УДК 004.415.538 : 004.053

Анализ и решение проблем реализации платформенного подхода к построению верификационной инфраструктуры для СнК и СФ-блоков

К.А. Безгласная, Я.С. Колбасов, И.А. Медведев, Ф.М. Путря

ОАО НПЦ «ЭЛВИС», kolbasov@elvees.com, fputrya@elvees.com

***Аннотация* – Маршрут проектирования современных систем на кристалле предполагает создание автономных тестовых окружений (ТО) для моделей СФ-блоков и групп СФ-блоков, а также набора окружений для модели всей СнК, имитирующих все предполагаемые сценарии её применения. В результате роста количества СФ-блоков в составе современных СнК и сложности самих блоков время на разработку верификационной инфраструктуры стало соизмеримо с временем на разработку и отладку тестовых сценариев, которое в свою очередь тоже нелинейно возрастает. В статье рассматривается ряд проблем, приводящих к росту трудозатрат на создание верификационной инфраструктуры, и пути их решения. Основным вектором описываемых решений является унификация верификационных компонентов (ВК) и автоматизация в области создания верификационной инфраструктуры.**

***Ключевые слова* – верификация, СнК, UVM, унификация, многократное использование, тестовое окружение, автоматизация.**

I. Введение

Верификационная инфраструктура — это набор средств, необходимых для функциональной верификации проектов. При этом актуальной задачей является минимизация сроков разработки верификационной инфраструктуры, приводящая к возможности как можно раньше приступить к исследованию новых проектов. На современном этапе развития технологий и методов верификации больших проектов, в т.ч. и СнК, наибольший выигрыш дают методы сокращения трудозатрат, подразумевающие увеличение доли повторного использования программного кода верификационных компонентов и автоматизацию процесса создания кода (например, генераторы ТО).

Методология UVM [1] изначально ориентирована на повторное использование программного кода верификационных компонентов в рамках автономных тестовых окружений. Она предоставляет жесткую структуру для реализации автономных тестовых окружений для СФ-блоков и групп блоков различной функциональности. Однако в процессе разработки возникают разночтения в оформлении и реализации компонентов, например: именование портов, способ конфигурации, взаимодействие с интерфейсами, что препятствует быстрой интеграции кода от различных разработчиков в рамках окружения, включающего несколько агентов. Возникает необходимость создания дополнительных правил разработки агентов.

Если описанные выше проблемы можно решить расширением набора правил и средств их контроля в рамках существующего подхода, то в случае ТО уровня СнК универсального подхода к организации ТО системного уровня на данный момент нет. В результате большинство компаний имеют собственные решения в области тестирования модели СнК. Наличие нескольких принципиально различающихся подходов и отсутствие единых методик верификации СнК на системном уровне усложняет задачу выбора стратегии верификации СнК и создания верификационной инфраструктуры.

При переходе к верификации на уровне системы возникают следующие трудности, связанные с особенностями реализации методологии UVM, а именно:

1) Отсутствие стандартного решения для ТО СнК.
2) Автоматизированная генерация элементов тестовых окружений возможна только при условии четко определенных интерфейсов для верификационных агентов, которые в UVM остаются на усмотрение разработчика ВК.

II. Требования к тестовым окружениям для верификации СнК

Верификация на уровне системы подразумевает создание тестового окружения, имитирующего окружение СнК, а также разработку тестов и тестовых сценариев. Разработка такого ТО – задача крайне трудоемкая, а в условиях современного рынка, диктующего сокращение цикла разработки проектов, и ограниченности ресурсов - зачастую невыполнимая. Задача разработки ТО системного уровня усложняется отсутствием стандартизованных на данный момент решений. В целом можно выделить три основных подхода: тест управляет моделью СнК через интерфейс управляющей шины (классический UVM подход), смешанные тесты, управляемые программой на модели ЦПУ и смешанные тесты, управляемые тестовым окружением.

