УДК 681.326:519.613

С.А. ЗАЙЧЕНКО, В.И. ХАХАНОВ

ПРОЕКТИРОВАНИЕ САМОТЕСТИРУЕМЫХ ЦИФРОВЫХ СИСТЕМ НА ОСНОВЕ АППАРАТНОЙ РЕАЛИЗАЦИИ МОНИТОРОВ ТЕМПОРАЛЬНЫХ АССЕРЦИЙ

Рассматривается метод аппаратной реализации проверки логико-временных ограничений поведения цифровых систем. Получили дальнейшее развитие принципы проектирования встроенной диагностической инфраструктуры, обслуживающей функциональные модули цифровых систем на кристаллах. Демонстрируются возможности для уменьшения параметра time-to-market за счет использования аппаратной эмуляции ассерционных мониторов в процессе тестовой функциональной верификации по сравнению с традиционными и более простыми программными решениями.

1. Введение

Развитие индустрии проектирования и изготовления цифровых систем на кристаллах (SoC – System-On-Chip [1]) определяется совокупным влиянием нескольких ключевых факторов, отражающих рыночную потребность:

1) Функциональность SoC. Существует диктуемая спросом необходимость расширения реализуемых в чипе функций, поддержки новых интерфейсов приема/передачи информации, увеличения объема необходимой для хранения данных встроенной памяти. Эти факторы усложняют структуру SoC, приводят его к нежелательному физическому расширению (*area growth*) .

2) Быстродействие. Возрастает необходимость обмена и обработки больших объемов данных, выполнения сложных вычислительных операций за меньшее время. Это требует увеличения тактовых частот, граничащих с физическими параметрами задержек передачи сигналов в технологии реализации чипа (*speed growth*).

3) Энергопотребление. Требуется синхронизация большого количества элементов цифровой системы, разделенных длинными цепочками соединений. Сложность схемы синхронизации устройства приводит к проблеме повышенного потребления энергии (*power consumption growth*).

Технологии изготовления чипов характеризуются различными параметрами *area*, *speed* и *voltage*. Очевидно, большая плотность интеграции элементов на единицу площади уменьшает размеры устройства, увеличивает его скорость и уменьшает энергопотребление. Согласно [2], производительность системы (параметр *performance*) определяется произведением перечисленных трех факторов и значительно растет в технологиях с большей степенью интеграции (рис. 1).

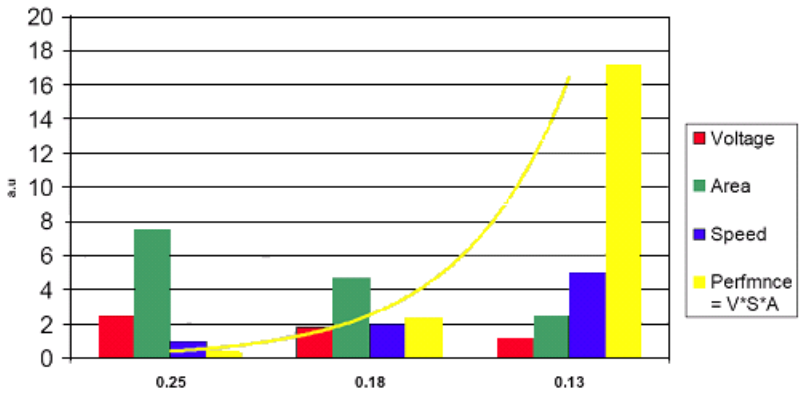

Рис. 1. Соотношение параметров производительности SoC в зависимости от интеграции

Вместе с производительностью также увеличивается вероятность появления дефектов в процессе изготовления: существенно ухудшается параметр *yield* (выход годной продукции, %). В частности, практически неизбежно наличие дефектов при изготовлении встроенных в SoC элементов памяти. Очевидно, конечной целью проектирования всегда является получение продукта, отвечающего требованиям спецификации, а не отсутствие дефектов в SoC, как таковых. Проблему повышения параметра *yield* решают декомпозицией функциональности на меньшие блоки *(leveraging)* и введением избыточности (*redundancy*) [3]. Функцию отключенного небольшого дефектного блока может реализовать подключенный резервный (spare) блок, находящийся в кристалле.

Актуальной задачей проектирования является создание внутренней инфраструктуры SoC (I-IP), осуществляющей самотестирование, диагностирование дефектных блоков, а также автоматическое восстановление работоспособности [4]. В функциональный блок может быть встроен набор тестов, логика обработки результатов встроенного теста и система динамического реконфигурирования связей. Самотестирование системы основано на подаче встроенного тестового воздействия на входы внутреннего блока (недоступного напрямую извне) и мониторинге его реакции. Процесс воздействия и диагностирования управляется встроенным в SoC микропроцессором.

Традиционным подходом является автоматическая генерация теста, ориентированного на полное покрытие дефектов, и составление таблиц неисправностей для поиска дефектов в случае их обнаружения [5]. В данном исследовании рассматривается альтернативный метод, ориентированный не на выявление дефектов, а на подтверждение возможности выполнения существенной для SoC функциональности.

В основу идеи положено использование ассерций – логико-временных ограничений поведения цифровых систем, базирующихся на формальном математическом аппарате линейной темпоральной логики (LTL – Linear Temporal Logic) [6,7]. Наиболее популярным языком описания ассерций является PSL (Property Specification Language) [8]. Ассерции формально задают требования к функциональности системы. Они типично применяются для функциональной верификации моделей: существуют как формальные методы проверки ассерций [9], так и подходы к их анализу в рамках тестовых методов верификации [10,11].

Перспективной сферой применения ассерций является инфраструктура самотестирования SoC. В работе рассматривается возможность синтеза аппаратного представления монитора, проверяющего темпоральные утверждения [12]. Процесс синтеза основан на ранее разработанной модели динамических регистровых очередей (DRTLQ – Dynamic RTL Queues), осуществляющих программную валидацию ассерций во время моделирования [13,14]. Синтезированный аппаратный ассерционный монитор лучше отражает основные функциональные требования, а также обладает дополнительными преимуществами при локализации дефектов.

Возможность аппаратной реализации ассерционного монитора может быть также использована для ускорения функциональной верификации. Такой подход перспективен в сочетании с технологией HES (Hardware Embedded Simulation) компании Aldec Inc. [15]. С применением программно-аппаратной ко-симуляции тестируемые блоки цифровой системы вместе с синтезированными ассерционными мониторами могут быть размещены в программируемом аппаратном прототипе. Подача тестовых воздействий и анализ отладочной информации при этом осуществляется программно.

Цель исследования – разработка метода аппаратной реализации мониторов темпоральных ассерций, ориентированной на создание встроенной развитой диагностической инфраструктуры цифровых систем на кристаллах, что дает возможность улучшить параметр time-to-market за счет аппаратного ускорения процесса тестовой функциональной верификации.

Задачи исследования:

1) Определение подмножества линейной темпоральной логики с детерминированной семантикой, реализуемой в виде структур уровня регистровых передач.

