УДК 681.326:519.613

С.А. ЗАЙЧЕНКО, Е.И. ЛИТВИНОВА, И.А. ПОБЕЖЕНКО

МОДЕЛЬ ИНТЕРПРЕТАЦИИ ВЫСОКОУРОВНЕВЫХ ОПЕРАТОРОВ LTL-ЛОГИКИ

На основе анализа математического аппарата линейной темпоральной логики в применении к верификации цифровых систем разрабатывается модель интерпретации высокоуровневых операторов LTL-логики для проверки справедливости формул во время симуляции – режим глобального времени.

1. Введение

Актуальность исследования. Наиболее сложной и дорогостоящей составляющей современного цикла проектирования систем на кристаллах (SoC – System-on-Chip) является функциональная верификация. Под функциональной верификацией понимают процесс поиска, обнаружения и устранения ошибок в модели системы, приводящих к нарушению спецификации. Согласно мнению экспертов индустрии автоматизации проектирования электроники (EDA – Electronic Design Automation), при использовании архитектуры ASIC (Application Specific Integrated Circuit) доля верификации в проектных затратах превышает 70%. Такая высокая стоимость достижения качества системы определяется многими аспектами:

– неизбежным большим количеством ошибок и недоработок, допускаемых инженерами в модели системы, в тестах, а также непосредственно в спецификации системы;

– несовершенством средств диагностирования в системах автоматизации, затрудняющим локализацию и устранение причины возникновения ошибки;

– недостаточной производительностью и точностью программных систем автоматизации, повышение качества которых растет в несколько раз медленнее увеличения сложности обрабатываемых моделей.

Комплексное решение перечисленных проблем, способное в значительной степени снизить затраты на верификацию систем на кристалле, есть ключевая настоящая задача индустрии EDA. Согласно материалам публикаций ведущих мировых компаний данной сферы (Cadence Design Systems, Synopsys Inc., Mentor Graphics Corporation, IBM, Intel, Sun Microsystems, Cisco Systems Inc., Atrenta, Aldec Inc.), существенные их усилия сосредоточены на выработке эффективных комплексных методов верификации систем, способных:

– в несколько раз снизить вероятность возникновения ошибок за счет уменьшения участия человека в рутинных элементах процесса;

– обеспечить обнаружение и диагностирование абсолютного большинства допущенных неточностей на ранних фазах проектирования для сокращения времени устранения возникшей проблемы;

– на порядок улучшить производительность и надежность систем автоматизации верификации за счет повышения уровня абстракции как самих моделей, так и тестовых воздействий.

Цель исследования – существенное уменьшение стоимости функциональной верификации систем на кристаллах при помощи новых моделей и методов проверки темпоральных ассерций в рамках процесса анализа поведения системы под влиянием тестовых воздействий.

Для достижения цели необходимо решить следующие *задачи*: 1. Анализ и состояние проблемы в части описания математического аппарата линейной темпоральной логики в применении к верификации цифровых систем; формальной семантики языковых стандартов описания ассерций (распространенные языки описания ассерций и их свойства). 2. Анализ режимов интерпретации локального и глобального времени. 3. Разработка модели интерпретации высокоуровневых операторов LTL-логики для проверки справедливости формул во время симуляции – режим глобального времени.

Объект исследования – проектируемая цифровая система, содержащая миллионы вентилей, реализуемая в кристаллах программируемой логики, представленная на HDL-языках описания аппаратуры, а также программная среда и набор тестов, обеспечивающих комплексную функциональную верификацию модели проектируемой системы.

Предмет исследования – модели и методы функциональной верификации проектируемой цифровой системы, использующие линейную темпоральную логику и языки описания ассерций, а также программные средства, осуществляющие процесс функциональной верификации методом имитационного моделирования.

2. Математический аппарат линейной темпоральной логики в применении к верификации цифровых систем

2.1. Моделирование параллельных систем и ограничений. Любая система на кристалле является параллельной (concurrent). В общем случае под параллельными понимаются программно-аппаратные системы с несколькими параллельными процессами, которые непрерывно обмениваются сообщениями с внешней средой. Для таких цифровых систем характерны следующие признаки:

параллельность – компоненты системы могут одновременно выполнять различные задачи и периодически обмениваться друг с другом синхронизирующими сообщениями;

недетерминированность – следующее состояние системы либо неизвестно, либо частично известно;

бесконечность существования – в отличие от функциональных систем, параллельная система не обязательно имеет конечное состояние.

Большинство цифровых аппаратных систем имеют конечное число состояний, определяющееся набором достижимых значений всех сигналов. В отличие от аппаратных систем, большинство программных систем, где применяются динамические структуры данных, имеют бесконечное число состояний, что значительно усложняет процессы их верификации.

Задача верификации параллельных систем состоит в доказательстве наличия или отсутствия в их работе определенных последовательностей состояний, называемых ограничениями. Если число состояний конечно, представляется возможным формальное доказательство удовлетворения системой заданных ограничений.

Очевидно, анализ работы параллельных систем также как и анализ ограничений предполагают формальные рассуждения об изменении состояния системы с течением времени. Формальное утверждение о том, что система всегда или в выбранном интервале времени удовлетворяет заданному логико-временному ограничению последовательностей состояний, называется темпоральным утверждением или темпоральной ассерцией. Необходимы формальные механизмы для моделирования параллельных систем и механизмы описания ассерций.

В целях верификации модели параллельных систем удобно представлять в виде структур Крипке [1]:

$$M = (\Sigma : 2^D \cup \{true, false\}, S, S_0, R, L), \quad (1)$$

где M – моделируемая система; Σ – алфавит системы, или множество всех атомарных высказываний (все возможные значения переменных системы D); S – множество состояний системы; $S_0 \subseteq S$ – начальное состояние системы; $R \subseteq S \times S$ – функция переходов между каждым из состояний; $L : S \times \sum \to \{0,1\}$ – функция, называемая оценкой, сопоставляющая каждое из состояний с подмножеством атомарных высказываний, справедливым для данного состояния. Выражение вида $L(s \in S, p \in \sum) = 1$ означает, что атомарное высказывание p истинно в состоянии s, а выражение вида $L(s \in S, p \in \sum) = 0$ означает противоположное.

