Метод формальной спецификации аппаратуры с конвейерной организацией и его приложение к задачам функционального тестирования

А.С. Камкин
kamkin@ispras.ru

Аннотация. В работе рассматривается метод формальной спецификации аппаратуры с конвейерной организацией, который основан на пред- и постусловиях стадий выполнения операций. Данный метод может быть использован для функционального тестирования моделей аппаратуры, поскольку на основе спецификаций предлагаемого вида можно решать основные задачи тестирования: проверку правильности поведения системы и генерацию тестовой последовательности. Метод был успешно применен для тестирования нескольких модулей промышленного микропроцессора. В результате тестирования были найдены критичные ошибки, не обнаруженные при использовании других подходов.

1. Введение

Современные электронные устройства является очень сложными системами, производство которых можно начинать только после тщательного проектирования. Проектирование электроники осуществляется с помощью специализированных *языков описания аппаратуры* (*HDLs*, *Hardware Description Languages*), например, Verilog или VHDL [1], позволяющих абстрагироваться от деталей расположения и соединения отдельных элементов электронной схемы. Проект устройства на таком языке является исполнимой программой, у которой имеются параметры, соответствующие входам устройства, и называется *моделью*[1]. После окончания проектирования

[1] Такие модели также называют *HDL-моделями* или *RTL-моделями*, поскольку проектирование ведется на *уровне регистровых передач* (*RTL*, *Register Transfer Level*).

модель преобразуется в точное описание расположения и соединения друг с другом отдельных элементов электронной схемы устройства.

Поскольку современное аппаратное обеспечение является очень сложным, при его проектировании часто делаются ошибки. Ошибка проектирования впоследствии может проявиться как некорректное, несоответствующее требованиям поведение устройства в определенной ситуации. Ошибки в аппаратуре способны приносить огромный ущерб, однако их исправление в уже изготовленном устройстве практически невозможно и требует колоссальных затрат [2]. Поэтому бо́льшую часть ошибок крайне желательно выявить и исправить еще в процессе проектирования, при разработке модели.

Для проверки корректности моделей аппаратуры используют различные методы *функциональной верификации* [3, 4]. Наиболее широко используемым на практике методом верификации является *имитационное тестирование* (*simulation-based verification*)[2]. Этот метод состоит в программной имитации работы устройства, описываемого моделью, с помощью *симулятора*, в рамках ряда специально разработанных *тестовых сценариев*, которые представляют собой основные варианты использования функций этого устройства, возможные при его эксплуатации. Главной задачей тестирования является проверка соответствия поведения системы предъявляемым к ней требованиям. Для возможности автоматизации такой проверки требования к системе должны быть представлены в машиночитаемой форме, которую называют *формальными спецификациями*.

Настоящая работа посвящена формальной спецификации *аппаратуры с конвейерной организацией*. В общих словах, конвейерная организация означает, что в один и тот же момент времени устройство может обрабатывать сразу несколько операций, однако поток операций, подаваемых на выполнение, является последовательным [5]. Конвейерная обработка является основным способом повышения производительности разного рода устройств; в частности, по такому принципу проектируются микропроцессоры, их модули и подсистемы. В статье рассматривается метод формальной спецификации конвейерной аппаратуры, основанный на пред- и постусловиях стадий выполнения операций, и описывается его приложение к задачам функционального тестирования.

Несколько слов о том, как организована статья. Во втором разделе, который начинается сразу за этим абзаца, даются общие сведения об аппаратуре с конвейерной организацией. Третий раздел описывает предлагаемый метод формальной спецификации. В четвертом разделе описывается применение метода для решения задач тестирования: проверки правильности поведения и генерации тестовой последовательности. Пятый раздел содержит сравнение

[2] В дальнейшем для краткости будем называть имитационное тестирование аппаратуры просто *тестированием*, хотя под тестированием обычно понимается проверка готовой микросхемы.

рассматриваемого метода с существующими подходами. В шестом разделе описывается практическая апробация подхода. Наконец, в последнем, седьмом разделе делается заключение и очерчиваются направления дальнейших исследований.

2. Аппаратура с конвейерной организацией

Модели аппаратуры на HDL-языках представляют собой системы из нескольких взаимодействующих *модулей*. Как и в традиционных языках программирования, модули используются для декомпозиции сложной системы на множество независимых или слабо связанных подсистем. Каждый модуль имеет интерфейс – набор входов и выходов, через которые осуществляется соединение модуля с окружением, и реализацию, определяющую способ обработки модулем входных сигналов: вычисление значений выходных сигналов и изменение внутреннего состояния. Для удобства будем считать, что спецификация и тестирование осуществляются на уровне отдельных модулей.

Как правило, модули цифровой аппаратуры работают синхронно под управлением *тактового сигнала*. Фронты тактового сигнала разбивают непрерывное время на дискретный набор интервалов, называемых *тактами*. Что делать модулю на текущем такте, определяется значениями входных сигналов и внутренним состоянием модуля. Часть входов модуля определяет операцию, которую модулю следует выполнить, – такие входы называются *управляющими*; другая часть определяет аргументы операции, – такие входы называются *информационными*. Среди реализуемых модулем операций обычно присутствует «пустая» операция *NOP* (*No Operation*), означающая бездействие, отсутствие операции. Состояние модуля также можно разбить на две составляющие: *состояние управления* и *состояние данных*. Первая из них является частью управляющей логики модуля и используется для управления процессами выполнения операций. Состояние данных, как видно из названия, представляет собой внутренние данные модуля.

Рассмотрим, как осуществляется выполнение модулем однотактной операции (см. рис. 1). До начала очередного такта окружение устанавливает код операции и аргументы на соответствующих входах модуля (на рис. 1 – *сигнал подачи операции*). Операция запускается с началом такта. За этот такт модуль производит необходимые вычисления, изменяет внутреннее состояние и устанавливает значения выходных сигналов, которые окружение может использовать, начиная со следующего такта (*результат операции*).

