УДК 004.415.53

В.П. Иванников, А.С. Камкин, М.М. Чупилко

ПРОВЕРКА КОРРЕКТНОСТИ ПОВЕДЕНИЯ HDL-МОДЕЛЕЙ ЦИФРОВОЙ АППАРАТУРЫ НА ОСНОВЕ ДИНАМИЧЕСКОГО СОПОСТАВЛЕНИЯ ТРАСС

V.P. Ivannikov, A.S. Kamkin, M.M. Chupilko

VERIFYING CORRECTNESS OF HDL-MODEL BEHAVIOR ON THE BASIS OF DYNAMICAL TRACE MATCHING

Проверка корректности поведения HDL-моделей является неотъемлемой частью динамической верификации аппаратуры. Как правило, она основана на сравнении поведения HDL-модели с поведением эталонной модели, разработанной на языке программирования. В процессе верификации на обе модели подается одна и та же последовательность стимулов; реакции перехватываются и сравниваются друг с другом. Из-за абстрактности эталонной модели сопоставление трасс не является тривиальной задачей: порядок событий может не совпадать, а некоторые события одной трассы могут отсутствовать в другой. Рассмотрен метод динамического сопоставления трасс для моделей аппаратуры разного уровня абстракции. Метод успешно применен в нескольких промышленных проектах по верификации модулей микропроцессоров.

ДИНАМИЧЕСКАЯ ВЕРИФИКАЦИЯ; МОДУЛЬНАЯ ВЕРИФИКАЦИЯ; HDL; СОПОСТАВЛЕНИЕ ТРАСС; ЧАСТИЧНО-УПОРЯДОЧЕННЫЕ МУЛЬТИМНОЖЕСТВА.

Correctness checking of HDL-model behavior is an integral part of dynamical verification of hardware. As a rule, it is based on comparing of HDL-model behavior and reference model behavior, developed in high-level programming languages. Being verified, both models are applied with the same stimulus sequences; their reactions are caught and checked against each other. Due to the abstractness of the reference model, the checking is not a trivial task as event sequences can be different and some events of one trace may miss in the other one. A methodology of dynamical trace matching for hardware models of different abstraction levels is considered in the paper. The methodology has been successfully used in a number of industrial projects of unit-level microprocessor verification.

DYNAMICAL VERIFICATION; UNIT-LEVEL VERIFICATION; HDL; TRACE MATCHING; PARTIALLY ORDERED MULTISETS.

Несмотря на развитие формальных методов, *динамическая верификация* остается основным методом проверки сложной цифровой аппаратуры [1]. Объектом верификации выступают не сами устройства, а их модели, разработанные на специальных *языках описания аппаратуры* (Hardware Description Languages – HDL), например, на Verilog или VHDL (такие модели называются *HDL-моделями*) [2]. С одной стороны, HDL-модели представляют основу для автоматизированного производства интегральных схем; с другой стороны, это имитационные модели, поведение которых можно анализировать в специальных средах (*симуляторах*). Для автоматизации проверки корректности поведения HDL-моделей разрабатываются *мониторы*, перехватывающие события на входах и выходах (*стимулы* и *реакции*) и проверяющие допустимость получаемой трассы [3].

Существует множество подходов к формальному описанию поведения, позволяющих автоматизировать создание мониторов: *расширенные регулярные выражения* [4], *контрактные спецификации* [5], *системы пра-*

вил [6], *темпоральные утверждения* [7], однако указанные формализмы не покрывают всех потребностей индустрии. Большинство компаний, проектирующих аппаратуру, для оценки проектных решений используют *исполнимые модели*, разработанные на языках программирования (прежде всего, на C и C++) [8]. Экономически целесообразно использовать эти модели и для верификации создаваемой аппаратуры, в частности, для проверки корректности поведения HDL-моделей.

При наличии исполнимой модели применяют следующую схему проверки корректности поведения: эталонная модель (по терминологии тестирования соответствия, *спецификация*) выполняется совместно с проверяемой HDL-моделью (*реализацией*); на спецификацию поступает та же последовательность стимулов, что и на реализацию; реакции обеих моделей перехватываются монитором и сравниваются. Основная проблема указанной схемы связана с обобщенным характером спецификации, из-за чего порядок реакций, выдаваемых спецификацией и реализацией, а также их состав могут различаться. Перед тем как использовать эталонную модель для мониторинга, необходимо понять, насколько она абстрактна и насколько можно доверять производимым ею трассам.

В статье предлагается способ построения мониторов для HDL-моделей аппаратуры, адаптируемый для эталонных моделей разного уровня абстракции. Рассматриваемый подход может быть формализован на основе модели *частично упорядоченных мультимножеств* [9]. Поведение спецификации и реализации описывается *временны́ми трассами* над общим алфавитом. Сведения об абстрактности спецификации позволяют обобщать последовательности выдаваемых реакций в частично упорядоченные мультимножества, в которых для каждой реакции задан допустимый *временной интервал*. Монитор проверяет, что трасса реализации является *линеаризацией* обобщенной трассы спецификации (или ее подмножества) и реакции реализации удовлетворяют ограничениям, заданными временны́ми интервалами.

Основные понятия и обозначения

В дальнейшем будем использовать следующие обозначения: Σ – конечный алфавит *событий*; $\mathbb{T}$ – *временна́я область* (для определенности, $\mathbb{N}$). Последовательности событий называются *словами*. Σ^* обозначает множество всех (конечных) слов над алфавитом Σ.

В контексте динамической верификации HDL-моделей удобно структурировать алфавит событий, разбив его на два множества: множество *входных событий* (*стимулов*) I и множество *выходных событий* (*реакций*) O, а также сопоставив каждому событию его *порт*: $port : \Sigma \to \{1, 2, ..., k\}$. На содержательном уровне стимул представляет собой посылку запроса к устройству, а реакция – выдачу ответа. При этом события на разных портах могут происходить параллельно.

Определение 1 (Временно́е слово [10]). Временны́м словом (timed word) w над алфавитом Σ и временно́й областью $\mathbb{T}$ называется конечная последовательность $(a_1, t_1)...(a_n, t_n)$ временны́х событий $(a_i, t_i) \in \Sigma \times \mathbb{T}$, удовлетворяющая следующим ограничениям:

1) для каждого $1 \le i < n$ выполняется неравенство $t_i \le t_{i+1}$;

2) для всех $1 \le i, j \le n$, таких что $i \ne j$ и $t_i = t_j$, $port(e_i) \ne port(e_j)$. □

В параллельных системах, к которым относится аппаратура, используется концепция *независимости* событий: два события считаются *независимыми*, если между ними нет причинно-следственной связи (для таких событий не накладываются ограничения на их относительный порядок). Эта идея лежит в основе двух формальных моделей параллельных вычислений: (1) *трасс Мазуркевича* [11] и (2) *частично упорядоченных мультимножеств* [9]. В настоящей статье мы будем придерживаться более общей второй модели.