2) Разработка методов синтеза аппаратного представления ассерционных мониторов.

3) Сравнение временных характеристик вычислительных процессов проверки темпоральных ассерций при аппаратной и программной реализации, выявление ограничений обоих подходов.

4) Создание модели процесса ускоренной тестовой функциональной верификации с размещением синтезированного ассерционного монитора в аппаратном прототипе.

2. Формальная семантика LTL-формул в языке PSL

Пусть требуется установить справедливость LTL-формулы f для верифицируемой системы M. Как система, так и формула могут быть приведены к виду автоматов Buchi [16]:

$$A = \{Q, \Sigma, \partial, I, F\}, \quad (1)$$

где Q – множество всех состояний системы; Σ – алфавит системы; ∂ – функция перехода между состояниями по набору утверждений из алфавита; $I \in Q$ и $F \in Q$ – подмножества начальных и удовлетворяющих (штатных) состояний системы соответственно.

Пусть также имеется слово w, представляющее собой последовательность булевых продвижений a_0, a_1, a_2... в рамках алфавита системы Σ, такой, что каждому i соответствует некоторое состояние системы $q_i \in Q$, и для всех i выполняется соотношение $(q_i, a_i, q_{i+1}) \in \partial$. Тогда множество L(A) всех возможных слов w, порождаемых автоматом A, называется языком, принимаемым (понимаемым) автоматом. Задача определения справедливости LTL-формулы сводится к сопоставлению языков, принимаемых автоматом, соответствующим верифицируемой системе, и автоматом, сконструированным для LTL-формулы, обратной интересующей:

$$L(A(M)) \bigcap L(A(\neg f)) = \varnothing. \quad (2)$$

Ниже перечислены операторы LTL:

1) Базовые операторы:

– $\pi \models b$ (справедливость булевого выражения на единичном вычислительном пути);
– $\pi \models \neg f \Leftrightarrow \pi \not\models f$ (отрицание формулы);
– $\pi \models f \vee g \Leftrightarrow (\pi \models f) \vee (\pi \models g)$ (дизъюнкция);
– $\pi \models f \wedge g \Leftrightarrow (\pi \models f) \wedge (\pi \models g)$ (конъюнкция);
– $\pi \models X!f \Leftrightarrow \pi_1 \models f$ (оператор next!);
– $\pi \models f \ U \ g$ (оператор until!).

2) Производные операторы:

– $\pi \models F\ f \Leftrightarrow [true\ U\ f]$ (оператор eventually!);
– $\pi \models G\ f \Leftrightarrow \neg F\ f \Leftrightarrow \neg[true\ U\ f]$ (оператор always);
– $\pi \models [f\ W\ g] \Leftrightarrow [f\ U\ g] \vee \neg[true\ U\ f]$ (оператор weak until).

В языке PSL введены собственные, более выразительные с точки зрения пользователя, последовательностные операторы, семантика которых основывается на перечисленных базовых LTL-операторах [17]. Среди введенных операторов имеются такие:

– временного сдвига;
– репетиции;
– последовательностной импликации;
– асинхронного прерывания.

Далее рассматривается семантика этих операторов более подробно в целях последующего сравнения возможностей программной и аппаратной реализации мониторов утверждений на их основе:

1. Оператор непересекающегося временного сдвига (non-overlapping time shift) применяется для конкатенации последовательностей событий (sequence concatenation) и означает начало выполнения второй последовательности на следующем тактовом цикле после завершения выполнения первой последовательности:

```
default clock = posedge clk;
sequence s1 = {a & b};
sequence s2 = {c | d};
sequence s1_s2 = {s1 ; s2}
```

$$\pi \models \{f;g\} \Leftrightarrow \exists i, \pi = \pi^{0..i}\pi^{i+1}, (\pi^{0..i} \models f) \wedge (\pi^{i+1} \models g). \quad (3)$$

2. Оператор пересекающегося временного сдвига (overlapping time shift) применяется для склеивания последовательностей событий (sequence fusion) и означает начало выполнения второй последовательности на тактовом цикле, где завершается выполнение первой последовательности:

default clock = **posedge** clk;
sequence s1 = {a & b};
sequence s2 = {c | d};
sequence s1_s2 = {s1 : s2}

$$\pi \models \{f : g\} \Leftrightarrow \exists i, \pi = \pi^{0..i}\pi^{i+1}, (\pi^{0..i} \models f) \wedge (\pi^{i} \models g). \quad (4)$$

3. Оператор консекъютивной последовательностной репетиции (consecutive repetition) применяется для повторения определенной последовательности-операнда заданное количество раз таким образом, чтобы начало выполнения очередной итерации производилось на следующем тактовом цикле после завершения предыдущей итерации:

default clock = **posedge** clk;
sequence s1 = {a & b};
sequence s1_3 = s1[*3:5];

$$\pi \models f[*i:k], i \le k \Leftrightarrow \overbrace{\{f;f;\cdots f\}}^{\text{i раз}} \vee \overbrace{\{f;f;\cdots f\}}^{\text{i+1 раз}} \vee \cdots \vee \overbrace{\{f;f;\cdots f\}}^{\text{k раз}}. \quad (5)$$

Также существует форма последнего оператора с бесконечным интервалом:

default clock = **posedge** clk;
sequence s1 = {a & b};
sequence s1_inf = s1[*];

$$\pi \models f[*] \Leftrightarrow \exists i, \pi = \pi^{0..i}\pi^{i+1}, (\pi^{0..i} \models f) \wedge (\pi^{i+1..} \models f[*]). \quad (6)$$

Язык PSL предоставляет пользователю дополнительные производные синтаксические формы операторов репетиции, покрывающие ряд частных случаев, таких как i=k, i=0 или 1. Последовательность-операнд также может быть опущена в описании. Это означает пропуск заданного числа циклов, что по сути эквивалентно репетиции булевого выражения, всегда принимающего истинное значение.

4. Оператор неконсекъютивной булевой репетиции (non-consecutive repetition) применяется только для булевых выражений и означает его многократное повторение, при котором нет обязательного требования следования текущей итерации непосредственно на следующем такте после завершения предыдущей:

default clock = **posedge** clk;
sequence s1 = (a && b)[=3];

$$\pi \models b[= k] \Leftrightarrow \{\{\neg b[*];b\};\neg b[*]\}. \quad (7)$$

Разновидностью оператора неконсекъютивной репетиции является оператор переходной репетиции (goto repetition), который гарантирует завершение итерации истинным значением выражения-операнда:

default clock = **posedge** clk;
sequence s1 = (a && b)[->3];

$$\pi \models b[\rightarrow k] \Leftrightarrow \{\neg b[*];b\}. \quad (8)$$

5. Оператор принадлежности последовательности другой последовательности:

default clock = **posedge** clk;
sequence s1 = {a ; b ; c};
sequence s2 = {d ; e[*4]};
sequence s3 = s1 **within** s2;

$$\pi \models f \text{ within } g \Leftrightarrow \{[*];f;[*]\} \&\& g. \quad (9)$$

