УДК 681.326:519.713

В.И. ХАХАНОВ, И.А. ПОБЕЖЕНКО, В.А. ВАСИЛЕНКО, С.В. ЧУМАЧЕНКО

МЕТОД ВЕРИФИКАЦИИ HDL-КОДА НА ОСНОВЕ ТРАНЗАКЦИОННОГО ЛОГИЧЕСКОГО ГРАФА

Предлагается логический метод диагностирования ошибок программного HDL-кода, который использует транзакционный граф программы и ее компонентов, что позволяет определять критические точки для мониторинга выполнения программы в целях установки в них ассерционных операторов, ориентированных на существенное (40%) уменьшение времени верификации программной модели изделия. Приводятся примеры синтеза транзакционного графа и диагностирования ошибок программных модулей, подтверждающие эффективность практического использования метода.

Стоимость и временные затраты на создание проекта распределяются: на написание кода – 30%, на тестирование и верификацию – 70%. Поэтому тестовое обеспечение и сервисное обслуживание проекта и готового изделия есть важные составляющие обеспечения качества цифровой системы на кристалле, которые закладываются уже на стадии системного проектирования в виде инфраструктуры сервисного обслуживания, включающей тестопригодность, применение стандартов IEEE 1500, 11.49, 1450, механизмы ассерций, транзакционные графы управляющего и операционного автоматов, а также программных кодов. Обеспечение тестопригодности есть не что иное, как дополнительные аппаратурные и программные затраты, а также избыточное время для их создания. Тем не менее, издержки, составляющие порядка 5% от полезной функциональности, дают существенный эффект в части: повышения выхода годной продукции – 3-4%; уменьшения общего времени создания продукта – 35%; повышения производительности средств тестирования и верификации – x10 раз.

Предлагается метод верификации системных HDL-моделей, ориентированный на существенное повышение качества проектируемых компонентов цифровой системы, выполняющей функцию вейвлет-преобразования, что также позволяет уменьшить время разработки (time-to-market) путем использования программной избыточности в виде механизма ассерций. Методы ориентированы на поиск ошибок и дефектов с заданной глубиной в программном HDL-коде путем введения в критические точки транзакционной модели наблюдателя в виде ассерционной избыточности. Для определения критических точек используется известная в hardware design and test технология вычисления управляемости и наблюдаемости структурных компонентов программного кода в целях улучшения его тестопригодности для диагностирования семантических ошибок.

Цель – разработка логического метода диагностирования ошибок программного кода на основе синтеза транзакционного графа, позволяющего определять критические точки мониторинга HDL-кода для установки в них механизма ассерций, что позволяет на 40% уменьшить время верификации программной модели изделия.

Задачи исследования: 1. Описание современных технологий и маршрутов тестопригодного проектирования HDL-моделей на основе ассерционной избыточности, использования обязательных компонентов Testbench и Coverage. 2. Разработка аналитической модели верификации HDL-кода на основе механизма ассерций. 3. Создание метрики для оценивания тестопригодности HDL-кода на основе синтеза транзакционного графа. 4. Создание новой иерархической модели программного кода вейвлет-преобразования в целях подсчета тестопригодности для имплементации ассерций в критические точки мониторинга. 5. Синтез таблиц неисправностей для программных моделей вейвлет-преобразования в целях их анализа для поиска ошибок и дефектов. 6. Практическое использование логического метода для поиска семантических ошибок HDL-кода.

Источники исследования: 1. Модели, методы и средства создания тестов и testbench [10-12]. 2. Средства верификации системных моделей на основе механизма ассерций [13-16]. Стандарты тестопригодного проектирования программных продуктов от общества IEEE [17-19]. Современные разработки для верификации и анализа тестопригодности цифровых систем на кристаллах [7, 8, 14, 20-23].

1. Актуальные технологии тестопригодности HDL-кода

В плане предлагаемых исследований интересной представляется опубликованная в EE Times (12.2009) десятка перспективных технологий от Gartner Research Group (Gary Smith) для ближайших лет. В ней нашли отражение только технологии, связанные с разработкой специализированных цифровых изделий на кристаллах, хотя развитие программных технологий также будет оказывать сильное влияние на состояние рынка электроники. Глобально важными остаются технологии снижения потребляемой мощности и решения, направленные на уменьшение содержания ценных материалов в продукте. Данные технологии выступают двигателями многих направлений развития электроники, перечисленных ниже: 1. Биологическая обратная связь или электроника, управляемая мыслью. 2. Печатная электроника на основе использования органических материалов. 3. Пластиковая память на основе полимеров, проявляющих ферроэлектрические свойства. 4. Безмасочная литография на основе использования электронного луча для создания топологии схемы. 5. Параллельная обработка данных для многоядерных гетерогенных графических процессоров. 6. Сбор энергии от механических и электрических процессов в окружающей среде. 7. Биоэлектроника и wetware, сочетающие биологические объекты и электронику для медицины. 8. Резистивное ОЗУ или мемристор с эффектом памяти как четвертый пассивный элемент электронной схемы, дополняющий резистор, конденсатор и индуктивность. 9. Переходные отверстия в кремнии (Through-Silicon-Via – TSV) для создания реальных 3D-SiP и кристаллов. 10. Различные технологии батарей на основе сочетания никеля и лития.

За последние 30 лет происходит взаимовыгодный обмен технологиями между аппаратными и программными разработчиками, в качестве которых выступают ведущие компании планеты (Intel, Microsoft, Cadence, Mentor Graphics, Synopsys, Aldec). Следует привести некоторые отдельные факты плодотворного сотрудничества. 1. IEEE стандарты граничного сканирования [12] на уровне платы и кристалла породили механизм ассерций для верификации программных продуктов. 2. Тестопригодность [1,2], управляемость и наблюдаемость цифровых структур адаптирована для оценки качества программного кода и последующего его улучшения для быстрой отладки модели. 3. Графовые модели регистровых передач [18] адаптируются для оценки и улучшения тестопригодности программных продуктов путем установки ассерций. 4. Разделение автомата на управляющую [10, 16] и операционную части используется для упрощения процесса верификации программного кода, который фактически имеет аналогичные составляющие. 5. Компонент testbench [14, 9], используемый для тестирования аппаратных проектов, появляется и в программных продуктах, реализованных на уровне языков C++ и выше. Testbench – специализированная структура операционного устройства, реализованная в HDL-коде и предназначенная для тестирования и верификации цифрового проекта с помощью средств моделирования, ассерционной избыточности, а также компонентов, обеспечивающих управление, наблюдение и принятие решения о техническом состоянии проверяемого объекта. Он включает сгенерированные вручную или автоматически входные и выходные данные, описывающие идеальную модель устройства и составляющие основу testbench. 6. Инфраструктуры сервисного обслуживания F-IP в рамках I-IP [11] используются для встроенного тестирования компонентов программной системы на основе механизма ассерций. 7. Адресность компонентов SoC, реализованных в аппаратном исполнении [18-20], предоставляет также и программному продукту свойство самовосстановления работоспособности средствами I-IP на расстоянии, благодаря наличию связи кристалла с внешним миром посредством Internet или беспроводных технологий. Дистанционная коррекция программных ошибок возможна благодаря использованию ПЛИС, куда, в случае обнаружения неисправности, можно записать новый bit stream, не имеющий ошибок, что фактически создает новую аппаратуру путем повторного программирования кристалла.

Жизненный цикл программного или аппаратного изделия представлен на рис. 1. Он имеет стадии: проектирование, усовершенствование проекта и производства, сопровождение изделия. Здесь важно найти новые средства для поднятия кривой вверх и ее сжатия по оси времени за счет: быстрого устранения ошибок разработки; исправления кода, имплементированного в память системы на кристалле; выпуска service pack, корректирующего ошибки путем его распространения через Internet или спутники.

