УДК 681.396.6

SYSTEMVERILOG УТВЕРЖДЕНИЯ В SYSTEMVERILOG-2009

***Галина Александровна Яицкова**, канд. техн. наук., с.н.с.*
Тел.: 495 129 87 36, e-mail: tvv@niisi.ras.ru
Научно-исследовательский институт системных исследований РАН
(НИИСИ РАН)
www.niisi.ru

Заключительной, третьей частью работ по обзору конструкций языка SystemVerilog-2009, предназначенных для систематической верификации проектов, является описание набора конструкций и способов их применения, основанных на утверждениях (assertions). Информационная безопасность проекта обеспечивается расширением возможностей основных конструкций языка проектирования и конструкций, связанных с верификацией, поскольку имеется в виду проектирование очень сложных объектов, включающих IP-блоки.

Ключевые слова: утверждение, последовательность, свойство, синхронизирующая последовательность

Введение

В работе [1] было представлено краткое описание отличительных особенностей версии языка IEEE Std.1800 SystemVerylog-2009, который явился слиянием множества разработок в этой области [1] и является на сегодняшний день самым совершенным языком проектирования аппаратуры предельно возможных размеров и сложности. В частности, в стандарте языка значительно расширен аппарат верификации в области использования утверждении, а именно, SystemVerilog Assertion (SVA). В рамках настоящей работы будут подробно рассмотрены конструкции именно этой части языка [2], демонстрирующие возможности проведения верификации, что само по себе обеспечивает безопасность информации, получаемой от приборов, верификация которых осуществлялась с использованием SVA.


Г.А. Яицкова

Заметим, что конструкции языка для двух других составляющих организации верификации были рассмотрены, в частности, на предыдущих сессиях IT+S&E [3; 4]. Собственно, основные понятия языка верификации с использованием утверждений рассматривались раньше, в работах по PSL [5; 6]. Однако совершенное воплощение верификация на базе утверждений приобрела в SVA SystemVerilog-2009. Поэтому здесь мы не будем останавливаться на описании основных понятий, а только перечислим их в соответствии со стандартом. И особо обратим внимание на конструкции, позволяющие отображать темпоральные зависимости поведения проекта и управлять ими в рамках верификации.

1. Основной аппарат SVA

1.1 Утверждения. Понятия и определения

Утверждение имеет вид assertion предложения, в котором выражено верификационное предписание: -**assert -** определить выполнение данного свойства; -**assume -** принять предположение о выполнении данного свойства, при этом средства моделирования проверяют выполняемость свойства, а средства формального анализа считают высказанное предположение истинным; -**cover -** следить за покрытием свойства.

Существует два вида утверждений: немедленные (immediate assertion) и параллельные (concurrent assertion).

1.1.1 Немедленные утверждения

Немедленные утверждения подчиняются семантике событийного моделирования,

выполняются как предложения в процедурном блоке и используются, прежде всего, при моделировании, например:

always @(**posedge** clk) **if** (s3== 3) **assert** (s1 || s2);

Здесь всегда при положительном фронте clk и при равенстве s3 трем, хотя бы один из сигналов s1 или s2 истинен (установлен).

Существует два типа немедленных утверждений - простые (simple) и отсроченные (deferred). В простых утверждениях реакция pass или fail вырабатывается незамедлительно. В отсроченных утверждениях вычисление значения свойства откладывается на конец временного слота, обеспечивая некоторый уровень защиты от непреднамеренных многократных вычислений в переходном процессе. Примеры синтаксиса: simple_immediate_assert_statement ::= **assert** **(** expression **)** action_block; deferred_immediate_cover_statement ::= **cover #0** **(** expression **)** statement_or_null; action_block ::= statement _or_null | [statement]

Явная команда задержки в deferred_immediate_assertion_ statement, равной #0, требует приостановки процесса и отнесения события к области Inactive текущего временного слота, так что процесс вычисления свойства возобновляется на следующей итерации Inactive - Active. Отсроченное утверждение можно использовать вне процедурного кода в конструкции module_common_item.

