УДК 681.326:519.613

В. И. Хаханов, И. В. Хаханова, д-ра техн. наук,
Е. И. Литвинова, канд. техн. наук
Харьковский национальный университет радиоэлектроники
(Украина, 61166, Харьков, пр. Ленина, 14, фак-т КИУ,
тел. (057) 70-21-421, (057) 70-21-326,
E-mail: hahanov@kture.kharkov.ua, kiu@kture.kharkov.ua),
О. А. Гузь, канд. техн. наук
Донецкий ин-т автомобильного транспорта
(Украина, 83086, Донецк, пр. Дзержинского, 7, кафедра СКС,
тел. (063) 470-52-09, E-mail: olesya_guz@ukr.net.)

Тестирование и верификация HDL-моделей цифровых систем на кристаллах

(Статью представил д-р техн. наук В. П. Симоненко)

Предложена технология тестирования и верификации цифровых систем для диагностирования и исправления ошибок HDL-моделей, основанная на совместном использовании механизма (системы) ассерций и тестопригодного проектирования. Представлена универсальная модель программного компонента в виде транзакционного графа. Показаны логические функции тестопригодности HDL-моделей, полученные на основе его использования. Приведены примеры анализа тестопригодности методом подсчета управляемости и наблюдаемости транзакционного и управляющего графов.

Запропоновано технологію тестування і верифікації цифрових систем для діагностування та виправлення помилок HDL-моделей, яка базується на спільному використанні механізму (системи) асерцій і тестопридатного проектування. Наведено універсальну модель програмного компоненту у вигляді транзакційного графа. Показано логічні функції тестопридатності HDL-моделей, базовані на основі його використанні. Наведено приклади аналізу тестопридатності методом підрахунку керованості та спостережності транзакційного і керуючого графів.

К л ю ч е в ы е с л о в а: тестирование, тестопригодность, верификация, HDL-модель.

Ведущие компании мира, Cadence, Synopsis, Mentor Graphics, большое внимание уделяют созданию цифровых изделий на кристаллах, что практически не поддается автоматизации. На этой стадии процесс отладки программного кода занимает почти 70% общего времени, определяемого как time-to-market. Предлагаемая технология тестирования и верификации системных HDL-моделей (Hardware Description Language (HDL) — язык описания аппаратуры), ориентирована на существенное повышение ка-

чества проектируемых компонентов цифровых систем на кристаллах (yield — выход годной продукции) и уменьшение времени создания изделия (time-to-market) на основе использования среды моделирования, тестопригодного анализа логической структуры HDL-программы и механизма ассерций.

Данная технология позволяет осуществлять поиск ошибок с заданной глубиной в программном HDL-коде за приемлемое для разработчика время введением в критические точки программной модели ассерционной избыточности, вычисляемой с помощью синтезированных логических функций тестопригодности. Последние определяют качество программного кода с помощью синтеза дизъюнктивной нормальной формы (ДНФ), для которой оценка по Квайну формирует количественную характеристику тестопригодности компонента или программы, представленной последовательностью операторов. Используемые при проектировании и тестировании аппаратуры критерии тестопригодности (управляемость и наблюдаемость) применены для оценки качества программного кода. Цель — улучшение технологии тестирования и верификации цифровых систем для диагностирования и исправления ошибок HDL-моделей с помощью синтеза транзакционного графа (ТГ) программы, совместного использования механизма ассерций и технологий тестопригодного проектирования стандарта IEEE 1500.

Задачи исследования:

1. Классификация технологий тестопригодного проектирования и верификации системных HDL-моделей для создания цифровых систем на кристаллах.

2. Разработка обобщенной модели верификации и тестирования системной HDL-модели на основе использования ассерций как адаптация технологий IEEE 1500 стандарта к проверке программного HDL-кода.

3. Разработка метрики оценивания тестопригодности HDL-моделей на основе синтеза логических функций тестопригодности.

4. Применение модели ассерций для верификации IP-core фильтра дискретного косинусного преобразования.

5. Практические результаты и дальнейшие исследования.

Источники исследования:

1. Технологии и средства создания тестов и testbench (тестов проверки работоспособности) описаны в [1—3].

2. Модели и методы верификации системных моделей на основе механизма ассерций описаны в [4—7]. При тестопригодном проектировании программных продуктов использованы стандарты IEEE [8—10], а также инновационные решения для верификации и анализа тестопригодности системных HDL-моделей [11—18].

Тестопригодность программно-аппаратных продуктов. Инициирующим ядром появления новых технологий тестирования и верификации в программной и компьютерной инженерии следует считать силиконовый кристалл, являющийся основой для создания вычислительных и (или) коммуникационных устройств. Кристалл рассматривается как испытательный полигон для апробации новых средств и методов трассировки, размещения, синтеза и анализа компонентов. Технологические решения, выдержавшие испытания временем в микроэлектронике, далее трансформируются и адаптируются к макроэлектронике, представленной компьютерными системами и сетями. Приведем некоторые исторические факты, связанные с преемственностью развития и взаимопроникновения технологических инноваций в программных и аппаратных продуктах.

1. Стандарты граничного сканирования [4, 6, 18] на уровне платы и кристалла привели к появлению механизма ассерций для тестирования и верификации программных продуктов [12—18].

2. Метрика анализа тестопригодности [1, 2] (управляемости и наблюдаемости) цифровых структур адаптирована к оценке программного кода для определения критических мест и последующего улучшения программного продукта относительно уменьшения времени верификации и повышения его качества.

3. Технологии анализа качества покрытия [18] наперед заданных неисправностей тестовыми последовательностями использованы при создании таблицы покрытия функциональностей программных продуктов модулем testbench для оценки достоверности тестов, установления диагноза и исправления ошибок.

4. Графовые модели регистровых передач [15—17] использованы при тестировании программных продуктов, которые с помощью структурно-логического анализа приведены к более технологичной форме. Таковым является ТГ как модель HDL-кода, записываемый в виде алгебраической формы для подсчета тестопригодности при определении критических компонентов (точек) программы для установки ассерций.

5. Разделение автомата на управляющую [1, 7] и операционную части применено для упрощения процесса синтеза testbench и верификации программного кода на основе синтеза графов управления и транзакционной передачи данных.

6. Кривая жизненного цикла аппаратного изделия [8, 9] адекватно отображает временные стадии изменения yield при создании, тиражировании и сопровождении программного продукта.

7. Платформенно-ориентированный синтез [3] HDL-кода с использованием существующих наборов компонентов под управлением GUI (Graphi-

cal User Interface — графический интерфейс пользователя) изоморфен технологии объектно-ориентированного программирования на основе использования наработанных ведущими компаниями библиотек. Применение технологии ESL (Electronic System Level — технологии системного проектирования на основе использования библиотек) [3, 11] в программировании дает возможность использовать программные компоненты готовых функциональностей из базовых библиотек, используемых для создания новых программных продуктов. В этом случае основная процедура проектирования заключается в выполнении мэппинга, который ориентирован на покрытие функций спецификации существующими компонентами, где новый код составляет не более 10 % проекта.

8. Понятие теста проверки неисправностей или функциональностей, используемое для тестирования аппаратных проектов применено в качестве testbench [12—18] для верификации и отладки программных продуктов, представленных описаниями на уровне системных языков (C++, SystemVerilog, Vera, e).

9. Платформенно-ориентированный синтез [3] testbench с использованием существующих библиотек тестов (ALINT) для многократно используемых, стандартизованных функциональных компонентов применен для генерирования тестов программных модулей на основе наработанных библиотек от ведущих компаний мира.

10. Стандартные решения сервисного обслуживания функциональностей F-IP в рамках инфраструктуры I-IP [4—8] использованы при создании инфраструктуры встроенной верификации компонентов программного продукта для устранения ошибок дефектного программного модуля.

11. Обеспечение двумерности структуры взаимосвязанных функциональных компонентов (IP-cores) создаваемого программного продукта ориентировано на использование мультиядерных архитектур для технологичного распараллеливания вычислительных процессов тестирования и верификации [3, 4, 13, 17], что существенно уменьшает параметр time-to-market.

12. Создание адресного пространства для функциональностей SoC (System-on-a-Chip — система на кристалле), реализованных в аппаратном и программном исполнении [9—11], обеспечивает цифровой системе свойство самовосстановления работоспособности программных и аппаратных компонентов при использовании альтернативных или избыточных ресурсов инфраструктуры сервисного обслуживания I-IP. Примером могут быть мультипроцессорные аппаратные продукты, устойчивые к возникающим дефектам. При этом отказавший адресуемый компонент может быть заменен резервным в процессе выполнения функции. Свойство адре-

суемости используется при создании критических программных продуктов, где наличие адресуемых диверсных (мультиверсных) резервных компонентов обеспечивает отказоустойчивость системы при возникновении дефектов.

13. Интересной и привлекательной для рынка является проблема автономного внутрикристального встроенного тестирования, диагностирования и ремонта без применения внешних средств [9—11], которой занимаются все ведущие компании. К решению проблемы привлечены современные беспроводные и Internet технологии дистанционного сервисного обслуживания. Однако несанкционированный доступ к содержимому кристалла на расстоянии может привести к нежелательным деструктивным последствиям и выходу из строя цифрового изделия. Тем не менее, специфика современных цифровых систем на кристаллах состоит в удивительной возможности исправлять ошибки на расстоянии, благодаря наличию связи кристалла с внешним миром посредством Internet или беспроводных технологий (wi-fi, wi-max, bluetooth, satellite), присутствующих в силиконе. Дистанционная коррекция программных ошибок возможна при использовании памяти (занимающей до 94 % силикона) SoC для хранения программ, куда можно, в случае обнаружения некорректности, записать новый код, не имеющий ошибок. Дистанционная коррекция аппаратных ошибок стала возможной при использовании программируемых логических интегральных схем, куда можно, в случае обнаружения неисправности, записать новый битовый поток, не имеющий ошибок, который фактически создает новую аппаратуру путем повторного программирования кристалла.

