НАУКОВО-ТЕХНІЧНИЙ ЖУРНАЛ *ISSN 1681-4886*

6 (79)' 2009

Виходить 6 разів на рік
Видається з 23 квітня 1996 р.

ІНФОРМАЦІЙНО -КЕРУЮЧІ СИСТЕМИ НА ЗАЛІЗНИЧНОМУ ТРАНСПОРТІ

INFORMACIJNO-KERUÛCI SISTEMI NA ZALIZNICNOMU TRANSPORTI

Видання

Державної адміністрації залізниць України

Української державної академії залізничного транспорту

УДК 681.326:519.613

ХАХАНОВ В.И., д.т.н., профессор,
ХАХАНОВА И.В., д.т.н., профессор,
ЛИТВИНОВА Е.И., к.т.н., доцент (ХНУРЭ),
ГУЗЬ О.А., к.т.н., доцент (ДИАТ)

Верификация HDL-кода на основе механизма ассерций

Введение

Ведущие компании планеты, такие как Cadence, Synopsis, Mentor Graphics, большое внимание уделяют системной стадии создания цифровых изделий на кристаллах, практически неподдающейся автоматизации. Здесь на процесс отладки программного кода теряется до 70% общего времени, определяемого как time-to-market. Предлагается технология тестирования и верификации системных HDL-моделей, ориентированная на существенное повышение качества проектируемых компонентов цифровых систем на кристаллах (віходной продукции - yied) и уменьшение времени разработки (time-to-market) путем использования среды моделирования, тестопригодного анализа логической структуры HDL-программы и механизма ассерций.

Технология позволяет осуществлять поиск ошибок с заданной глубиной в программном HDL-коде за приемлемое для разработчика время путем введения в критические точки программной модели ассерционной избыточности, вычисляемой с помощью синтезированных логических функций тестопригодности. Последние определяют качество программного кода путем синтеза дизъюнктивной нормальной формы (ДНФ), для которой оценка по Квайну формирует количественную характеристику тестопригодности компонента или программы, представленной последовательностью операторов. Используемые в hardware design and test критерии тестопригодности (управляемость и наблюдаемость) применены для оценки качества программного кода в целях его улучшения и эффективного диагностирования семантических ошибок.

Цель – улучшение технологии тестирования и верификации цифровых систем для диагностирования и исправления ошибок HDL-моделей путем синтеза транзакционного графа 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 программного обеспечения тестирования (*testbench*) представлены в работах [1-3]. 2. Модели и методы верификации системных моделей на основе механизма ассерций описаны в публикациях [4-7]. Тестопригодное проектирование программных продуктов использует 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 захватываются и адапти-

руются к макроэлектронике, представленной компьютерными системами и сетями. Вот 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. Метрику анализа тестопригодности [1,2] (управляемости и наблюдаемости) цифровых структур можно адаптировать для оценки программного кода в целях 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 используются для создания таблицы покрытия функциональностей программных продуктов модулем testbench в целях оценки достоверности тестов, установления диагноза и исправления ошибок.

4. Графовые модели регистровых передач Тэтта-Абрахама [15] С.Г. Шаршунова [16-17] используются при тестировании программных продуктов, которые путем их структурно-логического анализа приводятся к более технологичной форме. Таковой является транзакционный граф как модель HDL-кода, записываемый в виде алгебраической формы представления графа для подсчета тестопригодности в целях определения критических компонентов (точек) программы для установки ассерций.

5. Разделение автомата на управляющую [1,7] и операционную части применяется 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] также адекватно отображает временные стадии изменения yield при создании, тиражировании и сопровождении программного продукта.

7. Платформенно-ориентированный синтез [3] HDL-кода (platform-based electronic system-level design) с использованием существующих наборов компонентов (chipset) под управлением GUI изоморфен технологии объектно-ориентированного программирования на основе использования наработанных ведущими компаниями библиотек. Применение технологии Electronic System Level (ESL) [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 применяется 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 (platform-based testbench synthesis) с использованием существующих библиотек тестов (ALINT) со стандартизованными функциональными компонентами для генерирования тестов программных модулей на основе библиотек ведущих компаний.

10. Стандартные решения сервисного обслуживания функциональностей F-IP в рамках инфраструктуры I-IP [4-8] используются для создания инфраструктуры встроенной верификации компонентов программного продукта в целях 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, реализованных как в аппаратном, так и в программном исполнении [9-11], предоставляет цифровой системе замечательное свойство самовосстановления работоспособности программных и аппаратных компонентов путем использования альтернативных или избыточных ресурсов инфраструктуры сервисного обслуживания I-IP. Примером тому может служить мультипроцессорное исполнение аппаратных продуктов, которое является устойчивым к возникающим дефектам. При этом отказавший адресуемый компонент может быть заменен резервным в процессе выполнения функциональности. Свойство адресуемости используется при создании критических программных продуктов, где наличие адресуемых диверсных (мультиверсных) резервных компонентов обеспечивает отказоустойчивость системы при возникновении дефектов.

13. Интересной и рыночно привлекательной является проблема автономного внутрикристального встроенного тестирования, диагностирования и ремонта без применения внешних средств [9-11], которой занимаются все ведущие компании. К решению проблемы привлекаются современные беспроводные и Internet технологии дистанционного сервисного обслуживания. Обратная сторона медали заключается в несанкционированном доступе к содержимому кристалла на расстоянии, что может привести к нежелательным деструктивным последствиям и выводу из строя цифрового изделия. Тем не менее, специфика современных цифровых систем на кристаллах заклю-

чается 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 становится возможной благодаря использованию 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 благодаря использованию программируемых логических интегральных схем, куда можно в случае обнаружения неисправности записать новый bit stream, не имеющий ошибок, который фактически создает новую аппаратуру путем повторного программирования кристалла.