Если от автономных ТО требуется возможность запуска направленных и условно-случайных тестов и возможность управления из теста верификационными компонентами тестового окружения и реализация сбора функционального покрытия, то от тестового окружения системы требуется возможность запуска тестовых последовательностей для проверки корректности интеграции СФ-блоков на уровне цепей и сигналов, и взаимодействия между СФ-блоками при решении приближенных к прикладным задач. К специфическим задачам можно отнести: анализ быстродействия при решении этих задач, анализ уязвимостей на границах частотных доменов и доменов питания, а также сбор различных “интегральных” оценок конечной системы, таких как: загруженность коммутаторов, пиковая переключательная активность и потребляемая мощность, время инициализации чипа, работа системы с различными режимами экономии потребляемой мощности, оценка качества тестирования (покрытия).

Как правило, перед началом верификации СнК на уровне системы уже имеются верификационные окружения для СФ-блоков составляющих систему, содержащие большинство описанных элементов. Задачей в этом случае является минимизация трудозатрат на перенос необходимого кода из автономных окружений на окружения системного уровня.

Помимо разработки аппаратной модели СнК неотъемлемой частью маршрута проектирования СнК является разработка ПО. Данный аспект следует учитывать уже на этапе верификации, так как в настоящее время сложность и трудоемкость задачи по разработке и верификации программного обеспечения для современных СнК вплотную приближается к задаче по разработке аппаратной части самой СнК [2]. А в мировой практике зачастую и превышает её. Поэтому актуальным направлением развития можно считать методы параллельной функциональной верификации аппаратной части СнК и верификации ПО для неё (SW-HW coverification). Учитывая, что по статистике более 80 % СнК содержат хотя бы один модуль ЦПУ [3], эти две задачи имеют достаточно точек пересечений.

Совместное использование одного программного кода как для тестов, так и для разработки ПО, с точки зрения трудозатрат и времени разработки является крайне эффективным решением. Поэтому увеличение доли такого кода становится одним из приоритетных требований к маршруту верификации современных СнК. К этому добавляется возможность ранней разработки и отладки такого кода уже на автономных тестовых окружениях. Для UVM такую задачу принято решать использованием стандартного c_stimulus_pkg API либо аналогичного интерфейса. Основной принцип заключается в поддержке тестовым окружением вызовов из кода С программы DPI-функций для записи/чтения регистров и ячеек памятей, вызова задержек симуляции и прочих системных вызовов.

Далее будем рассматривать SystemVerilog (SV) в качестве основного открытого языка написания ТО.

III. Подходы к построению тестовых окружений для верификации СнК

A. *SV тест управляет моделью СнК посредством master-интерфейса*

В данном подходе в модели СнК ЦПУ подменяется активным верификационным UVM агентом, исполняющим тестовые сценарии для СФ-блоков. Этот подход имеет ряд преимуществ, в их числе возможность запускать на модели СнК большинство автономных тестов для СФ-блоков, написанных ранее на языках SV и C. Также при данном подходе сохраняется характерный для автономных UVM окружения способ управления из теста верификационными компонентами периферийных устройств (рис. 1).

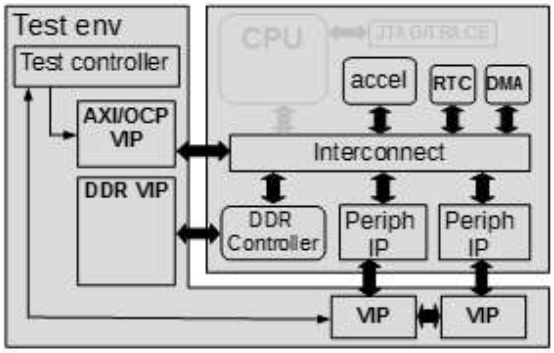

Рис. 1. Схема классического UVM подхода к верификации СнК

Но так как исполнение тестов производится средствами рабочей станции, на которой моделируется тестовое окружение, то модель трафика полученных тестов будет отличаться от модели трафика тестов, исполняемых на ЦПУ, на работу с которым рассчитана разрабатываемая СнК. Данная проблема усугубляется при переходе к многоядерной архитектуре ЦПУ. Также модификации тестов могут потребоваться, если разрядность шины данных или адресов интерфейса ЦПУ не совпадает с разрядностью системной шины верифицируемого СФ-блока.