6. Оператор последовательностной импликации, означающий требование успешного выполнения последовательности событий при условии успешного выполнения другой последовательности событий:

default clock = **posedge** clk;
sequence s1 = {a ; b};
sequence s2 = {c ; e[*4]; d};
sequence s3 = s1 **|->** s2; $\pi \models f \mid\rightarrow g \Leftrightarrow \forall i, 0 \le i < |\pi|, \pi^{0..i} \models f, \pi^{i..} \models g$. (10)

7. Оператор асинхронного прерывания последовательности, применяемый в случае необходимости сброса процесса анализа LTL-формулы в начальное состояние:

default clock = **posedge** clk;
sequence s1 = {a ; b ; c};
property p1 = s1 **async_abort** rst;

$\pi \models f$ **async_abort** $b \Leftrightarrow$
$\forall i, 0 \le i < |\pi|, \pi^{0..i} \models f, \pi^{i..i+1} \models b$. (11)

Язык PSL предоставляет возможность ссылок на значения выражения в предыдущих тактовых циклах (оператор *prev*).

Также немалый интерес представляют высокоуровневые операторы-свойства в языке PSL, позволяющие выражать отношения между другими свойствами и последовательностями:

- $f \rightarrow g \Leftrightarrow \neg f \vee g$ (импликация свойств);
- $f \leftrightarrow g \Leftrightarrow (f \wedge g) \vee (\neg f \wedge \neg g)$ (двусторонняя импликация свойств);
- always $f \Leftrightarrow G\ f$ (выполнение свойства на всех вычислительных путях);
- **n**ever $f \Leftrightarrow \mathbf{G}\ \neg f$ (запрет на выполнение свойства на всех вычислительных путях);
- **eventually!** $f \Leftrightarrow [f\ \mathbf{U}\ g]$ (выполнение свойства на одном из вычислительных путей);
- f **until_** $g \Leftrightarrow [f\ \mathbf{U}\ (f \wedge g)]$ (оператор until с пересечением последовательностей);
- семейства операторов next, next_event и before.

3. Синтез аппаратной модели ассерционных мониторов из модели DRTLQ

В работе [13] представлена модель динамических регистровых очередей, представляющая собой модель процесса программной верификации темпоральных утверждений. Предложенная модель, в первую очередь, рассчитана на ее применение при функциональной верификации системы с использованием тестовых воздействий в рамках процесса симуляции. При этом подходе система проверки темпоральных формул сообщает о ее работоспособности на каждом из вычислительных путей независимо друг от друга. Тем самым она предоставляет детальную диагностическую информацию. Семантика языка PSL при этом ограничивается так называемым «простым подмножеством».

В работе [14] предлагается альтернативный вариант интерпретации семантики LTL-формул – так называемый «режим глобального времени». В данном варианте модели процесса верификации за счет отказа от детальной диагностической информации достигается более высокое быстродействие анализа, а также значительно расширяется набор поддерживаемых конструкций языка PSL.

Оба подхода используют понятие DRTLQ-*события*, несущего информацию о состоянии обработки вычислительного пути в данный момент, номера тактов порождения и активации события, а также связи с другими событиями. Каждое из них всегда относится к конкретному анализируемому вычислительному пути. Последний обычно имеет одну начальную точку, но, поскольку формулы могуть содержать ветвления, допустимо наличие нескольких событий, связанных с одним и тем же вычислительным путем. Все события, соответствующие одному вычислительному пути, связаны между собой и образуют «кольцо активации». Это дает возможность уничтожения всей цепочки событий сразу после завер-

шения одного из альтернативных сценариев вычислительного пути. Кроме того, каждый анализирующий элемент в модели DRTLQ может обрабатывать несколько событий одновременно, относящихся к разным путям. Вполне допустима ситуация, когда результат для более позднего вычислительного пути известен ранее, чем результат более раннего маршрута.

Очевидно, аппаратная реализация мониторов темпоральных формул существенно отличается от программного подхода, где их обработка базируется на возможности динамического выделения памяти для событий и связей между ними. Также при аппаратной реализации сложнее обеспечить вывод диагностической информации, поскольку число одновременно наблюдаемых выходов всегда ограничено размером используемой шины обмена данных. Тем не менее, синтез аппаратной модели мониторов темпоральных утверждений является востребованной практической задачей, решение которой весьма желанно на рынке электронных технологий.

Пусть имеется последовательность событий *s1* = {*a*; [*2]; b}, где требуется высокий уровень сигнала *b*, следующий через 3 такта за аналогичным сигналом *a*. Существуют лишь две ситуации, когда такая последовательность может не выполниться успешно:

– сигнал *a != 1* на первом такте анализа;
– сигнал *b != 1* на четвертом такте анализа.

Во всех остальных случаях последовательность выполнится успешно и потребует 4 такта для выполнения. Проверку последовательности можно реализовать в виде конечного автомата [18], переходы в котором осуществляются по тактовому сигналу (рис. 2).

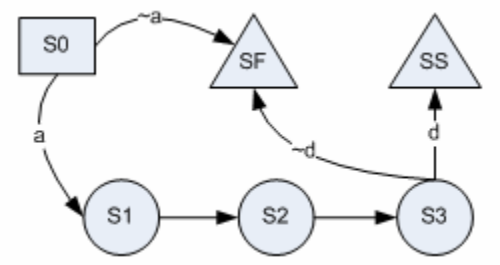

Рис. 2. Проверка последовательности *s1* при помощи конечного автомата

Завершение работы автомата в состоянии *SF* означает неудачу, в то время как состояние *SS* является индикатором успешного вычисления. Такая модель вполне пригодна для логического синтеза и может быть легко реализована аппаратно с использованием нескольких триггеров для индикации состояний и логических элементов для поддержания переходов между состояниями. В то же время подобный подход имеет существенный недостаток: отсутствие возможности одновременного анализа пересекающихся последовательностей, стартующих с различных начальных тактов. Необходимость параллельного анализа пересекающихся последовательностей в модели на основе конечных автоматов может быть удовлетворена дупликацией анализирующей логики. Очевидно, это приводит к значительному увеличению размерности проверяющего устройства.

Модель DRTLQ предлагает более компактную аппаратную реализацию, допускающую параллельный анализ пересекающихся последовательностей (рис. 3).

