УДК 681.326:519.613

С.А. ЗАЙЧЕНКО, С.В. ЧУМАЧЕНКО

DRTLQ-МОДЕЛЬ ДЛЯ ФУНКЦИОНАЛЬНОЙ ВЕРИФИКАЦИИ ЦИФРОВЫХ СИСТЕМ НА ОСНОВЕ ЛИНЕЙНОЙ ТЕМПОРАЛЬНОЙ ЛОГИКИ

Предлагается аналитическая модель верификации, которая характеризуется использованием динамических регистровых очередей для анализа темпоральных ассерций в процессе моделирования тестов для цифровых систем на кристаллах, что обеспечивает высокое быстродействие моделирования и заданную глубину диагностирования ошибок кода.

1. Введение

Актуальность исследования. Специализированные и стандартизованные системы, согласно топ-десятке от Gartner Research Group, есть одно из главных направлений развития рынка электронных технологий. Оно определяется существенными инвестициями со стороны ведущих компаний планеты, формирующих индекс NASDAQ, включая Intel, Xilinx, Apple, которые предлагают интересные решения в виде планшетов, телекоммуникационных и DSP изделий. Данные компании ориентируются на дальнейшее развитие сегмента рынка, основанного на предоставлении новых сервисов со стороны компактных и энергосберегающих цифровых систем на кристаллах с годовым выпуском более 2,5 миллиардов.

Вентильная мощность силиконового кристалла, составляющая до 1 млрд транзисторов, заставляет производителей искать новые эффективные пути системного и RTL- проектирования, тестовой и временной верификации, а также синтеза функциональных модулей на основе языков System Verilog и Verilog с использованием IEEE стандартов тестопригодного проектирования и новейших ESL Design технологий. Что касается проектирования цифровых проектов, то здесь следует выделить наиболее сложный и дорогостоящий этап современного цикла проектирования – функциональная верификация, как процесс поиска, обнаружения и устранения ошибок системной модели относительно спецификации, который составляет до 70% общего времени создания проекта. Для достижения качества проекта, удовлетворяющего требованиям рынка электронных технологий, ведущие компании координируют свои действия по созданию и внедрению современных технологий тестирования и верификации, которые формируют инфраструктуры тестопригодного проектирования моделей системного регистрового и вентильного уровней описания проекта. Такие действия приводят к повышению эффективности проектирования под эгидой специализации и стандартизации решений, определяемых совместным использованием трех взаимно противоречивых параметров: качество, быстродействие, аппаратурные затраты.

Таким образом, исследование направлено как на создание новых моделей и методов верификации, так и на эффективное устранение ошибок , связанных с: 1) ошибками, допускаемыми инженерами в системной модели, тестах и спецификации в процессе проектирования; 2) несовершенством средств диагностирования в системах автоматизации, затрудняющих локализацию и устранение причины возникновения ошибки; 3) недостаточной производительностью и точностью программных систем автоматизации, качество которых растет существенно медленнее увеличения сложности обрабатываемых моделей. Комплексное решение проблемы верификации системных моделей позволит в значительной степени снизить затраты на проектирование цифровых систем на кристаллах. Согласно исследованиям ведущих мировых компаний в области EDA (Cadence Design Systems, Synopsys Inc., Mentor Graphics Corporation, Magma, IBM, Intel, Sun Microsystems, Cisco Systems Inc., Atrenta, Aldec Inc.) усилия ученых должны быть сосредоточены на создании эффективных методов верификации, способных: 1) в десятки раз снизить вероятность возникновения ошибок за счет уменьшения участия человека в процессе проектирования; 2) обеспечить обнаружение и диагностирование допущенных неточностей на ранней стадии проектирования в целях сокращения времени и стоимости устранения несоответствий спецификации; 3) на порядок повысить производительность и надежность систем верифи-

кации за счет повышения уровня абстракции, как самих моделей, так и тестовых воздействий.

Цель исследования – разработка моделей и методов функциональной верификации цифровых систем на кристаллах на основе использования темпоральных ассерций для тестового диагностирования ошибок в процессе программно-аппаратного моделирования, обеспечивающего существенное повышение качества цифрового изделия, а также уменьшение временных и материальных затрат проектирования.

Для достижения цели необходимо решить следующие *задачи*: 1. Разработать математическую аналитическую модель верификации цифровых систем на основе использования динамических регистровых очередей для анализа ассерций линейной темпоральной логики. 2. Спроектировать эффективные структуры данных и разработать модель процесса верификации, осуществляющих подготовительную обработку языковых описаний ассерций и их проверку в рамках цикла имитационного моделирования системы на кристалле в HDL-симуляторе.

Объект исследования – процесс проектирования и верификации цифровых систем на кристаллах с помощью языков описания аппаратуры высокого уровня и HDL-симулятора.

Предмет исследования – системные модели и методы проектирования, верификации цифровых систем на кристаллах на основе использования линейной темпоральной логики и языков описания ассерций

Методы исследования – булева алгебра, линейная темпоральная логика, теория множеств, теория графов, теория цифровых автоматов – для построения математической модели верификации; объектно-ориентированный анализ, теория алгоритмов, методы проектирования программных систем, теория формальных языков, методы динамического моделирования цифровых систем – для построения структур данных и разработки программной системы верификации; экспериментальные исследования производительности верификации моделей цифровых систем – для достижения быстродействия предложенных методов; методы логического синтеза, методы верификации пересечений тактовых доменов, методы анализа функционального покрытия, методы генерации ограниченных псевдослучайных тестовых воздействий – для выработки маршрутов практического применения разработанной системы.

Источники: алгоритмы перехода от традиционных моделей конечных автоматов к структурам Крипке и алгоритмы обратного перехода [1, 2], основы темпоральной логики [3], автоматы Buchi [4-7], алгоритмы и модели минимизация автоматов [8-10], темпоральные ограничения при имитационном моделировании системы (симуляции) [11], методы формальной верификации [12, 13].

2. Модель динамических регистровых очередей

2.1 Основные элементы модели динамических регистровых очередей. Типичная математическая база анализа ассерций во время симуляции, состоящая в трансформации операторов LTL-логики к детерминированным конечным автоматам:

$$A = \{\Sigma, Q, q_0, F, \partial\},$$

(где Σ – алфавит системы, Q – множество всех состояний автомата, $q_0 \in Q$ – единственное начальное состояние, $F \subseteq Q$ – множество конечных состояний, $\partial : Q \times \Sigma$ – функция переходов), лишь частично способна удовлетворить основные требования к верификации на основе ассерций. Основной проблемой классической модели является экспоненциальный рост количества состояний в зависимости от структурной сложности реализуемой темпоральной формулы. Кроме того, накладывается существенное ограничение на языки описания ассерций в виде простого подмножества, поскольку классическая модель неспособна осуществить проверку ряда темпоральных конструкций с приемлемой для практического применения вычислительной сложностью. Поддержка введенного в [14] режима глобального времени практически невозможна на основе модели конечных автоматов.

Очевидно, для обеспечения существенно лучшей производительности анализа ассерций и поддержки темпоральных операторов, определяемых для бесконечных вычислительных путей, необходима принципиально иная модель. Такими свойствами обладает предлагаемая в данном исследовании модель динамических регистровых очередей (DRTLQ –

Dynamic Register-Transfer-Level Queues). Модель ориентирована на высокопроизводительный анализ элементов линейной темпоральной логики и эффективность обнаружения и локализации нарушений. Ключевые понятия, вводимые в модели, представляют собой обобщенную надстройку над общеизвестными терминами LTL и языков описания ассерций.

Модель DRTLQ можно логически разделить на четыре взаимодействующих уровня, соответствующих трем уровнями в языках описания ассерций, определенным в [14] (рис. 1).

