УДК 004

Разработка потактовой поведенческой модели системы на кристалле на языке C++

А.Б. Слепов

Учреждение Российской академии наук Научно-исследовательский институт системных исследований РАН, sir-lexa@yandex.ru

***Аннотация* — Рассматриваются проблемы разработки потактовых поведенческих моделей микропроцессорных систем и «систем на кристалле» на языке C++ и пути их решения. Оцениваются преимущества и недостатки потактовых моделей перед чисто поведенческими, а также рассматривается использование потактовых моделей для верификации RTL-моделей.**

***Ключевые слова* — поведенческое моделирование, микропроцессорные системы, СнК.**

I. Введение

Процесс проектирования «систем на кристалле» (СнК) состоит из ряда этапов: разработки спецификации и эталонной поведенческой модели, разработки модели уровня регистровых передач (RTL) на одном из языков проектирования аппаратуры, верификации, получения списка соединений (netlist) и синтеза топологии [1]. Процесс верификации является достаточно важным этапом разработки, т.к. от качества проведённой верификации зависит количество ошибок в итоговой реализации СнК.

Способы верификации различны. Возможна верификация средствами языка проектирования аппаратуры (например, использование SystemVerilog Assertions при создании проекта на языке SystemVerilog), либо с применением языка программирования высокого уровня (например, использование функций, написанных на языке C, при разработке проекта на Verilog с помощью интерфейса VPI). Другой способ – верификация с помощью потактовой поведенческой модели на языке C++.

Поведенческие модели на языке C++ создаются для решения различных задач – в качестве эталонных моделей для разработки RTL-моделей, для верификации, а также для написания и отладки тестов и другого программного обеспечения ещё до выхода с конвейера готовой СнК [5]. Важную роль играет степень детализации модели. Так, для создания поведенческой модели микропроцессора, с целью написания тестовых программ и другого ПО, не обязательно подробное моделирование конвейера – достаточно провести декодирование и выполнение инструкций в том порядке, в котором они будут выполняться в реальном микропроцессоре. Напротив, для верификации нужна более детализированная модель с потактовым моделированием многоступенчатого конвейера.

В данной работе идёт речь о поведенческих и потактовых поведенческих моделях. Для краткости будем называть потактовые поведенческие модели просто потактовыми. Применение потактовой модели на языке C++ для верификации RTL-модели «системы на кристалле» рассматривается подробно.

Выбор C++ как языка создания поведенческих моделей обусловлен тем, что это один из наиболее распространённых языков программирования, чрезвычайно мощный, универсальный, поддерживающий различные стили и техники программирования. Поведенческие модели, написанные на C++, обладают высоким быстродействием. Возможно создание не только самой модели требуемого устройства, но и пользовательского интерфейса (текстового или графического) для работы с ней. Основным требованием к разработчику поведенческой модели является знание основ объектно-ориентированного программирования.

Разработка модели на языке C++ связана с некоторыми трудностями по причине отсутствия специализированных библиотек системного уровня для создания поведенческих моделей цифровых устройств. Эти трудности могут быть решены использованием специализированной библиотеки SystemC, разрабатываемой инициативной группой Open SystemC Initiative. Однако программирование с использованием библиотеки SystemC обладает и некоторыми недостатками – используемые конструкции громоздки и при больших объёмах кода производительность модели, написанной с использованием SystemC, меньше, чем модели, написанной на чистом C++. Больше об использовании SystemC при моделировании СнК вы можете прочитать в работе [4], а в этой работе речь пойдёт о решении трудностей при написании модели на C++.

В основе данной работы лежит опыт разработки потактовой поведенческой модели системы на кристалле К128, разрабатываемой в НИИСИ РАН. При разработке поведенческой модели К128, в свою очередь,

учитывался предыдущий опыт разработки потактовых поведенческих моделей в НИИСИ РАН [3].

II. Разработка потактовой модели

Рассмотрим типовую систему на кристалле (рис.1). Микропроцессор имеет возможность работать с внешней и накристальной памятью. Имеются стандартные интерфейсы для внешних устройств – UART, I2C, SPI. Для высокоскоростного обмена данными с внешними устройствами используется контроллер прямого доступа к памяти (DMA) и шина PCI.

Структура потактовой модели представляет собой вложенные функциональные блоки-классы разного уровня иерархии. Самый верхний уровень – тестовое окружение (testbench), на нём располагаются моделируемая СнК, ОЗУ, терминал. В составе СнК – управляющий процессор, контроллер DMA, накристальная память, контроллеры периферийных интерфейсов. В составе управляющего процессора – блоки регистров, кэш, блок трансляции адреса и другие блоки.