Сближение и взаимопроникновение технологий приводит к изоморфным методам проектирования, тестирования и верификации по отношению к программным и аппаратным комплексам, что по существу является закономерным процессом ассимиляции прогрессивных концепций и решений. Тому способствует факт, что наиболее важные параметры жизненного цикла изделия, такие как time-to-market и yield, становятся соизмеримыми по времени и выходу 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.

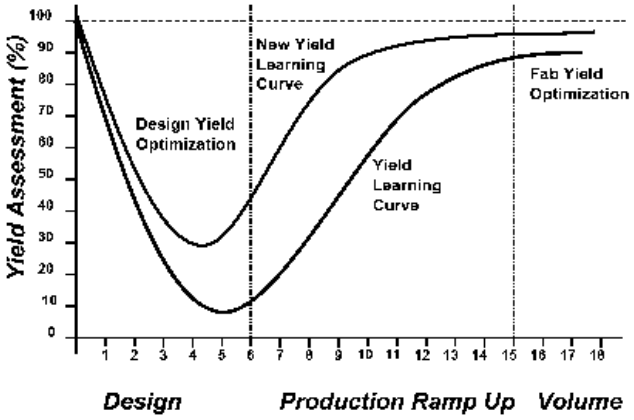

Рисунок 1 - График жизненного цикла программно-аппаратного комплекса

В контексте жизненного цикла существуют две актуальные проблемы, которые связаны с поднятием графика (кривой) вверх по оси ординат, а также с компрессией упомянутой кривой по временной оси, означающей уменьшение параметра time-to-market. Здесь повышение yield происходит на всех стадиях: design – за счет устранения ошибок разработки; production ramp up – за счет исправления кода, имплементированного в память системы на кристалле; volume – за счет выпуска service packs, корректирующих ошибки путем распространения нового кода через Internet или satellites.

Согласно [12-18] стоимость верификации программно-аппаратных продуктов на основе ASIC, IP-core, SoC составляет 70% от общих затрат проектирования. Аналогичная оценка, около 80 %, определяет размерность testbench-кода от общей длины формального описания проекта. Задачи, решаемые в процессе верификации, связаны с устранением ошибок проектирования как можно на более ранней его стадии в целях 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 ассерций [15-17] позволяет выполнять анализ основных специфицированных условий в процессе моделирования проекта и диагностировать ошибки в случае их обнаружения 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:

$$(T,F) \oplus (S,A) = L_s. \qquad (1)$$

где T, F – соответственно тестовые воздействия и функциональное покрытие эталонной модели с ожидаемыми реакциями; S, A – соответственно HDL-модель, подлежащая проверке, и механизм ассерций для верификации и точного диагностирования ошибок в программном коде. Тестирование аппаратной реализации основано на использовании 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, L_h – списки ошибок, получаемые на стадиях верификации проекта и тестирования готового цифрового изделия.

Для понимания соотношений между ключевыми понятиями: верификация, валидация, ассерция – вводятся следующие определения [13–15]. Верификация – есть процесс анализа системы или компонентов для определения корректности формальных преобразований входного описания на каждой стадии проектиро-

вания. Валидация – есть процесс определения работоспособности системы и ее компонентов путем проверки ее соответствия основным требованиям спецификации после выполнения каждой стадии проектирования. Сертификация – есть письменная гарантия валидности системы (компонентов) требованиям спецификации при ее использовании по назначению. Ассерция – есть высказывание системного уровня, определяющее корректность преобразований в текущем этапе процесса проектирования относительно входного описания или требований спецификации. Следуя данному определению, ассерция записывается в виде предиката $A_i = T_i \oplus S_i \to d(A_i)$, 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 $T_i \oplus S_i$ ассерции можно дифференцировать на три типа $d(A_i) = \{A^c, A^g, A^w\}$ – прекращение верификации HDL-модели, переход к проверке следующего программного блока, выдача сообщения на консоль. Механизм ассерций – есть система высказываний, представленная в виде вектора $A = (A_1, A_2, ..., A_i, ..., A_n)$, 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 функциональностей, не привязанных к параметру времени. Матрица ассерций является более точным инструментом, который рассматривает состояния компонентов HDL-модели во времени и пространстве. Такие же функции – повышение глубины диагностирования – имеют место быть и в стандартах граничного сканирования (IEEE 11.49, IEEE 1500), где тестирование компонентов цифровой системы регулируется контроллером тестового доступа (TAP).

Структурная модель процесса проектирования в части взаимодействия процедур валидации и верификации представлена на рис. 2. Здесь следует отметить что верификация определяет корректность выполненных преобразований на рассматриваемом этапе проектирования, а валидация определяет соответствие технического состояния системы требованиям спецификации на каждой стадии проектирования.

Стратегии верификации и тестирования имеют различные модели для 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).

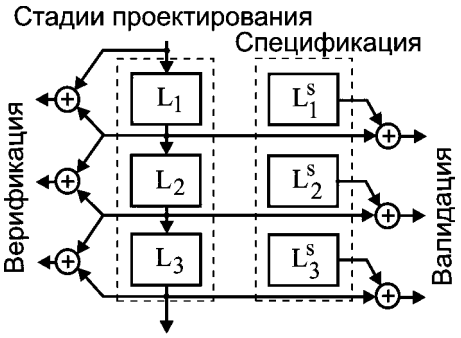

Рисунок 2 - Взаимодействие процессов валидации и верификации

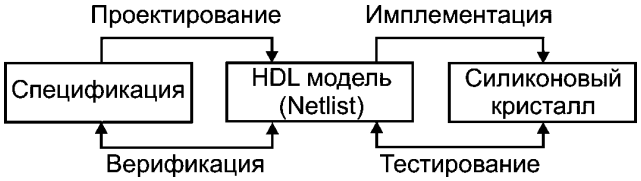

Рисунок 3 - Стратегия разработки проекта

Конечный результат есть netlist или отлаженная HDL-модель регистрового уровня. Следующий итеративный процесс – синтез и имплементация проекта в силиконовый кристалл. Тестирование здесь проверяет 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 здесь будет стоить до миллиона долларов.