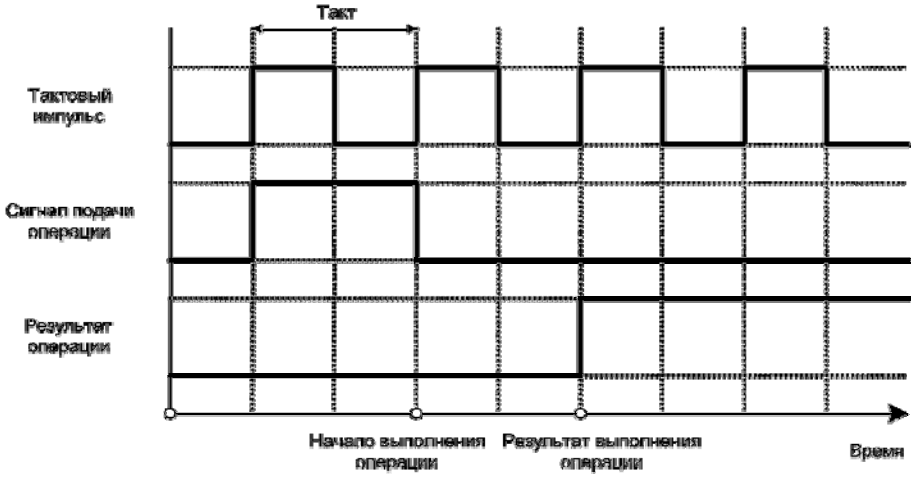

Рис. 1. Временная диаграмма сигналов для однотактной операции.

В отличие от однотактной операции, результат многотактной операции вычисляется постепенно такт за тактом. Пусть операция A выполняется модулем за L тактов. Тогда на каждом такте $i \in \{1, ..., L\}$ модуль выполняет некоторую *микрооперацию* A_i, а окружение после окончания каждого такта получает некоторый частичный результат. Представление многотактной операции A в виде последовательности микроопераций $(A_1, ..., A_L)$ называется *декомпозицией операции A на стадии*[3].

В этой статье акцент делается на конвейерные модули, то есть модули, в которых операции подаются на выполнение последовательно одна за другой, но процессы их выполнения могут пересекаться по времени (см. рис. 2). Такие модули широко распространены, особенно в практике проектирования микропроцессоров. Можно выделить два типа конвейеров: *конвейеры без блокировок* (*pipelines without interlocked stages*) и *конвейеры с блокировками* (*pipelines with interlocked stages*). В конвейерах первого типа выполнение последовательных стадий, относящихся к одной операции, осуществляется на последовательных тактах: если стадия A_i была выполнена на такте j, то стадия A_{i+1} будет выполнена на такте j+1. В конвейерах второго типа это свойство, вообще говоря, не выполнено – между тактами, на которых выполняются последовательные стадии, могут быть задержки: если стадия A_i была выполнена на такте j, то стадия A_{i+1} будет выполнена на такте $j+1+\delta$, где $\delta \geq 0$ – величина задержки между стадиями, которая определяется на основе состояния управления модуля.

[3] В данной статье термины «стадия» и «микрооперация» являются синонимами.

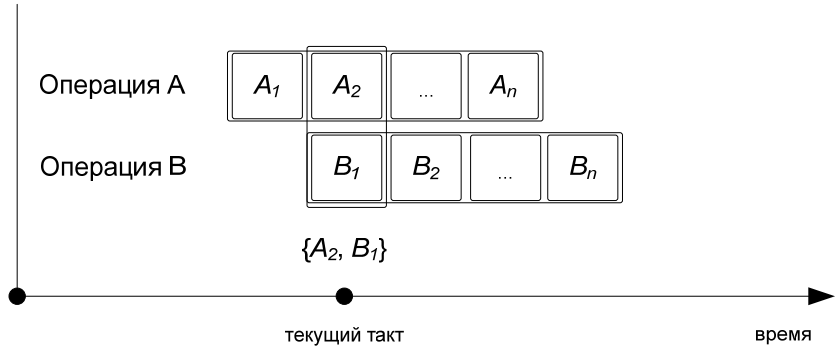

Рис. 2. Выполнение двух операций на конвейере.

Блокировки в конвейере предназначены для того, чтобы избежать конфликтов использования ресурсов между операциями и сохранить целостность потока данных. Например, в приведенной ниже ассемблерной программе инструкция `add` складывает содержимое регистра `r2` с содержимым регистра `r3` и сохраняет результат в регистре `r1`. Следом за ней идет инструкция `sub`, которая использует значение регистра `r1`. В этом случае доступ к регистру `r1` из инструкции `sub` блокируется до тех пор, пока инструкция `add` не запишет в него результат – в противном случае инструкция `sub` считает некорректное значение.

```
add r1, r2, r3 // запись в регистр r1
sub r4, r1, r5 // чтение из регистра r1
```

Конвейерная организация аппаратуры позволяет увеличить производительность, но одновременно привносит сложность в реализацию, что сказывается и на сложности тестирования. Во-первых, усложняется проверка правильности поведения системы, поскольку стадии разных операций могут перекрываться по времени и это необходимо учитывать. Во-вторых, из-за наличия нетривиальной управляющей логики увеличивается пространство состояний, что усложняет построение тестовой последовательности.

3. Формальная спецификация конвейера

В данном разделе рассматривается метод формальной спецификации аппаратуры с конвейерной организацией, основанный на пред- и постусловиях стадий выполнения операций. Формальные спецификации предлагаемого вида называются *контрактными спецификациями конвейера*.

3.1. Вспомогательные понятия

Введем основные понятия теории автоматов, на которых базируется контрактная спецификация конвейера.

Определение. *Автоматом* называется четверка $\langle S, X, Y, T\rangle$, в которой:

- S – множество состояний;
- X – множество стимулов;
- Y – множество реакций;
- $T \subseteq S \times X \times Y \times S$ – отношение переходов.

  Каждый переход $t \in$ T имеет вид (s_t, x_t, y_t, s'_t), где:
  - $s_t \in S$ – пресостояние;
  - $x_t \in X$ – стимул;
  - $y_t \in Y$ – реакция;
  - $s'_t \in S$ – постсостояние. ■

Модель автомата, расширенная контекстными переменными, входными параметрами стимулов, выходными параметрами реакций, предикатами и функциями переходов, определенными над контекстными переменными и входными параметрами, называется *расширенным автоматом*. Расширенные автоматы широко используются в информатике для моделирования программных и аппаратных систем. Обычно считают, что автомат, лежащий в основе расширенного автомата, моделирует *управляющую логику* системы, а контекстные переменные, параметры и функции, определенные над ними, – *потоки данных*.

Для удобства объединим контекстные переменные и параметры расширенного автомата общим термином *переменные*. В последующих определениях будем считать, что для каждой переменной v задано множество возможных значений Dom_v, называемое *доменом переменной* v.

Определение. Пусть V – некоторое множество переменных. Функцией означивания переменных из V называется отображение ν, которое каждой переменной $x \in V$ ставит в соответствие ее значение $\nu(x) \in Dom_x$. ■

Множество всех функций означивания переменных из V будем обозначать символом Dom_V и называть *множеством значений переменных из* V.