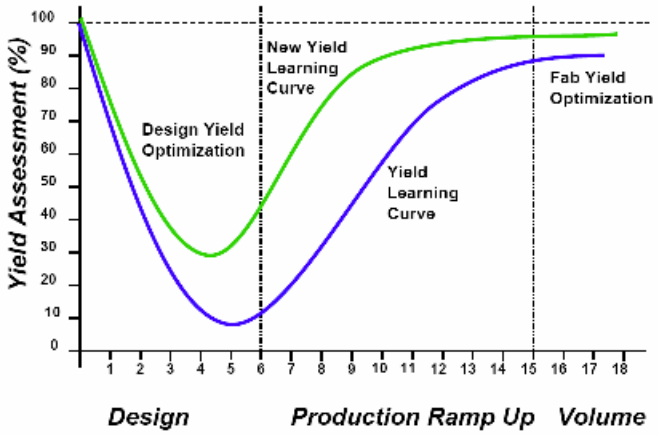

Рис. 1. Жизненный цикл программно-аппаратного продукта

Сущность данного исследования заключается в повышении уровня годной продукции и уменьшении времени создания проекта за счет имплементации в HDL-код программной избыточности в виде механизма ассерций [4, 5]. Для этого необходимо синтезировать транзакционные графы программных модулей, выполнить анализ их тестопригодности, найти критические точки мониторинга, установить в них ассерционные операторы. Все упомянутое выше дает возможность уменьшить время отладки программного проекта в среднем на 40% и повысить выход годной продукции на 3-4%.

2. Инфраструктура процесса верификации проекта

Используется уравнение обнаружения ошибок или дефектов на уровне системы или программных компонентов (T, A – тестовые и ассерционные воздействия с ожидаемыми реакциями; P, S, F – спецификация, HDL-модель функциональное покрытие оценки полноты теста для верификации проектируемого изделия): $T \oplus S = L, (T, A) \oplus (P, S, F) = L$. Для более точного понимания соотношений между ключевыми понятиями вводятся определения [5, 6]. Верификация – процесс анализа программных компонентов для определения правильности преобразований входного описания на очередной стадии проектирования. Валидация – определение работоспособности программных компонентов путем проверки соответствия требованиям спецификации на каждой стадии проектирования.

Ассерция – есть инверсное HDL-высказывание системного уровня, предназначенное для раннего определения ошибок проектирования относительно требований спецификации при моделировании проекта на тестовых воздействиях до и после выполнения синтеза. Предельное число ассерций есть диверсная модель проекта, на практике число ассерционных операторов составляет не более 5% HDL-кода проекта. Существует практически стандарт использования ассерций в среде верификации, представленной на рис. 2.

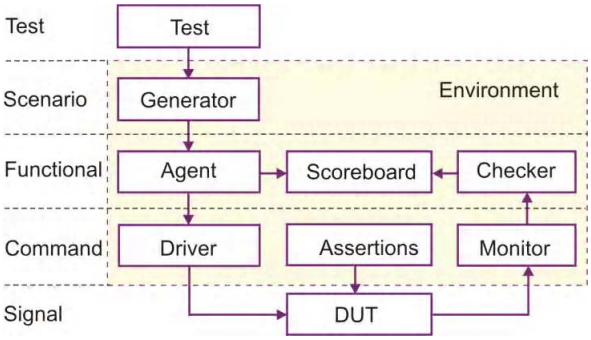

Рис. 2. Иерархическая среда верификации

Компоненты каждого уровня иерархии представляют собой транзакторы. Тестовый уровень иерархии состоит из слоев: 1) Test – компонент верхнего уровня, задает ограничения генератору последовательностей и конфигурирует режимы моделирования при каждом запуске. 2) Уровень сценариев содержит генератор (generator), который создает последовательности с ограничениями, полученными от test, а также все сценарии по псевдослучай-

ной генерации тестов. 3) Функциональный уровень формирует структуры данных для их обработки и передачи на командный (command) слой. Здесь проверяется правильность данных, выдаваемых DUT, с помощью компонентов: агент (agent), счетчик (scoreboard) и анализатор (checker).

Агент используется для обработки данных, передаваемых драйверу (driver). Счетчик верификационной среды содержит высокоуровневую «золотую» модель проекта. Он получает одинаковые с DUT тестовые последовательности и определяет корректную реакцию DUT. Анализатор сравнивает выходную информацию из DUT, доставленную посредством монитора (monitor) с ожидаемой эталонной реакцией, полученной от счетчика. 4) Командный уровень обеспечивает связь с DUT. Драйвер управляет значениями на входах DUT. Монитор следит за выходной информацией из DUT и отправляет ее анализатору. Внешние ассерции используются как составляющая часть проекта командного уровня. 5) Сигнальный (signal) уровень содержит верифицируемое устройство и интерфейс входных, выходных сигналов. Ассерции – блоки, добавляемые в исходный код проекта для наблюдения и управления поведением модели проекта. Они могут быть представлены операторами if для сообщений об ошибках, проявляющихся в процессе тестирования DUT. Ассерции создаются разработчиком или могут быть взяты из существующих библиотек для проверки типовых функций. Фирма Synopsys поставляет систему моделирования VCS с библиотекой SystemVerilog-ассерций. Одно из направлений – это обеспечение полной поддержки актуального стандарта SystemVerilog'2009. Инфраструктура интегрирует опыт и технологии всех ведущих компаний планеты в области тестирования и верификации цифровых проектов. Здесь основные компоненты: семейство языковых средств, ориентированных на ввод проекта; компиляторы с языков описания аппаратуры; средства отладки проекта; ориентированные на пользователя средства управления проектом и компоненты визуализации результатов моделирования.

Иначе, ассерция – предикат системного уровня, определяющий некорректности процесса проектирования относительно требований спецификации. Механизм ассерций совместно с HDL-моделью, тестом и функциональным покрытием представляет собой структуру, на которой можно формировать алгоритмы диагностирования дефектов или принятия решений о существовании ошибки в программном компоненте.

Аналитическая модель описания и решения логических уравнений для верификации HDL-кода представлена системой предикатов:

$$
\begin{aligned}
&P(m,A) = Q(m,A) = [0{,}0-1{,}0]; \qquad Q(m,A) = \frac{1}{3}[d(m,A) + \mu(m \in A) + \mu(A \in m)];\\
&P(m,A) = 1 \leftarrow m = A; \qquad Q(m,A) = \{Q_1, Q_2, ..., Q_i, ..., Q_n\};\\
&P(m,A) = \max_i Q_i(m,A); \qquad A = (A_1, A_2, ..., A_i, ..., A_m);\\
&P(m,A) = m \Delta (A_1, A_2, ..., A_i, ..., A_m); \qquad \Delta = \{\wedge, \vee, \neg\};\\
&A_i = (A_{i1}, A_{i2}, ..., A_{ij}, ..., A_{is}); \qquad A_{ij} = (A_{ij1}, A_{ij2}, ..., A_{ijr}, ..., A_{msq});\\
&m = (m_1, m_2, ..., m_r, ..., m_q); \qquad P(m,A) = m \wedge \{A_1, A_2, .., A_i, ..., A_m\} = p;\\
&m \wedge (A_1 \vee A_2 \vee ... \vee A_i \vee ... \vee A_m) = \max_i Q_i(m,A), \text{ (GRAF)};\\
&p = \{p_1, p_2, ..., p_r, ..., p_n\}; \qquad p = m \wedge (A_1 \vee A_2 \vee ... \vee A_i \vee ... \vee A_m);\\
&p_r(\in p) = m \underset{i=\overline{1,m}}{\overset{j=\overline{1,s}}{\wedge}} A_{ij} \leftarrow \begin{cases} \forall i \exists j \, (m \wedge A_{ij} \neq \varnothing); \\ \exists i \forall j (m \wedge A_{ij} = \varnothing) \wedge d(m, A_{ij}) = \max; \end{cases}\\
&m \wedge A_{ij} = m_r \overset{q}{\underset{r=1}{\wedge}} A_{ijr} = \begin{cases} m \leftarrow m_r \overset{q}{\underset{r=1}{\wedge}} A_{ijr} = m_r; \\ A_{ij} \leftarrow m_r \overset{q}{\underset{r=1}{\wedge}} A_{ijr} = A_{ijr}; \\ m \wedge A_{ij} \leftarrow m_r \overset{q}{\underset{r=1}{\wedge}} A_{ijr} = m_r \vee A_{ijr}; \\ \varnothing \leftarrow \exists (m_r \overset{q}{\underset{r=1}{\wedge}} A_{ijr} = \varnothing); \end{cases} \qquad (1)\\
&P(m,A) = p_r(\in p) \leftarrow Q_i(m, p_r) = \max, \; r = \overline{1,n}.
\end{aligned}
$$