Определение 2 (Частично упорядоченное мультимножество [9]). Σ-размеченным частичным порядком называется кортеж $\langle V, \prec, \lambda \rangle$, где V – конечное множество вершин, $\prec \subseteq V \times V$ – отношение частичного порядка и $\lambda : V \to \Sigma$ – функция

разметки. Два Σ-размеченных частичных порядка называются эквивалентными, если они изоморфны относительно $\prec$ и λ (либо совпадают, либо отличаются только названием вершин). Частично упорядоченным мультимножеством (pomset, partially ordered multiset) над алфавитом Σ называется класс эквивалентности Σ-размеченных частичных порядков. □

Для удобства мы будем использовать конкретного представителя (конкретный размеченный частичный порядок) для обозначения частично упорядоченного мультимножества. *Линеаризацией* частично упорядоченного мультимножества $\langle V, \prec, \lambda\rangle$ называется размеченный полный порядок $\langle V, \leq, \lambda\rangle$, где $\prec \subseteq \leq$.

Определение 3 (Временна́я трасса [12]). Временно́й трассой над алфавитом Σ и временно́й областью $\mathbb{T}$ называется четверка $\langle V, \prec, \lambda, \theta\rangle$, где $\langle V, \prec, \lambda\rangle$ – частично упорядоченное мультимножество, а $\theta : V \to \mathbb{T}$ – функция времени, удовлетворяющая следующему условию: для всех $x, y \in V$, из $x \prec y$ вытекает $\theta(x) \leq \theta(y)$. □

Множество всех трасс над алфавитом Σ и временно́й областью $\mathbb{T}$ обозначается $\mathbb{M}_\theta(\Sigma, \mathbb{T})$ или, для краткости, $\mathbb{M}$. Заметим, что определенные ранее временны́е слова являются частным случаем временны́х трасс (это трассы с тривиальным отношением частичного порядка на множестве вершин – отношением равенства).

Для заданной непустой трассы $\sigma = \langle V, \prec, \lambda, \theta\rangle$ введем обозначения:

$$begin(\sigma) = \min_{x \in V}\{\theta(x)\};$$
$$end(\sigma) = \max_{x \in V}\{\theta(x)\};$$
$$V_{[t,t+\Delta t]} = \{x \in V \mid \theta(x) \in [t, t+\Delta t]\};$$
$$\sigma_{[t,t+\Delta t]} = \langle V_{[t,t+\Delta t]}, \prec_{|V_{[t,t+\Delta t]}}, \lambda_{|V_{[t,t+\Delta t]}}, \theta_{|V_{[t,t+\Delta t]}}\rangle.$$

Пусть $\mathcal{TI}(\mathbb{T})$ – множество временны́х интервалов во временно́й области $\mathbb{T}$, то есть $\mathcal{TI}(\mathbb{T}) = \{[t, t+\Delta t] \mid t, t+\Delta t \in \mathbb{T}\}$.

Определение 4 (Интервальная трасса). Интервальной трассой над алфавитом Σ и временно́й областью $\mathbb{T}$ называется четверка $\sigma = \langle V, \prec, \lambda, \theta\rangle$, где $\langle V, \prec, \lambda\rangle$ – частично упорядоченное мультимножество, а $\delta : V \to \mathcal{TI}(\mathbb{T})$ – функция, сопоставляющая каждой вершине временно́й интервал. □

В дальнейшем мы будем иметь дело с *расширенными интервальными трассами*, описываемыми пятерками $\sigma = \langle V, \prec, \lambda, \theta, \delta\rangle$. В таких трассах с каждой вершиной $x \in V$ связан как момент времени $\theta(x)$, так и временно́й интервал $\delta(x) = [\theta(x) - \Delta t^-(x), \theta(x) + \Delta t^+(x)]$. Будем считать, что функции $\Delta t^\pm : V \to \mathbb{T}$ ограничены: существуют константы $\Delta T^\pm > 0$, такие что $|\Delta t^\pm(x)| \leq \Delta T^\pm$ для всех $x \in V$, и, кроме того, их значения зависят только от событий: если $\lambda(x) = \lambda(x')$, то $\Delta t^\pm(x) = \Delta t^\pm(x')$.

Динамическая верификация на основе исполнимых моделей

Временно́е слово (временна́я трасса с тривиальным частичным порядком) описывает конкретное выполнение HDL-модели (*реализации*), в то время как расширенная интервальная трасса описывает поведение эталонной модели (*спецификации*). Наша цель – проверить *в динамике* (on-the-fly), что трасса реализации w_I (временно́е слово) соответствует трассе спецификации σ_S (расширенной интервальной трассе). Поясним, почему в качестве спецификационной трассы рассматривается расширенная интервальная трасса. В процессе выполнения спецификация порождает конкретную трассу w_S, описываемую временны́м словом. Прямое сравнение двух временны́х слов, w_I и w_S, на равенство возможно только для *потактово точной* спецификации. Как правило, спецификация абстрактнее реализации, особенно в отношении временны́х свойств. Для того чтобы использовать абстрактную спецификацию для проверки поведения реализации, в ее коде с помощью специальных средств задаются ограничения на порядок реакций и время их возникновения (рис. 1). Подробнее об этом говорится в разделе «Инструментальная поддержка и опыт применения».

Отношение соответствия. В дальнейшем мы будем называть временны́е трассы с тривиальным частичным порядком (порядком, заданным отношением равенства) *трассами выполнения*. Если $\Sigma = I \cup O$, то трассы выполнения над алфавитом I будем