С учетом приведенных определений и пояснений модель среды или макропроцесса верификации программной стадии проекта ориентирована на уменьшение времени создания изделия и повышение уровня выхода годной продукции за счет использования избыточности кода в виде механизма ассерций и использования Testbench совместно с метрикой определения качества теста или функциональной полноты.

Инфраструктура тестирования и верификации HDL-модели представлена на рис. 4, где спецификация проекта, описанная на формальном языке высокого уровня, является исходной информацией для: создания метрики оценивания качества теста в виде функционального покрытия; HDL-модели проекта; теста с эталонными реакциями – testbench, ассерционной структуры, служащей дополнением к основной модели, что необходимо для ускорения проверки и отладки проекта. Среда верификации представлена системой моделирования, блоком тестирования (Testbench), механизмом ассерций (Assertion Engine) 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, ориентированные на проверку функциональностей (переменные, функции, последовательности), параметры которых определяются в корзине функционального покрытия. Механизм ассерций – модельная избыточность, дополняющая testbench, в части проверки внутренних по времени и пространству состояний проекта, представленная вход-выходными высказываниями 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.

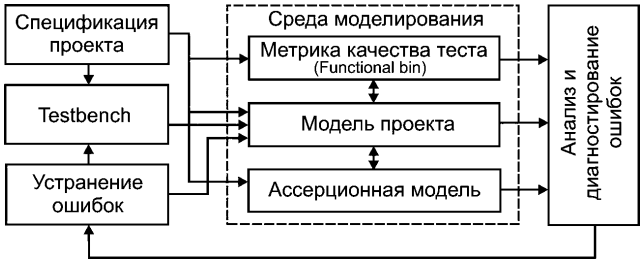

Рисунок 4 - Среда верификации проекта

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 Model Under Verification (MUV). В этом случае необходимо иметь генератор входных последовательностей или testbench без выходных реакций. Блок Assertion Engine является надстройкой для модулей testbench и MUV и служит для сравнения результатов моделирования в целях формирования двоичного вектора экспериментальной проверки, в качестве которого здесь выступает вектор (состояний) ассерций:

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

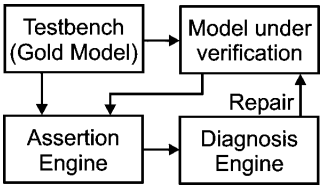

Рисунок 5 - Технология использования ассерций для верификации проекта

Хотя 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 в целях последующего принятия решения о продолжении процесса моделирования, прекращения или перехода к выполнению анализа другого фрагмента программного кода.

Интерес представляет и последовательность действий по созданию среды верификации, где единственным аргументом является спецификация проекта, все остальное – производные от нее. На рис. 6 изображена структура взаимосвязей процесса проектирования и диагностирования для исправления ошибок в HDL-коде программы.

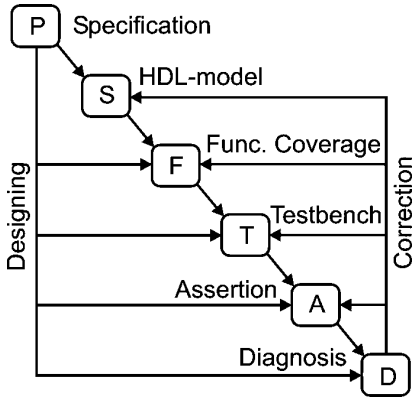

Рисунок 6 - Маршрут верификации проекта

Практически, 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 создаваться независимо от HDL-модели и другим разработчиком. 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, что означает – выполнять дос-

таточно простые условия тестопригодности (рис. 5 и 6). Здесь необходимо решать задачи: 1) Написание testbench и генерирование функциональных покрытий: 2) Определение критических точек программного кода для установки ассерций; 3) Корректное написание самих 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 в целях 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 параллельно и независимо друг от друга средой моделирования. Ассерционная модель определяет отклонения от поведения объекта в существенных точках пространственно-временной эталонной структуры. Формат ассерций должен соответствовать формату HDL-модели проекта. Ассерции ориентированы на диагностирование семантических ошибок HDL-модели путем использования testbench и MUV. Аналитическая модель инфраструктуры верификации представлена в следующем виде (P – спецификация проекта, S – soft-модель, A – ассерционная модель, T – Testbench, F – корзина покрытия функциональностей, d – модуль диагностирования ошибок и C – условия диагностирования ошибок):

$$
\begin{aligned}
&M = \{P, S, A, T, F, d, C\},\\
&1)\ S = f_1(P) = \left|S_{ij}\right|;\\
&2)\ F = f_2(P, S) = \{F_1, F_2, ..., F_i, ..., F_n\};\\
&3)\ T = f_3(P, S, F) = \{T_1, T_2, ..., T_i, ..., T_n\};\\
&4)\ A = f_4(P, S, F, T) = \left|A_{ij}\right|;\\
&5)\ D = f_5(P, S, F, T, A) = \left|L_{ij}\right| \in \{L_s, L_h\};\\
&6)\ C = [\bigcup_{i=1}^{n} F_i \in F = P] \wedge [\bigcup_{i=1}^{n} T_i \in T = F];\\
&7)\ L_s = (T, F) \oplus (S, A);\\
&8)\ L_h = (T) \oplus (S, B).
\end{aligned}
\qquad (2)
$$

Формула 6 в (2) определяет условия полноты проверки функциональностей тестом (testbench) относительно спецификации. Уравнение 7 задает функцию вычисления ошибок проектирования при переходе от системного к регистровому уровню, используя все атрибуты верификационной инфраструктуры. Равенство 8 регламентирует нахождение неисправностей на стадии эксплуатации цифровой системы на кристалле.