Определение. *Расширенным автоматом* называется шестерка $\langle S, V, I \cup O, X, Y, T\rangle$, в которой:

- S – множество состояний;
- V – множество контекстных переменных;

Функция означивания контекстных переменных называется контекстом расширенного автомата.

- $I \cup O \subseteq V$ – множество входных и выходных параметров;

  Заметим, что в данном определении множество входных и выходных параметров является подмножеством множества контекстных переменных.

- X – множество стимулов;

  С каждым стимулом $x \in X$ связано множество входных параметров $In_x \subseteq I$.
  Множество значений входных параметров стимула x будем обозначать Dom_x.

- Y – множество реакций;

- T – отношение переходов.

  С каждым переходом $t \in T$ связаны два подмножества множества контекстных переменных:

  - $Use_t \subseteq V \setminus O$ – множество используемых контекстных переменных;
  - $Def_t \subseteq V \setminus I$ – множество определяемых контекстных переменных.

  Каждый переход $t \in T$ имеет вид $(s_t, x_t, y_t, \gamma_t, \delta_t, s'_t)$, где:

  - $s_t \in S$ – пресостояние;
  - $x_t \in X$ – стимул;
  - $y_t \in Y$ – реакция;
  - γ_t: $Dom_x \times Dom_{Use} \to \{true, false\}$ – охранный предикат;
  - δ_t: $Dom_x \times Dom_{Use} \to Dom_{Def}$ – функция обновления контекста;
  - $s'_t \in S$ – постсостояние. ■

Определение. *Конфигурацией* расширенного автомата называется пара $(s, \nu) \in S \times Dom_V$, состоящая из состояния и контекста. ■

В дальнейшем будем рассматривать расширенные автоматы с выделенной начальной конфигурацией (s_0, ν_0). Такие автоматы называются *инициальными* и описываются семерками $\langle S, V, (s_0, \nu_0), I \cup O, X, Y, T\rangle$. Будем использовать следующие обозначения:

$$Tran(s) = \{t \in T \mid s_t = s\}$$
$$Init(s) = \{x_t \mid t \in Tran(s)\}$$

Определение. Предусловием стимула x в состоянии s называется предикат $pre_{s,x}$: $Dom_x \times Dom_V \to \{true, false\}$, определяемый равенством

$pre_{s,x}(p, \nu) = \vee_{\gamma \in Cond(s,x)} \gamma$, где $Cond(s, x) = \{\gamma_t \mid s_t = s, x_t = x\}$. Если $Cond(s, x) = \varnothing$, $pre_{s,x}$ полагается равным *false*. ■

Определение. Пара $x[p]$, состоящая из стимула x и набора значений его входных параметров p, называется *инициализированным стимулом*. ■

Множество всех инициализированных стимулов будем обозначать символом $Param(X)$. Аналогичное определение и связанные с ними обозначения имеют место и для переходов расширенного автомата.

Определение. Пара $t[p]$, состоящая из перехода t и набора значений входных параметров p соответствующего стимула x_t называется *инициализированным переходом*. ■

Определение. Инициализированный переход $t[p]$ называется допустимым в конфигурации (s, ν), если $s_t = s$, $x_t \in Init(s)$ и $\gamma_t(p, \nu) = true$. ■

Расширенный автомат функционирует следующим образом. Находясь в некоторой конфигурации, он получает инициализированный стимул и вычисляет множество допустимых инициализированных переходов. После этого он выполняет один из переходов, недетерминированно выбранный из вычисленного множества. В процессе выполнения расширенный автомат изменяет значения входных параметров, обновляет контекст и переходит из пресостояния в постсостояние.

3.2. Спецификация конвейера без блокировок

Для наглядности сначала определим контрактную спецификацию для случая конвейера без блокировок.

Определение. *Контрактной спецификацией конвейера без блокировок* длины L называется шестерка $\langle V, \nu_0, I \cup O, X \cup \{\tau\}, Z \cup \{\varepsilon\}, \rho\rangle$, в которой:

- V – множество контекстных переменных;
- $\nu_0 \in Dom_V$ – начальный контекст;
- $I \cup O \subseteq V$ – множество входных и выходных параметров;
- $X \cup \{\tau\}$ – множество стимулов;

  С каждым стимулом $x \in X \cup \{\tau\}$ связано множество входных параметров $In_x \subseteq I$.
  Множество значений входных параметров стимула x будем обозначать Dom_x.
  Множество $P_L = (X \cup \{\tau\})^L$ называется множеством *состояний управления*, а его элемент $\pi_0 = (\tau, ..., \tau)$ – начальным состоянием управления.
  Для каждого стимула x также заданы:

  - γ_x: $P_L \to \{true, false\}$ – охранный предикат стимула;

- pre_x: $Dom_x \times Dom_V \to \{true, false\}$ – предусловие стимула.

Множество стимулов включает выделенный стимул τ, называемый тактовым стимулом, для которого выполнены следующие свойства:

- $In_\tau = \varnothing$;
- $\gamma_\tau \equiv true$;
- $pre_\tau \equiv true$.

- $Z \cup \{\varepsilon\}$ – множество стадий;

Для каждой стадии z заданы:

- $Use_z \subseteq V \setminus O$ – множество используемых контекстных переменных;
- $Def_z \subseteq V \setminus I$ – множество определяемых контекстных переменных;
- $post_z$: $Dom_{Use} \times Dom_{Def} \to \{true, false\}$ – постусловие стадии.

Множество стадий включает выделенную стадию ε, называемую пустой стадией, для которой выполнены следующие свойства:

- $Use_\varepsilon = \varnothing$;
- $Def_\varepsilon = \varnothing$;
- $post_\varepsilon \equiv true$.

- ρ: $X \cup \{\tau\} \to (Z \cup \{\varepsilon\})^L$ – функция композиции операций.

Последовательность стадий $\rho(x)$ называется операцией стимула x. Функция ρ обладает следующим свойством: $\rho(\tau) = (\varepsilon, ..., \varepsilon)$[4]. ■

Для интерпретации контрактной спецификации конвейера будем использовать *оператор сдвига конвейера* и *оператор связывания стадий конвейера*.

Определение. Функция $\circ$: $(X \cup \{\tau\}) \times P_L \to P_L$, определяемая равенством

$$x_0 \circ (x_1, ..., x_L) = (x_0, ..., x_{L-1}),$$

называется оператором сдвига конвейера. ■

Определение. Функция θ: $P_L \to 2^Z$, определяемая равенством

$$\theta(\pi) = \{\rho_1(\pi_1), ..., \rho_L(\pi_L)\} \setminus \{\varepsilon\},$$

называется оператором связывания стадий конвейера. ■

Смысл этих операторов достаточно прост: оператор сдвига конвейера добавляет в начало конвейера новый стимул, сдвигая при этом

[4] Стимул τ и соответствующая ему последовательность пустых стадий $(\varepsilon, ..., \varepsilon)$ формализуют операцию *NOP*.

