# Современные методы функциональной верификации цифровых HDL-проектов: методология ABV, библиотеки OVL и QVL

**Андрей Лохов (Москва)**

**В статье описана технология введения в цифровой проект формальной модели на основе ассертов, что позволяет формализовать процесс оценки качества созданного теста, ранее выполняемого интуитивно.**

В процессе развития методов, применяемых при проектировании цифровых систем, ситуация развивалась следующим образом. Сначала разработчики использовали только схемотехнику и проектировали цифровые блоки из элементарных вентилей, накапливая библиотеки отработанных блоков. Затем, с ростом количества элементарных вентилей, задействованных в проекте, постепенно произошёл качественный переход от схемотехники к языкам описания аппаратуры – HDL (Hardware Description Language). В настоящее время необходимость применения HDL-языков не вызывает сомнений. К маршруту проектирования, состоящему из основных этапов – создание HDL, моделирование и верификация, синтез, размещение и трассировка, – по мере необходимости добавляются новые этапы.

Однако с дальнейшим ростом объёма и функциональной сложности проектов время, требуемое на верификацию, увеличивается, доходя в отдельных случаях до 90% и более от всего времени разработки проекта, и всё чаще возникают проекты, в которых «стандартный» маршрут просто не работает. Например, функциональная сложность проекта может возрасти настолько, что команда инженеров по верификации не сможет предусмотреть все потенциальные проблемы и их верифицировать. Как следствие – законченный проект будет содержать функциональные ошибки.

Типичным и наиболее известным примером является ошибка, присутствовавшая в одном из первых процессоров серии Pentium фирмы Intel. Выход в продажу процессора, содержащего ошибку, чуть не привёл Intel к банкротству. Для решения новых проблем потребовалось усовершенствовать «классический» маршрут проектирования путём добавления новых подходов и концепций.

В данной статье будут рассмотрены следующие технологии, методологии, подходы и концепции:

- методология ABV (Assertion Based Verification – верификация, основанная на ассертах);
- библиотеки OVL и QVL.

## Формальная модель – ассерты

В настоящее время для верификации сложных цифровых проектов уже недостаточно просто создать HDL-описание поведения системы и приступить к написанию тестов для верификации, т.к. существует большая вероятность пропустить внутреннюю ошибку, которая либо не будет воспроизведена тестами, либо не приведёт к появлению неправильных данных на наблюдаемых сигналах.

Для «вылавливания» таких труднодиагностируемых ошибок ведущими разработчиками оборудования, такими как фирма Intel, наряду с функциональным HDL-описанием (HDL-моделью), разрабатывается формальная модель. Наиболее популярными инструментами описания формальной модели являются языки SystemVerilog Assertions и PSL – Property Specification Language. Оба языка реализуют схожую идеологию ассертов (от англ. *assertion* – утверждение), описывающих правила работы сигналов.

Основная идея методологии ABV заключается в том, чтобы формализовать информацию (знания о работе проекта), которой раньше владел только разработчик, и использовать её при моделировании в автоматическом режиме.

Для примера рассмотрим буфер FIFO. Текст тестбенча буфера на языке SystemVerilog может выглядеть так:

```
module fifo_tb;
// ...
// device under test
- DUT
FIFO #(.DEPTH(31), .WIDTH(8)) DUT
(
.CLK(clock),
.RSTb(nReset),
.DATA(data),
.Q(q),
.WENb(nWrite),
 .RENb(nRead),
.FULL(full),
.EMPTY(empty));
// ...
initial begin
//...
// Тестовые воздействия
// ...
end

//...
// Блок ассертов -
// проверка допустимости состоя-
ний буфера FIFO
//
property WriteFull;
@(posedge clock) !nWrite |-> full
!= 1;
endproperty
property ReadEmpty;
@(posedge clock) !nRead |-> empty
!= 1;
endproperty
assert property (WriteFull) else
$fdisplay(FD | 1, «%t: ASSERT:
Writing full FIFO!",
$realtime);
assert property (ReadEmpty) else
$fdisplay(FD | 1, «%t: ASSERT:
Reading empty FIFO!",
$realtime);
//
// Конец блока ассертов
endmodule
```

При «традиционном» подходе блок ассертов, показанный выше, не описывается в проекте, и менеджер по верифика-