Сближение и взаимопроникновение технологий приводит к изоморфным методам проектирования, тестирования и верификации относительно программных и аппаратных комплексов, что, по существу, является закономерным процессом ассимиляции прогрессивных концепций и решений. Этому способствует тот факт, что наиболее важные параметры жизненного цикла изделия, такие как time-to-market и yield, становятся соизмеримыми по времени и объему выхода годной продукции. Кривая жизненного цикла аппаратного изделия, представленная на рис. 1, с точностью до изоморфизма отображает временные этапы программного продукта, который проходит аналогичные стадии: проектирование, увеличение объема выпуска продукции, производство с доработкой и сопровождением изделия.

В контексте жизненного цикла существуют две актуальные проблемы, которые связаны с поднятием кривой вверх по оси ординат, а также с компрессией этой кривой по временной оси, означающей уменьшение

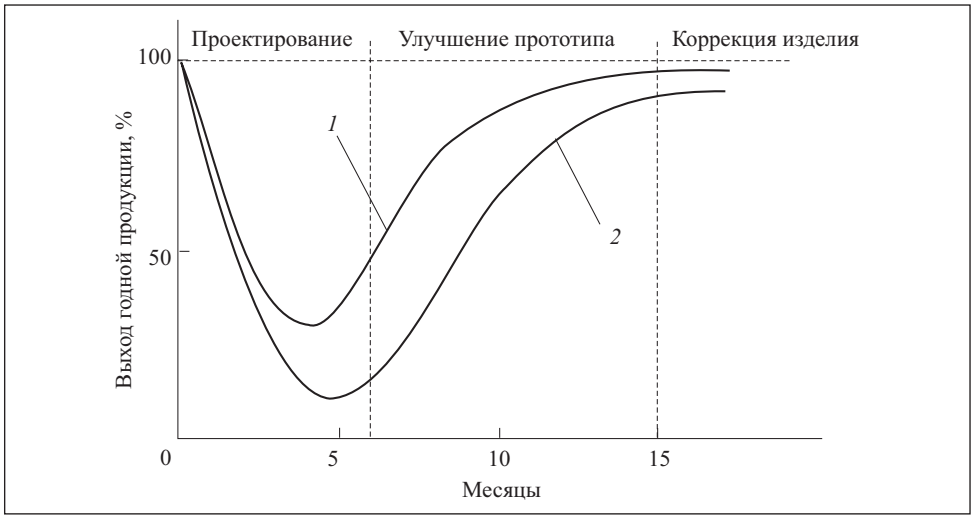


Рис. 1. График жизненного цикла программно-аппаратного комплекса: *1* — внедрение новых технологий проектирования и производства; *2* — функциональная зависимость создания изделия

параметра time-to-market. Как видно на рис. 1, увеличение выхода годной продукции происходит на всех стадиях:

1) проектирование — в результате быстрого устранения ошибок с помощью внедрения новых технологий;

2) прототипирование — в результате исправления кода, имплементированного в память системы на кристалле;

3) производство — в результате выпуска программ, корректирующих ошибки посредством Internet или спутников.

Согласно [12—18] стоимость верификации программно-аппаратных продуктов на основе ASIC (Application-Specific Integrated Circuit — специализированная интегральная микросхема), IP-core (Intellectual Property Core — отлаженный функциональный аппаратный модуль, готовый к имплементации в кристалл) и SoC составляет 70 % общих затрат проектирования. Приблизительно 80 % общей длины формального описания проекта составляет размерность testbench-кода. Задачи, решаемые в процессе верификации, связаны с устранением ошибок проектирования на возможно более ранней его стадии для приведения кода прототипа в соответствие с его спецификацией. Пропуск ошибки увеличивает ее стоимость на порядок при переходе от уровня проектирования блока до уровня кристалла и далее к системе.

Введение в проект программной избыточности — механизма асерций [15—17] — позволяет выполнять анализ основных специфициро-

ванных условий в процессе моделирования проекта и диагностировать ошибки на ранних стадиях проектирования программы (C++, System Verilog, Vera) или аппаратуры (HDL).

Инфраструктура процесса верификации и тестирования проекта. Модель процесса верификации проекта на системном уровне можно представить в виде обобщенного уравнения диагноза $T \oplus S = L$, или более подробно для обнаружения ошибок в программных компонентах:

$$(T, F) \oplus (S, A) = L_s.$$

Здесь T и F — тестовые воздействия и функциональное покрытие эталонной модели с ожидаемыми реакциями; S и A — проверяемая HDL-модель и механизм ассерций для верификации и точного диагностирования ошибок в программном коде. Тестирование аппаратной реализации основано на использовании логической операции $\oplus$ — суммы по mod 2 (Xor) — для аналитического выражения $(T) \oplus (S, B) = L_h$, где B — избыточность в виде регистра граничного сканирования от стандарта IEEE 1500, используемая в качестве дополнения к функциональной модели для обеспечения требуемой глубины диагностирования компонентов цифровой системы на кристалле. При этом L_s и L_h — списки ошибок, получаемые на стадиях верификации проекта и тестирования готового цифрового изделия.

Для понимания соотношений между ключевыми понятиями — верификация, валидация, ассерция — введены следующие определения [13—15].

Верификация — процесс анализа системы или компонентов для определения корректности формальных преобразований входного описания на каждой стадии проектирования.

Валидация — процесс определения работоспособности системы и ее компонентов проверкой соответствия требованиям спецификации после выполнения каждой стадии проектирования.

Сертификация — экспертная (юридическая) гарантия валидности системы (компонентов) требованиям спецификации при ее использовании по назначению.

Ассерция — HDL-высказывание системного уровня, предназначенное для раннего определения ошибок проектирования относительно требований спецификации при моделировании проекта на тестовых воздействиях до и после выполнения синтеза.

Типы и свойства ассерций:

1. Предельное число ассерций есть диверсная модель проекта, на практике число ассерционных операторов составляет 5 % кода проекта.

2. Мгновенные или однотактные (immediate) ассерции вставляются в текст верифицируемой программы в форме инверсных if-операторов и выполняются в общем порядке следования строк кода.

3. Параллельные ассерции (циклические или многотактные) моделируются параллельно и независимо от исполняемой модели проекта. Вычислительный процесс организуется в циклы моделирования, что позволяет анализировать состояние проекта в моменты времени, определенные синхроимпульсами ввода ассерций. Assert-операторы находятся также в теле программы. Параллельная ассерция защищена от состязаний сигналов в модели наличием временного интервала симуляции, который задает моменты считывания (preponed) сигналов и вычисления (postponed) ассерции.

4. Внешние ассерции представляют собой диверсную модель проекта в виде отдельного модуля, записанного на языках System Verilog или PSL [13]. Внешние ассерции наблюдают за сигналами портов функционального модуля и передают их из проекта в тестовую среду.

5. Ассерции, в общем случае, не являются синтезируемым подмножеством HDL-языков. Более того, механизм ассерций, функциональные покрытия и генераторы тестов, направленные на выявление ошибок проектирования, формируют языковую среду HVL (Hardware Verification Language) верификации проектов.

Формально ассерция записывается в виде предиката $A_i = T_i \oplus S_i \rightarrow$ $\rightarrow d(A_i)$, который на множестве состояний эталонной и реальной моделей принимает значения {true — 1, false — 0, don't care — X}. Здесь X — неопределенное состояние ассерции или ее отсутствие для анализируемого программного компонента. В соответствии с упомянутым определением и по функциональной реакции $d(A_i)$ на сравнение состояний $T_i \oplus S_i$ можно дифференцировать ассерции трех типов: $d(A_i) = \{A^c, A^g, A^w\}$, т.е. прекращение верификации HDL-модели, переход к проверке следующего программного блока, выдача сообщения на консоль. Программный блок — упорядоченная совокупность операторов или строк языка, ориентированная на выполнение отдельной функциональности или ее части в пределах одной программы.

Механизм ассерций — система высказываний, представленная в виде вектора $\mathbf{A} = (A_1, A_2, ..., A_i, ..., A_n)$, а также средства их анализа $d(A)$, предназначенные для верификации и поиска ошибок HDL-модели системного уровня в пространстве и во времени. Цель введения механизма ассерций — получение точного диагноза для устранения семантических ошибок HDL-кода программы. Вектор ассерций ориентирован на проверку функциональности и логического состояния компонентов HDL-модели во времени и пространстве. Такие же функции, т.е. углубление диагностирования, выполняют стандарты граничного сканирования (IEEE 11.49, IEEE 1500), где тестирование компонентов цифровой системы регулируется контроллером тестового доступа.