Здесь определены предикаты: 1) $P(m,A)=Q(m,A)$ – задает аналитическую модель вычислительного процесса в виде предиката, определенного интегральным критерием принадлежности вектора экспериментальной проверки множеству технических состояний, одно из которых есть исправное, в интервале $Q(m,A)=[0,0-1,0]$, на совокупности введенных операций: объединение, пересечение и дополнение; 2) граф-упорядоченная совокупность $A=(A_1,A_2,...,A_i,...,A_m)$ взаимодействующих таблиц неисправностей программных компонентов, которые объединяются на верхнем уровне обобщенным графом целевой функциональности; 3) упорядоченная совокупность вектор-строк таблицы неисправностей каждого программного компонента $A_i=(A_{i1},A_{i2},...,A_{ij},...,A_{is})$, каждая из которых $A_{ij}=(A_{ij1},A_{ij2},...,A_{ijr},...,A_{msq})$ принимает значения из троичного алфавита $\{0,1,x\}$. Предикат $A_i=(A_{i1},A_{i2},...,A_{ij},...,A_{is})=1$ представлен совокупностью векторов, формирующих многозначную таблицу неисправностей. Вектор $A_{ij}=(A_{ij1},A_{ij2},...,A_{ijr},...,A_{msq})$ определяет собой решение – совокупность неисправных или ошибочных программных модулей, проверяемых тестом $T_{ij}\in T$, где каждая переменная задается в троичном алфавите $A_{ijr}\in\{0,1,x\}=\beta$. Взаимодействие $P(m,A)$ входного вектора запроса (экспериментальной проверки) $m=(m_1,m_2,...,m_r,...,m_q)$ с графом таблиц неисправностей $A=(A_1,A_2,...,A_i,...,A_m)$ формирует множество решений $p=\{p_1,p_2,...,p_r,...,p_n\}$, каждое из которых имеет интегральную оценку качества $Q(m,A)=\{Q_1,Q_2,...,Q_i,...,Q_n\}$. Выбор лучшего из них осуществляется на основе критерия:

$$P(m,A)=p_r(\in p)\leftarrow Q_i(m,p_r)=\max,\ r=\overline{1,n}.$$

Критерий качества выбранного решения определяется в виде следующих аддитивных оценок:

$$\begin{aligned}
&Q=\frac{1}{3}[d(m,A)+\mu(m\in A)+\mu(A\in m)],\\
&d(m,A)=\frac{1}{n}\left(n-\left|m_i\bigcap_{i=1}^{n}A_i=\varnothing\right|\right);\\
&\mu(m\in A)=2^{|m\cap A|-|A|}\leftarrow|m\cap A|=\left|m_i\bigcap_{i=1}^{n}A_i=x\right|\ \&\ |A|=\left|\bigcup_{i=1}^{n}A_i=x\right|;\\
&\mu(A\in m)=2^{|m\cap A|-|m|}\leftarrow|m\cap A|=\left|m_i\bigcap_{i=1}^{n}A_i=x\right|\ \&\ |m|=\left|\bigcup_{i=1}^{n}m_i=x\right|.
\end{aligned}\qquad(2)$$

Интегральная метрика для оценивания качества запроса есть функция качества (выбора) взаимодействия векторов $m\cap A$, которая определяется средней суммой трех нормированных параметров: кодовое расстояние $d(m,A)$, функция принадлежности $\mu(m\in A)$ и эффективность использования входного запроса – функция принадлежности $\mu(A\in m)$. Нормирование параметров, входящих в метрику, позволяет оценивать уровень взаимодействия вектора экспериментальной проверки и таблиц неисправностей в интервале [0,1]. Если зафиксировано предельное максимальное значение каждого параметра, равное 1, то векторы равны между собой, что соответствует точному диагнозу. Минимальная оценка, Q = 0, фиксируется в случае полного несовпадения векторов по всем n координатам.

Инфраструктура – совокупность моделей, методов и средств описания, анализа и синтеза структур данных для решения функциональных задач.

Функционирование инфраструктуры диагностирования имеет интерфейс, который представлен на рис. 3. При подаче на вход вектора экспериментальной проверки m, маскированного двоичным вектором X, инфраструктура должна сформировать на выходе вектор A, максимально непротиворечивый запросу m, а также оценку качества полученного решения в виде функции $Q=f(m,A);\ A=g(m,A)$.

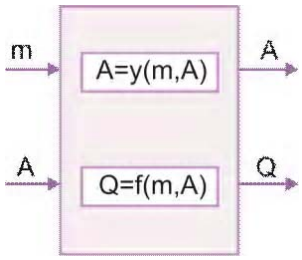

Рис. 3. Интерфейс системы верификации

Данная модель определяет индустриальный стандарт верификации программных продуктов и HDL-моделей, которого придерживаются все компании, которые входят в топ 200 мировых лидеров, формирующих индекс NASDAQ (2308,42 пункта – 04.01.2010) на Уолл-стрит. Поэтому очень важно все научные исследования ориентировать на имплементацию в существующие стандарты, маршруты и технологии мировых лидеров, что дает возможность выходить с практическими предложениями на рынок EDA. Что касается инфраструктуры сервисного обслуживания SoC вейвлет-преобразования, то она, несущественно отличаясь от приведенных моделей двухуровневой иерархией, имеет вид, представленный на рис. 4.

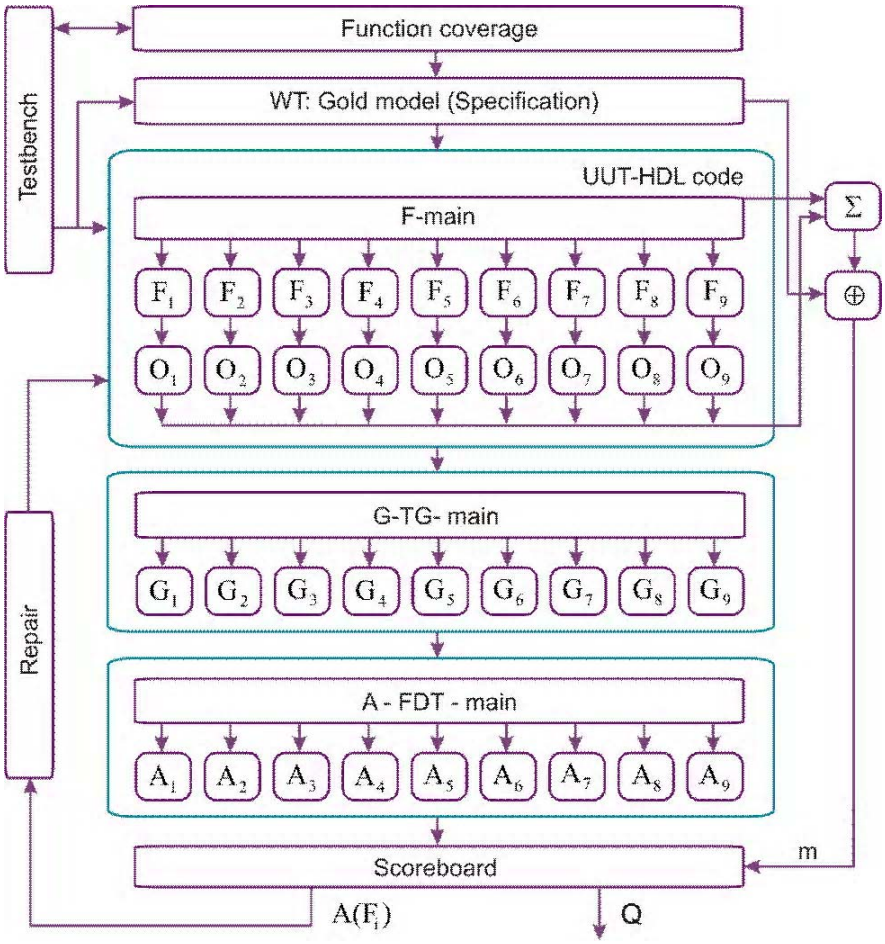

Рис. 4. Инфраструктура верификации WT SoC проекта

Здесь по спецификации строится модель проекта (9 программных модулей и одна основная программа), которая записывается в виде HDL-кода. Затем по модели строятся транзакционные графы, по которым находятся критические точки для мониторинга основного модуля и подпрограмм с помощью ассерционных операторов $O_i \in O$. Далее по тран-