предшествующие; оператор связывания стадий конвейера возвращает множество стадий, выполняемых в текущем состоянии управления.

Определение. Множество стадий называется *конфликтным*, если для некоторых стадий $y \neq z$ из этого множества выполнено хотя бы одно из следующих условий:

- $Use_y \cap Def_z \neq \varnothing$ – конфликт типа «чтение-запись»;
- $Def_y \cap Def_z \neq \varnothing$ – конфликт типа «запись-запись».

Если ни для каких стадий $y \neq z$ ни одно из перечисленных условий не выполнено, множество стадий называется *согласованным*. ■

Определение. Состояние управления π называется *согласованным*, если для всех $1 \le i \ne j \le L$, таких что $\rho_i(\pi_i) \ne \varepsilon$ и $\rho_j(\pi_j) \ne \varepsilon$, выполняется $\rho_i(\pi_i) \ne \rho_j(\pi_j)$ и множество стадий $\theta(\pi)$ является согласованным. ■

Определение. Контрактная спецификация конвейера называется *согласованной*, если для всех стимулов x и состояний управления π из того, что π согласованно и $\gamma_x(\pi) = true$, вытекает, что $x \circ \pi$ согласованно. ■

В дальнейшем будем рассматривать только согласованные контрактные спецификации конвейера. Пусть $\Sigma = \langle V, \nu_0, I \cup O, X \cup \{\tau\}, Z \cup \{\varepsilon\}, \rho \rangle$ – контрактная спецификация конвейера без блокировок длины L. Ее можно интерпретировать как инициальный расширенный автомат $\Delta = \langle S, V, (\pi_0, \nu_0), I \cup O, X \cup \{\tau\}, Y \cup \{\xi\}, T \rangle$ с выделенными тактовым стимулом τ и пустой реакцией ξ, устроенный следующим образом:

- $S = P_L$ – состояниями расширенного автомата Δ являются состояния управления контрактной спецификации Σ;
- $\pi_0 = (\tau, ..., \tau)$ – начальным состоянием является начальное состояние управления;
- $Y = 2^Z$ – реакциями расширенного автомата Δ являются множества стадий контрактной спецификации Σ;
- $\xi = \varnothing$ – пустой реакцией является пустое множество стадий;
- отношение переходов T расширенного автомата Δ для всех $\pi \in S$ и $x \in X \cup \{\tau\}$ таких, что $\gamma_x(\pi) = true$, содержит все возможные переходы $t = (\pi, x, y, \gamma, \delta, \pi')$, в которых:
  - $Use_t = \cup_{z \in \theta(\pi')} Use_z$

множество используемых контекстных переменных получается объединением соответствующих множеств для стадий из $\theta(\pi')$;

- $Def_t = \cup_{z \in \theta(\pi')} Def_z$

  множество определяемых контекстных переменных получается объединением соответствующих множеств для стадий из $\theta(\pi')$;

- $\pi' = x \circ \pi$

  постсостояние получается сдвигом конвейера;

- $y = \theta(\pi')$

  реакция определяется как результат связывания стадий, выполненного в постсостоянии перехода;

- $\gamma(p, \nu) = pre_x(p, \nu)$

  охранный предикат определяется как предусловие соответствующего стимула;

- результирующий контекст $\nu' = \delta(p, \nu)$ для всех $\nu \in Dom_V$, таких что $pre_x(p, \nu) = true$, удовлетворяют предикату:

$$\Phi_{\pi, x}(p, \nu, \nu') = \vee_{z \in \theta(\pi')} post_z(\nu|_{Use}, \nu'|_{Def}). \blacksquare$$

Определение. Предикат $\Phi_{\pi, x}(p, \nu, \nu')$ называется *тестовым оракулом* стимула x в состоянии управления π. ■

Тестовому оракулу отводится основная роль в проверке правильности поведения тестируемого устройства (см. раздел «*Проверка правильности поведения*»).

Определение. Автомат, лежащий в основе расширенного автомата $\Delta(\Sigma)$, интерпретирующего контрактную спецификацию Σ, называется *управляющим автоматом* спецификации Σ и обозначается $\Omega(\Sigma)$. ■

Управляющий автомат является формальной моделью управляющей логики устройства. На основе обхода графа состояний управляющего автомата осуществляется генерация тестовой последовательности (см. подраздел 4.2).

3.3. Спецификация конвейера с блокировками

Обобщим введенное ранее понятие контрактной спецификации на случай конвейера, в котором возможны блокировки. Дадим несколько вспомогательных определений.

Определение. *Состоянием обработки стимула* или *процессом* называется пара (x, l), в которой:

- $x \in X \cup \{\tau\}$ – обрабатываемый стимул;
- $l \in \{1, ..., L\}$ – номер стадии обработки стимула. ■

Определение. *Состоянием управления* называется множество состояний обработки стимулов. Пустое множество состояний обработки стимулов называется начальным состоянием управления. ■

Множество всевозможных состояний управления для конвейера длины L будем обозначать символом P_L.

Определение. *Контрактной спецификацией конвейера с блокировками* длины L называется шестерка $\langle V, \nu_0, I \cup O, X \cup \{\tau\}, Z \cup \{\varepsilon\}, \rho\rangle$, в которой:

- V – множество контекстных переменных;
- $\nu_0 \in Dom_V$ – начальный контекст;
- $I \cup O \subseteq V$ – множество входных и выходных параметров;
- $X \cup \{\tau\}$ – множество стимулов;

  Для каждого стимула $x \in X \cup \{\tau\}$ заданы:

  - $In_x \subseteq I$ – множество входных параметров;
  - γ_x: $P_L \to \{true, false\}$ – охранный предикат стимула;
  - pre_x: $Dom_x \times Dom_V \to \{true, false\}$ – предусловие стимула.

- $Z \cup \{\varepsilon\}$ – множество стадий;

  Для каждой стадии $z \in Z \cup \{\varepsilon\}$ заданы:

  - $Use_z \subseteq V \setminus O$ – множество используемых контекстных переменных;
  - $Def_z \subseteq V \setminus I$ – множество определяемых контекстных переменных;
  - γ_z: $P_L \to \{true, false\}$ – охранный предикат стадии;
  - $post_z$: $Dom_V \times Dom_V \to \{true, false\}$ – постусловие стадии.