Тем не менее, из представленных трех данный подход требует наименьшего количества трудозатрат на модификацию программного кода автономных тестовых окружений для запуска первого теста и в настоящее время считается наиболее распространенным среди UVM-сообщества. В частности, данный подход рекомендуют специалисты компании Cadence.

B. SV-тест управляет тестом на модели ЦПУ

Такой подход предполагает разбиение алгоритма теста на две части. Первая часть в виде набора функций и специальной библиотеки, скомпилированных под систему команд ЦПУ (модель которого используется при моделировании СнК), загружается в модель программно-доступной памяти. Вторая моделируется в составе тестового окружения на рабочей станции и является ведущей по отношению к первой. В начале симуляции работы СнК на модели ЦПУ запускается специальный обработчик, который ожидает вызов из SV-теста для исполнения заранее скомпилированных функций. Как правило, в качестве буфера для обмена данными выделяют небольшой диапазон памяти, программно-доступной как со стороны модели ЦПУ, так и со стороны ТО.

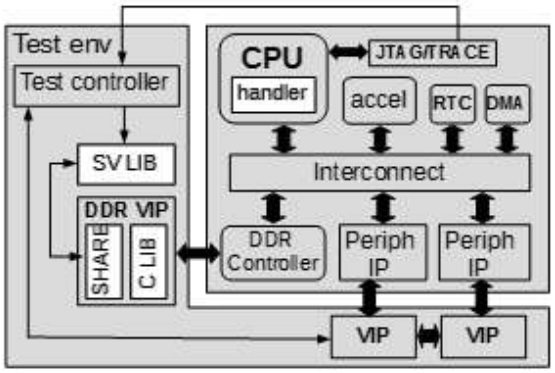

Рис. 2. Схема подхода «SystemVerilog тест ведущий»

В таком подходе SV-тест сохраняет механизмы управления верификационными компонентами ТО. А исполнение функций для работы с регистрами, памятью и прерываниями на модели ЦПУ дает достаточно точную модель трафика конечной системы. И в целом позволяет проверить интеграцию программной и аппаратной части СнК для одно- и многоядерных систем.

С другой стороны такая структура накладывает ограничение на сами тесты при отладке на автономном окружении блока. Она вынуждает разработчика тестов работать сразу с двумя языками программирования. Кроме того, данный подход требует достаточно сложной инфраструктуры и единого протокола взаимодействия как со стороны ТО, так и со стороны библиотеки, исполняемой на модели ЦПУ. В настоящий момент данный подход используется в коммерческом продукте Questa inFact [4].

C. Тест, исполняемый на модели ЦПУ, управляет тестовым окружением

В этом подходе ведущей становится программа, исполняемая на модели ЦПУ. Механизм управления ВК [5] выполнен в виде абстракции на уровне TLM и может быть реализован на любом внешнем интерфейсе, поддерживающем адресацию транзакций. Таким интерфейсом может служить, например, порт внешней памяти. С помощью такого механизма программа может вызывать любые последовательности из состава периферийных и вспомогательных верификационных компонентов.

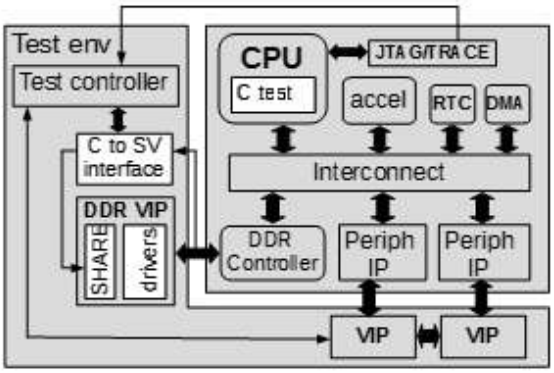

Рис. 3. Схема подхода «C++ тест ведущий»

Достоинством данного подхода является то, что однажды написанный и отлаженный на автономном тестовом окружении C++ тест, может переноситься практически без изменений на ТО системы. Этот же тест может впоследствии использоваться на отладочной плате и прототипе. Кроме того, такая структура позволяет разработчику тестов не обращать внимание на особенности реализации тестового окружения и отлаживать на автономном окружении уже связки тестов с программным драйвером устройства.