закционным графам строятся таблицы (А) несправных модулей, которые используются для поиска и исправления ошибок. Качество теста гарантируется полнотой покрытия функциональностей, которая влияет на глубину диагностирования и качество проекта в целом. Система верификации и диагностирования содержит следующие компоненты: 1. Модуль ввода ассерций, представленный транслятором языка описания ассерций или проекта. 2. Внутренняя модель описания и обработки выражений в виде логико-временных отношений, алгоритмов диагностирования или директив верификации. 3. Модуль анализа ассерций для проверки заданных формальных логико-временных ограничений в процессе моделирования. 4. Блок синтеза квазиоптимальной модели анализа ассерций по имеющимся языковым конструкциям. 5. Структуры данных, обеспечивающие взаимодействие и совместимость между множеством ассерций и функциональностью в процессе компиляции и моделирования.

3. Анализ тестопригодности графов HDL-моделей

Граф HDL-модели создается путем семантического анализа кода для выявления: 1) всех вершин, определяемых как приемники и источники транзакций данных (регистр, счетчик, память или массивы, вход-выходные шины, векторы, логические или арифметические переменные), которые упоминаются в строках кода; 2) направления приема или передачи данных, формирующих множество дуг между вершинами. Дуги могут быть отмечены числом транзакций. Граф создает условия для оценки его тестопригодности, а значит и для определения качества структуризации кода программного продукта. Транзакционный граф записывается в виде алгебраической (логической) формы, что дает возможность не только получить все решения относительно достижимости вершины, но и оценивать тестопригодность каждой вершины и графа в целом.

Ассерционная избыточность HDL-модели должна быть эффективной в целях повышения тестопригодности структуры разработанного или написанного кода. Качество программного продукта на основе анализа графа транзакций (TG – Transaction Graph) определяется выражением:

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

$$U_i = \frac{1}{T}\sum_{j=1}^{x_i} T_j^i \times \frac{1}{d_i^x \vee t_i^x};\ N_i = \frac{1}{T}\sum_{j=1}^{y_i} T_j^i \times \frac{1}{d_i^y \vee t_i^y}. \qquad (3)$$

Качество зависит от управляемости U, наблюдаемости N, а также дополнительных затрат (в знаменателе функции Q) для формирования функциональной корзины, теста, ассерций. Управляемость (наблюдаемость) есть функция от числа операторов, входящих в вершину (исходящих из вершины) транзакционного графа, а также от структурной глубины рассматриваемой вершины относительно входной (выходной) шины

Логические функции управляемости и наблюдаемости каждой вершины транзакционного графа записываются в форме конъюнкции дизъюнктивных термов:

$$U_i = \frac{1}{T}(\bigwedge_{j=1}^{x_1} T_{1j}^x)\ldots(\bigwedge_{j=1}^{x_{i-1}} T_{i-1j}^x)(\bigvee_{i=1}^{x_i} T_{ij}^x) \quad N_i = \frac{1}{T}\bigwedge_{j=1}^{x_i} T_j^i(\bigvee_{i=1}^{x_i} T_j^i d_i^x \vee t_i^x). \qquad (4)$$

Число дизъюнктивных термов соответствует количеству входящих в вершину дуг, конъюнкций – структурной глубине компонента. Если управляемость и наблюдаемость вершины графа вычисляется на основании алгебраической формы представления графа [3], то формулы подсчета критериев имеют следующий вид:

$$U_i = \frac{1}{t_{max}^x \times n_t^x} \times \sum_{i=1}^{n_t^x}\sum_{j=1}^{k_i^x}(t_{max}^x - \left|t_{ij}^x\right| + 1);\ \ N_i = \frac{1}{t_{max}^y \times n_t^y} \times \sum_{i=1}^{n_t^y}\sum_{j=1}^{k_i^y}(t_{max}^y - \left|t_{ij}^y\right| + 1), \qquad (5)$$

где $t_{max}^x, n_t^x, k_i^x, \left|t_{ij}^x\right|$ – для критерия управляемости конъюнктивный терм максимальной длины; количество термов в логической функции управляемости; количество транзакций в

текущем терме функции; мощность рассматриваемой транзакции. Такие же обозначения используются и для наблюдаемости – $t^{y}_{max}, n^{y}_{t}, k^{y}_{i}, \left|t^{y}_{ij}\right|$.

Анализ тестопригодности графа управления. Учитывая, что автоматная модель программного продукта представлена взаимодействием операционного и управляющего автомата (рис. 5 слева), то наряду с моделированием транзакционного графа необходимо иметь возможность анализировать тестопригодность граф-схемы алгоритма управления (ГСА).

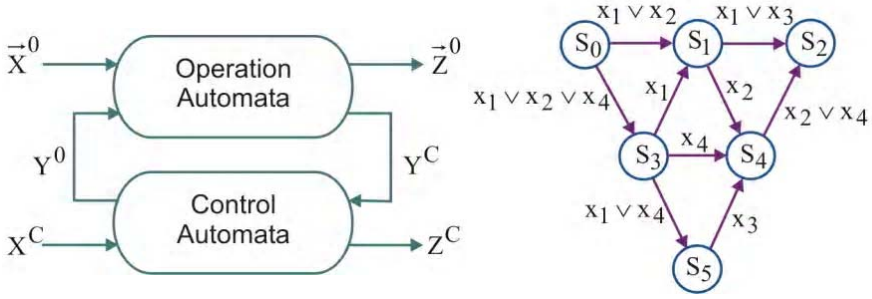

Рис. 5.Автоматная модель HDL-программы

Предлагается представить ГСА в виде содержательного графа управления (СГУ), который является подобным транзакционному графу. Здесь вершины есть операции программного кода, а дуги представляют условия перехода из одной вершины в другую для выполнения команды, обозначенной вершиной-стоком. Следовательно, для СГУ можно использовать процедуры, ранее разработанные для подсчета критериев тестопригодности транзакционного графа в части управляемости и наблюдаемости. Примером содержательного графа может служить рис. 5 (справа), имеющий 6 вершин и 9 дуг.

Подсчет управляемостей графа, реализующего алгоритм управления и представленного на рис. 5, имеет следующий вид:

$$S_3 = T_3^3;\ S_1 = T_3^3T_4^1 \vee T_1^2;\ \ S_2 = T_3^3T_4^1T_6^1T_7^2 \vee T_1^2T_2^2 \vee T_3^3T_5^1T_7^2 \vee T_3^3T_4^1T_7^2 \vee T_3^3T_8^2T_9^1T_7^2;$$

$$S_4 = T_3^3T_4^1T_6^1 \vee T_3^3T_5^1 \vee T_3^3T_8^2T_9^1;\ \ S_5 = T_3^3T_8^2.$$

Подсчет наблюдаемостей графа, представленного на рис. 5, содержит следующие выражения:

$$S_3 = T_7^2T_5^1 \vee T_7^2T_9^1T_8^2 \vee T_7^2T_6^1T_4^1 \vee T_2^2T_4^1;\ \ S_1 = T_2^2 \vee T_7^2T_6^1;$$

$$S_0 = T_2^2T_1^2 \vee T_7^2T_6^1T_4^1T_3^3 \vee T_7^2T_5^1T_3^3 \vee T_7^2T_5^1T_3^3 \vee T_7^2T_9^1T_8^2T_3^3 \vee T_2^2T_4^1T_3^3;\ S_4 = T_7^2;\ \ S_5 = T_7^2T_9^1.$$

Тестопригодность графа проекта и каждого компонента подсчитывается как произведение управляемости и наблюдаемости:

$$Q = \frac{1}{Z}(U \times N) = (\frac{1}{n}\sum_{i=1}^{n} U_i) \times (\frac{1}{n}\sum_{i=1}^{n} N_i). \qquad (6)$$

4. Диагностирование программных модулей WT SoC

Ниже представлены компоненты инфраструктуры для верификации и диагностического обслуживания двух модулей WT SoC: 1) транзакционные графы и построенные на их основе логические функции тестопригодности, управляемости и наблюдаемости HDL-моделей; 2) таблицы и графики оценивания тестопригодности всех вершин и всех графов программного HDL-кода; 3) таблицы неисправностей программных модулей WT SoC проекта для поиска дефектов на основе логических операций; 4) выполнены диагностические эксперименты, которые подтверждают эффективность и валидность предложенных моделей и метода поиска дефектов.

