КОМПЬЮТЕРНАЯ ИНЖЕНЕРИЯ И ТЕХНИЧЕСКАЯ ДИАГНОСТИКА

УДК 681.326:519.613

СТРУКТУРЫ ДАННЫХ И МОДЕЛИ РЕАЛИЗАЦИИ БАЗОВЫХ ЭЛЕМЕНТОВ ДИНАМИЧЕСКИХ РЕГИСТРОВЫХ ОЧЕРЕДЕЙ

ЗАЙЧЕНКО С.А., ХАХАНОВ В.И., ЧУМАЧЕНКО С.В.

Предлагаются структуры данных и алгоритмы обработки важнейших элементов модели – цепочек событий и функций-очередей. Предлагается модифицированный метод синтеза структур данных регистрового уровня, который отличается аппаратной реализацией операторов расширенной линейной темпоральной логики, что обеспечивает возможность параллельного анализа пересекающихся последовательностей событий при использовании программно-аппаратного HDL-симулятора.

Введение

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

Функция цели формулируется как повышение эффективности процесса проектирования на основе специализации и стандартизации технологических решений, которая определяется поиском минимума среднего значения трех взаимно противоречивых относительных параметров: уровень ошибок проекта L, время верификации T, программно-аппаратная избыточность H:

$$E = F(L,T,H) = \min[\frac{1}{3}(L + T + H)],$$

$$Y = (1 - P)^n;$$

$$L = 1 - Y^{(1-k)} = 1 - (1 - P)^{n(1-k)};$$

$$T = \frac{(1 - k) \times H^s}{H^s + H^a}; \; H = \frac{H^a}{H^s + H^a}.$$

Параметр L, как дополнение к выходу годной продукции Y, зависит от тестопригодности проекта k, вероятности P существования неисправных компонентов и числа необнаруженных ошибок n. Время верификации определяется: тестопригодностью или структурной сложностью программного кода k, умноженной на количество строк функционального кода, отнесенного к общему числу строк проекта. Программно-аппаратная избыточность находится в функциональной зависимости от структурной сложности ассерционного кода, отнесенной к общему числу строк проекта. При этом ассерционная избыточность должна обеспечивать заданную глубину диагностирования ошибок кода функциональности за время time-to-market, определенное заказчиком.

Задачи: 1) Аналитический обзор структур данных регистрового уровня; 2) Усовершенствовать метод синтеза структур данных регистрового уровня для программно-аппаратной реализации операторов расширенной линейной темпоральной логики в HDL-симуляторе.

Источники: моделирование функциональности и ассерций [18-20, 22, 23], верификация [2,3,5,8,9,12], тестирование цифровых проектов [4,15,21], модель DRTLQ [22, 23], ассерционный монитор [1, 6,7,10, 11, 13,14, 16,17].

1. Обеспечение обработки событий

Генерация, активация, транспортирование и уничтожение событий представляют собой динамическую составляющую модели DRTLQ, соединяющую во время выполнения структурные компоненты модели. Совокупность связанных между собой событий формирует поток активации, абстрагирующий в DRTLQ модели вычислительные пути верифицируемой системы. Легковесность структур данных для событий, продуманность организации связей, эффективным способом предоставляющих возможности итерирования, расширения, разбиения, транспортирования цепочек событий – все эти факторы играют немало-

важную роль в обеспечении быстродействия системы анализа темпоральных ассерций в целом.

С различными элементами модели DRTLQ могут быть связаны логические точки наблюдения событий, фиксирующие их прохождение через элемент на текущем шаге моделирования. В качестве таких точек могут выступать входы и выходы последовательностных функций, внутренние контрольные точки очередей, репетиций и темпоральных свойств, а также ассерционный монитор. Доступ к данным и манипуляции с цепочками событий в точках наблюдения осуществляется при посредничестве специальной программной абстракции, называемой контейнером событий. К числу основных задач контейнера относятся контролированное итерирование событий по одной из цепочек, сопровождаемое очисткой от лишних событий, стыковка новых событий, копирование цепочки, слияние цепочек.

Политика уничтожения невостребованных событий в модели DRTLQ построена таким образом, чтобы избежать избыточных операций поиска ссылок на уничтожаемые события. В ходе работы модели события могут уничтожаться по двум типичным сценариям:

1) Происходит удовлетворение вычислительного потока. Такая ситуация может наступить в результате доставки одного из событий, принадлежащему потоку, к контрольной точке – ассерционному монитору или темпоральному свойству – с разрешающим поставленную задачу анализа значением атрибута ρ^{VAL}. В таком случае все порожденные потоком события должны быть уничтожены. Однако другие события потока в этот момент могут транспортироваться в глубине модели по левым односвязным цепочкам. На событие, принадлежащее разрешенному потоку, может существовать ссылка по левой цепочке от события, относящегося к совершенно другому потоку активации. Соответственно удалять такое событие нельзя, и оно должно существовать пока существует левая цепочка, ссылающаяся на него. Вместо удаления на всех событиях устанавливается атрибут $\rho^{\mathrm{RDEAD}} = 1$, свидетельствующий о завершении потока. Этот атрибут учитывается при итерировании цепочек транспортирования, и удаляется только при переходе к событию от события, предыдущего по левой цепочке. Исключение составляет событие с атрибутом $\rho^{\mathrm{RTOP}} = 1$, поскольку гарантируется, что такое событие не транспортируется через последовательностные элементы модели. Ситуация наглядно демонстрируется на рис 1.