Ассерционная избыточность является функцией от критических точек HDL- модели, предельное число которых может быть равно количеству временных фреймов функциональных компонентов, определенных спецификацией.

Априорно, координатам вектора ассерций присваиваются значения X. Затем определяются критические координаты, число которых будет достаточным для проведения верификационного эксперимента в целях поиска 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 транзакционного графа программного кода. Координатам вектора соответствует матрица достижимостей транзакционного графа 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]:

$$
\begin{cases}
L_s(\mathrm{A}) = \left(\bigcap\limits_{\forall \mathrm{i}(\mathrm{A}_i = 1)} A_i\right) \setminus \left(\bigcup\limits_{\forall \mathrm{i}(\mathrm{A}_i = 0)} A_i\right);\\
L_m(\mathrm{A}) = \left(\bigcup\limits_{\forall \mathrm{i}(\mathrm{A}_i = 1)} A_i\right) \setminus \left(\bigcup\limits_{\forall \mathrm{i}(\mathrm{A}_i = 0)} A_i\right).
\end{cases}
\qquad (3)
$$

Система уравнений предназначена для поиска одиночных и кратных ошибок путем использования вектора ассерционных состояний. Длина ассерционного вектора равна числу вершин в графе или количеству программных блоков в функционально-логической структуре HDL-кода. Векторная модель среды верификации имеет вид:

$$
\begin{bmatrix} T_1 \\ T_2 \\ T_i \\ T_n \end{bmatrix} \oplus \begin{bmatrix} S_1 \\ S_2 \\ S_i \\ S_n \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_i \\ A_n \end{bmatrix} \xrightarrow{d} \begin{bmatrix} L_1 \\ L_2 \\ L_i \\ L_n \end{bmatrix}
\qquad (4)
$$

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

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

Затем существенные {0,1}-координаты вектора ассерций маскируют матрицу достижимостей для получения списка программных блоков с ошибками путем выполнения одной из процедур, определенных в (3).

Демонстрация технологии диагностирования неисправных блоков представлена следующим примером. Пусть имеется функционально-логический граф HDL-модели, изображенный на рис. 7.

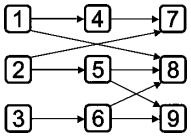

Рисунок 7 - Функционально-логический граф HDL-модели

Структура взаимосвязей графа представлена 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 (4), содержит списки блоков-предшественников для каждой вершины графа, которые получены 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 (4) для графа, представленного на рис. 7, трансформируется к виду:

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

(5)

В результате выполнения диагностирования по системе уравнений (3), заключающейся в пересечении всех неисправных компонентов, соответствующих единичным координатам вектора ассерций, с последующим вычитанием объединения всех неисправных модулей, соответствующих нулевым координатам вектора A, получается следующий список дефектных программных блоков (при условии существования только одного ошибочного модуля):

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

При использовании второго уравнения из выражения (3) можно получить список всех программных блоков, которые могут иметь ошибки, при условии существования нескольких дефектных компонентов:

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

Для иллюстрации следующего примера диагностического эксперимента можно убрать из рассмотрения первые два вектора, которые не являются существенными для процесса обработки списков неисправных блоков на основе анализа координат ассерционного вектора.

Процедура вычисления списка одиночных дефектов имеет вид:

$A_1 = X$		$L_1 = S_1$		
$A_2 = X$		$L_2 = S_2$		
$A_3 = X$		$L_3 = S_3$		
$A_4 = X$		$L_4 = S_1, S_4$		
$A_5 = 0$	$\xrightarrow{d}$	$L_5 = S_2, S_5$	$(L_8 \cap L_9) -$ $-(L_5 \cup L_7)$	S_3, S_6
$A_6 = X$		$L_6 = S_3, S_6$		
$A_7 = 0$		$L_7 = S_1, S_2, S_4, S_7$		
$A_8 = 1$		$L_8 = S_1, S_2, S_3, S_5, S_6, S_8$		
$A_9 = 1$		$L_9 = S_2, S_3, S_5, S_6, S_9$		

Для множественных неисправных блоков на одном и том же векторе ассерций результат имеет большее число дефектных модулей:

$A_1 = X$		$L_1 = S_1$		
$A_2 = X$		$L_2 = S_2$		
$A_3 = X$		$L_3 = S_3$		
$A_4 = X$		$L_4 = S_1, S_4$		
$A_5 = 0$	$\xrightarrow{d}$	$L_5 = S_2, S_5$	$(L_8 \cup L_9) -$ $-(L_5 \cup L_7)$	$S_3, S_6,$ S_8, S_9
$A_6 = X$		$L_6 = S_3, S_6$		
$A_7 = 0$		$L_7 = S_1, S_2, S_4, S_7$		
$A_8 = 1$		$L_8 = S_1, S_2, S_3, S_5, S_6, S_8$		
$A_9 = 1$		$L_9 = S_2, S_3, S_5, S_6, S_9$		

Интересны и другие формы описания векторной модели проведения диагностического эксперимента. Ниже представлена матрица достижимостей, вставлен-

ная в модель диагностирования, где ассерции изображены в виде троичного вектора:

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 результат поиска программных блоков выполнен при условии существования в проекте одного неисправного модуля, который записан в последней строке. Процедура диагностирования на основе ассерционного вектора А путем выполнения 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

