УДК 519.714

Практические аспекты верификации проектов СБИС

А.А. Сохацкий

Сиско Системс Инк., asokhats@cisco.com

***Аннотация* — В статье рассматриваются практические аспекты функциональной верификации проектов на основе опыта работы автора в компании Сиско Системс Инк. Акцентируется внимание на верификации СБИС на уровне блоков и их функциональных групп (кластеров) путем моделирования на псевдослучайных входных последовательностях с учетом ограничений и достижением покрытия тестами конструкций и функций проекта. При моделировании используются отраслевые стандарты: язык моделирования SystemVerilog и универсальная верификационная библиотека классов UVM (Universal Verification Methodology) [1]. Тезисно излагаются следующие аспекты:**

1) **основы построения окружения для верификации проекта – элементы инфраструктуры, построенные на основе UVM классов: состав и структура верификационного окружения, конфигурационное дерево, дополнительные фазы моделирования, набор базовых классов, производных от классов UVM, компоненты общего назначения;**
2) **вопросы автоматизации создания верификационного окружения, включая генерацию структуры и начального кода для базовых компонентов; создания структур данных, обеспечивающих программный доступ к регистрам и памяти тестируемого устройства, его инициализации, обработки прерываний. В качестве общего источника данных выступает спецификация регистров устройства, выполненная на языке описания регистров System RDL (Register Description Language) [2];**
3) **технологические этапы верификации проектов и используемый инструментарий, включая поддержку подготовки и сопровождения плана тестирования, статической проверки кода, моделирования, выполнения регрессионных тестов, хранения и совместного использования кода.**

***Кляючевые слова* — СБИС, верификация, моделирование, RTL, SystemVerilog, UVM, SVA, System RDL.**

I. Введение

Статья основана на опыте верификации проектов сетевых СБИС в компании Сиско Системс Инк. В основном это проекты устройств, обрабатывающих потоки пакетов сетевых сообщений, в частности, маршрутизаторы, коммутаторы. В подобных работах требуется верифицировать сложные СБИС, содержащие несколько миллиардов транзисторов и большое количество функциональных блоков, требующих автономной верификации. Верификационные коллективы проектов состоят из десятков инженеров, работа которых длится месяцами. Всё это требует уделения особого внимания вопросам технологии верификации, создания и поддержания единого технологического процесса для простоты интеграции, горизонтального (между блоками одного уровня и от проекта к проекту) и вертикального (от блока нижнего уровня к кластеру, состоящему из ряда блоков, и к СБИС) повторного использования компонентов верификационных сред, динамичного перераспределения ресурсов коллективов и, конечно, повышения качества и сокращения времени разработки. Последовательный технологический процесс должен покрывать все этапы верификации от анализа проектной спецификации до лабораторных испытаний изготовленной микросхемы и последующего сопровождения проекта.

Статья базируется на опыте работы автора в составе распределенной географически верификационной группы, внесшей существенный вклад в разработку единой верификационной методологии компании и ее реализации. В качестве основного метода верификации проектов используется метод моделирования тестируемого устройства на псевдослучайных входных последовательностях с учетом ограничений и с последующим достижением полноты покрытия конструкций проекта, функционального покрытия и покрытия утверждений SVA (System Verilog Assertions). Верифицируемое устройство представлено на уровне регистровых передач (Register Transfer Level, RTL) и описано на языке Verilog. В статье не рассматриваются другие перспективные методы, дополняющие метод моделирования, такие как методы формальной верификации и использование эмуляторов.

Верификация проектов проводится на нескольких уровнях: блоков, кластеров или супер-блоков, СБИС, системном. В данной классификации супер-блок (кластер) содержит несколько блоков нижнего уровня, тестируемых совместно как единый функциональный блок. Статья фокусируется на методологии верификации на уровне блоков и супер-блоков. При этом во многих аспектах эта методология применима и для верхних уровней. Следует обратить внимание, что в рассматриваемой методологии лишь некоторые компоненты, разработанные для проверки блоков, повторно используются для проверки СБИС, например агенты интерфейсов. В то же время в описываемом технологическом процессе языки моделирования, используемые на уровне блоков и СБИС, различны: SystemVerilog на уровне блоков и супер-блоков и C++

на уровне СБИС. Верификационное окружение на уровне СБИС, построенное на базе C++, является достаточно независимым от окружения на уровне блоков и супер-блоков, построенных на базе SystemVerilog. Это дает дополнительную уверенность в качестве верификации и исключает зависимость начала верификации на уровне СБИС от готовности всех компонентов на уровне блоков. Использование C++ для проверки на уровне СБИС позволяет повторно использовать C++ код для тестовых испытаний изготовленной СБИС в лаборатории.