2) Последовательностная функция в рамках реализуемой семантики соответствующего ей LTL-оператора принимает решение об уничтожении события. Такое решение может быть принято относительно копий событий, возникших в точках разветвления модели. Например, пусть имеется два события, одно из которых является копией другого, порожденной функцией-разделителем AND. Оба события сходятся в функции-соединителе, на основе атрибутов событий принимается решение о результате вычисления последовательности, и лишь одно из пришедших событий должно продвинуться далее к выходу соединителя. В такой ситуации известна ссылка на событие по левой цепочке, и его можно уничтожить, предварительно исключив из кольца активации, что возможно в любой момент транспортирования в силу двусторонних связей по правой цепочке. Ситуация наглядно показана на рис. 2 (начальная ситуация аналогична рис. 1а).

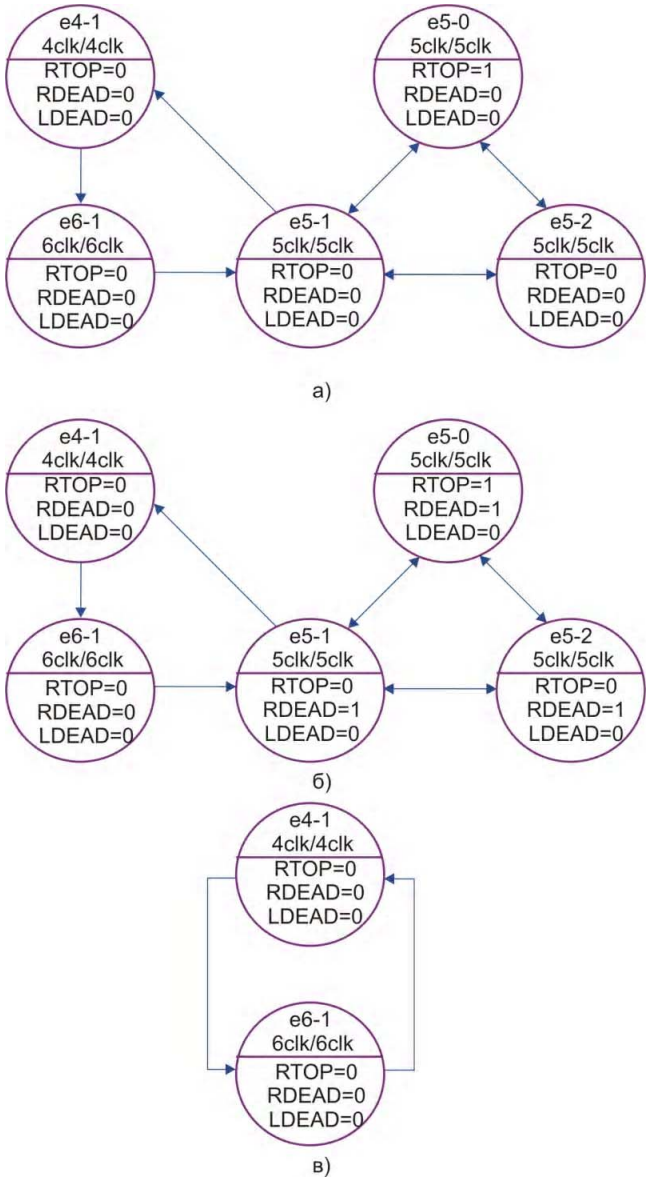

Рис. 1. Уничтожение DRTLQ-событий со стороны потока активации (а –начальная ситуация; б – установка атрибута $\rho^{\mathrm{RDEAD}} = 1$; в – вид после серии итераций)

Итерирование колец событий происходит исключительно под контролем контейнера с целью осуществления автоматической очистки. Итерирование может происходить в прямом направлении по левой цепочке или в одном из направлений по правой цепочке, в зависимости от выбранного типа контейнера. При итерировании по левой цепочке осуществляется отделение от кольца транспортирования событий с установленным атрибутом $\rho^{\mathrm{RDEAD}}(e) = 1$. Аналогично,

при итерировании по правой цепочке происходит отделение от кольца активации событий с атрибутом $\rho^{LDEAD}(e)=1$. Получив гарантию отсутствия ссылок на событие, контейнер уничтожает его в фоновом режиме.

