Проверка корректности поведения HDL-моделей цифровой аппаратуры на основе динамического сопоставления трасс

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

Федеральное государственное бюджетное учреждение науки
Институт системного программирования Российской академии наук (ИСП РАН),
109004, г. Москва, ул. Александра Солженицына, д. 25.
{ivan,kamkin,chupilko}@ispras.ru

Аннотация 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. В работе рассматривается метод динамического сопоставления трасс для моделей аппаратуры разного уровня абстракции. Метод был успешно применен в нескольких промышленных проектах по верификации модулей микропроцессоров.

1 Введение

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* (*HDL*, *Hardware Description Languages*), 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*, которые перехватывают события на входах и выходах (*стимулы* и *реакции*) и проверяют допустимость получаемой трассы [3].

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

Оставшаяся часть статьи организована следующим образом. В разделе 2 вводятся основные понятия и обозначения. Раздел 3 описывает предлагаемый метод динамического сопоставления трасс: в этом разделе определяется отношение соответствия между реализацией и спецификацией и рассматривается устройство монитора, проверяющего соответствие. В разделе 4 описывается опыт использования предлагаемого подхода для верификации модулей микропроцессоров. Раздел 5 содержит краткий обзор работ близких к нашей. В разделе 6 дается заключение и указываются направления дальнейших исследований.

2 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. Если u и v — два слова над одним

алфавитом и u конечно, то uv обозначает их *конкатенацию*. Для $w = uv$ говорят, что w является *продолжением* u с помощью v.

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*: $\mathsf{port} : \Sigma \to \{1, 2, ..., k\}$. 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 последовательность $(a_0, t_0)(a_1, t_1)...$ временны́х событий $(a_i, t_i) \in \Sigma \times \mathbb{T}$, удовлетворяющая следующим ограничениям:*

1. *для каждого $i \geq 0$ выполняется неравенство $t_i \leq t_{i+1}$;*
2. *для любого $T \in \mathbb{T}$ найдется $i \geq 0$, такой что $t_i > T$ (если w бесконечна);*
3. *для всех $i, j \geq 0$, таких что $i \neq j$ и $t_i = t_j$, $\mathsf{port}(e_i) \neq \mathsf{port}(e_j)$.* □

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* (*Mazurkiewicz*) [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 $\langle V, \preceq, \lambda \rangle$, где V — конечное множество вершин и $\lambda : V \to \Sigma$ — функция разметки. Два Σ-размеченных частичных порядка называются эквивалентными, если они изоморфны относительно $\preceq$ и λ (совпадают или отличаются 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 $\langle V, \preceq, \lambda \rangle$ называется размеченный полный порядок $\langle V, \leq, \lambda \rangle$, где $\preceq \subseteq \leq$.

Определение 3 (Временна́я трасса [12]) *Временно́й трассой над алфавитом Σ и временно́й областью $\mathbb{T}$ называется четверка $\langle V, \preceq, \lambda, \theta \rangle$, где $\langle V, \preceq, \lambda \rangle$ — частично упорядоченное мультимножество, а $\theta : V \to \mathbb{T}$ — функция времени, удовлетворяющая следующим условиям:*

1. *для всех* $x, y \in V$, *из* $x \prec y$ *вытекает* $\theta(x) < \theta(y)$;
2. *для любого* $t \in \mathbb{T}$ *существует срез* $C \subseteq V$, *такой что* $\mathsf{min}_{x \in C}\{\theta(x)\} \geq t$ *(если* V *бесконечно).* $\square$

Множество всех трасс над алфавитом Σ и временно́й областью $\mathbb{T}$ обозначается $\mathbb{M}_\theta(\Sigma, \mathbb{T})$. Заметим, что временны́е слова являются частным случаем временны́х трасс. Для заданной непустой трассы $\sigma = \langle V, \preceq, \lambda, \theta \rangle$ введем обозначения: $\mathsf{begin}(\sigma) = \mathsf{min}_{x \in V}\{\theta(x)\}$ и $\mathsf{end}(\sigma) = \mathsf{max}_{x \in V}\{\theta(x)\}$ (если σ бесконечна, $\mathsf{end}(\sigma) = \infty$); $\sigma_{[t,t+\Delta t]}$ — подтрасса трассы σ, состоящая из тех $x \in V$, для которых $\theta(x) \in [t, t + \Delta t]$. Пусть $\mathfrak{TI}(\mathbb{T})$ — множество временны́х интервалов во временно́й области $\mathbb{T}$ (то есть $\mathfrak{TI}(\mathbb{T}) = \{[t, t + \Delta t] \mid t, t + \Delta t \in \mathbb{T}\}$).