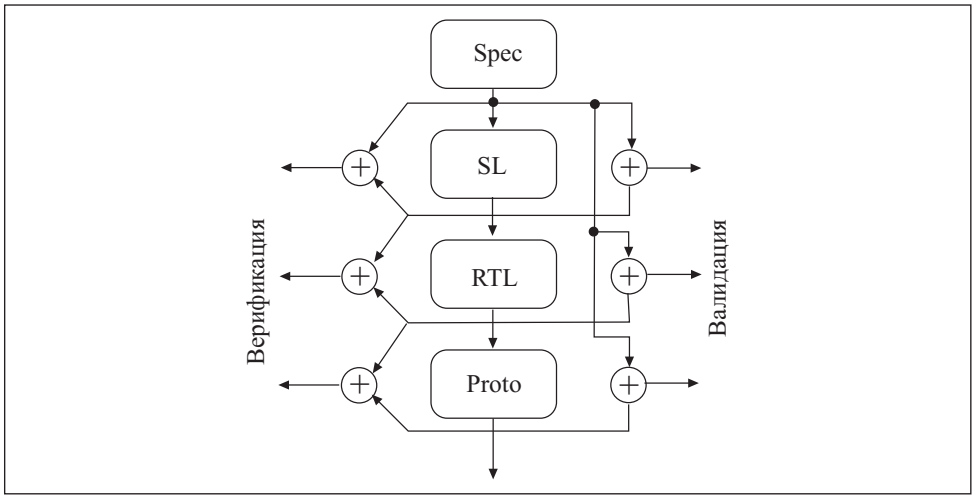


Рис. 2. Взаимодействие процессов валидации и верификации: Spec — спецификация; SL (system level) — системный уровень; RTL (register transfer level) — уровень регистровых передач; Proto (prototype) — прототип

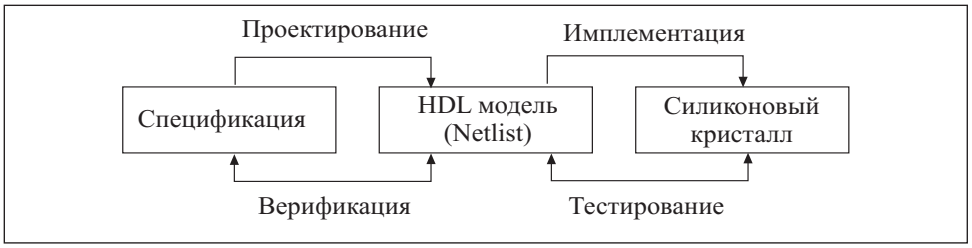


Рис. 3. Стратегия разработки проекта

Структурная модель процесса проектирования относительно взаимодействия процедур валидации и верификации представлена на рис. 2. Следует заметить, что верификация определяет корректность выполненных преобразований на рассматриваемом этапе проектирования, а валидация — соответствие технического состояния системы требованиям спецификации на каждой стадии проектирования.

Стратегии верификации и тестирования имеют различные модели приложения технологий, ориентированных на сокращение параметра time-to-market. Итеративный процесс верификации ориентирован на исправление ошибок HDL-модели системного уровня, полученной по спецификации изделия (рис. 3). Конечным результатом является описание связей компонентов цифровой структуры (netlist) или отлаженная HDL-модель регистрового уровня. Следующий итеративный процесс — синтез и имплементация

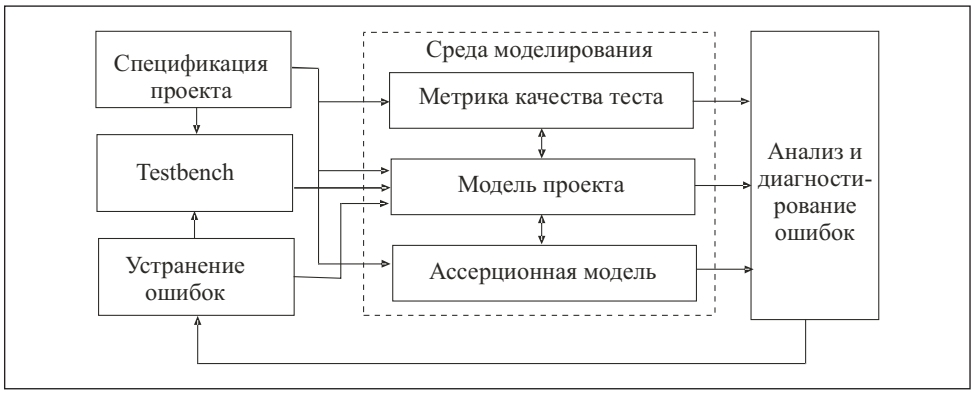


Рис. 4. Среда верификации проекта

проекта в силиконовый кристалл. Здесь тестированием проверяется корректность аппаратной имплементации HDL-модели в регистровый или вентильный уровень описания проекта в микросхеме программируемой логики. Для кристаллов ASIC такая технология нецелесообразна, поскольку перепрограммирование ошибки стоит почти миллион долларов.

Согласно приведенным определениям и пояснениям модель среды или макропроцесса верификации программной стадии проекта ориентирована на уменьшение времени создания изделия и повышение уровня выхода годной продукции в результате использования избыточности кода в виде механизма ассерций и testbench совместно с метрикой определения качества теста или функциональной полноты.

Инфраструктура тестирования и верификации HDL-модели представлена на рис. 4. Спецификация проекта, описанная на формальном языке высокого уровня, является исходной информацией для следующего:

создания метрики оценивания качества теста в виде функционального покрытия;

HDL-модели проекта;

теста с эталонными реакциями — testbench, ассерционной структуры, являющейся дополнением к основной модели, что необходимо для ускорения проверки и отладки проекта.

Среда верификации представлена системой моделирования, блоком тестирования Testbench, механизмом ассерций и собственно системным кодом модели на языках VHDL, Verilog, System Verilog. Модуль Testbench задает входные стимулы и эталонные реакции на них, записанные на HDL-языках и ориентированные на проверку функциональностей (переменные, функции, последовательности), параметры которых определяются в корзине функционального покрытия.

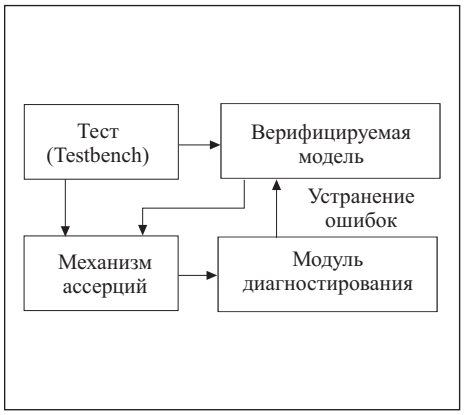


Рис. 5. Схема технологии использования ассерций для верификации проекта

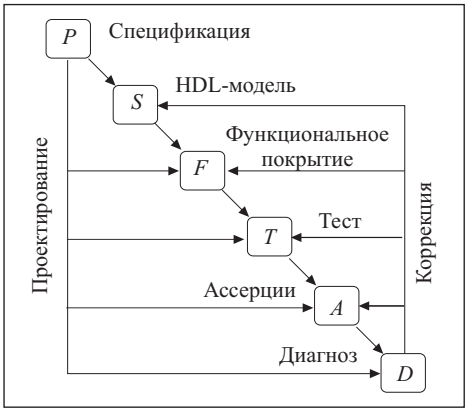


Рис. 6. Маршрут верификации проекта

Механизм ассерций — модельная избыточность, дополняющая testbench при проверке внутренних по времени и пространству состояний проекта, представленная высказываниями на входе-выходе и предназначенная для ускорения тестирования, верификации, диагностирования и исправления ошибок проектирования в системном коде. Ассерции можно сгенерировать не только по спецификации, но и по HDL-модели, убирая ненужные конструкции, а остальные — модифицировать к форме ассерций. При этом существует вероятность повторения в ассерции программной ошибки HDL-модели, которая не будет идентифицирована в процессе моделирования.

Упрощенная структура взаимодействия механизма ассерций с другими компонентами среды верификации и диагностирования представлена на рис. 5. Testbench представляет собой эталонную модель объекта проектирования в виде входных и выходных реакций. Часто вместо Testbench используется уже проверенная и доступная модель другой компании, относительно которой проверяется верифицируемая модель (MUV — Model Under Verification). В этом случае необходимо иметь генератор входных последовательностей или testbench без выходных реакций. Блок механизма ассерций является надстройкой для testbench верифицируемой модели и предназначен для сравнения результатов моделирования при формировании двоичного вектора экспериментальной проверки, в качестве которого используется вектор (состояний) ассерций:

$$\mathbf{A} = (A_1,\ A_2,\ ...,\ A_i,\ ...,\ A_n),\quad A_i = T_i \oplus S_i,\quad A_i = \{0, 1, X\}.$$

Несмотря на то, что ассерционные операторы могут присутствовать в теле программы Testbench или MUV, они не являются принадлежностью упомянутых программ. Assert-операторы обрабатываются отдельно и параллельно с моделированием MUV, поэтому они не оказывают влияния на функционирование HDL-модели проекта. Кроме того, ассерции не являются синтезируемыми и заканчивают свое существование после отладки системного кода. Имея аналоги операторов в HDL-языках в виде условных команд, assert-операторы предназначены для сравнения технических состояний компонентов эталонной и проектируемой моделей при последующем принятии решения о продолжении процесса моделирования, прекращении или переходе к выполнению анализа другого фрагмента программного кода.

Представляет интерес последовательность действий при создании среды верификации, где единственным аргументом является спецификация проекта, все остальное — производные от нее. На рис. 6 представлена структура взаимосвязей процессов проектирования и диагностирования для исправления ошибок в HDL-коде программы.