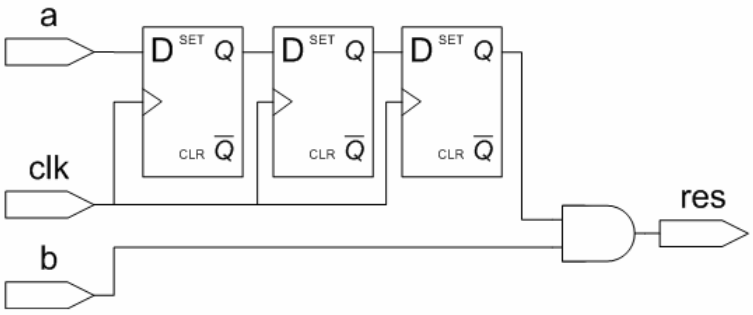

Рис. 3. Проверка последовательности *s1* при помощи аппаратного DRTLQ-монитора

Вместо объекта-события, присущего программной реализации модели DRTLQ, в аппаратной реализации продвижение анализируемого потока активации ассерции моделируется высоким уровнем сигнала. Очевидно, простой сигнал может означать лишь успешность или неудачу анализа и не несет никакой информации о времени своего создания, активации, завершения. Такие параметры могут быть установлены только на основе местоположения сигнала внутри реализующей ассерцию проверяющей схемы. Известно, что последовательность *s1* дает результат на 4-м такте после активации. Соответственно, результат проверки последовательности можно наблюдать на выходе *res* на 4-м такте с момента начала анализа. Высокий уровень выходного сигнала *res* означает успешное завершение анализа.

В программной реализации модели DRTLQ операторы временного сдвига реализуются при помощи FIFO-очередей. На каждом такте очередь считывает цепочку событий из входного порта и помещает ее в свои внутренние ячейки. Также по тактовому сигналу очередь осуществляет сдвиг цепочки событий на одну позицию по направлению от входа к выходу. Аппаратным представлением однотактной очереди может служить D-триггер, который функционирует аналогичным образом. Реализация многотактных очередей предполагает последовательное соединение нескольких D-триггеров в многобитовые регистры. Ширина каждого из них соответствует числу тактов ожидания события в очереди. Событие, ожидающее прохождения через очередь, хранится в одном из промежуточных триггеров, соответстующем номеру итерации ожидания.

На первом такте анализа последовательности *s1* входной сигнал *a* считывается первым триггером. Далее в течение двух тактов значение сигнала *a* на первом такте продвигается через регистр ожидания. Наконец, на 4-м такте это значение поступает на вход вентиля And, другим входом которого является уровень второго интересующего сигнала *b*.

Именно реализация очередей в виде регистров открывает возможность для параллельного анализа пересекающихся последовательностей. Продвигаемое событие, связанное с конкретным потоком активации, на выбранном такте может находиться лишь в некоторой части схемы, соответствующей этому такту. Элементы схемы, соответствующие остальным тактам, в это время находятся в состоянии ожидания. Более наглядно данный факт может быть продемонстрирован следующим образом (рис. 4):

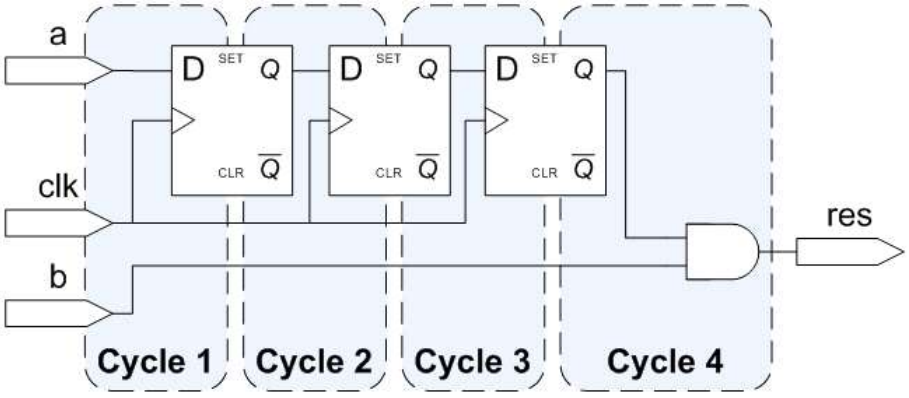

Рис. 4. Разделение элементов монитора последовательности *s1* на отдельные такты анализа

Элементы лишь одного тактового цикла одновременно задействованы при анализе, что можно использовать для организации конвейерной обработки. Когда последовательность, начатая на первом такте, находится на завершающем четвертом цикле, в предыдущих звеньях могут обрабатываться еще три последовательности, начатые на втором, третьем и четвертом тактах соотвественно. Таким образом, результаты анализа ассерции можно последовательно считывать на выходе схемы, на каждом такте. Ниже представлена временная диаграмма работы проверяющей схемы. Здесь ассерция вычисляется успешно, лишь начиная со второго такта, когда *a(2) = 1, b(5) = 1.* Во всех остальных начальных тактах требованию последовательности не отвечает один из интересующих сигналов (рис. 5).

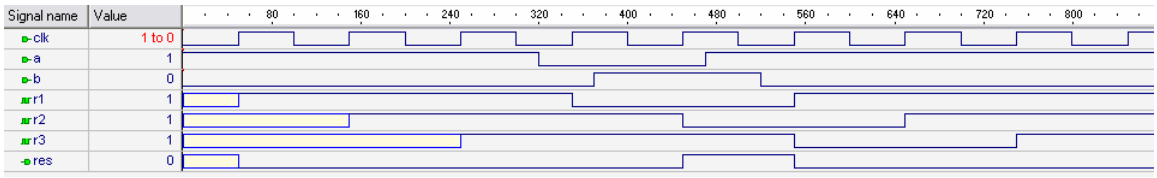

Рис. 5. Временная диаграмма работы монитора последовательности *s1*

Приведенная схема аппаратной реализации проверки темпоральных формул дает четкий ответ на вопрос об успешном или неудачном результате проверки, начиная с выбранного стартового такта. Это может быть достаточной информацией для выявления наличия функциональной проблемы как факта. Однако один лишь булевый (логический) результат не дает полной желаемой диагностической информации. Успешное выполнение последовательности событий может быть прервано на одном из промежуточных циклов. Информация о том, какой именно цикл оказал влияние на результат верификации в целом, может быть существенной для локализации причины функциональной проблемы. В рамках одного тактового цикла перечень ситуаций, приводящих к ошибке, типично существенно ограничен. Предоставление устройством-монитором информации о точном моменте выявления нарушения последовательности значительно упрощает последующие диагностические процедуры.

Далее рассматривается пример последовательности *s2* = {*a*[*2]; b}, означающей повторение высокого уровня сигнала *a* в течение двух тактов с начала последовательности, за которым следует высокий уровень сигнала *b* на третьем такте. Ниже представлена схема устройства-монитора для *s2* без дополнительных диагностических выходов (рис. 6).

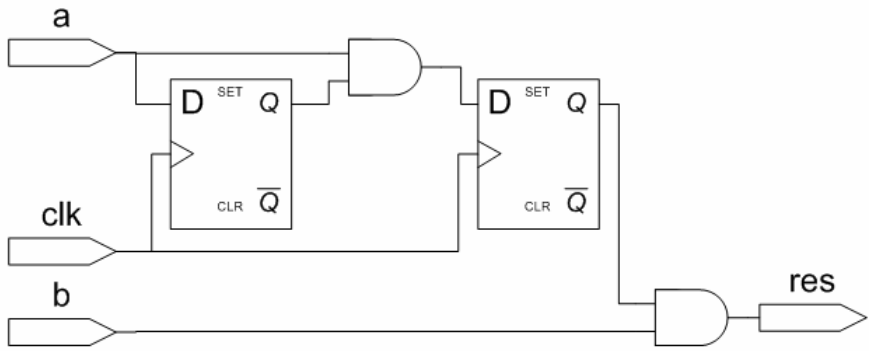

Рис. 6. Проверка последовательности *s2* при помощи аппаратного DRTLQ-монитора

Последовательность может неудачно завершиться на любом из тактов: низкий уровень сигнала *a* предопределяет неудачный итоговый результат проверки последовательности, как на первом, так и на втором такте. Введение дополнительных избыточных выходов в схему монитора позволяет проводить наблюдение за досрочно прерываемыми последовательностями (рис. 7).