II. Построение верификационного окружения

A. *Базовый язык и верификационная библиотека*

Выбор базового языка и верификационной библиотеки на уровне блоков и супер-блоков очевиден – использование отраслевых стандартов – языка SystemVerilog и верификационной библиотеки UVM. Мы использовали богатый набор возможностей языка SystemVerilog в описании параллельных процессов, функционального покрытия, генерации случайных наборов, включая использовние недавно включенной в стандарт SystemVerilog конструкции *soft constraint*, позволяющей задавать начальное ограничение, легко заменяемое в тесте.

При построении верификационного окружения и тестов использование библиотеки UVM оказалось весьма полезным. В частности, эффективными оказались ее следующие возможности: стандартизация фаз моделирования; интерфейсные агенты; обмен данными, включая использование портов типа analysis; последовательности, механизм замены классов (*factory replacement*) и др. Вокруг основных классов UVM мы расположили слой производных классов, предполагаемых к использованию вместо оригинальных UVM классов, и разработали дополнительные требования, включенные в состав верификационной методологии.

B. *Структура верификационного окружения*

Верификационное (тестовое) окружение в рассматриваемом подходе построено из следующих компонентов:

1) интерфейсный верификационный компонент (IVC) соответствует RTL-интерфейсу и содержит, как правило, один, но, возможно, несколько UVM агентов (agents). В рассматриваемом подходе в состав стандартного агента входит монитор, секвенсер (sequencer) и два драйвера: основной драйвер в направлении передачи данных, названный upstream driver, и драйвер обратной связи, регулирующий возможность приема данных, названный downstream driver. Если базовый UVM агент предполагает активный (active) и пассивный (passive) режимы в зависимости от активности драйвера, то мы ввели режимы ‘active downstream’ и ‘active upstream’. Единый IVC используется для блоков, соединенных общим интерфейсом. При этом, как правило, в верификационном окружении одного блока агент будет в режиме ‘active upstream’, а другого – ‘active downstream’.

2) Модульный верификационный компонент (MVC) соответствует верифицируемому модулю аппаратуры: блоку или супер-блоку. Он содержит указатели на интерфейсные компоненты (IVC) модуля, обеспечивает инициализацию и его конфигурирование, текущую проверку правильности его функционирования во время моделирования с использованием проверочных компонентов scoreboard и эталонных моделей, проверку его состояния в конце моделирования (EOT check). MVC предоставляет доступ к некоторым компонентам общего назначения, в частности, для авто-инициализации (Auto Initialization), доступа к регистрам устройства (CSR Access), обработки прерываний (Interrupt Error Handler), проверки производительности (Rate monitor), фонового доступа к регистрам (CSR Tickler) и т.д.

3) Общий (Common) верификационный компонент (CVC) содержит классы, используемые в нескольких интерфейсных и модульных компонентах, например классы транзакций. Следует заметить, что в дополнении к CVC существует отдельная категория компонентов общего назначения, описанных ниже в отдельном разделе.

Верификационное (тестовое) окружение (testbench, TB) для блока нижнего уровня (см. рис. 1) включает один MVC компонент и несколько IVC компонентов для интерфейсов блока, включая стандартные компоненты для управления сбросом устройства (Reset IVC) и доступа к его регистрам (CSR IVC). Верификационное окружение также содержит экземпляры компонентов общего назначения, управляющие и координирующие доступ к регистрам и обработку прерываний: менеджер доступа к регистрам и база данных регистров (CSR Access Manager & Registers DB), менеджер обрабоки прерываний и дерево прерываний (Interrupt Manager & Interrupt Tree). Пример структуры верификационного окружения показан на рис. 1. Там же показаны конфигурационные классы для всех компонентов окружения (IVC config, MVC config, TB config). Названия компонентов на рисунке соответствуют оригинальным терминам UVM и классам инфрастуктуры компании Сиско Системс Инк.

Модульные и интерфейсные верификационные компоненты блока повторно используются в верификационном окружении суперблока. Кроме того, тестовое окружение супер-блока содержит модульный компонент для суперблока, отражающий функционирование суперблока в целом, включая функциональную проверку на границах суперблока. Пример показан на рис. 2. На рисунке IVC X на входе супер-блока находится в активном режиме upstream active, в то время как на выходе IVC Z – в активном режиме downstream active. В данном случае драйвер контролирует сигнал, разрешающий пересылку транзакций. IVC_Y находится в пассивном режиме (passive): драйвер отключен, а монитор пересылает транзакции для проверки в компоненты scoreboard, расположенные в компонентах MVC блоков A и B.