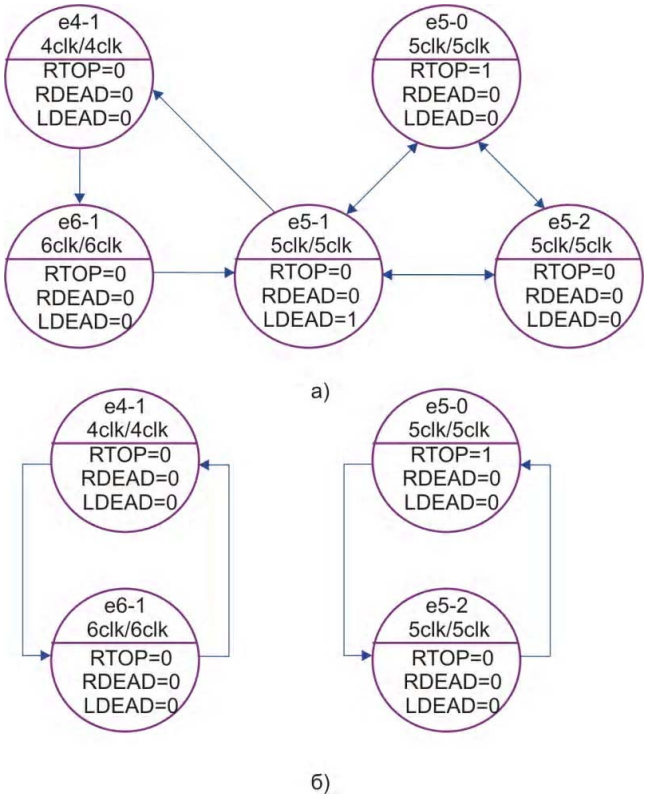

Рис. 2. Уничтожение DRTLQ-событий со стороны левой цепочки (а – установка атрибута $\rho^{LDEAD}=1$; б – вид после окончания операции)

Стоимость всех манипуляций над цепочками событий, включая итерирование с автоматической очисткой, слияние с автоматической элиминацией эквивалентных событий, ничтожно мала по сравнению с вычислительной сложностью решаемых на их основе задач.

2. Структуры данных для реализации функций-очередей. Важнейшей и наиболее часто используемой последовательностной функцией является очередь [23, (6)-(8)], обеспечивающая реализацию временного сдвига, связывающего отношения между сигналами в различных тактовых циклах. Количество элементов-очередей, создаваемых при трансляции ассерций из языкового описания к модели DRTLQ, для среднестатистической LTL-формулы составляет не менее 1/2 от общего количества элементов. Практически каждая применяемая на практике LTL-формула задействует хотя бы одну очередь (что объясняет название модели DRTLQ). Такая частота инстанцирования обуславливает высокие требования к производительности реализации.

Следует отметить существенную разницу между свойствами очередей с конечными и бесконечными интервалами. Время нахождения события в очереди конечной длины определяется границами интервала $[N:M], N \le M, N \ge 0, M > 0$ и не превышает M тактов моделирования в худшем случае. Напротив, время нахождения события в очереди при значении $M \to \infty$ определяется внешними условиями, такими как разрешение потока активации, породившего событие, либо удовлетворение логической последовательностной функции в том случае, когда очередь является частью ответвленного операнда. Помимо временной характеристики, реализация случая с бесконечным интервалом в обязательном порядке требует использования динамических структур данных, накапливающих внутренние цепочки событий очереди, поскольку количество цепочек, которые одновременно могут находиться в рамках очереди невозможно предсказать заранее. Для реализации очередей с конечными значениями параметров N и M представляется возможным ограничиться выделением блока памяти фиксированного размера.

Соответственно, предлагаются две альтернативные модели реализации: табуляционная очередь, реализующая случай конечных интервалов, а также динамическая очередь, обеспечивающая поддержку случая $M \to \infty$. Вторая модель способна обрабатывать оба варианта очередей, однако статические ограничения открывают более рациональный путь для реализации.

Табуляционная модель реализации предполагает выделение статического массива событийных контейнеров для левых цепочек, а также учет позиций табуляции (рис. 3):

$$Q^{\mapsto}_{[N:M]} :< \Pi_0 \cdots \Pi_{M-1}; p_{input}, p_{min}, p_{max} \in [0; M-1] >; \quad (1)$$

где $\Pi_0,...,\Pi_{M-1}$ – фиксированное число контейнеров событий, $p_{input}, p_{min}, p_{max}$ – позиционные переменные табуляции входного контейнера, минимального и максимального выходного контейнеров. При выполнении условия $N=M$ позиции p_{min} и p_{max} равны между собой, и представляется возможным заменить две позиции единственной позицией p_{output}.

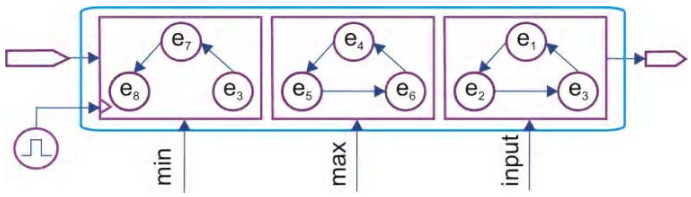

Рис. 3. Табуляционная модель реализации очередей

Основная идея модели (1) состоит в подмене процедуры транспортирования событий из контейнера Π_{i-1} в контейнер Π_i, предусмотренной введенным в [23] требованием продвижения события внутри очереди $\forall i, 0 < i < M, r_i(t) = r_{i-1}(t-1)$, более дешевой схемой косвенной интерпретации индексов. В частности, выражение $\Pi_{p_{input}}$ обозначает контейнер событий для цепочки $r_0(t)$ из [23], а выражения $\Pi_{p_{min}}$ и