Рис. 1. Уровни модели DRTLQ в контексте уровней семантики ассерций

Наиболее общим понятием модели DRTLQ является ассерционный процесс – совокупность элементов булевого, темпорального и верификационного уровня семантики ассерций, реализующая верификацию логически связанной группы темпоральных утверждений. Наиболее близким к ассерционному процессу языковым термином является единица верификации (verification unit) из языка PSL. Ассерционным процессом AP в модели DRTLQ называется целостная совокупность множества верификационных переменных B, списка текущих потоков активации Ω, очереди последовательностных функций F и темпоральных свойств P, одного ассерционного монитора M, как правило, относящаяся к определенному функциональному блоку системы и реализующая верификацию логически связанной группы темпоральных утверждений в процессе моделирования:

$$AP = \{B, \Omega, F, P, M\}. \quad (1)$$

Ассерционный монитор M в модели DRTLQ – вершина темпорального утверждения в ассерционном процессе, осуществляющая верификацию некоторого одного темпорального свойства и управление обработкой результата этой верификации в рамках родительского тест-бенча. Главной задачей ассерционного монитора является обеспечение задаваемой тест-бенчем реакции на результат верификации темпорального утверждения.

В простейших случаях такой реакцией может быть некоторое текстовое сообщение в консоли системы моделирования, визуальная демонстрация результата анализа ассерции на временных диаграммах и других графических инструментов отладки. Возможны и другие виды реакции на результат верификации утверждения, зависящие от конкретного используемого верификационного языка. В обычном режиме интерпретации уровень последовательностных функций непосредственно взаимодействует с монитором.

Атомарным объектом анализа ассерций в модели DRTLQ является верификационная переменная $b \in B$ – любой объект верифицируемой системы, текущее значение или состояние которого используется как операнд темпоральных соотношений в рассматриваемом ассерционном процессе и тем или иным преобразованием может быть приведено к булевому типу {0, 1}.

В качестве верификационной переменной может выступать как элементарный объект верифицируемой модели – константа (параметр), скалярный или векторный сигнал, переменная встроенного типа, так и сложные объекты, такие как выражения, функции, объекты классов.

Модель DRTLQ полностью абстрагируется от особенностей конкретного типа верифицируемых данных, языка описания или среды проектирования. Обеспечение преобразования текущего значения верификационной переменной к булевому типу, а также уведомление об изменении текущего значения относятся к обязанностям системы моделирования, которая взаимодействует с верификационной системой. Такое отстранение от деталей верифицируемой модели имеет ряд немаловажных преимуществ:

– абстрагирование от типов данных способствует ускорению анализа ассерций, поскольку только система моделирования, непосредственно задающая формат и способы преобразования данных, способна произвести необходимое приведение с оптимальной производительностью;

– значительное упрощение и ускорение анализа ассерций достигается за счет отказа от рассмотрения небулевых значений, таких как X и Z – любое верификационное утверждение, требующее учета 4-значности, может быть сведено к выражению с булевым результатом;

– абстрагирование от типов данных делает модель DRTLQ универсальной относительно предметной области данных – этот факт эффективно применяется при верификации высокоуровневых моделей систем на кристалле, в частности, на основе технологии SystemC, где в качестве переменной может быть использовано состояние объекта-транзактора.

Также верификационные переменные могут представлять другие элементы языков описания ассерций, не имеющие эквивалента в языках моделирования верифицируемых систем, в частности:

– абстрактное понятие перманентной истины и лжи (булевы константы, используемые для реализации семантики более сложных операторов);

– признак окончания анализа заданной последовательности событий (понятие endpoint в языке PSL, встроенные функции ended/matched языков SystemVerilog и OVA);

– локальные переменные ассерционного процесса (SystemVerilog).

Вычислительные процедуры, осуществляющие анализ темпоральных операторов в ассерционном процессе, выполняются при изменении значений верификационных переменных. Тактовой верификационной переменной называется верификационная переменная, момент изменения значения которой инициирует обработку группы элементов ассерционного процесса.

В общем случае, ассерционный процесс может иметь несколько тактовых переменных. Если таковых не определено, вычисления инициируются изменением значения любой из участвующих в соотношении верификационных переменных.

2.2. Потоки активации и транспортирование событий. В моменты изменения верификационные переменные создают события:

$$e = \{e_{\leftarrow}, e_{\uparrow\rightarrow}, e_{\downarrow\rightarrow}, \rho, t_b, t_a\}, \qquad (2)$$

где $e_{\leftarrow}, e_{\uparrow\rightarrow}, e_{\downarrow\rightarrow}$ – связи события с другими событиями; ρ – характеристический вектор события; t_b и t_a – время создания и активации события соответственно, измеряемое в количестве тактовых событий с момента начала моделирования.

Событие называется активированным, если в ходе обработки оно достигло элемента ассерционного процесса, создающего поток активации. Поток (или кольцо) активации $\omega_k \in \Omega$ в модели DRTLQ – динамическое множество всех обрабатываемых событий $e^i_k, i \geq 0, k \geq 0$, соответствующих одному вычислительному пути π, начинающемуся с некоторого тактового цикла с порядковым номером k.

Событие может быть связано с другими событиями в двух различных видах цепочек. Правая цепочка $r_{\rightarrow}$, формируемая двунаправленными событийными связями $e_{\uparrow\rightarrow}, e_{\downarrow\rightarrow}$, соответствует событиям, связанным одним кольцом активации. Основная цель связывания событий в правые цепочки состоит в возможности каскадных операций над всем потоком (например, уничтожение при завершении потока). Левая однонаправленная цепочка $r_{\leftarrow}$, формируемая событийными связями $e_{\leftarrow}$, соответствует событиям различных колец активации, которые одновременно транспортируются через выбранный элемент модели. Правая цепочка неупорядочена, она формируется свободно, здесь порядок не играет никакой роли, а левая цепочка упорядочена по времени создания события, что принципиально важно для конвейерной обработки нескольких потоков активации. Наглядно цепочки событий можно изобразить на рис. 2.

Характеристический вектор события ρ – короткое двоичное слово, транспортируемое в составе события, индивидуальные биты которого моделируют семантические флаги-модификаторы, оказывающие влияние на обработку события. Важнейшие флаги-модификаторы представлены в табл. 1.

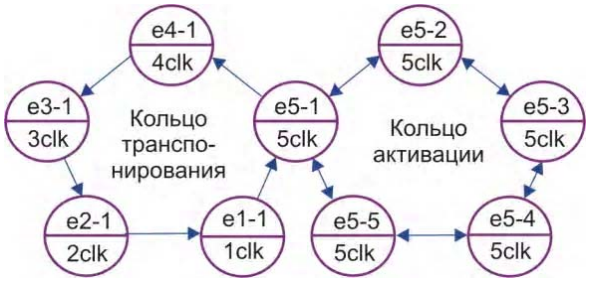

Рис. 2. Цепочки событий в модели DRTLQ

Таблица 1. Компоненты характеристического вектора события

Обозначение	Описание
ρ^{VAL}	Текущее значение события
ρ^{LDEAD}	Событие должно быть отделено от левой цепочки
ρ^{RDEAD}	Событие должно быть отделено от правой цепочки
ρ^{ACTIVE}	Событие активировано
ρ^{RTOP}	Событие является вершиной потока активации
ρ^{ABORT}	Событие исходит от элемента прерывания анализа

В ассерционном процессе всегда существует не более одного потока активации, соответствующего некоторому определенному начальному моменту времени. Одновременно может существовать несколько потоков активации, однако все они будут относиться к вычислительным путям, соответствующим различным начальным моментам. Событие может принадлежать не более чем одному потоку активации. Активированное событие всегда относится к единственному потоку активации. Если событие не относится ни к одному потоку активации, оно не является активированным. Моменты t_b и t_a в общем случае могут не совпадать (в частности, при анализе операторов последовательностной импликации), но для большинства простых темпоральных формул $t_b = t_a$. В ряде случаев событие может не быть активировано в течение своего существования, если путь его обработки носит вспомогательный характер для анализа определенного потока активации. Активация события осуществляется первым элементом на пути обработки события, удовлетворение критерия работы которого позволяет производить дальнейший анализ вычислительного пути.