- ρ: $X \cup \{\tau\} \to (Z \cup \{\varepsilon\})^L$ – функция композиции операций. ■

Заметим, что по сравнению с определением контрактной спецификации конвейера без блокировок в данном определении появились охранные предикаты стадий – именно они используются для описания блокировок конвейера.

Определение. Процесс (x, l) называется *активным* в состоянии управления π, если $\gamma_z(\pi) = true$, где $z = \rho_l(x)$. В противном случае процесс называется *заблокированным*. ■

Определение. Множество $Enabled(\pi) = \{(x, l) \in \pi \mid \gamma_z(\pi) = true$, где $z = \rho_l(x)\}$ называется *множеством активных процессов* в состоянии управления π. ■

Определение. Множество $Locked(\pi) = \pi \setminus Enabled(\pi)$, являющееся дополнением множества $Enabled(\pi)$, называется *множеством заблокированных процессов* в состоянии управления π. ■

Отличие в интерпретации контрактной спецификации конвейера с блокировками по сравнению со случаем без блокировок заключается в измененной семантике операторов сдвига конвейера и связывания стадий.

Определение. Функция $\circ: (X \cup \{\tau\}) \times P_L \to P_L$, которая для всех пар (x, π) принимает значение $x \circ \pi$, равное объединению следующих множеств:

- $Locked(\pi)$;
- $\{(x, l+1) \mid (x, l) \in Enabled(\pi) \wedge l < L\}$;
- $\{(x, 1)\}$, если $x \neq \tau$; $\varnothing$, если $x = \tau$.

называется оператором сдвига конвейера. ■

Определение. Функция $\theta: P_L \to 2^Z$, определяемая равенством

$$\theta(\pi) = \{\rho_l(x) \mid (x, l) \in Enabled(\pi)\} \setminus \{\varepsilon\},$$

называется оператором связывания стадий конвейера. ■

4. Приложение к задачам тестирования

Рассмотрим применение контрактных спецификаций конвейера для решения основных задач тестирования: проверки правильности поведения и генерации тестовой последовательности.

4.1. Проверка правильности поведения

Определим отношение соответствия между контрактными спецификациями конвейера и инициальными расширенными автоматами[5]. Рассмотрим контрактную спецификацию $\Sigma = \langle V^S, \nu_0{}^S, I^S \cup O^S, X^S \cup \{\tau^S\}, Z^S \cup \{\varepsilon^S\}, \rho^S\rangle$ конвейера без блокировок длины L и инициальный расширенный автомат $\Delta = \langle S^I, V^I, (s_0{}^I, \nu_0{}^I), I^I \cup O^I, X^I \cup \{\tau^I\}, Y^I \cup \{\xi^I\}, T^I\rangle$ с выделенными тактовым стимулом τ^I и пустой реакцией ξ^I. Введем несколько вспомогательных понятий.

Определение. Сюръективное отображение $\varphi_S{}^{\rightarrow}: P_L{}^S \to S^I$, обладающее свойством $\varphi_S{}^{\rightarrow}(\varnothing) = s_0{}^I$, называется *функцией соответствия состояний*. ■

[5] Для возможности формального определения отношения соответствия предполагается, что реализация описывается расширенным автоматом.

Определение. Отображение $\varphi_X{}^{\rightarrow}: Param(X^S \cup \{\tau^S\}) \to Param(X^I \cup \{\tau^I\})$, обладающее тем свойством, что $\varphi_X{}^{\rightarrow}(x^S) = \tau^I$ тогда и только тогда, когда $x^S = \tau^S$ называется *функцией соответствия стимулов*. ■

Определение. Биективное отображение $\varphi_Y{}^{\leftarrow}: Y^I \cup \{\xi^I\} \to 2^Z$, где $Z = Z^S$, обладающее тем свойством, что $\varphi_Y{}^{\leftarrow}(y^I) = \varnothing$ тогда и только тогда, когда $y^I = \xi^I$ называется *функцией соответствия реакций*. ■

Определение. Отображение $\varphi_V{}^{\leftarrow}: Dom_{V(I)} \to Dom_{V(S)}$, где $V(I) = V^I$ и $V(S) = V^S$, обладающее свойством $\varphi_V{}^{\leftarrow}(\nu_0{}^I) = \nu_0{}^S$, называется *функцией соответствия контекстов*. ■

Определение. Будем говорить, что инициальный расширенный автомат соответствует контрактной спецификации для заданных функций соответствия $\langle\varphi_S{}^{\rightarrow}, \varphi_X{}^{\rightarrow}, \varphi_Y{}^{\leftarrow}, \varphi_V{}^{\leftarrow}\rangle$, если для всех $\pi^S \in P_L{}^S$, $x^S[p^S] \in Param(X^S \cup \{\tau^S\})$ и $\nu^I \in Dom_{V(I)}$ из того, что $\gamma_x(\pi^S) = true$, вытекает, что $x^I \in Init(s^I)$, причем если $pre_x(p^S, \nu^S) = true$, то $pre_{s,x}(p^I, \nu^I) = true$, где $s^I = \varphi_S{}^{\rightarrow}(\pi^S)$, $x^I[p^I] = \varphi_X{}^{\rightarrow}(x^S[p^S])$ и $\nu^S = \varphi_V{}^{\leftarrow}(\nu^I)$, при этом для каждого инициализированного перехода $t^I[p^I] \in Enabled(s^I, x^I, \nu^I)$ выполнено $s'_t = \varphi_S{}^{\rightarrow}(\pi'^S)$, $\varphi_Y{}^{\leftarrow}(y_t) = \theta(\pi'^S)$ и $\Phi_{\pi,x}(p^S, \nu^S, \nu'^S) = true$, где $\pi'^S = x^S \circ \pi^S$ и $\nu'^S = \varphi_V{}^{\leftarrow}(\nu'_t)$. ■

Заметим, что реализация может содержать неспецифицированные переходы, то есть переходы, в которых для определенного набора значений входных параметров и контекста выполнен охранный предикат, но для всех соответствующих стимулов спецификации нарушается либо охранный предикат, либо предусловие. Таким образом, контрактная спецификация Σ для каждого инициального расширенного автомата Δ и заданного набора функций соответствия φ определяет подавтомат, состоящий только из специфицированных переходов, который будем называть *спецификационным подавтоматом*.