Системный граф совокупного программного продукта – HDL-кода, реализующего вейвлет-преобразование, представлен на рис. 6.

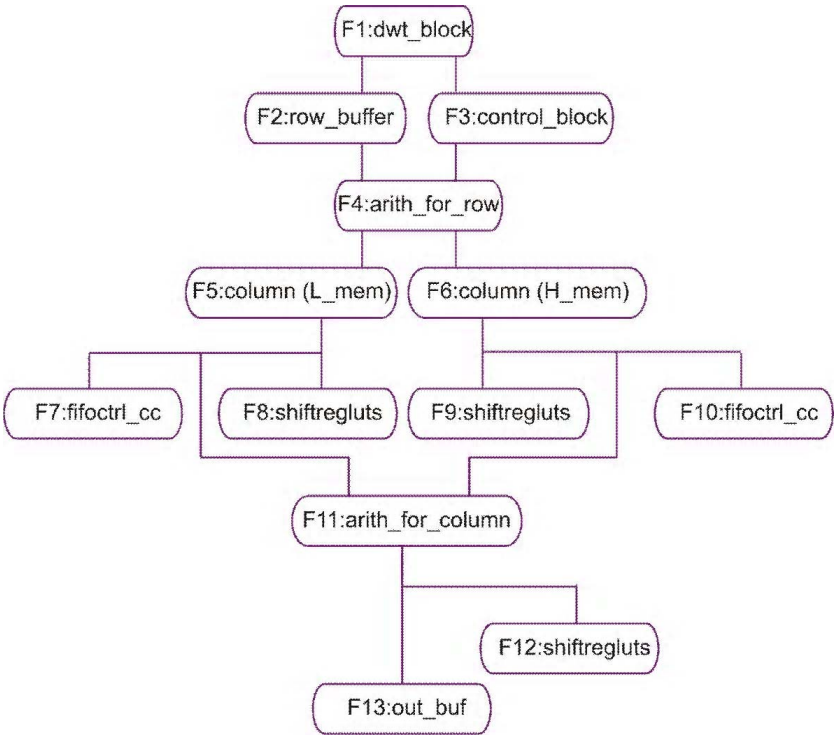

Рис. 6. Граф HDL-кода вейвлет-преобразования

Структурный анализа графа для определения тестопригодности дает возможность создать транзакционный граф, представленный на рис. 7.

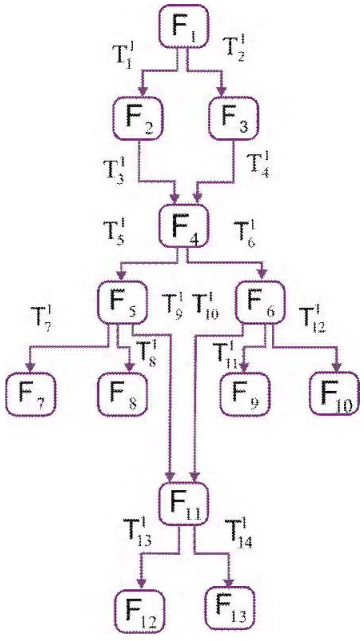

Рис. 7. Транзакционный граф main-HDL-кода

На основе транзакционного графа строятся логические функции управляемости и наблюдаемости, представленные ниже.

1) Управляемость входов:

$$F_2 = T_1^1;\ F_3 = T_2^1;\ F_4 = T_1T_3 \vee T_4T_2;\ F_5 = T_1^1T_5^1T_3^1 \vee T_5^1T_4^1T_2^1;\ F_6 = T_6^1T_1^1T_3^1 \vee T_6^1T_4^1T_5^1;$$

$F_7 = T_7^1T_1^1T_5^1T_3^1 \vee T_7^1T_5^1T_4^1T_2^1$; $F_8 = T_8^1T_1^1T_5^1T_3^1 \vee T_8^1T_5^1T_4^1T_2^1$; $F_9 = T_{11}^1T_6^1T_1^1T_3^1 \vee T_{11}^1T_6^1T_4^1T_5^1$;

$F_{10} = T_{12}^1T_6^1T_1^1T_3^1 \vee T_{12}^1T_6^1T_4^1T_5^1$; $F_{11} = T_{10}^1T_6^1T_1^1T_3^1 \vee T_{10}^1T_6^1T_4^1T_5^1 \vee T_9^1T_5^1T_1^1T_3^1 \vee T_9^1T_5^1T_4^1T_2^1$;

$F_{12} = T_{13}^1T_{10}^1T_6^1T_1^1T_3^1 \vee T_{13}^1T_{10}^1T_6^1T_4^1T_5^1T_{13}^1 \vee T_{13}^1T_9^1T_5^1T_1^1T_3^1 \vee T_{13}^1T_9^1T_5^1T_4^1T_2^1$;

$F_{13} = T_{14}^1T_{10}^1T_6^1T_1^1T_3^1 \vee T_{14}^1T_{10}^1T_6^1T_4^1T_5^1 \vee T_{14}^1T_9^1T_5^1T_1^1T_3^1 \vee T_{14}^1T_9^1T_5^1T_4^1T_2^1$.

2) Наблюдаемость выходов:

$F_{11} = T_{13}^1 \vee T_{14}^1$; $F_5 = T_7^1 \vee T_8^1 \vee T_9^1T_{13}^1 \vee T_9^1T_{14}^1$; $F_6 = T_{11}^1 \vee T_{12}^1 \vee T_{10}^1T_{13}^1 \vee T_{10}^1T_{14}^1$;

$F_4 = T_5^1T_7^1 \vee T_5^1T_8^1 \vee T_5^1T_9^1T_{13}^1 \vee T_5^1T_9^1T_{14}^1 \vee T_6^1T_{11}^1 \vee T_6^1T_{12}^1 \vee T_6^1T_{10}^1T_{13}^1 \vee T_6^1T_{10}^1T_{14}^1$;

$F_3 = T_4^1T_5^1T_7^1 \vee T_4^1T_5^1T_8^1 \vee T_4^1T_5^1T_9^1T_{13}^1 \vee T_4^1T_5^1T_9^1T_{14}^1 \vee T_4^1T_6^1T_{11}^1 \vee T_4^1T_6^1T_{12}^1 \vee T_4^1T_6^1T_{10}^1T_{13}^1 \vee T_4^1T_6^1T_{10}^1T_{14}^1$;

$F_2 = T_3^1T_5^1T_7^1 \vee T_3^1T_5^1T_8^1 \vee T_3^1T_5^1T_9^1T_{13}^1 \vee T_3^1T_5^1T_9^1T_{14}^1 \vee T_3^1T_6^1T_{11}^1 \vee T_3^1T_6^1T_{12}^1 \vee T_3^1T_6^1T_{10}^1T_{13}^1 \vee T_3^1T_6^1T_{10}^1T_{14}^1$;

$F_1 = T_2^1T_4^1T_5^1T_7^1 \vee T_2^1T_4^1T_5^1T_8^1 \vee T_2^1T_4^1T_5^1T_9^1T_{13}^1 \vee T_2^1T_4^1T_5^1T_9^1T_{14}^1 \vee T_2^1T_4^1T_6^1T_{11}^1 \vee T_2^1T_4^1T_6^1T_{12}^1 \vee T_2^1T_4^1T_6^1T_{10}^1T_{13}^1 \vee$
$\vee T_2^1T_4^1T_6^1T_{10}^1T_{14}^1 \vee T_1^1T_3^1T_5^1T_7^1 \vee T_1^1T_3^1T_5^1T_8^1 \vee T_1^1T_3^1T_5^1T_9^1T_{13}^1 \vee T_1^1T_3^1T_5^1T_9^1T_{14}^1 \vee T_1^1T_3^1T_6^1T_{11}^1 \vee T_1^1T_3^1T_6^1T_{12}^1 \vee$
$\vee T_1^1T_3^1T_6^1T_{10}^1T_{13}^1 \vee T_1^1T_3^1T_6^1T_{10}^1T_{14}^1$.