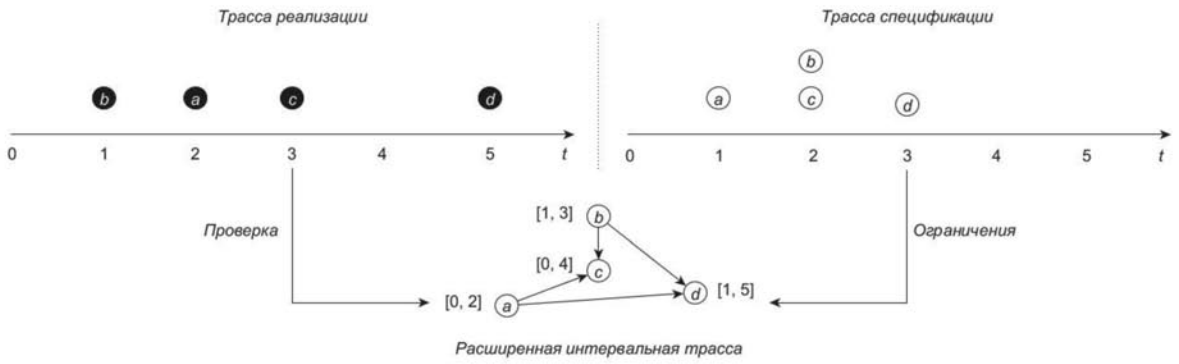

Рис. 1. Схема проверки соответствия между реализацией и спецификацией

называть *входными последовательностями*, а трассы выполнения над алфавитом O – *выходными последовательностями*. Отметим, что тривиальный частичный порядок в трассах выполнения отражает тот факт, что реализация рассматривается как «черный ящик», и причинно-следственные связи между ее событиями если и известны, то не от самой реализации. Множества входных и выходных последовательностей обозначаются $\mathbb{I}_\theta(\Sigma,\mathbb{T})$ и $\mathbb{O}_\theta(\Sigma,\mathbb{T})$ соответственно. Для краткости будем использовать сокращенные обозначения: $\mathbb{I}=\mathbb{I}_\theta(\Sigma,\mathbb{T})$ и $\mathbb{O}=\mathbb{O}_\theta(\Sigma,\mathbb{T})$.

Определение 5 (Поведение). Детерминированным поведением над алфавитом $\Sigma = I \cup O$ и временной областью $\mathbb{T}$ называется (частичная) функция $\mathcal{B}:\mathbb{I}\times\mathbb{T}\to\mathbb{O}$, удовлетворяющая следующим ограничениям:

1) для всех $w\in\mathbb{I}$ и $t\in\mathbb{T}$ выполняется неравенство $end(\mathcal{B}(w,t))\le t$;

2) для всех $w\in\mathbb{I}$ и $t\in\mathbb{T}$ справедливо равенство $\mathcal{B}(w,t)=\mathcal{B}(w_{[0,t]},t)$. □

Поведение описывает, каким образом входная последовательность преобразуется в выходную последовательность, принимая во внимание момент времени, в который производится наблюдение.

Предположим, что у нас имеется исполнимая спецификация. Рассмотрим, как ее можно использовать для динамической проверки поведения реализации. Для этого расширим определение поведения, позволив спецификации возвращать расширенные интервальные трассы. Обозначим множество всех таких трасс символом $\mathbb{O}_{\theta\delta}=\mathbb{O}_{\theta\delta}(\Sigma,\mathbb{T})$. Таким образом, поведение спецификации – это функция $\mathcal{B}:\mathbb{I}\times\mathbb{T}\to\mathbb{O}_{\theta\delta}$, удовлетворяющая указанным в определении ограничениям (используемые обозначения имеют смысл и для расширенных интервальных трасс).

Пусть $\mathcal{I}$ и $\mathcal{S}$ – *поведение реализации* и *поведение спецификации* соответственно. Для заданной входной последовательности $w\in\mathbb{I}$ и момента времени $t\in\mathbb{T}$ рассмотрим выход реализации и спецификации: $\mathcal{I}(w,t)=\langle V_\mathcal{I},=,\lambda_\mathcal{I},\theta_\mathcal{I}\rangle$ и $\mathcal{S}(w,t)=\langle V_\mathcal{S},\prec_\mathcal{S},\lambda_\mathcal{S},\theta_\mathcal{S},\delta_\mathcal{S}\rangle$. Введем обозначения:

$$past_\mathcal{I}^{\Delta t}(w,t)=\{y\in\mathcal{I}(w,t)\mid\theta_\mathcal{I}(y)\le(t-\Delta t^-(y))\};$$

$$past_\mathcal{I}(w,t)=\{y\in\mathcal{I}(w,t)\mid\theta_\mathcal{I}(y)\le t\};$$

$$past_\mathcal{S}^{\Delta t}(w,t)=\{x\in\mathcal{S}(w,t)\mid\theta_\mathcal{S}(x)\le(t-\Delta t^+(x))\};$$

$$past_\mathcal{S}(w,t)=\{x\in\mathcal{S}(w,t)\mid\theta_\mathcal{S}(x)\le t\};$$

$$match(x,y)=(\lambda_\mathcal{I}(y)=\lambda_\mathcal{S}(x))\wedge(\theta_\mathcal{I}(y)\in\delta_\mathcal{S}(x)).$$

Заметим, что в определении $past_\mathcal{I}^{\Delta t}(w,t)$ используется функция Δt^-, заданная для выходной трассы спецификации (для трассы реализации, не являющейся расширенной интервальной трассой, функции $\Delta t^\pm$ не определены).

Определение 6 (Отношение соответствия). Говорят, что поведение реализации $\mathcal{I}$ соответствует поведению спецификации $\mathcal{S}$, если $dom\mathcal{I}=dom\mathcal{S}$ ($\mathcal{I}$ и $\mathcal{S}$ определены на одном множестве входных последовательностей) и для всех $w\in dom\mathcal{S}$ и $t\in\mathbb{T}$ существует бинарное

отношение $\mathcal{M}(w,t) \subseteq \{(x,y) \in past_{\mathcal{S}}(w,t) \times \times past_{\mathcal{I}}(w,t) \mid match(x,y)\}$, называемое сопоставлением, такое что:

1) $\mathcal{M}(w,t)$ взаимно однозначно;

2) для каждой реакции спецификации $x \in past_{\mathcal{S}}^{\Delta t}(w,t)$ существует реакция реализации $y \in past_{\mathcal{I}}(w,t)$, такая что $(x,y) \in \mathcal{M}(w,t)$;

3) для каждой реакции реализации $y \in past_{\mathcal{I}}^{\Delta t}(w,t)$ существует реакция спецификации $x \in past_{\mathcal{S}}(w,t)$, такая что $(x,y) \in \mathcal{M}(w,t)$;

4) для всех $(x,y),(x',y') \in \mathcal{M}(w,t)$ если $x \prec x'$ то $\theta_{\mathcal{I}}(y) \le \theta_{\mathcal{I}}(y')$.