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

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

Также интересной для диагностирования HDL-модели по структурно-логическому (транзакционному) графу представляется алгебраическая форма описания графовых структур [18]. Ее преимущества заключаются в компактности задания матрицы достижимостей, которая к тому же обладает свойством структуризации графа в форме задания всех путей, нагруженных на каждую вершину. 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 на рис. 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	$L_9 = S_2 S_5 S_9 \vee S_3 S_6 S_9$
	$L_s = S_3, S_6$

$L_s = (L_8 \wedge L_9) - (L_5 \vee L_7) =$
$= (S_2 S_5 \vee S_3 S_6) - S_2 S_5 =$
$= S_3 S_6$

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 других ДНФ, соответствующих единичному значению ассерционной координаты, левая часть конъюнктивного терма, включая переменную, ранее определенную нулем, удаляется. 3) Для единичных функций выполняется пересечение (объединение) оставшихся термов, если выполняется диагностирование одиночных (кратных) неисправных блоков.

Такие преобразования ДНФ путем специального моделирования нулевых сигналов переменных в единичных, относительно ассерций, логических функциях представления графа в целях 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, включающей тысячи строк кода, описывающего системные структуры на языках VHDL, Verilog. Для реализации технологии верификации необходимо сгенерировать транзакционный граф, описывающие структуру программного кода в компонентах и операторах HDL-языка.

Анализ тестопригодности программных HDL-моделей

Достаточно существенная избыточность HDL-модели предполагает ее эффективное использование в целях повышения тестопригодности структуры разрабо-

танного или написанного кода. Существующие стандарты тестопригодного проектирования аппаратуры можно адаптировать для верификации HDL-кода системных и регистровых программных моделей. Для этого используется граф регистровых или транзакционных передач С.Г. Шаршунова [10], который предоставляет пользователю информацию о взаимосвязях булевых и регистровых переменных, памяти и интерфейсных шинах. Такие данные достаточно легко можно получить автоматически, анализируя синтаксис строк HDL-кода. Сгенерированный граф должен покрывать функциональности компонентов программной модели и задавать все существующие связи для приема, передачи и преобразования информации между вершинами графа транзакций (TG – Transaction Graph) [18]. Для интегрального оценивания тестопригодности Q транзакционного графа вводятся следующие критерии:

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

$$U_i = \frac{1}{T}\sum_{j=1}^{x_i} T_j^i \times \frac{1}{d_i^x \vee t_i^x};\ N_i = \frac{1}{T}\sum_{j=1}^{y_i} T_j^i \times \frac{1}{d_i^y \vee t_i^y}. \quad (6)$$

Управляемость и наблюдаемость есть метрика оценивания тестопригодности не соединительных линий, а компонентов HDL-кода, таких как: регистр, счетчик, память или массивы, вход-выходные шины, векторы, логические или арифметические переменные. Указанные выше характеристики имеют функциональную зависимость от структурной глубины нахождения компонента (относительно входов) или длины конъюнктивного терма – $d_i^x \vee t_i^x$, ($\mathrm{d}_\mathrm{i}^\mathrm{y} \vee \mathrm{t}_\mathrm{i}^\mathrm{y}$ – относительно выходов) а также от процентного отношения числа команд $\frac{1}{T}\sum_{j=1}^{x_i} T_j^i$, имеющих входной (выходной – $\frac{1}{T}\sum_{j=1}^{y_i} T_j^i$) доступ к вершине при анализе данной программы. Тестопригодность *Q*, представленная в (6), зависит от управляемости *U*, наблюдаемости (*N*), а также от модельной избыточности (*Z*), представленной компонентами: метрика функционального покрытия (*F*), testbench (*B*), механизм ассерций (*A*). Управляемость (наблюдаемость) есть функция от числа операторов, входящих в вершину (исходящих из вершины) транзакционного графа, а также от структурной глубины рассматриваемого элемента – расстояния от входов (выходов) или от количества временных тактов, необходимых для управления (наблюдения) компонента в заданном состоянии на временной оси.

Приведенный критерий тестопригодности может быть также использован и для оценки качества графсхемы управления вычислительным процессом. Здесь рассматриваются только 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. Количество условий выполнения совокупности операций в каждой вершине, объединенное операцией Or, повышает тестопригодность графа в части 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. При этом управляемость и наблюдаемость будет определяться оценкой по Квайну вычислительной сложности КНФ. В общем случае логические функции управляемости и наблюдаемости текущей вершины (транзакционного) графа задаются конъюнкцией дизъюнктивных термов – первая строка в (7):

$$1)\ U_r^f = \bigwedge_{i=1}^{n_r^x}\Big(\bigvee_{j=1}^{x_i} T_{rij}^x\Big);\ N_r^f = \bigwedge_{i=1}^{n_r^y}\Big(\bigvee_{j=1}^{y_i} T_{rij}^y\Big);$$
$$2)\ U_r^f = \bigvee_{i=1}^{x_r}\Big(\bigwedge_{j=1}^{n_i^x} T_{ij}^x\Big);\ N_r^f = \bigvee_{i=1}^{y_r}\Big(\bigwedge_{j=1}^{n_i^y} T_{ij}^y\Big). \quad (7)$$

Здесь функция управляемости U_r^f (наблюдаемости N_r^f) определяется конъюнкцией всех вершин-предшественников n_r^x (преемников n_r^y), где каждая из них имеет x_i входящих (исходящих y_i) дуг-транзакций, соединенных знаками дизъюнкции. Мощность дизъюнктивных термов соответствует количеству входящих в вершину дуг, а число конъюнкций есть структурная глубина местоположения рассматриваемого компонента в транзакционном графе. Далее конъюнктивная форма преобразуется к виду ДНФ – вторая строка в (7), где число термов для функции управляемости (наблюдаемости) $x_r(y_r)$ равно всем возможным путям формирования состояния рассматриваемой вершины, а длина терма управляемости (наблюдаемости) $n_i^x(n_i^y)$ есть условие достижимости вершины – структурная глубина от входов (выходов).

