КОМПЬЮТЕРНАЯ ИНЖЕНЕРИЯ И ТЕХНИЧЕСКАЯ ДИАГНОСТИКА

УДК681.326:519.613

# ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ HDL-МОДЕЛЕЙ КОМПОНЕНТОВ SOC. I

*ХАХАНОВ В.И., ЛИТВИНОВА Е.И., ЧУМАЧЕНКО С.В., ПОБЕЖЕНКО И.А., NGENE CHRISTOPHER UMERAH*

Предлагается технология тестирования и верификации системных HDL-моделей, ориентированная на существенное повышение качества проектируемых компонентов цифровых систем на кристаллах (yield) и уменьшение времени разработки (time-to-market) путем использования среды моделирования, тестопригодного анализа логической структуры HDL-программы для квазиоптимального размещения механизма ассерций.

## 1. Введение

Технология тестирования и верификации системных HDL-моделей 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, определяемые с помощью синтезированных логических функций тестопригодности. Таким образом, используемые в hardware design and test критерии управляемости и наблюдаемости 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 совместного использования механизма ассерций и технологий тестопригодного проектирования.

*Задачи исследования*: 1. Классификация технологий тестопригодного проектирования HDL-моделей для создания цифровых систем на кристаллах. 2. Разработка модели верификации и тестирования системной HDL-модели на основе использования ассерций. 3. Разработка метрики оценивания тестопригодности HDL-моделей на основе новой логической функции тестопригодности. 4. Применение технологической модели ассерций для верификации IP-core фильтра на основе дискретного косинусного преобразования. 5. Практические результаты и направления дальнейших исследований.

Источники исследования: 1. Технологии и средства создания тестов и 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 [11-17].

## 2. 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 адаптируются 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] на уровне платы и кристалла привели к появлению механизма ассерций для тестирования и верификации программных продуктов.

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] 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 [6] 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) для reusable стандартизованных функциональных компонентов используется для генерирования тестов программных модулей на основе наработанных библиотек от ведущих компаний планеты.

10. Стандартные решения сервисного обслуживания функциональностей F-IP в рамках инфраструктуры I-IP [12] 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, 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 заключается 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. 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 разработ-

ки; 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.

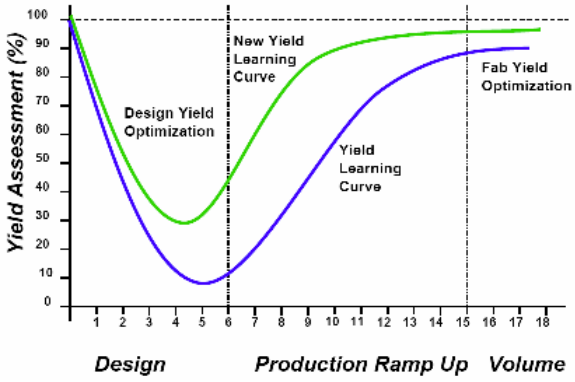


Рис. 1. График жизненного цикла программно-аппаратного комплекса

Согласно [13] 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] – 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).

### 3. Инфраструктура процесса верификации 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:

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

Здесь: $T, F$ – тестовые воздействия и функциональное покрытие эталонной модели с ожидаемыми реакциями; $S, A$ – 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 [13,14]. Верификация – есть процесс анализа системы или компонентов для определения корректности формальных преобразований входного описания на каждой стадии проектирования. 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 $A_i = T_i \oplus S_i \rightarrow d(A_i)$, который на множестве состояний эталонной и реальной моделей принимает значения $\{\text{true} - 1, \text{false} - 0, \text{don't care} - X\}$. 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 и в стандартах граничного сканирования (IEEE 11.49, IEEE 1500), где тестирование компонентов цифровой системы регулируется контроллером тестового доступа (TAP).

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. 2). 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.

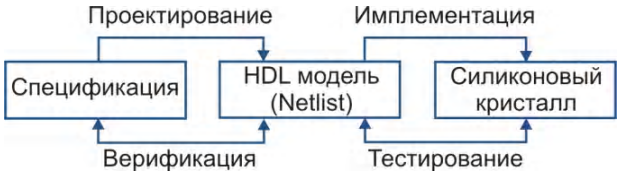


Рис. 2. Стратегия разработки проекта

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. 3, 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.