Логические функции управляемости и наблюдаемости обрабатываются на основе использования формулы (3), что дает возможность определить минимальные оценки вершин в части наблюдаемости, для которых строятся ассерции как дополнительные точки мониторинга программного кода. В табл. 1 такими критическими точками являются F_1, F_2, F_3.

В соответствии со значениями параметров тестопригодности на рис. 8 представлены графики управляемости, наблюдаемости всех вершин транзакционного графа главной программы, которая визуально иллюстрирует критические точки F_1, F_2, F_3 по наблюдаемости, интересные для установки в них ассерций.

Интегральные оценки качества тестопригодности, не учитывающие и учитывающие мультипликативность транзакционных дуг всего программного продукта, представлены в табл. 3. Общая оценка тестопригодностей двух видов графа (мульти- и одиночные дуги) имеет вид: $Q(F_0) = \{Q(F_0^S) = 0{,}33766; Q(F_0^M) = 0{,}61525\}$.

Данные оценки являются существенными при сравнении нескольких вариантов проекта, сделанных различными разработчиками. Естественно, что проект, имеющий большую интегральную оценку тестопригодности, принимается к реализации.

Таблица 1. Тестопригодность графа

C	U(S)	U(M)	N(S,N)	Q(S)	Q(M)
F1	1,00	1,00	0,3	0,3	0,65
F2	1,00	1,00	0,375	0,375	0,6875
F3	1,00	1,00	0,375	0,375	0,6875
F4	0,5	0,5	0,5	0,25	0,5
F5	0,33	0,33	0,75	0,2475	0,54
F6	0,33	0,33	0,75	0,2475	0,54
F7	0,25	0,25	1,00	0,25	0,625
F8	0,25	0,25	1,00	0,25	0,625
F9	0,25	0,25	1,00	0,25	0,625
F10	0,25	0,25	1,00	0,25	0,625
F11	0,25	0,25	1,00	0,25	0,625
F12	0,25	0,25	1,00	0,25	0,625
F13	0,25	0,25	1,00	0,25	0,625
Q=				0,15731	0,32538

Как результат моделирования неисправностей на тесте, который проверяет все ошибочные программные модули, строится табл. 2. Используя данную таблицу и формулы, определяющие логический метод диагностирования, можно определить дефектные компоненты программного кода с наперед заданной глубиной диагностирования.

Здесь векторы m1 и m2 определяют результат диагностического эксперимента, выполненного по схе-

ме рис. 4. Аналитическая запись процесса диагностирования основана на формулах (1) и имеет следующий вид:

$$A = m \wedge (A_1 \vee A_2 \vee ... \vee A_i \vee ... \vee A_m) \approx$$

$$F(m_1) = m_1 \wedge (F_1 \vee F_2 \vee F_3 \vee F_4 \vee F_5 \vee F_6 \vee F_7 \vee F_8 \vee F_9 \vee F_{10} \vee F_{11} \vee F_{12} \vee F_{13}) = F_{12};$$

$$F(m_2) = m_2 \wedge (F_1 \vee F_2 \vee F_3 \vee F_4 \vee F_5 \vee F_6 \vee F_7 \vee F_8 \vee F_9 \vee F_{10} \vee F_{11} \vee F_{12} \vee F_{13}) = F_1 \vee F_3 \vee F_4;$$

$$Q_1(m_1, F_{12}) = 1;\ Q_2[m_2, (F_1 \vee F_3 \vee F_4)] = 0{,}75;$$

Таблица 2. Неисправности, обнаруживаемые тестом

Test	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	m1	m2
t1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	X
t2	1	1	0	1	0	1	0	0	1	0	0	0	0	0	X
t3	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1
t4	1	0	1	1	0	1	0	0	0	0	1	0	1	0	1
t5	1	0	1	1	0	1	0	0	0	1	0	0	0	0	1

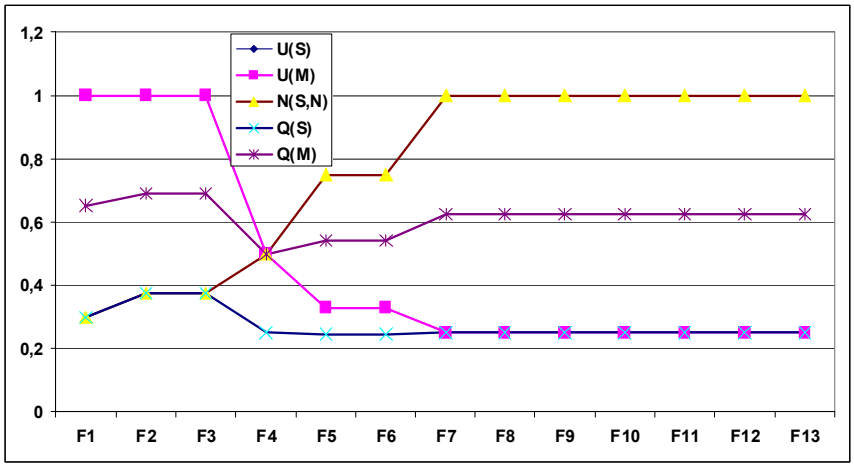

Рис. 8. Графики параметров тестопригодности

Здесь, в первом случае критерий качества, вычисляемый в соответствии с выражением (2), равен 1, что свидетельствует о присутствии данного дефекта в программном коде. Во втором случае, когда критерий не равен 1, можно говорить о возможном присутствии любого сочетания из трех неисправностей.

Далее представлено диагностическое обеспечение одного из программных модулей нижнего уровня, который является компонентом Row_buffer транзакционного графа main-программы (рис. 9). Для указанного компонента выполняются все вычислительные процедуры по аналогии с обработкой модуля main.

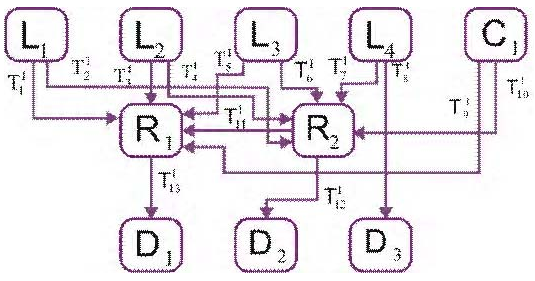

Рис. 9. Компонент Row_buffer транзакционного графа

На основе транзакционного графа строятся логические функции управляемости и наблюдаемости, представленные ниже.

Таблица 3. Тестопригодность структуры

C	Q(S)	Q(M)
F1	0,181	0,49009
F2	0,548	0,774
F3	0,20224	0,5294
F4	0,15663	0,44615
F5	0,25758	0,58531
F6	0,25758	0,58531
F7	0,2296	0,52823
F8	0,54	0,77
F9	0,54	0,77
F10	0,2296	0,52823
F11	0,15663	0,44615
F12	0,54	0,77
F13	0,55067	0,77533
Q(F0)=	0,33766	0,61525

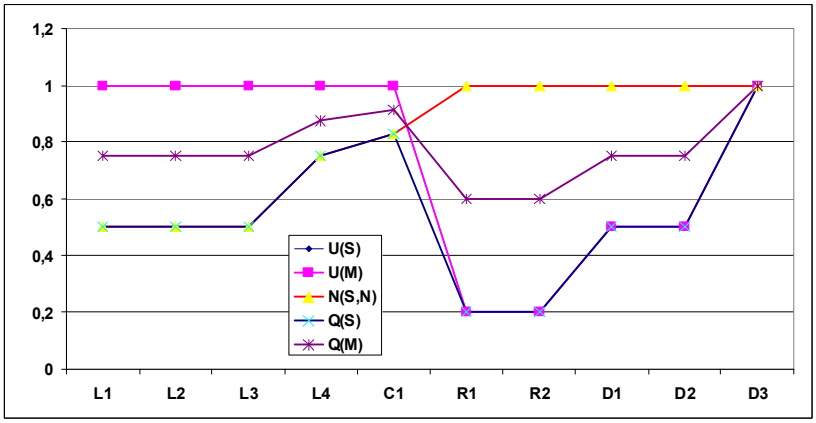

Рис. 10. Графики параметров тестопригодности Row_buffer

Как результат моделирования частей подпрограммы Row_buffer на тесте строится табл. 4 для неисправностей блоков.