Если выполнены условия тестопригодности и правильно расставлены ассерции в критических точках программного кода для диагностирования всех компонентов, то процедура совместного моделирования механизма ассерций и HDL-модели может однозначно идентифицировать последовательность строк программного кода с семантической ошибкой. Testbench и вектор (матрица) ассерций должны быть созданы другим разработчиком независимо от HDL-модели. Это обеспечивает диверсификацию упомянутых моделей относительно единой спецификации, а также обнаружение и исправление ошибок в HDL-проекте.

Таким образом, в процессе создания системного кода проекта разработчик должен ориентироваться на технологичные и простые процедуры процесса верификации HDL-модели, т.е. выполнять достаточно простые условия тестопригодности (см. рис. 5, 6). При этом должны быть решены следующие задачи:

1. Написание Testbench и генерирование функциональных покрытий.
2. Определение критических точек программного кода для установки ассерций.
3. Корректное написание ассерций.
4. Создание эффективных алгоритмов диагностирования семантических ошибок в HDL-коде на основе анализа системы ассерций как реакции на тест эталонной и реальной моделей в процессе моделирования проекта.

Аналитическая модель инфраструктуры верификации. Для идентификации обобщенного состояния HDL-модели формируются эталонные реакции (сигнатуры) критических точек (переменные, регистры, память) во времени и в пространстве. Затем выполняется анализ HDL-модели для

ее диагностирования путем сравнения эталонных и экспериментальных реакций (сигнатур) для формирования ассерционного вектора, описывающего сравнение состояний (эталонного и реального) компонентов объекта во времени и в пространстве. Целесообразно формировать ассерционный вектор независимо от HDL-модели проекта. Ассерционная и функциональная модели обрабатываются средой моделирования параллельно и независимо одна от другой.

Ассерционная модель определяет отклонения в поведении объекта в наиболее важных точках пространственно-временной эталонной структуры. Формат ассерций должен соответствовать формату HDL-модели проекта. Ассерции ориентированы на диагностирование семантических ошибок HDL-модели при использовании Testbench и MUV. Аналитическая модель инфраструктуры верификации имеет следующий вид:

$$M = \{P, S, A, T, F, d, C\}.$$

Здесь P — спецификация проекта; S — программная модель,

$$S = f_1(P) = (S_1, S_2, ..., S_i, ..., S_n);$$

F — корзина покрытия функциональностей,

$$F = f_2(P, S) = (F_1, F_2, ..., F_i, ..., F_n);$$

T — Testbench,

$$T = f_3(P, S, F) = (T_1, T_2, ..., T_i, ..., T_n);$$

A — ассерционная модель,

$$A = f_4(P, S, F, T) = (A_1, A_2, ..., A_i, ..., A_n);$$

d и C — модуль и условия диагностирования ошибок,

$$d = f_5(P, S, F, T, A) = (L_1, L_2, ..., L_i, ..., L_n) \in \{L_s, L_h\},$$

$$C = \left[\bigcup_{i=1}^{n} F_i \in F = P\right] \wedge \left[\bigcup_{i=1}^{n} T_i \in T = F\right], \quad (1)$$

где

$$L_s = (T, F) \oplus (S, A); \quad (2)$$

$$L_h = (T) \oplus (S, B). \quad (3)$$

Формула (1) определяет условия полноты проверки функциональностей тестом (testbench) относительно спецификации. Уравнение (2) задает функцию вычисления ошибок проектирования при переходе от системного к регистровому уровню с использованием атрибутов верификационной инфраструктуры. Равенство (3) регламентирует нахождение неисправностей при эксплуатации цифровой системы на кристалле.

Ассерционная избыточность является функцией от критических точек HDL-модели, предельное число которых может быть равно числу временных фреймов функциональных компонентов, определенных спецификацией. Априорно координатам вектора ассерций присваивают значения *X*. Затем определяют критические координаты, число которых будет достаточным для проведения верификационного эксперимента при поиске ошибочных программных блоков с заданной глубиной диагностирования. Такие координаты идентифицируются единицами. В процессе моделирования координаты вектора модифицируются в сторону уменьшения единиц. Каждой координате вектора **A** ставится в соответствие список всех вершин предшественников ТГ программного кода. Координатам вектора соответствует матрица достижимостей ТГ или списки вершин-предшественников, заданные в любой другой форме. По фактическому двоичному состоянию элементов вектора **A** выполняется безусловная процедура диагностирования списка *L* неисправных программных блоков $d(A)$, определяемая следующими выражениями [18]:

$$L_s(A)=\left(\bigcap_{\forall i(A_i=1)} A_i\right)\Big\backslash\left(\bigcup_{\forall i(A_i=0)} A_i\right); \tag{4}$$

$$L_m(A)=\left(\bigcup_{\forall i(A_i=1)} A_i\right)\Big\backslash\left(\bigcup_{\forall i(A_i=0)} A_i\right). \tag{5}$$

Система уравнений (4), (5) предназначена для поиска одиночных и кратных ошибок при использовании вектора ассерционных состояний. Длина ассерционного вектора равна числу вершин в графе или числу программных блоков в функционально-логической структуре HDL-кода. Векторная модель среды верификации имеет следующий вид:

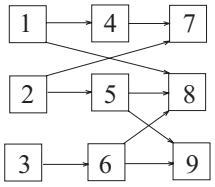
$$\begin{vmatrix} T_1 \\ T_2 \\ T_i \\ T_n \end{vmatrix} \oplus \begin{vmatrix} S_1 \\ S_2 \\ S_i \\ S_n \end{vmatrix} = \begin{vmatrix} A_1 \\ A_2 \\ A_i \\ A_n \end{vmatrix} \overset{d}{\rightarrow} \begin{vmatrix} L_1 \\ L_2 \\ L_i \\ L_n \end{vmatrix}. \tag{6}$$

В процессе моделирования выполняется сравнение реакций Testbench и HDL-модели, что формирует состояния координат ассерционного вектора:

$$\mathbf{A}_i = f\,(T_i, S_i) = T_i \oplus S_i,\ \ \mathbf{A}_i = \{0, 1, X\}.$$

Затем к существенным {0, 1}-координатам вектора ассерций применяют операцию конъюнкции для получения списка программных блоков с ошибками, выполняя одну из процедур, определенных в (4), (5).

Рассмотрим технологию диагностирования неисправных блоков на следующем примере. Пусть имеется функционально-логический граф HDL-модели



(7)

Структуру взаимосвязей графа представим следующей матрицей достижимостей:

M	1	2	3	4	5	6	7	8	9
1	1								
2		1							
3			1						
4	1			1					
5		1			1				
6			1			1			
7	1	1		1			1		
8	1	1	1		1	1		1	
9		1	1		1	1			1

Векторная модель диагностирования, заданная выражением (6), содержит списки блоков-предшественников для каждой вершины графа, полученной из матрицы достижимостей. Каждой вершине графа ставится в соответствие ассерция, которая в процессе моделирования может быть доопределена значением {0,1}. В данном случае векторная модель поиска ошибочных программных блоков (6) для графа (7) трансформируется к виду

$$\begin{vmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{vmatrix} \oplus \begin{vmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \\ S_9 \end{vmatrix} = \begin{vmatrix} A_1 = X \\ A_2 = X \\ A_3 = X \\ A_4 = 0 \\ A_5 = X \\ A_6 = X \\ A_7 = 1 \\ A_8 = 1 \\ A_9 = X \end{vmatrix} \overset{d}{\rightarrow} \begin{vmatrix} L_1 = S_1 \\ L_2 = S_2 \\ L_3 = S_3 \\ L_4 = S_1, S_4 \\ L_5 = S_2, S_5 \\ L_6 = S_3, S_6 \\ L_7 = S_1, S_2, S_4, S_7 \\ L_8 = S_1, S_2, S_3, S_5, S_6, S_8 \\ L_9 = S_2, S_3, S_5, S_6, S_9 \end{vmatrix}.$$

В результате выполнения диагностирования по системе (4), (5), состоящей в пересечении всех неисправных компонентов, соответствующих единич-

ным координатам вектора ассерций, с последующим вычитанием объединения всех неисправных модулей, соответствующих нулевым координатам вектора **A**, получаем список дефектных программных блоков (при условии существования только одного ошибочного модуля):

$$L_s(A_4=0, A_7=1, A_8=1) = L_7 \cap L_8 \setminus L_4 =$$
$$= S_1, S_2, S_4, S_7 \cap S_1, S_2, S_3, S_5, S_6, S_8 \setminus S_1, S_4 = S_2.$$

Используя уравнение (5), можно получить список всех программных блоков, которые могут иметь ошибки, при условии существования нескольких дефектных компонентов:

$$L_m(A_4=0, A_7=1, A_8=1) = L_7 \cup L_8 \setminus L_4 =$$
$$= S_1, S_2, S_4, S_7 \cup S_1, S_2, S_3, S_5, S_6, S_8 \setminus S_1, S_4 = S_2, S_3, S_5, S_6, S_7, S_8.$$