Вызовы подпрограмм предложений pass и fail отсроченных утверждений помещаются в очередь сообщений отсроченных утверждений. Но, если достигается точка выключения отсроченного утверждения в результате выполнения некоторых событий, то эта очередь очищается, чтобы избежать нежелательных и, возможно, ложных сообщений об отказах (механизм возникновения можно найти в [2; 7]).

1.1.2 Параллельные утверждения

Параллельные утверждения основаны на временной семантике и используют при вычислении свойств значения переменных, выбранных в определенные моменты времени согласно их часам. Часы свойства – это некоторые периодические события, в момент наступления которых происходит (очередное) вычисление свойства. Например, этими событиями могут служить прямые или обратные фронты периодического сигнала. Периоды срабатывания этих часов называются циклами, а вычисление свойства происходит в моменты срабатывания часов.

Если переменная, используемая в конструкции assertion – входная переменная clocking блока, то переменная должна выбираться clocking блоком с задержкой выборки #1step. Пример рис.1 демонстрирует логику темпорального взаимоотношения шагов моделирования, циклов часов (clocking) и выбираемых значений переменных.

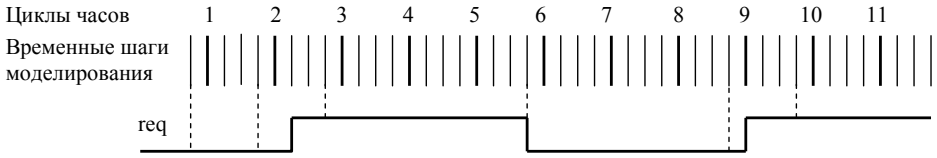

Рис.1 Лигика темпоральных взаимоотношений в параллельных (concarrent) утверждениях. Здесь значение reg фиксируется как низкое в моменты 1 и 2; в 3 – высокое; в 6 – низкое; 9 – низкое; 10 – высокое.

1.2 Конструирование SVA блоков

В модели любого проекта функциональность может быть представлена комбинацией множества логических событий. Этими событиями могут быть простые булевы выражения, которые вычисляются в тот же самый момент, или могут быть события, которые вычисляются через период времени, через много циклов. Последовательность (**sequence**) – это то, что представляет такие события (аналогично в[5]). Несколько последовательностей могут быть скомбинированы логически или последовательно для образования более сложных последовательностей. Свойство (**property**) представляет поведение этих сложных последовательностей. Само по себе

свойство ничего не доказывает. **assert (assume, cover)** проверяет property. При этом базовый синтаксис **assert** выглядит как: assertion_name: assert property (property name).

Таким образом, построение SVA-чекера (checker) выглядит следующим образом: a) создать булевы выражения;

b) создать последовательностные выражения; c) создать свойство; d) проверить (доказать) свойство.

1.3 Последовательности

Базовый синтаксис последовательности: sequence_expr ::= cycle_delay_range sequence_expr { cycle_delay_range sequence_expr }

1.3.1 Виды последовательностей

Свойство часто строится, исходя из последовательного поведения на пути продвижения времени, согласно его часам. Простейшее последовательное поведение линейно. *Линейная последовательность* - конечный список булевских выражений SV в линейном порядке увеличивающегося времени на конечном интервале срабатывания часов. Если вычисления булевских выражений дают true на каждом шаге часов, то свойство выполняется.

Последовательности могут быть составлены с помощью *конкатенации* в виде списка. В каждой конкатенации определяется значение задержки выполнения (используя символы ##) с конца выполнения одной последовательности в списке до начала следующей последовательности в списке. Последовательности с ##0 (нулевая задержка) – *перекрывающиеся последовательности*.

Пример конкатенации: (##1 b ##1 c) ##0 (d ##1 e ##1 f) - перекрывающаяся конкатенация.