Если для некоторых $w \in dom\mathcal{S}$ и $t \in \mathbb{T}$ сопоставления не существует, то говорят, что $\mathcal{I}$ не соответствует $\mathcal{S}$, при этом w называется контрпримером. □

Рис. 2 иллюстрирует определение отношения соответствия для некоторой входной последовательности (будучи неважной, она на рисунке не показана) и момента времени $t = 4$. Верхняя часть рисунка показывает реакции реализации (черные кружки с белыми надписями: *b*, *a* и *c*), нижняя – реакции спецификации (белые кружки с черными надписями: *a*, *b*, *c* и *d*). Обозначим вершины трассы (собственно, кружки) символами y_b, y_a и y_c (для реализации) и x_a, x_b, x_c и x_d (для спецификации). Между вершинами реализационной трассы нет причинно-следственных связей. Вершины спецификационной трассы частично упорядочены (непосредственное предшествование событий показано стрелками: $x_a \to x_c$, $x_b \to x_c$, $x_a \to x_d$ и $x_b \to x_d$) и помечены временными интервалами ($\delta(x_a) = [0,2]$, $\delta(x_b) = [1,3]$, $\delta(x_c) = [0,4]$ и $\delta(x_d) = [1,5]$). Сопоставление реакций отмечено пунктирными линиями ((x_a, y_a), (x_b, y_b) и (x_c, y_c)). Легко видеть, что изображенное отношение является сопоставлением (в смысле данного выше определения): (1) оно взаимно однозначно; (2 и 3) оно включает все реакции с истекшим «временем жизни»; (4) оно сохраняет спецификационный порядок: (a) $x_a \prec x_c$ и $\theta(y_a) = 2 \le 3 = \theta(y_c)$; (b) $x_b \prec x_c$ и $\theta(y_b) = 1 \le 3 = \theta(y_c)$. Безусловно, каждая пара этого отношения удовлетворяет условию сопоставления реакций: (a) $\lambda(x_a) = \lambda(y_a) = a$ и $\theta(y_a) = 2 \in [0,2] = \delta(x_a)$; (b) $\lambda(x_b) = \lambda(y_b) = b$ и $\theta(y_b) = 1 \in [1,3] = \delta(x_b)$; (c) $\lambda(x_c) = \lambda(y_c) = c$ и $\theta(y_c) = 3 \in [0,4] = \delta(x_c)$.

Динамическое сопоставление трасс. Монитор, осуществляющий сопоставление трасс реализации и спецификации, может быть представлен как *временной автомат* [10] с двумя типами входных портов: (1) порты для получения *спецификационных реакций* и (2) порты для получения *реали-*

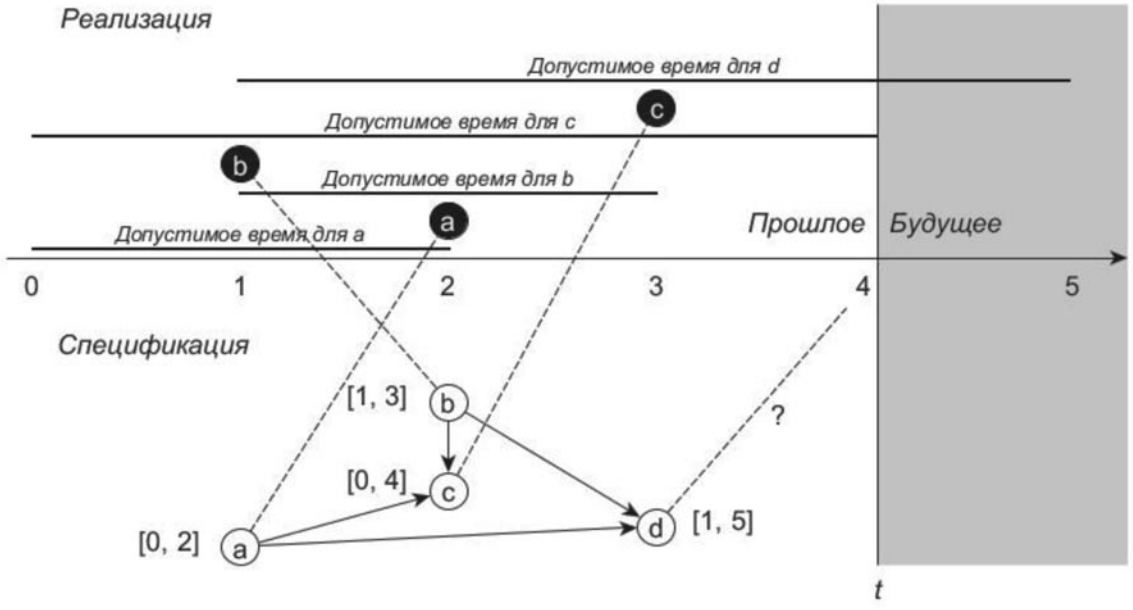

Рис. 2. Соответствие между реализацией и спецификацией

зационных реакций. Когда автомат обнаруживает расхождение в трассах реализации и спецификации, он переходит в определенное состояние, информируя, тем самым, что реализация не соответствует спецификации.

Ниже представлено формальное описание монитора в виде системы *действий с условиями* (guarded actions) [13]. Каждое *действие* атомарно и выполняется, как только соответствующее *условие* становится истинным. Действия и условия зависят от внешней переменной *t*, отражающей *текущее время*, и реакций, выдаваемых реализацией и спецификацией в ответ на одну и ту же последовательность стимулов (I и S соответственно). Значение переменной t монотонно возрастает в процессе верификации. Запись $y \in I[t]$ ($x \in S[t]$) означает, что в момент времени t реализация (спецификация) выдает реакцию $y(x)$. Пометка $[\uparrow \theta_{\mathcal{I}}(y)]$ в действии *onSpecOut* говорит о том, что перебор реакций $y \in past_I$ осуществляется в порядке возрастания $\theta_{\mathcal{I}}(y)$.

Пусть X – подмножество вершин трассы спецификации, $\prec$ – заданное на множестве X отношение частичного порядка, y – вершина трассы реализации. Введем следующие обозначения:

$$\min_{\prec}(X) = \{x \in X \mid \nexists x' \in X.\, x' \neq x \wedge x' \prec x\};$$

$$match(X, y) = \{x \in X \mid match(x, y)\}.$$

Кроме того, определим две функции, на которых основано описание монитора: *первичный арбитр* ($arbiter_S$) и *вторичный арбитр* ($arbiter_I$):

$$arbiter_S(X) = \begin{cases} \min_{\prec}(X) & \text{если } X \neq \varnothing, \\ \phi & \text{иначе } (\phi \notin X); \end{cases}$$

$$arbiter_I(X, y) = \begin{cases} \arg\min_{x \in match(X,y)} \{\theta_S(x)\} \\ \text{если } match(X, y) \neq \varnothing, \\ \phi \text{ иначе.} \end{cases}$$

Первичный арбитр для заданного множества X вершин спецификационной трассы возвращает его подмножество, состоящее из минимальных по отношению частичного порядка $\prec$ элементов. Вторичный арбитр сопоставляет вершину y реализационной трассы с вершинами спецификационной трассы из множества X.