Интересным представляется нестандартное решение, когда критерии управляемости и наблюдаемости текущей вершины транзакционного графа вычисляются на основании построенных логических функций управляемости и наблюдаемости (U_i, N_i) и интегральной оценки тестопригодности (*Q*) при использо-

вании аппарата – алгебраической формы представления графа [18]. Формулы подсчета упомянутых оценок имеют следующий вид:

$$U_i = \frac{1}{t^x_{\max} \times n^x_t} \times \sum_{i=1}^{n^x_t} \sum_{j=1}^{k^x_i} (t^x_{\max} - \left|t^x_{ij}\right| + 1);$$

$$N_i = \frac{1}{t^y_{\max} \times n^y_t} \times \sum_{i=1}^{n^y_t} \sum_{j=1}^{k^y_i} (t^y_{\max} - \left|t^y_{ij}\right| + 1); \qquad (8)$$

$$Q = \frac{1}{n} \sum_{i=1}^{n} (U_i \times N_i),$$

где $t^x_{\max}, n^x_t, k^x_i, \left|t^x_{ij}\right|$ – конъюнктивный терм максимальной длины для определения критерия управляемости; количество термов в логической функции управляемости; количество транзакций (букв) в текущем терме функции; мощность рассматриваемой транзакции в терме. Аналогичные обозначения используются и при подсчете критерия наблюдаемости – $t^y_{\max}, n^y_t, k^y_i, \left|t^y_{ij}\right|$ каждой вершины транзакционного графа.

Для фрагмента графа, представленного на рис. 8, преобразование конъюнктивной формы в дизъюнктивную структуру для вершины V_3 по выражению (7) формирует логическую функцию управляемости:

$$V_3 = (T_1 \vee T_2 \vee T_3)T_5T_8 \vee (T_4 \vee T_6)T_8 \vee (T_7 \vee T_9) =$$
$$= T_1T_5T_8 \vee T_2T_5T_8 \vee T_3T_5T_8 \vee T_4T_8 \vee T_6T_8 \vee T_7 \vee T_9;$$
$$V2 = (T_1 \vee T_2 \vee T_3)T_5 \vee T_4 \vee T_6 =$$
$$= T_1T_5 \vee T_2T_5 \vee T_3T_5 \vee T_4 \vee T_6;$$
$$V_1 = T_1 \vee T_2 \vee T_3.$$

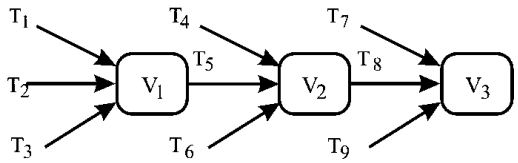

Рисунок 8 - Фрагмент графа для подсчета управляемости

Следуя правилам формулы (8), определяется управляемость компонента V_3:

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

Для другого фрагмента графа, представленного на рис. 9, преобразование конъюнктивной формы в дизъюнктивную структуру позволяет определить логическую функцию наблюдаемости вершины V_1:

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

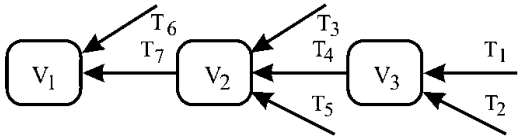

Рисунок 9 - Фрагмент графа для подсчета наблюдаемости

По правилам (8) также определяется управляемость компонента V_1:

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

Для случая, когда дуги в транзакционном графе имеют весовые коэффициенты, показывающие число операторов, задействованных в передаче информации между вершинами (мультидуги), формулы подсчета тестопригодности несколько усложняются:

$$U_i = \frac{\sum_{i=1}^{n^x_t} \prod_{j=1}^{k^x_i} b^x_{ij} (t^x_{\max} - \left|t^x_{ij}\right| + 1)}{t^x_{\max} \times (\sum_{i=1}^{n^x_t} \prod_{j=1}^{k^x_i} b^x_{ij})};$$

$$N_i = \frac{\sum_{i=1}^{n^y_t} \prod_{j=1}^{k^y_i} b^y_{ij} (t^y_{\max} - \left|t^y_{ij}\right| + 1)}{t^y_{\max} \times (\sum_{i=1}^{n^y_t} \prod_{j=1}^{k^y_i} b^y_{ij})}. \qquad (9)$$

Верификация DCT IP-core, Xilinx

Представленные модели верификации программного HDL-кода проверены на реальном проекте Xilinx IP-core в целях определения наличия в нем ошибок. При этом удалось получить положительный результат относительно неверной семантики работы программы для последующего исправления кода. Фрагмент модуля дискретного косинусного преобразования представлен листингом 1 [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 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 транзакционный граф, представленный на рис. 10, который для module Xilinx имеет 28 вершин-компонентов (входная и выходная шины, логические и регистровые переменные, векторы и память).