1.3.2 Операторы последовательностей

Перечислим в порядке старшинства операторы SVA, которые позволяют проводить над последовательностями различные действия, и прокомментируем их.

Повторение в последовательности: - *последовательное* повторение (consequtive-repetition) **[*]**: b[*2] ## 1a эквивалентно b##1 b##1 a; - повторение *с переходом* (**go_to**-repetition) **[->]**: a##1 b[->2:10]##c эквивалентно a##1 ((!b[*0:$]##1 b)[*2:10])##1 c; - *непоследовательное* повторение (nonconsequtive-repetition) **[=]** аналогично go_to-repetition, но с расширением последовательности на произвольное число циклов часов: a##1 b[=2:10]##1 c эквивалентно a##1 ((!b[*0:$]##1 b)[*2:10])##1 !b[*0:$]##1 c.

Задержка считывания в циклах часов **##**: ##2 a эквивалентно 'true##1 'true##1 a.

Условия на последовательности: **throughout**: exp **throughout** seq – последовательность выполняется, если выражение exp выполняется на всем интервале выполнения seq; **within**: seq1 **within** seq2 - последовательность выполняется, если в течение выполнения seq2 выполняется seq1; **intersect**: seq1 **intersect** seq2 - последовательность выполняется, если seq1 и seq2 одновременно начинаются и одновременно успешно заканчиваются.

Оператор **and**: seq1 **and** seq2 последовательность выполняется, если seq1 и seq2 при одновременном старте обе выполняются. Оператор **and** используется, когда ожидают соответствие для обоих операндов, но времена окончания последовательностей операндов могут быть различными: sequence_expr ::= ... | sequence_expr **and** sequence_expr.

Оператор **or**: seq1 **or** seq2 последовательность выполняется, если при одновременном старте seq1 или seq2 выполняется. Оператор **or** используется, когда ожидается, что, по крайней мере, одна из двух последовательностей операндов будет иметь соответствие: sequence_expr ::= ...sequence_expr **or** sequence_expr. Если в в качестве одного из операндов используется последовательность с конкатенацией, то проверка на соответствие составной последовательности может дать положительный результат в точках всего диапазона задания конкатенации последовательности.

Оператор **first_match**. Оператор **first_match** выбирает только первое соответствие из возможно многих соответствий последовательности его операнда при попытке вычисления. Например, в упомянутом выше операторе **or** с последовательностью с конкатенацией применение оператора **first_match** будет подразумевать окончание результирующей последовательности в момент первого совпадения.

1.3.3 Методы последовательностей

Существуют три метода для определения конечной точки последовательности: **ended**, **triggered** и **matched**. Вызов этих методов осуществляется при помощи следующего синтаксиса: sequence_instance.sequence_method.

1.4 Свойства

1.4.1 Объявление свойств.

Свойство определяет поведение проекта. *Именованное свойство* может использоваться при верификации как предположение, для проверки утверждения или как спецификация покрытия.

Для объявления именованного свойства используется **property** конструкция: property_declaration ::= **property** property_identifier[([property_port_list])]**;** {assertion_variable_declaration} property_spec **;** **endproperty** [**:** property_identifier]

1.4.2 Операторы свойств

Действия со свойствами осуществляются с помощью операторов, описанных ниже в порядке их предшествования, за исключением операторов последовательности, которые требуют круглых скобок.

Оператор **disable iff** может быть приписан к *property_expr* при реализации *property_spec*. Выражение оператора **disable iff** называют *блокирующим условием* (*disable condition*). Оператор **disable iff** позволяет задать приоритетные сбросы. Вычисление *property_spec* - это параллельное вычисление его свойства (*property_expr*) и блокирующего условия оператора **disable iff**. Если до завершения вычисления свойства блокирующее условие становится истинным, то дальнейшее вычисление свойства блокируется.

