — признак валидности данных на текущем такте.

```
entity ast128 is
    port (
        rx_st_sop   : in  std_logic;                        -- startofpacket
        rx_st_eop   : in  std_logic;                        -- endofpacket
        rx_st_err   : in  std_logic;                        -- error
        rx_st_valid : in  std_logic;                        -- valid
        rx_st_empty : in  std_logic;                        -- empty
        rx_st_ready : out std_logic;                        -- ready
        rx_st_data  : in  std_logic_vector(127 downto 0);   -- data
        rx_st_bar   : in  std_logic_vector(7 downto 0);     -- rx_st_bar
        ...
    );
end ast128;
```

Рис. 1. Интерфейс 128-битного варианта Avalon-ST

- rx_st_ready — сигнал готовности приемного устройства (в отличие от других сигналов, он является выходным для устройства).

Здесь представлена приемная (RX) часть интерфейса; передающая (TX) почти в точности такая же, но направления сигналов противоположны.

2. Принципы работы эмулятора PCI Express

Передача данных через эмулируемый PCIe осуществляется по клиент-серверной схеме, где роль сервера выполняет запущенный процесс собственно эмулятора, а клиентами являются обычные программы, работающие с устройством через PCIe.

Коммуникации между клиентом и сервером осуществляются двояко: через сокетный интерфейс Беркли [6] и посредством специального сегмента системной памяти, общего для клиента и для сервера. Сокетный интерфейс используется для инициализации клиента, а также в процессе работы для передачи PCI-запросов от клиента к серверу. PCI-запросы в обратном направлении (от сервера к клиенту) поступают напрямую в общий сегмент памяти, минуя сокетный механизм.

Технически клиентская часть эмулятора представляет собой библиотеку, с которой необходимо слинковать программу. Эта библиотека имеет интерфейс, названный mmdev, который представляет собой замену некоторым системным вызовам:

- mmdev_open() и mmdev_mmap() служат для инициализации клиента и дают ему доступ к общему сегменту системной памяти и к сегменту виртуальной памяти, связанному с устройством;
- mmdev_close() и mmdev_munmap() используются клиентом для завершения работы;
- mmdev_memcpy() заменяет стандартную функцию копирования памяти при обращении к памяти устройства, она генерирует PCI-запросы на чтение и запись.

Все перечисленные функции интерфейса mmdev имеют в точности те же прототипы, что и соответствующие им системные вызовы.

Для использования эмулятора обычная программа должна быть модифицирована следующим образом:

1. При запуске необходимо вызвать процедуру инициализации, которая обеспечит установку сокетного соединения с сервером.

2. Все операции доступа в сегмент памяти, связанный с устройством, должны быть заменены на специальный вызовы: вместо стандартного вызова memcpy() должен использоваться mmdev_memcpy(), а операция разыменования указателя должна быть выражена через него.

Здесь стоит отметить, что требование (2) является одной из основных трудностей на пути к использованию данного эмулятора. К сожалению, других способов перехвата обращений программы в память, по-видимому, нет. Например, рассматривалась идея использования механизма pagefault, но средства, предоставляемые операционной системой, оказались недостаточными для наших целей. С другой стороны, реализация библиотеки mmdev обеспечивает некоторую безопасность: если произошло прямое обращение по указателю в память эмулируемого устройства, то гарантированно произойдет segmentation fault, что позволит быстро локализовать ошибку и исправить ее путем замены обращения по указателю на вызов mmdev_memcpy().

Для удобства пользователя библиотека mmdev представлена в двух реализациях. Одна реализация, описанная выше, служит для работы с эмулятором. Другая — является тривиальной прослойкой между интерфейсом mmdev и соответствующими системными вызовами; она предназначена для работы с настоящим (не эмулируемым) устройством. Соответственно, есть разные способы использования библиотеки: слинковать пользовательскую программу с одной из реализаций статически или же сразу с обеими. Во втором случае выбор между реализациями будет осуществляться при запуске программы динамически, с помощью переменной среды.

