УДК 658.512.011:681.326:519.713

НГЕНЕ КРИСТОФЕР УМЕРАХ (NGENE CHRISTOPHER UMERAH),
В.И. ХАХАНОВ, С.А. ЗАЙЧЕНКО, Е.И. ЛИТВИНОВА, О.Б. СКВОРЦОВА

МОДЕЛИ И МЕТОДЫ ВЕРИФИКАЦИИ И ДИАГНОСТИРОВАНИЯ SOC HDL-КОДА

Предлагается xor-метрика отношений объектов в векторном логическом пространстве и основанная на ней структурно-аналитическая модель тестирования цифровых систем на кристаллах. Описываются ассерционно-ориентированные модели и методы верификации и диагностирования функциональных нарушений HDL-кода, которые дают возможность существенно уменьшить время проектирования программных и аппаратных продуктов. Показывается архитектурная модель мультиматричного процессора с ограниченной системой логических команд для решения задач встроенного диагностирования.

Актуальность. Тенденция последних лет в части создания новых коммуникационных, вычислительных и информационных сервисов, полезных для человека, обращает внимание на создание все более специализированных гаджетов (gadget), обладающих существенными преимуществами перед персональными компьютерами и ноутбуками: энергопотребление, компактность, вес, стоимость, функциональные возможности, дружественность интерфейса. Практически вся десятка лучших за 2010 год специализированных изделий (Apple iPad, Samsung Galaxy S, Apple MacBook Air, Logitech Revue, Google Nexus One (HTC Desire), Apple iPhone 4, Apple TV, Toshiba Libretto W100, Microsoft Kinect, Nook Color) реализована в виде цифровых систем на кристаллах. К 2012 году рынок мобильной и беспроводной связи перейдет на 20 нм (итоги январского Саммита-2011 альянса Common Platform). Дальнейшее развитие технологий по годам: 2014 – 14 нм, 2016 – 11 нм. В 2015 году 55% сотовых телефонов станут смартфонами, планшетные компьютеры заменят ноутбуки и нетбуки. Суперфоны (Nexus-1, Google) станут той соединительной тканью, которая свяжет все остальные устройства и сервисы. Переход от вычислительных платформ к мобильным устройствам с малым форм-фактором приведет к существенному снижению энергопотребления во всем мире. Надвигается следующая волна компьютеризации под названием «интернет вещей», которая приведет к широкому распространению датчиковых сетей, включая их интеграцию в человеческое тело. Мировой рынок перечисленных выше устройств и гаджетов насчитывает сегодня порядка 3 миллиардов изделий. Для их эффективного проектирования, производства и эксплуатации создаются новые технологии и инфраструктуры сервисного обслуживания на стадиях проектирования, производства и эксплуатации. Один из возможных шагов в данном направлении представлен ниже в виде структуры публикации как технологии верификации T^v: M^t – метрика и модель процессов тестирования, H^c – HDL-код проекта, G^t – синтез графа транзакций программных блоков,

$\{M^f, M^s\}$ – построение двух моделей верификации HDL-кода (таблица функциональных нарушений и матрица активизации программных блоков), $\{D^c, D^r, D^m\}$ – разработка трех методов (анализа строк, столбцов таблицы и матрицы в целом) диагностирования функциональных нарушений, использующих механизм ассерций (ассерция – логическое высказывание, определяющее семантические ошибки программного кода), P^m – создание архитектуры мультиматричного процессора для параллельного анализа табличных данных, R – имплементация моделей методов и средств в систему Riviera компании Aldec:

$$T^v = M^t \to H^c \to G^t \to \begin{Bmatrix} M^f \to \begin{Bmatrix} D^c \\ D^r \end{Bmatrix} \\ M^s \to D^m \end{Bmatrix} \to P^m \to R.$$

Цель – существенное уменьшение времени проектирования и повышение качества цифровых систем на кристаллах за счет разработки ассерционно-ориентированной инфраструктуры, моделей и методов верификации и диагностирования HDL-кода. Информация, необходимая для поиска блоков с функциональными нарушениями, определяется в процессе моделирования(выполнения) программного кода. Эффективность проектирования цифрового изделия определяется средним и нормированным в интервале [0,1] интегральным критерием:

$$E = F(L, T, H) = \min[\frac{1}{3}(L + T + H)],\ Y = (1 - P)^n;\ L = 1 - Y^{(1-k)} = 1 - (1 - P)^{n(1-k)};$$

$$T = [(1 - k) \times H^s]/(H^s + H^a);\ H = H^a/(H^s + H^a).$$

Критерий учитывает: уровень ошибок проекта L, время верификации T, программно-аппаратную избыточность, определяемую механизмами ассерций и средствами сервисного обслуживания H. Параметр L, как дополнение к параметру Y, характеризующему выход годной продукции, зависит от тестопригодности проекта k, вероятности P существования неисправных компонентов и числа необнаруженных ошибок n. Время верификации определяется тестопригодностью проекта k [3,4], умноженной на структурную сложность аппаратно-программной функциональности, отнесенную к общей сложности проекта в строках кода. Программно-аппаратная избыточность находится в функциональной зависимости от сложности ассерционного кода и других затрат, отнесенных к общей сложности проекта. При этом программная или аппаратная, избыточность должна обеспечивать заданную глубину диагностирования ошибок функциональности и время выхода изделия на рынок, определенное заказчиком.

Задачи: 1) Создание метрики и структурно-аналитической модели тестирования цифровых систем на кристаллах. 2) Усовершенствование моделей и методов поиска функциональных нарушений на основе механизма ассерций для повышения быстродействия верификации и диагностирования HDL-кода. 3) Разработка архитектурной модели мультиматричного процессора для решения задач диагностирования.

Источники: 1. Модели формулирования и классификации задач технической диагностики [1-6]. 2. Диагностирование и верификация цифровых систем на кристаллах [9-17, 22-15]. 3. Аппаратура и матричные процессоры для ускорения процессов тестирования [18-21].

1. Модель процессов тестирования и верификации

Предлагаются технологичные и эффективные процесс-модели и методы диагностирования функциональных нарушений в программных и/или аппаратных продуктах. Используются регистровые или матричные (табличные) структуры данных, которые ориентированы на параллельное выполнение логических операций при поиске дефектных компонентов.

Проблема синтеза или анализа компонентов произвольной системы может быть сформулирована в виде взаимодействия (симметрической разности – аналог xor-операции на булеане) в кибернетическом пространстве ее модели F с входными воздействиями T и реакциями L:

$$f(F, T, L) = \varnothing \to F \Delta T \Delta L = \varnothing\ .$$

Киберпространство – совокупность взаимодействующих по метрике информационных процессов и явлений, использующих в качестве носителя компьютерные системы и сети. В частности, компонент пространства представлен k-мерным (кортежем) вектором $a=(a_1,a_2,...,a_j,...,a_k)$, $a_j=\{0,1\}$ в двоичном алфавите. Нуль-вектор есть k-мерный кортеж, все координаты которого равны нулю: $a_j=0,\ j=\overline{1,k}$.

Метрика β кибернетического (двоичного) пространства определяется единственным равенством, которое формирует нуль-вектор для xor-суммы расстояний d_i между ненулевым и конечным числом точек (объектов), замкнутых в цикл:

$$\beta=\bigoplus_{i=1}^{n} d_i=0.$$

Расстояние (по Хэммингу) между двумя объектами (векторами) a и b определяется в виде производного вектора: $d_i=d(a,b)=a_j\bigoplus_{j=1}^{k} b_j$. Иначе: метрика β векторного логического двоичного пространства есть равная нуль-вектору xor-сумма расстояний между конечным числом точек (вершин) графа, образующих цикл. Сумма n-мерных двоичных векторов, задающих координаты точек цикла, равна нуль-вектору. Данное определение метрики оперирует отношениями, что позволяет сократить систему аксиом с трех до одной и распространить ее действие на любые конструкции n-мерного киберпространства. Классическое задание метрики для определения взаимодействия одной, двух и трех точек в векторном логическом пространстве является частным случаем β-метрики при $i=1,2,3$ соответственно:

$$M=\begin{cases} d_1=0\leftrightarrow a=b; \\ d_1\oplus d_2=0\leftrightarrow d(a,b)=d(b,a); \\ d_1\oplus d_2\oplus d_3=0\leftrightarrow d(a,b)\oplus d(b,c)=d(a,c). \end{cases}$$

Метрика β кибернетического многозначного пространства, где каждая координата вектора (объекта) определена в алфавите, составляющем булеан на универсуме примитивов мощностью p: $a_j=\{\alpha_1,\alpha_2,...,\alpha_r,...,\alpha_m\},\ m=2^p$, есть равная $\varnothing$-вектору (по всем координатам) симметрическая разность расстояний между конечным числом точек, образующих цикл:

$$\beta=\bigtriangleup_{i=1}^{n} d_i=\varnothing. \qquad (1)$$

Равенство пустому вектору симметрической разности покоординатного теоретико-множественного взаимодействия (1) подчеркивает равнозначность компонентов (расстояний), формирующих уравнения, где единственная координатная операция $d_{i,j}\Delta d_{i+1,j}$, используемая, например, в четырехзначной модели Кантора, определяется соответствующей Δ-таблицей:

Δ	0	1	x	$\varnothing$
0	$\varnothing$	x	1	0
1	x	$\varnothing$	0	1
x	1	0	$\varnothing$	x
$\varnothing$	0	1	x	$\varnothing$

$\cap$	0	1	x	$\varnothing$
0	0	$\varnothing$	0	$\varnothing$
1	$\varnothing$	1	1	$\varnothing$
x	0	1	x	$\varnothing$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

$\cup$	0	1	x	$\varnothing$
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
$\varnothing$	0	1	x	$\varnothing$

a	0	1	x	$\varnothing$
$\tilde{a}$	1	0	$\varnothing$	x

(2)

Здесь также приведены таблицы истинности для других базовых теоретико-множественных операций, далее используемых по тексту. Число примитивных символов, образующих замкнутый относительно теоретико-множественных координатных операций алфавит, может быть увеличено. При этом мощность алфавита (булеана) определяется выражением $m=2^p$, где p – число примитивных символов. Введенная метрика представляет

не только теоретический интерес, но имеет и практическую направленность на обобщение и классификацию задач технической диагностики путем создания модели xor-отношений на множестве из четырех основных компонентов. Процедуры синтеза тестов, моделирования неисправностей и поиска дефектов можно свести к xor-отношениям на графе (рис. 1) полного взаимодействия четырех вершин (функциональность, устройство, тест, дефекты) $G = \{F, U, T, L\}$.