Обратной стороной этого подхода является то, что для каждого верификационного агента, вызываемого в тестах, потребуется создание средства управления агентом со стороны процессора и явное описание функций взаимодействия процессора и агента. Но создав их один раз, дальше можно их использовать для различных тестов и для различных проектов.

Также по сравнению с другими подходами уменьшается количество трудозатрат при верификации сложных алгоритмов, например, путем замены программной реализации части алгоритма на аппаратную.

Из таблицы 1 видно, что первый подход выигрывает у двух других во времени запуска первого теста на уровне СнК, но проигрывает в уровне верификации ЦПУ и ПО. А третий подход, по сравнению со вторым, будет давать значительный выигрыш во времени разработки и отладки тестов на этапе прототипирования и верификации ПО.

Исходя из этого, агенты и автономные тестовые окружения СФ-блоков необходимо разрабатывать с учетом обеспечения возможности управления ими со стороны модели центрального процессора.

Таблица 1

Сравнение реализации требований к тестовым окружениям для СнК, в парадигме использования UVM

	UVM(оригинальный)	**UVM управляет программой на ЦПУ**	**Программа на ЦПУ управляет окружением UVM**
Средства исполнения теста	Рабочая станция	Рабочая станция - SystemVerilog для ведущей части, модель ЦПУ - C/C++ для ответной части	C/C++, исполняемый на модели ЦПУ. Вызываемые функции агентов на рабочей станции
Возможности верификации аппаратной части СнК	Интеграция СФ-блоков, взаимодействие СФ-блоков с моделями коммутатора и памятей	Интеграция СФ-блоков, взаимодействие СФ-блоков с моделями коммутатора и памятей, интеграция ЦПУ	Интеграция СФ-блоков, взаимодействие СФ-блоков с моделями коммутатора и памятей, интеграция ЦПУ
Возможность переноса тестовых последовательностей из автономных ТО на системный уровень	Перенос SV- и C-тестов с минимальными изменениями (при условии сходства ведущего интерфейса с управляющим для устройства либо изначального создания тестов на более высоком уровне абстракции)	Перенос SV-тестов с изменениями. Требуется написание аналогов SV функций на C/C++, а также создание и сопровождение платформы для их поддержки.	Перенос тестов на C++ производится практически без изменений. Требуется создание C++ to SV интерфейса для автономного тестового окружения
Управление верификационными компонентами для периферийных устройств	Да, через виртуальный транзактор (sequencer)	Да, через виртуальный транзактор (sequencer)	Да, через C++ to SV интерфейс (интерфейс реализуется как для C так и для SV части)
Возможность коверификации аппаратной и программной частей проекта	Фрагменты программного кода (без учета архитектуры и особенностей целевого ЦПУ), обработчики прерываний	Фрагменты программного кода, компоненты диагностики, обработчики исключений и прерываний	Простые драйвера, фрагменты драйверов, компоненты диагностики, обработчики исключений и прерываний
Возможность использования системных тестов для тестировочных плат и прототипа	Возможно использование фрагментов C-тестов	Возможно использование C/C++ функций, в качестве основы для тестов	C/C++ тесты, отлаженные на модели данного ЦПУ. Переносятся практически без изменений

IV. Многоуровневый подход к взаимодействию с интерфейсами

Анализ агентов ряда внутрикристальных и периферийных интерфейсов позволяет сделать вывод о возможности создавать тестовые последовательности на более высоком уровне абстракции.

Такие последовательности не будут зависеть от конкретной реализации интерфейса и их можно использовать не только между проектами, но и между различными интерфейсами. Задание трафика выполняется на нескольких уровнях абстракции:

1) поток данных с определенными характеристиками, динамически формирующимися по заданным ограничениям;

2) последовательность универсальных транзакций (например, uvm_tlm_generic_payload), не зависящих от конкретного интерфейса, полученная из потока данных;

3) последовательность транзакций, соответствующих особенностям конкретного интерфейса (получена путем динамического расширения уровня универсальных транзакций на базе заданных ограничений).