Проверка правильности поведения осуществляется для заданных функций соответствия, которые выполняют функции преобразования данных из спецификационного представления в реализационное ($\varphi_S{}^{\rightarrow}$ и $\varphi_X{}^{\rightarrow}$) и наоборот из реализационного представления в спецификационное ($\varphi_Y{}^{\leftarrow}$ и $\varphi_V{}^{\leftarrow}$). Компоненты тестовой системы, которые реализуют такие функции, называются *адаптерами*. Схема проверки правильности поведения, приведенная ниже, основана на определении отношения соответствия:

```
(π^S, ν^S) ← (∅, ν_0^S);
if(s^I ≠ φ_S^→(π^S) ∨ ν^S ≠ φ_V^←(ν^I))
   { error(«Ошибка инициализации»); }
// пока тест не завершен
while(¬isTestComplete()) {
   // получить очередной стимул от тестовой системы
   x^S[p^S] ← getNextStimulus();
```

```
// проверить выполнимость охранного предиката и предусловия
 стимула
if((γx(π^S) ∧ pre_x(p^S, ν^S)) {
   // преобразовать стимул в реализационное представление
   x^I[p^I] ← φ_X→(x^S[p^S]);
   // подать стимул на реализацию
   applyStimulus(x^I[p^I]);
   // подождать один такт
   waitOneCycle();
   // получить состояния, реакцию и контекст реализации
   s^I ← getState();
   y^I ← getReaction();
   ν^I ← getContext();
   // преобразовать реакцию/контекст в реализационное
   представление
   y^S ← φ_Y←(y^I);
   ν'^S ← φ_V←(ν^I);
   // вычислить новое состояние управления
   π^S ← x^S ∘ π^S;
   if(s^I ≠ φ_S→(π^S))
      { error(«Ошибка несоответствия состояний»); }
   if(y^S ≠ θ(π^S))
      { error(«Ошибка несоответствия реакций»); }
   if(¯Φ_π,x(p^S, ν^S, ν'^S))
      { error(«Ошибка несоответствия контекстов»); }
 }
}
```

В представленной выше схеме проверки используются следующие обобщенные функции тестовой системы:

- *isTestComplete* – проверяет завершена ли генерации тестовой последовательности;
- *getNextStimulus* – получает очередной инициализированный стимул от генератора тестовой последовательности;
- *applyStimulus* – подает инициализированный стимул на реализацию;
- *waitOneCycle* – ожидает один такт;
- *getState* – получает состояние управления реализации;
- *getReaction* – получает реакцию реализации;
- *getContext* – получает контекст реализации.

Часто при тестировании состояние управления реализации является скрытым (либо к нему отсутствует доступ, либо доступ есть, но от деталей реализации управляющей логики целесообразно абстрагироваться), поэтому функция

получения состояния управления *getState* и функция соответствия состояний $\varphi_S^{\rightarrow}$ неопределенны. В этом случае проверки вида $s^I \neq \varphi_S^{\rightarrow}(\pi^S)$ опускаются. Сравнение $y^S \neq \theta(\pi^S)$ также не осуществляется, поскольку реакции устройства (внутренние действия по обработке стимулов), как правило, не наблюдаемы. Проверки, которые выполняются на практике, базируются на предикате $\Phi_{\pi,x}(p^S, \nu^S, \nu'^S)$, то есть на наблюдении за изменениями контекста (состояния данных и выходов модуля).

В схеме проверки правильности поведения заложена следующая классификация ошибок в модулях микропроцессоров: *ошибки несоответствия состояний*, *ошибки несоответствия реакций* и *ошибки несоответствия контекстов*. Ошибки несоответствия состояний являются формализацией ошибок в управляющей логике. Именно на обнаружения ошибок этого типа в первую очередь нацелен предлагаемый метод автоматизации тестирования. Поскольку проверки правильности поведения основаны на предикате $\Phi_{\pi,x}$, предполагается, что ошибки несоответствия состояний проявляются в некорректном изменении контекста, в частности, выходов модуля.

4.2. Генерация тестовой последовательности

Основная особенность конвейерных модулей заключается в сложной организации управляющей логики. В данном разделе рассматривается метод генерации тестовой последовательности, нацеленный на создание различных ситуаций именно в управляющем компоненте модуля. Метод основан на контрактных спецификациях конвейера. Пусть задана спецификация $\Sigma = \langle V, \nu_0, I \cup O, X \cup \{\tau\}, Z \cup \{\varepsilon\}, \rho \rangle$.

Определение. *Тестовой последовательностью* для контрактной спецификации Σ называется произвольная последовательность переходов расширенного автомата $\Delta(\Sigma)$, интерпретирующего спецификацию Σ, допустимая в начальной конфигурации $(\varnothing, \nu_0)$. ■

Формализацией управляющей логики модуля, описываемого спецификацией Σ, является управляющий автомат $\Omega(\Sigma)$. С точки зрения спецификации цель тестирования задается как покрытие всех достижимых состояний управляющего автомата. Предлагаемый метод генерации тестовой последовательности основан на обходе графа состояний управляющего автомата.

Для обхода мы используем *неизбыточные алгоритмы*, то есть алгоритмы, для работы которых достаточно информации о пройденном подграфе. Входной информацией неизбыточных алгоритмов обхода графов является неизбыточное описание графа [6]

Определение. *Неизбыточным описанием* графа состояний называется тройка $\langle V, X, enabled \rangle$, где:

- V – множество вершин;

- X – множество стимулов;
- *enabled*: $V \to 2^X$ – функция, которая для каждой вершины графа возвращает множество допустимых в ней стимулов. ■

В работах [6-9] исследованы неизбыточные алгоритмы обхода разных классов графов. В частности, в статье [8] рассмотрен неизбыточный алгоритм обхода α_{dfsm}[6] на классе детерминированных сильно-связных конечных графов, а в [6] – алгоритм α_{ndfsm} на классе графов, имеющих детерминированный сильно-связный полный остовный подграф. Следует отметить, что алгоритм α_{ndfsm} применяется и в тех случаях, когда граф заведомо детерминирован. Дело в том, что этот алгоритм относится к «жадным» алгоритмам и в ряде случаев позволяет значительно сократить длину тестовой последовательности.

Заметим, что управляющий автомат является конечным (если множество стимулов конечно) и детерминированным. Несложно показать, что граф состояний управляющего автомата является сильно-связным, если в нем нет *тупиковых состояний*, то есть состояний $\pi \neq \varnothing$, таких, что $Enabled(\pi) = \varnothing$ (на практике это условие тоже выполнено – наличие тупикового состояния сигнализирует об ошибке в спецификации). Таким образом, условия применения неизбыточных алгоритмов выполнены.