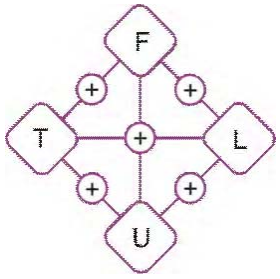


Рис. 1. Граф взаимодействия компонентов технической диагностики

Такой граф порождает четыре базовых треугольника, которые формируют 12 практически полезных триад отношений, формулирующих задачи технической диагностики:

$T \oplus F \oplus L = 0$	$T \oplus L \oplus U = 0$	$T \oplus F \oplus U = 0$	$F \oplus L \oplus U = 0$
1) $T = F \oplus L$	4) $T = L \oplus U$	7) $T = F \oplus U$	10) $F = L \oplus U$
2) $F = T \oplus L$	5) $L = T \oplus U$	8) $F = T \oplus U$	11) $L = F \oplus U$
3) $L = T \oplus F$	6) $U = T \oplus L$	9) $U = T \oplus F$	12) $U = F \oplus L$

Введение вершины U в граф взаимодействия компонентов технической диагностики расширяет функциональные возможности модели, появляются новые свойства полученной системы. Введение в структуру новой вершины должно иметь весомые аргументы в пользу ее целесообразности. Что касается представленного на рис. 1 графа, содержательно все формульно описанные задачи можно классифицировать в группы следующим образом.

Группа 1 – теоретические эксперименты (на модели функциональности), без устройства: 1) Синтез теста по модели функциональности для заданного списка неисправностей. 2) Построение модели функциональности на основе заданного теста и списка неисправностей. 3) Моделирование неисправностей функциональности на заданном тесте.

Группа 2 – реальные эксперименты (на устройстве), без модели функциональности:4) Синтез теста путем физической эмуляции дефектов в устройстве. 5) Определение списка неисправностей устройства при выполнении диагностического эксперимента. 6) Верификация теста и дефектов в эксперименте на реальном устройстве.

Группа 3 – тестовые эксперименты (верификация), без дефектов: 7) Синтез теста путем сравнения результатов моделирования модели и реального устройства. 8) Синтез функциональности по реальному устройству и заданному тесту. 9) Верификация теста и модели функциональности относительно реального устройства с существующими неисправностями.

Группа 4 – эксперименты в процессе функционирования, на рабочих воздействиях: 10) Проверка правильности поведения реального устройства на существующих или заданных дефектах. 11) Проверка работоспособности устройства относительно существующей модели в процессе функционирования. 12) Верификация функциональности и списка дефектов относительно поведения реального устройства.

Наиболее популярными задачами из перечисленного выше списка являются: 1, 3, 5, 8, 9. Можно ввести и другую классификацию типов задач, которая дает возможность определить на графе $G = (F, U, T, L)$ все концептуальные пути решения целевых проблем: синтеза тестов, определения модели функциональности, генерирования модели дефектов и проектирования устройства:

$$1)\ T = F \oplus L;\quad 4)\ F = T \oplus L;\quad 7)\ L = T \oplus F;\quad 10)\ U = T \oplus L;$$
$$2)\ T = U \oplus L;\quad 5)\ F = U \oplus L;\quad 8)\ L = T \oplus U;\quad 11)\ U = T \oplus F;$$
$$3)\ T = F \oplus U;\quad 6)\ F = T \oplus U;\quad 9)\ L = F \oplus U;\quad 12)\ U = F \oplus L.$$

Все конструкции, представленные в отношениях, обладают замечательным свойством обратимости. Компонент, вычисляемый с помощью двух других, может быть использован в качестве аргумента для определения любого из двух исходных. Поэтому здесь можно говорить о транзитивной обратимости каждой триады отношений на полном графе, когда по двум любым компонентам всегда и однозначно можно восстановить или определить третий. При этом формат представления каждого компонента должен быть одинаковым по форме и размерности (векторы, матрицы). На основе предложенной метрики и моделей тестирования далее рассмотрены более подробно методы диагностирования дефектов или функциональных нарушений.

2. Модель поиска функциональных нарушений в программных блоках

Используется уравнение пространства $f(F,T,L,U)=0 \rightarrow F \oplus T \oplus L \oplus U = 0$, которое трансформируется к виду $L = (T \oplus F) \oplus (T \oplus U)$. Диагностирование дефектов (функциональных нарушений) сводится к сравнению результатов модельного $(T \oplus F)$ и натурного $(T \oplus U)$ экспериментов, которое формирует список функциональных нарушений L, присутствующих в объекте диагностирования. Формула-модель процесса поиска блока F_i с функциональными нарушениями сводится к выбору решения посредством определения хог-взаимодействия между тремя компонентами:

$$L = F_i \leftarrow [(T \oplus F_i) \overset{p}{\underset{i=1}{\oplus}} (T \oplus U_i)] = 0.$$

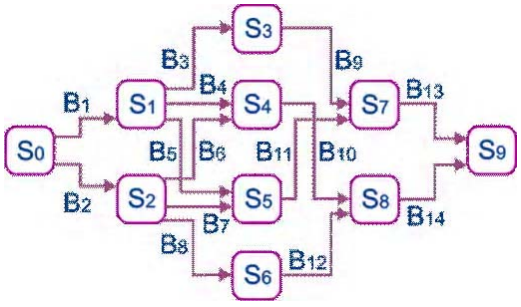
Аналитическая модель верификации HDL-кода с использованием механизма темпоральных ассерций (дополнительных линий наблюдения) ориентирована на достижение заданной глубины диагностирования и представлена в следующем виде:

$$\begin{array}{ll} M = f(F,A,B,S,T,L), & F = (A * B) \times S;\ S = f(T,B); \\ A = \{A_1, A_2, ..., A_i, ..., A_n\}; & B = \{B_1, B_2, ..., B_i, ..., B_n\}; \\ S = \{S_1, S_2, ..., S_i, ..., S_m\}; & S_i = \{S_{i1}, S_{i2}, ..., S_{ij}, ..., S_{ip}\}; \\ T = \{T_1, T_2, ..., T_i, ..., T_k\}; & L = \{L_1, L_2, ..., L_i, ..., L_n\}. \end{array} \quad (3)$$

Здесь $F = (A * B) \times S$ – функциональность, представленная графом (рис. 2) транзакций программных блоков (Code-Flow Transaction Graph – CFTG), где $S = \{S_1, S_2, ..., S_i, ..., S_m\}$ – вершины или состояния программного продукта при моделировании тестовых сегментов. Иначе граф можно идентифицировать как ABC-граф – Assertion Based Coverage Graph. Каждое состояние $S_i = \{S_{i1}, S_{i2}, ..., S_{ij}, ..., S_{ip}\}$ определяется значениями существенных переменных проекта (булевы, регистровые переменные, память). Ориентированные дуги графа представлены совокупностью программных блоков

$$B = (B_1, B_2, ..., B_i, ..., B_n),\ \overset{n}{\underset{i=1}{\cup}} B_i = B;\ \overset{n}{\underset{i=1}{\cap}} B_i = \varnothing,$$

где каждому из них может быть поставлена в соответствие ассерция $A_i \in A = \{A_1, A_2, ..., A_i, ..., A_n\}$.



$$B = (B_1B_3B_9 \vee (B_2B_7 \vee B_1B_5)B_{11})B_{13} \vee ((B_1B_4 \vee B_2B_6)B_{10} \vee B_2B_8B_{12})B_{14} =$$
$$= B_1B_3B_9B_{13} \vee B_2B_7B_{11}B_{13} \vee B_1B_5B_{11}B_{13} \vee B_1B_4B_{10}B_{14} \vee B_2B_6B_{10}B_{14} \vee B_2B_8B_{12}B_{14}.$$

Рис. 2. Пример ABC-графа для HDL-кода

Каждая дуга B_i – группа операторов кода – формирует состояние вершины $S_i = f(T, B_i)$ в зависимости от теста $T = \{T_1, T_2, ..., T_i, ..., T_k\}$. Каждой вершине может быть поставлен ассерционный монитор, объединяющий ассерции входящих в вершину дуг $A(S_i) = A_{i1} \vee A_{i2} \vee ... \vee A_{ij} \vee ... \vee A_{in}$. Вершина может иметь более одной входящей (исходящей) дуги. Множество блоков с функциональными нарушениями дано списком $L = \{L_1, L_2, ..., L_i, ..., L_n\}$.