Поскольку все события в рамках потока активации связаны, удовлетворение потока активации моментально уничтожает всю цепочку связанных с ним событий. Условие удовлетворения потока активации определяется целью верификации. Если целью является опровержение возможности некоторого поведения в системе, то первое же событие с текущим значением $\rho^{VAL} = 1$, достигающее некоторой контрольной точки в модели, опровергает утверждение в целом. Лишь в случае, если все события потока активации на пути обработки при достижении контрольной точки имели значение $\rho^{VAL} = 0$, опровергающее утверждение удовлетворяется успешно. Если целью верификации является, наоборот, доказательство определенного поведения системы, то критерии удовлетворения потока активации диаметрально противоположны: первое событие, достигающее контрольной точки с текущим значением $\rho^{VAL} = 0$, опровергает утверждение, и лишь успешное завершение всех альтернативных событий потока активации со значением $\rho^{VAL} = 1$ говорит о достоверности утверждения.

Транспортирование событий осуществляется путем перегруппировки левых цепочек. Левые цепочки ассоциируются с элементами модели более высокого уровня, формирую-

щими пути транспортирования событий. Пусть $r^1_{\leftarrow} : \{e^1_1 \cdots e^1_{N_1}\}$ представляет собой левую цепочку, из которой транспортируются события, а $r^2_{\leftarrow} : \{e^2_1 \cdots e^2_{N_2}\}$ – левую цепочку, к которой они направляются. Суть задачи простого $r'^2_{\leftarrow} : r^2_{\leftarrow} \cup r^1_{\leftarrow}$ и копирующего транспортирования $r'^2_{\leftarrow} : к(r^2_{\leftarrow}) \cup к(r^1_{\leftarrow})$ состоит в корректном расширении цепочки $r^2_{\leftarrow}$ событиями из цепочки $r^1_{\leftarrow}$ с сохранением относительного порядка событий в результирующей цепочке $r'^2_{\leftarrow}$:

$$\begin{cases} e_{\leftarrow}(e'^2_0) = \varepsilon; \\ \forall k, 1 \le k < |r'^2_{\leftarrow}| - 1, e'^2_{k-1} = e_{\leftarrow}(e'^2_k) \Rightarrow t_b(e'^2_{k-1}) < t_b(e'^2_k). \end{cases} \quad (3)$$

Слияние цепочек событий происходит начиная с правого края с одновременным итерированием в единственно допустимом направлении, при этом не теряется позиция вставки, что обеспечивает линейную сложность алгоритма относительно суммы $|r^1_{\leftarrow}| + |r^2_{\leftarrow}|$. Вставка предполагает изменение связей $e_{\leftarrow}$ непосредственно вносимого в цепочку события, а также события, следующего за ним. При копирующем транспортировании оригинальные цепочки не модифицируются, в то время как простое транспортирование полностью очищает $r^1_{\leftarrow}$ и перезаписывает результатом $r'^2_{\leftarrow}$ цепочку $r^2_{\leftarrow}$.

Помимо основной задачи (3), осуществляется дополнительная задача элиминации лишних событий, руководствующаяся двумя критериями:

1. Если $\rho^{RDEAD}(e^k_1) = 1$, т.е. вычислительный поток события уничтожен, событие должно быть отключено от левой цепочки и уничтожено. Гарантируется, что событие с установленным флагом ρ^{RDEAD} будет уничтожено при первой же попытке транспортирования; соответственно, можно утверждать, что для любого события целевой цепочки $\rho^{LDEAD}(e^k_2) = 0$.

2. Если для двух соседних событий в результирующей цепочке выполняется условие $t_b(e'^2_{k-1}) = t_b(e'^2_k)$, а $\rho(e'^2_{k-1}) = \rho(e'^2_k)$ (характеристические векторы идентичны), то одно из событий можно уничтожить.

Частным случаем задачи транспортирования (3) является случай, при котором $|r^2_{\leftarrow}| = 0$. Такое упрощение сводит задачу к переносу тех элементов цепочки $r^1_{\leftarrow}$ в результирующую цепочку $r'^2_{\leftarrow}$, которые не попадают под действие описанных выше критериев элиминации. Этот процесс также характеризуется линейной сложностью, однако только относительно $|r^1_{\leftarrow}|$, и, в силу отсутствия необходимости редактирования событийных связей, имеет существенно лучшие показатели производительности.

2.3. Последовательностные функции в DRTLQ-модели. Последовательностной функцией $f \in F$ в модели DRTLQ называется некоторая вычислительная процедура, характеризующаяся множествами входных, внутренних и выходных событий, осуществляющая их пошаговое преобразование на каждом тактовом цикле верификации. Результатом последовательностной функции в некоторый момент времени (номер тактового цикла с начала моделирования), является содержимое множества выходных событий в этом моменте времени. Активирующей последовательностной функцией является такая последовательностная функция, выходные события которой, во-первых, инициируют некоторый поток активации, соответствующий текущему тактовому циклу, если таковой еще не был инициирован; во-вторых, назначают связи между событиями по правой цепочке, в соответствии с принадлежностью к инициированному потоку; в-третьих, устанавливают флаг $\rho^{ACTIVE} = 1$.

С точки зрения обработки одного события последовательностные функции можно интерпретировать как конечные автоматы. Событие, поступив на вход последовательност-

ной функции, попадает в некоторое начальное состояние. Из начального состояния в зависимости от особенностей вычислительной процедуры и внешних условий событие попадает в одно из внутренних состояний. Наконец, достигнув конечного состояния, событие передается на выход функции и используется следующими элементами модели.

Детали вычислительного процесса, скрытого последовательностной функцией, зависят от ее типа. Многие функции также предусматривают параметризацию. К основным последовательностным функциям относятся:

- функция-генератор – последовательностная функция, связанная с некоторой верификационной переменной $b \in B$, не имеющая входных и внутренних событий, мгновенно порождающая событие e в выходном множестве с текущим значением верификационной переменной:

$$f_{gen(b\in B)}(t) = [e_0 : \{e_{\leftarrow} = e_0, \begin{cases} e_{\uparrow\rightarrow} = \varepsilon, \rho^{VAL} = b(t) \\ e_{\downarrow\rightarrow} \end{cases}\}]. \quad (4)$$

- функция конъюнктивного конкатенирования – последовательностная функция, имеющая два операнда, одним из которых обязательно является функция-генератор, значение выходных событий которого используется в качестве условия транспортирования цепочек событий из второго операнда:

$$f_{conj}(t, f_1, f_{gen}) : a = \rho^{VAL}(e^0 = first(f_{gen}(t))), \begin{cases} a = 1 \Rightarrow f_{conj}(t, f_1, f_{gen}) = f_1(t); \\ a = 0 \Rightarrow f_{conj}(t, f_1, f_{gen}) = \varnothing; \end{cases} \quad (5)$$

где $first(f(t))$ – первое событие, принадлежащее множеству (в случае операнда-генератора первое событие является единственным);

– функции-очереди, моделирующие временные интервалы;

– функции-репетиции, реализующие циклический анализ булевого выражения или другой последовательностной функции;

– функции, реализующие логические последовательностные операторы, такие как OR, AND, INTERSECT, WITHIN, а также импликации $|\rightarrow$ и $|\Rightarrow$.

Наиболее часто применяемой последовательностной функцией в предлагаемой модели является функция-очередь, лежащая в основе выбранного названия DRTLQ. Очереди применяются для моделирования интервалов между интересующими событиями в системе. Наглядно очередь можно представить в виде цепочки триггеров. На рис. 3 показана схема обработки последовательности событий $\{a;[*2];b\}$ на основе очереди, начинающейся с момента, в котором сигнал $a = 1$, и в прошествии 3 тактовых циклов завершающейся событием $b = 1$.