Мониторинг активности интерфейса также производится с использованием указанных уровней, отличаясь лишь направлением преобразования транзакций. Предложенный подход позволяет повторно использовать уже созданные тестовые сценарии, средства мониторинга активности интерфейса и анализа загруженности интерфейса для вновь создаваемых интерфейсов. Кроме простого повторного использования кода такой подход позволяет создавать тестовые окружения коммутаторов, сочетающих в себе большую номенклатуру внутрикристальных и периферийных интерфейсов, используя при этом единый механизм взаимодействия со всеми верификационными компонентами.

Для ускорения переноса тестовых сценариев необходим учет в структуре вновь создаваемых агентов возможности передачи транзакции, сгенерированной на более высоком уровне абстракции.

V. Классификация верификационных компонентов

Создание ТО СФ-блока, группы блоков или СнК осуществляется посредством использования ВК, отвечающих требованиям UVM. В начале статьи было показано, что для облегчения интеграции ВК, создаваемых различными группами инженеров, необходимо усилить требования к коду таких агентов. В силу разнообразия устройств и интерфейсов создать единых шаблон для всех компонентов не получится, однако можно провести классификацию компонентов и выделить счетное число групп компонентов, для каждой из которых будет применим единый шаблон.

Верификационные компоненты можно классифицировать по нескольким признакам:

A. По способу передачи информации:

1) без адреса;
2) с адресом;
    a) адресом приемника;
        - полная адресация;
        - адресация по номеру устройства;
    b) адресом приемника и источника.

Данное деление ВК позволяет создать универсальную базовую транзакцию для каждого класса с помощью TLM Generic Payload с соответствующими дополнениями для обеспечения максимальной степени повторного использования высокоуровневых тестовых последовательностей.

B. По необходимости параметризации:

1) не параметризируемые;
2) параметризируемые.

Работа с не параметризируемыми интерфейсами осуществляется посредством стандартного механизма UVM.

Современные тестовые окружения, написанные на SystemVerilog, используют интерфейсы с одной стороны и виртуальные интерфейсы с другой для связи статических модулей с динамическими классами. При использовании в рамках одного окружения интерфейсов одного типа, но с разными параметрами, становится затруднительным работать со всем набором агентов как с единым массивом [6]. Для решения этой проблемы все особенности, связанные с изменяемыми параметрами предлагается инкапсулировать в интерфейсе, оставив для связи с драйвером объект, обеспечивающий независимые от параметров API связи с интерфейсом. Классы агентов в этом случае нет необходимости явно параметризировать, что позволяет работать с ними на уровне ТО как с объектами одинакового типа.

C. По необходимости конфигурации карты памяти:

1) С картой памяти.
2) Без карты памяти.

В случае одного управляющего компонента и нескольких управляемых компонентов возникает необходимость имитации коммутации. В библиотеку верификационных компонентов добавляется дополнительный блок с логикой распределения тестовых запросов, и в агенте управляющего устройства необходимо предусмотреть конфигурацию карты памяти.

VI. Требования к шаблонам верификационных компонентов и их автоматическая генерация

В предыдущем параграфе приведены три ортогональных друг к другу группы свойств агентов. Для каждого сочетания свойств из этих трех групп может быть создан шаблон агента. Счетное множество шаблонов уже упрощает работу с агентами при их интеграции в ТО высокого уровня, однако ко всем типам агентов можно предъявить еще ряд требований:

1) соответствие требованиям методологии UVM;
2) обеспечение единообразия именования членов классов и портов;
3) использование UVM TLM портов для обеспечения двунаправленного взаимодействия между различными верификационными компонентами и/или другими элементами тестового окружения;
4) обеспечение работы между ВК и моделью памяти посредством драйвера памяти;
5) обеспечение активного и пассивного режима работы агента;
6) конфигурация агента осуществляется посредством конфигурационного объекта;
7) класс транзакции, используемый ВК, наследуется от универсальной транзакции, выбранной исходя из того к какому классу относится агент;
8) не параметризируемые ВК связываются с интерфейсами посредством виртуального интерфейса, параметризированные посредством объекта с унифицированным API.