Модель HDL-кода, представленная в форме ABC-графа, отображает не только структуру программного кода, но и тестовые срезы функциональных покрытий, формируемые с помощью программных блоков, входящих в рассматриваемую вершину. Последняя определяет отношение между достигнутым на тесте пространством переменных и потенциально возможным, которое формирует функциональное покрытие как мощность состояния i-вершины графа $Q = \mathrm{card}C_i^r / \mathrm{card}C_i^p$. В совокупности все вершины графа должны составлять полное покрытие пространства состояний переменных программного кода, которое формирует качество теста, равное 1 (100%): $Q = \mathrm{card}\bigcup_{i=1}^{m} C_i^r / \mathrm{card}\bigcup_{i=1}^{m} C_i^p = 1$. Кроме того, механизм ассерций $< A, C >$, существующий на графе, позволяет выполнять мониторинг дуг (code-coverage) $A = \{A_1, A_2, ..., A_i, ..., A_n\}$ и вершин (functional coverage) $C = \{C_1, C_2, ..., C_i, ..., C_m\}$. Ассерции на дугах отвечают за диагностирование функциональных нарушений в программных блоках. Ассерции на вершинах графа дают дополнительную информацию о качестве тестов (ассерций) в целях их улучшения или дополнения. Транзакционный граф программного кода дает возможность: 1) использовать аппарат тестопригодного проектирования для оценки качества программного продукта; 2) оценить затраты на создание тестов, диагностирование и исправление функциональных нарушений; 3) оптимизировать синтез теста путем решения задачи покрытия минимальным множеством активизированных путей всех дуг (вершин). Например, минимальный тест для приведенного ABC-графа имеет шесть сегментов, которые активизируют все существующие пути:

$$T = S_0S_1S_3S_7S_9 \vee S_0S_1S_4S_8S_9 \vee S_0S_1S_5S_7S_9 \vee S_0S_2S_4S_8S_9 \vee S_0S_2S_5S_7S_9 \vee S_0S_2S_6S_8S_9.$$

Тесту можно поставить в соответствие следующую матрицу активизации программных блоков:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}
T_1	1	.	1	.	.	.	.	.	1	.	.	.	1	.
T_2	1	.	.	1	.	.	.	.	.	1	.	.	.	1
T_3	1	.	.	.	1	.	.	.	.	.	1	.	1	.
T_4	.	1	.	.	.	1	.	.	.	1	.	.	.	1
T_5	.	1	.	.	.	.	1	.	.	.	1	.	1	.
T_6	.	1	.	.	.	.	.	1	.	.	.	1	.	1

Матрица активизации иллюстрирует факт неразличимости на тесте функциональных нарушений в блоках 3 и 9, 8 и 12, которые составляют два класса эквивалентностей при наличии одной ассерции (монитора) в вершине 9. Для устранения такой неразличимости необходимо создать два дополнительных монитора в вершинах 3 и 6. В результате, 3 ассерции на вершинах $A = (A_3, A_6, A_9)$ дают возможность различить между собой все блоки программного кода. Таким образом, граф позволяет не только синтезировать оптимальный тест, но и определять минимальное число ассерционных мониторов здесь и далее в вершинах для поиска неисправных блоков с заданной глубиной диагностирования.

Увеличение числа ассерционных мониторов приводит к модификации таблицы активизации. Иначе, на заданном тесте и механизме ассерций необходимо однозначно решать задачу диагностирования функциональных нарушений программного кода с глубиной до программного блока. При этом число ассерций и тестовых сегментов должно быть мини-

мально допустимым для кодовой идентификации всех блоков: $|T|+|A| \geq \log_2|B| = \text{card}T + \text{card}A \geq \log_2 \text{card}B$. Первоначально количество мониторов-ассерций равно числу тестовых сегментов. Таблица активизации программных модулей позволяет выполнять идентификацию блоков кода с функциональными нарушениями на обобщенном векторе экспериментальной проверки (ассерционного мониторинга) $V = (V_1, V_2, ..., V_i, ..., V_n), V_i = \{0,1\}, V_i = T_i \oplus B_j, \forall j(B_{ij} = 1)$. Координата вектора $V_i = T_i \oplus B_j = 1$ идентифицирует факт «падения» тест-сегмента на подмножестве активизированных им блоков. В соответствии с вектором V, заданным на таблице активизации, с учетом приведенного выше правила вычисления его координат:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	V
T_1	1	.	1	.	.	.	.	.	1	.	.	.	1	.	0
T_2	1	.	.	1	.	.	.	.	.	1	.	.	.	1	1
T_3	1	.	.	.	1	.	.	.	.	.	1	.	1	.	0
T_4	.	1	.	.	.	1	.	.	.	1	.	.	.	1	1
T_5	.	1	.	.	.	.	1	.	.	.	1	.	1	.	0
T_6	.	1	.	.	.	.	.	1	.	.	.	1	.	1	1

можно построить логическую функцию функциональных нарушений программного продукта, которая упрощается на основе использования координат вектора экспериментальной проверки V:

$$B = (\overline{T}_1 \vee B_1 \vee B_3 \vee B_9 \vee B_{13}) \wedge (\overline{T}_2 \vee B_1 \vee B_4 \vee B_{10} \vee B_{14}) \wedge$$
$$\wedge (\overline{T}_3 \vee B_1 \vee B_5 \vee B_{11} \vee B_{13}) \wedge (\overline{T}_4 \vee B_2 \vee B_6 \vee B_{10} \vee B_{14}) \wedge$$
$$\wedge (\overline{T}_5 \vee B_2 \vee B_7 \vee B_{11} \vee B_{13}) \wedge (\overline{T}_6 \vee B_2 \vee B_8 \vee B_{12} \vee B_{14});$$
$$\{V, T\} = (010101) \rightarrow$$
$$B = (0 \vee B_1 \vee B_4 \vee B_{10} \vee B_{14}) \wedge (0 \vee B_2 \vee B_6 \vee B_{10} \vee B_{14}) \wedge (0 \vee B_2 \vee B_8 \vee B_{12} \vee B_{14}) =$$
$$= (B_1 \vee B_4 \vee B_{10} \vee B_{14}) \wedge (B_2 \vee B_6 \vee B_{10} \vee B_{14}) \wedge (B_2 \vee B_8 \vee B_{12} \vee B_{14}) =$$
$$= B_1B_2 \vee B_4B_2 \vee ... \vee B_3B_6B_{12} \vee ... \vee B_{14}.$$

После преобразования конъюнктивной нормальной формы (КНФ) к дизъюнктивной нормальной форме (ДНФ) полученные термы содержат все возможные решения в виде покрытия единичных координат вектора экспериментальной проверки одиночными или кратными функциональными нарушениями программных блоков. Выбор лучшего решения связан с определением терма ДНФ минимальной длины. В данном примере оптимальным решением является терм, содержащий один блок $B = B_{14}$, который своим функциональным нарушением покрывает три единицы в векторе экспериментальной проверки $V = (010101)$. Данный факт также очевиден из сравнения двух последних столбцов матрицы активизации B.

Другое аппаратно ориентированное аналитическое решение связано с синтезом многовыходовой комбинационной схемы – дешифратора по матрице активизации программных блоков:

$$B_1 = T_1T_2T_3\overline{T}_4\overline{T}_5\overline{T}_6; B_2 = \overline{T}_1\overline{T}_2\overline{T}_3T_4T_5T_6; B_3 = \overline{T}_1T_2T_3T_4T_5T_6; \ ... \ B_{14} = \overline{T}_1T_2\overline{T}_3T_4\overline{T}_5T_6.$$

Такое устройство имеет число входов, равное количеству тестовых сегментов, а выходных линий, – равное числу программных блоков. При подаче на входы дешифратора вектора экспериментальной проверки формируется единичное значение на одном из его выходов. При этом номер выхода соответствует блоку, имеющему функциональные нарушения. Такая схема представляет интерес для создания замкнутой в цикл инфраструктуры верификации и исправления функциональных нарушений, где адрес неисправного блока может быть использован для его автоматической замены на альтернативный модуль из существующей библиотеки диверсных решений.

Определенный интерес представляет задача однозначной идентификации блока с нарушениями путем анализа матрицы активизации. Для рассматриваемого примера графа и соответствующей ему таблицы необходимо поставить три обобщенных монитора в вершинах 3, 6, 9 АВС-графа. Такое эвристическое решение требует изменения структуры матрицы активизации путем добавления разрядов в некоторых координатах таблицы B (механизме ассерций) и вектора экспериментальной проверки V:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	V	A
T_1	1	.	11	.	.	.	.	.	01	.	.	.	1	.	00	3,9
T_2	1	.	.	1	.	.	.	.	.	1	.	.	.	1	1	9
T_3	1	.	.	.	1	.	.	.	.	.	1	.	1	.	0	9
T_4	.	1	.	.	.	1	.	.	.	1	.	.	.	1	1	9
T_5	.	1	.	.	.	.	1	.	.	.	1	.	1	.	0	9
T_6	.	1	.	.	.	.	.	11	.	.	.	01	.	1	01	6,9

В общем случае количество ассерций, различающих n функциональных блоков, соединенных последовательно, равно n-1. Для решения задачи мониторинга программного кода в целях достижения максимальной глубины поиска дефектных модулей можно использовать таблицу активизации, по которой достаточно просто установить классы эквивалентных программных блоков, которым соответствуют одинаковые столбцы. В каждом классе, имеющем мощность, равную n блоков, необходимо задать n-1 ассерционный монитор для всех блоков, исключая последний из них. При этом таблица активизации становится неоднородной по размерности числа битов в каждой координате, которая определяется минимальным числом мониторов для распознавания программных блоков, соединенных последовательно. Кроме того, в таблице активизации появляется столбец ассерционных мониторов, задающий последовательность вершин графа для снятия состояния ассерций в процессе моделирования. В случае более чем 50% существенности всех ассерционных мониторов для диагностирования большинства блоков целесообразно сделать таблицу активизации однородной на полном полученном множестве ассерций. В данном случае полученная структура превращается в таблицу функций неисправностей (ТФН) [2], где точки в координатах обозначают нулевые векторы ассерционных состояний на одном тест-сегменте:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	V	A
T_1	101	.	101	.	.	.	.	.	001	.	.	.	001	.	000	3,6,9
T_2	001	.	.	001	.	.	.	.	.	001	.	.	.	001	001	3,6,9
T_3	001	.	.	.	001	.	.	.	.	.	001	.	001	.	000	3,6,9
T_4	.	001	.	.	.	001	.	.	.	001	.	.	.	001	001	3,6,9
T_5	.	001	.	.	.	.	001	.	.	.	001	.	001	.	000	3,6,9
T_6	.	101	.	.	.	.	.	010	.	.	.	001	.	001	001	3,6,9

Она предоставляет полную информацию о поведении программного продукта на тестовых сегментах, где реакции создаются с помощью минимального множества ассерционных мониторов, достаточного для распознавания любого программного модуля с функциональными нарушениями. Процесс-модель диагностирования сводится к одной итерации сравнения вектора экспериментальной проверки со столбцами матрицы активизации или ТФН:

$$L = L \bigvee_{j=1}^{n} B_j \leftarrow \sum_{i=\overline{1,k}}^{r=\overline{1,m}} \left(B_{ijr} \bigoplus_{i=\overline{1,k}}^{r=\overline{1,m}} V_{ir}\right) = (0 \vee \min).$$

В данном примере использование приведенного выше критерия приводит к следующему результату: $L = B_{14} \leftarrow \forall i, r(B_{i,14,r} \oplus V_{i,r} = 0)$. Сигналы несовпадения, равные 1, отсутствуют при выполнении хог-операции между вектором V и столбцом 14 матрицы активизации. Таким образом, если тест уже имеется, то для повышения глубины диагностирования

остается единственный путь, связанный с увеличением количества ассерционных мониторов, которое повышает размерность матрицы активизации до таблицы функций неисправностей в соответствии с выражением: $Q = |T| \times |B| \times |A| = \mathrm{card}T \times \mathrm{card}B \times \mathrm{card}A$. Поэтому введение каждой новой ассерции должно иметь весомые аргументы, связанные с повышением глубины поиска дефектов, поскольку добавление каждой новой ассерции увеличивает размерность ТФН в два раза.