В следующем примере исключим из рассмотрения первые два вектора, не существенные для процесса обработки списков неисправных блоков на основе анализа координат ассерционного вектора. Процедура вычисления списка одиночных дефектов имеет следующий вид:

$$\left|\begin{array}{c} A_1=X \\ A_2=X \\ A_3=X \\ A_4=X \\ A_5=0 \\ A_6=X \\ A_7=0 \\ A_8=1 \\ A_9=1 \end{array}\right| \overset{d}{\rightarrow} \left|\begin{array}{c} L_1=S_1 \\ L_2=S_2 \\ L_3=S_3 \\ L_4=S_1, S_4 \\ L_5=S_2, S_5 \\ L_6=S_3, S_6 \\ L_7=S_1, S_2, S_4, S_7 \\ L_8=S_1, S_2, S_3, S_5, S_6, S_8 \\ L_9=S_2, S_3, S_5, S_6, S_9 \end{array}\right| \left|\begin{array}{c} (L_8 \cap L_9) - \\ -(L_5 \cup L_7) \end{array}\right| \left|\begin{array}{c} S_3, S_6 \end{array}\right|.$$

Для множественных неисправных блоков на таком же векторе ассерций содержится большее число дефектных модулей:

$$\left|\begin{array}{c} A_1=X \\ A_2=X \\ A_3=X \\ A_4=X \\ A_5=0 \\ A_6=X \\ A_7=0 \\ A_8=1 \\ A_9=1 \end{array}\right| \overset{d}{\rightarrow} \left|\begin{array}{c} L_1=S_1 \\ L_2=S_2 \\ L_3=S_3 \\ L_4=S_1, S_4 \\ L_5=S_2, S_5 \\ L_6=S_3, S_6 \\ L_7=S_1, S_2, S_4, S_7 \\ L_8=S_1, S_2, S_3, S_5, S_6, S_8 \\ L_9=S_2, S_3, S_5, S_6, S_9 \end{array}\right| \left|\begin{array}{c} (L_8 \cup L_9) - \\ -(L_5 \cup L_7) \end{array}\right| \left|\begin{array}{c} S_3, S_6 \\ S_8, S_9 \end{array}\right|.$$

Интересны и другие формы описания векторной модели диагностического эксперимента. Например матрица достижимостей, вставленная в модель диагностирования, где ассерции представлены троичным вектором, имеет вид

A	L	1	2	3	4	5	6	7	8	9
X	L_1	1								
X	L_2		1							
X	L_3			1						
X	L_4	1			1					
0	L_5		1			1				
X	L_6			1			1			
0	L_7	1	1		1			1		
1	L_8	1	1	1		1	1		1	
1	L_9		1	1		1	1			1
	L_s			1			1			

$$L_s = (L_8 \wedge L_9) \wedge \wedge (\overline{L_5 \vee L_7}) = = (001001000) = = S_3, S_6$$

Здесь поиск программных блоков выполнен при условии существования в проекте одного неисправного модуля, который записан в последней строке. Процедура диагностирования на основе ассерционного вектора **A** при выполнении векторных операций конъюнкции, дизъюнкции и отрицания определена в правой части матрицы. Аналогичные вычисления для случая существования кратных дефектов в программных модулях проекта дают следующий результат:

A	L	1	2	3	4	5	6	7	8	9
X	L_1	1								
X	L_2		1							
X	L_3			1						
X	L_4	1			1					
0	L_5		1			1				
X	L_6			1			1			
0	L_7	1	1		1			1		
1	L_8	1	1	1		1	1		1	
1	L_9		1	1		1	1			1
	L_m			1			1		1	1

$$L_m = (L_8 \vee L_9) \wedge \wedge (\overline{L_5 \vee L_7}) = = (001001011) = = S_3, S_6, S_8, S_9$$

На практике результат диагностирования гарантирует наличие хотя бы одного дефектного блока из списка компонентов, подозреваемых в наличии неисправностей, определенных в L_s, L_m [18].

Представляет интерес для диагностирования HDL-модели по структурно-логическому (транзакционному) графу также алгебраическая форма описания графовых структур [18]. Ее преимущества заключаются в компактности задания матрицы достижимостей, которая обладает свойством структуризации графа в виде задания всех путей, принадлежащих каждой вершине. Кроме того, сочетания дефектных компонентов, определяемые конъюнктивными термами, дают более точный результат диагностирования по сравнению с заданием неисправных модулей в виде неупорядоченного множества элементов. Для структуры, представленной графом (7), алгебраическая форма графа и вычисление списка одиночных неисправностей имеют следующий вид:

A	L
X	$L_1 = S_1$
X	$L_2 = S_2$
X	$L_3 = S_3$
X	$L_4 = S_1\, S_4$
0	$L_5 = S_2\, S_5$
X	$L_6 = S_3\, S_6$
0	$L_7 = S_1 S_4 S_7 \vee S_2 S_7$
1	$L_8 = S_1 S_8 \vee S_2 S_5 S_8 \vee S_3 S_6 S_8$
1	$L_9 = S_2 S_5 S_9 \vee S_3 S_6 S_9$
	$L_s = S_3, S_6$

$$\left| \begin{array}{c} L_s = (L_8 \wedge L_9) - (L_5 \vee L_7) = \\ = (S_2 S_5 \vee S_3 S_6) - S_2 S_5 = \\ = S_3 S_6 \end{array} \right| .$$

Процедура анализа логических функций графовой структуры содержит три пункта:

1. Все термы и переменные в ДНФ, соответствующей нулевому значению ассерционной координаты, равны нулю.

2. В ДНФ, соответствующих единичному значению ассерционной координаты, левая часть конъюнктивного терма, включая переменную, ранее равную нулю, удаляется.

3. Для единичных функций выполняется пересечение (объединение) оставшихся термов, если выполняется диагностирование одиночных (кратных) неисправных блоков.

Такие преобразования ДНФ специальным моделированием нулевых сигналов переменных в единичных, относительно ассерций, логических

функциях представления графа для получения списка неисправных программных модулей приведены в правой части алгебрологической модели диагностирования.

Следующий пример также подтверждает состоятельность анализа логических функций для вычисления множественных неисправных блоков:

A	L
X	$L_1 = S_1$
X	$L_2 = S_2$
X	$L_3 = S_3$
X	$L_4 = S_1 S_4$
0	$L_5 = S_2 S_5$
X	$L_6 = S_3 S_6$
0	$L_7 = S_1 S_4 S_7 \vee S_2 S_7$
1	$L_8 = S_1 S_8 \vee S_2 S_5 S_8 \vee S_3 S_6 S_8$
1	$L_9 = S_2 S_5 S_9 \vee S_3 S_6 S_9$
	$L_m = S_3, S_6, S_8, S_9$

$$L_m = (L_8 \vee L_9) - (L_5 \vee L_7) = (S_1 S_8 \vee S_2 S_5 S_8 \vee S_3 S_6 S_8 \; S_2 S_5 S_9 \vee S_3 S_6 S_9) - (S_2 S_5 \vee S_1 S_4 S_7 \vee S_2 S_7) = S_3 S_6 S_8 \vee S_3 S_6 S_9 .$$

Предложенная технология верификации является достаточно простой, ориентирована на обработку HDL-моделей большой размерности, включающих тысячи строк кода, описывающего системные структуры на языках VHDL, Verilog. Для реализации технологии верификации необходимо сгенерировать ТГ, описывающий структуру программного кода в компонентах и операторах HDL-языка.

Анализ тестопригодности программных HDL-моделей. Достаточно существенная избыточность HDL-модели позволяет эффективно использовать ее для повышения тестопригодности структуры разработанного или написанного кода. Существующие стандарты тестопригодного проектирования аппаратуры можно адаптировать для верификации HDL-кода системных и регистровых программных моделей. Для этого используют граф регистровых или транзакционных передач [10], предоставляющий пользователю информацию о взаимосвязях булевых и регистровых переменных, памяти и интерфейсных шинах, называемых транзакторами. Данные для синтеза графа получают при анализе синтаксиса строк HDL-кода в целях установления источника и приемника информации, которые являются вершинами (транзакторами) графа, соединенными ориентированной дугой. При этом должно быть учтено число операторов, нагруженных на данную дугу. Построенный таким образом ТГ [18] покрывает все функциональности (транзакции) программной модели и задает все связи между

транзакторами, которые соответствуют приему, передаче и преобразованию информации. Для интегрального оценивания тестопригодности ТГ вводят следующие критерии:

$$Q = \frac{Z(S)}{Z(S)+Z(F)+Z(B)+Z(A)} \frac{1}{n} \sum_{i=1}^{n} (U_i \times N_i);$$

$$U_i = \frac{1}{T} \sum_{j=1}^{x_i} T_j^i \frac{1}{d_i^x \vee t_i^x}; \; N_i = \frac{1}{T} \sum_{j=1}^{y_i} T_j^i \frac{1}{d_i^y \vee t_i^y}, \tag{8}$$

где n — число вершин ТГ; x_i — число команд, формирующих доступ к входной вершине; y_i — число команд, определяющих вершину как источник данных. Тестопригодность (8), зависит от управляемости U, наблюдаемости N, а также от стоимости реализации Z следующих компонентов: метрики функционального покрытия F, testbench B, механизма ассерций A, функциональности S.

Управляемость и наблюдаемость есть метрика оценивания тестопригодности не соединительных линий, а компонентов HDL-кода, таких как регистр, счетчик, память или массивы, шины входа-выхода, векторы, логические или арифметические переменные. Управляемость вершины имеет функциональную зависимость от структурной глубины d_i^x нахождения транзактора относительно входов или длины конъюнктивного терма t_i^x. Наблюдаемость вершины имеет аналогичную зависимость $d_i^y \vee t_i^y$ относительно выходов. Для подсчета тестопригодности можно использовать один из параметров d_i^x, t_i^x (d_i^y, t_i^y). Оценки управляемости и наблюдаемости зависят также от процентного отношения числа команд T, имеющих входной $\frac{1}{T}\sum_{j=1}^{x_i} T_j^i$ (выходной $\frac{1}{T}\sum_{j=1}^{y_i} T_j^i$) доступ к вершине при анализе данной программы, к общему числу команд.