Таблица 4. Неисправности компонента Row_buffer на тесте

Test	L1	L2	L3	L4	C1	R1	R2	D1	D2	m1	m2	m3
$A_1(t_1)$	1	0	0	0	0	1	0	1	0	1	X	1
$A_2(t_2)$	0	1	0	0	0	0	1	0	1	0	X	1
$A_3(t_3)$	0	0	1	0	0	1	1	1	0	1	0	0
$A_4(t_4)$	0	0	0	1	0	0	1	0	1	0	1	0
$A_5(t_5)$	0	0	0	0	1	1	1	1	0	1	0	0

Метод логического умножения вектор-столбцов. Он основан на применении операции логического умножения или конъюнкции вектора экспериментальной проверки на столбцы таблицы неисправностей $m_i \wedge (F_1 \vee F_2 \vee ... \vee F_j \vee ... \vee F_n)$ и подсчете качества взаимодействия векторов $Q_j(m_i \wedge F_j)$ в целях выбора лучшего из них. Предикатная запись процесса получения решения в виде совокупности ошибок, присутствующих в HDL-коде, представлена ниже:

$$P(m_i, F) = m_i \wedge (F_1 \vee F_2 \vee ... \vee F_j \vee ... \vee F_n) = \max Q_j(m_i \wedge F_j);$$

$$F^s = (F_j \in F) \leftarrow Q_j(m_i \wedge F_j) = \{1 \vee \max\},\ j = \overline{1,n};$$

$$F_i^m = (F_j \vee F_r) \leftarrow m_i \wedge (F_j \bigvee_{j=\overline{1,n-1}}^{r=\overline{j+1,n}} F_r) = \{1 \vee \max Q_{jr}[m_i \wedge (F_j \vee F_r)]\} \leftarrow$$

$$\leftarrow \forall j[Q_j(m_i \wedge F_j)] \neq 1;$$

$$F_i^m = \forall F_j \leftarrow (m_i \bigwedge_{j=1}^{n^*} F_j = F_j) \,\&\, (\bigvee_{j=1}^{n^*} F_j = m_i).$$

При этом столбец $F_j \in F$ фактически идентифицирует метрику поведения неисправности или дефектного блока на тестовых последовательностях. Достоинство метода – выбор всегда лучшего решения из всех возможных как для одиночных, так и для кратных дефектов. В последнем случае, если одиночный дефект не идентифицируется оценкой, равной 1, выполняется дизъюнкция таких вектор-строк (главное отличие метода от существующих технологий), которая сформирует оценку качества, равную 1 или максимально близкую к единице: $Q_{jr}(m_i \wedge (F_j \vee F_r))] = 1 \vee \max$. По существу, в список кратных дефектов включаются такие одиночные, которые при логическом умножении на вектор экспериментальной проверки дают результат в виде рассматриваемого вектор-столбца.

Используя таблицу и процедуры диагностирования (1), можно определить дефектные компоненты программного кода модуля Row_buffer методом логического умножения вектор-столбцов таблицы истинности на вектор экспериментальной проверки. Здесь векторы m1 и m2 формируют результат диагностического эксперимента, выполняемый по ранее определенной технологии (см. рис. 4). Результат диагностирования, с учетом (1), имеет следующий вид:

$$F(m_1) = m_1 \wedge (L_1 \vee L_2 \vee L_3 \vee L_4 \vee C_1 \vee R_1 \vee$$
$$\vee R_2 \vee D_1 \vee D_2) = R_1 \vee D_1;$$
$$F(m_2) = m_2 \wedge (L_1 \vee L_2 \vee L_3 \vee L_4 \vee C_1 \vee R_1 \vee$$
$$\vee R_2 \vee D_1 \vee D_2) = L_4 \vee D_2;$$
$$Q_1[m_1, (R_1 \vee D_1)] = 1;$$
$$Q_2[m_2, (L_4 \vee D_2)] = 0{,}75;$$

В случае неоднозначности (множественности) диагноза или определения места дефекта используется ассерция, которая осуществляет мониторинг блока, определенного в качестве дефектного. Для установления точного диагноза необходимо в худшем случае выполнить n-1 проверку, где n – мощность обнаруженных дефектных блоков, при условии, что в программном модуле существует один ошибочный блок.

Метод логического умножения вектор-строк. Другая стратегия определения ошибок программного кода по таблице неисправностей связана с анализом ее строк: 1) Вычисление произведения конъюнкции единичных строк ($A_i \leftarrow m_i = 1$) на отрицание дизъюнкции нулевых строк ($A_i \leftarrow m_i = 0$) для одиночных дефектных блоков. 2) Вычисление произведения дизъюнкции единичных строк ($A_i \leftarrow m_i = 1$) на отрицание дизъюнкции нулевых строк ($A_i \leftarrow m_i = 0$) для одиночных дефектных блоков:

$$\begin{cases} F^s = (\bigwedge\limits_{\forall m_i = 1} A_i) \wedge (\overline{\bigvee\limits_{\forall m_i = 0} A_i}); \\ F^m = (\bigvee\limits_{\forall m_i = 1} A_i) \wedge (\overline{\bigvee\limits_{\forall m_i = 0} A_i}); \end{cases} \qquad (7)$$
$$m = (m_1, m_2, ..., m_i, ..., m_n);$$
$$A = (A_1, A_2, ..., A_i, ..., A_n);$$

Выполнение диагностического эксперимента по (7) для вектора экспериментальной проверки $m_1 = (10101)$, заданного в последней таблице неисправностей, дает результат: $F^s(m_1, A) = R_1 \vee D_1$, который не хуже, чем ранее полученный методом логического умножения.

Диагностирование кратных дефектов методом логического умножения столбцов. Если, например, вектору $m_3 = (11000)$ экспериментальной проверки нет соответствующего столбца в табл. 4, то в данном случае вычисляются критерии качества (принадлежности) для всех столбцов таблицы неисправностей:

$$Q(m_3, A) = [Q(m_3, L_1) = 0{,}93;\ Q(m_3, L_2) = 0{,}93;\ Q(m_3, L_3) = 0{,}8;$$
$$Q(m_3, L_4) = 0{,}8;\ Q(m_3, C_1) = 0{,}8;\ Q(m_3, R_1) = 0{,}8;$$
$$Q(m_3, R_2) = 0{,}73;\ Q(m_3, D_1) = 0{,}8;\ Q(m_3, D_2) = 0{,}86].$$

Далее выбираются столбцы, имеющие максимальное значение критерия качества: $F^m = L_1 \vee L_2 \leftarrow [Q(m_3, L_1) = 0{,}93;\ Q(m_3, L_2) = 0{,}93]$. Дизъюнкция данных векторов дает вектор-столбец, равный вектору экспериментальной проверки: $L_1(10000) \vee L_2(01000) = m_3(11000)$. Вывод: в структуре присутствует кратный дефект, составленный из двух одиночных.