$\Pi_{p_{max}}$ – контейнеры для цепочек событий $r_{N-1}(t)$ и $r_{M-1}(t)$ соответственно. Фактически, на каждом шаге моделирования в табуляционной очереди не происходит никакого реального транспортирования цепочек событий между контейнерами, а выполняются циклические сдвиги каждой из позиционных переменных по принципу:

$$p_{next} \leftarrow \begin{cases} p+1, p < (M-1); \\ 0, p = (M-1). \end{cases} \quad (2)$$

Реальное транспортирование происходит лишь для цепочки событий, считываемой со входа очереди. Очевидно, позиции табуляции при циклическом сдвиге никогда не пересекутся, и $\Pi_{p_{input}}$ на любом шаге моделирования будет свободным для размещения новых событий. Также события, находящиеся в контейнерах в интервале между p_{min} и p_{max} транспортируются с копированием (за исключением последнего контейнера $\Pi_{p_{max}}$) для обеспечения результата на выходе очереди согласно принципу $r_{M-1}(t-1) \cup \bigcup_{i=N-1}^{M-2} \kappa(r_i(t-1))$ из [23].

Ниже представлена оценка вычислительной сложности одного шага моделирования табуляционной очереди относительно параметров N и M:

$$t: Q^{\mapsto}_{[N:M]}(N,M) = t_{input} + t_{shift} + t_{output};$$

$$t_{input}(N,M) = t_{\leftarrow} \times \left|\Pi_{p_{input}}\right| = O(1);$$

$$t_{sbift}(N,M) = O(1);$$

$$t_{output}(N,M) = (M-N) \times t_{\leftarrow} \times \overline{|\Pi_i|}, \quad i \in [N-1:M-1] = O(1); \quad (3)$$

где t_{input} – время считывания входного множества, t_{shift} – время сдвига внутренних событий (сводится к 0), t_{output} – время формирования выходного множества событий, $t_{\leftarrow}$ – время транспортирования одного события из контейнера в другой контейнер, $\left|\Pi_{p_{input}}\right|$ – размер входного множества событий, $\overline{|\Pi_i|}$ – средний размер множеств событий, направляемых на выход. Из (3) очевидна константная зависимость производительности табуляционной очереди от параметров структурной сложности, и линейная зависимость от числа транспортируемых через очередь событий. Требования к памяти растут линейно относительно значения параметра M.

Модель динамической очереди представляет собой односвязный список блоков B (рис. 4), включающих контейнер для хранения событий Π_i и время считывания контейнера со входа очереди t_{read}:

$$Q^{\infty}_{[N:..]}: \langle B: \{b_0..b_i..\} \rangle, \forall b \in B: \langle \Pi, t_{read} \rangle. \quad (4)$$

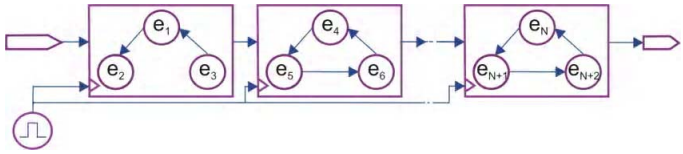

Рис. 4. Динамическая модель реализации очередей

На каждом шаге для хранения вновь поступивших на вход очереди событий выделяется новый блок b, помещаемый в начало. Далее для формирования выходной цепочки событий осуществляется итерирование списка. Достижение блоком минимального времени ожидания достигается проверкой $t - t_{read} \geq N$, и события всех блоков, начиная с первого, для которого данный критерий успешно выполнился, объединяются в результирующее выходное множество. Блок уничтожается, если цепочка принадлежащих ему событий становится пустой (в результате учета правил элиминации невостребованных событий).

Время выполнения одной итерации анализа динамической очереди, также как и требования к памяти, не зависит от значения параметра N, а определяются линейно относительно числа накопленных очередью блоков. Чем дольше событие находится внутри очереди в ожидании внешнего условия разрешения, тем медленнее функционирует сама очередь. Однако влияние данного фактора на производительность нельзя оценить статически, поскольку время ожидания определяется выполняемым тестом.

3. Расширения событий

Реализация многих последовательностных функций тесно связана с понятием расширений событий. Расширение события представляет собой цепочку связанных блоков вспомогательных информационных блоков, прикрепляемых к событию, предназначенных для внутренних целей реализации конкретной последовательностной функции, обрабатывающей событие в данный момент. Расширения могут быть иерархическими, поскольку в модели допускается вложенность логических функций (рис. 5).

Используются различные виды расширений для логических последовательностных функций, для последовательностных импликаций, репетиций. Рассмотрим структуру типичного событийного расширения, реа-

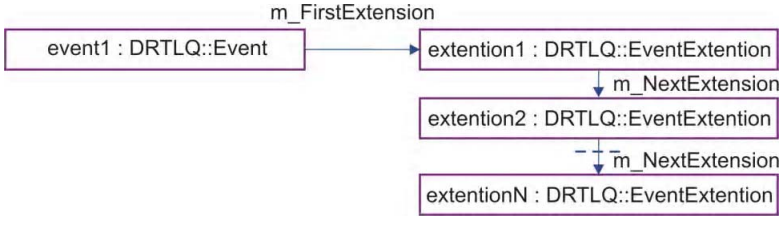

Рис. 5. Иерархические расширения событий в модели DRTLQ

лизующего логические функции AND, OR, INTERSECT:

$$X_L :< X_{NEXT}, t_{\subset}, \rho = \{\rho^{VAL}; \rho^{RESOLVED}\}, N_T, N_C >, \quad (5)$$

где X_L – событийное расширение для логических функций, X_{NEXT} – следующее прикрепленное расширение, $t_{\subset}$ – время создания расширения, ρ – характеристический вектор расширения, состоящий из атрибутов ρ^{VAL} (результирующее значение функции), $\rho^{RESOLVED}$ (индикатор разрешения группы), N_T – общее число разветвлений, порожденных функцией, N_C – число разветвлений с завершенным анализом.