Каждой реальной вычислительной процедуре в рассматриваемой параллельной системе должен соответствовать некоторый вычислительный путь в рамках структуры (1). Такой вычислительный путь называется справедливым (fair). Вычислительный путь, не имеющий соответствующей процедуры в рассматриваемой системе, называется несправедливым (unfair). Задачей верификации является доказательство корректности поведения систем исключительно на множестве достижимых вычислительных путей. В определенных случаях задача верификации может быть применена не ко всей системе (1), а лишь к ее подмножеству. Интересующая часть модели в таких случаях выделяется при помощи ограничений справедливости (fairness constraints).

По своей природе структура Крипке является конечным автоматом, у которого отсутствуют входные переменные, выходы, а функция перехода R зависит только от внутреннего состояния. В литературе подробно описаны алгоритмы перехода от традиционных моделей конечных автоматов к структурам Крипке и алгоритмы обратного перехода [2].

2.2. Становление математического аппарата темпоральной логики. Проблема определения понятия времени является одним из центральных философских вопросов, и попытки формализации логики, учитывающей время, известны со времен древнегреческих философов. Среди основных характеристик, описанных в литературе моделей времени, выделяют следующие:

– дискретность или непрерывность времени;
– элементарная единица времени: миг или интервал;
– конечность или бесконечность времени;
– наличие/отсутствие начального и конечного момента времени;
– последовательность или цикличность времени;
– линейный или древовидный характер времени;
– направление времени (вперед, назад);
– относительность или абсолютность времени.

Наиболее распространены два принципиальных типа моделей времени, различающиеся по выбранному типу элементарной временной единицы: точечные и интервальные. Точечная модель представляет собой множество моментов времени (миги) и допускает операции сравнения между моментами (равенство, ранее, позднее). Точечная модель дискретна, транзитивна, ациклична и бесконечна. Для рассуждения о прошлых и будущих событиях в точечной модели используется некоторая нулевая точка, соответствующая текущему времени. Интервальная модель допускает не только операции сравнения между элементарными промежутками времени, а также разрешает анализ пересечений и включений

интервалов. Интервальная модель может быть как дискретной, так и непрерывной, она также ациклична и бесконечна.

Современная темпоральная логика в ее применении к компьютерным наукам была описана Pnueli в 1977 г. [3], в работах которого было предложено применять набор операторов моделирования времени для задания спецификаций и верификации сложных параллельных систем. Такая логика относится к классу точечных моделей, течение времени считается дискретным.

Темпоральная логика позволяет задать порядок событий во времени без внесения понятия времени в явном виде. Применение темпоральной логики для описания ограничений параллельных систем показало свою эффективность в таких распространенных задачах как:

– спецификация требований к параллельным системам;

– планирование запуска программных процессов и потоков под управлением операционной системы;

– синхронизация работы параллельных процессов;

– верификация цифровых систем.

2.3. Основные классы и операторы темпоральной логики. Существует два основных класса темпоральной логики, применяемых для задач верификации систем с конечным числом состояний:

– линейная темпоральная логика (LTL – Linear Temporal Logic);

– ветвящаяся темпоральная логика (CTL – Computation Tree Logic).

Линейная темпоральная логика одновременно рассматривает лишь один возможный сценарий изменения будущих состояний системы, центральным понятием здесь является вычислительный путь. Ветвящаяся темпоральная логика одновременно рассматривает несколько возможных вариантов изменения будущих состояний системы, центральным объектом здесь является дерево вычислительных процессов.

В применении к функциональной верификации систем оба класса темпоральной логики характеризуются различными свойствами. LTL-логика более проста в понимании для пользователя, более естественна при написании формул, так как течение времени имеет линейный характер. Кроме того, линейное представление времени сочетается с принципами динамического моделирования системы. CTL-логика более универсальна при описании вычислительных процессов и ограничений общего плана. Она удобна для методов формальной верификации – для проверки свойств и доказательства дедуктивных теорем. Однако древовидное представление течения времени (несколько сценариев будущего) не подходит для методов динамической верификации, где выбор конкретных вычислительных путей определяется не методом проверки модели, а входной тестовой последовательностью. В данном исследовании рассматривается применение темпоральной логики в рамках процесса имитационного моделирования системы (в HDL-симуляторе), соответственно свойства LTL-логики необходимо рассмотреть более подробно.

2.4. Классификация LTL-ограничений. С точки зрения цели специфицирования различают два принципиальных вида темпоральных ограничений, которые можно задавать при помощи утверждений темпоральной логики:

– инвариантные ограничения (safety) – заданная последовательность состояний никогда не случится (например, ресурс никогда не будет использован двумя процессами одновременно);

– достижимые ограничения (liveness) – желаемая последовательность состояний будет достигнута (например, пришедший запрос когда-нибудь обязательно будет обработан).

Типичным инвариантным ограничением будет формула вида $G(\neg f)$ (нечто плохое не произойдет ни на одном из вычислительных путей), или подобного вида $G(f)$ (некоторое условие безопасности должно соблюдаться для всех вычислительных путей).

В качестве достижимого ограничения наиболее часто может выступать формула вида $F(f)$ (хотя бы в одном из случаев нечто желаемое обязательно будет зафиксировано).

2.5. LTL-ограничения с тактовыми переменными. Модель параллельных систем (1) подразумевает дискретное течение времени при переходе из состояния в состояние, при

котором интервал изменения времени при выполнении перехода не оговаривается. Для верификации реальных систем на кристалле применяется уточненная структура Крипке, учитывающая тактирование. Множество переменных системы здесь разделено на информационные D и тактовые C:

$$M^C = (\Sigma : 2^D \cup \{\text{true}, \text{false}\}, C, S, S_0, R, L, \Delta), \quad (2)$$

где Σ, D, S, S_0, R, L унаследованы из базовой структуры (1); C – множество тактовых переменных; $\Delta : S \times S \times C, \Delta \overset{\text{def}}{\geq} 0$ – функция временного сдвига, задающая изменение тактовой переменной $c \in C$ при переходе между смежными состояниями. Переход $s_i \Rightarrow s_j$ называется тактовым, если изменяется хотя бы одна тактовая переменная:

$$R(s_i, s_j) = 1, \exists c \in C, \Delta(s_i, s_j, c) > 0. \quad (3)$$

Только тактовый переход может изменить параметр времени. Считается, что изменения информационных переменных происходят мгновенно.