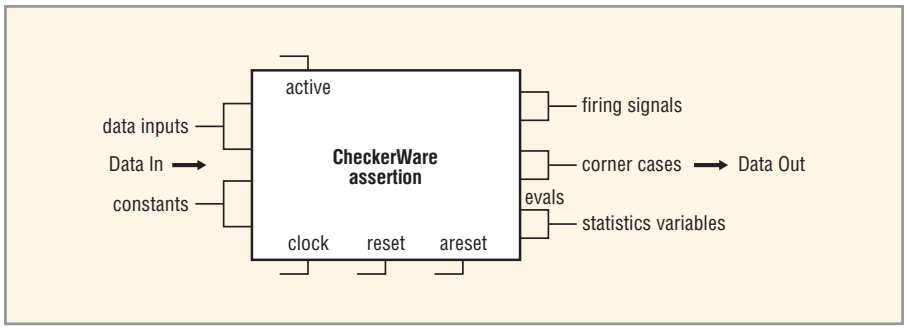



**Рис. 1. Условное представление библиотечного элемента**

ции должен либо проверять вручную требуемые ограничения (запись при переполнении или чтение из пустого буфера) путём наблюдения за диаграммами сигналов, либо написать дополнительный блок, который будет автоматически отслеживать соблюдение условия.

Применяя методологию ABV, разработчик проекта рядом с вхождением буфера формулирует дополнительные знания, которые раньше оставались не документированными, а именно, вставляет блок ассертов, который означает, что в процессе моделирования на каждом такте синхросигнала *clk* должно отслеживаться условие отсутствия выхода за рамки ограничений. При этом разработчик может задать способ реакции на нарушение заданного утверждения (ассерта): выдать сообщение, остановить моделирование и т.д. Обычно такие ассерты располагаются в важных (горячих) точках внутри проекта, например на стыке блоков.

Кроме описанной выше задачи контроля горячих точек проекта, ассерты позволяют выполнять ещё рад задач и в целом являются достаточно мощным и гибким инструментом верификации, в задачи которого входит:

- отслеживание выполнения утверждений (*assert*);
- поиск заданных последовательностей(*sequence*);
- генерация тестовых последовательностей;
- сбор статистической информации, например, о покрытии (*CoverPoint*).

## Библиотеки ассертов

Ассерты удобно использовать для контроля правильности работы проекта в т.н. «горячих точках». «Отслеживание» может заключаться не только в проверке конкретных сигналов и переменных, но и в накоплении дополнительной статистической информации, например, о полноте покрытия. Отчёт может быть представлен в виде одной или нескольких таблиц, например, при помощи различных менеджеров верификации.

Однако при использовании ассертов возникает ряд неудобств, связанных в основном с необходимостью изучения новых языковых конструкций и подходов, а также с возможностью допустить ошибку при составлении ассертов. Для решения данных проблем были разработаны библиотеки ассертов, которые позволяют снизить требования к знанию языка ассертов (SystemVerilog Assertions или PSL), т.е. разработчику придётся изучить только основы работы с ассертами, не вникая в детали.

Существуют различные варианты реализации этих библиотек, как свободно распространяемых, например OVL, так и коммерческих, например, QVL и CheckerWare.

В общем случае библиотечный элемент можно представить в виде блока, показанного на рисунке 1. Как правило, библиотечный элемент содержит следующие элементы:

- *Data In* – входные порты:
  - *data inputs* – входные порты для подключения к сигналам и переменным, которые необходимо верифицировать;
  - *constants* – порты для задания параметров блока;
  - *active* – активация блока;
  - *clock* – синхросигнал;
  - *reset/areset* – сброс.
- *Data Out* – выходные порты:
  - *firing signals* – сигналы – индикаторы, которые «загораются», т.е. принимают значение true, при нарушении проверяемых блоком условий;
  - *corner cases* – счётчики, показывающие, сколько раз блок зафиксировал заданные пограничные условия;
  - *statistics variables* – счётчики, отображающие различную статистическую информацию.