Наиболее важным элементом системы на кристалле является микропроцессорное ядро. Первой возникающей проблемой при моделировании его поведения является одновременный доступ к регистрам из нескольких участков кода, поскольку в течение одного такта регистры управления и регистры общего назначения могут неоднократно использоваться для чтения и записи. При этом, при чтении должно возвращаться старое значение регистра, даже если в текущем такте в регистр была произведена запись. Решением этой проблемы является разделение моделирования такта на два этапа. Первый этап – основной, в нём производится большая часть работы. Второй этап – корректировочный, в нём происходит обновление состояния регистров и других необходимых переменных.

Моделирование поведения контроллеров шин передачи данных осуществляется таким же способом. Однако, следует обратить внимание на то, что для исключения зависимости результата (здесь имеется в виду потактовая точность) от порядка моделирования поведения блоков следует сначала выполнять основные этапы моделирования всех устройств, затем корректирующие этапы всех устройств.

Следующая сложная ситуация – наличие в моделируемой системе устройств, работающих на разной частоте. Конечно, решить этот вопрос можно введением внутренних счётчиков, проверяющих условие наступления нужного такта, для каждого устройства, работающего на меньшей в N раз частоте. Однако, более красивым решением является создание блока-планировщика, задачей которого будет обеспечение моделирования такта устройства через заданные периоды времени. Он же ведёт учёт времени, прошедшего с начала моделирования. Автор рекомендует использовать целые числа для хранения переменных времени и выполнения с ними арифметических операций, т.к. операции с целыми числами всегда точные и, в абсолютном большинстве случаев, 64-битных целых хватает для выполнения требуемых задач. Для создания планировщика разработчик должен уметь работать с указателями на функции. Планировщик производит в заданные моменты времени вызов функций через указатели на них.

Ряд описанных выше ситуаций решается использованием библиотеки SystemC, однако следует определиться со степенью её использования. Если Вы решили использовать SystemC, совсем необязательно, что-

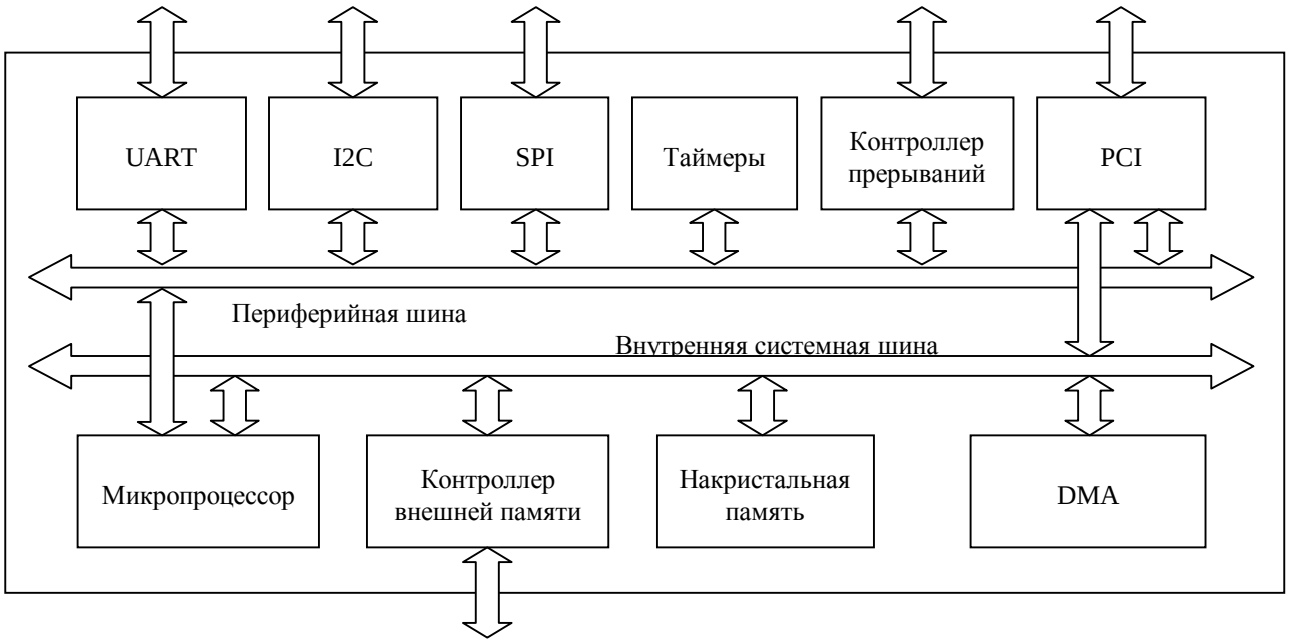

Рис. 1. Пример системы на кристалле

бы проектируемый блок был целиком описан с помощью средств этой библиотеки. В настоящее время SystemC часто используется при описании моделей уровня транзакций (Transaction Level Modeling), однако модели микропроцессорных ядер обычно разрабатываются на C++ [2].