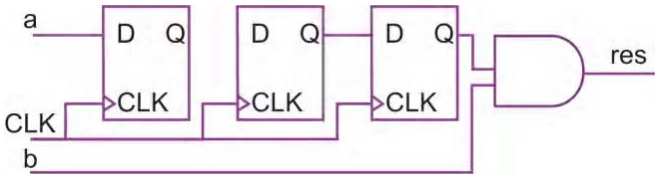

Рис. 3. Наглядная иллюстрация понятия DRTLQ-очередей

В данном случае функция-очередь $f_{\#}$ является операндом функции конъюнктивного конкатенирования f_{conj} с $f_{gen(b)}$, и имеет функцию-генератор $f_{gen(a)}$ на входе $f(t, \{a;[*2];b\}) = f_{conj}(t, f_{\#[3:3]}(t, f_{gen(a)}), f_{gen(b)})$.

В общем виде, функция-очередь, имеющая идентичные конечные интервалы, функционирует путем хранения и продвижения к выходу списка цепочек событий, считанных с входа в различные моменты моделирования, и упорядоченного по времени считывания. Цепочка событий, достигнувшая последней в очереди ячейки, является фактическим результатом функции на следующем такте обработки:

$$f_{\#[N:N]}(t, f_1(t)), N > 0 : \langle r_0(t), \cdots r_{N-1}(t) \rangle; \; r_0(t) = f_1(t);$$

$$\forall i,0<i<N, r_i(t)=r_{i-1}(t-1);\ f_{\#[N:N]}(t,f_1(t))=r_{N-1}(t-1). \quad (6)$$

Задача усложняется, если значения интервалов очереди различаются. Копии событий, время нахождения которых в очереди превысило минимальный заданный интервал, но еще не достигло максимального интервала, направляются на выход. Сами события остаются внутри очереди до момента достижения максимального интервала, и продолжают транспортирование по очереди. Копии событий связывают по правой цепочке:

$$f_{\#[N:M]}(t,f_1(t)),N>0,N<M:\langle r_0(t),\cdots r_{M-1}(t)\rangle;\ r_0(t)=f_1(t);$$

$$\forall i,0<i<M, r_i(t)=r_{i-1}(t-1);\ f_{\#[N:M]}(t,f_1(t))=r_{M-1}(t-1)\cup\bigcup_{i=N-1}^{M-2}\kappa(r_i(t-1)). \quad (7)$$

Если $M\to\infty$, суть вычислительной процедуры (6) не изменяется, однако структура внутренних цепочек событий $\langle r_0(t),\cdots r_{M-1}(t)\rangle$ преобразуется из статической формы в динамическую. Очередь с бесконечным интервалом будет генерировать и продвигать к выходу копии считанного события до тех пор, пока не будет остановлен породивший событие поток активации.

Если $N=0$, копии считанных с входа событий моментально транспортируются к выходу, независимо от обработки события внутри очереди:

$$f_{\#[0:M]}(t,f(t)_1),M>0=f_{\#[1:M]}(t,f_1(t))\cup\kappa(f_1(t)). \quad (8)$$

Функции-репетиции в модели DRTLQ разделяются на булевы и сложные, в зависимости от типа операнда (булево выражение и SERE соответственно). В целом, схема обработки булевых репетиций имеет определенное сходство с обработкой очередей. Дополнительно к сопоставлению времени регистрации в очереди, в репетициях проверяется значение входного булевого выражения. В том случае, если выражение не выполняется, все цепочки событий, в текущий момент хранящиеся в репетиции, поступают на выход со значением 0, и репетиция полностью очищается:

$$f_{[*N]}(t,f_{gen(b\in B)}(t),N>0:\langle v(t),r_0(t),\cdots r_{N-1}(t)\rangle;$$

$$v(t)=\rho^{VAL}(first(f_{gen(b)}(t)));$$

$$\begin{cases} v(t)=1\Rightarrow\begin{cases} r_0(t)=f_{gen(b)}(t);\\ \forall i,0<i<N, r_i(t)=r_{i-1}(t-1);\\ f_{[*N]}(t,f_{gen(b)}(t))=r_{N-1}(t);\end{cases}\\ v(t)=0\Rightarrow\begin{cases} \forall i,k,0\le i<N,0\le k<|r_i(t-1)|,\rho^{VAL}(e_k\in r_i(t-1))\leftarrow 0;\\ f_{[*N]}(t,f_{gen(b)}(t))=\bigcup_{i=0}^{N-1}r_i(t-1);\\ \forall i,0\le i<N, r_i(t)=\varnothing.\end{cases}\end{cases} \quad (9)$$

Аналогично очереди (7), интервальная репетиция требует создания и продвижения к выходу копий цепочек событий, выдержанных внутри репетиции минимальное количество итераций, но не достигших максимума:

$$v(t)=1\Rightarrow f_{[*N:M]}(t,f_{gen(b\in B)}),N>0,N<M=\bigcup_{i=N}^{M-1}\kappa(r_i(t-1)). \quad (10)$$

Случаи $M\to\infty$ и $N=0$ определяются аналогичным очередям образом, разумеется, с учетом процедуры полного очищения репетиции при $v(t)=0$.

Реализация галопирующих репетиций модифицирует базовую репетиционную функцию (6) добавлением элемента (t), который должен обеспечивать буферизацию последнего успешного входного события до момента появления следующего. Соответственно, галопирующая репетиция никогда не выполняет полную очистку, как обычная репетиция:

$$f_{[\to N]}(t, f_{gen(b\in B)}(t)), N > 0 : \langle v(t),\ \ (t), r_0(t), \cdots r_{N-1}(t)\rangle;$$

$$v(t) = \rho^{VAL}(first(f_{gen(b)}(t))); \begin{cases} v(t) = 1 \Rightarrow\ \ (t) = f_{gen(b)}(t); \\ v(t) = 0 \Rightarrow\ \ (t) = \kappa(\ \ (t-1)); \end{cases}$$

$$\begin{cases} r_0(t) = \ \ (t); \\ \forall i, 0 < i < N, r_i(t) = r_{i-1}(t-1); \\ f_{[\to N]}(t, f_{gen(b)}) = r_{N-1}(t). \end{cases} \qquad (11)$$

Неконсекъютивные репетиции расширяют реализацию галопирующих репетиций дополнительным выходным буфером, порождающим копии ранее транспортированных на выход событий до удовлетворения потока:

$$f_{[=N]}(t, f_{gen(b)}(t)) = \kappa(f_{[\to N]}(t-1, f_{gen(b)}(t-1))) \cup f_{[\to N]}(t, f_{gen(b)}). \qquad (12)$$

Наиболее сложными являются функции, реализующие логические операторы над последовательностями. Пусть имеется SERE $s = \{\{a;b\} \| \{c;d\}\} \&\& \{!e[*2]\}$. На рис. 4 в графическом виде представлена DRTLQ-реализация данного SERE, а формула (11) демонстрирует то же SERE в аналитическом виде:

$$f_s(t) = f^1_{\supset}(t);\ f^1_{\supset}(t) = join_{2-\&\&}(t, f^2_{\supset}(t), f_{[*2:2]}(t, f_{gen(\neg e)}(t)));$$

$$f^2_{\supset}(t) = join_{2-\|}\begin{pmatrix} t, \\ f_{conj}(t, f_{\#[1:1]}(t, f_{conj}(t, f^2_{\subset}(t), a)), b), \\ f_{conj}(t, f_{\#[1:1]}(t, f_{conj}(t, f^2_{\subset}(t), c)), d); \end{pmatrix};$$

$$f^2_{\subset}(t) = split_{2-\|}(t, f^1_{\subset}(t));\ \ f^1_{\subset}(t) = split_{2-\&\&}(t, f_{gen(1)}(t)). \qquad (13)$$

Любой логический последовательностный оператор реализуется двумя DRTLQ-функциями: разделителем $f_{\subset}$ (splitter) и соединителем $f_{\supset}$ (joiner). Функция-соединитель является вершиной пути операндов, а начало каждого из операндов исходит из функции-разделителя. Ряд операторов (WITHIN, импликации) могут иметь только два операнда, но степень разветвления логических операций (AND, OR, INTERSECT) не ограничивается. Разделитель всегда имеет единственный вход и множество выходов, соответствующее количеству операндов. Событие, считываемое разделителем с входа, расслаивается на самостоятельные копии, каждая из которых транспортируется по пути одного из операндов. Соединитель имеет единственный выход, порождая выходные цепочки событий в соответствии с реализуемой функцией.