## Библиотеки OVL и QVL

Библиотеки OVL (Accellera's Open Verification Library) и QVL (Questa Veri-

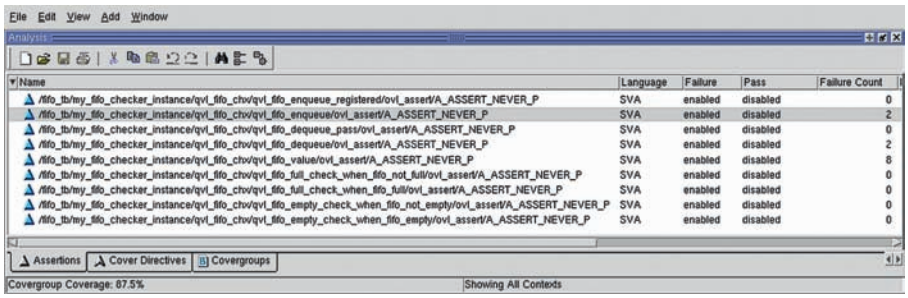



**Рис. 2. Ассерты, порождённые проверочным блоком**

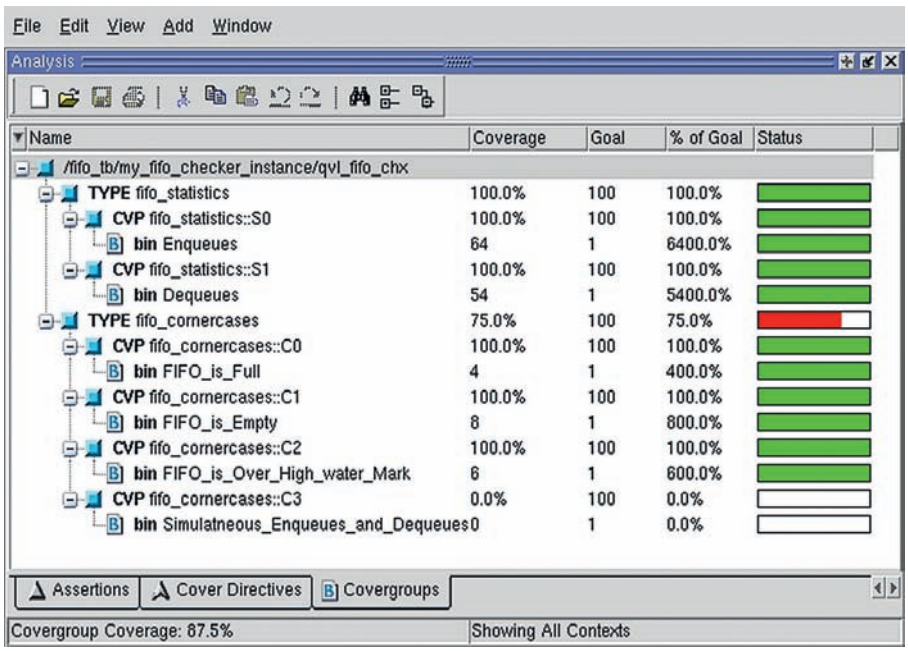



**Рис. 3. Группы событий, порождённые проверочным блоком FIFO**

fication Checkers Library) – это набор библиотечных ассертов, которые проверяют заданные параметры проекта. Эти библиотечные элементы подключаются к проекту посредством единого унифицированного интерфейса и содержат как проверочные блоки для тестирования простых структур, таких как счётчики, буферы, константы и т.д., так и элементы для тестирования сложных протоколов, таких как протоколы шин AMBA, PCI, USB, интерфейсов DDR и т.д., далее – мониторы.

Проверочные блоки (checkers) являются компонентами модулей, цель которых – гарантировать, что определённые условия всегда выполняются. Блоки состоят из одного или нескольких *property* (свойство), *message* (сообщения), *severity* (серьезность) и *cover points* (счётчики событий).

*Property* – это атрибут проекта, который надо верифицировать ассертом. Существует два вида *property*: комбинаторный и временной. Комбинаторный *property* отслеживает взаимоотношения между сигналами в пределах одного такта. Временной *property* отслеживает взаимоотношения между сигналами в разных тактах. Причём вполне возможно, что два отслеживаемых события будет разделять бесконечно большое количество тактов.

*Message* – это строка, которая будет выводиться в окно протокола работы системы в случае, если проверочный блок зафиксирует нарушение.

Параметр *severity* задаёт способ реакции на нарушение, например, остановить моделирование или только вывести сообщение и т.д.

*Cover points* – это флаги, счётчики или переменные, которые показывают, сколько раз произошло интересующее событие. *Cover points*, содержащиеся в QVL-компонентах, можно разделить на два класса: статистические и крайние случаи. Статистические *cover points* измеряют активность различных частей компонента проверочного блока. Например, количество тактов, в которых выполнялось измерение.

Флаги крайних случаев отслеживают пограничные состояния. Это могут быть редкие или необычные состояния, «сложно достижимые» состояния или различные нетипичные состояния контролируемого модуля. «Правильный» испытательный трассировщик должен покрывать все эти состояния, т.к. они являются типичными источниками ошибок. Для примера рассмотрим буфер FIFO:

```
`include «std_ovl_defines.h"
`include «std_qvl_defines.h"
module fifo_tb;
// ...
// device under test
- DUT
FIFO #(.DEPTH(31), .WIDTH(8)) DUT
(
.CLK(clock),
.RSTb(nReset),
.DATA(data),
.Q(q),
.WENb(nWrite),
.RENb(nRead),
.FULL(full),
.EMPTY(empty));
// ...
initial begin
// ...
// Тестовые воздействия
// ...
end
// ...
// Чекер qvl_fifo
//
qvl_fifo #(

.severity_level(`QVL_ERROR),
        .property_type(`QVL_ASSERT),
        .msg("QVL_VIOLATION : «),

.coverage_level(`QVL_COVER_NONE),
        .depth(31),
        .width(8),
        .pass(0),
        .registered(0),
        .high_water(0),
        .full_check(0),
        .empty_check(1),
        .value_check(1),
        .latency(0),
        .preload_count(0))
        my_fifo_checker_instance(
                .clk(clock),
                .reset_n(nReset),
                .active(nReset),
                .enq(~nWrite),
                .deq(~nRead),
                .full(full),
                .empty(empty),
                .enq_data(data),
                .deq_data(q),
                .preload(data_pre-
load)
        );
endmodule
```