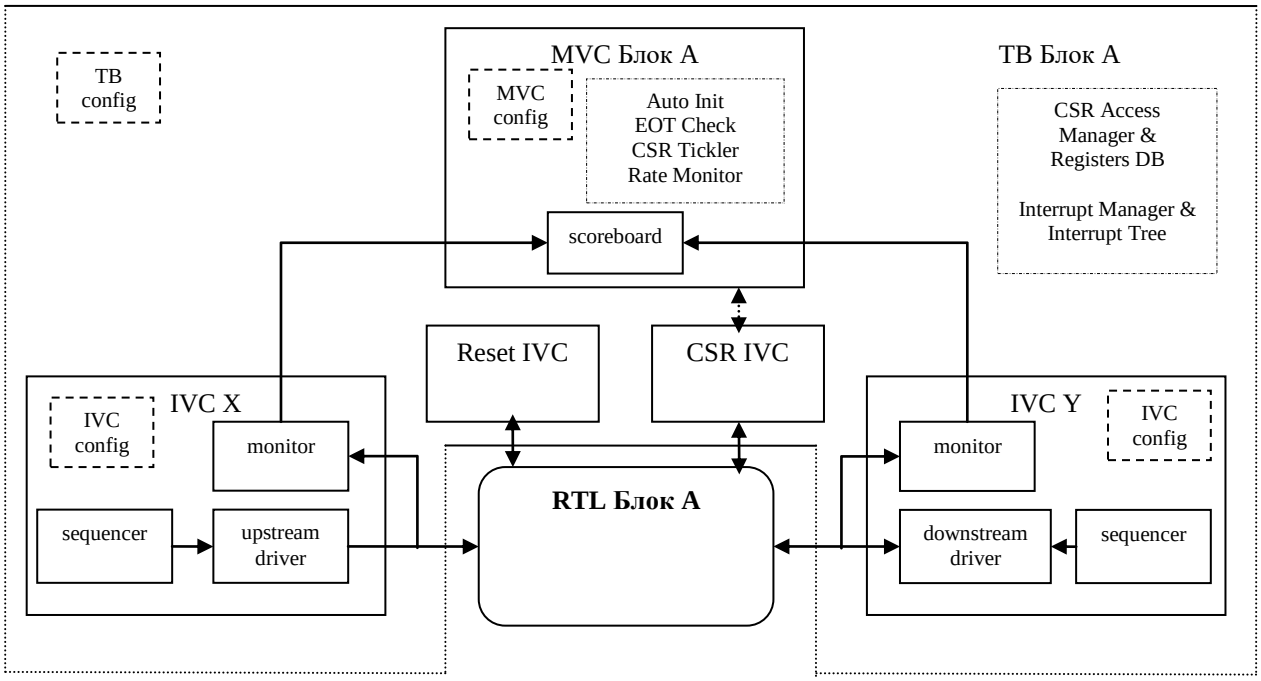

Рис. 1. Пример структуры верификационного окружения блока A

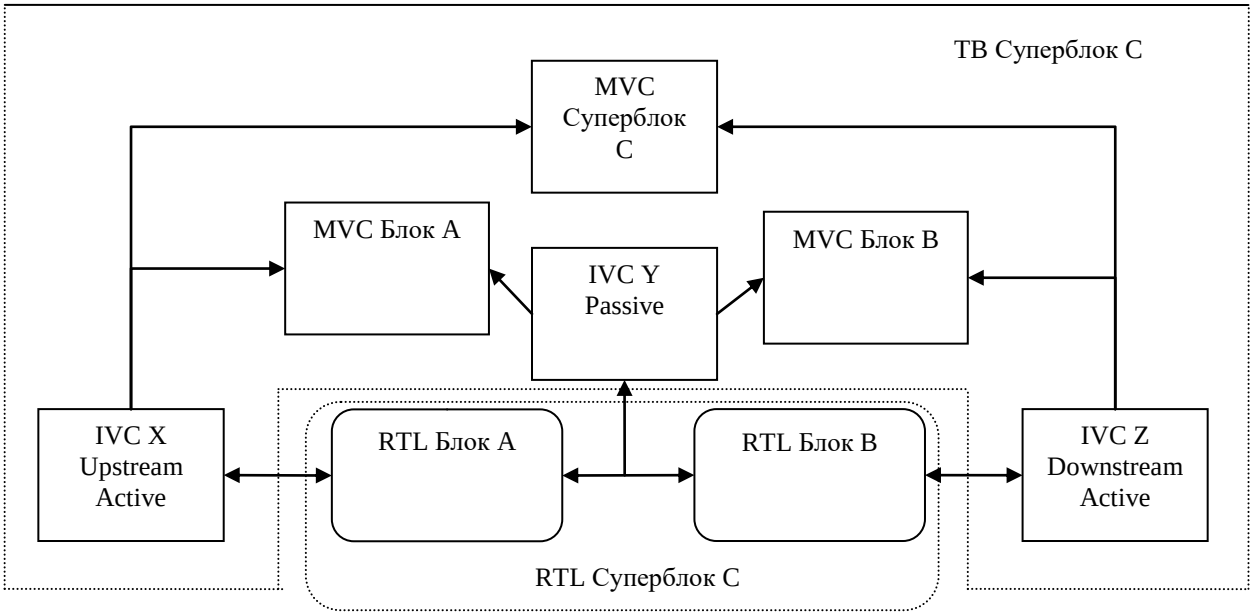

Рис. 2. Пример структуры верификационного окружения суперблока C

C. Конфигурационное дерево

Ниже рассматривается оригинальный подход задания конфигурации, применяемый в компании.

Как правило, каждый компонент верификационного окружения содержит экземпляр конфигурационного класса с набором рандомизируемых (random) переменных. Ограничения типа soft используются для задания типовых значений рандомизируемых переменных, диапазона или распределения случайных величин. В отдельных случаях компоненты верхнего уровня содержат общую конфигурацию для компонентов нижнего уровня. Интерфейсные агенты содержат единую конфигурацию для драйвера, монитора и секвенсера, например переменные, содержащие значения режимов работы интерфейса, его временных характеристик.

Конфигурационные классы компонентов верхнего уровня содержат экземпляры конфигурационных классов компонентов нижнего уроня. Например, конфигурация тестового окружения содержит экземпляры конфигураций MVC, IVC и компонентов общего назначения. Таким образом, конфигурационное дерево соответствует дереву компонентов за возможным исключением конфигураций компонентов нижнего уровня.

При построении верификационного окружения сначала строится и рандомизируется конфигурационное дерево, а затем строится дерево компонентов. Обычно конфигурационное дерево рандомизируется за один прием за исключением общей конфигурации кластера, которая может быть создана и рандомизирована перед построением всего дерева. Пользователь может изменить ограничения конфигурационных классов из тестового окружения, теста или в отдельных случаях непосредственно присвоить значение конфигурационным переменным, в том числе из командной строки симулятора.

Использование конфигурационного дерева в качестве основного инструмента конфигурирования тестового окружения считаем более последовательным и надежным подходом, чем применение конфигурационной базы данных UVM (uvm_config_db). Одной из сложностей, возникающей при отладке кода с использованием uvm_config_db, является использование шаблонов строк для поиска компонентов и отсутствие диагностики в случае ошибки при задании строки поиска или имени переменной. В других приложениях мы ограниченно используем uvm_config_db в основном для задания стандартной последовательности секвенсеру или для передачи виртуальных интерфейсов.