Таким образом, граф $F = (A * B) \times S$ выгодно отличается от существующих аналогов (количество вершин равно числу переменных, дуга соответствует одной команде или множеству, но неупорядоченных в соответствии с последовательностью операторов программного блока) [4, 22-25]: 1) Имеет минимальное число дуг, равное количеству линейных блоков программного продукта. 2) Каждой дуге (вершине) ставится в соответствие ассерция, отвечающая за правильное функционирование блока (нескольких блоков). 3) Вершина, в которую входят дуги, представляет собой минимальную совокупность только тех переменных, которые модифицируются программными блоками, входящими в нее. Такие переменные легко могут быть получены путем анализа операторов программных блоков. 4) Поскольку граф есть макроструктура программного продукта, он может быть достаточно просто построен автоматически. Следовательно, имеется возможность создавать инфраструктуру (место и вид ассерций, матрицы активизации блоков и состояния ассерций) верификации, диагностирования и восстановления работоспособности программного продукта с минимальным участием человека. 5) Наличие в графе не реального, а модельного времени существенно уменьшает размерность матриц для верификации и диагностирования функциональных нарушений. 6) Наличие структурной модели программного продукта дает возможность применить алгоритмические методы синтеза тестов активизации всех вершин и дуг графа. 7) Программный блок аргумент, а вершина есть производная от него. Такая зависимость определяет линейную вычислительную сложность для синтеза графа в целях верификации программного продукта. В упомянутых выше аналогах аргументом является вершина, что приводит к квадратичной сложности задачи синтеза графа при анализе кода.

Что касается качества модели диагностирования функциональных нарушений, то она показывает эффективность использования пары (тест, ассерции) для заданной глубины диагностирования. Оценка качества модели функционально зависит от длины теста $|T|$, числа ассерций наблюдения $|A|$, количества распознаваемых блоков с функциональными нарушениями N_d на общем числе программных блоков N:

$$Q = E \times D = \frac{]\log_2 N[}{|T| \times |A|} \times \frac{N_d}{N}.$$

Эффективность диагностирования есть отношение минимального числа двоичных разрядов, необходимых для идентификации (распознавания) всех блоков к реальному количеству разрядов кода, представленному произведением длины теста на число ассерций в каждом из них. Если первая дробь оценки равна 1 и все блоки с ФН распознаются ($N_d = N$), то тест и ассерции оптимальны, что доставляет критерию качества модели диагностирования значения, равного 1. Для примера матрицы АВС-графа, представленного на рис. 2, существует 2 решения (с одной и тремя ассерциями):

$$Q_1 = \frac{]\log_2 14[}{|6| \times |1|} \times \frac{10}{14} = 0{,}5;$$
$$Q_2 = \frac{]\log_2 14[}{|6| \times |3|} \times \frac{14}{14} = 0{,}2.$$

Несмотря на то, что качество модели лучше в первом варианте из-за меньшего объема таблицы активизации, вторая модель – более предпочтительна, поскольку она имеет максимальную глубину диагностирования, когда все 14 блоков распознаются за счет добавления двух ассерций. Оценка позволяет определить минимальные затраты по

длине теста и числу ассерций для создания модели с максимальной глубиной диагностирования.

Интерес представляет оценивание качества структуры кода проекта с позиции диагностируемости (diagnosability) блоков программного кода. Цель анализа – определить количественную оценку структуры графа и места (вершины) для установки ассерционных мониторов, создающих максимальную глубину диагностирования функциональных нарушений программных блоков. Здесь важна не управляемость и наблюдаемость, как в тестопригодности (testability), а различимость программных блоков с функциональными нарушениями, в пределе представляющая ноль блоков с эквивалентными (неразличимыми) нарушениями. Такая оценка может быть полезной для сравнения графов, реализующих одинаковую функциональность. Здесь необходимо оценивать структуру графа с позиции потенциально заложенной в нем глубины поиска функциональных нарушений программного продукта. Возможным вариантом может быть диагнозопригодность ABC-графа как функция, зависящая от таких смежных дуг при каждой вершине (формирующих число N_n), одна из которых – входящая, другая – исходящая. Такие дуги составляют пути без схождений и разветвлений (N – общее количество дуг в графе):

$$D = \frac{N - N_n}{N}.$$

Каждая вершина, объединяющая 2 дуги, которые входят в число N_n, называется транзитной. Оценка N_n есть число неразличимых функциональных нарушений (ФН) программных блоков. Места потенциальной установки мониторов для различения ФН – транзитные вершины. С учетом приведенной оценки диагнозопригодности D качество модели диагностирования программного продукта принимает вид:

$$Q = E \times D = \frac{]\log_2 N[}{|T| \times |A|} \times \frac{N - N_n}{N}.$$

Правила синтеза диагнозопригодных программных продуктов: 1) Тест (testbench) должен создавать минимальное число одномерных путей активизации, покрывающих все вершины и дуги ABC-графа. 2) Базовое число мониторов-ассерций равно количеству конечных вершин графа, не имеющих исходящих дуг. 3) В каждой вершине, имеющей одну входную и одну выходную дугу, может быть размещен дополнительный монитор. 4) Параллельно независимые блоки кода имеют n мониторов и один тест или один объединенный монитор и n тестов. 5) Последовательно соединенные блоки имеют 1 тест активизации последовательного пути и n-1 монитор или n тестов и n мониторов. 6) Вершины графа, имеющие различное число входных и выходных дуг, создают условия для диагностируемости данного участка за счет одномерных тестов активизации без установки дополнительных мониторов. 7) Совокупность тестовых сегментов (testbench) должна составлять 100% покрытие функциональных режимов (functional coverage), заданных вершинами ABC-графа. 8) Функция диагнозопригодности прямо пропорциональна длине теста, числу ассерций и обратно пропорциональна двоичному логарифму от числа программных блоков:

$$D = \frac{N - N_n}{N} = f(T, A, N) = \frac{|T| \times |A|}{]\log_2 N[}.$$

Диагнозопригодность как функция, зависящая от структуры графа (программного продукта), теста и ассерционных мониторов, всегда может быть приведена к единичному значению. Для этого существует два альтернативных пути. Первый – увеличение тестовых сегментов, активизирующих новые пути, для различения эквивалентных неисправностей (ФН) без наращивания ассерций, если структура графа программных блоков имеет такой потенциал связей. Второй – размещение дополнительных ассерционных мониторов в транзитных вершинах графа. Возможен и третий, гибридный вариант, основанный на совместном применении двух перечисленных выше путей. Отношение трех компонентов (число программных блоков, мощность механизма ассерций и длина теста) при единичном значении качества модели диагностирования и диагнозопригодности формирует плоскость

оптимальных решений $D = 1 \to \frac{|T| \times |A|}{]\log_2 N[} = 1 \to]\log_2 N[= |T| \times |A|$. Она может быть полезной для выбора квазиоптимального варианта альтернативного пути достижения полной различимости на паре $|T| \times |A|$ функциональных нарушений программных блоков.

Дискретная производная как практически полезное отношение. Дискретная производная есть симметрическая разность $A \Delta B = C$ двух объектов или явлений, где координаты их параметров заданы одинаковым булеаном на универсуме примитивов.

Булева производная есть булева разность $A \oplus B = C$ двух объектов или явлений, где координаты их параметров заданы двоичным алфавитом.

Здесь существенно, что производная есть отношение, значит – объектов или явлений должно быть два. Они должны быть описаны в одинаковых алфавитах, образующих (составляющих) булеан.

Под данные определения попадают отношения двух объектов, двух компонентов объекта, двух компонентов различных объектов, компонента и объекта:

$$A \oplus B = C;$$

$$a_i \oplus a_j = a_{ij};$$

$$a_i \oplus b_i = c_i;$$

$$a_i \oplus A = A_i.$$

Здесь следует договориться, что xor-операция будет означать и симметрическую разность, если мощность алфавита описания переменных больше двух. Размерность xor-отношения может быть как форматом целого, так и его части. Она зависит от функции цели – увидеть различие в целом – формат результата максимальный или определить степень принадлежности части к целому – формат результата минимальный. Дискретная (булева) производная инвариантна по отношению к форме (аналитическая, табличная или графовая) описания процесса или явления. Цель производной – найти различия (расстояние) в объектах или явлениях (далее для уменьшения количества слов – в объектах) путем xor-сравнения. Для определения расстояния используется бета-метрика, которая вычисляет взаимодействие любого конечного числа (n = 1,2,3, …) объектов в киберпространстве, замкнутых в цикл:

$$\beta = \bigoplus_{i=1}^{n} d_i = 0.$$

Производные на аналитической форме описания (логических) объектов достаточно исследованы и представлены многочисленными публикациями. Относительно разностного анализа графовых структур здесь область исследования представлена специфическими моделями, ориентированными на решение оптимизационных задач [26]. Производная по структурным компонентам графа дает возможность решать задачи: 1) распознавания графов и его фрагментов; 2) определения путей между двумя вершинами; 3) покрытия всех вершин и дуг минимальным множеством путей; 4) диагностирования технического состояния объектов; 5) преобразования графовых структур.