3. Устройство сервера эмулятора

Двоякая реализация клиентской библиотеки обеспечивает выполнение важного принципа моделирования: виртуальная версия, предназначенная для симуляции, должна как можно меньше отличаться от реальной, которая загружается в ПЛИС. Реализация серверной части также следует этому принципу. А именно, подразумевается, что в дизайне устройства выделен модуль, интерфейс которого в точности совпадает с одной из версий представленного выше Avalon ST. Этот модуль-приложение, назовем его app, по сути является объектом тестирования, отладки и дальнейшей разработки. Все остальное — относительно неизменная обвязка, позволяющая проводить симуляцию и создавать прошивку для ПЛИС, то есть модуль допускает два способа использования.

Первый способ — это помещение модуля в «настоящий» дизайн верхнего уровня (toplevel design), предназначенный для загрузки в ПЛИС. В этом дизайне, разумеется, должен использоваться аппаратный модуль — адаптер PCIe из имеющихся в ПЛИС.

На рис. 2 показана работа некоторой пользовательской программы pgm (которая, вообще говоря, может быть запущена в виде не одного процесса, а нескольких) в аппаратном окружении, то есть когда в системе имеется ПЛИС с загруженным в нее дизайном. При этом обмен данными между процессами и устройствами полностью идет через PCIe.

Второй способ, симуляция — подразумевает агрегацию модуля в специальный дизайн верхнего уровня (toplevel design) с пустым интерфейсом (см. рис. 3). То, что интерфейс дизайна пустой, означает, что у него нет видимой периферии, и он не связан с внешней средой. Именно это и нужно для симуляции.

На рис. 4 представлена схема работы программы в режиме эмуляции. Для проведения симуляции должны быть запущены сервер (обозначен на схеме как процесс emu-server)

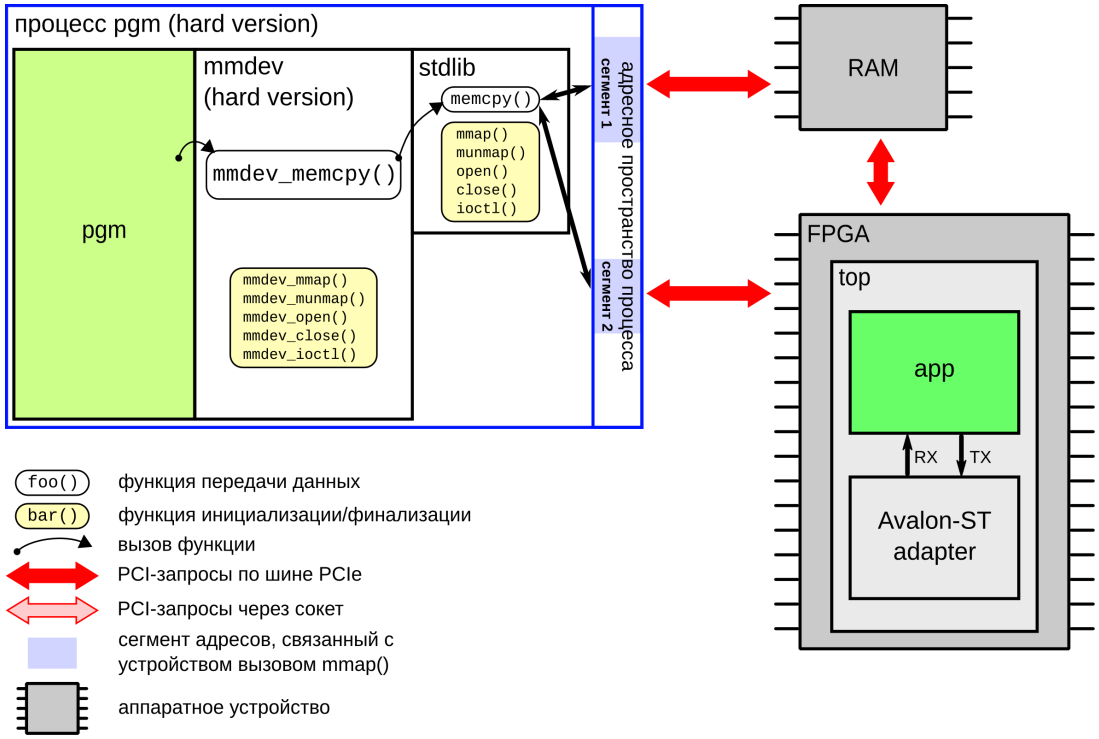

Рис. 2. Схема работы программы pgm с устройством app в обычном (аппаратном) режиме

```
entity emu_top is
end emu_top;
```