Диагностирование кратных дефектов методом логического умножения вектор-строк – сценарий F^m *(7)*. В соответствии с вектором $m_3 = (11000)$ экспериментальной проверки обрабатываются единичные и нулевые строки табл. 4, что приводит к следующему результату:

$$A(m_3^1, A) = A_1(100001010) \vee A_2(010000101) = (110001111);$$

$$A(m_3^0, A) = A_3(001001110) \vee A_4(000100101) \vee A_5(000011110) = (111111111);$$

$$F^m = A(m_3^1, A) \wedge \overline{A(m_3^0, A)} = (110001111) \wedge \overline{(111111111)} = (000000000).$$

В данном случае метод фиксирует отсутствие дефектов, когда на самом деле они однозначно имеются – вектор экспериментальной проверки имеет единичные координаты, что является недостатком метода логического умножения вектор-строк в части диагностирования кратных неисправностей. Однако метод логического умножения вектор-столбцов всегда дает результат, максимально приближенный к реальности.

Таким образом, предложенные аналитические модели и методы, проведенные эксперименты позволяют определять за два цикла логическую семантическую неисправность программного блока или группы операторов: в первом цикле вычисляется неисправный модуль (подпрограмма), во втором – ошибка в некоторой его части, мощность которой регулируется числом ассерционных операторов.

5. Выводы

1. Предложена инфраструктура верификации HDL-модели проекта, ориентированная на ускорение отладки программного кода на основе использования программной избыточности в виде механизма ассерций, который позволяет осуществлять поиск семантических ошибок с заданной глубиной диагностирования.

2. Предложен новый метод логического умножения строк и столбцов таблицы неисправностей для поиска одиночных и кратных дефектов на основе иерархической структуры таблиц дефектов, соответствующей программным модулям цифрового проекта путем нахождения оптимального решения, оцениваемого с помощью интегрального критерия качества, использующего функции принадлежности.

3. Получила дальнейшее развитие модель процесса определения тестопригодности проекта на основе использования графа транзакций, предназначенного для поиска критических точек программного кода для установки ассерционных операторов, повышающих управляемость и наблюдаемость структуры проекта.

4. Разработана инфраструктура верификации и диагностического обслуживания WT SoC: 1) Транзакционные графы и построенные на их основе логические функции тестопригодности, управляемости и наблюдаемости HDL-моделей. 2) Таблицы и графики оценивания тестопригодности всех вершин и всех графов программного HDL-кода. 3) Таблицы неисправностей всех программных модулей WT SoC проекта для поиска дефектов на основе логических операций. 4) Выполнены диагностические, в том числе и на реальных объектах, эксперименты, которые подтверждают эффективность и валидность предложенных моделей и метода поиска дефектов.

5. Предложены технологические мероприятия и рекомендации, ориентированные на улучшение процесса проектирования цифровых систем на кристаллах путем интеграции тестопригодного анализа и механизма ассерций в процедуры синтеза программных и аппаратных продуктов.

6. Показан пример практического использования анализа тестопригодности транзакционного и управляющего графов для реального DSP-проекта и последующего внедрения в критические точки программного кода механизма ассерций в целях поиска и устранения ошибок, что дало возможность найти и исправить некорректные фрагменты кода, необнаруженные авторами программы.

7. Индустриальная значимость предложенных моделей и метода заключается в высокой рыночной привлекательности и заинтересованности технологических компаний в новых решениях верификации программно-аппаратных изделий на системном уровне проектирования для уменьшения time-to-market и повышения выхода годной продукции – yield.

Список литературы: 1. *Daubechies I.* Factoring wavelet transforms into lifting steps / Daubechies I. and Sweldens W. Bell Laboratories. Lucent Technologies, 1996. 368p. **2.** *Daubechies I. and Sweldens W.* Factoring wavelet transforms into lifting schemes / I. Daubechies and W. Sweldens // The J. of Fourier Analysis and Applications. 1998. Vol. 4. P. 247-269. **3.** *Chui C.K.* An Introduction to Wavelets / C.K. Chui. New York – London: Academic Press, 1992. 266 p. **4.** *Taubman David S.* JPEG2000: image compression fundamentals, standards and practice / David S. Taubman, Michael W. Marcellin. Norwell: Kluwer Academic Pulishers, 2002. 774 p. **5.** *Петухов А.П.* Введение в теорию базисов всплесков / Петухов А.П. СПб.: Изд. СПбГТУ, 1999. 131с. **6.** *DeVore R.* Image Compression Trough Wavelet Transform Coding / R. DeVore, W. Jawerth, B. Lucier // IEEE Trans. on Information Theory. 1992. Vol. 39, No. 2. P. 719-746. **7.** *Bergeron J.* Writing Testbenches Using SystemVerilog / J. Bergeron // Springer Science and Business Media, Inc., 2006. 414 p. **8.** Bergeron J. Verification Methodology Manual for System Verilog /J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale. Springer Science and Business Media, 100 Inc., 2006. 510 p. **9.** *Abramovici M.* Digital System Testing and Testable Design / M. Abramovici, M.A. Breuer and A.D. Friedman. Comp. Sc. Press. 1998. 652 p. **10.** *Bayraktaroglu* Ismet The Construction of Optimal Deterministic Partitionings in Scan–Based BIST Fault Diagnosis: Mathematical Foundations and Cost–Effective Implementations / Ismet Bayraktaroglu, Alex Orailoglu // IEEE Transactions on Computers. 2005. P.61–75. **11.** Densmore Douglas A Platform–Based taxonomy for ESL Design / Douglas Densmore, Roberto Passerone, Alberto Sangiovanni–Vincentelli // Design&Test of computers. 2006. P. 359–373. **12.** *DaSilva* Francisco Overview of the IEEE P1500 Standard / Francisco DaSilva, Yervant Zorian, Lee Whetsel, Karim Arabi, Rohit Kapur // ITC International Test Conference. 2003. P. 988–997. **13.** *Rashinkar P.* System–on–chip Verification: Methodology and Techniques / Rashinkar P., Paterson P., Singh L. Kluwer Academic Publishers, 2002. 324 p. **14.** *Zorian Yervant.* What is Infrustructure IP? / Yervant Zorian. // IEEE Design & Test of Computers. 2002. P. 5–7. **15.** *Zorian Yervant* Guest editors' introduction: Design for Yield and reliability / Yervant Zorian, Dmytris Gizopoulos // IEEE Design & Test of Computers. 2004. P. 177–182. **16.** *Zorian Yervant.* Guest Editor's Introduction: Advances in Infrastructure IP // IEEE Design and Test of Computers. 2003. Vol. 20, № 3. P. 49. **17.** *Thatte S.M.* Test generation for microprocessors / S.M. Thatte, J.A. Abraham // IEEE Trans. Comput. 1980. Vol. 29, No 6. P. 429–441. **18.** *Шаршунов С.Г.* Построение тестов микропроцессоров. 1. Общая модель. Проверка обработки данных // Автоматика и телемеханика. 1985. №11. С. 145–155. **19.** *Jerraya A.A.* System Level Synthesis SLS / A.A. Jerraya // TIMA Laboratory. Annual Report, 2002. P. 65–75. **20.** *Ghenassia Frank.* Transaction Level Modeling with SystemC / Frank Ghenassia. TLM Concepts and Applications for Embedded Systems. Published by Springer, 2005. 282 p. **21.** *Foster* Harry Assertion–based design / Harry Foster, Adam Krolnik, David Lacey. Kluwer Academic Publishers. Second edition. Springer, 2005. 392 p. **22.** *Meyer A.S.* Principles of Functional Verification / Meyer A.S. Elsevier Science, 2004. 206 p. **23.** *Хаханов В.И.* Проектирование и тестирование цифровых систем на кристаллах / В.И. Хаханов, Е.И. Литвинова, О.А. Гузь. Харьков: ХНУРЭ, 2009. 484с.

Поступила в редколлегию 02.09.2009

Хаханов Владимир Иванович, декан факультета КИУ ХНУРЭ, д-р техн. наук, профессор кафедры АПВТ 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.

Побеженко Ирина Александровна, аспирантка кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-421. E-mail: kiu@kture.kharkov.ua.

Василенко Василина Александровна, аспирантка кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем и сетей. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-421. E-mail: kiu@kture.kharkov.ua.

Чумаченко Светлана Викторовна, д-р техн. наук, профессор кафедры АПВТ ХНУРЭ. Научные интересы: математическое моделирование и вычислительные методы, методы дискретной оптимизации. Увлечения: спорт, музыка, поэзия, путешествия. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326, e-mail: ri@kture.kharkov.ua.