3. Метод векторно-логического анализа столбцов

Методы поиска функциональных нарушений в блоках операторов кода используют предварительно построенную таблицу ФН $B = [B_{ij}]$, где строка есть отношение между тестовым сегментом и подмножеством активизированных на данном сегменте программных блоков $T_i \approx (B_{i1}, B_{i2}, ..., B_{ij}, ..., B_{in})$. Столбец таблицы формирует отношение между программным блоком и тестовыми сегментами $B_j \approx (T_{1j}, T_{2j}, ..., T_{ij}, ..., T_{pj})$, которые активизируют его. Иначе, столбец есть вектор ассерций, идентифицирующий функциональное нарушение в соответствующем блоке. На стадии моделирования определяется реакция

$m = (m_1, m_2, ..., m_i, ..., m_p)$ механизма ассерций на тест путем формирования каждого разряда $m_i = (A_1 \vee A_2 \vee ... \vee A_i \vee ... \vee A_k)$, $A_i = \{0,1\}$ как реакции ассерций на тест-сегмент T_i. Поиск ФН основан на определении xor-операции между вектором состояния ассерций и столбцов таблицы ФН $m \oplus (B_1 \vee B_2 \vee ... \vee B_j \vee ... \vee B_n)$. Выбор решения определяется вектором B_j с минимальным числом единичных координат, формирующих программные блоки с ФН, проверяемые на тестовых сегментах. Процесс диагностирования по таблице ФН на основе реакции $m = (m_1, m_2, ..., m_i, ..., m_n), m_i = \{0,1\}$ на тест сводится к методам векторно-логического анализа столбцов или строк.

Первый метод основан на применении векторной xor-операции между m-реакцией функциональности на тест, формально рассматриваемой в качестве входного вектор-столбца, и столбцов таблицы неисправностей $m \oplus (B_1 \vee B_2 \vee ... \vee B_j \vee ... \vee B_m)$. Для подсчета качества взаимодействия векторов $Q_j(m \oplus B_j)$ в целях выбора лучшего решения определяются столбцы с минимальным числом единиц результирующего вектора. Они идентифицируют и формируют дефектные блоки с функциональными нарушениями, проверяемые на тестовых наборах. Аналитическая модель процесса получения решения в виде списка блоков с ФН, присутствующих в программном продукте, представлена в следующем виде:

$$L = L \underset{j=1}{\overset{n}{\vee}} B_j \leftarrow \sum_{i=1}^{k} (B_{ij} \overset{k}{\underset{i=1}{\oplus}} m_i) = (0 \vee \min). \quad (4)$$

Здесь фигурирует вектор экспериментальной проверки, который является входным для последующего анализа таблицы ФН:

$$m = f(A,B) \oplus f^*(A,B,L) \quad (5)$$

есть результат проведения тестового эксперимента – сравнение функционалов (состояний выходов) эталонного $f(A,B)$ и реального $f^*(A,B,L)$ устройства с дефектами L на тестовых наборах A. Во втором случае, если множество дефектных блоков $L > 1$, это означает наличие эквивалентных на данном тесте и механизме ассерций, функциональных нарушений.

Процесс-модель поиска оценки лучшего решения с минимальным числом единичных координат из не менее, чем двух альтернатив, представлена на рис. 3 и имеет следующие операции. 1) Первоначально в вектор-результат Q, в котором будет сохранено лучшее решение, заносятся единичные значения во все координаты (худшее решение) и одновременно осуществляется операция slc сдвига влево с уплотнением единиц текущего вектора Q_i. 2) Выполняется сравнение двух векторов: Q и очередной оценки Q_i из списка решений. 3) Реализуется векторная операция and $(Q \wedge Q_i)$, результат которой сравнивается с содержимым вектора Q, что дает возможность изменить его, если вектор Q_i имеет меньшее число единичных значений. 4) Процедура поиска оценки лучшего решения повторяется n раз:

$$Q = Q\overline{(\vee((Q \wedge Q_i) \oplus Q))} \vee Q_i(\vee((Q \wedge Q_i) \oplus Q));$$
$$Y = \vee((Q \wedge Q_i) \oplus Q);$$
$$Q = Q\overline{Y} \vee Q_i Y.$$

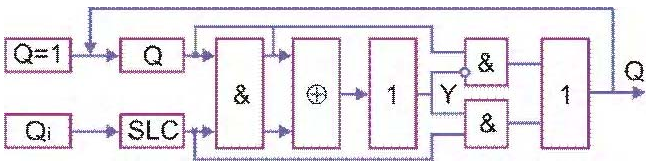
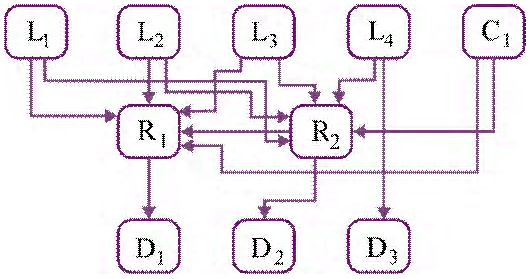


Рис. 3. Процесс-модель выбора решения

Достоинство метода векторно-логического анализа столбцов – выбор лучшего решения из всех возможных одиночных и кратных ФН. По существу, в список дефектов включают-

ся такие одиночные ФН, которые при логическом умножении на вектор экспериментальной проверки дают результат в виде вектор-столбца. Дизъюнкция всех столбцов, составляющих решение, равна вектору экспериментальной проверки $\bigvee_{j=1}^{r}(B_j \in B) = m$. Далее рассматривается пример анализа таблицы ФН блока Row_buffer, представленного на рис. 4.



Test	B_1 L_1	B_2 L_2	B_3 L_3	B_4 L_4	B_5 C_1	B_6 R_1	B_7 R_2	B_8 D_1	B_9 D_2	B_{10} D_3	m_1	m_2
T_1	1	0	0	0	0	1	0	1	0	1	0	1
T_2	0	1	0	0	0	0	1	0	1	0	1	1
T_3	0	0	1	0	0	1	1	1	0	1	0	1
T_4	0	0	0	1	0	0	1	0	1	0	1	0
T_5	0	0	0	0	1	1	1	1	0	1	0	0
T_6	1	0	0	0	0	0	1	0	1	0	1	1
T_7	0	1	0	0	0	1	0	1	0	0	0	1
T_8	0	0	1	0	0	1	0	1	0	0	0	1
T_9	0	0	1	0	0	0	1	0	1	0	1	0
T_{10}	0	0	0	1	0	1	1	1	0	0	0	0
T_{11}	0	0	0	1	0	0	0	0	0	1	0	0
T_{12}	0	0	0	0	1	0	1	0	1	0	1	0
T_{13}	0	0	0	0	1	1	1	1	0	0	0	0

Рис. 4. Row_buffer транзакционный граф и таблица ФН

На основе процедуры диагностирования (4) и таблицы ФН (см. рис. 3) можно определить дефектные компоненты методом анализа столбцов таблицы ФН. Здесь векторы m_1, m_2 формируют результаты диагностического эксперимента, выполненные по процедуре (5). Результат диагностирования одиночных и кратных ФН имеет следующий вид:

$$L^s(m_1) = m_1 \wedge (\bigvee_{j=1}^{10} B_j) = B_9 \to D_2;\ L^m(m_2) = m_2 \wedge (\bigvee_{j=1}^{10} B_j) = B_1 \vee B_2 \to L_1 \vee L_2;$$

$$Q(m_1, D_2) = 1;\ Q[m_2, (L_1 \vee L_2)] = \frac{1}{3}(\frac{4}{13} + \frac{1}{4} + 1) = 0{,}52.$$

В первом случае диагноз определен в виде одного дефектного блока D_2, присутствующего в транзакционном графе, качество решения равно 1. Во втором случае процедура диагностирования выявила наличие двух дефектных модулей $L_1 \vee L_2$, которые не смогли сформировать идеальную оценку качества. Тем не менее, решение является лучшим среди всех возможных, которое максимально приближено к вектору экспериментальной проверки по критерию принадлежности $Q[m_2, (L_1 \vee L_2)]$. Вычислительная сложность метода анализа столбцов определяется следующей зависимостью:

$$Z^c = 3n^2 + n^2 = 4n^2;\ Z^r = 3n + n = 4n.$$

Здесь первая оценка учитывает выполнение координатных операций над матрицей, размерностью $n \times n$. Вторая оценка определяет вычислительную сложность регистровых параллельных операций для подсчета критериев качества и обработки матрицы соответственно.

4. Метод векторно-логического анализа строк

Метод предназначен для определения места дефектов или ФН программного кода и состоит из двух процедур: 1) вычисление логического произведения конъюнкции строк, отмеченных единичными значениями вектора $T_i(m_i = 1)$, на отрицание дизъюнкции нулевых строк $T_i(m_i = 0)$ для одиночных дефектных блоков; 2) вычисление логического произведения дизъюнкции единичных строк на отрицание дизъюнкции нулевых строк для кратных дефектных блоков:

$$L^s = (\bigwedge_{\forall m_i = 1} T_i) \wedge (\overline{\bigvee_{\forall m_i = 0} T_i});$$
$$L^m = (\bigvee_{\forall m_i = 1} T_i) \wedge (\overline{\bigvee_{\forall m_i = 0} T_i}). \qquad (6)$$

Формулы интересны тем, что они не привязаны к критериям качества диагностирования, а оперируют лишь двумя компонентами, таблицей ФН и вектором экспериментальной проверки. Выполнение процедуры диагностирования по формулам (4) для вектора экспериментальной проверки $m_1 = (0101010010010)$, заданного в последней таблице ФН, дает результат: $L^s(m_1, T) = D_2$, который не хуже, чем ранее полученный методом анализа столбцов. Для вектора экспериментальной проверки $m_2 = (1110011100000)$ результат диагностирования имеет вид: $L^m(m_2, T) = L_1 \vee L_2$. Вычислительная сложность метода анализа строк определяется следующей зависимостью: $Z^c = n^2;\ Z^r = n.$ Первая оценка предназначена для подсчета числа координатных операций, вторая определяет вычислительную сложность процесса обработки на основе регистровых параллельных операций. Предложенные методы диагностирования ФН для программных и аппаратных продуктов есть один из наиболее существенных компонентов инфраструктуры сервисного обслуживания проектируемых изделий.