Формулы с учетом тактовых переменных рассматривают только те переходы между состояниями вычислительного пути, в которых имеет место сдвиг времени. Для таких формул используется специальная форма записи с указанием тактовой переменной:

$$\pi \models^c f. \quad (4)$$

Вычислительный путь π, для которого выполняется следующее соотношение, называется тактовым циклом относительно тактовой переменной c:

$$\left\{ \begin{array}{l} |\pi| > 0, \\ \pi^{|\pi|-1} \models c, \\ \exists j, 0 \leq j < (|\pi| - 1), \pi^j \not\models c. \end{array} \right\}. \quad (5)$$

Учет тактовых переменных влияет на семантику некоторых операторов LTL. Например, оператор X в варианте с тактированием рассматривает не любое следующее изменение состояния системы, а только следующий тактовый переход (с точки начала анализа должны быть достигнуты моменты текущего j и следующего k тактового перехода):

$$\pi \models^c X(f) \Rightarrow \exists j, k, 0 \leq j < k < |\pi|, \left\{ \begin{array}{l} \pi_{0..j} \not\models c, \\ \pi_{j..j} \models c, \\ \pi_{j+1..k} \not\models c, \\ \pi_{k..k} \models c, \\ \pi_k / \models^c f \end{array} \right\}. \quad (6)$$

Семантика остальных темпоральных операторов с учетом тактирования видоизменяется аналогичным образом.

2.6. Формальная проверка LTL-ограничений. Для формальной проверки LTL-ограничений применяют модель недетерминированных автоматов Bьchi [4]:

$$A = \{Q, \Sigma, \partial, I, F\}, \quad (7)$$

где Q – множество всех состояний системы; Σ – конечный алфавит системы; ∂ – функция перехода между состояниями по набору утверждений из алфавита; $I \in Q$ и $F \in Q$ – подмножества начальных и конечных (удовлетворяющих) состояний системы соответственно. Пусть имеется слово, представляющее собой последовательность булевых продвижений $a_0, a_1, \ldots$ в рамках алфавита системы, такое, что каждому i соответствует некоторое состояние системы $q_i \in Q$, и для всех i выполняется соотношение $(q_i, a_i, q_{i+1}) \in \partial$. Тогда множество $L(A)$ всех возможных слов, порождаемых автоматом A, называется языком, принимаемым автоматом.

Автоматы Bьchi отличаются от традиционных моделей конечных автоматов по двум принципиальным признакам:

– традиционный конечный автомат способен определить принадлежность моделируемому языку входной последовательности конечной длины; автоматы Bьchi рассматривают бесконечные последовательности;

– в традиционном варианте модели успешное считывание автоматом входной последовательности определяется завершением процесса в удовлетворяющем состоянии; в варианте автоматов Bьchi успешным проходом считают ситуацию, когда последовательность проходит через одно из удовлетворяющих состояний бесконечное число раз.

Существует ряд алгоритмов трансформирования любой LTL-формулы к модели автомата Bьchi [5-7] и минимизации автоматов [8].

Модель верифицируемой системы (1) может быть приведена к виду:

$$\begin{gathered} Q = S, I = S_0, \\ (s \in S, P \in \Sigma, s' \in S) \in \partial \Leftrightarrow (s, s') \in R, L(s) = P. \end{gathered} \tag{8}$$

Задача формального определения удовлетворения системы некоторой LTL-формуле сводится к сопоставлению языков, принимаемых автоматом $L(A_M)$, соответствующим верифицируемой системе, и автоматом $L(A_f)$, сконструированным для LTL-формулы:

$$L(A_M) \subseteq L(A_f). \tag{9}$$

Пересечение двух автоматов Bьchi осуществляется при помощи операции декартового произведения компонентов-множеств. Результирующая модель является двумерным автоматом Bьchi:

$$\begin{gathered} A' = A_M \times A_{\neg f} = \{Q', \Sigma', \partial', I', F'\} \Rightarrow \\ \Rightarrow \begin{cases} Q' = Q_M \times Q_{\neg f}, \\ \Sigma = \Sigma_M \times \Sigma_{\neg f}, \\ I' = I_M \times I_{\neg f}, \\ F' = F_M \times F_{\neg f}, \\ \partial' = (\partial_M, \partial_{\neg f}). \end{cases} \end{gathered} \tag{10}$$

Более рациональным вариантом с точки зрения вычислительной сложности является трансформация задачи (10) к задаче пересечения $L(A_M)$ с автоматом, обратным интересующей LTL-формуле [9]. Очевидно, если существует хотя бы одна последовательность событий, принимаемая как автоматом системы, так и автоматом обратной формулы, то система не удовлетворяет интересующему свойству:

$$L(A_M) \bigcap L(A_{\neg f}) = L(A_M \bigcap A_{\neg f}) = L(A_M \times A_{\neg f}) = \varnothing. \tag{11}$$

Система M удовлетворяет LTL-формуле f, если язык автомата (10) не принимает ни одной входной последовательности. Чтобы опровергнуть данный факт (т.е. доказать несоблюдение системой интересующего ограничения) с учетом особенностей условия удовлетворения автоматов Bьchi, требуется найти в автомате A' хотя бы один цикл, для которого выполняется совокупность следующих условий:

1. Достижимость цикла из множества начальных состояний I'.
2. Наличие в цикле хотя бы одного состояния, принадлежащего F'.
3. Сильная связность цикла, т.е. из каждого входящего в цикл состояния должен существовать хотя бы один непустой путь в каждое из других состояний цикла.

Последнее условие является необходимым для бесконечного прохождения бесконечной последовательности через удовлетворяющее состояние. Обнаруженный цикл, удовлетворяющий перечисленным условиями, и путь из начального состояния к данному циклу составляют верификационный контр-пример, т.е. вычислительный путь, опровергающий свойство системы. Анализ вычислительной сложности формальной проверки LTL-ограничений, подробно рассмотренный в работе [9], демонстрирует нелинейную зависимость времени проверки от числа состояний системы и структурной сложности интересующей LTL-формулы (“взрыв состояний” – state explosion) [10]:

$$O(L(A_M \times A_{\neg f})) = O(|Q_M| * 2^{|Q_{\neg f}|}). \tag{12}$$

Такая высокая стоимость формальной проверки накладывает существенные ограничения на применимость метода к крупным моделям. При оценке применимости формальных методов проверки следует также учесть факт экспоненциального роста количества состояний при переводе формулы LTL в автоматную модель $A_{\neg f}$. Производительность рассмотренного метода существенно улучшается определением ограничений справедливости, исключающих из рассмотрения недостижимые состояния.