Упрощение сборки различных ВК происходит за счет следующих требований:

9) обеспечение единообразной файловой структуры, сохраняющейся от проекта к проекту;
10) все файлы, относящиеся к агенту, должны быть собраны в package.

Наряду с общими требованиями к ВК существуют частные требования, которые напрямую зависят от специфики интерфейса. Классификация ВК поможет систематизировать частные требования и, зная к какому классу принадлежит разрабатываемый компонент, использовать соответствующую группу предъявляемых к данному классу требований.

Требования, перечисленные выше, позволяют облегчить интеграцию компонентов в рамках одного ТО.

A. *Генерация верификационных компонентов*

Генератор верификационных компонентов, созданный на основе вышеописанных требований к ВК, обеспечивает их обязательное соблюдение. Генераторы, создающие заготовку верификационного кода, позволят максимально автоматизировать процесс создания верификационных компонентов, При этом сгенерированный код на выходе будет обладать максимальной степенью повторного использования как по вертикали абстракций (на разных уровнях абстракции), так и по горизонтали проектов (от проекта к проекту)

B. *Оценка результатов*

Процесс создания ВК можно условно разделить на 3 этапа:

1) создание структуры шаблонов ВК, которая представляет набор базовых ВК, соответствующих всем вышеописанным требованиям;

2) наполнение ВК функционалом, который определяется интерфейсом;

3) отладка структуры ВК.

Рис. 3. Оценка среднего времени, затрачиваемого на различные этапы создания ВК в трудоднях

Процесс создания ТО также можно разделить на 3 этапа:

1) создание шаблона ТО, соответствующего вышеописанным требованиям к ТО;

2) наполнение ТО функционалом, который определяется тестируемым устройством;

3) отладка ТО.

На основе диаграмм (рис. 3, 4) можно сделать вывод о том, что хотя использование генератора ВК на этапе создания ВК сохраняет незначительное количество времени, на этапе создания ТО системного уровня применение генератора ВК играет значительную роль и уменьшает время, затрачиваемое на создание начальной версии ТО на 55%.

Рис. 4. Оценка среднего времени, затрачиваемого на различные этапы создания ТО системного уровня

VII. Заключение

В данной работе проанализированы подходы к верификации СнК на системном уровне в контексте задачи коверификации программной части и аппаратуры. Более удобным является тот подход, где тест, исполняемый на модели ЦПУ, управляет тестовым окружением. Показано, что требований, предъявляемых методологией UVM, недостаточно для организации платформенного подхода в рамках предприятия. Предложена классификация верификационных компонентов. Разработан ряд дополнительных общих требований к агентам, и на основе предложенной классификации ряд частных требований к разным классам агентов, в т.ч. с учетом специфики их повторного использования на уровне СнК. Для гарантии унификации агентов и снижения трудозатрат на разработку верификационной инфраструктуры предложено использование генераторов кода агентов и ТО, созданных на основе сформулированных требований и отличающихся от известных на данный момент генераторов.

Литература

[1] Universal Verification Methodology (UVM) 1.1 Class Reference URL: http://www.accellera.org (дата обращения: 13.01.2014).

[2] Avinun R., Validate hardware/software for nextgen mobile/consumer apps using software-on-chip system development tools, embedded.com. December 14, 2010.

[3] The 2012 Wilson Research Group Functional Verification Study URL: http://www.mentor.com/products/fv/multimedia/ (дата обращения: 03.02.2014).

[4] Automating Software Driven Hardware Verification with Questa Infact URL: http://www.mentor.com/products/fv/multimedia/automating-software-driven-hardware-verification-with-questa-infact (дата обращения: 03.02.2014).

[5] Jin-kwon P., Cheon-su L., Jae-shin L., Min-jong L. System On Chip (SoC) Device Verification System Using Memory Interface // U.S. Patent US8239708 B2. May 29.

[6] 2009 Clasen G., Flexible UVM Components: Configuring Bus Functional Models // Verification Horizons. 2013. V. 3. №2. P. 58-63.