Формулы (6) могут быть модифицированы, если ввести следующие обозначения:

$$a = (\bigwedge_{\forall m_i = 1} T_i);\ b = (\bigvee_{\forall m_i = 0} T_i);\ c = (\bigvee_{\forall m_i = 1} T_i);$$
$$L^s = a\bar{b} = a \oplus ab = a(a \oplus b) = a(b \oplus 1);$$
$$L^m = c\bar{b} = c \oplus cb = c(c \oplus b) = c(b \oplus 1);$$
$$L = \begin{cases} a\bar{b} = a\bar{b} = a \oplus ab = a(a \oplus b) = a(b \oplus 1); \\ c\bar{b} = c\bar{b} = c \oplus cb = c(c \oplus b) = c(b \oplus 1) \leftarrow a\bar{b} = 0. \end{cases}$$

Любое выражение в правой части уравнений может быть использовано для определения функционального нарушения в программном или аппаратном изделии. Различие заключается в наличии или отсутствии инверсии, заменяемой xor-операцией, которая для задач диагностирования и распознавания образов зачастую является более предпочтительной. В таком случае процесс-модель диагностирования одиночных (используется a-компонент) или кратных (b-компонент) дефектов (функциональных нарушений) на основе анализа таблицы ФН будет иметь эффективную векторно-ориентированную вычислительную технологию $L = (b \oplus 1)(a \vee c)$ встроенного сервисного обслуживания программных и/или аппаратных продуктов. С позиции теории множеств это означает определение результата теоретико-множественного вычитания $L = (a \vee c) \setminus b = (a \setminus b) \vee (c \setminus b)$ в алгебрологическом векторном пространстве. Для таких операций необходим мультиматричный процессор,

строго ориентированный на параллельное выполнение нескольких логических операций над матрицами данных.

5. Матричный метод поиска функциональных нарушений в программных блоках

Метод диагностирования функциональных нарушений в программных блоках в дополнение к графу транзакций программных блоков (3) использует триаду матриц одного формата:

$$M = B \oplus A \oplus L = 0,\ L = B \oplus A \leftarrow L_{ij} = B_{ij} \oplus A_{ij} \leftarrow \{B_{ij}, A_{ij}, L_{ij}\} = \{0,1\};$$

$$B = [B_{ij}],\ A = [A_{ij}],\ L = [L_{ij}],\ i = \overline{1,n};\ j = \overline{1,m};\ \oplus = a\bar{b} \vee \bar{a}b.$$

Здесь матрицы формируют: B – активизацию блоков на тестовых сегментах в процессе моделирования; A – активность ассерций, соответствующих блокам, на тестовых сегментах также в процессе моделирования; L – дефектные блоки, полученные в результате выполнения xor-операции над двумя предыдущими матрицами. Процедура покоординатного анализа матриц использует двоичную xor-операцию, например:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	.	.	.	.	.	.	.	.
T_2	.	.	.	.	.	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	.	.	.	.	.	.	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	.	.	.	.	.	.

Полученный результат $L = B \oplus A$ в виде L-матрицы $[L_{ij}] = (T \times B \times \{0,1\})$, все координаты которой равны нулю, свидетельствует об отсутствии функциональных нарушений в программном продукте относительно предложенного плана верификации в формате (тест – функциональные блоки – активизация $[B_{ij}] = (T \times B \times \{0,1\})$, тест – ассерции – реакция $[A_{ij}] = (T \times A \times \{0,1\})$. Другой модельный эксперимент свидетельствует о наличии в программном коде функциональных нарушений $L = \{B_1, B_2, B_3, B_5, B_6\}$:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	1	.	.	1	.	.	1	.
T_2	.	0	0	.	0	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	0	.	0	.	.	0	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	0	1	1	.	.	.
$\vee A_i$	1	1	0	1	1	1	1	1

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	.	.	.	.	.	.	.	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	1	.	1	.	.	1	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	1	.	.	.	.	.
$\vee L_i$	1	1	1	0	1	1	0	0

Здесь представлены результаты векторных операций над всеми строками двух таблиц $\vee L_i$=11101100 и $\vee A_i$=11011111. Их логическое умножение с предварительной инверсией первого вектора дает координаты блоков с функциональными нарушениями, отмеченные единицами. В данном примере вектор формирует только один блок (00100000)&(11101100)=(00100000). Что является основанием для уменьшения дефектных блоков? Если предположить, что проверка первого блока в соответствии с верификационным планом должна сопровождаться выявлением дефектов на первом и шестом тестах, что не выполняется, то блок 1 можно исключить из списка неисправных. Аналогично можно поступить и с модулями 2, 5, 6. Тогда скорректированный результат будет иметь только один блок с функциональными нарушениями: $L = \{B_3\}$. Процедура уточнения результата диагностирования может быть также формализована к следующему виду: $L = B_j \leftarrow B_j \wedge L_j = B_j, j = \overline{1,m}$. Если результат сравнения отрицательный $B_j \oplus L_j = 0$, то

данный факт свидетельствует как о некорректности программного кода, так и об ассерционной или тестовой несостоятельности, включая функциональное покрытие. Для диагностирования программного кода в соответствии с процесс-моделью, имеющей вид

$$L(B,T) = (B \oplus A) \rightarrow L(B) = (\overline{\underset{i=1,n}{\vee} A_i}) \wedge (\underset{i=1,n}{\vee} L_i),$$

необходимо рассмотреть следующие пункты:

1. Покрытие (coverage) – любая метрика выбора теста и определения его полноты. Покрытие кода – code coverage – метрика теста, ориентированная на гарантированное подтверждение выполнения всех строк кода. Выполняется декомпозиция программного кода на блоки $B = \{B^s, B^t\} \leftarrow B^s \cap B^t = \varnothing, B^s \cup B^t = B$. Каждый блок принадлежит к одному из двух типов: последовательность операторов без ветвления или цепь временной задержки $B_i \in \{B^s, B^t\}$. Выполняется установка ассерционных мониторов активности блоков на тесте в начале ветвления или в первом такте цепи временной задержки. В процессе моделирования ассерции формируют матрицу активизации программных блоков на каждом тест-сегменте $B_{ij} = T_i \oplus B_j \in \{0,1\}$. Если блок активен (ассерция выполнена) на тесте (testbench), значение координаты матрицы равно 1, в противном случае – $B_{ij} = 0$. Testbench – входные условия для тестирования HDL-кода и соответствующие им выходные реакции, задающие преобразования в функциональном подпространстве проверяемого устройства.

2. Функциональное покрытие – functional coverage – метрика теста, гарантирующая достижение всех существенных состояний в пространстве определения переменных и функций программного продукта. Выполняется декомпозиция функциональности программного продукта на графы: управляющий и транзакционный $F = \{F^c, F^t\} \leftarrow F^c \cap F^t = \varnothing, F^c \cup F^t = F$. Это дает возможность существенно уменьшить размерность задач, связанных с построением соответствующих покрытий, задающих область определения переменных управления и потоков данных. Создание теста и последующая верификация (coverage driven verification) используют упомянутые графы с ограничениями (constraints), взятыми из спецификации. Синтезируемый тест для графа управления обеспечивает активизацию всех логических и арифметических переменных, участвующих в инициировании транзакций программного продукта. Способ активизации переменных или синтез теста: псевдослучайная или детерминированная (алгоритмическая) генерация тестовых воздействия, а также ручное написание входных стимулов. Форма задания покрытия: сокращенная таблица истинности, булево уравнение, двоичная диаграмма решений (binary decision diagram), граф-схема алгоритма. Тест для второго графа оперирует потоками данных, которые на системном уровне не всегда следует проверять ввиду отсутствия дефектов типа коротких замыканий между переменными или константными неисправностями в них. Граф транзакций может быть использован для создания верификационного плана существенных интерфейсных параметров программного продукта. Для этого необходимо использовать интерфейсные ассерции, оперирующие глобальными переменными.

3. Матрица ассерций программных блоков имеет вид, аналогичный структуре активизации блоков $A = [A_{ij}]$. Здесь формат ассерции как логического высказывания, использующего существенные переменные программного блока $f(X) = A_{ij} = \{0,1\}$, отвечает за функционирование соответствующего активизированного на тесте модуля $B_{ij} = 1$. Высказываний в блоке может быть несколько, раздельных для повышения глубины диагностирования или объединенных по функции or. В данном (последнем) случае ассерция несет ответственность за исправное функционирование блока. Ассерция имеет два значения: 1 – блок работает исправно, 0 – существуют функциональные нарушения. Ассерции представлены двумя уровнями иерархии: интерфейсные и блоковые $A = \{A^i, A^b\}$. Первые ориентированы на проверку существенных параметров спецификации, общих для программного продукта, которые являются внешними по отношению к последнему. Вторые встраиваются в программный блок, который не имеет разветвлений. Мощность команд или строк кода – не более 20 – определяется числом операторов, размещаемых на экране. Такой блок может включать операторы задержки по времени или событию.

6. Имплементация моделей и методов в систему верификации

Практическая реализация моделей и методов верификации была интегрирована в среду моделирования Riviera компании Aldec (рис. 5). Введенные в систему модули ассерций и диагностирования усовершенствовали существующий процесс верификации, что дало возможность на 15% уменьшить общее время проектирования цифрового изделия.