Неизбыточное описание графа состояний управляющего автомата имеет вид $\langle P_L, X \cup \{\tau\}, enabled\rangle$, где множество допустимых стимулов для каждого состояния π определяется следующим образом:

$$enabled(\pi) = \{x \in X \cup \{\tau\} \mid \gamma_x(\pi) = true\}.$$

При тестировании управляющей логики конкретные значения параметров стимулов не важны – можно использовать любые значения, удовлетворяющие соответствующему предусловию. В предложенном методе предполагается, что для любого стимула x и контекста ν существует набор параметров $p \in Dom_x$, который удовлетворяет условию $pre_x(p, \nu) = true$. На практике это не всегда так, то есть возможны ситуации, когда для некоторого стимула x существует контекст ν, для которого не существует значений параметров, удовлетворяющих предусловию. Получается, что в одном и том же состоянии управляющего автомата в некоторых ситуациях предусловие стимула выполнено, а в некоторых нет. Эта проблема решается добавлением в состояние обходимого графа конвейера дополнительной информации, на основе которой определяется выполнимость предусловий.

5. Сравнение с существующими подходами

В данном разделе приводится сравнение предлагаемого метода спецификации и тестирования аппаратуры с конвейерной организацией с существующими

[6] В указанной работе этот алгоритм обозначается символом A_2.

подходами. Сравнение разбито на две части: в первой из них сравниваются методы спецификации, во второй – методы генерации тестовой последовательности.

5.1. Методы спецификации аппаратуры

В настоящее время для спецификации аппаратного обеспечения широко используются так называемые *языки верификации аппаратуры* (*HVL*, *Hardware Verification Languages*) [1]. К таким языкам относятся PSL, OpenVera, SystemVerilog и другие. HVL-языки включают в себя конструкции языков программирования, языков описания аппаратуры, а также специальные средства, ориентированные на верификацию. Средства спецификации, используемые в современных языках верификации аппаратуры, базируются на *темпоральной логике линейного времени* (*LTL*, *Linear Temporal Logic*) и *темпоральной логике деревьев вычислений* (*CTL*, *Computation Tree Logic*) [1].

Логика CTL используется преимущественно для формальной верификации систем. Для целей симуляции и тестирования больший интерес представляет логика линейного времени. Все языки, использующие LTL, имеют схожие средства спецификации. Они оперируют с ограниченными по времени последовательностями событий, для которых можно обращаться как к прошлому, так и к будущему. Из простых последовательностей можно строить более сложные с помощью логических связок, таких как *И* и *ИЛИ*, или используя регулярные выражения. На последовательностях событий можно с помощью темпоральных операторов задавать утверждения (assertions).

В подходах на основе темпоральных логик, упор делается на *временную декомпозицию операций*. Для каждой операции сначала выделяется ее временн*ая* структура – допустимые последовательности событий и задержки между ними; затем определяются предикаты, описывающие отдельные события; после этого предикаты, относящиеся к одному моменту времени, могут быть некоторым образом сгруппированы. В подходе, предлагаемом нами, основной акцент ставится на *функциональную декомпозицию операций*. Первым делом выделяется функциональная структура операции в виде последовательности стадий; каждая стадия специфицируется; временн*ая* привязка спецификаций осуществляется в процессе тестирования с помощью вычисления на каждом такте условий блокировки стадий.

Мы полагаем, что функциональная структура операции более устойчива по сравнению с временн*ой*. Тем самым, подходы, основанные на функциональной декомпозиции операций, позволяют разрабатывать спецификации более устойчивые к изменениям реализации по сравнению с подходами на основе темпоральных логик. К достоинствам предлагаемого метода можно отнести его наглядность. Контрактная спецификация конвейера базируется на известных разработчиками аппаратуры понятиях таких, как стадия операции, блокировка стадии и других.

5.2. Методы генерации тестовой последовательности

Методы генерации тестов для аппаратуры с конвейерной организацией в основном исследуются в контексте тестирования микропроцессоров и их модулей. Многие исследователи сходятся во мнении, что удобным средством генерации тестовых последовательностей являются автоматные модели.

Для построения тестов, покрывающих отдельные ситуации, часто используют методы проверки моделей [10-18]. Генерация тестов, полно покрывающих управляющую логику устройства, как правило, основана на обходе графа состояний автоматной модели [19, 20]. Автоматная модель либо извлекается автоматически на основе статического анализа кода реализации [19, 21], либо строится вручную [14, 20]. Автоматическое извлечение модели является сложной задачей, требующей либо наличия в коде аннотаций разработчиков [19], либо привлечения эвристик [10, 11]. Для сложного аппаратного обеспечения автоматическое извлечение автоматной модели управляющей логики практически неосуществимо. При построении модели вручную возникает другая проблема – построенную модель сложно отлаживать [20]. Поскольку на основе такой модели предполагается генерация тестов, важно, чтобы модель точно описывала тестируемый компонент, так как в противном случае цели тестирования не будут достигнуты.

В предлагаемом методе автоматная модель управляющей логики извлекается из формальных спецификаций аппаратного обеспечения. Во-первых, это делает метод масштабируемым на достаточно сложные устройства (см. раздел 6). Во-вторых, это решает задачу отладки моделей, поскольку формальные спецификации, из которых извлекается автоматная модель, также используются для проверки правильности поведения.

6. Опыт практического применения метода

Описанный в статье метод спецификации и тестирования аппаратуры с конвейерной организацией был применен для тестирования модулей промышленного микропроцессора с MIPS64-совместимой архитектурой [22]: *буфера трансляции адресов* (*TLB*, *Translation Lookaside Buffer*) и *модуля кэш-памяти второго уровня* (*L2 cache*). Для разработки тестов использовался инструмент CTESK из набора инструментов UniTESK [23], расширенный библиотекой PIPE, реализующей основные концепции предлагаемого подхода.

6.1. Тестирование буфера трансляции адресов

Память тестируемого буфера трансляции адресов состоит из 64 ячеек, которые составляют *объединенный TLB* (*JTLB*, *Joint TLB*). Кроме того, для повышения производительности модуль содержит два дополнительных буфера из 4 ячеек каждый: *TLB данных* (*DTLB*, *Data TLB*) и *TLB инструкций* (*ITLB*, *Instructions TLB*). DTLB используется при трансляции адресов данных, ITLB – при трансляции адресов инструкций. Наличие двух дополнительных буферов

позволяет производить две операции трансляции адреса одновременно: одну для выборки инструкции (через ITLB), вторую для загрузки или сохранения данных (через DTLB).

Модуль TLB реализует операции чтения, записи, проверки наличия ячейки в буфере, а также операции трансляции адресов данных и инструкций. Интерфейс модуля состоит из приблизительно 30 входов и стольких же выходов. Модуль реализован на языке Verilog, его описание (не считая библиотек) составляет около 3500 строк кода. Формальные спецификации и тесты были разработаны одним человеком приблизительно за 2.5 месяца, а их объем составил около 3500 строк кода. В результате тестирования было найдено 10 ошибок в реализации модуля, включая критичные. Найденные ошибки не были обнаружены при тестировании микропроцессора с помощью других методов[7].