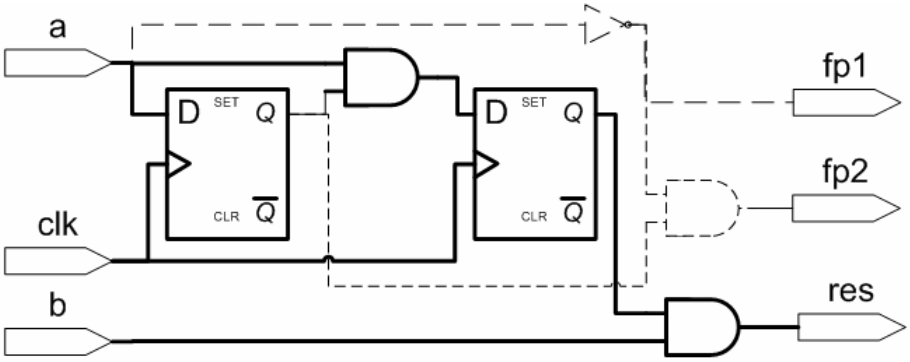

Рис. 7. Проверка *s2* при помощи монитора с диагностическими выходами

Введенные дополнительные выходы *fp1* и *fp2* называются *предикторами* (failure predictors) и сигнализируют о прерывании последовательности на первом и втором такте (рис. 8).

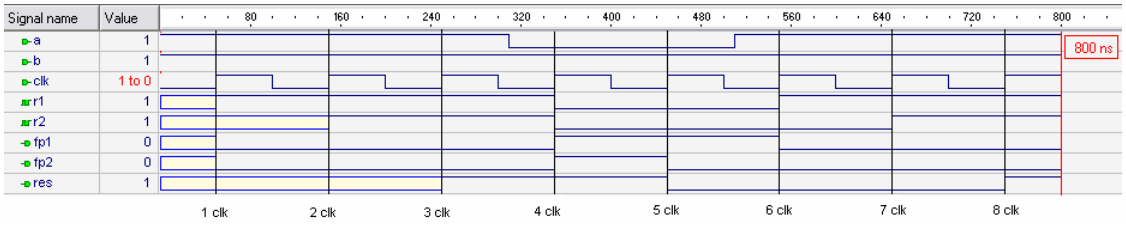

Рис. 8. Временная диаграмма работы монитора s2 с диагностическими выводами

Если выходы устройства-монитора подключить к программе-тестбенчу (под управлением симулятора Riviera компании Aldec. Inc [19]), то можно получить текстовую статистику анализа последовательности. Очевидно, что неудачный результат прерванных последовательностей, начатых на 3, 4 и 5 тактах, наблюдается сразу в момент прервывания:

KERNEL: Warning: Assertion PASSED at time: 250ns (3 clk), start-time: 50ns (1 clk)
KERNEL: Error: Assertion FAILED at time: 350ns (4 clk), start-time: 350ns (4 clk)
KERNEL: Error: Assertion FAILED at time: 350ns (4 clk), start-time: 250ns (3 clk)
KERNEL: Warning: Assertion PASSED at time: 350ns (4 clk), start-time: 150ns (2 clk)
KERNEL: Error: Assertion FAILED at time: 450ns (5 clk), start-time: 450ns (5 clk)
KERNEL: Warning: Assertion PASSED at time: 750ns (8 clk), start-time: 550ns (6 clk).

Одним из существенных преимуществ программной реализации модели DRTLQ является линейная структурная сложность монитора в зависимости от количества и вложенности входных языковых конструкций [13]. В качестве примера рассматривается последовательность с более сложным ветвлением: *s3* = {*a*[*1:2]; b[*1:2]; c}, которая подразрумевает 4 возможных успешных варианта событий, в зависимости от варьирования интервалов двух задействованных репетиций. При программной реализации монитора тело каждой репетеции представлено в модели в единственном экземпляре. Ветвление вариантов сценария достигается за счет создания связанных копий транспортируемых событий (одна копия продвигается далее, в то время как другая остается внутри репетиции). Если один из вариантов ветвления достигает монитор в успешном состоянии, все другие автоматически выводятся из рассмотрения путем уничтожения остальных копий через их связь с разрешающим событием. Последовательность завершается неудачно лишь тогда, когда все варианты приводят к отрицательному результату.

При аппаратной реализации структурной компактности достичь не удается. Принципиальным требованием для конвейерной обработки пересекающихся последовательностей является невозможность задействования одних и тех же логических и последовательностных элементов модели для хранения сразу нескольких событий. Соответственно, аппаратная реализация мониторов для последовательностей с ветвлением требует, чтобы каждый из вариантов развития событий был представлен структурно (рис. 9).

Самый короткий вариант последовательности занимает 3 такта, в то время как самый длинный – 5. Следует отметить выделенные пунктиром связи между точками раннего завершения последовательностей и входами синхронного сброса триггеров, соответствующих следующим, параллельно выполняемым ответвленным подпоследовательностям. Таким образом достигается эффект уничтожения параллельных веток событий, в анализе которых более нет необходимости.

В приведенную на рис. 9 модель монитора могут быть введены дополнительные диагностические выходы-предикторы:

– неудачного завершения на первом такте: *~a(1)*;
– неудачного завершения на третьем такте: *a(1) & ~b(2) & ~b(3)*;
– досрочного успешного завершения на третьем, четвертом тактах (соотвественно первые два входа элемента Or, определяющего значение выхода *res*).

Необходимость структурного представления всех вариантов развития событий в формулах с ветвлением является серьезным ограничением. В программной реализации моде-

ли DRTLQ последовательное соединение элементов временного сдвига и репетиции не вызывало никаких трудностей с ростом структурной сложности. А при аппаратной реализации количество задействованных элементов как произведение числа возможных вариантов, вносимых каждым из элементов последовательности.