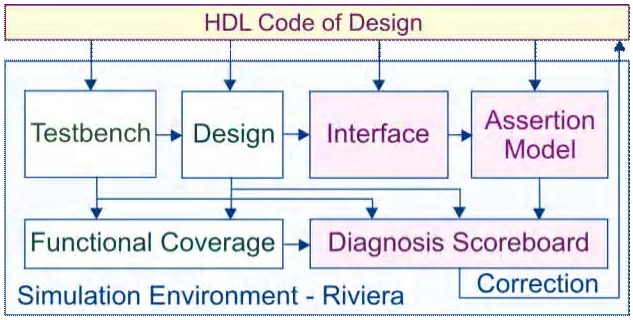


Рис. 5. Интеграция разработок в систему Riviera

Фактически применение ассерций дает возможность уменьшить объем testbench кода, а значит, существенно (x3) сократить время ручного проектирования (рис. 6), которое является наиболее дорогостоящим компонентом. Кроме того, механизм ассерций позволяет также повысить глубину диагностирования функциональных нарушений программных блоков до уровня 10-20 операторов HDL-кода.

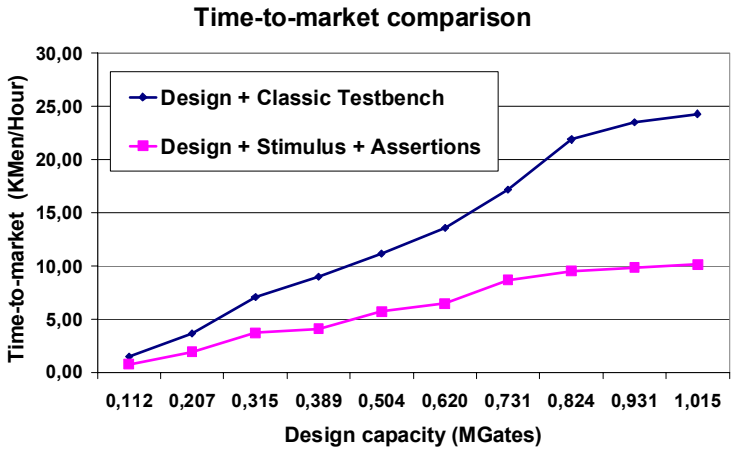


Рис. 6. Сравнительный анализ методов верификации

Благодаря взаимодействию средств моделирования и механизма ассерций, автоматически размещаемого внутри HDL-кода, появилась возможность доступа средств диагностирования к значениям всех внутренних сигналов. Это дает возможность оперативно идентифицировать место и вид функционального нарушения, а также уменьшить время обнаружения ошибок в процессе эволюции изделия при нисходящем проектировании. На основе использования практики применения ассерций для 50 реальных проектов (от 5 тыс. до 5 млн вентилей) наработаны сотни специальных решений, включенных в библиотеку верификации VTL (Verification Template Library), которая обобщает наиболее востребованные на рынке EDA (Electronic Design Automation) темпоральные ограничения при верификации широкого класса цифровых изделий. Программная реализация предложенной системы анализа ассерций и диагностирования HDL-кода входит в состав многофункциональной интегрированной среды Aldec Riviera для моделирования и верификации HDL-моделей. Высокая производительность и технологичное сочетание системы анализа ассерций с HDL-симулятором фирмы Aldec во многом обеспечивается за счет интеграции с внутренними компонентами симулятора, включая компиляторы HDL-языков. Обработка результатов работы системы анализа ассерций обеспечивается в среде Riviera представительным набором визуальных инструментов, облегчающих диагностирование и устранение

функциональных нарушений. Модель анализа ассерций также может быть реализована аппаратным способом с некоторыми ограничениями на подмножество поддерживаемых языковых конструкций. Продукт Riviera с компонентами ассерционной темпоральной верификации, позволяющими на 3-5% повысить качество проектов, в настоящее время занимает лидирующие позиции на мировом рынке электронных технологий, с числом инсталляций 5 000 в год, в 200 компаниях и университетах более чем 20 стран планеты.

7. Мультиматричный процессор бинарных операций и инфраструктура верификации

Для реализации эффективных с позиции времени и затрат вычислительных процессов, связанных с диагностированием функциональных нарушений, необходим простой по архитектуре процессор с минимальной системой команд, где в качестве операндов выступают не только булевы переменные, но и более сложные структуры, такие как регистры и матрицы. Такой процессор должен выполнять в параллельном режиме операции над всеми разрядами регулярных операндов, не требуя специальных компиляторов распараллеливания вычислительных процессов.

Мультиматричный процессор (ММП) есть такая минимальная архитектура инструкций-примитивов, где каждый из них ориентирован на параллельное выполнение только одной (and, or, xor, slc) операции над соответствующей матрицей (двумерный массив данных). Количество командно-ориентированных матриц-примитивов создает систему – гетерогенный мультиматричный процессор бинарных операций с буфером М (рис. 7).

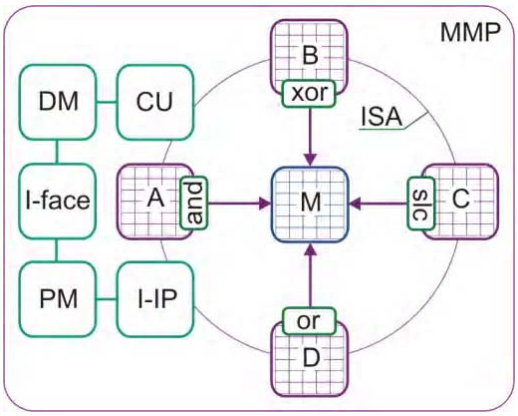


Рис. 7. Мультиматричный процессор бинарных операций

Здесь представлены стандартные блоки: памяти данных DM и программ PM, управления CU, интерфейс I-face и сервисного обслуживания I-IP, а также мультиматричный модуль процессора, включающий 4 блока памяти со встроенными в них операциями (A – and, B – xor, C – slc, D – or – shift left crowding) и буферную память M. Мультиматричный процессор (MMP) ориентирован на параллельное выполнение в данном случае одной из четырех инструкций (ISA – Instruction Set Architecture), оперирующей матрицами двоичных данных одинаковой размерности: $M = M\{and, or, xor, slc\}\{A, B, C, D\}$ с занесением результата в буфер M. Особенность ММП в том, что не ячейка матрицы имеет систему команд из четырех операций, а каждая команда имеет собственную матрицу ячеек в качестве данных для параллельной обработки, что существенно упрощает структуру управления и устройства в целом. Вся сложность ММП перенесена на структуры данных, где память матрицы имеет одну аппаратно-реализованную встроенную команду, что позволяет иметь примитивную систему управления параллельными вычислительными процессами (SIMD – Single Instruction Multiple Data), последовательностную по своей сути, а значит, нет необходимости создавать сверхсложные компиляторы, ориентированные на распараллеливание вычислительных процессов. Представленная архитектура ММП адаптируется к выполнению логических операций над операндами регистрового уровня. Прототип ММП интегрирован в плату аппаратного ускорения процессов моделирования и верификации HES™ компании Aldec.

На основе мультиматричного (-регистрового) процессора создана инфраструктура (рис. 8) верификации HDL-кода проектируемых цифровых систем на кристаллах, которая является модификацией I-IP стандарта 1500 [3, 4, 11, 14]. Здесь фигурируют четыре процесс-модели: тестирование на стадии моделирования, диагностирование функциональных нарушений, оптимизация диагноза, восстановление работоспособности.

1. Процесс-модель тестирования включает HDL-модель, механизм ассерций, testbench и coverage (покрытие). Последнее оценивает качество теста проверки всех состояний проекта. В результате моделирования синтезируется матрица активизации программных блоков B и матрица ассерционных реакций A на тестовые сегменты, которая может быть трансформирована к вектору состояния ассерций m путем применения функции or к вектор-столбцам A-матрицы:

$$\begin{cases} B = (T \oplus F); \\ m = \bigvee\limits_{j=1}^{m} A_j \leftarrow A = (T \oplus A^c). \end{cases}$$

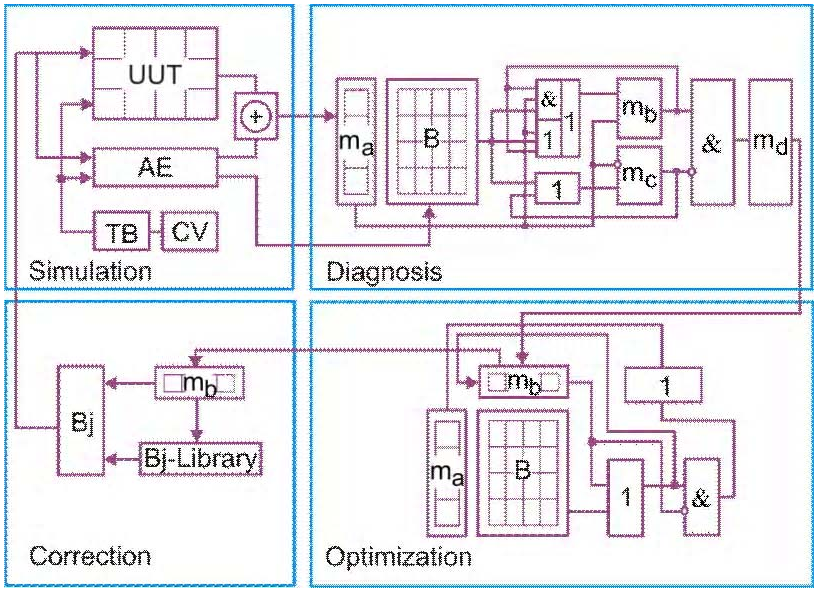


Рис. 8. Инфраструктура верификации HDL-кода

2. Два последних компонента используются во второй процесс-модели для диагностирования блоков HDL-кода. Результатом диагностирования является вектор дефектов, формирующий подмножество блоков m_d с функциональными нарушениями. При этом не исключены ошибки как в testbench, так и в ассерционных операторах, отвечающих за тестирование и мониторинг программных блоков. Тройственная неопределенность диагноза $D = \{B_j, T_i, A_{ij}\}$ характерна при отсутствии точной идентификации блока в процедуре сравнения столбцов матрицы активизации с вектором ассерционных реакций.

3. Третий блок решает задачу минимизации числа блоков, подозреваемых в наличии функциональных нарушений, до одного из них. При этом используется матрица активизации блоков и диагноз m_d, полученный в предыдущей процесс модели.