Использование потактовой модели, написанной на C++, для верификации RTL-модели может помочь обнаружить ряд ошибок, поскольку модель на C++ и RTL-модель пишут разные люди, и вероятность существования ошибки в одном и том же месте в моделях существенно ниже. Основным блоком, нуждающимся в верификации, является ядро микропроцессора. Хорошим способом верификации является сравнение трасс счётчика команд и последовательностей изменений регистров, полученных при прогоне тестовой задачи на двух моделях. Трасса счётчика команд представляет собой файл, содержащий строчки с описаниями пройденных линейных участков кода: начало и конец участка, а также количество проходов этого участка. Файл с последовательностью изменений регистров содержит строчки с описаниями записей в регистры – номер регистра и записанные данные – в том порядке, в котором они были записаны. Сравнение файлов, полученных на двух моделях, выполняется с помощью скриптов, написанных на одном из языков подготовки сценариев (scripting language), либо с помощью утилиты diff.

Популярным и эффективным является моделирование с прогоном случайных тестов. Метод заключается в том, что тестовая задача содержит случайно сгенерированную (но с заданными ограничениями) последовательность команд. При прохождении таких последовательностей возможно выявление сложных ошибочных ситуаций, которые невозможно выявить с помощью тестов, разработанных вручную. Данный метод требует наличия эталонной модели, функцию которой и выполняет потактовая поведенческая модель.

III. Разработка поведенческой модели

Для создания поведенческой модели требуется намного меньше усилий и временных затрат, чем для потактовой. Во-первых, как уже было сказано, для моделирования поведения ядра микропроцессора необязательно моделировать каждую стадию конвейера в отдельности – достаточно смоделировать результат выполнения инструкции.

Во-вторых, в моделировании поведения контроллеров шин передачи данных нет необходимости. Если процессор должен записать слово данных в ОЗУ, он обращается напрямую к ОЗУ, и производит запись.

Чисто поведенческая модель также обладает одной особенностью – поскольку доступ к ОЗУ осуществляется мгновенно, численные показатели производительности, получаемые на тестовых задачах, будут отличаться от ожидаемых в реальном микропроцессоре. Таким образом, невозможно провести оценку производительности микропроцессора при выполнении используемых алгоритмов.

Если требуется, чтобы численные показатели при выполнении тестовой программы на поведенческой модели совпадали с ожидаемыми, необходимо уделить особое внимание моделированию задержек при передаче данных. Для этого целесообразно создание отдельного блока-арбитра, который будет являться связующим звеном между отправителем данных и их получателем. Например, если процессор хочет считать слово данных из ОЗУ, он обращается к блоку-арбитру. Блок-арбитр через заданное время задержки передаёт запрос в блок ОЗУ и получает ответ, который через заданное время задержки поступает в микропроцессор. В случае если сразу несколько устройств претендуют на доступ к ОЗУ, блок-арбитр также сам оценивает ситуацию и распределяет задержки.

IV. Выбор уровня детализации

Выбор уровня детализации модели следует производить в зависимости от поставленной задачи. Несомненно, на C++ возможно написание потактовой поведенческой модели, по детализации близкой к RTL-модели, но на её создание будет потрачено не меньше времени, чем на создание RTL-модели, и работать она будет в сотню раз медленнее, чем чисто поведенческая модель той же моделируемой системы (рис.2, 3).

Если среди целей создания модели на C++ нет верификации, то выбор следует остановить на поведенческой модели. Если же эта задача поставлена, то необходимо представление о том, для верификации каких именно блоков требуется создание модели на

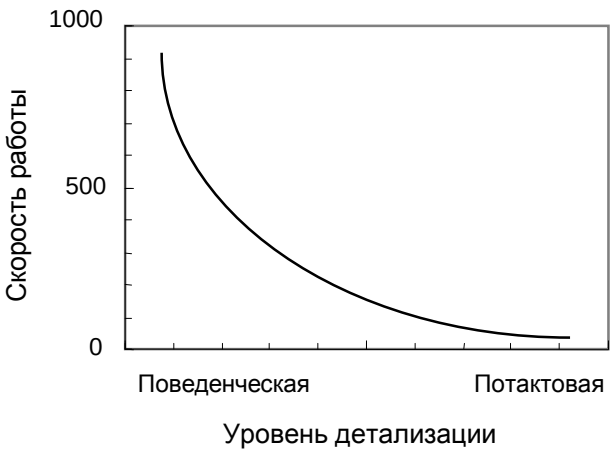

Рис. 2. Зависимость скорости работы C++ модели от уровня её детализации

C++. В СнК главные функциональные блоки, нуждающиеся в верификации – микропроцессорные ядра. Контроллеры шин передачи данных и контроллеры периферийных интерфейсов нуждаются не столько в верификации с помощью модели на C++, сколько в необходимом наборе тестовых воздействий. При этом возможно проведение оценки покрытия тестовой программой кода RTL-модели с помощью анализатора покрытия кода (например, Covered для Verilog).