D. Дополнительные фазы моделирования

Мы используем стандартные фазы моделирования, определенные UVM для построения, редактирования связей и выполнения моделирования. Но к ним добавили дополнительные фазы, в частности, дополнительные фазы инициализации и конфигурирования устройства для упрощения разработки и синхронизации инициализации и конфигурирования. Например, выделили фазу разрешения (включения) блока. Таким образом, сначала выполняются подготовительная инициализация и конфигурирование блоков, а потом блоки включаются в работу. Другим примером дополнительной фазы является фаза сохранения и восстановления состояния устройства. Это позволяет выполнить инициализацию один раз, сохранить состояние и восстанавливать это состояние для различных тестов.

E. Слой базовых классов

Следуя типовому подходу к созданию верификационной среды, мы разработали дополнительный слой классов вокруг некоторых базовых классов UVM. Например, *scoreboard* класс был существенно расширен для упрощения вычисления значений ожидаемых выходных транзакций для учета возможности раннего прихода выходной транзакции, пока входная еще полностью не получена. В класс *csco_scoreboard* также встроены возможности контроля производительности верифицируемого устройства путем сопряжения с монитором производительности – смотри ниже.

Другим базовым классом является класс управления утверждениями SVA (System Verilog Assertions). Он используется для разрешения и запрещения проверки утверждений во время выполнения определенных фаз моделирования. Например, отдельные классы утверждений разрешены только в конце моделирования, а другие – запрещены во время сброса устройства. Следует обратить внимание, что использование SVA является важной составляющей процесса верификации. Разработчики RTL-описаний ответственны за помещение утверждений в описания разрабатываемых ими блоков и в описания компонентов аппаратуры общего назначения, например FIFO или счетчиков.

Другим примером базового класса является класс, обеспечивающий последовательный способ настройки конфигурации из командной строки симулятора.