Определение 4 (Интервальная трасса) *Интервальной трассой над алфавитом* Σ *и временно́й областью* $\mathbb{T}$ *называется четверка* $\sigma = \langle V, \preceq, \lambda, \delta \rangle$, *где* $\langle V, \preceq, \lambda \rangle$ *— частично упорядоченное мультимножество, а* $\delta : V \to \mathfrak{TI}(\mathbb{T})$ *— функция, ассоциирующая с каждой вершиной временной интервал. Языком интервальной трассы* σ *является множество* $\mathfrak{L}(\sigma) = \{\langle V, \preceq, \lambda, \theta \rangle \in \mathbb{M}_\theta(\Sigma, \mathbb{T}) \mid \forall x \in V \;.\; \theta(x) \in \delta(x)\}$. $\square$

Множество всех интервальных трасс над алфавитом Σ и временно́й областью $\mathbb{T}$ обозначается $\mathbb{M}_\delta(\Sigma, \mathbb{T})$. В дальнейшем мы будем иметь дело с парами трасс $\langle \sigma_\theta, \sigma_\delta \rangle$, где σ_θ — временна́я трасса, а σ_δ — интервальная трасса, такие что $\sigma_\theta \in \mathfrak{L}(\sigma_\delta)$. Каждая такая пара описывается пятеркой $\langle V, \preceq, \lambda, \theta, \delta \rangle$ и называется *расширенной интервальной трассой*. Множество всех расширенных интервальных трасс обозначается $\mathbb{M}_{\theta\delta}(\Sigma, \mathbb{T})$.

3 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 $w_I \in (\Sigma \times \mathbb{T})^*$ соответствует трассе спецификации $\sigma_S \in \mathbb{M}_{\theta\delta}(\Sigma, \mathbb{T})$. Поясним, откуда берется расширенная интервальная трасса σ_S. 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. Сведения о степени абстрактности спецификации позволяют обобщить конкретное временно́е слово w_S до расширенной интервальной трассы σ_S, смягчая тем самым проверку соответствия между реализацией и спецификацией (см. Рисунок 1).

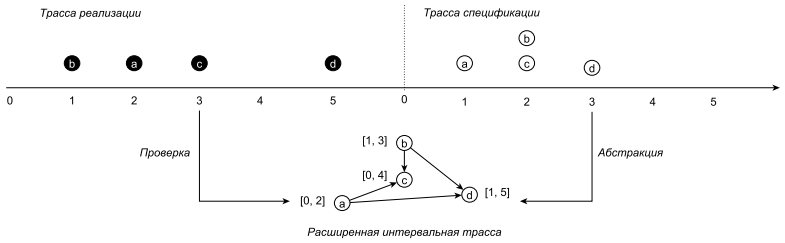

Рис. 1. Схема проверки соответствия между реализацией и спецификацией

3.1 Отношение соответствия

В дальнейшем мы будем называть трассы с тривиальным частичным порядком (порядком, заданным отношением равенства) *трассами выполнения*. Если $\Sigma = I \cup O$, то трассы выполнения над алфавитом I будем 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 сокрашенные 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 временно́й областью $\mathbb{T}$ называется (частичное) отображение $\mathcal{B} : \mathbb{I} \times \mathbb{T} \to \mathbb{O}$, удовлетворяющее следующим ограничениям:*