Задачей элемента-разделителя является ассоциация поступивших на вход цепочек событий с обрабатываемой логической функцией. Ассоциация заключается в прикреплении к событийным структурам данных расширения (5), которое используется далее элементом-соединителем для определения результата вычисления. Одному входному событию ставится в соответствие единственное расширение, и каждое порожденная разделителем и транспортируемая далее по путям-операндам копия событий ассоциируется с одним и тем же расширением (рис. 6). Разделитель создает $N_T - 1$ копий поступившего на вход события, поскольку оригинальное событие также продолжает транспортирования по ветке одного из операндов. Совокупность оригинального события, всех событий, порожденных от него разделителем, а также прикрепленного расширения называются последовательностной группой.

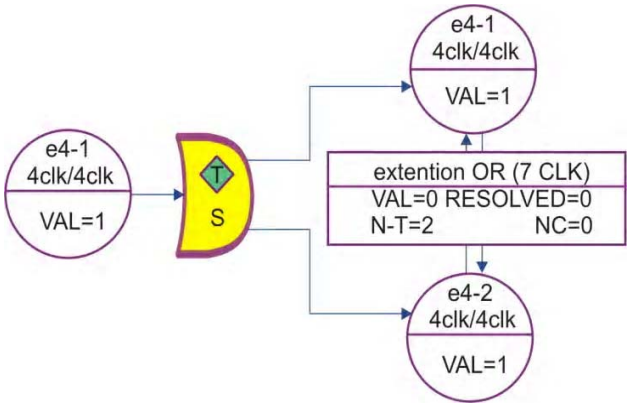

Рис. 6. Ассоциация событий с расширением при транспортировании через элемент-разделитель функции логического OR

Лишь одно событие из группы может достигнуть выхода элемента-соединителя, и его результирующее значение ρ^{VAL} будет определяться значением ρ^{VAL} от расширения. Начальные значения атрибута ρ^{VAL} расширения отличаются в зависимости от реализуемой функции: $\rho^{VAL} = 1$ для функций AND и INTERSECT, но $\rho^{VAL} = 0$ для функции OR. Эта особенность определяется алгоритмом работы соответствующих элементов-соединителей.

В частности, для всех событий e, транспортируемых через пару разделитель-соединитель функции логического OR, с прикрепленным к ним расширениям X_L, обработка соединителем состоит из следующих шагов:

$$1)\ N_C(X_L) \leftarrow N_C(X_L) + 1;$$

$$2)\ \rho^{RESOLVED}(X_L) = 0 \Rightarrow$$

$$\Rightarrow \begin{cases} \left.\begin{cases} \rho^{VAL}(e) = 0 \\ N_T(X_L) = N_C(X_L) \end{cases}\right\} \Rightarrow \begin{cases} \rho^{RESOLVED}(X_L) \leftarrow 1; \\ \rho^{VAL}(X_L) \leftarrow 0; \\ \text{propagate e;} \end{cases} \\ \rho^{VAL}(e) = 1 \Rightarrow \begin{cases} \rho^{RESOLVED}(X_L) \leftarrow 1; \\ \rho^{VAL}(X_L) \leftarrow 1; \\ \text{propagate e;} \end{cases} \end{cases}.$$

$$3)\ \left.\begin{cases} N_C(X_L) = N_C(X_L) \\ \rho^{RESOLVED}(X_L) = 1 \end{cases}\right\} \Rightarrow \text{eliminate } X_L. \quad (6)$$

Группа OR может быть разрешена в двух случаях: либо один из операндов завершается успешно, либо неудачно завершается каждый из операндов. Событие e, разрешающее группу, транспортируется к выходу соединителя незамедлительно, при этом от него отделяется расширение X_L. В том случае, если группа разрешается досрочно, соединитель дожидается завершения последнего операнда, а затем уничтожает расширение. Все поступающие после момента разрешения группы события не влияют на результат и уничтожаются.

Подобным образом осуществляется обработка групп AND с различием от (6) в условиях разрешения группы и инверсии результатов на шаге 2:

$$\rho^{RESOLVED}(X_L) = 0 \Rightarrow$$

$$\Rightarrow \begin{cases} \left.\begin{cases} \rho^{VAL}(e) = 1 \\ N_T(X_L) = N_C(X_L) \end{cases}\right\} \Rightarrow \begin{cases} \rho^{RESOLVED}(X_L) \leftarrow 1; \\ \rho^{VAL}(X_L) \leftarrow 1; \\ \text{propagate e;} \end{cases} \\ \rho^{VAL}(e) = 0 \Rightarrow \begin{cases} \rho^{RESOLVED}(X_L) \leftarrow 1; \\ \rho^{VAL}(X_L) \leftarrow 0; \\ \text{propagate e;} \end{cases} \end{cases}. \quad (7)$$

Группа AND разрешается с неудачным статусом первым же событием с $\rho^{VAL}(e)=0$, в то время как успешное разрешение требует получения $\rho^{VAL}(e)=1$ от каждого из N_T операндов соединителя.