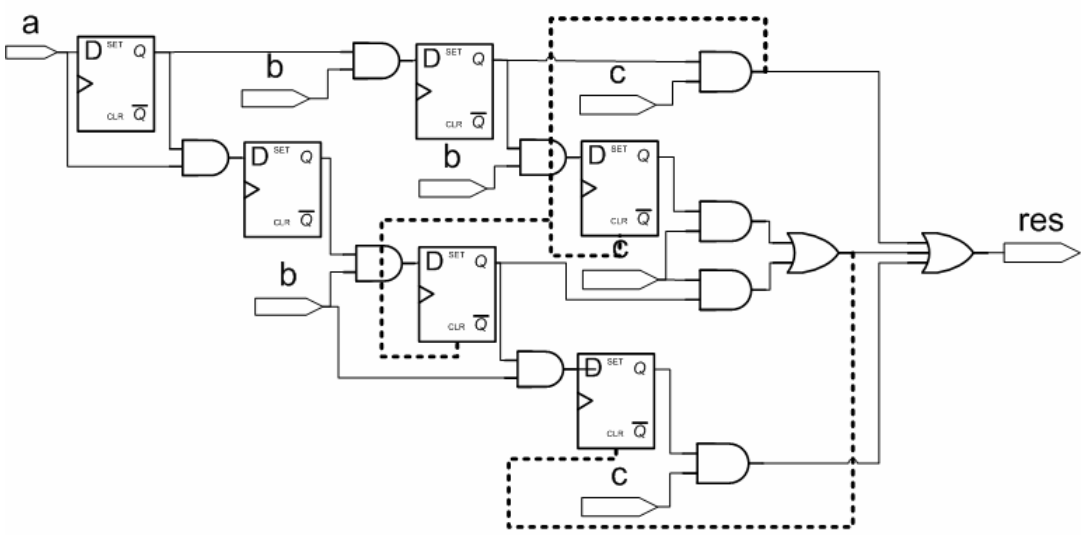

Рис. 9. Аппаратная реализация монитора последовательности *s3* с ветвлением

Такое ограничение делает невозможным параллельный анализ пересекающихся последовательностей недетерминированной длины. Например, невозможна реализация проверки последовательности *s4* = {*a*[*]; b}, означающей повторение высокого уровня сигнала *a* на протяжении некоторого интервала времени неизвестной длины до появления высокого уровня сигнала *b*. В данном недетерминированном случае более подходит модель на основе конечных автоматов. Такая последовательность может быть проанализирована только одиночно для выбранного начального такта. Анализ очередной последовательности может быть начат по завершению предыдущей, после асинхронного сброса всех используемых триггеров. В отличие от паралелльно анализируемых последовательностей, результаты одиночной последовательности можно наблюдать на выходах до момента начала анализа новой последовательности (рис. 10).

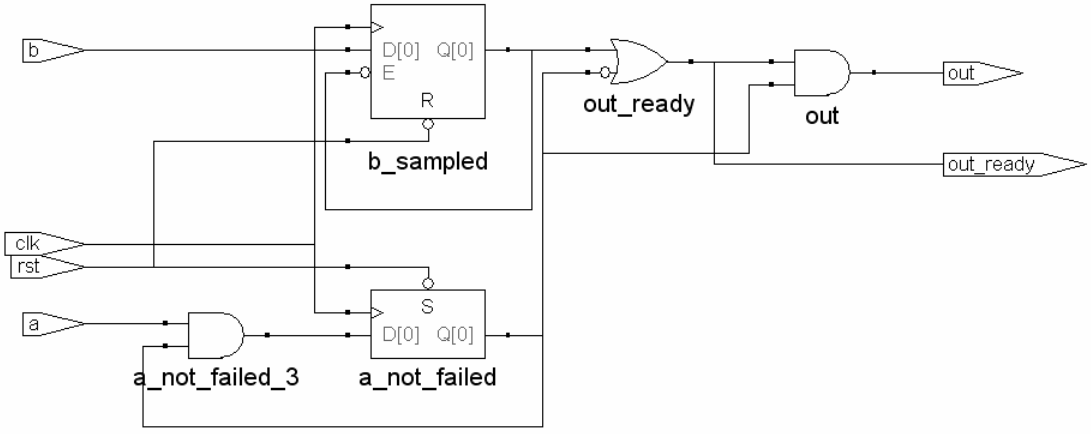

Рис. 10. Аппаратная реализация монитора одиночной последовательности *s4*

Применение последовательностных операторов AND/OR/INTERSECT, а также операторов последовательностной импликации (|=>, |->) вводит в модель большое количество промежуточных состояний и условий сброса, что делает затруднительным подготовку плоского монитора. В программной реализации модели DRTLQ вложенность последова-

тельностных операторов эффективно обрабатывывалась при помощи метода иерархического разрешения последовательностей по вложенным расширениям событий [14]. В аппаратной реализации применяется более очевидный иерархический подход. Такой способ реализации допустим и при программном моделировании. Однако упомянутый метод иерархического разрешения групп демонстрирует значительно лучшие характеристики производительности. Такая разница подходов обуславливается природой вычислительных процессов верификации. В аппаратной реализации элементы устройства-монитора функционируют параллельно, и отсечение части вычисленных результатов (например, при поступлении нуля на вход оператора AND) не имеет принципиального влияния на производительность анализа. Программная же модель обрабатывает элементы устройства-монитора последовательно. Своевременное блокирование избыточной активации, не приводящей к изменению результата вычисления в целом, дает существенный выигрыш в производительности.

Далее рассматривается пример *s5* = {*a*[*2]; b} & {c[*1:2]; d}, использующий функцию AND на последовательностях-операндах. Задача выполняется тогда и только тогда, когда обе последовательности операнда успешно завершаются. При этом не накладывается ограничений на одновременность завершения операндов, как в операторе INTERSECT. Поскольку операнды *s5* подобны случаям, рассмотренным на рис. 6 и 9, детали их реализации можно опустить. Известно, что первый операнд имеет 3 выхода: нормального завершения последовательности на 3-м такте, а также два выхода-предиктора неудачного досрочного завершения на 1-м и 2-м такте. Соответственно, второй операнд имеет 4 выхода: нормального завершения на 3 такте, выход-предиктор неудачного завершения на 1-м такте и два выхода-предиктора, соответствующие успешному и неудачному завершению на 2-м такте. Результирующая последовательность требует успешных результатов выполнения обоих операндов, поэтому предикторы неудачного завершения можно объединить вентилем OR. Успешное же завершение, напротив, потребует использования вентиля AND. Корректная обработка предиктора успешного завершения в случае оператора AND требует «выравнивания» времени его учета с результатом другого операнда путем внесения дополнительной однотактной очереди (рис. 11).