4. Модуль исправления функциональных нарушений ориентирован на ручной поиск ошибок в одном программном блоке, представленном вектором m_b. Возможен также автоматический режим исправления ошибок в блоках, если в инфраструктуре верификации предусмотрена библиотека диверсных программных модулей, имеющих аналогичные функциональности.

Предложенная инфраструктура является одним их шагов на пути создания автомата верификации программных блоков.Далее представлен пример диагностирования функцио-

нального нарушения на основе использования матрицы активизации. Вектор ассерционных реакций получен из соответствующей матрицы $A_{ij} = \{1 \to \text{failed}, 0 \to \text{passed}\}$ путем дизъюнктивного объединения содержимого каждой строки:

$$m_i = \bigvee_{j=1}^{m} A_{ij} =$$

T	m
T_1	.
T_2	1
T_3	.
T_4	1
T_5	.
T_6	.
T_7	.
T_8	1

=

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	.	.	.	.	.	.	.	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	1	.	1	.	.	1	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	1	.	.	.	.	.

Последующее выполнение xor-операции между вектором ассерций и столбцами матрицы активизации блоков позволяет найти лучшее решение, которое определяется минимальным кодовым расстоянием $L = L \vee B_j \leftarrow \sum_{i=1}^{n}(B_{ij} \oplus_{i=1}^{n} m_i) = (0 \vee \min)$:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

T	m
T_1	.
T_2	1
T_3	.
T_4	1
T_5	.
T_6	.
T_7	.
T_8	1

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	1	.	.	1	.	1	1	1
T_3	.	.	.	.	.	1	1	1
T_4	.	1	.	1	1	.	1	1
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	1	1	.	.	.	1	1	1
$d(A,B_j)$	4	4	0	4	2	4	6	6

Результат диагностирования – блок 3 имеет функциональные нарушения, поскольку три ассерции «упали» на тестовых сегментах 2,4 и 8, которые в таком сочетании активизируют только блок с номером 3. Если выполнять процесс диагностирования, используя не вектор, а матрицу ассерций, то процесс поиска дефектных блоков будет иметь следующий вид:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	.	.	.	.	.	.	.	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	1	.	1	.	.	1	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	1	.	.	.	.	.

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	.	.	.	.	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	.	.	.	.	.	.	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	.	1	1	.	.	.
$d(A,B_j)$	2	2	0	3	2	2	3	3

Результат диагностирования аналогичен предыдущему – блок 3 имеет функциональные нарушения, поскольку кодовое расстояние равно нулю только для столбца с номером 3.

8. Заключение

1. Представлена структурная модель отношений на множестве из четырех основных компонентов технической диагностики (функциональность, устройство, тест, дефекты), которая характеризуется полным xor-взаимодействием всех вершин графа и транзитивной обратимостью каждой триады отношений, что позволяет определить и классифицировать пути решения практических задач, включая синтез тестов, моделирование неисправностей и поиск дефектов.

2. Предложена новая модель программного продукта в форме графа блочных транзакций, а также новый матричный метод диагностирования функциональных нарушений, которые характеризуются технологичностью подготовки данных в процессе поиска некорректных блоков, что дает возможность существенно уменьшить время проектирования цифровых систем на кристаллах.

3. Усовершенствованы методы поиска функциональных нарушений, которые отличаются параллелизмом выполнения векторных операций над строками таблицы ФН, что дает возможность существенно (x10) повысить быстродействие вычислительных процедур, связанных с диагностированием и восстановлением работоспособности программных и аппаратных продуктов.

4. Разработана архитектура мультиматричного процессора, ориентированного на повышение быстродействия процедур встроенного диагностирования функциональных нарушений в программном или аппаратном изделии, которая отличается использованием параллельных логических векторных операций and, or, xor, slc, что дает возможность существенно (x10) повысить быстродействие диагностирования одиночных и/или кратных дефектов (функциональных нарушений).

5. Предложена инфраструктура верификации и диагностирования HDL-кода проектируемых цифровых систем на кристаллах, которая имеет четыре процесс-модели для тестирования, диагностирования, оптимизации и исправления ошибок, замкнутые в цикл, что дает возможность уменьшить время отладки кода в процессе создания проекта.

6. Практическая реализация моделей и методов верификации была интегрирована в среду моделирования Riviera компании Aldec. Введенные в систему модули ассерций и диагностирования усовершенствовали существующий процесс верификации, что дало возможность на 15% уменьшить общее время проектирования цифрового изделия.

Список литературы: 1.*Основы* технической диагностики / Под. ред. П.П.Пархоменко. М.: Энергия, 1976.460с. **2.** *Пархоменко П.П., Согомонян Е.С.* Основы технической диагностики (Оптимизация алгоритмов диагностирования, аппаратурные средства) / Под ред. П.П. Пархоменко.М.: Энергия. 1981. 320 с. **3.** *Инфраструктура* мозгоподобных вычислительных процессов / М.Ф. Бондаренко, О.А. Гузь, В.И. Хаханов, Ю.П. Шабанов-Кушнаренко. Харьков: Новое слово. 2010. 160 с. **4.** *Проектирование* и верификация цифровых систем на кристаллах / В.И. Хаханов, И.В. Хаханова, Е.И. Литвинова, О.А. Гузь. Харьков: Новое слово. 2010. 528с. **5.** *Семенец В.В., Хаханова И.В., Хаханов В.И.* Проектирование цифровых систем с использованием языка VHDL. Харьков: ХНУРЭ. 2003. 492 с. **6.** *Хаханов В.И., Хаханова И.В.* VHDL+Verilog = синтез за минуты. Харьков: ХНУРЭ. 2006. 264с. **7.** *Хаханов В.И.* Техническая диагностика цифровых и микропроцессорных структур: Учебник. К.: ИСИО, 1995. 242с. **8.** *Скобцов Ю.А.* Логическое моделирование и тестирование цифровых устройств Ю.А. Скобцов, В.Ю. Скобцов. Донецк: ИПММ НАН Украины, ДонНТУ, 2005. 436 с. **9.** *IEEE* Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture IEEE Std 1149.7-2009. 985 p. **10.** *Da Silva F., McLaurin T., Waayers T.* The Core Test Wrapper Handbook. Rationale and Application of IEEE Std. 1500™. Springer. 2006. XXIX. 276 p. **11.** *Marinissen E.J., Yervant Zorian.* Guest Editors' Introduction: The Status of IEEE Std 1500.IEEE Design & Test of Computers. 2009. No26(1). P.6-7. **12.** *IEEE* Std 1800-2009 IEEE Standard for System Verilog-Unified Hardware Design, Specification, and Verification Language. http://ieeexplore.ieee.org/servlet/opac?punumber= 5354133. **13.** *Marinissen E.J.* Testing TSV-based three-dimensional stacked ICs // DATE 2010. 2010. P.1689-1694. **14.** *Benso A., Di Carlo S., Prinetto P., Zorian Y.* IEEE Standard 1500 Compliance Verification for Embedded Cores // IEEE Trans. VLSI Syst. 2008. No 16(4).P. 397-407. **15.** *Ubar R., Kostin S., Raik J.* Embedded diagnosis in digital systems // 26th International Conference "Microelectronics", MIEL 2008. 2008.P. 421-424. **16.** *Elm M., Wunderlich H.-J.* Scan Chain Organization for Embedded Diagnosis // Design, Automation and Test in Europe, DATE '08. 2008. P. 468–473. **17.** *Bulent I. Dervisoglu.* A Unified DFT Architecture for Use with IEEE 1149.1 and VSIA/IEEE P1500 Compliant Test Access Controllers. Proceedings of the Design Automation Conference. 2001. P. 53-58. **18.** *Chenlong Hu, Ping Yang, Ying Xiao, Shaoxiong Zhou.* Hardware design and realization of matrix converter based on DSP & CPLD // 3rd International Conference Power Electronics Systems and Applications. 2009. P. 1-5. **19.** *Dave N., Fleming K., Myron King, Pellauer M., Vijayaraghavan M.* Hardware Acceleration of Matrix Multiplication on a Xilinx FPGA // 5th IEEE/ACM International Conference Formal Methods and Models for Codesign. 2007. P.97-100. **20.** *Loucks W.M., Snelgrove M., Zaky S.G.* A Vector Processor Based on One-BitMicroprocessors // IEEE Micro.Volume 2, Issue 1.1982. P. 53-62. **21.** *Hilewitz Y., Lauradoux C., Lee R.B.* Bit matrix multiplication in commodity processors // International Conference Application-Specific Systems, Architectures and Processors. 2008. P. 7-12. **22.** *Soon, J.L.K.; Low Ching Ling;* DEV. Design explorer for verification. Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium: 2009. P.

413– 416. **23.** *Rafe V.; Rafeh R.; Azizi S.; Miralvand M.R.Z.;* Verification and Validation of Activity Diagrams Using Graph Transformation. Computer Technology and Development, 2009. ICCTD '09. 2009. P. 201- 205. **24.** *Xiaoxi Xu; Cheng-Chew Lim;* Using Transfer-Resource Graph for Software-Based Verification of System-on-Chip. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume: 27, Issue: 7. 2008. P. 1315 – 1328. **25.** *Zhongjun Du; Zhengjun Dang*. A New Algorithm Based Graph-Search for Workflow Verification. Information Engineering and Computer Science (ICIECS), 2010. 2nd International Conference: 2010. P. 1–3. **26.** *Горбатов В.А., Горбатов А.В., Горбатова М.В.* Дискретная математика. М: Высшая школа, 2006. 448с.

Поступила в редколлегию 16.03.2011

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.

Хаханов Владимир Иванович, декан факультета КИУ ХНУРЭ, д-р техн. наук, проф. 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.

Зайченко Сергей Александрович, аспирант кафедры АПВТ ХНУРЭ. Научные интересы: проектирование и верификация цифровых систем на кристаллах. Увлечения: музыка, путешествия, литература. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-326. E-mail: sergeyz@aldec.com.

Литвинова Евгения Ивановна, д-р техн. наук, проф. кафедры АПВТ 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.

Скворцова Ольга Борисовна, канд. техн. наук, доцент кафедры АПВТ 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: olgas@aldec.com.