Операторы формы **strong** и **weak.** Свойство, именуемое еще свойством последовательности, имеет три формы: *sequence_expr*, **weak**(*sequence_expr*) и **strong**(*sequence_expr*). Свойство **strong**(*sequence_expr*) принимает значение true тогда и только тогда, когда существует непустое соответствие *sequence_expr*. Свойство **weak**(*sequence_expr*) принимает значение true тогда и только тогда, когда не существует никакого конечного префикса, который свидетельствовал бы о невыполнимости соответствия *sequence_expr*. Если **strong** или **weak** оператор опущен, то вычисление *sequence_expr* зависит от предложения утверждения, в котором оно используется.

Оператор отрицания **not.** Оператор **not** переключает значение свойства и силу свойства.

Оператор дизъюнкции **or**: property_expr **or** property_expr. Свойство вычисляется как true тогда и только тогда, когда, по крайней мере, одно из *property_expr1* и *property_expr2* вычисляется как true.

Оператор конъюнкции **and**: property_expr **and** property_expr. Свойство вычисляется как true тогда и только тогда, когда и *property_expr1* и *property_expr2* вычисляются как true.

Оператор условий **if-else.** Может иметь две формы: **if (**expression_or_dist**)** property_expr либо **if (**expression_or_dist**)** property_expr **else** property_expr. Свойство первой формы вычисляется как true тогда и только тогда, когда либо *expression_or_dist* вычисляется как false, либо *property_expr* вычисляется как true. Свойство второй формы вычисляется как true тогда и только тогда, когда либо *expression_or_dist* вычисляется как true и первое *property_expr* вычисляется как true, либо *expression_or_dist* вычисляется как false и второе *property_expr* вычисляется как true.

Импликация. Импликация определяет, что проверка свойства выполняется условно при выполнении левого операнда. property_expr ::= ... | sequence_expr | -> property_expr или | sequence_expr | => property_expr . В этой конструкции используется предусловие (левый операнд) при проверке выражения свойства (правого операнда). Определены две формы импликации: перекрывающаяся (оператор |->) и неперекрывающаяся (оператор |=>). Для перекрывающейся импликации: если существует соответствие предшествующего *sequence_expr*, тогда конечная точка этого соответствия является стартовой точкой вычисления последующего *property_expr*. Для неперекрывающейся импликации: стартовая точка вычисления последующего *property_expr* есть следующее срабатывание часов после конечной точки соответствия.

Операторы **implies** и **iff.** Свойство *property_expr1* **implies** *property_expr2* принимает значение true тогда и только тогда, когда либо *property_expr1* принимает значение false, либо *property_expr2* принимает значение true. Свойство вида *property_expr1* **iff** *property_expr2* принимает значение true тогда и только тогда, когда либо и *property_expr1*, и *property_expr2* принимают значение false, либо и *property_expr1*, и *property_expr2* принимают значение true.

Операторы **nexttime** и **s_nexttime**. **nexttime** property_expr (слабая форма оператора nexttime) принимает значение true тогда и только тогда, когда либо выражение *property_expr* принимает значение true, на следующем срабатывании часов, либо последующего срабатывание часов нет; **s_nexttime** property_expr (сильная форма nexttime) принимает значение true тогда и только тогда, когда существует следующее срабатывание часов и *property_expr* принимает значение true, на этом срабатывании часов.

Операторы **always** и **s_always**: **always** *property_expr* принимает значение true тогда и только тогда, когда *property_expr* выполняется на каждом текущем или будущем срабатывании часов. **always** [*cycle_delay_const_range_expression*] *property_expr* принимает значение true тогда и только тогда, когда *property_expr* выполняется на каждом текущем или будущем срабатывании часов, внутри диапазона, заданного значением *cycle_delay_const_range_expression*. Существование всех срабатываний часов в этом диапазоне не является обязательным. **s_always** [*constant_range*] *property_expr* принимает значение true тогда и только тогда, когда все текущие или будущие срабатывания часов, заданные значением *constant_range*, существуют, а *property_expr* выполняется на каждом из этих срабатываний часов. Сильная форма оператора always допускается только для ограниченного диапазона.