2.7. Проверка LTL-ограничений динамическими методами. Альтернативным методом верификации LTL-формул является проверка выполнения темпоральных ограничений во время имитационного моделирования системы (симуляции) [11]. Требуется исполняемое представление верифицируемой системы (1), например, скомпилированная модель на языках описания аппаратуры – Verilog/VHDL. Представляется возможным построить некоторый наблюдающий конечный автомат, реализующий LTL-формулу, подключаемый к программе-симулятору, и сигнализирующий о нарушениях ограничений. Такой тип верификации обладает рядом преимуществ и недостатков по сравнению с формальными методами:

1) Проблема "взрыва состояний" ограничивает практическую применимость формальных методов на больших компонентах ($>2^{20}$ состояний). Динамические методы не моделируют пространство состояний в явном виде, поэтому применимы к компонентам на несколько порядков более сложным, чем предельные компоненты формальных методов.

2) Формальные методы требуют построения моделей на высоком уровне абстракции. Возникают проблемы адекватности и точности этих абстракций. Модели для динамических методов могут быть существенно более детализированными.

3) Формальные методы гарантируют полное покрытие достижимых состояний системы. Покрытие при использовании динамических методов на несколько порядков ниже, поскольку оно определяется внешним тестом, исполняемым симулятором.

4) Формальные методы гарантируют 100% точность обнаружения нарушений или подтверждения удовлетворения заданных ограничений на вычислительных путях бесконечной длины. Динамические методы могут обнаружить лишь нарушение инвариантного ограничения или достижение желаемой последовательности в рассмотренных симуляцией отрезках вычислительных путей. Если анализ ограничения не завершен по окончанию теста, симуляция не может гарантировать, что свойство будет выполняться или опровергаться при расширении теста дополнительными векторами.

Для проверки LTL-формулы во время симуляции должен быть построен традиционный детерминированный конечный автомат:

$$A = \{\Sigma, Q, q_0, F, \partial\}, \tag{13}$$

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. Нарушением инвариантного ограничения будет считаться попадание автомата в одно из состояний F, так же как и успешный анализ желаемой последовательности. Непопадание в такую ситуацию по завершению анализа означает незавершенный анализ, и может быть разрешено в зависимости от выбранных ограничений условной справедливости.

Поскольку автомат (13) определен как детерминированный, возможные опциональные ветвления в темпоральных ограничениях должны преобразовываться в набор индивидуальных свойств, не имеющих ветвления. В таком случае результат верификации определяется конъюнкцией индивидуальных вычислений в случае инвариантного ограничения и дизъюнкцией в случае поиска желаемой последовательности.

Проверка LTL-формул во время симуляции характеризуется линейной сложностью относительно количества состояний в наблюдающем автомате и не зависит от сложности верифицируемой системы, что существенно лучше, чем оценка (12) для формальных методов. Однако следует учесть, что проблема "взрыва состояний" проявляется и в динамических методах: в общем случае количество состояний в автомате (13) для LTL-операторов растет экспоненциально в зависимости от структурной сложности формулы.

3. Формальная семантика языковых стандартов описания ассерций (распространенные языки описания ассерций и их свойства)

На прикладном уровне спецификация ограничений, использующих математический аппарат линейной темпоральной логики, описанный выше, осуществляется при помощи языков описания ассерций. Первое поколение LTL-языков зародилось в рамках исследовательских работ, ориентированных на методы формальной верификации [12, 13].

Несколько позже ведущими EDA-компаниями был предложен ряд языков, направленных на использование LTL для проверки свойств системы во время симуляции. Важной особенностью этих языков являлась простота стыковки верификационных моделей с сигналами реальной HDL-модели проверяемой цифровой системы. К наиболее известным языкам этого поколения относятся:

– Язык OpenVera Assertions (OVA), предложенный компанией Synopsys как часть унифицированного маршрута функциональной верификации OpenVera, реализованного в симуляторе VCS [14]. Данный язык полностью ориентировался на динамические методы. Внешне синтаксис языка слабо напоминает формальные описания и более похож на Verilog. В настоящее время язык постепенно исчезает с EDA-рынка.

– Язык Sugar, позднее переименованный в Property Specification Language (PSL), предложенный компанией IBM. Данный язык в равной степени ориентируется как на динамические, так и на формальные методы. Кроме того, в PSL существуют конструкции не только LTL, но и CTL-логики. PSL предоставляет различные нотации для булевого уровня (элементарные выражения могут быть заданы на языке Verilog, VHDL, SystemC, SystemVerilog, GDL по желанию пользователя). Также предлагаются альтернативные нотации для темпоральных операторов: предоставляется выбор между математическим (операторы называются F, X, U, G, также как и в оригинальной математической нотации) и прикладным стилями (операторы именуются ключевыми словами на английском языке). В 2005г. язык был стандартизирован, и в настоящее время известен как IEEE-1850 Property Specification Language.

– Язык SystemVerilog (раздел Concurrent Assertions), предложенный EDA-консорциумом Accellera, называемый HDVL-языком (Hardware Description & Verification Language). Наиболее мощный язык с точки зрения предоставляемых языковых конструкций и степени интеграции верификационной модели с моделью проверяемой цифровой системы. Изначально ориентировался только на динамические методы, однако в результате стандартизации в виде документа IEEE-1800 в 2005г., а также в предстоящей в 2009г. редакции в язык были введены конструкции для формальных методов. В перспективе язык может вытеснить остальные LTL-языки с рынка EDA, поскольку продвигается большинством лидеров рынка.

В данной работе большая часть описания LTL-ассерций использует язык PSL в силу близости его синтаксиса к математической форме. Большинство конструкций темпорального уровня из языков OVA и SystemVerilog подобны конструкциям PSL по сути и отличаются, в основном, стилем синтаксиса и стыковки с HDL-моделями.

Во всех трех языках выделяют три уровня иерархии модели верификации:

1. Булев уровень (Boolean Layer) – уровень примитивных логических выражений, исчисляемых за один временной слот, являющихся строительными примитивами для конструкций других уровней.

2. Темпоральный уровень (Temporal Layer) – уровень последовательностей и темпоральных свойств, задающих отношения между сигналами с течением времени, являющийся набором последовательностных регулярных выражений (SERE – Sequential Extended Regular Expressions).

3. Верификационный уровень (Verification Layer) – уровень верификационных директив, определяющих предназначение описываемых темпоральных формул, обеспечивающий интеграцию верификационной модели с проверяемой цифровой системой, другими компонентами тест-бенча.