Рис. 3. Пустой HDL-интерфейс

и пользовательская программа (представлена процессом pgm), которая подсоединяется к серверу посредством сокетного интерфейса. Вообще говоря, пользовательских процессов может быть несколько, и все они будут клиентами сервера-эмулятора.

На этот раз обмен данными между процессорами и устройствами осуществляется иначе: доступ к памяти по-прежнему происходит через системную шину PCIe, а к эмулируемому устройству — через сокет. Таким образом реализуется вышеупомянутый принцип моделирования: при обоих режимах использования модуль-приложение app и пользовательская программа pgm остаются одними и теми же. Все изменения, вносимые в тестируемый модуль и в пользовательскую программу, в равной степени отражаются и при симуляции, и при запуске «в железе».

Для реализации симуляционной версии дизайна верхнего уровня был выбран GHDL [7] — открытая реализация компилятора VHDL, предназначенная только для симуляции. Важной особенностью GHDL, которая существенно используется в реализации эмулятора, является возможность привязки к методам, написанным на языке Си.

Реализация дизайна верхнего уровня с пустым интерфейсом emu_top является частью сервера. Ее устройство показано на рис. 5. Здесь ключевой особенностью является использование внешних (foreign) процедур, которые реализованы не на VHDL, а на Си. Они вызываются на каждом такте. В качестве параметров им передаются специальным образом запакованные сигналы Avalon ST: процедура line128_rx() обрабатывает приходящий в устройство поток (сигналы ast_rx_*), процедура line128_tx() обрабатывает, соответственно, исходящий поток (сигналы ast_tx_*). Параметры пакуются и передаются согласно спецификации