Операторы **until, s_until, until_with, s_until_with**: property_expr1 **until** property_expr2 (слабая форма без перекрытия), property_expr1 **s_until** property_expr2 (сильная форма без перекрытием), property_expr1 **until_with** property_expr2 (слабая форма с перекрытием), property_expr1 **s_until_with** property_expr2 (сильная форма с перекрытием). **until** в форме без перекрытия принимает значение true, если *property_expr1* принимает значение true на каждом срабатывании часов, начиная с первой попытки вычисления, и продолжает выполняться как минимум до одного срабатывания часов раньше срабатывания часов, при котором *property_expr2* принимает значение true. **until** в любой из форм с перекрытием принимает значение true, если *property_expr1* принимает значение true при каждом срабатывании часов, начиная с первой попытки вычисления, и вплоть до (включительно) срабатывания часов, на котором *property_expr2* принимает значение true. **until** в любой из сильных форм требует существования текущего или будущего срабатывания часов, в котором *property_expr2* принимает значение true, в то время как в любой из слабых форм свойства **until**, такое требование отсутствует. **until** в любой из слабых форм принимает значение true, если *property_expr1* принимает значение true во всех срабатываниях часов, даже если *property_expr2* не выполняется никогда.

Операторы **eventually, s_eventually**: **s_eventually** property_expr (сильная форма), **eventually[** constant_range**]** property_expr (слабая форма с диапазоном),

s_eventually[cycle_delay_const_range_expression**]** property_expr (сильная форма с диапазоном). Слабая форма оператора **eventually** допустима только в ограниченном диапазоне и принимает значение true тогда и только тогда, когда либо существует текущее или будущее срабатывание часов в диапазоне, заданном значением *constant_range*, в котором *property_expr* истинно, либо текущее срабатывание или не все будущие срабатывания часов внутри диапазона, заданного *constant_range*, существуют. Сильное свойство **s_eventually** *property_expr* принимает значение true тогда и только тогда, когда существует текущее или будущее срабатывание часов, в котором *property_expr* принимает значение true. Сильное свойство eventually с диапазоном **s_eventually** [*cycle_delay_const_range_expression*] *property_expr* принимает значение true тогда и только тогда, когда существует текущее или будущее срабатывание часов в диапазоне, заданном значением *cycle_delay_const_range_expression*, в котором *property_expr* принимает значение true. Диапазон для сильной формы свойства типа eventually может быть неограниченным.

Операторы преждевременного прерывания: **accept_on (**expression_or_dist**)** property_expr, **reject_on (**expression_or_dist**)** property_expr, **sync_accept_on (**expression_or_dist**)** property_expr, **sync_reject_on (**expression_or_dist**)** property_expr, где *expression_or_dist* называется *условием прерывания.* Для вычисления **accept_on** (*expression_or_dist*) *property_expr* и **sync_accept_on** (*expression_or_dist*) *property_expr* сначала вычисляется соответствующее *property_expr*. Если условие прерывания становится истинным, то вычисление свойства в целом дает результат true. В противном случае результат вычисления свойства в целом равен результату вычисления выражения *property_expr*. Для вычисления **reject_on** (*expression_or_dist*) *property_expr* и **sync_reject_on** (*expression_or_dist*) *property_expr* необходимо вычисление соответствующего *property_expr*. Если во время вычисления условие прерывания становится истинным, то вычисление свойства в целом дает результат false. В противном случае - результату вычисления выражения *property_expr*. Операторы **accept_on** и **reject_on** вычисляются с дискретностью временного шага моделирования, как и **disable iff**, но их условие прерывания вычисляется с использованием выбранных значений как регулярное логическое выражение в утверждениях. Операторы **accept_on** и **reject_on** представляют асинхронные сбросы. Операторы **sync_accept_on** и **sync_reject_on** вычисляются с дискретностью временного шага моделирования, когда происходит событие синхронизации. Операторы **sync_accept_on** и **sync_reject_on** представляют собой синхронные сбросы.