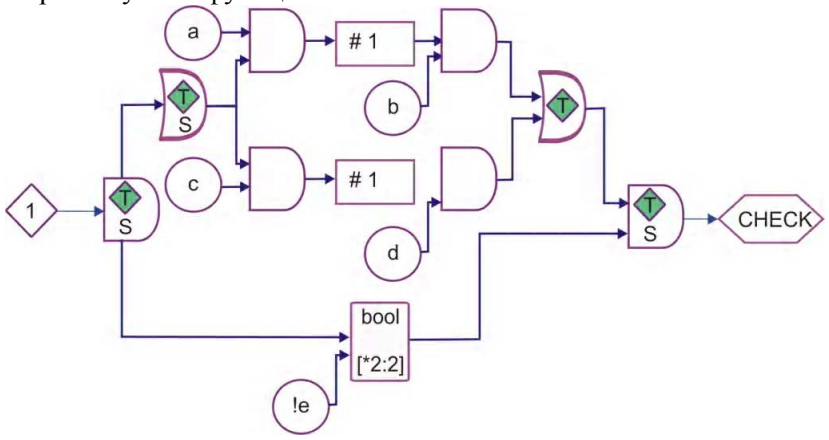

Рис. 4. DRTLQ-реализация последовательности с логическими операторами

Аналогичным образом реализуются репетиции с операндами-последовательностями: вводятся функция-разделитель и функция-соединитель, причем разделитель, определяющий направление транспортирования событий, находится выше по иерархии, чем соедини-

тель, помещаемый на нижнем уровне. В зависимости от количества итераций, пройденных событием в рамках репетиции, событие либо направляется разделителем на выход, либо возвращается к соединителю на очередную итерацию.

2.4. Цикл работы ассерционного процесса. Анализ ассерций в модели DRTLQ осуществляется по событию, доставка которого в ассерционный процесс является обязанностью родительской системы моделирования. Каждая итерация цикла работы всегда соответствует уникальному моменту времени. Ситуация, когда одному моменту времени соответствует более одной итерации анализа, исключается.

Итерации анализа делятся на типы в зависимости от момента запуска:

- регулярная итерация – запускается по тактовому событию (если тактового события не определено, по событию изменения любой из множества верификационных переменных, принадлежащих ассерционному процессу);
- итерация прерывания – запускается при срабатывании условия прерывания (операторы async_abort и sync_abort), прерывает все активные вычислительные потоки путем обработки множеств внутренних событий элементов-очередей и элементов-репетиций;
- финальная итерация – запускается один раз в конце моделирования, транспортирует все цепочки событий непосредственно монитору на обработку с учетом требований условной справедливости (безусловная, слабая, сильная).

Финальная итерация и итерация прерывания предназначаются только для поддержки специальных видов ассерционных языковых конструкций и сводятся к итерированию подготовленных заранее отдельных элементов системы (очередей и репетиций) и обработке множеств их внутренних событий.

Регулярная итерация анализа ассерций включает в себя следующие шаги:

1. Считывание значений всех ассерционных переменных.
2. Запуск шага вычислительной процедуры каждой из последовательностных функций, в соответствии с очередью обработки.
3. Запуск шага обработки ассерционного монитора.

Очередь обработки последовательностных функций упорядочивает запуск всех элементов для осуществления полной обработки модели за один проход. Первыми выполняются функции-генераторы, порождающие события от ассерционных переменных. Далее выполняются все последовательностные функции, в соответствии с иерархией.

Ассерционный монитор непосредственно не контактирует с уровнем последовательностей, а обрабатывает события по запросу. Для обеспечения взаимодействия уровня последовательностей с ассерционным монитором, вершиной иерархии последовательностных функций, т.е. последним элементом в очереди на выполнение, выполняется служебная функция действия f_{ACTION}. Данная функция не имеет ни внутренних, ни выходных событий, поскольку ее единственной задачей является доставка поступившей на вход цепочки событий к ассерционному монитору. Функции действия применяются также для реализации режима глобального времени, однако в этом случае цепочки поступивших событий доставляются не монитору, а темпоральным свойствам.

Пусть $r_{\leftarrow}$ – цепочка событий, транспортированных к монитору с последовательностного уровня. Независимо от типа монитора, сначала проверяется статус активации каждого события: если $\rho^{ACTIVE}(e \in r_{\leftarrow}) \neq 1$, событие уничтожается без каких-либо нотификаций внешней среде. Дальнейшие шаги обработки событий на уровне монитора зависят от его типа.

Предположим, обрабатываемая ассерция моделирует инвариантное LTL-ограничение вида never({...seq...}). В таком случае монитор называется запрещающим, и принцип его работы следующий:

$$\forall e \in r_{\leftarrow}, \begin{cases} \rho^{VAL}(e)=1 \Rightarrow \begin{cases} \text{report(failed);} \\ \forall e' \overset{\text{recursively}}{\in} \{e_{\uparrow\rightarrow}, e_{\downarrow\rightarrow}\}(e), \rho^{RDEAD}(e') \leftarrow 1; \end{cases} \\ \rho^{VAL}(e)=0 \Rightarrow \begin{cases} \rho^{LDEAD}(e) \leftarrow \rho^{RDEAD}(e) \leftarrow 1; \\ e_{\uparrow\rightarrow}(e) = e_{\downarrow\rightarrow}(e) = e \Rightarrow \text{report(passed).} \end{cases} \end{cases} \quad (14)$$

Если значение события $\rho^{VAL}(e)=1$, это означает, что дочерняя последовательность выполнена успешно, а значит, ассерция нарушена. В таком случае весь поток активации ω, соответствующий времени активации t_a рассматриваемого события $e \in r_{\leftarrow}$, признается удовлетворенным, и дальнейший его анализ (могут существовать другие события этого потока) не имеет смысла. Все события, относящиеся к данному потоку, уничтожаются моментально, используя связи события $e_{\uparrow\rightarrow}, e_{\downarrow\rightarrow}$ по правой цепочке. Монитор сигнализирует внешней среде о неудаче анализа ассерции, вызывая абстрактную процедуру report с неудачным статусом.

Если же значение события $\rho^{VAL}(e)=0$, это означает, что один из сценариев последовательности удовлетворяет свойству. В таком случае, событие просто уничтожается из всех цепочек. Если событие было последним в собственном потоке активации, монитор сигнализирует внешней среде об успехе анализа ассерции, на этот раз вызывая процедуру report с положительным статусом.

Работа разрешающего монитора, моделирующего инвариантное ограничение вида always({...seq...}), подобна запрещающему монитору. Различие состоит только в противоположности условия срабатывания: событие с отрицательным значением $\rho^{VAL}(e)=0$ является в данном случае определяющим неудачный статус и досрочное завершение анализа потока активации. Напротив, $\rho^{VAL}(e)=1$ считается успешным результатом, и если такое событие последнее в потоке активации, то ассерция удовлетворяется на данном вычислительном пути.

Также существует третий тип монитора – ожидающий, соответствующий LTL-ограничениям достижимости вида eventually!({...seq...}):

$$\forall e \in r_{\leftarrow}, \begin{cases} \rho^{VAL}(e)=1 \Rightarrow \begin{cases} \text{report(passed);} \\ \text{recursively} \\ \forall e' \in \{e_{\uparrow\rightarrow}, e_{\downarrow\rightarrow}\}(e), \rho^{RDEAD}(e') \leftarrow 1; \end{cases} \\ \rho^{VAL}(e)=0 \Rightarrow \rho^{LDEAD}(e) \leftarrow \rho^{RDEAD}(e) \leftarrow 1. \end{cases} \quad (15)$$

В случае прихода события с положительным значением $\rho^{VAL}(e)=1$ поток активации удовлетворяется, и монитор сигнализирует об успешном статусе анализа. Отрицательный статус $\rho^{VAL}(e)=0$ лишь уничтожает поступившее событие. Никакие действия с потоком активации не происходят, поскольку, в соответствии с [14, формула (26)], в режиме локального времени свойство eventually! не может дать отрицательного результата, за исключением конца моделирования.