Разумной альтернативой при создании модели СнК на C++ с целью верификации является потактовое моделирование ядра микропроцессора с поведенческим моделированием контроллеров периферийных устройств и других блоков.

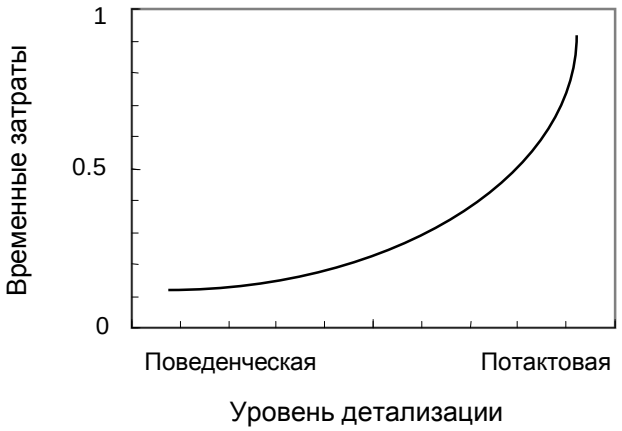

Рис. 3. Зависимость времени, затраченного на создание C++ модели от уровня её детализации

V. Интерфейс модели

Интерфейс поведенческой модели зависит от требований, предъявляемых будущими пользователями и от фантазии разработчика. Самый простой вариант – неинтерактивная модель. Модель запускается из командной строки с набором ключей, считывает файл конфигурации, выполняет тестовую задачу и завершает работу. Достоинством является то, что возможен прогон набора тестовых задач в «пакетном режиме» (автоматически, один за другим). Поэтому такой режим работы обязательно должен быть предусмотрен.

Возможно создание простого текстового пользовательского интерфейса. Пользователь вводит нужную ему команду, модель выполняет соответствующее действие. Кроме того, при наличии в моделируемой системе порта UART, хорошим вариантом является возможность работы с терминалом. Первый способ работы заключается в создании ещё одного процесса и запуска консоли-терминала во втором окне, взаимодействие терминала с моделью можно реализовать с помощью сокетов. Второй способ – подключение к псевдо-терминалу (в UNIX-системах) и работа с ним. При желании возможно организовать даже удалённую работу с поведенческой моделью.

Встречаются также модели с псевдографическим интерфейсом. Однако, если требуется, чтобы поведенческая модель имела свой графический интерфейс, лучшим вариантом будет его создание с помощью кроссплатформенных инструментов – GTK+ или Qt.

Кроме наличия собственного интерфейса, интересной возможностью является отладка тестовых программ на поведенческой модели с помощью отладчика GDB или DDD – графического интерфейса к нему. Во-первых, для этого необходимо, чтобы микропроцессорное ядро имело архитектуру, поддерживаемую отладчиком. Во-вторых, необходимо включить в код поведенческой модели код программы gdbserver. Далее поведенческая модель запускается и ожидает соединения, запускается отладчик и соединяется с моделью. Отладчик загружает объектный файл отлаживаемой программы для себя, затем загружает программу в модель. Всё готово к отладке.

VI. Заключение

На C++ возможно создание поведенческих и потактовых поведенческих моделей цифровых устройств, микропроцессорных систем и «систем на кристалле». Потактовые и поведенческие модели имеют разную степень детализации, обладают разной производительностью и служат для разных задач. Выбор степени детализации и интерфейса зависит от поставленной задачи и количества времени, выделенного на разработку. В работе рассмотрены некоторые сложные ситуации, возникающие в процессе создания поведенческой модели СнК на C++, и приведены рекомендации по их решению.

Литература

[1] Евтушенко Н., Немудров В., Сырцов И. Методология проектирования систем на кристалле. Основные приницпы, методы, программные средства // Электроника: наука, технология, бизнес. – 2003. – №6 – С. 7-11.

[2] Кравченко В., Радченко Д. Виртуальное прототипирование для аппаратно-программной верификации СБИС // Электроника: наука, технология, бизнес. – 2003. – №7 – С. 34-37.

[3] Подобаев В.Н. Методы отладки RTL-модели процессора К3 с помощью потактовой поведенческой си-модели // Научная сессия МИФИ-2005. Том 1. – С. 109 – 110.

[4] Шагурин И., Канышев В. Применение языка SystemC и средств разработки на его основе для проектирования «систем на кристалле» // Chip News. – 2006. – №9(112) – С. 51-56.

[5] Shanthamoorthi Velusamy, Anurag Mehta. Unified modeling in C++ and SystemC // Design & Reuse Industry Articles. – 2008.