Ниже приведен порядок проверки условий и запуска действий в момент времени t (при несоблюдении этого порядка возможны ложные сообщения об ошибках):

1) инициализация (*onInnitialize*);

2) прием реакции (*onSpecOut*, *onImplOut*);

3) превышение лимита времени (*onSpecTime*, *onImplTime*);

4) завершение (*onFinalize*).

Когда говорят, что некоторое свойство φ выполняется в момент времени t, имеется в виду, что φ выполняется после завершения всех действий, активированных в момент t. Для многопортовых систем (что типично для аппаратуры) монитор можно разбить на слабо связанные компоненты, параллельно обслуживающие отдельные порты.

Утверждение 1 (Корректность). Входная последовательность является контрпримером тогда и только тогда, когда монитор завершается с $verdict = false$.

Схема доказательства. Докажем, что если $verdict = false$, то w, входная последовательность, – контрпример. Согласно описанию монитора, имеются две возможности, когда переменной *verdict* присваивается значение *false*:

1) при выполнении действия *onSpecTime*;

2) при выполнении действия *onImplTime*.

В обоих случаях можно показать, что w – контрпример. Рассмотрим для примера случай 1. Пусть t – время завершения работы монитора (вызова оператора **terminate**). Выполнение действия *onSpecTime* в момент времени t подразумевает, что существует реакция спецификации $x \in past_S$, такая что $(\theta_S(x) + \Delta t^+(x)) \leq t$ (следовательно, $x \in past_S^{\Delta t}(w, t)$). Предположим, что существует сопоставление $\mathcal{M}$ (то есть w – не контрпример). Тогда найдется реакция реализации $y \in past_{\mathcal{I}}(w, t)$, такая что $(x, y) \in \mathcal{M}$ (см. п. 2 определения отношения соответствия). По какой-то причине эта реакция не была сопоставлена монитором с реакцией x. Не ограничивая общности рассуждений, положим, что реакция y пришла раньше реакции x. Возможны две причины, по которым эти реакции не были

Действие 1 *onSpecOut*$[x], x \in S[t]$
Если: *true*
Вход: x
$past_S \Leftarrow past_S \cup \{x\}$
if $x \in arbiter_S(past_S)$ **then**
  for all $y \in past_I$ [$\uparrow \theta_{\mathcal{I}}(y)$] **do**
    if x = arbiter$_I$({x}, y) **then**
      $past_S \Leftarrow past_S \setminus \{x\}$
      $past_I \Leftarrow past_I \setminus \{y\}$
      $match \Leftarrow match \cup \{(x, y)\}$
      break
    end if
  end for
end if

Действие 2 *onImplOut*$[y], y \in I[t]$
Если: *true*
Вход: y
$past_I \Leftarrow past_I \cup \{y\}$
$x \Leftarrow arbiter_I(arbiter_S(past_S), y)$
if $x \neq \phi$ **then**
  $past_S \Leftarrow past_S \setminus \{x\}$
  $past_I \Leftarrow past_I \setminus \{y\}$
  $match \Leftarrow match \cup \{(x, y)\}$
end if

Действие 3 *onSpecTime*$[x], x \in past_S$
Если: $\left(\theta_S(x) + \Delta t^+(x)\right) \leq t$
Вход: x
$past_S \Leftarrow past_S \setminus \{x\}$
verdict $\Leftarrow$ *false*
trace («Missing output»)
terminate

Действие 4 *onImplTime*$[y], y \in past_I$
Если: $\left(\theta_{\mathcal{I}}(y) + \Delta t^-(y)\right) \leq t$
Вход: y
$past_I \Leftarrow past_I \setminus \{y\}$
verdict $\Leftarrow$ *false*
trace («Unexpected output»)
terminate

Действие 5 *onInitialize*
Если $t = 0$
Вход: $\varnothing$
$past_S \Leftarrow \varnothing$
$past_I \Leftarrow \varnothing$
$match \Leftarrow \varnothing$

Действие 6 *onFinalize*
Если
$\mathbf{max}\left(\mathbf{end}(S) + \Delta T^+, \mathbf{end}(I) + \Delta T^-\right) \leq t$
Вход: $\varnothing$
verdict $\Leftarrow$ *true*
terminate

сопоставлены при выполнении действия *onSpecOut* для реакции x:

a. при получении реакции x было выполнено условие $x \notin arbiter_S(past_S)$ (см. описание действия *onSpecOut* и определение функции $arbiter_I$);

b. к этому моменту реакция y уже была сопоставлена с другой реакцией спецификации (обозначим эту реакцию x^*).

Рассмотрим случай *a*. Условие $x \notin arbiter_S(past_S)$ выполняется, только если существует реакция спецификации $x' \neq x,$, такая что $x' \prec x$ и $x' \in arbiter_S(past_S)$ (см. определение функции $arbiter_S$). Если таких реакций несколько, пусть x' – та из них, на которой достигается минимум функции θ_S. Пусть y' – реакция реализации, такая что $(x', y') \in \mathcal{M}$. Отметим, что реакция y' пришла не позже реакции y (это следует из п. 4 определения отношения соответствия), и она не была сопоставлена с реакцией x' (в противном случае в момент прихода реакции x было бы выполнено условие $x' \notin past_S$ и, как следствие, $x' \notin arbiter_S(past_S)$). Заметим, что решение о сопоставлении реакции x' с одной из

реакций реализации должно быть принято не позднее момента времени t, даже если $(\theta_S(x') + \Delta t^+(x')) > t$: поскольку $x' \prec x$, то

$$\theta_I(y'^*) \le \theta_I(y^*) \le (\theta_S(x) + \Delta t^+(x)) \le t,$$

где y^* и y'^* — произвольные реакции реализации, которые могут быть сопоставлены с x и x' соответственно. Причиной, по которой реакции x' и y' не были сопоставлены, может быть только b. Таким образом, случай a для пары реакций (x, y) сводится к случаю b для пары реакций (x, y).