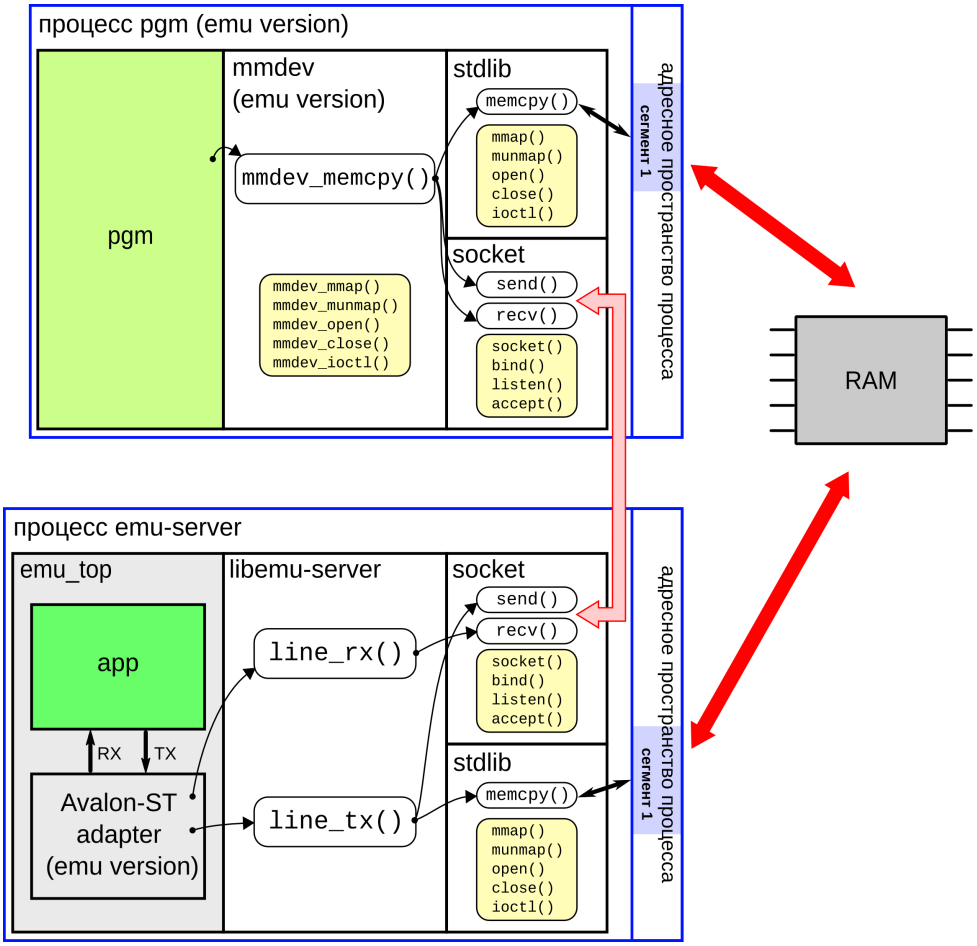

Рис. 4. Схема работы программы pgm с устройством app в режиме симуляции

GHDL [8]. Преобразование сигналов получается довольно громоздким, так что в приведенном листинге оно скрыто отточиями. Модуль-приложение, инстанциированный здесь как VHDL-компонент app, принимает поток PCI-запросов через сигналы ast_rx_* и выдает через ast_tx_*.

Реализация процедуры line128_tx() довольно проста: она в течение нескольких вызовов (один вызов за такт) аккумулирует части PCI-пакета и, когда наберется целый пакет, формирует вызов системной функции копирования memcpy(), имеющий следующий прототип:

```
void *memcpy(void *dest, const void *src, size_t n);
```

Целевой адрес dest и размер сообщения n для этого вызова получаются декодированием заголовка принятого PCI-пакета, а данные извлекаются из тела пакета. Нормальное поведение устройства подразумевает, что все данные направляются в общий сегмент памяти, где могут быть прочитаны клиентами. Для предотвращения опасного поведения, вызванного некорректной работой тестируемого устройства, желательно включить проверку целевого адреса.

В обратную сторону PCI-запросы передаются следующим образом. Вызов mmdev_memcpy() на клиенте передает свои аргументы на сервер через сокет. Там формируется PCI-пакет и помещается в специальную очередь, доступ к которой имеет процедура line128_rx(). Она, как было сказано выше, вызывается из VHDL-дизайна

```
architecture emu_top128 of emu_top is
    procedure line128_tx(...) is ...
    attribute foreign of line128_tx : procedure is "VHPIDIRECT line128_tx";

    procedure line128_rx(...) is ...
    attribute foreign of line128_rx : procedure is "VHPIDIRECT line128_rx";
    ...
begin
    process (clk, reset)
    begin
        if rising_edge(clk) then
            line128_rx(...);
            line128_tx(...);
        end if;
    end process;

    app : ast128
        port map (
            clk   => clk,
            reset => reset,
            ast_rx_... => ...,
            ast_tx_... => ...,
        );
    ...
end emu_top128;
```

Рис. 5. Реализация дизайна верхнего уровня 128-битного варианта Avalon-ST

верхнего уровня на каждом такте, и ее работа заключается в том, чтобы передать через свои параметры очередную часть PCI-пакета согласно протоколу Avalon ST.

Заключение

В режиме эмуляции цикл отладки очень короткий. Сборка занимает буквально несколько секунд — примерно столько же, как у среднего размера программы на языке Си, что несравнимо быстрее сборки прошивки для ПЛИС, которая обычно составляет порядка получаса. Скорость исполнения теста (симуляции) существенно зависит от сложности дизайна, и она, разумеется, на несколько порядков хуже, чем у аппаратной реализации. На практике редко требуется провести симуляцию больше чем на несколько тысяч тактов, что занимает от нескольких секунд до нескольких минут времени.