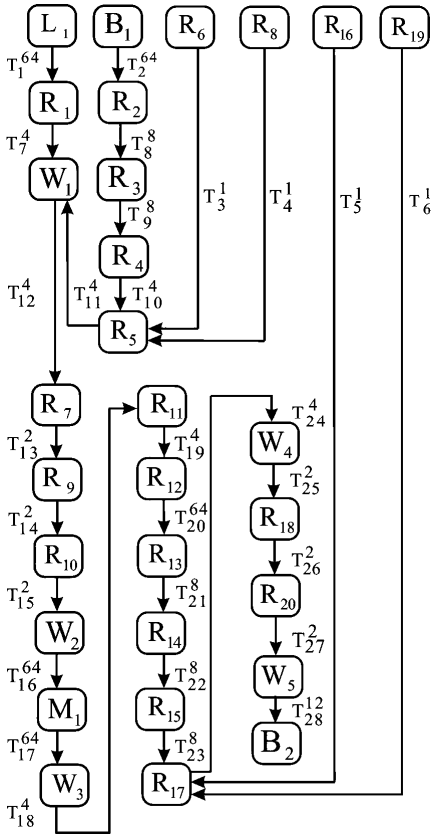

Рисунок 10 - Транзакционный граф Xilinx модели

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

$$U(B_2) = T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}(T_5^1 \vee T_6^1 \vee T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}(T_1^{64}T_7^4 \vee T_{11}^{4}T_2^{64}T_8^8T_9^8T_{10}^4 \vee T_{11}^{4}T_3^1 \vee T_{11}^{4}T_4^1)) =$$

$$= T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_5^1 \vee T_6^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4} \vee T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}T_1^{64}T_7^4 \vee$$

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

$$\vee T_{11}^{4}T_3^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8} \vee T_{11}^{4}T_4^1T_{28}^{12}T_{27}^{2}T_{22}^{2}T_{25}^{2}T_{24}^{4}T_{12}^{4}T_{13}^{2}T_{14}^{2}T_{15}^{2}T_{15}^{64}T_{17}^{64}T_{18}^{4}T_{19}^{4}T_{20}^{64}T_{21}^{8}T_{22}^{8}T_{23}^{8}.$$

Для вершины L_1 ДНФ функции наблюдаемости имеет вид:

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

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 (8), можно определить критерии управляемости (наблюдаемости) для всех компонентов HDL-модели. Здесь можно рассматривать два варианта (сценария) обсчета программной модели. 1) Учитывается только графовая структура, где вес каждой дуги равен 1, независимо от числа транзакций в программном коде. 2). Все дуги графа отмечаются реальным количеством транзакций, имеющих место быть между двумя вершинами транзакционного графа. Оценки тестопригодности описанных процедур могут существенно различаться друг от друга. Пользователь должен определиться, что важнее только структура программного кода – применить первый сценарий, или иметь более сложную и точную модель транзакций, распределенных во времени, на множестве графовых компонентов. В качестве примера ниже приводится процедура вычисления управляемости для B_2:

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

Применение аналогичных вычислений управляемостей (наблюдаемостей) для других вершин графа (рис. 10) дает результат в виде графика, представленного на рис. 11, которые позволяют определить крити-

ческие точки для установки необходимых ассерций. Такой вершиной может быть компонент R_{15}, если транзакционный граф представлен одиночными дугами. Для случая, когда дуги отмечены реальным количеством транзакций, критические вершины принадлежат компонентам, находящимся ближе к выходной шине B_2. Здесь существенным представляется не структура графа, а вес входящей дуги, который в большей степени оказывает негативное влияние, если структурная глубина рассматриваемого компонента достаточно высока. Используется формула (8) вычисления тестопригодности с мультипликативными членами $U_i \times N_i$, что дает оценку ниже, чем любой из сомножителей (управляемость, наблюдаемость).