F. Компоненты общего назначения

Набор компонентов общего назначения построен на основе базовых классов.

Важным компонентом общего назначения является компонент, обеспечивающий доступ к программным регистрам и памяти. Для этих целей UVM содержит набор классов uvm_reg или UVM RAL (Register Abstraction Layer). Функциональные особенности нашего подхода:

1) доступ к памяти использует тот же программный интерфейс, что и доступ к регистрам, с возможностью хранения текущих значений в ассоциативном массиве (associative array) только для тех элементов массива, к которым был произведен доступ;

2) чтение и запись являются методами класса клиента доступа, что позволяет хранить общие для клиента установки (в RAL чтение и запись являются методами регистровых классов), при этом мы сохранили имена методов RAL.

Ряд компонентов общего назначения разработан на базе классов доступа к регистрам.

1) Классы поддержки инициализации и конфигурирования устройства, содержащие набор виртуальных задач, вызываемых во время инициализации. Например, это задачи инициализации памяти с помощью специальных аппаратных модулей, задачи конфигурирования регистров, прерываний, разрешения (включения) блока. Задачи вызываются во время определенных дополнительных фаз моделирования и наполняются конкретным содержанием в производных классах для компонентов определенного блока.

2) Обработчик ошибок и прерываний (IEH, Interrupt Error Handler) отслеживает сигнал прерывания, обнаруживает источник прерывания путем навигации по дереву прерываний, проверяет, ожидается ли это прерывание (ранее это ожидание должно быть задано из тестового окружения или теста), выполняет обработку, например, читает связанные с прерыванием регистры, очищает регистр прерывания.

3) Компонент проверки состояния в конце выполнения теста (End of Test Check, EOT), в частности, проверяет, что компоненты аппаратуры, в частности, FIFO (First In First Out), FSM (Finite State Machine), вернулись в исходное состояние.

4) Компонент фонового доступа к программным регистрам и памяти (CSR Tickler) выполняет разрешенный фоновый доступ (как правило, чтение) во время моделирования для проверки возможных конфликтов между доступом к памяти и регистрам со стороны аппаратуры и программного обеспечения.

Другой групой компонентов повторного использования являются компоненты поддержки тестирования производительности.

1) Монитор производительности или темпа трафика – скорости передачи (Rate Monitor) – вычисляет параметры производительности (объема передаваемых данных и количества транзакций за единицу времени) для интерфейса аппаратуры и, если требуется, отдельно для различных групп транзакций, например, для транзакций, имеющих определенный приоритет.

2) Контроллер производительности или темпа трафика (Rate Controller) управляет темпом посылки входных транзакций на основе общих ограничений интерфейса или ограничений отдельных входных потоков.

III. Автоматизация создания верификационного окружения

A. *Генерация компонентов верификационного окружения*

Библиотека UVM предоставляет большое количество возможностей, но зачастую кодирование компонентов, их настройка и соединение требуют существенного времени. Генератор верификационных компонентов (VC Gen) упрощает создание верификационного окружения, ускоряет разработку и обеспечивает единый стиль. Пользователю просто нужно ответить на несколько вопросов, задаваемых генератором в интерактивном графическом режиме. Генератор VC Gen создает компоненты IVC, MVC, CVC и TB (тестовое окружение). Вопросы для генерации MVC, в числе прочих, включают следующие:

- каков список компонентов IVC?
- Какие проверочные компоненты scoreboard должны быть установлены?
- Требуется ли установить компонент для авто-инициализации?

В результате работы VC Gen создает SystemVerilog код двух типов:

1) начальный код, который инженер-верификатор должен отредактировать и наполнить содержанием с учетом особенностей блока или интерфейса;

2) код, создающий структуру IVC, MVC или другого компонета, обеспечивающий связь между компонентами нижнего уровня и установку компонентов общего назначения. Файлы, содержащие подобный код, называются топологическими файлами; пользователи не должны их редактировать.

Достоинство этого подхода заключается в том, что при добавлении новой функции, внесении изменений и дополнений в базовые классы и элементы инфраструктуры часто нет необходимости что-либо редактировать вручную в тестовом окружении блока, а просто нужно запустить VC Gen и перегенерировать топологические файлы.

B. *Генерация компонетов, связанных с доступом к программным регистрам тестируемого устройства*

Мы используем единое описание программно-доступных регистров и памяти на языке описания регистров System RDL [2]. Описание включает размер полей регистра, возможность чтения/записи полей регистра со стороны программного обеспечения. RDL-описание является источником для генерации различного исполняемого кода и документации:

1) RTL-кода модулей аппаратуры для представления регистров и доступа к регистрам и модулям памяти;

2) документации в формате HTML;

3) кода, используемого для верификации блока и СБИС.