2.5. Темпоральные свойства. Темпоральное свойство $p \in P$ в модели DRTLQ – совокупность логических и темпоральных отношений между булевыми выражениями, последовательностными функциями и другими темпоральными свойствами, задающее пространство поведения системы. При помощи темпоральных свойств задаются утверждения о поведении системы на протяжении всего цикла моделирования. Элементы-свойства создаются только в режиме глобального времени и являются промежуточным звеном между последовательностными функциями и ассерционным монитором.

При наличии в модели уровня темпоральных свойств задачи проверки критериев удовлетворения потоков активации (14), (15) выполняются именно этими элементами. Вспомогательные последовательностные функции f_{ACTION} доставляют выходные цепочки событий на первый уровень иерархии темпоральных свойств. С ассерционным монитором взаимодействует только вершина иерархии свойств. Роль самого монитора упрощается и сводится к обработке статуса единственного за весь сеанс моделирования события.

Каждое свойство состоит из следующих структурных элементов:

$$p \in P : \{p_{\uparrow}, P_{\downarrow} \in P\} \times \{\mathrm{INIT}, \mathrm{ABORT}, \mathrm{EVAL}, \mathrm{FINAL}\}, \tag{16}$$

где $p_{\uparrow}$ – родительское свойство; $P_{\downarrow}$ – множество дочерних свойств; INIT – рекурсивная процедура активации вычислительного потока; ABORT – рекурсивная процедура отмены активации всех текущих вычислительных потоков; EVAL – процедура обработки события, FINAL – процедура обработки завершения моделирования.

Тело процедур и дополнительные элементы структуры различаются в зависимости от типа свойства. К основным типам свойств относятся:

– унарные свойства, в свою очередь подразделяющиеся на группы:

– буферные свойства (BUF, NOT);

– глобальные свойства (always , never , eventually!);

– бинарные свойства ($\rightarrow, \leftrightarrow, |\rightarrow, |\Rightarrow,$ until *, before *, next *) – свойства, имеющие по два операнда-свойства;

– коммутативные свойства (&&, ||) – свойства, которые могут иметь неограниченное число равноправных операндов-свойств.

Унарные свойства в качестве операнда могут иметь как другие свойства, так и последовательностную функцию.

Цикл работы ассерционного процесса в режиме глобального времени отличается от простого режима дополнительными шагами по обработке уровня темпоральных свойств:

1. На каждом шаге моделирования осуществляется активация свойств путем вызова процедуры INIT на свойстве верхнего уровня. Активации обеспечивают разрешение создания вычислительных потоков на уровне последовательностей. В отличие от мониторов, всегда разрешающих активацию вычислительных потоков, свойство может блокировать часть дочерних последовательностей в случае необходимости.

2. События, транспортируемые функциями действия f_{ACTION} с последовательностного уровня, приводят к запуску процедуры EVAL у свойств первого уровня. В зависимости от этой процедуры событие может быть либо уничтожено, либо транспортировано к родительскому свойству. Если событие признается удовлетворяющим процедуру EVAL свойства верхнего уровня, оно направляется к монитору, и весь цикл анализа ассерции завершается.

3. Финальная итерация предполагает вызов процедуры FINAL для всех свойств p, принадлежащих модели. Если анализ ассерции не был завершен на момент остановки симуляции, результат определяется именно в этот момент.

Рассмотрим простейшее темпоральное свойство BUF, которое может либо иметь последовательностный вход, создаваемый при продвижении булевого выражения или последовательности на уровень свойств, либо дочернее свойство-операнд:

$$\begin{aligned} p_{\mathrm{BUF}} : \{p_{\uparrow}, \begin{bmatrix} P_{\downarrow} = \varnothing, f_{\mathrm{ACTION}} \in F; \\ P_{\downarrow} : \{p_1 \in P\}; \end{bmatrix}, \Omega_{\mathrm{BUF}} \in \Omega\} \times \\ \times \{\mathrm{INIT}_{\mathrm{BUF}}, \mathrm{ABORT}_{\mathrm{BUF}}, \mathrm{EVAL}_{\mathrm{BUF}}, \mathrm{FINAL}_{\mathrm{BUF}}\}, \end{aligned} \tag{17}$$

где f_{ACTION} – дочерняя последовательностная функция; p_1 – дочернее свойство-операнд; Ω_{BUF} – множество активных вычислительных потоков, порожденных свойством. Очевидно, такое свойство не имеет дочерних свойств.

Процедура $\mathrm{INIT}_{\mathrm{BUF}}$ состоит в расширении Ω_{BUF} новым потоком активации ω_t, соответствующем текущему моменту времени, в который добавляется искусственное событие e_t^0. На уровне последовательностей наличие потока разрешает активацию событий, примыкающих по правой цепочке к e_t^0.

Процедура $\mathrm{ABORT}_{\mathrm{BUF}}$ состоит в очищении множества потоков Ω_{BUF} вместе с уничтожением всех порожденных правых цепочек событий.

Процедура $\mathrm{EVAL_{BUF}}$, вызываемая либо функцией действия $\mathrm{f_{ACTION}}$ с последовательностного уровня, либо дочерним свойством p_1, удаляет из множества Ω_{BUF} поток ω', для которого время активации равно времени активации пришедшего с входа события $t_A(\mathrm{first}(\omega')) = t_A(e)$. Самое событие e направляется в процедуру EVAL родительского свойства.

Процедура $\mathrm{FINAL_{BUF}}$ для всех вычислительных потоков Ω_{BUF} выбирает порожденное при активации событие e^0 и направляет к родительскому свойству: статус определяется режимом условной справедливости.

Аналогичным образом функционирует свойство NOT, имеющее идентичную структуру, а также процедуры активации и ее отмены:

$$p_{NOT} : \{p\uparrow, \begin{bmatrix} P_\downarrow = \varnothing, f_{ACTION} \in F; \\ P_\downarrow : \{p_1 \in P\}; \end{bmatrix}, \Omega_{NOT} \in \Omega\} \times \begin{Bmatrix} INIT_{NOT} = INIT_{BUF}; \\ ABORT_{NOT} = ABORT_{BUF}; \\ EVAL_{NOT}; \\ FINAL_{NOT}. \end{Bmatrix}. \quad (18)$$

Отличие процедур $\mathrm{EVAL_{NOT}}$ и $\mathrm{FINAL_{NOT}}$ от аналогов в BUF состоит в инверсии значения $\rho^{VAL}(e)$, транспортируемого к родительскому свойству.

Рассмотрим работу глобальных темпоральных свойств на примере свойства always:

$$p_G : \{p\uparrow, \begin{bmatrix} P_\downarrow = \varnothing, f_{ACTION} \in F; \\ P_\downarrow : \{p_1 \in P\}; \end{bmatrix}, \Omega_G \in \Omega\} \times \begin{Bmatrix} INIT_G; \\ ABORT_G = ABORT_{BUF}; \\ EVAL_G; \\ FINAL_G; \end{Bmatrix}. \quad (19)$$

Отличие процедуры $\mathrm{INIT_G}$ от $\mathrm{INIT_{BUF}}$ состоит в том, что в случае наличия дочернего свойства-операнда и открытых потоков активации в нем на каждом такте анализа создается дополнительный поток активации, помещаемый в Ω_{BUF}. Такой прием обеспечивает неявное ветвление потоков активации, порождаемых глобальными темпоральными операторами, без фактического ветвления вычислений на уровне дочерних последовательностей.