Управляемость (наблюдаемость) есть функция от числа операторов, входящих в вершину (исходящих из вершины) ТГ, а также от структурной глубины элемента — расстояния от входов (выходов) или от числа временных тактов, необходимых для управления (наблюдения) компонента в заданном состоянии на временной оси.

Приведенный критерий тестопригодности может быть также использован для оценки качества граф-схемы управления вычислительным процессом. В этом случае рассматриваются операторные вершины, нагруженные входными условиями, а также позиция вершины по отношению к началу или окончанию схемы управления. Позиция операторной вершины

коррелируется с временным тактом управления вычислительным процессом. Число условий выполнения совокупности операций в каждой вершине, объединенное операцией Or, повышает тестопригодность графа относительно управляемости. Аналогично вычисляется наблюдаемость, на которую влияет структурная глубина и мощность условий, создаваемая функциональными операциями And, Or.

В общем случае тестопригодность рассматриваемой вершины ориентированного графа может быть представлена логической функцией, заданной в виде конъюнктивной нормальной формы (КНФ). Тогда управляемость и наблюдаемость будет определяться оценкой по Квайну вычислительной сложности КНФ. При этом логические функции управляемости U_r^f и наблюдаемости N_r^f текущей вершины ТГ задаются конъюнкцией дизъюнктивных термов:

$$U_r^f = \bigwedge_{i=1}^{n_r^x}\left(\bigvee_{j=1}^{x_i} T_{rij}^x\right);\ N_r^f = \bigwedge_{i=1}^{n_r^y}\left(\bigwedge_{j=1}^{y_i} T_{rij}^y\right). \tag{9}$$

Здесь функции U_r^f и N_r^f определяются конъюнкцией всех вершин-предшественников n_r^x (преемников n_r^y), где каждая из них имеет x_i входящих (y_i исходящих) дуг-транзакций, соединенных знаками дизъюнкции. Мощность дизъюнктивных термов соответствует числу входящих в вершину дуг, а число конъюнкций есть структурная глубина местоположения рассматриваемого компонента в ТГ. Затем конъюнктивная форма преобразуется к ДНФ:

$$U_r^f = \bigvee_{i=1}^{x_r}\left(\bigwedge_{j=1}^{n_i^x} T_{ij}^x\right);\ N_r^f = \bigvee_{i=1}^{y_r}\left(\bigwedge_{j=1}^{n_i^y} T_{ij}^y\right), \tag{10}$$

где число термов x_r (y_r) для функции U_r^f (N_r^f) равно всем возможным путям формирования состояния рассматриваемой вершины, а длина терма управляемости (наблюдаемости) n_i^x (n_i^y) есть условие достижимости вершины, т.е. расстояние от входов (выходов) до вершины.

Представляется интересным нестандартное решение, когда критерии управляемости и наблюдаемости текущей вершины ТГ вычисляются на основании построенных логических функций управляемости и наблюдаемости U_i, N_i и интегральной оценки тестопригодности Q при использовании аппарата — алгебраической формы представления графа [18]. Формулы для расчета упомянутых оценок имеют следующий вид:

$$U_i = \frac{1}{t_{\max}^x n_t^x}\sum_{i=1}^{n_t^x}\sum_{j=1}^{k_i^x}\left(t_{\max}^x - \left|t_{ij}^x\right| + 1\right);$$

$$N_i = \frac{1}{t^y_{\max} n^y_t} \sum_{i=1}^{n^y_t} \sum_{j=1}^{k^y_i} (t^y_{\max} - \left| t^y_{ij} \right| + 1);$$
$$Q = \frac{1}{n} \sum_{i=1}^{n} (U_i N_i), \qquad (11)$$

где $t^x_{\max}, n^x_t, k^x_i, \left|t^x_{ij}\right|$ — соответственно конъюнктивный терм максимальной длины для определения критерия управляемости, число термов в логической функции управляемости, число транзакций (букв) в текущем терме функции, мощность рассматриваемой транзакции в терме. При расчете критерия наблюдаемости используются аналогичные обозначения: $t^y_{\max}$, $n^y_t, k^y_i, \left|t^y_{ij}\right|$ для каждой вершины ТГ.

Преобразование КНФ в ДНФ для вершин V_1 — V_3 графа, представленного на рис. 7, *а*, согласно (9), (10) позволяет сформировать логические функции управляемости:

$$U^f(V_1) = T_1 \vee T_2 \vee T_3;$$
$$U^f(V_2) = (T_1 \vee T_2 \vee T_3)T_5 \vee T_4 \vee T_6 = T_1T_5 \vee T_2T_5 \vee T_3T_5 \vee T_4 \vee T_6;$$
$$U^f(V_3) = (T_1 \vee T_2 \vee T_3)T_5T_8 \vee (T_4 \vee T_6)T_8 \vee (T_7 \vee T_9) =$$
$$= T_1T_5T_8 \vee T_2T_5T_8 \vee T_3T_5T_8 \vee T_4T_8 \vee T_6T_8 \vee T_7 \vee T_9.$$

Следуя (11), определяем управляемость компонента V_3:

$$U(V_3) = \frac{1}{t^x_{\max} n^x_t} \sum_{i=1}^{n^x_t} \sum_{j=1}^{k^x_i} (t^x_{\max} - \left|t^x_{ij}\right| + 1) = \frac{1}{3 \times 7}(1+1+1+2+2+3+3) = 0{,}61.$$

Для фрагмента графа, представленного на рис. 7, *б*, преобразование КНФ в ДНФ позволяет определить логическую функцию наблюдаемости вершины V_1:

$$N^f(V_1) = T_6 \vee T_7(T_3 \vee T_5 \vee T_4(T_1 \vee T_2) = T_6 \vee T_7T_3 \vee T_7T_5 \vee T_7T_4T_1 \vee T_7T_4T_2;$$
$$N^f(V_2) = T_3 \vee T_5 \vee T_4(T_1 \vee T_2) = T_3 \vee T_5 \vee T_4T_1 \vee T_4T_2; V_3 = T_1 \vee T_2.$$

Согласно (11) определяем наблюдаемость компонента V_1:

$$N(V_1) = \frac{1}{t^y_{\max} n^y_t} \sum_{i=1}^{n^y_t} \sum_{j=1}^{k^y_i} (t^y_{\max} - \left|t^y_{ij}\right| + 1) = \frac{1}{3 \times 5}(3+2+2+1+1) = \frac{9}{15} = 0{,}6.$$

Для случая, когда дуги в ТГ имеют весовые коэффициенты (b^x_{ij}) b^y_{ij}, соответствующие числу операторов, задействованных в передаче информа-

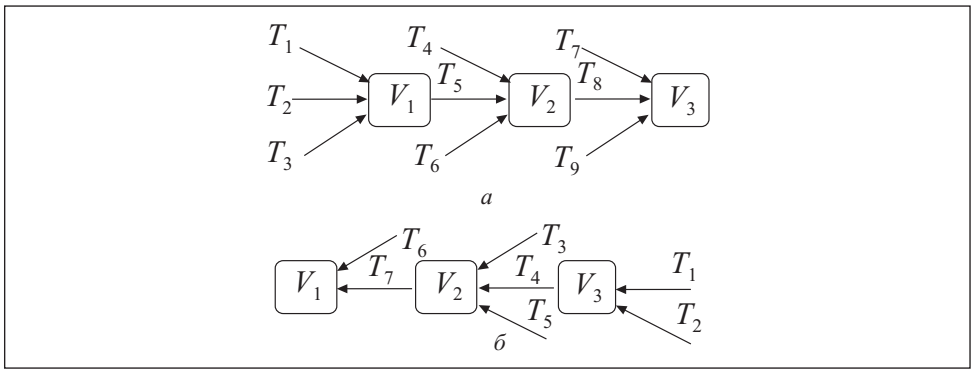


Рис. 7. Фрагменты графа для расчета управляемости (*а*) и наблюдаемости (*б*)

ции между вершинами (мультидугами), формулы для расчета тестопригодности усложняются:

$$U_i = \frac{\sum_{i=1}^{n_i^x} \prod_{j=1}^{k_i^x} b_{ij}^x (t_{\max}^x - |t_{ij}^x| + 1)}{t_{\max}^x \left(\sum_{i=1}^{n_i^x} \prod_{j=1}^{k_i^x} b_{ij}^x \right)}; \quad N_i = \frac{\sum_{i=1}^{n_i^y} \prod_{j=1}^{k_i^y} b_{ij}^y (t_{\max}^y - |t_{ij}^y| + 1)}{t_{\max}^y \left(\sum_{i=1}^{n_i^y} \prod_{j=1}^{k_i^y} b_{ij}^y \right)}.$$

Верификация дискретного косинусного преобразования IP-core Xilinx. Представленные модели верификации программного HDL-кода проверены на реальном проекте дискретного косинусного преобразования (DCT — Discrete Cosine Transform) IP-core Xilinx для определения наличия в нем ошибок. При этом удалось определить неверную семантику работы программы для последующего исправления кода. Фрагмент модуля DCT представлен листингом *1* [Open Source Xilinx.com]. Вся HDL-модель насчитывает 900 строк кода System Verilog.