Таким образом, основное преимуществе симуляции — это, прежде всего, короткий цикл отладки. Также очевидное удобство в получении отладочной информации: работает отладочная печать VHDL, и можно сохранить и потом изучить всю трассу значений сигналов, в то время как для аппаратной реализации средства сохранения трассы весьма ограничены. На одной машине может быть запущено несколько независимых экземпляров сервера, к каждому из которых может присоединиться множество клиентов. Наконец, в режиме симуляции нет опасности привести машину, на которой проводится тестирование, в нерабочее состояние из-за генерации некорректных обращений в PCIe.

Следует также перечислить основные недостатки данного инструмента:

- необходимость имитации периферийных устройств помимо PCI Express;
- неполная эквивалентность реализации языка VHDL в компиляторе GHDL и САПР Quartus фирмы Altera, а также, вероятно, в САПР Xilinx ISE, что может потребовать небольшой модификации кода;
- отсутствие поддержки языка Verilog;
- неполная (на текущий момент) реализация протокола PCIe.

Работа над эмулятором продолжается в настоящее время. Его исходные коды доступны под свободной лицензией [9]. По мере необходимости планируется добавить поддержку различных версий протокола Avalon ST, а также обеспечить более полное покрытие функционала PCIe. Автор выражают надежду, что данный инструмент окажется удобным и достаточно универсальным и сможет найти новых пользователей.

Работа выполнена при поддержке Программы фундаментальных исследований Президиума РАН № 18.

Литература

1. IP Compiler for PCI Express User Guide URL: `http://www.altera.com/literature/ug/ug_pci_express.pdf` (дата обращения: 26.08.2014)

2. Абрамов, С.М. Возможности суперкомпьютеров «СКИФ» ряда 4 по аппаратной поддержке в ПЛИС различных моделей параллельных вычислений / С.М. Абрамов, С.А. Дбар, А.В. Климов, Ю.А. Климов, А.О. Лацис, А.А. Московский, А.Ю. Орлов, А.Б. Шворин // Суперкомпьютерные технологии: разработка, программирование, применение (СКТ-2010): Материалы международной научно-технической конференции (Дивноморское, 27 сентября – 2 октября 2010). — Таганрог: Изд-во ТТИ ЮФУ, 2010. — Том 1. — С. 11–21.

3. Абрамов, С.М. Суперкомпьютеры «СКИФ» ряда 4 / С.М. Абрамов, В.Ф. Заднепровский, Е.П. Лилитко // Информационные технологии и вычислительные системы. — 2012. — № 1. — С. 3–16.

4. Абрамов, С.М. О разработке интерконнекта на активных оптоволоконных кабелях и программируемых логических интегральных схемах / С.М. Абрамов, И.А. Адамович, С.А. Блохин, А.В. Елистратов, Л.Я. Карачинский, Ю.А. Климов, И.И. Новиков, А.Ю. Пономарев, С.С. Ранцев, И.А. Фохт, А.Ю. Хренов, А.Б. Шворин, Ю.В. Шевчук // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции (Новороссийск, 23–28 сентября 2013). — М.: Изд-во МГУ, 2013. — С. 220–223.

5. Avalon Interface Specifications URL: `http://www.altera.com/literature/manual/mnl_avalon_spec.pdf` (дата обращения: 26.08.2014)

6. BSD Sockets Interface Programmer's Guide URL: `http://www.cs.put.poznan.pl/wswitala/download/pdf/B2355-90136.pdf` (дата обращения: 26.08.2014)

7. GHDL guide URL: `http://ghdl.free.fr` (дата обращения: 26.08.2014)

8. GHDL Restrictions on foreign declarations URL: `http://ghdl.free.fr/ghdl/Restrictions-on-foreign-declarations.html` (дата обращения: 26.08.2014)

9. Репозиторий исходных кодов эмулятора PCI Express URL: `https://github.com/shvorin/pcie-emu` (дата обращения: 26.08.2014)

Шворин Артем Борисович, инженер-программист Института программных систем им. А.К. Айламазяна Российской академии наук (Переславль-Залесский, Российская Федерация), art@shvorin.net.