Процедура $\mathrm{EVAL_G}$ состоит в завершении всех вычислительных потоков $\omega \in \Omega_G$, для которых выполняется соотношение $t_A(\mathrm{first}(\omega)) \le t_A(e)$, и транспортировании соответствующих потокам событий e^0 на уровень выше. Здесь ветвление свойств-операндов воссоздается, однако это не имеет такого влияния на производительность анализа, как если бы производить все вычисления на последовательностном уровне.

Процедура $\mathrm{FINAL_G}$ с успешным статусом завершает все стартовавшие потоки Ω_G. Очевидно, активность свойства always в момент остановки симуляции свидетельствует об успешном выполнении данного свойства.

Свойство never отличается от свойства always инверсией значений $\rho^{VAL}(e)$ транспортируемых событий в процедуре EVAL, а свойство eventually! – инверсией значений в процедуре FINAL.

Коммутативные свойства, такие как AND, имеют следующую структуру:

$$p_\wedge : \{p\uparrow, P_\downarrow : \{p_1, \cdots, p_N\}, \mu(t) : \langle k \in [0; N], v \in [0;1] \rangle\} \times \begin{Bmatrix} INIT_\wedge : \forall i, 0 \le i < N, INIT(p_i); \\ ABORT_\wedge : \forall i, 0 \le i < N, ABORT(p_i) \\ EVAL_\wedge; \\ FINAL_\wedge; \end{Bmatrix}, \quad (20)$$

где N – число операндов; $p_1, \cdots, p_N$ – свойства-операнды, а $\mu(t)$ – функция истории выполнения потока активации, каждый элемент которой соотносится с некоторым такто-

вым циклом и включает информацию о количестве завершенных на текущий момент операндов, а также общий статус завершения.

Работа процедур $\mathrm{INIT}_{\wedge}$ и $\mathrm{ABORT}_{\wedge}$ тривиальна и состоит из вызова аналогичных процедур на всех свойствах-операндах по очереди.

Работа процедуры $\mathrm{EVAL}_{\wedge}$ состоит из двух шагов:

1. Изменение статуса истории потока активации по принципу $v(\mu(t)) \leftarrow v(\mu(t)) \wedge \rho^{\mathrm{VAL}}(e)$ (очевидно, начальное значение $v = 1$). В случае изменения статуса $v(\mu(t-1)) = 1 \rightarrow v(\mu(t)) = 0$ поток активации признается неудачным, и событие e продвигается на уровень выше.

2. Инкремент числа выполненных операндов: $k(\mu(t)) \leftarrow k(\mu(t)) + 1$. Если новое значение $k(\mu(t)) = N$, поток активации признается успешным, и событие e также продвигается на уровень выше.

Работа процедуры $\mathrm{FINAL}_{\wedge}$ заключается в продвижении на уровень выше событий всех потоков активации, для которых имеется хотя бы один незавершенный поток операнда: $k(\mu(t)) < N$, а начальный статус по-прежнему не изменился: $v(\mu(t)) = 1$.

Отличие свойства OR в деталях вычислительных процедур. Во-первых, начальным значением $v(\mu(t))$ является 0. Во-вторых, условие досрочного завершения анализа потока в процедуре $\mathrm{EVAL}_{\vee}$ заключается в переходе $v(\mu(t-1)) = 0 \rightarrow v(\mu(t)) = 1$ и продвигает успешный статус. В-третьих, достижение значения $k(\mu(t)) = N$, при котором $v(\mu(t)) = 0$, завершает поток с неудачным статусом.

Аналогичным способом строится обработка темпоральных свойств $\rightarrow$ и $\leftrightarrow$, с той разницей, что число операндов здесь всегда равно двум.

Бинарное свойство, реализующее семейство операторов until*, имеет более сложную структуру и вычислительные процедуры:

$$\begin{aligned} p_U : \{p_{\uparrow}, P_{\downarrow} : \{p_1, p_2\}, v(t) :\in \{0,1,X\}, \{s_1, s_2, s'_1, s'_2\}(t) :\in \{0,1,X\}\} \times \\ \times \{\mathrm{INIT}_U, \mathrm{ABORT}_U = \{\mathrm{ABORT}(p_1); \mathrm{ABORT}(p_2)\}, \mathrm{EVAL}_U, \mathrm{FINAL}_U\}, \end{aligned} \tag{21}$$

где p_1, p_2 – дочерние свойства-операнды; $v(t)$ – функция текущего статуса вычислительного потока; $s_1(t)$ и $s_2(t)$ – функции фактического состояния потоков, принадлежащих операндам, а $s'_1(t)$ и $s'_2(t)$ – функции интерпретации состояния потоков, принадлежащих операндам.

Процедура INIT_U активирует свойства-операнды и кроме того создает записи для текущего момента t с начальными значениями $s_1(t) = s_2(t) = X$, а также $v(t) = X$.

Работа процедуры EVAL_U состоит из следующих шагов:

1. Обновление состояния операнда, с которого пришло событие: $s_{\left\lceil \begin{smallmatrix}1\\2\end{smallmatrix}\right.}(t) \leftarrow \rho^{\mathrm{VAL}}(e)$.

2. Обновление состояния ранее не разрешенных активированных потоков, начиная со стартовавших одновременно с событием, до текущего времени:

$$\begin{aligned} &\exists k, 0 \le k < (t_{\mathrm{now}} - t_A(e)), \\ s'_2(t_A(e)+k) = &\begin{cases} 0 \Rightarrow \begin{cases} s'_1(t_A(e)+k) = 0 \Rightarrow v(t_A(e)+k); \\ v(t_A(e)+k) \leftarrow X; \end{cases} \\ 1 \Rightarrow \begin{cases} k > 0 \Rightarrow v(t_A(e)+k) \leftarrow s'_1(t_A(e)+k-1); \\ k = 0 \Rightarrow v(t_A(e)+k) \leftarrow 1; \end{cases} \\ X \Rightarrow v(t_A(e)+k) \leftarrow X; \end{cases} \end{aligned} \tag{22}$$

3. Разрешение потоков (отправка события на уровень выше), для которых $v(t)$ на предыдущем шаге претерпело изменение $X \rightarrow 0$ или $X \rightarrow 1$.

В табл. 2 приведены определения функций $s'_1(t)$ и $s'_2(t)$ на основе функций $s_1(t)$ и $s_2(t)$, изменяющихся в зависимости от конкретного оператора.

Таблица 2. Определения бинарных темпоральных операторов

Оператор	$s'_1(t)$	$s'_2(t)$
f_1 until f_2	$s_1(t)$	$s_2(t)$
f_1 until_ f_2	$s_1(t)$	$s_1(t) \wedge s_2(t)$
f_1 before f_2	$\neg s_2(t)$	$s_1(t) \wedge \neg s_2(t)$
f_1 before_ f_2	$\neg s_2(t)$	$s_1(t) \wedge \neg s_2(t)$

Процедура $FINAL_U$ для всех потоков с $v(t) = X$ направляет событие, принадлежащее потоку, на родительский уровень со статусом, определяющимся соотношением $\neg strong(p_U) \wedge s_1(t_\infty)$ для операторов before* и $\neg strong(p_U) \wedge s_2(t_\infty)$ – для операторов until*.

Другие свойства определяются в модели DRTLQ аналогичным образом в соответствии с семантикой LTL-операторов.