Описание используется также при создании программного обеспечения устройства, содержащего СБИС.

Разработчик проекта блока наряду с RTL-кодом блока подготавливает описание программно-доступных регистров и памяти на языке System RDL. Помимо свойств, определенных стандартом System RDL, задаются дополнением свойства регистров, необходимые для создания программного обеспечения и верификации. Для регистров прерываний задается свойство, которое определяет действия, требуемые для восстановления после сбоев аппаратуры, например сброс устройства. Другим примером свойства, используемого для верификации, является расположение регистра или памяти в блоке. Эта информация используется для ускоренной записи и чтения регистра или памяти путем непосредственного доступа к регистру, минуя программный интерфейс (backdoor access). Следует отметить, что эта информация требуется только для регистров, расположенных в модулях аппаратуры вне генерируемого кода и описанных с использованием ключевого слова external на языке System RDL.

Специальные генераторы используются для создания следующих верификационных компонентов:

1) карты регистров (register map), хранящей информацию о программно-доступных регистрах и памяти; карта содержит ссылки на экземпляры классов регистров, используемых для хранения текущего значения регистра и необходимой информации для обеспечения доступа, включая адрес, разрешенный тип доступа, специальные свойства, например требование очистки регистра после чтения; UVM RAL содержит аналогичную карту регистров;

2) структур данных для непосредственного быстрого доступа (backdoor access) к регистрам и памяти, которая содержит информацию о расположении регистра в RTL-иерархии и других параметрах, необходимых для быстрого доступа;

3) классов, содержащих код для автоматической инициализации (auto-initialization) блока устройства, и конфигурационных классов, содержащих набор рандомизируемых переменных для полей всех конфигурационных регистров и памяти; конструкции soft constraints задают значение регистров, равное значению после включения питания или сброса (reset), это значение можно легко изменить из тестового ограничения или конкретного теста;

4) структур данных дерева прерываний;

5) списка регистров и элементов памяти для фонового доступа во время моделирования для создания и проверки правильной обработки конфликтов между функциональным обращением и обращением программного обеспечения к регистру или элементу памяти;

6) кода для проверки состояния регистров в конце модерирования, например отсутствия непрочитанных ячеек FIFO.

C. *Генерация элементов интерфейса и соединение с RTL кодом*

Этот генератор предназначен для упрощения описания SystemVerilog интерфейсов и установки их в тестовом окружении с присоединением к сигналам тестируемого устройства. Интерфейс описывается компактнее с заданием только необходимой информации о сигналах и для присоединения к сигналам тестируемого устройства может быть достаточно нескольких строк кода при использовании символа ‘*’ в качестве любого символа. Генератор берет на себя создание SystemVerilog кода для описания и присоединения интерфейса.

D. *Автоматические тесты.*

Примером автоматических тестов являются тесты проверки программного доступа к регистрам и элементам памяти, аналогичные тестам пакета uvm_reg; тесты псевдослучайным образом выполняют операции чтения, записи, сброса с проверкой результатов. Операции чтения и записи могут быть выполнены с использованием драйверов интерфейсов доступа (front door) или с использованием непосредственного быстрого доступа (back door);

IV. Технологический процесс верификации и инструментарий

Посмотрим более широко на технологический процесс верификации, не ограничиваясь этапами создания верификационной среды и тестов. Ниже приведены отдельные этапы технологического процесса верификации и используемый верификационной группой инструментарий различной сложности, эффективности и необходимости от поставщиков САПР, свободного доступа или разработанный собственными усилиями в компании. Заметим, что ведущие компании-производители САПР также предлагают комплексные решения для поддержки отдельных этапов. Сравнение решений выходит за рамки статьи. В качестве примера верификационная платформа Synopsys представлена здесь [3].

A. *Анализ спецификации проекта и его верификационного окружения*

Этот этап является важной частью технологического процесса и призван обеспечить полную и четкую спецификацию. Анализ подразумевает участие в обсуждении нескольких разработчиков аппаратуры и верификационных инженеров. Для упрощения процесса и гарантии учета всех замечаний используется разработанный в компании инструмент, позволяющий добавлять комментарии в документ и отвечать на эти комментарии, отслеживать утверждение документа со стороны участников.