1.4.3 Безопасность и живучесть

Операторы свойств **s_nexttime**, **s_always**, **s_eventually**, **s_until**, **s_until_with** и оператор последовательности **strong** являются сильными: они требуют того, чтобы некоторое терминальное событие происходило в будущем, а это требует достаточного количества срабатывания часов свойства, чтобы это условие могло быть выполнено. Операторы свойств **nexttime**, **always**, **until**, **eventually**, **until_with** и оператор последовательности **weak** являются слабыми: они не налагают никаких требований на терминальные условия и не требуют срабатывания часов.

Концепция слабых и сильных операторов тесно связана с важным понятием свойств безопасности. Свойства безопасности характеризуются тем, что их отказы происходят в конечное время, они обладают конечными контрпримерами. Например, свойство **always** aa относится к этому типу, так как оно нарушается, только если после конечного числа срабатываний часов существует срабатывание часов, в котором это свойство ложно, даже если количество срабатываний часов в расчете бесконечно велико. Напротив, отказ свойства **s_eventually** aa в расчете с бесконечно большим количеством срабатываний часов не может быть определен за конечное время. Такие свойства известны как свойства живучести, и они характеризуются тем, что для них не существует конечного контрпримера.

1.4 Поддержка множественной синхронизации

В контексте этого подраздела понятие синхронизация тождественно понятию часов свойства, а наступление синхронизующего события в свойстве (последовательности) это – срабатывание часов свойства.

1.4.1 Последовательности с множественной синхронизацией

Последовательности с множественной синхронизацией строятся соединением подпоследовательностей с единственной синхронизацией с использованием оператора конкатенации с единичной задержкой (##1) или с нулевой задержкой (##0). Единичная задержка, обозначаемая ##1, понимается как интервал от конечной точки первой последовательности, которая совпадает с фронтом первого синхроимпульса, до ближайшего, строго последующего, фронта второго синхроимпульса, с которого начинается вторая последовательность. Нулевая задержка, обозначаемая ##0, понимается как интервал от конечной точки первой последовательности, которая совпадает с фронтом первого синхроимпульса, до ближайшего, возможно, пересекающегося во времени, фронта второго синхроимпульса, с которого начинается вторая последовательность.

@(**posedge** clk0) sig0 ##1 @(**posedge** clk1) sig1

В примере соответствие последовательности начинается с соответствия sig0 в момент **posedge** clk0. Затем оператор задержки ##1 переводит время к ближайшему строго последующему моменту **posedge** clk1, и соответствие этой последовательности заканчивается в этот момент с соответствием sig1.

При конкатенации последовательностей с различной синхронизацией от каждой максимально разделенной подпоследовательности с единственной синхронизацией требуется допущение непустого соответствия.

1.4.2 Свойства с множественной синхронизацией

Синхросигнал может быть явно задан для любого свойства. Свойство называется свойством с множественной синхронизацией, если некоторые из его подсвойств обладают синхросигналом, отличным от синхросигнала основного свойства.

2. SVA в среде SystemVerilog

Выше уже упоминалось, что SVA использует основные конструкции языка, вводя лишь собственные ограничения, обусловленные его функциональностью. Здесь будет рассмотрен еще ряд вопросов, касающихся реализации в рамках системы SV, а также подмножества системных функций, используемых в рамках SVA.

2.1 Планирование

SV определяется как язык, использующий событийную базовую модель. Считается, что множество событий приходится на каждый временной слот [7]. Три временных области отводятся для вычисления и выполнения утверждений:- **Prepond** - значения для переменных утверждений выбираются в этой области. В этой области переменные не меняют свое состояние- **Observed** - все выражения свойств вычисляются в этой области. –**Reactive -** pass/fail коды, полученные при вычислении свойств, размещаются в этой области.