3. Выводы

1/ Аналитическая модель верификации HDL-кода с использованием механизма темпоральных ассерций ориентирована на достижение заданной глубины диагностирования и представлена в следующем виде:

$$M = f(F, L, T, C, A, t), C = \{C_1, C_2, ..., C_i, ..., C_m\}; L = \{L_1, L_2, ..., L_i, ..., L_n\};$$

$$A(t) = \{A_1, A_2, ..., A_i, ..., A_k\}; A \subseteq L;\ F = L \times C;\ k \le n; T = \{T_1, T_2, ..., T_i, ..., T_p\}.$$

Здесь C_i – группа операторов кода, нагруженная на вершину L_i (переменная, регистр, счетчик, память) и формирующая ее состояние; F – функциональность, представленная транзакционным графом $F = L \times C$ в виде декартова произведения множества вершин и дуг; A – совокупность темпоральных ассерций, как подмножество вершин транзакционного графа $A \subseteq L$. Метод поиска функциональных нарушений (ФН) блока операторов кода использует предварительно построенную таблицу ФН $B = [B_{ij}]$, где строка есть отношение между тестовым сегментом и подмножеством программных блоков $T_i = (B_{i1}, B_{i2}, ..., B_{ij}, ..., B_{in})$ с возможными ФН. Столбец таблицы формирует отношение между программным блоком и тестовыми сегментами $B_j = (T_{1j}, T_{2j}, ..., T_{ij}, ..., T_{pj})$, которые могут проверять блок с ФН. На стадии моделирования определяется обобщенная реакция $m = \{m_1, m_2, ..., m_i, ..., m_p\}$ механизма ассерций F на тест, путем формирования $m_i = (A_1 \vee A_2 \vee ... \vee A_i \vee ... \vee A_k)$, $A_i = \{0,1\}$ 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 совокупностью векторов B_j с минимальным числом единичных координат

$$B = \min_{j=1,n} [B_j = \sum_{i=1}^{p} (B_{ij} \overset{p}{\underset{i=1}{\oplus}} m_i)],$$

формирующих программные блоки с ФН, проверяемые на тестовых сегментах.

2. Для обеспечения существенно лучшей производительности анализа ассерций и поддержки режима полноценной интерпретации глобальных темпоральных операторов предложена модель динамических регистровых очередей (DRTLQ). Модель ориентирована на высокопроизводительный анализ элементов линейной темпоральной логики и эффективность обнаружения и локализации нарушений. Она состоит из четырех структурных уровней: 1) переменных верификации, обеспечивающих абстракцию выражений булевого уровня; 2) последовательностных функций, предназначенных для реализации регулярных выражений, учитывающих параметр времени; 3) темпоральных свойств и ассерционного монитора, контролирующих потоки активации при анализе формул; 4) ассерционного процесса, реализующего верификацию логически связанной группы темпоральных утверждений. В отличие от модели на основе конечных автоматов количество элементов модели растет линейно относительно размера моделируемой LTL-формулы. Высокий уровень компактности модели достигается за счет минимальной структурной сложности представления простых и интервальных временных сдвигов, циклических ограничений, логических разветвлений – конструкций, порождающих сложные деревья вычислительных процессов и приводящих модель на основе конечных автоматов к нелинейному росту количества состояний.

3. В отличие от конечных автоматов модель DRTLQ способна одновременно осуществлять конвейерный анализ нескольких пересекающихся вычислительных путей. Цикл работы DRTLQ модели обеспечивается пошаговым транспортированием событий от входов модели к выходному ассерционному монитору. Понятие левой цепочки (кольцо транспортирования) обеспечивает линейную сложность продвижения событий в зависимости от количества инициированных потоков и степени их пересечения по ходу анализа. Правая цепочка (кольцо активации) обеспечивает быстрое досрочное завершение вычислительного потока, что особенно эффективно в случае наличия сильных разветвлений.

4. Модель DRTLQ способна эффективно реализовать предложенный режим глобального времени, предполагающий полноценную интерпретацию сложных темпоральных операторов. Событийная модель обработки конструкций-свойств и вычислительные процедуры, контролирующие множества инициированных вычислительных путей, обеспечивают эффективную реализацию всех LTL-операторов с линейной сложностью. Производительность достигается иерархической декомпозицией событийных потоков. Теоретическая древовидность низкоуровневых вычислений, определяемая семантикой LTL, подменяется линейной нагрузкой на последовательностный уровень, а фактический учет сложных ветвлений эффективно реализуется на высоком уровне абстракции. Вычислительная сложность реализации такова, что применение на практике режима глобального времени оказывается существенно быстрее (от 1,25 до 5 раз) по сравнению с традиционным режимом интерпретации.

5. *Научная новизна полученных результатов.* Впервые предложена аналитическая модель верификации, которая характеризуется использованием динамических регистровых очередей для анализа темпоральных ассерций в процессе моделирования тестов для цифровых систем на кристаллах, что обеспечивает высокое быстродействие моделирования и заданную глубину диагностирования ошибок кода.

6. *Практическое значение полученных результатов.* Разработанная аналитическая модель верификации цифровых систем на основе использования динамических регистровых очередей для анализа ассерций линейной темпоральной логики доведена до практической реализации в виде программного обеспечения в составе верификационной системы Riviera (Aldec Inc.). В сочетании с другими реализованными методами верификации, данный программный продукт представляет собой комплексное полноценное решение по верификации System-on-Chip. Продукт занимает лидирующие позиции на мировом рынке программных средств в сфере EDA.

Список литературы: 1. *Clarke E., Grumberg O., Peled D.* Model Checking. 6th edition, MIT Press, 2008. 313p. **2.** Burch J.R., Clarke E.M., Long D.E. Representing circuits more efficiently in symbolic model checking // 28th ACM/IEEE Design Automation Conference, 1991. P. 403-407. 3. *Pnueli A.* The Temporal Logic of Programs // Proceedings of the 18th IEEE Symposium Foundations of Computer Science (FOCS 1977). 1977. P. 46-57. **4.** *Buchi J.R.* On a decision method in restricted second order arithmetic // Proceedings of International Congress on Logic, Methodology and Philosophy of Science, Stanford University Press, 1960. P. 1-11. **5.** *He A., Wu J., Li L.* An Efficient Algorithm for Transforming LTL Formula to Buchi Automaton // International Conference on Intelligent Computation Technology and Automation. 2008. P. 1215-1219. **6.** *Gastin P., Oddoux D.* Fast LTL to Buchi automata translation // 13th International Conference on Computer Aided Verification (CAV 2001), volume 2102 of Lecture Notes in Computer Science. P. 53-65. **7.** *Giannakopoulou D., Lerda F.* From states to transitions: Improving translation of LTL formulae to Buchi automata // 22nd International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002), volume 2529 of Lecture Notes in Computer Science. 2002. P. 308-326. **8.** *Etessami K., Holzmann G. J.* Optimizing Buchi automata // 11th International Conference on Concurrency Theory (CONCUR 2000), volume 1877 of Lecture Notes in Computer Science, 2000. P. 153-167. **9.** *Safra S.* On the complexity of omega-automata // 29th IEEE Symposium on Foundations of Computer Science, 1988. P. 319-327. **10.** *Valmari A.* The state explosion problem // Lectures on Petri Nets I: Basic Models, volume 1491 of Lecture Notes in Computer Science, 1998. P. 429-528. **11.** *Ferndandez J.C., Jard C., Jeron T., Viho A.* Using on-the-fly verification techniques for the generation of test suits // Proceedings of 1996 Workshop of Computer-Aided Verification. 1996. P. 348-359. **12.** *Voeten J.P.M., Van der Putten P.H.A., Geilen M.C.W., Stevens M.P.J.* Formal modelling of reactive hardware/software systems // ProRISC/IEEE'97, Utrecht : STW, Technology Foundation, 1997. P. 663—670. **13.** *Ribeiro O., Fernandes J., Pinto L.* Model Checking Embedded Systems with PROMELA // 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05). 2005. P. 378-385. **14.** *Зайченко С.А.* Модель интерпретации высокоуровневых операторов LTL-логики // АСУ и приборы автоматики. 2009. Вып. 149. С. 96-111.

Поступила в редколлегию 12.10.2009

Зайченко Сергей Александрович, аспирант кафедры АПВТ ХНУРЭ, начальник отдела разработки компании Aldec-Kharkov Ltd. Научные интересы: системы автоматизированного проектирования, моделирования и верификации цифровых систем на кристаллах. Увлечения: литература, музыка, футбол. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. (097)-367-62-93. E-mail: Sergei.Zaychenko@aldec.com

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