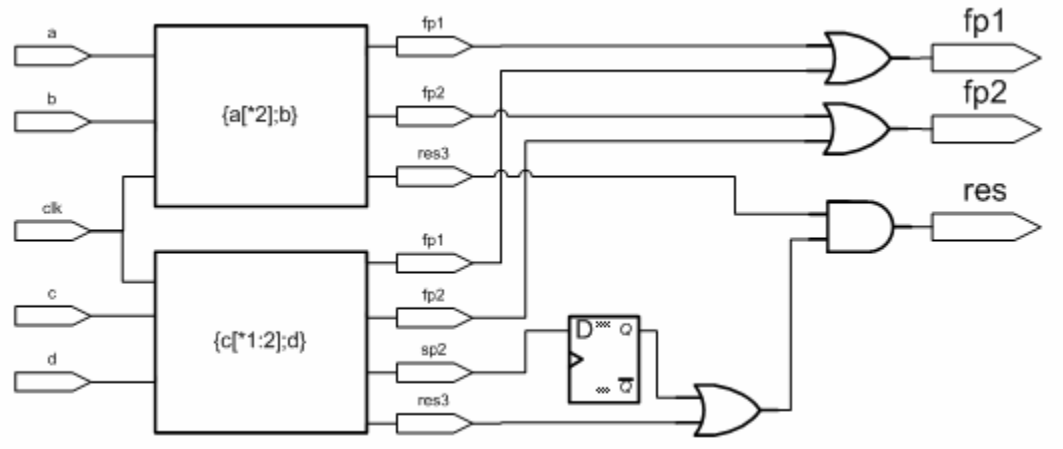

Рис. 11. Иерархический монитор для последовательности *s5*

Подобные манипуляции с выходами-результатами и -предикторами досрочного завершения последовательностей-операндов позволяют синтезировать мониторы полностью иерархически. В приведенном примере синтез операндов был произведен без какого-либо учета логики, находящейся на уровень иерархии выше. Более сложные случаи могут потребовать внесения дополнительных входов в мониторы-операнды для сброса с верхнего уровня иерархии состояния внутренних тригерров, соответствующих некоторому тактовому циклу (подобно случаю, рассмотренному на рис. 9).

4. Аппаратно-программный комплекс анализа ассерций на основе технологии HES

Типичный процесс программной верификации и диагностирования HDL-моделей цифровых устройств с использованием ассерций можно представить в следующем виде (рис. 12).

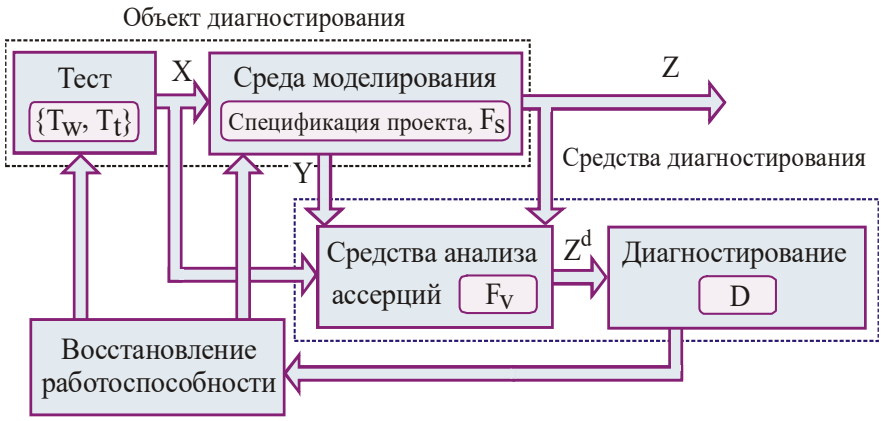

Рис. 12. Модель верификации и диагностирования на основе ассерций

HDL-описание проекта Fs вместе со связанными с ней описаниями ассерций Fv транслируется в системе симуляции в некоторое компилятивно-интерпретативное представление, помещаемое в рабочую библиотеку. Затем инициализируется процесс симуляции, и на модель проекта подаются тестовые воздействия. Дополнительно к основным выводам Z верифицируемого устройства средства анализа ассерций, встроенные в симулятор, предоставляют диагностическую информацию Z^d. При обнаружении ошибочного вывода применяются различные средства диагностирования (например, временные диаграммы, способные выводить состояние ассерций). Производится поиск и исправление ошибок – в проекте, в тесте или в самих ассерциях, после чего цикл верификации повторяется.

Отдельные звенья процесса верификации (рис. 12) могут быть реализованы с применением аппаратного ускорения (эмуляции) для достижения высокой производительности анализа. RTL-код верифицируемого блока подвергается логическому синтезу и вентильное представление блока помещается в FPGA, встроенное в плату HES. Аналогично, специальное средство синтеза генерирует вентильное представление анализируемых ассерций, также помещаемых в FPGA. Средство имплементации (специфичное для конкретного чипа) конфигурирует матрицу FPGA таким образом, что на вход ассерционных мониторов подаются реальные сигналы прототипа верифицируемого блока (рис. 13).

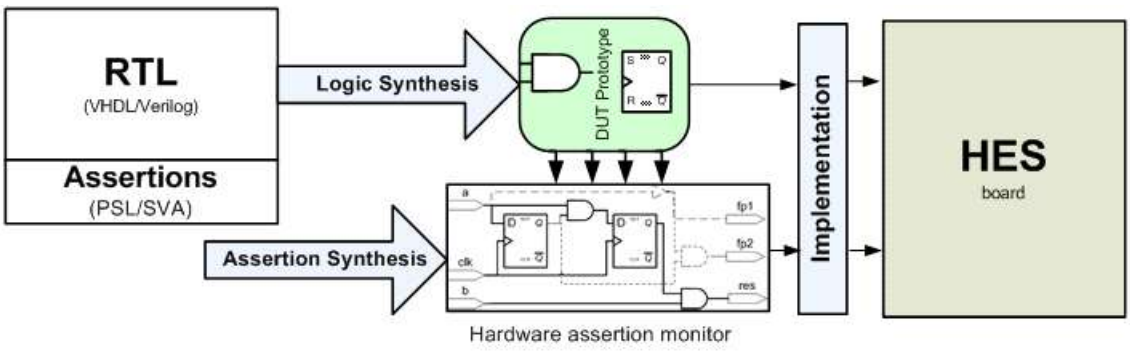

Рис. 13. Принцип аппаратного ускорения анализа ассерций

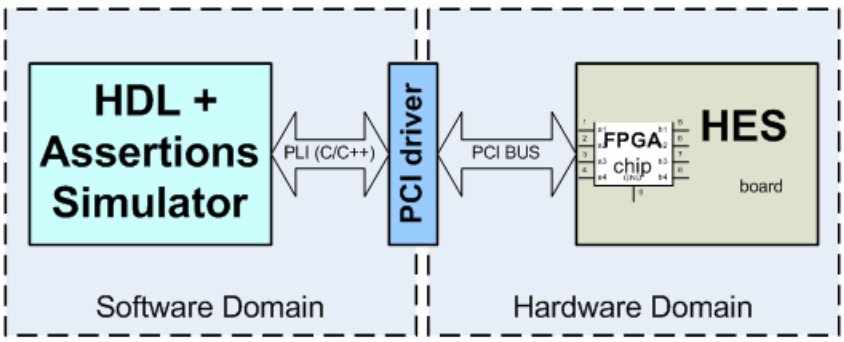

Рис. 14. Взаимодействие HDL-симулятора и платы-HES

Тестовые воздействия вместе со средствами диагностирования функционируют на стороне программного симулятора (рис. 14). Подача стимулов и считываение ответов устройства, а также выходов ассерций осуществляется через специальный слой системного программного обеспечения, управляемого на стороне тестбенча через стандартный интерфейс PLI. Непосредственное взаимодействие HES с программным обеспечением осуществляется через драйвер шины PCI, подключенной к плате. Скорость верификации в рассмотренной схеме в существенной мере определяется скоростью обмена данными между программной и аппаратной частями.