В случае динамических методов анализ языковых конструкций позволяет ответить – исполняется ли некоторая темпоральная формула верифицируемой системой на наблюдаемом линейном отрезке вычислительного пути, заданном выполненным тестом.

Последовательностные регулярные выражения PSL. Ключевой темпоральной абстракцией в языке PSL являются последовательностные регулярные выражения (SERE): любое булево выражение, включая константы, является SERE; если c – булево выражение, а r, r_1, r_2 – SERE, то следующие выражения также являются SERE; $\{r\}$ – скобки, влияющие на приоритет темпоральных операторов; $\{r_1; r_2\}$ – непересекающаяся конкатенация; $\{r_1 : r_2\}$ – пересекающаяся конкатенация, или склеивание; $\{r_1\}|\{r_2\}$, $\{r_1\}\&\{r_2\}$, $\{r_1\}\&\&\{r_2\}$ – последовательностные логические операторы OR, AND, INTERSECT соответственно; $[*0]$ – специальное SERE, означающее пустой вычислительный путь; $r[*]$ – бесконечная репетиция SERE; $r@c$ – оператор присвоения тактового контекста.

Ниже представлена формальная семантика конструкций SERE на некотором конечном вычислительном пути π:

1. Скобки влияют лишь на приоритет последовательностных операторов, и не влияют на интерпретацию выражения-операнда r:

$$\pi \models \{r\} \Leftrightarrow \pi \models r. \tag{14}$$

2. Булево выражение b справедливо на отрезке единичной длины, если оно выполняется в единственном состоянии системы на данном пути:

$$\pi \models b \Leftrightarrow |\pi| = 1 >, \pi^0 \models b. \tag{15}$$

3. Непересекающаяся конкатенация SERE справедлива, если π можно представить как конкатенацию двух отрезков, на каждом из которых по отдельности справедливы SERE-операнды:

$$\pi \models \{r_1; r_2\} \Leftrightarrow \exists \pi', \pi'', \begin{cases} \pi = \pi'\pi'', \\ \pi' \models r_1, \pi'' \models r_2. \end{cases} \tag{16}$$

4. Пересекающаяся конкатенация SERE подобна непересекающейся, с той разницей, что первый отрезок, составляющий конкатенацию, имеет единственную общую точку со вторым отрезком:

$$\pi \models \{r_1 : r_2\} \Leftrightarrow \exists \pi', \pi'', l, \begin{cases} \pi = \pi' l \pi'', |l| = 1, \\ \pi' l \models r_1, \pi'' l \models r_2. \end{cases} \tag{17}$$

5. Последовательностная конъюнкция и дизъюнкция SERE определяются результатами вычислений по отдельности, а для конъюнкции накладывается дополнительное ограничение на одновременное завершение анализа операндов:

$$\begin{aligned} \pi \models r_1 \mid r_2 &\Leftrightarrow (\pi \models r_1) \vee (\pi \models r_2), \\ \pi \models r_1 \,\&\&\, r_2 &\Leftrightarrow (\pi \models r_1) \wedge (\pi \models r_2). \end{aligned} \tag{18}$$

6. Оператор пересечения последовательностей подобен конъюнкции последовательностей, однако здесь снимается требование одновременности завершения анализа формул-операндов:

$$\pi \models r_1 \,\&\, r_2 \Leftrightarrow \pi \models \{\{r_1\}\&\&\{r_2; \text{true}[*]\}\} \mid \{\{r_1; \text{true}[*]\}\&\&\{r_2\}\}. \tag{19}$$

7. Оператор включения подпоследовательности в последовательность:

$$\pi \models \{r_1\}\text{within}\{r_2\} \Leftrightarrow \{[\text{true}*]; r_1; [\text{true}*]\}\&\&\{r_2\}. \tag{20}$$

8. Бесконечная репетиция выполняется либо на пустом отрезке, либо на конкатенации двух отрезков, на первом из которых выполняется формула-операнд, а на втором отрезке репетиция справедлива по рекурсии:

$$\pi \models r[*] \Leftrightarrow \begin{cases} \pi \models [*0], \\ \exists \pi', \pi'', \begin{cases} \pi = \pi'\pi'', \pi' \neq \varepsilon, \\ \pi' \models r, \\ \pi'' \models r[*] \end{cases} \end{cases}. \quad (21)$$

Наиболее часто применяемыми регулярными выражениями являются непересекающиеся конкатенации (называемые операторами временного сдвига в SystemVerilog и OVA), а также репетиции.

Целое константное выражение, задающее число итераций в репетициях, называется репликатором. При помощи репетиций удобно моделировать часто требуемые вычисления циклического характера, поэтому в PSL введен богатый синтаксический набор производных формул, основанных на репетициях. По принципу временного сдвига между итерациями репетиции делятся на:

– консекъютивные последовательностные репетиции $r[*n]$ – итерации сменяются с использованием оператора непересекающейся конкатенации;

– неконсекъютивные булевы репетиции $b[=n]$ – итерации с успешным выполнением булевого выражения могут быть разделены произвольным числом незначащих тактов;

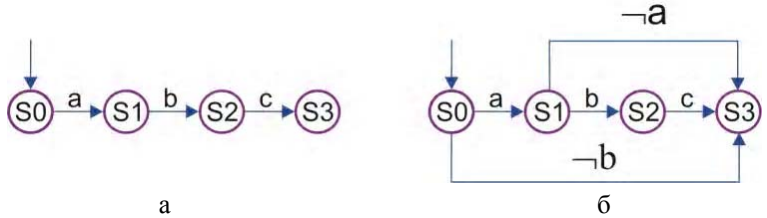
– галопирующие булевы репетиции $b[\rightarrow]$ – итерации с незначащими интервалами произвольной длины, обязательно завершающимися успешным выполнением булевого операнда.

С точки зрения детерминированности количества итераций репетиции могут иметь одиночный или интервальный репликатор, означающий возможное ветвление. Репликаторы, в свою очередь, могут иметь значения 0 (операнд не будет выполнен ни разу), любое конечное целое число итераций, а также значение бесконечности (операнд повторяется неограниченное число раз).

4. Режимы интерпретации локального и глобального времени

Очевидно, многие предоставляемые языком PSL темпоральные свойства в неявном виде подразумевают параллельный анализ пересекающихся последовательностей, в том числе с возможными вариантами ветвления времени. Такая выразительность семантики необходима для адекватного отображения требований спецификации, выраженных естественным языком, и хорошо согласуется с формальными методами. При интерпретации формул во время симуляции, напротив, наиболее благоприятным случаем является линейная последовательность состояний.