Листинг 1.

```
module Xilinx
`timescale 1ns/10ps
module dct ( CLK, RST, xin,dct_2d,rdy_out);
output [11:0] dct_2d;
input CLK, RST;
input[7:0] xin; /* input */
output rdy_out;
wire[11:0] dct_2d;
……
/* The first 1D-DCT output becomes valid after 14 + 64 clk cycles. For the first 2D-DCT
output to be valid it takes 78 + 1clk to write into the ram + 1clk to write out of the ram + 8 clks to
```

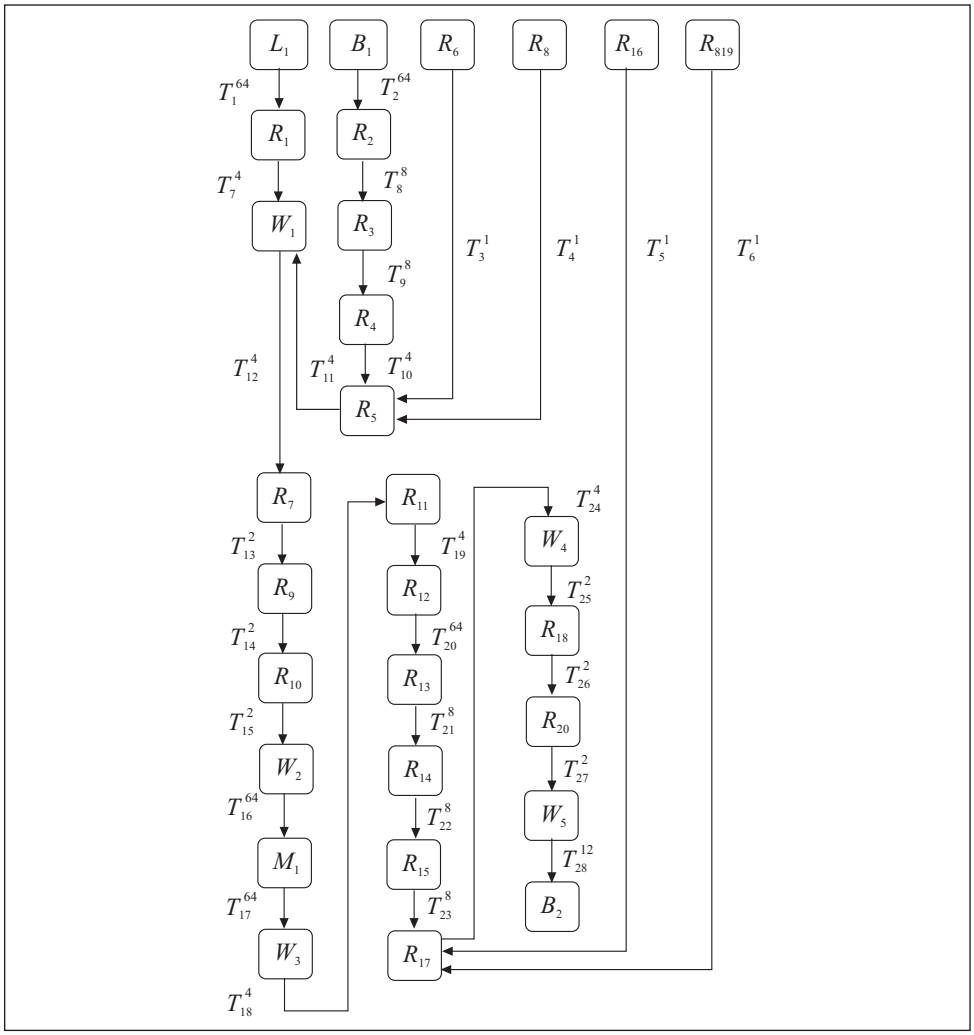


Рис. 8. Транзакционный граф Xilinx-модели

shift in the 1D-DCT values + 1clk to register the 1D-DCT values + 1clk to add/sub + 1clk to take compliment + 1 clk for multiplying + 2clks to add product. So the 2D-DCT output will be valid at the 94th clk. rdy_out goes high at 93rd clk so that the first data is valid for the next block*/

Endmodule

В соответствии с правилами тестопригодного анализа, приведенными выше, спроектирован ТГ (рис. 8), который для DCT-module Xilinx имеет 28 вершин-компонентов (входная и выходная шины, логические и регистровые переменные, векторы и память).

Идентификатор дуги имеет верхний индекс, обозначающий число транзакций в программе между исходящей и входящей вершинами. Для каждой вершины построены логические функции управляемости и наблюдаемости. Пример вычисления функции управляемости для вершины B_2 имеет следующий вид:

$$U^f(B_2) = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}(T_5^1 \vee T_6^1 \vee T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8} \times$$

$$\times (T_1^{64}T_7^4 \vee T_{11}^{4}T_2^{64}T_8^8T_9^8T_{10}^4 \vee T_{11}^{4}T_3^1 \vee T_{11}^{4}T_4^1)) =$$

$$= T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_5^1 \vee T_6^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4} \vee$$

$$\vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}T_1^{64}T_7^4 \vee$$

$$\vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}T_{11}^{4}T_2^{64}T_8^8T_9^8T_{10}^4 \vee$$

$$\vee T_{11}^{4}T_3^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8} \vee$$

$$\vee T_{11}^{4}T_4^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}.$$

Для вершины L_1 ДНФ функция наблюдаемости имеет вид

$$N^f(L_1) = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{23}^{8}T_{22}^{8}T_{21}^{8}T_{20}^{64}T_{19}^{4}T_{18}^{4}T_{17}^{64}T_{16}^{64}T_{15}^{2}T_{14}^{2}T_7^4T_1^{64}.$$

Синтезированные логические функции задают все возможные пути управления как во времени, так и в пространстве, что можно считать новой аналитической формой описания тестопригодности проекта. По ДНФ, следуя выражениям (11), можно определить критерии управляемости (наблюдаемости) для всех компонентов HDL-модели. Существует два варианта расчета программной модели:

1) учитывается только графовая структура, где вес каждой дуги равен единице, независимо от числа транзакций в программном коде;

2) всем дугам графа соответствует реальное число транзакций, находящихся между двумя вершинами ТГ.

Оценки тестопригодности описанных процедур могут быть существенно различными. Пользователь должен определить, что в данном случае важнее: если — структура программного кода, то следует применить первый сценарий, в противном случае необходимо иметь более сложную и точную модель транзакций, распределенных во времени, на множестве графовых компонентов. В качестве примера приведем процедуру вычисления управляемости для B_2:

$$U(B_2) = \frac{1}{22\times 6} \times (6+6+19+22+19+19) = 0{,}54.$$

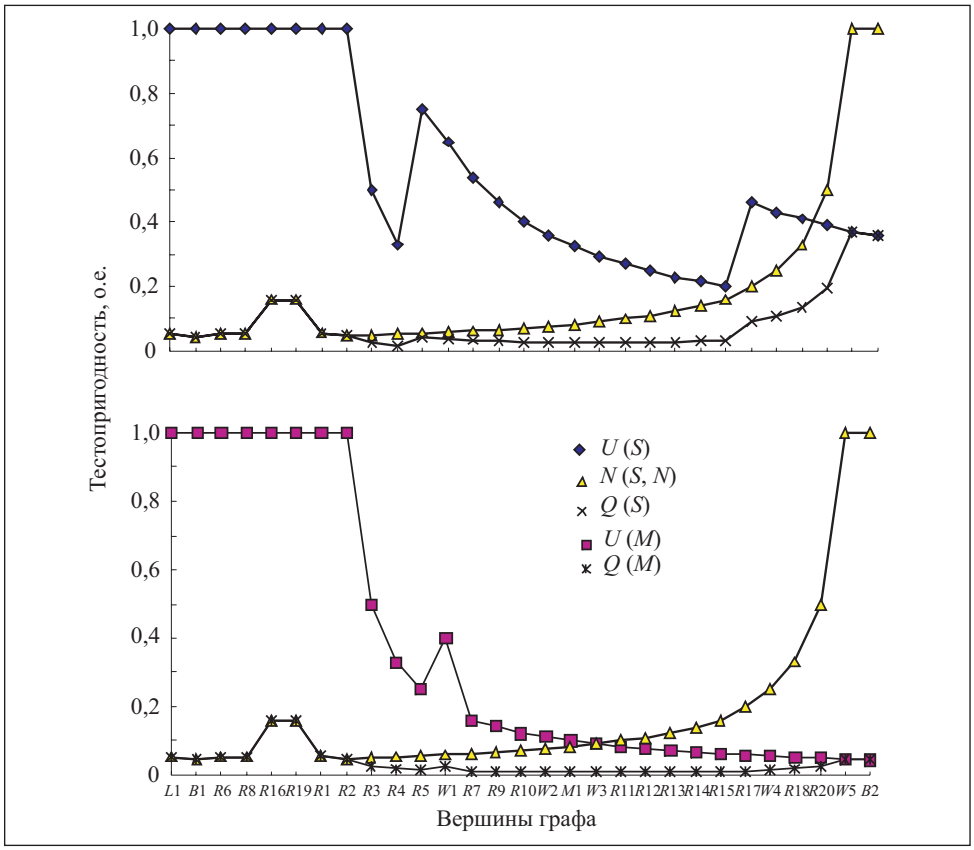


Рис. 9. Графики тестопригодности Xilinx-модели

Применив аналогичные вычисления управляемости (наблюдаемости) для других вершин графа (см. рис. 8), получим графики (рис. 9), позволяющие определить критические вершины для установки необходимых ассерций. Такой вершиной может быть компонент R_{15}, если ТГ представлен одиночными дугами. В случае, когда дуге приписано реальное число транзакций, критические вершины принадлежат компонентам, находящимся ближе к выходной шине B_2. При этом существенное значение имеет не структура графа, а вес входящей дуги, который оказывает негативное влияние в большей степени, если структурная глубина рассматриваемого компонента достаточно высока. Используя (11), можно получить оценку меньше, чем любой из сомножителей (управляемость, наблюдаемость).