B. *Разработка и сопровождение плана тестирования*

Для представления, разработки и сопровождения плана тестирования используется сетевая программа, разработанная в компании. Программа хранит набор целей, отражающих свойства проекта, которые требуется проверить, и набор метрик, которые предназначены для достижения целей. Примером метрик являются тесты, точки функционального покрытия, утверждения SVA для проверки и покрытия, размещенные внутри RTL-кода, использование формальных методов, обработка результатов моделирования. Статус покрытия метрик может быть обновлен по результатам выполнения регрессионных тестовых последовательностей. Шаблон тестового плана содержит элементы, обязательные для проверки функционирования, производительности, ошибочных ситуаций и т.д.

C. *Статическая проверка кода проекта и верификационного окружения*

В компании применяются программы для статической проверки (lint tools, линтеры) RTL-кода и кода верификационного окружения с целью раннего обнаружения ошибок кодирования и обеспечения соответствия стиля кодирования стилю, принятому в компании. Примеры проверок кода верификационного окружения:

- для сравнения величин, принимающих 4 значения, необходимо использовать соответствующую операцию сравнения: “===” вместо “==”;
- по крайней мере одна группа проверки покрытия covergroup должна быть установлена в коде монитора.

Для статической проверки кода верификационного окружения используется программа Verissimo [4]. В дополнение к встроенным правилам проверки в базу данных проверки были добавлены правила в соответствии со стилем кодирования верификационного окружения, принятым в компании.

В компании разработана программа проверки кода описания регистров на языке System RDL. В частности, ею проверяется наличие описания обязательных свойств (properties) регистров для генерации кода быстрого доступа для инициализации утройства.

D. Подготовка и компиляция RTL-кода и окружения и выполнение тестов

Для описания RTL-кода используется расширение языка Verilog с возможностью использования некоторых конструкций языка Perl. В процессе компиляции используется препроцессор. Он также позволяет сократить описание портов и внутренних сигналов, т.к. все порты внутренних компонентов автоматически описываются в компоненте более высокого уровня в качестве внутренних сигналов или портов.

Помимо традиционных средств, в частности программы make, разработана специальная программа – макропроцессор, упрощающая описание шагов построения, компиляции и моделирования с использованием макросов. Макросы могут быть адаптированы для конкретного блока или проекта. Макросы позволяют легко задать из командной строки, например, требования сбора информации о покрытии кода или уровень детальности отладочной печати.

E. Моделирование (X-оптимизм)

Одной из проблем моделирования RTL-кода является X-оптимизм. Например, для следующего фрагмента Verilog-кода:

if (a) b = 0; else b = 1,

если ‘a’ равно ‘X’, то проверочное условие ложно, выполняется оператор в ветви ‘else’, который приводит к присвоению ‘b’ значения 1. Это не соответствует поведению синтезируемого кода на уровне вентилей (gate-level), который присвоит значение ‘X’ результату. Это является одной из причин необходимости моделирования на уровне вентилей. Во многих случаях этого можно избежать путем использования специальных средств моделирования RTL-кода, таких как опции Synopsys VCS X-Prop [5].

F. Регрессионные тестовые последовательности и технологический процесс создания релизов

Технология создания релизов RTL-кода и верификационного окружения построена на базе системы хранения и совместного использования файлов (revision control system), позволяющую обеспечить параллельную разработку проекта группой инженеров, (возможно) распределенных географически.

В используемой технологии файлы верификационного окружения сгруппированы в пакеты, например пакеты IVC, MVC, тестового окружения. В свою очередь, пакеты входят в состав контейнеров, например, базового контейнера, контейнера блока, контейнера БИС. При выполнении релиза контейнера предварительно автоматически выполняется регрессионная минимальная тестовая последовательность, предназначенная для базовой проверки.

Полная регрессионная тестовая последовательность содержит полный список команд для выполнения тестов. Один и тот же тест может выполняться несколько раз с различным начальным значением датчика случайных чисел (random seed). Эта же последовательность используется для сбора информации о покрытии кода и функционального покрытия с передачей необходимых опций симулятору. Разработана сетевая программа, позволяющая наблюдать результаты регрессионных тестовых последовательностей.

G. Отслеживание ошибок проекта и окружения

Программа отслеживания ошибок (Bug Tracking) содержит базу данных обнаруженных ошибок и позволяет убедиться, что все ошибки исправлены, а запросы на улучшение адресованы или отложены. Ошибка проходит путь от поступления запроса до исправления, проверки и закрытия проблемы. Для отслеживания ошибок и запросов на улучшение базовых и повторно используемых верификационных компонентов была использована свободно распространяемая программа Mantis Bug Tracker [6].