- *для всех $w \in \mathbb{I}$ и $t \in \mathbb{T}$ выполняется неравенство $\mathsf{end}\big(\mathcal{B}(w,t)\big) \le t$;*
- *для всех $w \in \mathbb{I}$ и $t \in \mathbb{T}$ справедливо равенство $\mathcal{B}(w,t) = \mathcal{B}(w_{[0,t]},t)$;*
- *для всех $w \in \mathbb{I}$ и $t \in \mathbb{T}$ существует $wv \in \mathbb{I}$, продолжение w, и $\Delta t \ge 0$, такие что $\mathsf{end}\big(\mathcal{B}(wv, t+\Delta t)\big) \ge t$.* □

Поведение описывает, каким образом входная последовательность преобразуется в выходную последовательность, принимая во внимание момент времени, в которое производится наблюдение. Предположим, что у нас есть исполнимая спецификация. Рассмотрим, как ее можно использовать для динамической проверки поведения реализации. Расширим определение поведения, позволив спецификации возвращать расширенные интервальные трассы, а не конкретные последовательности, как это требуется. Обозначим множество всех таких трасс символом $\mathbb{O}_{\theta\delta}$.

Для заданной выходной трассы $\langle V, \preceq, \lambda, \theta, \delta\rangle \in \mathbb{O}_{\theta\delta}$, определим две функции, $\Delta t^{\pm}$, такие что для всех $x \in V$, $\delta(x) = [\theta(x) - \Delta t^{-}(x), \theta(x) + \Delta t^{+}(x)]$. Будем полагать, что функции $\Delta t^{\pm}$ ограничены (существуют константы $\Delta T^{\pm} > 0$, такие что $|\Delta t^{\pm}(x)| \le \Delta T^{\pm}$ для всех $x \in V$). Также положим, что значения

$\Delta t^{\pm}(x)$ зависят от события, а не от вершины: $\Delta t^{\pm}(x) = \Delta t^{\pm}(\lambda(x))$. Пусть $\mathfrak{I}$ и $\mathfrak{S}$ — *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: $\mathfrak{I}(w,t) = \langle V_{\mathfrak{I}}, \varnothing, \lambda_{\mathfrak{I}}, \theta_{\mathfrak{I}} \rangle$ и $\mathfrak{S}(w,t) = \langle V_{\mathfrak{S}}, \preceq_{\mathfrak{S}}, \lambda_{\mathfrak{S}}, \theta_{\mathfrak{S}}, \delta_{\mathfrak{S}} \rangle$. Введем обозначения:

$$
\begin{aligned}
\mathsf{past}^{\Delta t}_{\mathfrak{I}}(w,t) &= \{y \in \mathfrak{I}(w,t) \mid \theta_{\mathfrak{I}}(y) \le \big(t - \Delta t^{-}(y)\big)\};\\
\mathsf{past}_{\mathfrak{I}}(w,t) &= \{y \in \mathfrak{I}(w,t) \mid \theta_{\mathfrak{I}}(y) \le t\};\\
\mathsf{past}^{\Delta t}_{\mathfrak{S}}(w,t) &= \{x \in \mathfrak{S}(w,t) \mid \theta_{\mathfrak{S}}(x) \le \big(t - \Delta t^{+}(x)\big)\};\\
\mathsf{past}_{\mathfrak{S}}(w,t) &= \{x \in \mathfrak{S}(w,t) \mid \theta_{\mathfrak{S}}(x) \le t\};\\
\mathsf{match}(x,y) &= \big(\lambda_{\mathfrak{I}}(y) = \lambda_{\mathfrak{S}}(x)\big) \wedge \big(\theta_{\mathfrak{I}}(y) \in \delta_{\mathfrak{S}}(x)\big).
\end{aligned}
$$

Определение 6 (Отношение соответствия) *Говорят, что поведение реализации $\mathfrak{I}$ соответствует поведению спецификации $\mathfrak{S}$, если $\mathsf{dom}\mathfrak{I} = \mathsf{dom}\mathfrak{S}$ и для всех $w \in \mathsf{dom}\mathfrak{S}$ и $t \in \mathbb{T}$ существует бинарное отношение $\mathcal{M}(w,t) \subseteq \{(x,y) \in \mathsf{past}_{\mathfrak{S}}(w,t) \times \mathsf{past}_{\mathfrak{I}}(w,t) \mid \mathsf{match}(x,y)\}$ (называемое сопоставлением), такое что:*