Рассмотрим теперь случай b. Пусть N_I — общее число реакций реализации, произошедших до момента времени t включительно. Очевидно, что реакция x^* (с которой была сопоставлена реакция y) произошла не позже реакции x. Пусть y^* — реакция реализации, такая что $(x^*, y^*) \in \mathcal{M}$. Отметим, что реакция y^* произошла не раньше реакции y (в противном случае, поскольку в действии *onSpecOut* реакции реализации рассматриваются в порядке возрастания времени их прихода, реакция x^* сопоставилась бы с реакцией y^* или иной реакцией, отличной от y, которая произошла раньше). Покажем, что выполнено условие $match(x, y^*)$:

$$\lambda_I(y^*) = \lambda_S(x^*) = \lambda_I(y) = \lambda_S(x);$$

$$(\lambda_S(x^*) = \lambda_S(x)) \Rightarrow (\Delta t^+(x^*) = \Delta t^+(x)) \Rightarrow$$
$$(\theta_S(x) - \Delta t^-(x)) \le \theta_I(y) \le \theta_I(y^*) \le$$
$$\le (\theta_S(x^*) + \Delta t^+(x^*)) \le (\theta_S(x) + \Delta t^+(x)).$$

Тем не менее реакция y^*, как и реакция y, не была сопоставлена монитором с реакцией x. Применяя аналогичные рассуждения в отношении (x, y^*) и последующих пар реакций, получим, что общее число реакций реализации не ограничено числом N_I, что противоречит определению этого числа. Следовательно, сопоставления не существует, а значит, w — контрпример, что и требовалось доказать.

Теперь докажем, что если w — контрпример, то монитор завершается с $verdict = false$. Предположим противное, т. е. то, что монитор не завершается с $verdict = false$ (это равносильно невыполнению действий *onSpecTime* и *onImplTime*). Рассмотрим произвольный момент времени t, не превосходящий время завершения работы монитора (если он завершается). Покажем, что отношение *match*, построенное к моменту t, является сопоставлением (выполнены свойства 1—4 из определения отношения соответствия). Во-первых, *match* — взаимно однозначное отношение (поскольку при каждом обновлении *match* используются только новые, не принадлежащие *match*, реакции). Во-вторых и в-третьих, *match* включает все реакции спецификации $x \in past_S^{\Delta t}(w, t)$ и все реакции реализации $y \in past_I^{\Delta t}(w, t)$ (в противном случае выполнилось бы одно из действий *onSpecTime* или *onImplTime*, что привело бы к завершению монитора с $verdict = false$). В-четвертых, если $(x, y), (x', y') \in match$ и $x \prec x'$, то $\theta_I(y) \le \theta_I(y')$ (если $x \prec x'$ и $x \ne x'$, то сопоставление пары реакций (x, y) осуществляется не позже сопоставления (x', y'): условие $x' \in arbiter_S(past_S)$ может быть выполнено только после того, как реакция x будет удалена из множества $past_S$). Из существования сопоставления вытекает, что w — не контрпример, что противоречит условию утверждения. Следовательно, монитор завершается, причем $verdict = false$, что и требовалось доказать. □

На практике встречается аппаратура, в которой операции в некоторых ситуациях *отменяют* другие операции (конфликтующие с ними и имеющие более низкий приоритет). Например, операция записи может быть отменена другой операцией записи, адресующейся к той же ячейке памяти и запущенной сразу после рассматриваемой операции. Из-за абстрактности спецификации в ее терминах не всегда возможно выразить условия, при которых происходит отмена операций и соответствующие реакции не посылаются наружу. Принимая во внимание сказанное выше, определение спецификационного поведения может быть расширено: каждая спецификационная реакция дополнительно помечается признаком возможной отмены. Предложенный подход к организации мониторов может быть перенесен и на этот случай. Следует, однако, учесть, что если некоторое событие отменяется, то также отменяются все зависимые от него события.

Обзор существующих работ

В работе [14] используется модель *автомата с частично упорядоченными входными/выходными событиями* (Partial Order Input/Output Automaton – POIOA) для представления поведения спецификации и реализации. В статье предложен метод построения тестового набора, гарантирующего обнаружение ошибок определенного типа. Если (1) реализация сообщает о приеме неподдерживаемых стимулов, (2) можно установить порядок выдачи реакций, (3) время ответа реализации ограничено, и (4) каждый единичный переход в спецификации соответствует единичному переходу в реализации, можно определить соответствие между реализацией и спецификацией. Реализация соответствует спецификации, если она принимает допускаемые спецификацией стимулы и выдает описываемые спецификацией реакции в допустимом порядке. Определение отношения соответствия, данное в этой статье, близко к используемому нами. Главными отличиями нашего подхода являются поддержка необязательных реакций и контроль временны́х интервалов.

Статья [15] описывает подход к проверке поведения временны́х систем, основанный на *инвариантах трассы*. Рассматриваются два типа инвариантов: (1) *инварианты ожидания*, выражающие то свойство, что после заданной трассы всегда ожидается определенное событие (в определенном временно́м интервале), и (2) *инварианты наблюдения*, утверждающие, что между двумя заданными событиями всегда наблюдается определенная последовательность событий. Корректность поведения реализации проверяется в два этапа: (1) проверка соответствия инвариантов спецификации, выраженной в форме *временно́го конечного автомата* (Timed Finite State Machine – TFSM); (2) проверка выполнимости инвариантов для трассы реализации. Подход представляет теоретический интерес, однако его промышленное использование может быть затруднено необходимостью постоянного согласования проверяемых инвариантов со спецификацией.

В работе [16] рассматривается метод тестирования параллельных систем с помощью неявно заданных *асинхронных конечных автоматов* (Asynchronous Finite State Machines – AFSM). Поведение реализации проверяется только в *стационарных состояниях*, в которых не ожидается выдача реакций. Используется следующая процедура: (1) собираются все события и определяется их частичный порядок; (2) строятся и проверяются все возможные линеаризации множества событий (проверка базируется на пред- и постусловиях, определенных для каждого события). Считается, что реализация соответствует спецификации, если допускается хотя бы одна из построенных линеаризаций. Применение подхода возможно только для ограниченного класса входных последовательностей: требуется, чтобы регулярно возникали стационарные состояния. В нашем случае такого ограничения нет.

Инструментальная поддержка и опыт применения

Предложенный метод к проверке корректности поведения HDL-моделей реализован в инструменте C++TESK [17]. Библиотека инструмента содержит классы и макросы для создания компонентов систем динамической верификации аппаратуры (эталонных моделей, мониторов, генераторов стимулов и др.). Возможности C++TESK для разработки эталонных моделей аппаратуры (и, соответственно, мониторов) включают средства для посылки и приема пакетов данных (примитивы *send* и *receive*), ветвления и объединения параллельных процессов (*fork* и *join*), моделирования временны́х задержек (*delay*) и задания зависимостей между пакетами (*depends*).