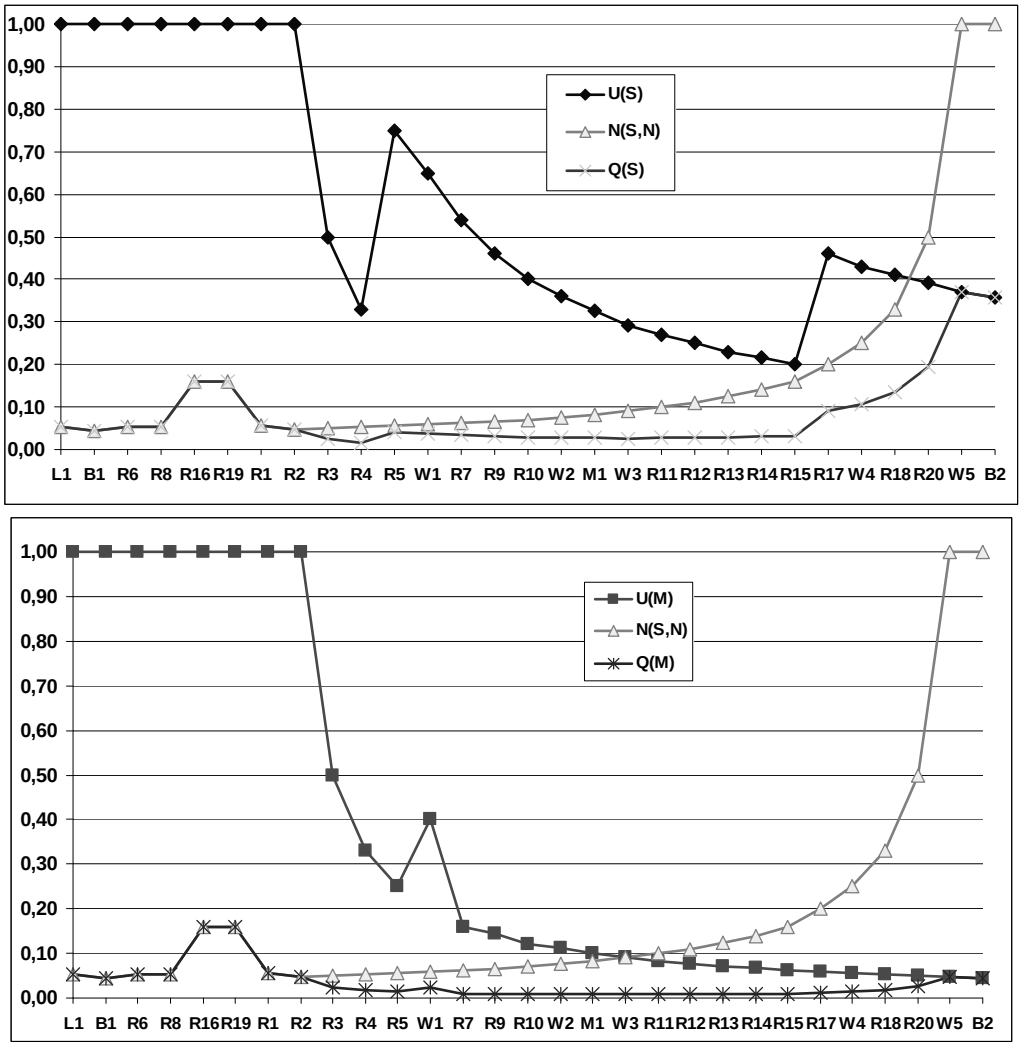

Рисунок 11 - Графики тестопригодности Xilinx модели

После определения управляемостей и наблюдаемостей вершин транзакционного графа выполняется подсчет обобщенного критерия тестопригодности программного кода по формуле (8). Для Xilinx 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 был выполнен анализ тестопригодности программного кода дискретного косинусного преобразования из Xilinx библиотеки. Построена транзакционная модель, вычислены характеристики тестопригодности и определены критические точки. В соответствии с числом и типами компонентов было 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 транзакционного графа, 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 Questa, Mentor Graphics 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

Рассмотрены инновационные технологии тестопригодного проектирования программных и аппаратных продуктов [1-17], ориентированные на существенное уменьшение времени верификации системных программных моделей цифровых изделий на кристаллах.

1. Показаны основные направления использования технологий тестопригодного проектирования цифровых систем на кристаллах в задачах тестирования и верификации программных HDL-моделей.

2. Представлена универсальная модель программного компонента в виде транзакционного графа, на котором решаются задачи анализа и оценки тестопригодности в целях определения критических точек для установки ассерций и достижения требуемой глубины диагностирования HDL-кода.

3. Предложены логические функции тестопригодности HDL-моделей на основе использования транзакционного графа в целях вычисления оценок управляемости и наблюдаемости программных HDL-моделей, что дает возможность определить критические точки для последующего улучшения кода проекта путем установки ассерций.

4. Показаны примеры анализа тестопригодности путем подсчета управляемости и наблюдаемости транзакционного и управляющего графов в целях определения критических точек с последующим решением практической проблемы поиска и устранения ошибок в реальном DSP-проекте от компании Xilinx.

5. 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 в целях уменьшения времени разработки (time-to market) и повышения выхода годной продукции (yield).

6. Дальнейшие исследования направлены на разработку стандартных интерфейсов для последующей инте-

грации моделей, методов и программных продуктов в технологические маршруты проектирования цифровых систем на кристаллах.

Литература

1. *Abramovici M., Breuer M.A. and Friedman A.D.* Digital System Testing and Testable Design Computer Science Press, – 1998.– 652 p.
2. *Bayraktaroglu Ismet, Orailoglu Alex.* 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. *Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli.* A Platform-Based taxonomy for ESL Design // Design&Test of computers.–September-October. - 2006.– P. 359-373.
4. *Francisco DaSilva, Yervant Zorian, Lee Whetsel, Karim Arabi, Rohit Kapur.* Overview of the IEEE P1500 Standard // ITC International Test Conference. - 2003. - P.988–997.
5. *Rashinkar P., Paterson P., Singh L.* System-on-chip Verification: Methodology and Techniques.–Kluwer Academic Publishers, 2002.– 324 p.
6. *Zorian Yervant.* What is Infrustructure IP? // IEEE Design & Test.– 2002.– P. 5-7.
7. *Zorian Yervant, Gizopoulos Dmytris*. Gest editors' introduction: Design for Yield and reliability // IEEE Design & Test of Computers.– 2004.– P. 177-182.
8. *Zorian Yervant.* Guest Editor's Introduction: Advances in Infrastructure IP // IEEE Design and Test of Computers.– 2003.– P. 49.
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. *Frank Ghenassia.* Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems. Published by Springer.– 2005.– 282 p.
13. *Bergeron, Janick.* Writing testbenches: functional verification of HDL models.– Boston: Kluwer Academic Publishers.– 2001.– 354 p.
14. *Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale.* Verification Methodology. Manual for SystemVerilog.– Springer, 2005.– 528 p.
15. *Harry Foster, Adam Krolnik, David Lacey.* Assertion-based design. Second edition. Kluwer Academic Publishers.– Springer, 2005.– 392 p.
16. *Rashinkar P., Paterson P., Singh L.* System-on-chip Verification: Methodology and Techniques. Kluwer Academic Publishers, 2002.– 393 p.
17. *Meyer A.S.* Principles of Functional Verification. 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.

Резюме

Предлагается технология тестирования и верификации цифровых систем для диагностирования и исправления ошибок HDL-моделей, основанная на совместном использовании механизма ассерций и тестопригодного проектирования. Она позволяет существенно повысить качество проектируемых компонентов цифровых систем на кристаллах и уменьшить время разработки

Запропоновано технологію тестування й верифікації цифрових систем для діагностування та виправлення похибок HDL-моделей, яка спирається на сумісне використання механізму асерцій та тестопридатного проектування, цифрових систем на кристалах та зменьшити час розробки

Testing and verification technology for system HDL-models is proposed. It focused on the essential improvement the quality of SoC components (yield) and reduction the time-to-market through the use of simulation environment, testability analysis the logical structure of HDL-program for quasi-optimal placement of assertion engine

Ключевые слова: тестирование, тестопригодность, верификация, HDL-модель

Поступила18.09.2009 г.