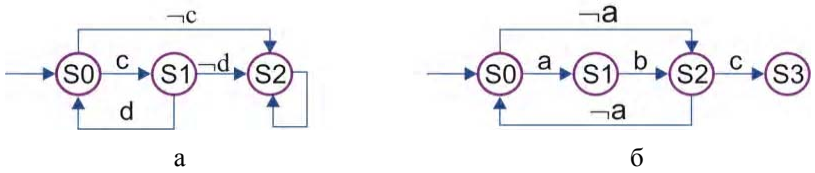
Суть проблемы заключается в экспоненциальном росте количества состояний при трансформировании языковых конструкций к модели детерминированного автомата (13). Одной из характерных конструкций, приводящих к существенному расширению компонентов автомата, является отрицание продвижения последовательности. Пусть имеется простейшая последовательность $r = \{a;b;c\}$. На рис. 1,а показан эквивалентный автомат, а на рис. 1,б – его комплемент $r' = \neg(\{a;b;c\})$. Поскольку отрицание последовательности нужно рассматривать как несоблюдение условий в любом из его состояний, то из каждого состояния, кроме завершающего, следует по дополнительному переходу к конечному состоянию:



а б

Рис. 1. Комплементирование последовательности в виде автомата

Но наибольшую степень роста компонентов автомата дают операторы G, F и U, представление которых в виде автомата генерирует циклические переходы. Особую сложность для преобразования в детерминированный автомат представляют выражения конъюнкции и дизъюнкции, имеющие в качестве операндов формулы с использованием перечисленных выше глобальных операторов. Реализация таких выражений требует сложных комбинаций декартового произведения и комплементирования автоматов-операндов, что делает производительность проверки формул во время симуляции мало реалистичной. Пусть имеется формула $f = G(\{a;b\}) \vee F(\{c;d\})$. Оценим количество состояний в эквивалентном проверяющем автомате. Для удобства приведем формулу к конъюнктивной нормальной форме $= \neg(G(\neg\{a;b\}) \wedge F(\neg\{c;d\}))$. На рис. 2,а и б показаны представления формул-операндов в виде отдельных автоматов:



а б

Рис. 2. Реализация глобальных LTL-операторов в виде автоматов

Каждая из формул-операндов порождает по 3 состояния и по 4 перехода. Реализовав конъюнкцию при помощи декартового произведения, получим следующий двумерный автомат, в развернутом виде представленный на рис. 3. Очевидно, комплементирование этого более сложного автомата способно лишь увеличить число переходов при инверсии успешных и неудачных переходов. Таким образом, в итоговом автомате имеется 9 состояний и 18 переходов. Учитывая динамику роста компонентов автомата, несмотря на относительную простоту последовательностей-операндов, процедура анализа постепенно теряет производительность. Добавление по одному дополнительному состоянию в последовательностях-операндах увеличит число состояний до 12, а число переходов – до 24. Неутешительная перспектива становится очевидной, если учесть факт, что требования спецификации реальных систем имеют существенно большую последовательностную глубину и степень ветвления.

В определенной степени на размер автомата (17) могут повлиять различные общеизвестные методы минимизации количества состояний. Однако в целях более существенного ограничения динамики роста компонентов автомата используется так называемое простое подмножество языковых конструкций, обрабатываемое в режиме локального времени. На семантику ряда LTL-операторов наносятся структурные ограничения на допустимые операнды, а интерпретация глобальных темпоральных операторов упрощается.

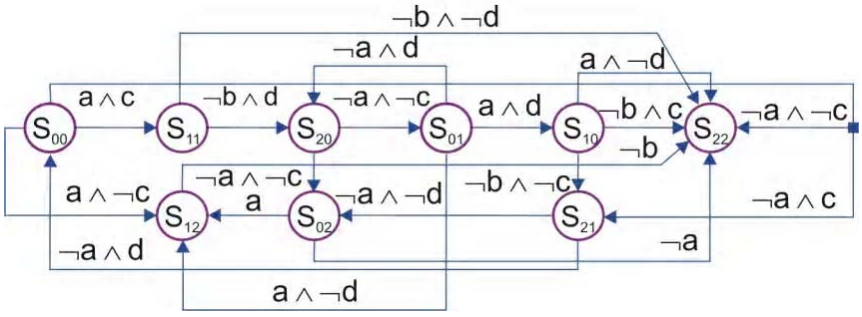


Рис. 3. Конъюнкция глобальных LTL-операторов в виде автомата

В частности, на операнды темпоральных свойств в простом подмножестве накладываются следующие ограничения:

1. Допускается отрицание только булевого выражения. Отрицание последовательностей или темпоральных свойств запрещается.

2. Не допускается помещение в оператор never операндов, не являющихся последовательностью или булевым выражением, поскольку такая конструкция при преобразовании к базовым LTL-операторам приведет к необходимости отрицания последовательности, а значит, к нарушению п. 1.

3. Операндом оператора eventually! может быть только последовательность или булево выражение для ограничения степени ветвления времени, порождаемого данным оператором.

4. Не более одного операнда логических операторов дизъюнкции и импликации, а также ни один из операндов оператора логической эквивалентности не может быть не булевым.

5. Операнды таких операторов, как until*, before*, next_e*, next_event_e* могут быть только булевыми.

Иначе упрощение простого подмножества можно назвать дополнительным уровнем линеаризации времени [15]. Такое упрощение в значительной степени ограничивает рост количества состояний и переходов в представляющем формулу автомате. Формулы простого подмножества доступны в понимании и практическом применении инженерами для ежедневных нужд верификации. Однако их использование ограничивает выразительность формул и ухудшает желаемую близость верификационных определений к требованиям спецификации, выраженным естественным языком.

Под уплощением интерпретации глобальных темпоральных операторов понимают такое преобразование LTL-формул, при котором единственный результат, связываемый с вычислительным путем бесконечной длины, заменяется бесконечным множеством результатов для всех возможных конечных вычислительных путей, начинающихся с различных начальных моментов времени. В частности, интерпретация ключевых глобальных темпоральных операторов, отображенная в табл. 1, такова, что на каждый из вычислительных путей, существующих в рамках рассматриваемого теста, накладывается отдельное независимое ограничение. Такой способ интерпретации повышает производительность анализа за счет устранения теоретически бесконечной фазы ветвления и предоставляет детальную диагностическую информацию о каждом проанализированном отрезке.