Более сложный алгоритм необходим для групп INTERSECT. Условия разрешения (7) дополняются ограничениями на одновременность прибытия всех событий группы к элементу-соединителю. На первом шаге работы соединителя-INTERSECT, считываются цепочки событий со всех входов соединителя, и формируются специальные множества прибывших на данном шаге событий $r(X_L)$, каждое из которых соответствует одной из групп:

$$r(X_L)=\{e_0,\cdots e_k\}\Rightarrow X_L(e)=X_L. \qquad (8)$$

Далее к каждому X_L, для которого $r(X_L)\neq\varnothing$, применяется следующая логика:

$$\left.\begin{cases}\bigwedge\limits_{k=0}^{|r(X_L)|}(\rho^{VAL}(e_k\in r(X_L))=1;\\ |r(X_L)|=N_T(X_L)\end{cases}\right\}\Rightarrow$$
$$\Rightarrow\begin{cases}\rho^{VAL}(X_L)\leftarrow 1;\\ \rho^{RESOLVED}(X_L)\leftarrow 1.\end{cases} \qquad (9)$$

Все другие случаи, например, неудача одного из потоков-операндов, или $|r(X_L)|\neq N_T(X_L)$ – несоблюдение одновременности прибытия – также приводят к моментальному разрешению группы, но с неудачным статусом. Очевидно, группа INTERSECT будет разрешена с тем или иным результирующим статусом в том случае, если на текущем тактовом цикле на элемент-соединитель поступило хотя бы одно из событий, принадлежащих группе.

Если во время подготовки модели представляется возможным статически определить невозможность одновременности завершения операндов, например, $\{a;b\}\,\&\&\,\{c;d;e\}$, имеет смысл заменить данную модель комбинацией очереди и конъюнктивной конкатенации с константой 0, поскольку данная замена эквивалентна с точки зрения выходных множеств событий, но при этом характеризуется значительно более высокой производительностью:

$$f_{conj}(t,f_{\#[2:2]}(t,f_{gen(a)}(t)),f_{gen(0)}(t)). \qquad (10)$$

Очевидно, отсутствие циклических вычислений обеспечивает линейную вычислительную сложность данной реализации логических функций.

Принципиально другие событийные расширения необходимы для поддержки последовательностного оператора within :

$$X_{WITHIN}:<X_{NEXT},t_{\subset},N_E,m\in[0:1]>; \qquad (11)$$

где X_{WITHIN} – прикрепляемое разделителем-WITHIN событийное расширение, N_E – число событий, относящихся к расширению, m – индикатор главного пути (если $m=1$, путь является главным и соответствует правому операнду, если же $m=0$, путь является подчиненным и соотносится с левым операндом). Число N_E увеличивается при копировании события в ходе процесса транспортирования, и уменьшается при уничтожении невостребованных копий. События подчиненного и главного путей получают различные расширения.

Роль разделителя в реализации оператора WITHIN, помимо расслоения события на два пути и прикрепления расширений, состоит, в соответствии с требованиями семантики [23], в повторной генерации копий событий подчиненного пути на каждом шаге анализа до момента разрешения группы:

$$f^{slave}_{\subset-WITHIN}(t,f_1(t))=\bigcup_{k=0}^{j}\kappa(f_1(t-k)),j\le t, \qquad (12)$$

где $f^{slave}_{\subset-WITHIN}$ – множество событий на подчиненном пути за разделителем WITHIN, f_1 – последовательностная функция, стоящая перед разделителем, j – число тактов, прошедших с момента разделения события.

Соединитель-WITHIN состоит из двух операндов $f_{slave}(t)$ и $f_{main}(t)$, табличной функции $\mu(t)$, возвращающей количество ожидаемых групп главного пути для каждого момента разделения, и множества S, сохраняющего список успешно завершенных активаций подчиненного пути:

$$join_{within}:$$
$$<f_{slave}(t),f_{main}(t),\mu(t),S:\{t_1,\cdots,t_N\}>. \qquad (13)$$

Работа соединителя на каждой итерации состоит из двух шагов:

1. Обработка всех событий, поступивших на вход с подчиненного пути. Момент времени $t_{\subset}$, относящийся к расширению события $e\in f_{slave}(t)$ с успешным значением $\rho^{VAL}(e)=1$, заносится во множество S, что автоматически отключает дальнейшую регенерацию события подчиненного пути элементом-разделителем. Автоматически уменьшается значение компонента N_E, но из-за процесса регенерации в элементе-разделителе, число N_E не может достигнуть значения 0 до остановки регенерации.