При этом скорость анализа ассерций фактически не зависит от сложности формул, которая влияет лишь на время синтеза вентильного представления ассерционных мониторов, а также на площадь, занимаемую монитором в FPGA-чипе. И хотя длина последовательностей безусловно влияет на результирующую частоту работы аппаратного прототипа, она, тем не менее, на 3-5 порядков выше аналогичного программного решения. В среднем, помещение основной вычислительной нагрузки в аппаратный прототип дает ускорение процесса верификации от 5 до 30 раз в зависимости от сложности проекта и ассерций.

5. Выводы

Предложенный в данной статье принцип аппаратной реализации проверки логико-временных отношений сигналов (ассерционных мониторов) может быть эффективно использован для решения таких важнейших задач:

1) Функциональная верификация SoC на различных этапах проектирования от моделей транзакционного уровня до этапов изготовления.

2) Создание развитой встроенной инфраструктуры самотестирования и восстановления работоспособоности SoC.

3) Ускоренная функциональная верификации на основе ассерций с применением аппаратного прототипирования (аппаратной эмуляции).

В ходе исследования были решены ключевые поставленные *задачи*:

1) Проанализирована семантика базовых и производных операторов линейной темпоральной логики, а также семантика расширенных последовательностных операторов языка PSL. Определено подмножество операторов с детерминированной семантикой, которую представляется возможным реализовать в аппаратной форме.

2) Разработаны методы синтеза аппаратного представления ассерционных мониторов с конвейерной архитектурой, способных производить одновременную проверку нескольких пересекающихся последовательностей событий. Ключевым преимуществом по сравнению с классической моделью на основе конечных автоматов является отсутствие необходимости структурного представления всех вариантов событий в аппаратном мониторе.

3) Произведено сравнение характеристик вычислительных процессов проверки темпоральных ассерций при аппаратной и программной реализации. Выявлены преимущества и ограничения обоих подходов. Предложены маршруты применения аппаратных и программных ассерционных мониторов в зависимости от этапа проектирования и целей верификации.

4) Предложена модель процесса ускоренной тестовой функциональной верификации систем на кристалле на основе размещения синтезированного аппаратного представления

ассерционных мониторов в FPGA-прототипе на основе технологии HES. Использование аппаратного прототипа значительно (на 3-4 порядка) ускоряет процесс функциональной верификации с использованием ассерций на ранних этапах проектирования.

Главный *научный результат* исследования состоит в разработке метода синтеза структур регистрового уровня, аппаратно реализующих семантику операторов расширенной линейной темпоральной логики, обеспечивающих возможность параллельного анализа пересекающихся последовательностей событий с сохранением структурной компактности модели.

Ключевым *практическим результатом* исследования является возможность уменьшения параметра time-to-market за счет встраивания в SoC мощной диагностической инфраструктуры, а также за счет аппаратного ускорения процесса тестовой функциональной верификации на ранних фазах проектирования.

Список литературы: 1. *Rashinkar P., Paterson P., Singh L.* System-on-chip Verification: Methodology and Techniques // Kluwer Academic Publishers, 2002. 393p. **2.** *Smith M.J.S.* Application-Specific Integrated Circuits, VLSI Systems Series, 1997, 1040p. **3.** *Zorian Y.* Design for Yield and Reliability // Proceedings of EWDTW'05, Odessa. P. 14. **4.** *Hsu Y.-C., Tabbara B., Chen Y.-A., and Tsai F.* Advanced techniques for RTL debugging // Proc. of the Design Automation Conf., 2003. P. 362–367. **5.** *Abramovici M., Breuer M.A. and Friedman A.D.* Digital systems testing and testable design // Computer Science Press. 1998. 652 p. **6.** *Emerson E.A.* Temporal and modal logic // In J. Van Leeuwen, editor, "Handbook of Theoretical Computer Science", Elsevier Science Publishers, 1990, volume B. P. 996–1072. **7.** *Clarke E.M.Jr, Grubmerg O., Peled D.A.* Model Checking // The MIT Process, 2002. 314p. **8.** *IEEE*-1850 "IEEE Standard for Property Specification Language (PSL)", 2005. 143p. **9.** *Drechsler R.* Advanced Formal Verification // Kluwer Academic Publishers, 2004. 277p. **10.** *Bergeron J.* Writing testbenches: functional verification of HDL models // Springer, 2003, 512p. **11.** *Foster H., Krolnik A., Lacey D.* Assertions-based Design // Kluwer Academic Publishers, 2003, 392p. **12.** *Forczek M, Hrynkiewicz K.* Formal properties evaluation, Proceedings of Programmable Devices and Systems Workshop 2003 (PDS-2003). P. 305-311. **13.** *Хаханов В. И., Зайченко С. А.* Модель динамических регистровых очередей для быстродействующего анализа линейных темпоральных ограничений", АСУ и приборы автоматики .2007. №136. С. 10-25. **14.** *Зайченко С. А., Хаханов В. И.* Эффективная функциональная верификация моделей цифровых систем на кристалле на основе ассерций глобального времени // АСУ и приборы автоматики. 2007. №137. С. 4-13. 15. http://www.aldec/com/products/hes. Доступно с экрана. **16.** *Geilen M., Dams D., Voeten, J.* Applying verification methods to Non-Exhaustive verification of Software/Hardware systems. Proceedings of CSSP-98, 9th Annual ProRISC/IEEE Workshop on Circuits, Systems and Signal Processing Mierlo, Netherlands, November 25-27. 1998. P. 177–183. **17.** *Havlicek J., Fisman D., Eisner C.* Basic Results on the Semantics of Accellera PSL 1.1 Foundation Language", Accellera Technical Report. 2004.57p. **18.** *Bormann J., Fedeli A., Frank R., Winkellman K.* Combined Static and Dynamic Verification, Research report, PROSYD forum, 2005. 19. http://www.aldec/com/products/riviera. Доступно с экрана.

Поступила в редколлегию 12.12.2007

Зайченко Сергей Александрович, аспирант кафедры АПВТ ХНУРЭ, начальник отдела разработки компании Aldec-Kharkov Ltd. Научные интересы: системы автоматизированного проектирования, моделирования и верификации цифровых систем на кристаллах. Увлечения: литература, музыка, футбол. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. (097)-367-62-93. E-mail: Sergei.Zaychenko@aldec.com

Хаханов Владимир Иванович, декан факультета КИУ ХНУРЭ, д-р техн. наук, профессор кафедры АПВТ ХНУРЭ. Научные интересы: техническая диагностика цифровых систем, сетей и программных продуктов. Увлечения: баскетбол, футбол, горные лыжи. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. (057)-702-13-26. E-mail: hahanov@kture.kharkov.ua