6.2. Тестирование модуля кэш-памяти второго уровня

Тестируемый модуль кэш-памяти второго уровня имеет объем 256 Кбайт и состоит из 8192 строк. Каждая строка содержит данные (4 двойных слова по 64 бит), тэг (18 старших бит физического адреса) и биты служебной информации. Память модуля является смешанной – в ней могут храниться как данные, так и инструкции. Кэш-память адресуется физическим адресом путем прямого отображения.

Модуль кэш-памяти реализует операции загрузки и сохранения данных (в разных режимах), выборки инструкций, а также управляющую операцию для изменения данных и управляющих битов. Интерфейс модуля содержит около 70 входов и 30 выходов. Реализация модуля на языке Verilog составляет около 3000 строк кода. Формальные спецификации и тесты были разработаны одним человеком приблизительно за 4 месяца, а их объем составил около 4000 строк кода. В результате тестирования были найдены 9 ошибок в реализации модуля.

7. Заключение

В статье рассмотрен метод формальной спецификации аппаратуры с конвейерной организацией, основанный на пред- и постусловиях стадий выполнения операций. Метод позволяет в наглядной, интуитивно понятной форме описывать функциональность конвейерных устройств и может быть использован для функционального тестирования. Метод был успешно применен для проверки нескольких модулей промышленного микропроцессора. В результате тестирования были найдены серьезные ошибки, не обнаруженные при использовании других подходов.

[7] Микропроцессор тестировался с помощью тестовых программ на языке ассемблера, разработанных вручную и полученных с помощью случайной генерации.

К настоящему моменту нами получен большой опыт использования технологии UniTESK и инструмента CTESK для спецификации и тестирования моделей аппаратуры [24]. Опыт показывает, что некоторые шаги разработки тестов могут быть полностью или частично автоматизированы [25]. Детальное исследование этого вопроса и разработка инструментальной поддержки для дальнейшей автоматизации разработки тестов является одним из приоритетных направлений дальнейшей работы.

Еще одним направлением исследований является обобщение метода на более сложные типы конвейерных устройств. В работе был рассмотрен лишь простейший случай, когда последовательность стадий каждой операции фиксирована. На практике встречаются конвейеры с ветвлениями, в которых цепочки стадий вычисляются динамически на основе некоторых промежуточных условий, а также параллельные конвейеры, позволяющие запускать несколько операций параллельно.

Литература

[1] S.A. Edwards. *Design and Verification Languages*. Technical Report, Columbia University, New York, USA, November 2004.

[2] B. Beizer. *The Pentium Bug – An Industry Watershed. Testing Techniques Newsletter*, TTN Online Edition, September 1995.

[3] J. Bergeron. *Writing Testbenches: Functional Verification of HDL Models*. Kluwer Academic Publishers, 2000.

[4] W. Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall, 2005.

[5] D. Patterson and J. Henessy. *Computer Organization and Design*. 3rd Edition, Morgan Kaufmann, 2005.

[6] А.В. Хорошилов. *Спецификация и тестирование систем с асинхронным интерфейсом*. Институт системного программирования РАН, Препринт 12, 2006.

[7] И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. *Использование конечных автоматов для тестирования программ*. Программирование, 2000, №2.

[8] И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. *Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай*. Программирование, 2003, №5.

[9] И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. *Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай*. Программирование, 2004, №1.

[10] D. Moundanos, J. Abraham, and Y. Hoskote. *A Unified Framework for Design Validation and Manufacturing Test*. ITC'1996: International Test Conference, 1996.

[11] D. Moundanos, J. Abraham, and Y. Hoskote. *Abstraction Techniques for Validation Coverage Analysis and Test Generation*. IEEE Transactions on Computers, Volume 47, 1998.

[12] D. Geist, M. Farkas, A. Landver, Y. Lichtenstein, S. Ur, and Y. Wolfsthal. *Coverage Directed Test Generation Using Symbolic Techniques*. FMCAD'1996: Formal Methods in Computer Aided Design, 1996.

[13] P. Mishra and N. Dutt. *Automatic Functional Test Program Generation for Pipelined Processors using Model Checking*. HLDVT'2002: High-Level Design Validation and Test Workshop, 2002.

[14] P. Mishra and N. Dutt. *Architecture Description Language Driven Functional Test Program Generation for Microprocessors using SMV*. CECS Technical Report 02-26, 2002.

[15] P. Mishra and N. Dutt. *Graph-Based Functional Test Program Generation for Pipelined Processors*. DATE'2004: Design, Automation and Test in Europe Conference and Exhibition, 2004.

[16] P. Mishra and N. Dutt. *Functional Coverage Driven Test Generation for Validation of Pipelined Processors*. DATE'2005: Design, Automation and Test in Europe Conference and Exhibition, 2005.

[17] H.M. Koo and P. Mishra. *Test Generation using SAT-based Bounded Model Checking for Validation of Pipelined Processors*. ACM Great Lakes Symposium on VLSI, 2006.

[18] H.M. Koo and P. Mishra. *Functional Test Generation using Property Decomposition for Validation of Pipelined Processors*. DATE'2006: Design, Automation and Test in Europe Conference and Exhibition, March 2006.

[19] R. Ho, C. Yang, M. Horowitz, and D. Dill. *Architecture Validation for Processors*. ISCA'1995: International Symposium on Computer Architecture, 1995.

[20] S. Ur and Y. Yadin. *Micro Architecture Coverage Directed Generation of Test Programs*. DAC'1999: Design and Automation Conference, 1999.

[21] Y. Hoskote, D. Moundanos, and J. Abraham. *Automatic Extraction of the Control Flow Machine and Application to Evaluating Coverage of Verification Vectors*. ICCD'1995: International Conference of Computer Design, 1995.

[22] *MIPS64 Architecture For Programmers*. Revision 2.0. MIPS Technologies Inc., 2003.

[23] http://www.unitesk.com.

[24] M. Chupilko, A. Kamkin, and D. Vorobyev. *Methodology and Experience of Simulation-Based Verification of Microprocessor Units Based on Cycle-Accurate Contract Specifications*. SYRCoSE'2008: The 2nd Spring Young Researchers Colloquium on Software Engineering, 2008.

[25] В.П. Иванников, А.С. Камкин, В.В. Кулямин, А.К. Петренко. *Применение технологии UniTESK для функционального тестирования моделей аппаратного обеспечения*. Препринт 8. Институт системного программирования РАН, Москва, 2005.