2. Обработка всех событий, поступивших на вход с главного пути. Для каждого $e\in f_{main}(t)$ выполняется разрешение группы, в состав которой оно входит. Если событие имеет успешный статус, группа однозначно разрешается. Если же событие имеет неудачный статус, группа разрешается только при выполне-

нии условия $N_E = 1$. Значение результирующего выходного события e', соответствующего группе главного пути, определяется как статусом события e, так и наличием успешно завершенных подчиненных групп:

$$\rho^{VAL}(e') \leftarrow \rho^{VAL}(e) \wedge (\exists t_i, t_i \in S, t_i \le t_{\subset}(e)), \quad (14)$$

Вычислительная сложность обработки оператора WITHIN в модели DRTLQ определяется разницей числа тактов анализа последовательностей-операндов. Чем длиннее последовательность главного операнда по сравнению с последовательностью дочернего операнда, тем больше событий будет регенерировано разделителем на дочернем пути. Пусть T_{MAIN} и T_{SLAVE} представляют собой число тактов, необходимых для анализа последовательностей-операндов. Вычислительная сложность обработки одной активации в целом определяется числом действий по транспортированию индивидуальных событий, которые необходимо произвести по главному и подчиненному путям:

$$N_{TOTAL} = N_{MAIN} + N_{SLAVE} = \\ = \overline{|f_1(t)|} \times (T_{MAIN} + \sum_{i=0}^{T_{MAIN} - T_{SLAVE}} (T_{SLAVE} - i)), \quad (15)$$

где N_{TOTAL} – общее число действий по транспортированию событий, N_{MAIN} и N_{SLAVE} – число действий на главном и подчиненном пути соответственно, а $\overline{|f_1(t)|}$ – среднее число событий, поступающих на вход разделителя на каждом шаге. Число N_{SLAVE} определяется необходимостью регенерации событий на подчиненном пути, что в худшем случае $\max((T_{MAIN} - T_{SLAVE}) \times T_{SLAVE})$ дает квадратичную сложность от длины операндов.

Реализация репетиций также предполагает использование событийных расширений, задающих число выполненных итераций:

$$X_{[*N:M]} : \langle X_{NEXT}, K_I \rangle, \quad (16)$$

где $X_{[*]}$ – расширение, прикрепляемое к событиям функциями-репетициями, K_I – число итераций, в течение которого событие уже содержится во внутренних множествах репетиции.

В случае булевой репетиции отсутствует пара разделитель-соединитель, и имеется единственный элемент с одним внутренним событийным контейнером (не считая дополнительных буферов на входе и на выходе в ряде вариантов). На каждом шаге работы у расширений (16) увеличивается счетчик K_I, затем проверяется достижение минимального числа итераций. Если репетиция не является интервальной, то от события отделяется расширение $X_{[*]}$, и само событие направляется на выход. Если же репетиция предполагает интервальное число итераций, то для всех событий, число K_I которых превысило минимум, но еще не достигло максимума, создается копия без расширения (16), направляемая на выход, а само событие остается внутри репетиции.

Репетиции с операндами-последовательностями, например $\{a;[*1];b\}[*3]$, используют идентичные (16) расширения для учета числа итераций. Такая функция содержит элемент-разделитель (loop) и элемент-соединитель (join), но, в отличие от логических функций, они располагаются в противоположном порядке (рис. 7). События могут возвращаться по обратной связи.

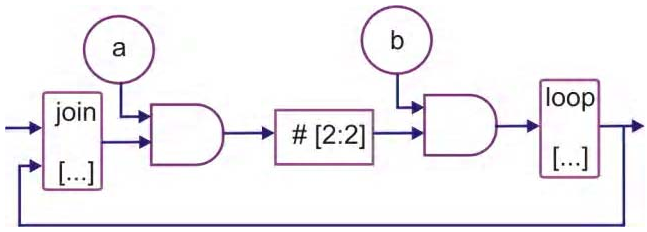

Рис. 7. Реализация репетиции с операндом-последовательностью

Вычислительная сложность репетиций с конечными границами интервала линейна относительно параметров N и M. При больших интервалах число событий во внутреннем множестве репетиции растет, однако время анализа одного события остается неизменным. Репетиции с $M \to \infty$ при длительном нахождении события в репетиции порождают большое количество копий. Это приводит к квадратичному росту количества транспортируемых событий, в зависимости от интервала времени между поступлением события на вход репетиции и до момента разрешения потока активации.

4. Выводы

Одним из ключевых факторов обеспечения производительности модели DRTLQ являются предложенные структуры данных и алгоритмы обработки важнейших элементов модели – цепочек событий и функций-очередей. Быстродействие основной динамической составляющей модели DRTLQ – транспортирования событий – определяется легковесностью событийных структур данных, продуманной организацией основных и дополнительных связей между структурными элементами модели, улучшающими параметры транспортирования, схемой отложенного уничтожения событий при разрешении потоков активации. Реализация элементов-очередей, являющихся наиболее распространенными элементами в модели DRTLQ, основывается на подмене фактического транспортирования событий логической интерпретацией, и, за исключением очередей с бесконечными интервалами, характеризуется константной вычислительной сложностью в зависимости от интервалов ожидания.

Научная новизна. Усовершенствован метод синтеза структур данных регистрового уровня, который отли-

чается аппаратной реализацией операторов расширенной линейной темпоральной логики, что обеспечивает возможность параллельного анализа пересекающихся последовательностей событий при использовании программно-аппаратного HDL-симулятора.

Практическая значимость. Интеграция ассерционных моделей и модифицированных структур данных с программным продуктом Riviera позволила существенно (20–80%) сократить временные затраты для моделирования функциональности и ассерций в процессе тестирования цифровых проектов.