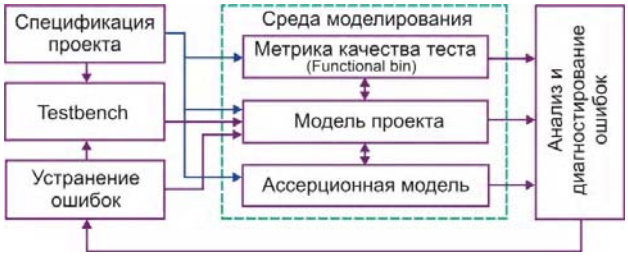


Рис. 3. Среда верификации проекта

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.

Интерес представляет и последовательность действий по созданию среды верификации, где единственным аргументом является спецификация проекта, все остальное – производные от нее. На рис. 4 изображена структура взаимосвязей процесса проектирования и диагностирования для исправления ошибок в HDL-коде программы.

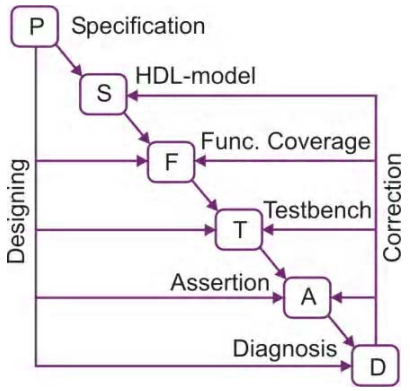


Рис. 4. Маршрут верификации проекта

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 достаточно простые условия тестопригодности (см. рис. 4). 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.

## 4. 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):

$$
\begin{aligned}
&M = \{P, S, A, T, F, d, C\},\\
&1)\ S = f_1(P) = (S_1, S_2, ..., S_i, ..., S_n);\\
&2)\ F = f_2(P, S) = \{F_1, F_2, ..., F_i, ..., F_n\};\\
&3)\ T = f_3(P, S, F) = \{T_1, T_2, ..., T_i, ..., T_n\};\\
&4)\ A = f_4(P, S, F, T) = (A_1, A_2, ..., A_i, ..., A_n);\\
&5)\ d = f_5(P, S, F, T, A) =\\
&= (L_1, L_2, ..., L_i, ..., L_n) \in \{L_s, L_h\}; \qquad (2)\\
&6)\ C = [\bigcup_{i=1}^{n} F_i \in F = P] \wedge [\bigcup_{i=1}^{n} T_i \in T = F];\\
&7)\ L_s = (T, F) \oplus (S, A);
\end{aligned}
$$

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 путем использования атрибутов верификационной инфраструктуры.

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]:

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

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:

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

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).

## 5. Анализ тестопригодности программных HDL-моделей

Достаточно существенная избыточность HDL-модели предполагает ее эффективное использование в целях повышения тестопригодности структуры разработанного или написанного кода. Существующие стандарты тестопригодного проектирования аппаратуры можно адаптировать 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 моде-

ли 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:

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

$$U_i = \frac{1}{T}\sum_{j=1}^{x_i} T_j^i \times \frac{1}{d_i^x \vee t_i^x}; \qquad (5)$$

$$N_i = \frac{1}{T}\sum_{j=1}^{y_i} T_j^i \times \frac{1}{d_i^y \vee t_i^y}.$$

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 – $d_i^x \vee t_i^x$ ( $d_i^y \vee t_i^y$ – 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 (5), 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 граф-схемы управления вычислительным процессом. Здесь 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. 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 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 (6):

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

$$2)\ U_r^f = \bigvee_{i=1}^{x_r}\Big(\bigwedge_{j=1}^{n_i^x} T_{ij}^x\Big);\ N_r^f = \bigvee_{i=1}^{y_r}\Big(\bigwedge_{j=1}^{n_i^y} T_{ij}^y\Big). \qquad (6)$$

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 (6), 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 использовании аппарата – алгебраической формы представления графа [18]. Формулы подсчета упомянутых оценок имеют следующий вид:

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

$$N_i = \frac{1}{t_{max}^y \times n_t^y} \times \sum_{i=1}^{n_t^y}\sum_{j=1}^{k_i^y}\left(t_{max}^y - \left|t_{ij}^y\right| + 1\right); \qquad (7)$$