1. *$\mathcal{M}(w,t)$ взаимно однозначно;*
2. *для каждой реакции спецификации $x \in \mathsf{past}^{\Delta t}_{\mathfrak{S}}(w,t)$ существует реакция реализации $y \in \mathsf{past}_{\mathfrak{I}}(w,t)$, такая что $(x,y) \in \mathcal{M}(w,t)$;*
3. *для каждой реакции реализации $y \in \mathsf{past}^{\Delta t}_{\mathfrak{I}}(w,t)$ существует реакция спецификации $x \in \mathsf{past}_{\mathfrak{S}}(w,t)$, 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 $\theta_{\mathfrak{I}}(y) \le \theta_{\mathfrak{I}}(y')$.*

Если для некоторых $w \in \mathbb{I}$ и $t \in \mathbb{T}$ вышеуказанные свойства нарушаются, то говорят, что $\mathfrak{I}$ не соответствует $\mathfrak{S}$, при этом $w_{[0,t]}$ называется контрпримером. □

Рисунок 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). Будем обозначать 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 (предшествование событий показано стрелками: $x_a \prec x_c$, $x_b \prec x_c$, $x_a \prec x_d$ и $x_b \prec x_d$) и помечены временны́ми 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 $\theta(y_c) = 3 \in [0,4] = \delta(x_c)$.

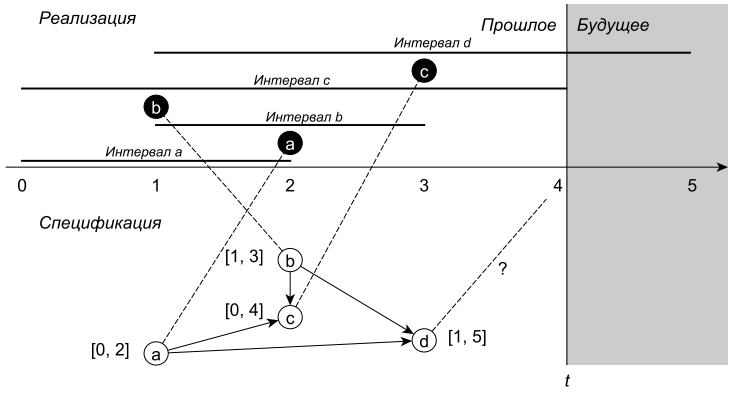

Рис. 2. Соответствие между реализацией и спецификацией

3.2 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 *реализационных 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*). 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 (S и I соответственно). Значение 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). Описание базируется на двух функциях: (1) *первичный арбитр* ($arbiter_S$) и (2) *вторичный арбитр* ($arbiter_I$), определенных ниже:

$$arbiter_S(X) = \begin{cases} \mathsf{min}_{\preceq}(X) & \text{если } X \neq \varnothing, \\ \phi & \text{иначе } (\phi \notin \Sigma); \end{cases}$$

$$arbiter_I(y, X) = \begin{cases} \mathsf{arg\ min}_{x \in X.\mathsf{match}(x,y)}\{\theta_{\mathbb{S}}(x)\} & \text{если } \exists x \in X \ .\ \mathsf{match}(x, y), \\ \phi & \text{иначе.} \end{cases}$$

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 $x = arbiter_I(y, \{x\})$ **then**
$past_S \Leftarrow past_S \setminus \{x\}$
$past_I \Leftarrow past_I \setminus \{y\}$
$match \Leftarrow match \cup \{\langle x, y\rangle\}$
break
end if
end for
end if

Действие 2 $onImplOut[y]$, $y \in I[t]$

Если: $true$
Вход: y
$past_I \Leftarrow past_I \cup \{y\}$
$x \Leftarrow arbiter_I(y, arbiter_S(past_S))$
if $x \neq \phi$ **then**
$past_S \Leftarrow past_S \setminus \{x\}$
$past_I \Leftarrow past_I \setminus \{y\}$
$match \Leftarrow match \cup \{\langle x, y\rangle\}$
end if