H. Проверочные списки

Несколько контрольных точек определены в течение верификационного процесса. В них проверяются результаты на соответствие критериям, заданным с помощью проверочных списков. Результаты проверки сохраняются. Списки и результаты проверки представлены в рассматриваемом процессе в виде таблиц Microsoft Excel. Проверочные списки содержат требования, касающиеся отдельных типовых компонентов устройства, или общие требования. Пример требования к интерфейсным верификационным компонентам: «для каждого интерфейса, где возможно, используйте ‘X’ для неиспользуемых в определенном режиме сигналов, например сигналов данных, если сигнал подтверждения valid не активен».

V. Выводы

Современная верификация проектов СБИС требует создания последовательного технологического процесса с использованием специальных инструментов и средств автоматизации. Наряду с использованием стандартных средств, в описанном подходе были реа-

лизованы усилия по созданию оригинальных компонентов инфраструктуры, повторно используемых компонентов, средств автоматизации построения верификационного окружения с использованием генераторов кода, инструментария сопровождения технологического процесса верификации. Затраты времени и ресурсов окупились за счет повышения качества разрабатываемых СБИС, уменьшения количества ошибок проектирования и снижения риска необходимости перепроектирования и повторного изготовления, а также за счет сокращения времени проектирования.

Благодарности

Автор выражает благодарность коллегам в компании Сиско Системс Инк., внесшим вклад в разработку последовательного и эффективного верификационного процесса, который автор имел честь представить. Особую благодарность автор выражает Энтони Цаю (Anthony Tsai), возглавляющему разработку и внедрение верификационной методологии для проверки блоков СБИС. Кроме того, автор благодарит своих российских коллег И.В. Ознобихина, доцентов и к.т.н. А.К. Полякова и В.В. Ерохина, профессора и д.т.н. Ю.А.Татарникова за помощь в подготовке статьи.

Литература

[1] URL: http://www.accellera.org/downloads/standards/uvm/uvm_users_guide_1.2.pdf (accessed 06.04.2016).
[2] URL: http://www.accellera.org/downloads/standards/systemrdl (accessed 06.04.2016).
[3] URL: http://www.synopsys.com/Tools/Verification/FunctionalVerification (accessed 06.04.2016).
[4] URL: http://www.dvteclipse.com/Verissimo_SystemVerilog_Testbench_Linter.html (accessed 06.04.2016).
[5] URL: https://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/vcs-xprop-ds.aspx(accessed 06.04.2016).
[6] URL: https://www.mantisbt.org/ (accessed 06.04.2016).

Practical Aspects of Design Verification of Complex Chips

A.Sokhatski

Cisco Systems Inc., asokhats@cisco.com

***Keywords* — Design Verification, RTL, SystemVerilog, UVM, SVA, System RDL.**

Abstract

Complex multi-billion up-to-date chips, with significant number of internal blocks require consistent Design Verification (DV) methodology across blocks and functional groups of blocks (clusters). It will simplify integration, horizontal and vertical reuse, switching resources and certainly will contribute in increasing quality and decreasing time to market. Consistent flow should cover DV from design spec review up to chip bring up in the lab and further support.

The paper considers constraint random simulation with further code, functional and assertion coverage closure as the main DV approach and goes through important practical aspects from the author's experience at Cisco Systems Inc.:

1) DV Environment basics focusing on what is done on the top of industry standard Universal Verification Methodology (UVM)[1]: environment structure, configuration, custom phases, company scope base classes layer, component reuse; special attention paid to components related to control plane register / memory access;

2) DV Environment Build Automation, including: assistance in building basic block DV environment components; generation of data structures for register access components set: register map, auto initialization code, interrupt error handler, end of test checker, background register access; generation done from shared source: registers description that is prepared in System RDL[2];

3) Important DV flow steps and tools used, including build and run, regression and release flow, linting[3].

As a result of consistent flow with tools support and development automation, we are getting back high quality chips and reduced development time.

References

[1] Universal Verification Methodology (UVM) 1.2 User's Guide. Available at: http://www.accellera.org/images//downloads/standards/uvm/uvm_users_guide_1.2.pdf (accessed 06.04.2016).
[2] SystemRDL v1.0: A specification for a Register Description Language. Available at: http://www.accellera.org/downloads/standards/systemrdl (accessed 06.04.2016).
[3] High Performance Functional Verification. Available at http://www.synopsys.com/Tools/Verification/FunctionalVerification (accessed 06.04.2016).
[4] Software Tools for Efficient Code Development and Analysis. Available at http://www.dvteclipse.com/Verissimo_SystemVerilog_Testbench_Linter.html (accessed 06.04.2016).
[5] VCS Xprop. Increasing the Efficiency of X-related Simulation and Design. https://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/vcs-xprop-ds.aspx (accessed 06.04.2016)
[6] MantisBT makes collaboration with team members & clients easy, fast, and professional. Available at: https://www.mantisbt.org/ (accessed 06.04.2016).