2.2 Системные функции значений

2.2.1 Функции анализа содержимого аргумента

Утверждения обычно используются, чтобы оценить определенные особенности выполнения проекта. Существуют следующие системные функции для облегчения описания в утверждениях такого функционирования: $onehot (<expression>) возвращает true, если только один бит выражения – high; $onehot0 (<expression>) возвращает true, если больше одного бита выражения – high; $isunknown (<expression>) возвращает true, если какой-либо бит в выражении X или Z; $countones, подсчитывающая число единиц в битовом векторе, при этом значения X и Z не считаются: $countones (expression).

2.2.2 Функции от выбранных значений

Функция $sampled возвращает выбранное значение выражения: $ sampled (expression).

Следующие функции обеспечивают возможность обнаружить изменение значений между двумя смежными срабатываниями часов: $rose(expression [, [clocking_event]]); $fell(expression [, [clocking_event]]); $stable(expression [, [clocking_event]]); $changed(expression [, [clocking_event]]).

Следующие функции позволяют получить доступ к выбранному значению выражения в предыдущем и следующем срабатывании глобальной синхронизации: $past_gclk (expression); $rose_gclk (expression); $fell_gclk (expression); $stable_gclk (expression); $changed_gclk (expression).

Следующие функции позволяют обнаруживать изменения текущего выбранного значения от выбранного значения в предыдущем срабатывании глобальной синхронизации или изменения текущего выбранного значения от значения в следующем срабатывании: $future_gclk (expression); $rising_gclk (expression); $falling_gclk (expression); $steady_gclk (expression); $changing_gclk (expression).

2.3 Системные функции покрытия

В SystemVerilog есть несколько встроенных системных функций для получения информации о тестовом покрытии: $coverage_ control, $coverage_get_max, $coverage_get, $coverage_merge и $coverage_save.

SystemVerilog также предоставляет системные задачи и системные функции для управлении множеством данных о покрытии и для выдачи отчетов: $set_coverage_db_name, $load_coverage_db и $get_coverage.

2.4 Вызов подпрограмм в последовательности

Задачи, задачи-методы, void функции, void функции-методы и системные задачи могут вызываться в конце успешного непустого соответствия последовательности.

В заключение следует сказать, что реализация SVA SV-2009 нашел свое отражение в MGC Questa_Sim версий, начиная с v.10.0. Система моделирования развивается и совершенствуется, в том числе и в плане полноты реализации SVA.

Автору не известно ни одной публикации ни на русском языке, ни в зарубежной литературе, описывающей столь лаконично, полно и целостно конструкции языка SVA SystemVerilog_2009.

Литература

1. *Яицков А.С.* Сравнительный обзор новой версии SystemVerilog // Информационные технологии в образовании, науке и бизнесе: труды конференции IT+S&E'2010. Ялта-Гурзуф. 2010.

2. Описание языка SystemVerilog – Unified Hardware Design, Specification, and Verification Language: сборник научных трудов / под ред. В.Б. Бетелина, П.П. Кольцова, А.С. Яицкова. – М.: НИИСИ РАН, 2010. Часть III. – 146 с.

3. *Яицкова Г.А.* Функциональное покрытие в SystemVerilog // Информационные технологии в образовании, науке и бизнесе: труды конференции IT+S&E'2012. Ялта-Гурзуф. 2012.

4. *Яицкова Г.А.* Рандомизация в SystemVerilog // Информационные технологии в образовании, науке и бизнесе: труды конференции IT+S&E'2013. Ялта-Гурзуф. 2013.

5. *Яицков А.С., Яицкова Г.А., Семашко А.В., Лагутина А.М.* Property Specification Language. Русифицированная версия: сб. Современные средства формальной верификации проектов / под ред. В.Б. Бетелина. – М.: НИИСИ РАН, 2008.