Таблица 1. Режимы интерпретации темпоральных операторов

Формула PSL	Полноценная интерпретация	Упрощенная интерпретация
always f	$\pi \models \text{always f} \Leftrightarrow \pi \models \text{G f}$ $\forall k, 0 \le k < \lvert\pi\rvert, \pi_k \models \text{f} \Rightarrow 1$ Единственный результат	$\pi \models \text{always f} \Leftrightarrow \forall k, k < \lvert\pi\rvert, \pi_k \models \text{f}$ $\forall k, 0 \le k < \lvert\pi\rvert,$ $\{\pi_0 \models \text{f} \Rightarrow 1, \ldots, \pi_k \models \text{f} \Rightarrow 1, \ldots\}$ Множественный результат
never f	$\pi \models \text{never f} \Leftrightarrow \pi \models \neg(\text{G} \neg \text{f})$ $\exists k, 0 \le k < \lvert\pi\rvert, \pi_k \not\models \text{f} \Rightarrow 1$ Нет ограничений на операнд	$\pi \models \text{never f} \Leftrightarrow \forall k, k < \lvert\pi\rvert, \pi_k \not\models \text{f}$ $\forall k, 0 \le k < \lvert\pi\rvert,$ $\{\pi_0 \not\models \text{f} \Rightarrow 1, \ldots, \pi_k \not\models \text{f} \Rightarrow 1, \ldots\}$ Операнд может быть только булевым выражением или последовательностью
eventually! f	$\pi \models \text{eventually! f} \Leftrightarrow \pi \models \text{F f}$ $\exists k, 0 \le k < \lvert\pi\rvert, \pi_k \models \text{f}$ Нет ограничений на операнд, путь, на котором операнд справедлив, должен существовать	$\pi \models \text{eventually! f} \Leftrightarrow \pi \models \{[*]; \text{f}\}$ Операнд может быть только булевым выражением или последовательностью, удовлетворяющий путь может не существовать

Режим глобального времени. Использование только простого подмножества (режима локального времени) значительно облегчает реализацию проверки формул в симуляторах. Такая верификация эффективна при необходимости детальной диагностики блока, в котором уже обнаружена функциональная ошибка, и основной целью является локализация. Полноценная интерпретация формул LTL-логики при динамической верификации дает единый результат для всех вычислительных путей. Такой подход скрывает детальные результаты

вычислений на каждом пути и, в первую очередь, эффективен для быстрого обнаружения функциональных проблем блока. Оба подхода можно эффективно сочетать в цикле функциональной верификации системы. Сначала все темпоральные формулы следует проверить в режиме глобального времени. В случае обнаружения нарушений с конкретными верификационными утверждениями проблемные формулы следует проверить в диагностическом режиме с упрощенной интерпретацией.

Фактором, препятствующим полноценной интерпретации LTL-операторов во время симуляции, является большая ресурсоемкость при реализации с использованием детерминированных конечных автоматов. Решив данную проблему, можно получить уникальные возможности для верификации. Использование полноценной интерпретации высокоуровневых операторов LTL-логики для проверки справедливости формул во время симуляции будем называть режимом глобального времени. Такое название исходит из наличия единственного результата, что отличает данный режим от множественного результата в случае интерпретации в стиле простого подмножества.

Пусть имеется PSL-свойство $\text{property p} = \text{always}\{a;b\} |=> \{[*2];c|$, означающее, что для всех вычислительных путей последовательность событий $a=1$ на первом такте, $b=1$ на следующем такте должна сопровождаться событием $c=1$ в прошествии трех тактов. Ниже представлена математическая интерпретация данной формулы:

$$\forall k, k < |\pi|, \begin{bmatrix} \pi_k \models a, \begin{bmatrix} \pi_{k+1} \models b, \pi_{k+4} \models c \\ \pi_{k+1} \not\models b \end{bmatrix} \\ \pi_k \not\models a \end{bmatrix} \Rightarrow \pi_{k..} \models p \,. \tag{22}$$

В упрощенном варианте интерпретации данной формулы для любого k – для каждого рассматриваемого такта моделирования – будет дан ответ об удовлетворении формулы на данном вычислительном пути. Соответственно, если рассмотрено 10 тактов, то будет дано 10 несвязанных между собой ответов о справедливости формулы. В варианте же интерпретации глобального времени, будет дан единственный ответ для всего цикла моделирования, причем справедливым он будет лишь тогда, когда все рассмотренные вычислительные пути удовлетворяют верифицируемой формуле.

Ниже представлены (табл. 2) результаты моделирования на одинаковом тесте в двух режимах. Как показано в примере, в упрощенном режиме для 6 векторов имеется 6 ответов об удовлетворении формулы, ответ для вычислительного пути, начинающегося с первого такта, является отрицательным, в то же время для последующих вычислительных путей результат положительный. В режиме глобального времени имеется единственный результат, поскольку на 5 такте выясняется ошибка на первом вычислительном пути, и дальнейшая обработка не имеет смысла.

Таблица 2. Режим интерпретации глобального времени

#	a	b	c	simple	global time
1	1	0	1	-	-
2	1	1	0	-	-
3	1	1	1	-	-
4	0	0	0	$PASS^{3}$ $PASS^{4}$	-
5	0	0	0	$FAIL^{1}$ $PASS^{5}$	$FAIL^{0}$
6	0	0	1	$PASS^{2}$ $PASS^{6}$	-

Подмена темпоральных свойств на упрощенные последовательностные элементы требует некоторых вычислительных затрат на анализ. Разумеется, эти затраты превышают полноценную интерпретацию лишь в том случае, если решить проблему роста количества состояний.

Пусть имеется PSL-свойство $\text{property p} = \text{eventually!}\{a;b;c\}$:

$$\pi \models \text{eventually!}\{a;b;c\} \Leftrightarrow \pi \models \{[*];a;b;c\}; \tag{23}$$

$$\pi \models \text{eventually!}\{a;b;c\} \Leftrightarrow \pi \models X!\{a;b;c\}. \quad (24)$$

Эквивалентную роль повторной активации последовательности-операнда по завершению анализа неудачного вычислительного пути играют бесконечная репетиция $[*]$ в упрощенной модели и простое темпоральное свойство в режиме глобального времени соответственно. При идентичности функционального поведения двух режимов интерпретации использование репетиции с бесконечным интервалом требует большего объема вычислительных ресурсов с генерацией дополнительных циклов между состояниями. Вариант глобального времени здесь лишь блокирует верификацию при появлении успешного события, не накапливая никаких дополнительных данных в памяти.