Действие 3 $onSpecTime[x], x \in past_S$

Если: $\big(\theta_{\mathcal{S}}(x) + \Delta t^+(x)\big) \leq t$
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: $\big(\theta_{\mathcal{I}}(y) + \Delta t^-(y)\big) \leq t$
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:
$\mathbf{max}\big(\mathbf{end}(S)+\Delta T^+, \mathbf{end}(I)+\Delta T^-\big) \leq t$
Вход: $\varnothing$
$verdict \Leftarrow true$
terminate

Ниже приведен порядок проверки условий и запуска действий внутри одного временно́го слота t (при несоблюдении этого порядка возможны ложные сообщения об ошибках):

1. инициализация ($onInitialize$);
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* $verdict = false$. □

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 вышесказанное, 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.

4 Инструментальная поддержка и опыт применения

Предложенный метод к проверке корректности поведения HDL-моделей был реализован в инструменте C++TESK [13]. Библиотека инструмента содержит классы и макросы для создания компонентов систем динамической верификации аппаратуры (эталонных моделей, мониторов, генераторов стимулов и других). 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 ($depends$).

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: $\Delta t^{\pm} \leq T$, где T имеет значение нескольких десятков тактов) и (3) *потактовые модели* (реализующие точное или почти точное моделирование времени: $\Delta t^{\pm} \leq 1$).

Инструмент C++TESK был использован для динамической верификации модулей промышленных микропроцессоров, разрабатываемых в НИИСИ РАН и ЗАО “МЦСТ”. Наш опыт уже был представлен в [14], но с тех пор он был расширен. Последняя информацию о применении инструмента и заложенного в нем метода представлена в Таблице 1. Как видно из таблицы, 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.

Название модуля	**Стадия разработки**	**Точность моделирования**	
		от	**до**
Буфер трансляции адресов	Поздняя / завершающая	Приближенная	Потактовая
Модуль арифметики (FPU)	Поздняя / завершающая	Без учета времени	—
Кэш-память 2-ого уровня (L2)	Промежуточная / поздняя	Приближенная	—
Коммутатор северного моста	Промежуточная / завершающая	Прилиженная	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-его уровня (L3)	Промежуточная	Приближенная	—

Таблица 1. Опыт применения предложенного метода

5 Обзор существующих работ

В работе [15] используется модель *автомата с частично упорядоченными входными/выходными событиями* (*POIOA*, *Partial Order Input/Output Automaton*) для представления поведения спецификации и реализации. В статье предлагается 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 [16] описывает подход к проверке поведения временны́х систем, основанный на *инвариантах трассы*. Рассматриваются два типа инвариан-

тов: (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 *временного конечного автомата* (*TFSM*, *Timed Finite State Machine*); (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 [17] рассматривается метод тестирования параллельных систем с помощью неявно заданных *асинхронных конечных автоматов* (*AFSM*, *Asynchronous Finite State Machines*). 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.

6 Заключение

Работа 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 **89**(2) (2003) 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 **33**(5) (2007) 272–282
6. Barringer, H., Rydeheard, D., Havelund, K.: Rule Systems for Run-Time Monitoring: From Eagle to RuleR. In: Proceedings of 7^{th} International Workshop on Runtime Verification. Revised Selected Papers. (2007) 111–125
7. Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology **20**(4) (2011) 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) 180–196
10. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science **126**(2) (1994) 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, NY, USA, Springer-Verlag New York, Inc. (1987) 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 Third Symposium on Information and Communication Technology. (2012) 31–40
13. ISPRAS: C++TESK Homepage. http://forge.ispras.ru/projects/cpptesk-toolkit/
14. Chupilko, M., Kamkin, A.: A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels. In: Proceedings of the 12^{th} Latin-American Test Workshop. (2011) 1–6
15. von Bochmann, G., S.Haar, C.Jard, Jourdan, G.V.: Testing Systems Specified as Partial Order Input/Output Automata. In: Proceedings of the 20^{th} IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems: 8^{th} International Workshop. TestCom '08 / FATES '08 (2008) 169–183
16. Andrés, C., Merayo, M., Núñez, M.: Formal Passive Testing of Timed Systems: Theory and Tools. Software Testing, Verification & Reliability **22**(6) (2012) 365–405
17. 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) 450–461