Поступила в редакцию 25 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 4, pp. 51–60

DOI: 10.14529/cmse140403

PCI EXPRESS EMULATOR FOR HARDWARE DESIGN MODELLING

A.B. Shvorin, Program Systems Institute of RAS (Pereslavl-Zalessky, Russian Federation)

This paper describes PCI Express emulator. This tool is aimed to simplify development and debugging of certain class hardware devices using PCI Express protocol. The emulator is capable to simulate a device under development by a conventional computer. It significantly reduces debug time.

Keywords: hardware design, hardware simulation, emulation, PCI Express

References

1. IP Compiler for PCI Express User Guide URL: `http://www.altera.com/literature/ug/ug_pci_express.pdf` (accessed: 26.08.2014)
2. Abramov S.M., Dbar S.A., Klimov A.V., Klimov Yu.A., Lacis A.O., Moskovskij A.A., Orlov A.Yu., Shvorin A.B. Vozmozhnosti superkomp'yuterov "SKIF" ryada 4 po apparatnoj podderzhke v PLIS razlichnyx mod elej parallel'nyx vychislenij [The Capability of "SKIF" Grade 4 Supercomputers to Hardware Support of Various Parallel Evaluation Models in FPGA] // Superkomp'yuternye texnologii: razrabotka, programmirovanie, primenenie (SKT-2010): Materialy mezhdunarodnoj nauchno-texnicheskoj konferencii (Divnomorskoe, 27 sentyabrya – 2 oktyabrya 2010) [Supercomputing Technology: Development, Programming, Applying: Proceedings of the International Scientific and Technical Conference (Divnomorskoe, Russia, September, 27 – October, 2, 2010)]. Taganrog, Publishing of Taganrog Technology Institute of the South Ural State University. 2010. Vol. 1. P. 11–21.
3. Abramov S.M., Zadneprovskij V.F., Lilitko E.P. Superkomp'yutery "SKIF" ryada 4 ["SKIF" grade 4 Supercomputers] // Informacionnye texnologii i vychislitel'nye sistemy [Information Technology and Computer Systems]. 2012. No 1. P. 3–16.
4. Abramov S.M., Adamovich I.A., Bloxin S.A., Elistratov A.V., Karachinskij L.Ya., Klimov Yu.A., Novikov I.I., Ponomarev A.Yu., Rancev S.S., Foxt I.A., Xrenov A.Yu.,

Shvorin A.B., Shevchuk Yu.V. O razrabotke interkonnekta na aktivnyx optovolokonnyx kabelyax i programmiruemyx logicheskix integral'nyx sxemax [The Development of Interconnect Based on Active Optical Cables and FPGA] // Nauchnyj servis v seti Internet: vse grani parallelizma: Trudy Mezhdunarodnoj superkomp'yuternoj konferencii (Novorossijsk, 23–28 sentyabrya 2013) [Scientific Research in the Internet: All the Faces of Parallelizm: Proceedings of the International Supercomputer Conference (Novorossiysk, Russia, September, 23–28, 2013)]. Moscow, Publishing of MSU. 2013. P. 220–223.

5. Avalon Interface Specifications URL: `http://www.altera.com/literature/manual/mnl_avalon_spec.pdf` (accessed: 26.08.2014)
6. BSD Sockets Interface Programmer's Guide URL: `http://www.cs.put.poznan.pl/wswitala/download/pdf/B2355-90136.pdf` (accessed: 26.08.2014)
7. GHDL guide URL: `http://ghdl.free.fr` (accessed: 26.08.2014)
8. GHDL Restrictions on foreign declarations URL: `http://ghdl.free.fr/ghdl/Restrictions-on-foreign-declarations.html` (accessed: 26.08.2014)
9. PCI Express emulator source code repository URL: `https://github.com/shvorin/pcie-emu` (accessed: 26.08.2014)

Received August 25, 2014.