Аналогично, существенных дополнительных затрат требует упрощенная интерпретация семейств темпоральных операторов until и before. Учитывая значительные ограничения на языковое подмножество, допустимое для операндов этих операторов, их использование при типичной реализации становится нерациональным с точки зрения производительности, несмотря на семантическую выразительность. Пусть имеется свойство property p = a until b[*2]. Его семантика в двух режимах анализа такова:

$$\pi \models \text{a until b}[*2] \Leftrightarrow \pi \models \{a[*0:\infty];b[*2]\};\ \pi \models \text{a until b}[*2] \Leftrightarrow \pi \models [\text{a W b}[*2]]. \quad (25)$$

Упрощенная интерпретация использует функцию репетиции с неограниченными минимальным и максимальным интервалами. Разумеется, при переводе к автомату (15) это существенно меньшая модель, чем полноценная интерпретация. Однако задействованный вид репетиции является одной из наиболее проблематичных конструкций с точки зрения сложности порождаемых циклических переходов между состояниями.

Время одной итерации верификации при типичной интерпретации LTL-формул можно оценить следующим соотношением:

$$N_{CLK} \times (\bar{t}_{SIM} + N_A \times \bar{t}_{ev}), \quad (26)$$

где N_{CLK} – число тактов моделирования; $\bar{t}_{SIM}$ – среднее время моделирования одного такта; N_A – число ассерций в модели; $\bar{t}_{ev}$ – среднее время обработки ассерции на одном такте. Следует отметить, что это суммарное время одинаково вне зависимости от результатов верификации. В режиме глобального времени соотношение изменяется:

$$N_{CLK}^{MAX} \times \bar{t}_{SIM} + \sum_{i=1}^{N_A} N_{CLK}^{i} \times \bar{t}'_{ev}. \quad (27)$$

Здесь число тактов моделирования может быть меньше и соответствует максимальному числу тактов, необходимому для выявления единственного ответа для каждой из ассерций. Это число различно для каждой из них. Кроме того, за счет более легковесных структур данных в режиме глобального времени типичное время анализа ассерции на одном такте будет меньше. При сравнении соотношений (26) и (27) становится очевидным:

– время анализа в режиме глобального времени ни при каких обстоятельствах не может быть больше времени анализа в обычном режиме;

– наибольший эффект снижения вычислительных затрат происходит при быстром выявлении нарушений (много ошибок на ранних фазах проектирования), когда значительно сокращается число тактов, на которых активно анализируется ассерция.

5. Выводы

1. Задача функциональной верификации параллельных систем, представляемых в виде структур Крипке, состоит в проверке логико-временных ограничений между атомарными высказываниями, ассоциированными с состояниями системы на всех возможных вычислительных путях. Утверждения об удовлетворении системой определенных ограничений, называемые ассерциями, отражают требования спецификации в виде формальной математической формулы, использующей аппарат темпоральной логики, который пришел в компьютерную инженерию из философии. В применении к верификации, в формальных методах используют логику CTL, в то время как логика LTL более совместима с динамическими

верификационными методами. Формальная проверка LTL-ограничений предполагает представление ассерций и верифицируемых систем в виде автоматов Bьchi, способных установить принадлежность моделируемому ограничению последовательностей состояний бесконечной длины. Результат верификации определяется отсутствием достижимых из начальных состояний бесконечных циклов при декартовом произведении двух автоматов Bьchi. Экспоненциальная сложность формальной проверки накладывает существенные ограничения на применимость метода к крупным моделям. Лучшую производительность верификации LTL-ограничений обеспечивают динамические методы, функционирующие путем конвертирования LTL к детерминированному конечному автомату. Несмотря на проблему низкого покрытия пространства состояний системы (зависящую от внешнего теста) в динамических методах, а также нелинейный рост размера проверяющих автоматов в зависимости от структурной сложности формулы, использование LTL-логики во время симуляции способно обеспечить обнаружение большинства функциональных нарушений с более приемлемой производительностью по сравнению с формальными методами.

2. Наиболее популярными языками описания ассерций являются PSL и SystemVerilog, характеризующиеся булевым, темпоральным и верификационным уровнями конструкций. Семантическая выразительность логико-временных соотношений с использованием данных языков определяется наличием широкого набора параметризуемых последовательностных регулярных выражений и темпоральных свойств. Языковые стандарты формально определяют понятие простого подмножества, ограничивающего набор допустимых конструкций таким образом, чтобы заблокировать возможности экспоненциального роста количества состояний при конвертировании LTL-формул в конечные автоматы. Простое подмножество рекомендовано к применению в динамических методах, однако его использование ограничивает выразительность формул и ухудшает желаемую близость формальных ограничений к требованиям спецификации.

3. Научная новизна. Предложена модель интерпретации высокоуровневых операторов LTL-логики для проверки справедливости формул во время симуляции – режим глобального времени. Такой подход скрывает детальные результаты вычислений на каждом пути и, в первую очередь, эффективен для быстрого обнаружения функциональных проблем блока. Напротив, простое подмножество предполагает упрощенную интерпретацию глобальных темпоральных операторов, называемую режимом локального времени. Оба подхода можно эффективно сочетать в цикле функциональной верификации системы, применяя режим глобального времени для быстрого выявления нарушений, а затем режим локального времени для детальной диагностики и локализации.

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, pp. 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.** *Bьchi 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 Bьchi 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 Bьchi 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 Bьchi 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.** *Bergeron J., Cerny E., Hunter A., Nightingale A.* Verification Methodology Manual for SystemVerilog. Springer, 2006. 528 p. **15.** *Ruah S., Fisman D., Ben-David S.* Automata Construction for On-The-Fly Model Checking PSL Safety Simple Subset. IBM, Tech. Rep. H-0234. 2005. 22 p.

Поступила в редколлегию 22.11.2009

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

Литвинова Евгения Ивановна, канд. техн. наук, доцент кафедры технологии и автоматизации производства РЭС и ЭВС ХНУРЭ. Научные интересы: алгоритмизация задач автоматизированного проектирования электронных вычислительных средств, автоматизация диагностирования и встроенный ремонт компонентов цифровых систем в пакете (SiP). Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. 70-21-421.

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