Литература: 1. *Foster H., Krolnik A., Lacey D.* Assertions-based Design. Kluwer Academic Publishers, 2003, 392p. **2.** Assertion-Based Verification // Synopsys Inc., Technical Report, May 2003, 14p. **3.** *Yeung P.* The Four Pillars of Assertions-Based Verification // Mentor Graphics Corporation, EuroDesignCon 2004, 21p. **4.** *Havlicek J., Y. Wolfsthal.* PSL and SVA: two standard assertion languages addressing complementary engineering needs // Design and Verification Conference and Exhibition, 2005. pp. 1-7. **5.** *Nandi A., Pal B., Chhetan N., Dasgupta P., Chakrabarti P.P.* HDBUG: A high-level debugging framework for protocol verification using assertions // IEEE Indicon Conference, Chennai, India, Dec. 2005, pp. 115–118. **6.** *Oliveira M., Hu A.* High-Level Specification and Automatic Generation of IP Interface Monitors // Design Automation Conference, 2002, pp. 129-134. **7.** *Morin-Allory K., Fesquet L., Borrione D.* Asynchronous assertion monitors for multi-clock domain system verification // 17th IEEE International Workshop on Rapid System Prototyping (RSP'06) , pp. 98-102. **8.** *Dahan A., Geist D., Gluhovsky L., Pidan D., Shapir G., Wolfsthal Y., Benalycherif L., Kamdem R., Lahbib Y.* Combining System Level Modeling with Assertion Based Verification // Sixth International Symposium on Quality of Electronic Design (ISQED'05), pp. 310-315. **9.** *Habibi A., Tahar S., Samarah A., Donglin Li, Mohamed O.A.* Efficient Assertion Based Verification using TLM. DATE'2006. PP. 33-38. **10.** *Ecker W., Esen V., Hull M.* Implementation of transaction-level assertion framework in SystemC // Design, Automation & Test in Europe Conference & Exhibition (DATE'07). 167p. **11.** *Habibi.A., Tahar S.* On the extension of SystemC by SystemVerilog assertions // Canadian Conference on Electrical & Computer Engineering, vol. 4, Niagara Falls, Ontario, Canada, May 2004. PP. 1869-1872. **12.** *Bombieri N., Fummi F., Pravadelli G., Fedeli A.* Hybrid Incremental Assertion-Based Verification for TLM design flows // IEEE Design and Test of Computers, March 2007. PP. 140-152. **13.** *Joshi M., Donovan K.* Assertion-Based Acceleration // Cadence Design Systems, Technical Report, June 2005. 6 p. **14.** *Wang S.* Assertions-based emulation methodology // Design & Reuse Electronic Journal, 2004, http://www.design-reuse.com/articles/4951/assertion-based-emulation-methodology.html. **15.** *Borrione D., Liu M., Ostier P., Fesquet L.* PSL-based online monitoring of digital systems // In Advances in Design and Specification Languages for SoCs - Selected Contributions from FDL'05. Springer, 2006. PP. 5-22. **16.** *Kakoee M.R., Riazati M., Mohammadi S.* Enhancing the testability of RTL designs using efficiently synthesized assertions // 9th International Symposium on Quality Electronic Design (ISQED 2008). PP. 230-235. **17.** *Das S., Mohanty R., Dasgupta P., Chakrabarti P.* Synthesis of SystemVerilog assertions // Proc. of the Design Automation & Test in Europe Conference (DATE'2006), Vol. 2. PP. 16. **18.** *Venkatasubramanian R., Hayes J.P., Murray B.T.* Low-Cost On-Line Fault Detection Using Control Flow Assertions // On-Line Testing Symposium, Greece, 2003, pp. 137-143. **19.** *Morin-Allory K., Borrione D.* A proof of correctness for the construction of property monitors // IEEE Intl. High Level Design Validation and Test Workshop, Dec. 2005, pp. 237–244. **20.** *Pinter G., Majzik I.* Automatic generation of executable assertions for run-time checking of temporal requirements // Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05). pp.111-120. **21.** *Ruf J., Hoffman D.W., Kropf T., Rosenstiel W.* Checking temporal properties under simulation of executable system description // Proc. of the International High Level Design Validation and Test Workshop, 2000, pp. 161-166. **22.** *Зайченко С.А., Литвинова Е.И., Побеженко И.А.* Модель интерпретации высокоуровневых операторов LTL-логики // АСУ и приборы автоматики. 2009. Вып. 149. С.96-111. **23.** *Зайченко С.А., Чумаченко С.В.* DRTLQ-модельдля функциональной верификации цифровых систем на основе линейной темпоральной логики // АСУ и приборы автоматики. 2010. Вып. 150. С. 33-48.

Поступила в редколлегию 27.12.2009

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

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

Хаханов Владимир Иванович, д-р техн. наук, профессор, профессор кафедры АПВТ ХНУРЭ, декан факультета компьютерной инженерии и управления ХНУРЭ. Научные интересы: техническая диагностика вычислительных устройств, систем, сетей и программных продуктов. Увлечения: баскетбол, футбол, горные лыжи. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 702-13-26. E-mail: hahanov@kture.kharkov.ua

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