6. *Яицков А.С., Яицкова Г.А.* Формальная верификация проектов на основе утверждений // Информационные технологии в образовании, науке и бизнесе: материалы конф. IT+S&E'2008. Ялта-Гурзуф. 2008.

7. Описание языка SystemVerilog – Unified Hardware Design, Specification, and Verification Language: сборник научных трудов / под ред. В.Б. Бетелина, П.П. Кольцова, А.С. Яицкова.– М.: НИИСИ РАН, 2009, Ч. I. – 144 с.

SystemVerilog Assertios of SystemVerilog-2009

Galina Alexanrovna Yaitskova, Candidate of Technical Sciences, Senior research associate
Research Institute of System Researches of Russian Academy of Sciences

The internal SystemVerilog Assertion of SystemVerilog-2009 order as a way of informational safety ensuring with using of assertions tools is presented.

Keywords: assertion, sequence, property, synchronized sequence.

УДК 519.716.32+519.854

КООРДИНАТНО-ЛИНЕЙНО РАЗРЕШИМЫЕ ФУНКЦИИ НАД ПРИМАРНЫМ КОЛЬЦОМ ВЫЧЕТОВ И МЕТОД ПОКООРДИНАТНОЙ ЛИНЕАРИЗАЦИИ

***Мирослав Владимирович Заец**, сотрудник*
Тел.:(916) 475-31-06, e-mail: mirzaets@hotmail.com
Федеральное государственное унитарное предприятие «Научно-исследовательский институт «Квант» ФГУП «НИИ КВАНТ»
www.rdi-kvant.ru

В статье рассматриваются и изучаются свойства нового класса функций над примарным кольцом вычетов, который обобщает класс полиномиальных функций и определенный ранее класс функций с вариационно-координатной полиномиальностью в работах [1], [2], [3]. Данные классы функций обладают тем свойством, что системы уравнений, составленные из таких функций, могут быть решены методом покоординатной линеаризации ([4], [5]).

Ключевые слова: функции с вариационно-координатной полиномиальностью, функции с координатно-линейной разрешимостью, полиномиальные функции, системы линейных уравнений, метод покоординатной линеаризации.

Введение

Известно, что системы полиномиальных уравнений над кольцом Галуа-Эйзенштейна (т.е. конечным коммутативным цепным кольцом) могут быть решены методом покоординатной линеаризации [5; 4]). Частным случаем такого кольца является примарное кольцо вычетов $\mathbb{Z}_{p^m}, m \in \mathbb{N}$. Суть рассматриваемого метода над $\mathbb{Z}_{p^m}$ заключается в последовательном нахождении p-ичных координат неизвестных переменных, при этом нахождение $(i+1)$-х координат при известных координатах меньшего порядка, сводится к решению системы линейных уравнений над полем $GF(p)$. В статьях [1; 2] было показано, что класс функций над кольцом вычетов $\mathbb{Z}_{2^m}$, обладающий таким свойством, шире класса полиномиальных при $m \geq 3$. Построенный класс был назван классом «вариационно-координатно полиномиальных функций» (ВКП-функций). В данной работе определяется расширение класса ВКП-функций (и как следствие, полиномиальных), а именно класс функций с координатно-линейной разрешимостью (КЛР-функций). Приводятся свойства введенного класса, а также описывается метод покоординатной линеаризации для решения систем уравнений, составленных из таких функций.

М.В. Заец

1. Определение и свойства координатно-линейно разрешимых функций

Обозначим класс всех полиномиальных функций от $n \in \mathbb{N}$ переменных над кольцом $\mathbb{Z}_{p^m}$ через $\boldsymbol{\mathcal{P}}_{p^m}(n)$. Договоримся функции от переменных $x_1, \dots, x_n$ записывать кратко $f(\mathbf{x}) = f(x_1, \dots, x_n)$, класс всех функций от n переменных над кольцом вычетов