Ниже приведены описания некоторых примитивов C++TESK, наиболее важных для затрагиваемой в статье темы (для наглядности их синтаксис несколько отличается от используемого в инструменте):

- *delay*(*n*) – моделирует временну́ю задержку (выполнение процесса прерывается, а возобновление работы планируется через *n* единиц времени);
- *receive*(*in*) : *pkg* – ждет до тех пор, пока

входной пакет не будет получен на входном порту (*in*), а затем возвращает этот пакет (*pkg*);

• *send*(*out*, *pkg*, *opt*) – посылает выходной пакет (*pkg*) через выходной порт (*out*), указывая, является ли данный пакет обязательным или опциональным (*opt*);

• *depends*(*pkg*1, *pkg*2) – указывает, что выходной пакет (*pkg*1) зависит по данным или связан причинно-следственной связью с другим пакетом (*pkg*2), входным или выходным.

Заметим, что каждый раз, когда пакет данных выдается наружу с помощью примитива *send*, он помечается временны́м интервалом $[t - \Delta t^{-}, t + \Delta t^{+}]$, где t – время начала посылки, а $\Delta t^{\pm}$ – пользовательские параметры выходного порта. Для каждого порта также задается режим работы: *fifo* (режим по умолчанию) или *unordered*. В режиме *fifo* при посылке пакета добавляется зависимость этого пакета от последнего пакета, переданного через тот же выходной порт и находящегося в очереди ожидания (еще не сопоставленного с пакетом реализации), в режиме *unordered* никакие зависимости не добавляются. Дополнительные ограничения на порядок пакетов, если они нужны, задаются с помощью примитива *depends*.

Инструмент позволяет создавать модели аппаратуры на разных уровнях абстракции (относительно точности моделирования времени): (1) *модели без учета времени* (описывающие общие причинно-следственные связи между пакетами данных без моделирования временны́х задержек между ними: $\Delta t^{\pm} = \infty$), (2) *модели с приближенным учетом времени* (частично специфицирующие внутренние схемы арбитража пакетов, но учитывающие время лишь примерно: $\Delta t^{\pm} \le T$, где T имеет значение нескольких десятков тактов) и (3) *потактовые модели* (реализующие точное или почти точное моделирование времени: $\Delta t^{\pm} \le 1$).

Инструмент C++TESK использован для динамической верификации моду-

Опыт применения предложенного метода

Название модуля	Стадия разработки	Точность разработки	
		от	до
Буфер трансляции адресов	Поздняя/ завершающая	Приближенная	Потактовая
Модуль арифметики (FPU)	Поздняя/ завершающая	Без учета времени	–
Кэш-память 2-го уровня (L2)	Промежуточная/ поздняя	Приближенная	–
Коммутатор северного моста	Промежуточная/ завершающая	Приближенная	Потактовая
Модуль доступа к памяти	Ранняя/ промежуточная	Без учета времени	Потактовая
Контроллер прерываний	Ранняя/ промежуточная	Без учета времени	Приближенная
Модуль поиска по таблице страниц	Поздняя	Приближенная	–
Контроллер банка кэш-памяти L2	Поздняя	Потактовая	–
Буфер команд	Поздняя/ завершающая	Потактовая	–
Кэш-память 3-го уровня (L3)	Промежуточная	Приближенная	–

лей промышленных микропроцессоров, разрабатываемых в НИИСИ РАН и ЗАО «МЦСТ». Наш опыт уже был описан в [18], но с тех пор он расширился. Последняя информация о применении инструмента и заложенного в нем метода представлена в таблице. Как видно из нее, подход поддерживает верификацию как с помощью абстрактных эталонных моделей (доступных на ранних стадиях проектирования), так и с помощью потактово точных моделей (доступных на заключительных этапах). Что важно, подход позволяет использовать для верификации C++-модели, изначально создаваемые для других целей (в частности, компоненты системного симулятора микропроцессора). Таким способом, например, был проверен модуль поиска по таблице страниц.

Статья сфокусирована на использовании исполнимых моделей для динамической верификации HDL-моделей. Проблема не так проста, как кажется на первый взгляд. Проверка того, что реализация и спецификация выдают одинаковые реакции в одинаковые моменты времени в большинстве случаев не является адекватной. Спецификация в силу своей абстрактности может не учитывать многих особенностей реализации, таких как упорядочивание событий и их распределение во времени. В работе предложены механизмы, позволяющие настраивать точность проверки поведения, учитывая степень абстрактности спецификации. К ним относятся: (1) описание отношения зависимости событий, (2) расширение временны́х меток событий до временны́х интервалов и (3) пометка некоторых событий как необязательных. Основываясь на предложенных механизмах, разработан метод проверки поведения HDL-моделей. Метод был реализован в инструменте C++TESK и применялся в более чем 10 проектах по верификации модулей микропроцессоров. Наши дальнейшие исследования связаны с диагностикой ошибочного поведения HDL-моделей на основе более глубокого анализа трасс, выполняемого после сеанса верификации. Целью такого анализа является упрощение локализации ошибок путем определения характера расхождений наблюдаемого поведения HDL-модели от эталонного.

СПИСОК ЛИТЕРАТУРЫ

1. **Wile B., Goss J., Roesner W.** Comprehensive Functional Verification: The Complete Industry Cycle. Morgan Kaufmann, 2005.

2. **Bergeron J.** Writing Testbenches: Functional Verification of HDL Models. Kluwer Academic, 2000.

3. **Glasser M.** Open Verification Methodology Cookbook. Springer, 2009.

4. **Sen K., Rosu G.** Generating Optimal Monitors for Extended Regular Expressions // Electronic Notes in Theoretical Computer Science. 2003. No. 89(2). Pp. 162–181.

5. **Ivannikov V., Kamkin A., Kossatchev A., Kuliamin V., Petrenko A.** The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models // Programming and Computer Software. 2007. No. 33(5). Pp. 272–282.

6. **Barringer H., Rydeheard D., Havelund K.** Rule Systems for Run-Time Monitoring: From Eagle to RuleR // In Proc. of 7th Internat. Workshop on Runtime Verification. Revised Selected Papers. 2007. Pp. 111–125.

7. **Bauer A., Leucker M., Schallhart C.** Runtime Verification for LTL and TLTL // ACM Transactions on Software Engineering and Methodology. 2011. No. 20(4). Pp. 14:1–14:64.

8. **Mintz M., Ekendahl R.** Hardware Verification with C++: A Practitioner's Handbook. Springer Science+Business Media, LLC. 2006.

9. **Pratt V.** The Pomset Model of Parallel Processes: Unifying the Temporal and the Spatial // In Seminar on Concurrency. 1984. Pp. 180–196.