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

где $t_{max}^x, n_t^x, k_i^x, \left|t_{ij}^x\right|$ – 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. 5, преобразование конъюнктивной формы в дизъюнктивную структуру для вершин $V_1 - V_3$ по выражению (6) формирует логические функции управляемости:

$$U^f(V_1) = T_1 \vee T_2 \vee T_3;$$

$$U^f(V_2) = (T_1 \vee T_2 \vee T_3)T_5 \vee T_4 \vee T_6 =$$

$$= T_1T_5 \vee T_2T_5 \vee T_3T_5 \vee T_4 \vee T_6;$$

$$U^f(V_3) = (T_1 \vee T_2 \vee T_3)T_5T_8 \vee (T_4 \vee T_6)T_8 \vee (T_7 \vee T_9) =$$

$$= T_1T_5T_8 \vee T_2T_5T_8 \vee T_3T_5T_8 \vee T_4T_8 \vee T_6T_8 \vee T_7 \vee T_9.$$

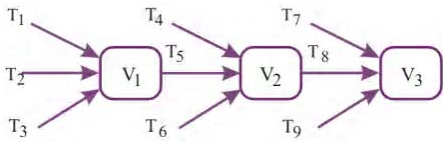


Рис. 5. Фрагмент графа для подсчета управляемости

Следуя правилам формулы (7), 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. 6, преобразование конъюнктивной формы в дизъюнктивную структуру позволяет определить логическую функцию наблюдаемости вершины $V_1$:

$$N^f(V_1) = T_6 \vee T_7(T_3 \vee T_5 \vee T_4(T_1 \vee T_2) =$$

$$= T_6 \vee T_7T_3 \vee T_7T_5 \vee T_7T_4T_1 \vee T_7T_4T_2;$$

$$N^f(V_2) = T_3 \vee T_5 \vee T_4(T_1 \vee T_2) =$$

$$= T_3 \vee T_5 \vee T_4T_1 \vee T_4T_2;\ V_3 = T_1 \vee T_2.$$

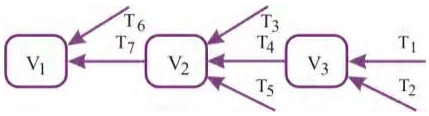


Рис. 6. Фрагмент графа для подсчета наблюдаемости

По правилам (7) 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:

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

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

## 6. Выводы

Рассмотрены инновационные технологии тестопригодного проектирования программных и аппаратных продуктов [1-17], ориентированные на эффективную разработку тестов и верификацию компонентов цифровых систем на кристаллах.

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

2. Представлена универсальная модель программного компонента в виде транзакционного графа, на котором можно решать задачи анализа тестопригодности в целях достижения требуемой глубины диагностирования HDL-кода.

3. Предложены логические функции тестопригодности HDL-моделей на основе использования транзакционного графа в целях вычисления оценок тестопригодности (управляемость и наблюдаемость) программных компонентов и всего HDL-проекта в целом.

4. Приведены примеры и графики оценивания тестопригодности (управляемости и наблюдаемости) программных моделей, представленных фрагментами транзакционных графов.

**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 of Computers. 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. 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 Laboratory. 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.

Поступила в редколлегию 14.06.2009

**Рецензент:** д-р техн. наук, проф. Кривуля Г.Ф.

**Хаханов Владимир Иванович**, д-р техн. наук, профессор кафедры АПВТ, декан факультета КИУ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем, сетей и программных продуктов. Увлечения: баскетбол, футбол, горные лыжи. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326, e-mail: hahanov@kture.kharkov.ua.

**Литвинова Евгения Ивановна**, канд. техн. наук, доцент кафедры технологии и автоматизации производства РЭС и ЭВС ХНУРЭ. Научные интересы: автоматизация диагностирования и встроенный ремонт компонентов цифровых систем в пакете кристаллов. Адрес: Украина, 61166, Харьков, пр. Ленина 14, тел. 70-21-421, e-mail: kiu@kture.kharkov.ua.

**Чумаченко Светлана Викторовна,** д-р техн. наук, проф. кафедры АПВТ ХНУРЭ. Научные интересы: математическое моделирование, методы дискретной оптимизации. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326, e-mail: ri@kture.kharkov.ua.

**Побеженко Ирина Александровна**, аспирантка кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326, e-mail: hahanov@kture.kharkov.ua.

**Ngene Christopher Umerah**, аспирант кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326, e-mail: hahanov@kture.kharkov.ua.