После определения управляемостей и наблюдаемостей вершин ТГ выполняем расчет обобщенного критерия тестопригодности программного кода по формуле (11). Для Xilinx DCT-модели такая оценка равна

0,382. Она характеризует качество проектного варианта, что представляется весьма существенным при сравнении нескольких альтернативных решений. В качестве примера позитивного использования разработанных моделей и методов выполнен анализ тестопригодности программного кода DCT из Xilinx библиотеки. Построена транзакционная модель, вычислены характеристики тестопригодности и определены критические вершины R_1, R_5, R_9, R_{15}. В соответствии с числом и типами компонентов разработано функциональное покрытие, фрагмент которого представлен листингом *2*.

Листинг 2.

```
c0: coverpoint xin
   {
     bins minus_big={[128:235]};
     bins minus_sm={[236:255]};
     bins plus_big={[21:127]};
     bins plus_sm={[1:20]};
     bins zero={0};
     }
 c1: coverpoint dct_2d
     {
     bins minus_big={[128:235]};
     bins minus_sm={[236:255]};
     bins plus_big={[21:127]};
     bins plus_sm={[1:20]};
     bins zero={0};
     bins zero2=(0=>0);
     }
endgroup
```

Для критических точек, определенных в результате анализа тестопригодности ТГ разработана ассерционная модель проверки основных характеристик дискретного косинусного преобразования. Существенный фрагмент кода механизма ассерций представлен листингом *3*.

Листинг 3.

```
sequence first( reg[7:0] a, reg[7:0]b);
    reg[7:0] d;
    (!RST,d=a)
     ##7 (b==d);
  endsequence
 property f(a,b);
    @(posedge CLK)
```

```
    // disable iff(RST||$isunknown(a)) first(a,b);
     !RST |=> first(a,b);
endproperty
  odin:assert property (f(xin,xa7_in))
   // $display("Very good");
  else $error("The end, xin =%b,xa7_in=%b", $past(xin, 7),xa7_in);
```

В результате верификации программной HDL-модели дискретного косинусного преобразования в среде моделирования Active-HDL были найдены неточности в восьми строках исходного кода HDL-модели:

```
//  add_sub1a <= xa7_reg + xa0_reg;//
```

Последующее исправление ошибок привело к появлению исправленного фрагмента кода, который показан в листинге *4*.

Листинг 4.

```
add_sub1a <= ({xa7_reg[8],xa7_reg} + {xa0_reg[8],xa0_reg});
add_sub2a <= ({xa6_reg[8],xa6_reg} +{xa1_reg[8],xa1_reg});
add_sub3a <= ({xa5_reg[8],xa5_reg} +{xa2_reg[8],xa2_reg});
add_sub4a <= ({xa4_reg[8],xa4_reg} + {xa3_reg[8],xa3_reg});
end
else if (toggleA == 1'b0)
begin
add_sub1a <= ({xa7_reg[8],xa7_reg} - {xa0_reg[8],xa0_reg});
add_sub2a <= ({xa6_reg[8],xa6_reg} - {xa1_reg[8],xa1_reg});
add_sub3a <= ({xa5_reg[8],xa5_reg} - {xa2_reg[8],xa2_reg});
add_sub4a <= ({xa4_reg[8],xa4_reg} - {xa3_reg[8],xa3_reg});
```

Выводы. Предложенная технология тестирования и верификации цифровых систем для диагностирования и исправления ошибок HDL- моделей позволяет существенно повысить качество проектируемых цифровых систем на кристаллах и уменьшить период их разработки. Практическая значимость предложенных методик и моделей заключается в рыночной привлекательности и высокой заинтересованности технологических компаний в инновационных решениях проблемы тестирования и верификации программно-аппаратных изделий на системном уровне проектирования для уменьшения time-to-market и увеличения выхода годной продукции. Дальнейшие исследования направлены на разработку стандартных интерфейсов для последующей интеграции моделей, методов и программных продуктов в технологические маршруты проектирования цифровых систем на кристаллах.

The procedure of testing and verification of digital systems for diagnosis and correction of errors of HDL-models is offered. The procedure is based on joint use of the mechanism (system) of assertion engine and testable design. A universal model of the software component is presented in a form of the transaction graph. Logical functions of testability of HDL-models were shown which are obtained on the basis of its use. Examples are presented for analysis of testability by the method of calculation of controllability and observability of transaction and control graphs.

1. *Abramovici M., Breuer M. A. and Friedman A. D.* Digital System Testing and Testable Design. — San Jose: Computer Science Press, 1998. — 652 p.
2. *Bayraktaroglu I., Orailoglu A.* The Construction of Optimal Deterministic Partitionings in Scan-based BIST Fault Diagnosis: Mathematical Foundations and Cost-effective Implementations // IEEE Transactions on Computers. — 2005. — P. 61—75.
3. *Densmore D., Passerone R., Sangiovanni-Vincentelli A.* A Platform-based Taxonomy for ESL Design // IEEE Design & Test of Computers. September-October, 2006. — P. 359—373.
4. *Francisco DaSilva, Yervant Zorian, Lee Whetsel at al* . Overview of the IEEE P1500 Standard // ITC International Test Conference. Charlotte, NC, USA — 2003. — P. 988—997.
5. *Rashinkar P., Paterson P., Singh L.* System-on-chip Verification: Methodology and Techniques.— Norwell, MA, USA: Kluwer Academic Publishers, 2002. — 324 p.
6. *Yervant Z.* What is Infrustructure IP? // IEEE Design & Test of Computers. — 2002. — P. 5—7.
7. *Yervant Z., Gizopoulos D.* Gest Editors' Introduction: Design for Yield and Reliability // Ibid. — 2004. — P. 177—182.
8. *Yervant Z.* Guest Editor's Introduction: Advances in Infrastructure IP // Ibid. — 2003. — 49 p.
9. *Thatte S. M., Abraham J.A.* Test Generation for Microprocessors // IEEE Trans. Comput. — 1980. — C-29, No 6. — P. 429—441.
10. *Шаршунов С. Г.* Построение тестов микропроцессоров. 1. Общая модель. Проверка обработки данных // Автоматика и телемеханика. — 1985. — № 11. — С. 145—155.
11. *Jerraya A. A.* System Level Synthesis SLS. — TIMA. Annual Report. — 2002. — P. 65 — 75.
12. *Ghenassia F.* Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems. — Norwell, MA, USA: Published by Springer. — 2005. — 282 p.
13. *Bergeron J.* Writing Testbenches: Functional Verification of HDL Models. — Boston: Kluwer Academic Publishers. — 2001. — 354 p.
14. *Bergeron J., Cerny E., Hunter A., Nightingale A.* Verification Methodology. Manual for SystemVerilog. — NY : Springer, 2005. — 528 p.
15. *Foster H., Krolnik A., Lacey D.* Assertion-based Design. Second edition. — Kluwer Academic Publishers. — NY: Springer, 2005. — 392 p.
16. *Rashinkar P., Paterson P., Singh L.* System-on-chip Verification: Methodology and Techniques. — Boston: Kluwer Academic Publishers, 2002. — 393 p.
17. *Meyer A. S.* Principles of Functional Verification. — NY: Elsevier Science, 2004. — 206 p.
18. *Хаханов В. И., Литвинова Е. И., Гузь О. А.* Проектирование и тестирование цифровых систем на кристаллах. — Харьков : ХНУРЭ, 2009. — 484 с.

Поступила 29.07.09;
после доработки 13.11.09

ХАХАНОВ Владимир Иванович, д-р техн. наук, профессор, декан факультета компьютерной инженерии и управления Харьковского национального университета радиоэлектроники. В 1978 г. окончил Харьковский ин-т радиоэлектроники. Область научных исследований — проектирование и тестирование цифровых систем и сетей.

ХАХАНОВА Ирина Витальевна, д-р техн. наук, профессор кафедры автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, который окончила в 1994 г. Область научных исследований — проектирование и верификация цифровых систем и сетей, цифровая обработка сигналов.

ЛИТВИНОВА Евгения Ивановна, канд. техн. наук, доцент кафедры технологии и автоматизации производства РЭС и ЭВС Харьковского национального университета радиоэлектроники. В 1985 г. окончила Харьковский ин-т радиоэлектроники. Область научных исследований — проектирование и тестирование цифровых систем и сетей на кристаллах.

ГУЗЬ Олеся Алексеевна, канд. техн. наук, доцент, зав. кафедрой специализированных компьютерных систем Донецкого института автомобильного транспорта. В 2002 г. окончила Харьковский национальный университет радиоэлектроники. Область научных исследований — проектирование и диагностика цифровых систем и сетей на кристаллах.