10. **Alur R., Dill D.** A Theory of Timed Automata // Theoretical Computer Science. 1994. No. 126(2). Pp. 183–235.

11. **Mazurkiewicz A.** Trace Theory // In Advances in Petri Nets 1986, Part II on Petri Nets: Applications and Relationships to Other Models of Concurrency. New York: Springer-Verlag, 1987. Pp. 279–324.

12. **Chieu D., Hung D.** Timed Traces and Their Applications in Specification and Verification of Distributed Real-time Systems // In Proc. of the 3rd Symp. on Information and Communication

Technology. 2012. Pp. 31–40.

13. **Dijkstra E.W.** Guarded Commands, Nondeterminacy and Formal Derivation of Programs // Communication of the ACM. 1975. No. 18(8). Pp. 453–457.

14. **von Bochmann G., Haar S., Jard C., Jourdan G.V.** Testing Systems Specified as Partial Order Input/Output Automata // In Proc. of the 20th IFIP TC 6/WG 6.1 Internat. Conf. on Testing of Software and Communicating Systems: 8th Internat. Workshop. 2008. Pp. 169–183.

15. **Andres C., Merayo M., Nunez M.** Formal Passive Testing of Timed Systems: Theory and Tools // Software Testing, Verification & Reliability. 2012. No. 22(6). Pp. 365–405.

16. **Kuliamin V., Petrenko A., Pakoulin N., Kossatchev A., Bourdonov I.** Integration of Functional and Timed Testing of Real-Time and Concurrent Systems // In Perspectives of System Informatics. 2003. Pp. 450–461.

17. ISPRAS: C++TESK Homepage [электронный ресурс] / URL: http://forge.ispras.ru/projects/cpptesk-toolkit/

18. **Chupilko M., Kamkin A.** A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels // In Proc. of the 12th Latin-American Test Workshop. 2011. Pp. 1–6.

REFERENCES

1. **Wile B., Goss J., Roesner W.** *Comprehensive Functional Verification: The Complete Industry Cycle*, Morgan Kaufmann, 2005.

2. **Bergeron J.** *Writing Testbenches: Functional Verification of HDL Models*, Kluwer Academic, 2000.

3. **Glasser M.** *Open Verification Methodology Cookbook*, Springer, 2009.

4. **Sen K., Rosu G.** Generating Optimal Monitors for Extended Regular Expressions, *Electronic Notes in Theoretical Computer Science*, 2003, No. 89(2), Pp. 162– 181.

5. **Ivannikov V., Kamkin A., Kossatchev A., Kuliamin V., Petrenko A.** The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models, *Programming and Computer Software*, 2007, No. 33(5), Pp. 272–282.

6. **Barringer H., Rydeheard D., Havelund K.** Rule Systems for Run-Time Monitoring: From Eagle to RuleR, *In Proceedings of 7th International Workshop on Runtime Verification. Revised Selected Papers*, 2007, Pp. 111–125.

7. **Bauer A., Leucker M., Schallhart C.** Runtime Verification for LTL and TLTL, *ACM Transactions on Software Engineering and Methodology*, 2011, No. 20(4), Pp. 14:1–14:64.

8. **Mintz M., Ekendahl R.** *Hardware Verification with C++: A Practitioner's Handbook*, Springer Science+Business Media, LLC, 2006.

9. **Pratt V.** The Pomset Model of Parallel Processes: Unifying the Temporal and the Spatial, *In Seminar on Concurrency*, 1984, Pp. 180–196.

10. **Alur R., Dill D.** A Theory of Timed Automata, *Theoretical Computer Science*, 1994, No. 126(2), Pp. 183–235.

11. **Mazurkiewicz A.** Trace Theory, *In Advances in Petri Nets 1986, Part II on Petri Nets: Applications and Relationships to Other Models of Concurrency*, New York, USA, Springer-Verlag, 1987, Pp. 279–324.

12. **Chieu D., Hung D.** Timed Traces and Their Applications in Specification and Verification of Distributed Real-time Systems, *In Proceedings of the 3rd Symposium on Information and Communication Technology*, 2012, Pp. 31–40.

13. **Dijkstra E.W.** Guarded Commands, Nondeterminacy and Formal Derivation of Programs, *Communication of the ACM*, 1975, No. 18(8), Pp. 453–457.

14. **von Bochmann G., Haar S., Jard C., Jourdan G.V.** Testing Systems Specified as Partial Order Input/Output Automata, *In Proceedings of the 20th IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems: 8th International Workshop*, 2008, Pp. 169–183.

15. **Andres C., Merayo M., Nunez M.** Formal Passive Testing of Timed Systems: Theory and Tools, *Software Testing, Verification & Reliability*, 2012, No. 22(6), Pp. 365–405.

16. **Kuliamin V., Petrenko A., Pakoulin N., Kossatchev A., Bourdonov I.** Integration of Functional and Timed Testing of Real-Time and Concurrent Systems, *In Perspectives of System Informatics*, 2003, Pp. 450–461.

17. ISPRAS: C++TESK Homepage. Available: http://forge.ispras.ru/projects/cpptesk-toolkit/

18. **Chupilko M., Kamkin A.** A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels, *In Proceedings of the 12th Latin-American Test Workshop*, 2011, Pp. 1–6.

ИВАННИКОВ Виктор Петрович – *директор Института системного программирования РАН.*
109004, Россия, Москва, ул. Александра Солженицына, д. 25.
E-mail: ivan@ispras.ru

IVANNIKOV, Viktor P. *Institute for System Programming of the Russian Academy of Sciences.*
109004, Alexander Solzhenitsyn Str. 25, Moscow, Russia.
E-mail: ivan@ispras.ru

КАМКИН Александр Сергеевич – *старший научный сотрудник отдела технологий программирования Института системного программирования РАН.*
109004, Россия, Москва, ул. Александра Солженицына, д. 25.
E-mail: kamkin@ispras.ru

KAMKIN, Alexander S. *Institute for System Programming of the Russian Academy of Sciences.*
109004, Alexander Solzhenitsyn Str. 25, Moscow, Russia.
E-mail: kamkin@ispras.ru

ЧУПИЛКО Михаил Михайлович – *научный сотрудник отдела технологий программирования Института системного программирования РАН.*
109004, Россия, Москва, ул. Александра Солженицына, д. 25.
E-mail: chupilko@ispras.ru

CHUPILKO, Mikhail M. *Institute for System Programming of the Russian Academy of Sciences.*
109004, Alexander Solzhenitsyn Str. 25, Moscow, Russia.
E-mail: chupilko@ispras.ru