Для проверки правильности работы этого буфера, помимо вставки в проект самого элемента, необходимо добавить экземпляр проверочного блока FIFO из библиотеки QVL – *qvl_fifo*. В такой конфигурации экземпляр *my_fifo_checker_instance* будет отслеживать критически важные события и в окне протокола работы выдавать сообщения типа:

```
? OVL_ERROR : ASSERT_NEVER :
QVL_VIOLATION : Dequeued FIFO
value did not equal the corre-
sponding enqueued value. : Test
expression is not FALSE : sever-
ity 1 : time 2450 ns :
fifo_tb.my_fifo_checker_in-
stance.qvl_fifo_chx.qvl_fifo_valu
e.ovl_error_t
```

Как видно из приведённого выше примера вывода диагностики, проверочные блоки не только проверяют правильность входных и выходных сигналов, но и контролирует сложное функциональное поведение элемента, например, эквивалентность записанных и считанных данных. Принципиальное отличие библиотеки QVL от OVL заключается в наличии в QVL т.н. мониторов, т.е. элементов для диагностики таких сложных устройств, как шины PCI или AMBA, интерфейс USB и т.д.

Для удобного анализа всех статистических данных можно воспользоваться средствами анализа ассертов, сводящими все ассерты в таблицы. Например, на рисунке 2 показана такая таблица, из которой видно, что событие *qvl_fifo_enqueue*, т.е. попытка записи в полный буфер, возникало два раза, событие *qvl_fifo_dequeue*, т.е. попытка чтения из пустого буфера, возникало два раза, и, наконец, событие *qvl_fifo_value*, т.е. прочитанное значение из буфера не соответствует ожидаемому, возникало восемь раз.

Механизм ассертов позволяет описывать и выполнять достаточно сложную аналитику в автоматическом режиме. Например, проверочный блок FIFO при помощи ассертов формирует аналитическую структуру, показанную на рисунке 2. При этом разработчик тестов может, основываясь на результатах анализа, оценить, насколько полон созданный им тест. Например, на рисунке 3 видно, что один «крайний» случай не был проверен, а именно – *Simulatneous_Enqueue_and_Dequeue* (одновременное чтение и запись данных), из чего можно сделать вывод, что существующие тесты требуют доработки.

Как видно из данного примера, введение в проект формальной модели на основе ассертов позволяет формализовать процесс оценки качества созданного теста, выполняемый ранее исключительно интуитивно.

## Заключение

Механизм ассертов предоставляет новые методы верификации, существенно повышающие производительность труда разработчиков тестов для цифровых устройств. В данной статье демонстрируются преимущества только одного из них, а именно возможность использования ассертов и библиотек ассертов для контроля «горячих» точек проекта.

## Литература

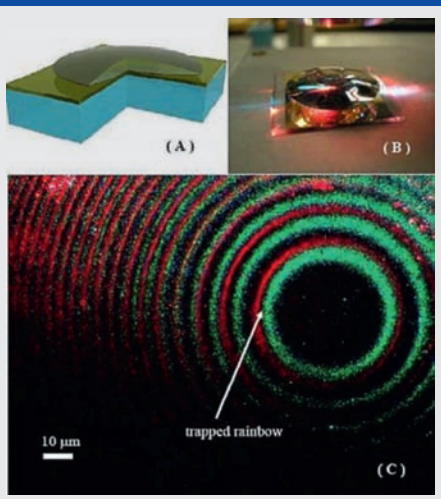

1. Questa SV/AFV User's Manual.
2. Questa Verification Library Checkers Data Book.
3. Questa Verification Library Monitors Data Book.


# Новости мира News of the World Новости мира

## Учёные поймали радугу

Оказывается, поймать радугу всё-таки возможно. Это сделано впервые с помощью простых линз, пластины стекла и золота, но не из праздного любопытства: техника может быть использована для хранения информации в виде света и развития оптических вычислений и телекоммуникаций. Оптические устройства обещают быть быстродействующими и более эффективными, чем нынешняя электроника, однако сложность представляет необходимость конвертирования сигналов из оптического вида в электрический и обратно. «Замедление» света или локализация в некотором пространстве может разрешить проблему с прямой обработкой электромагнитных волн.

В 2007 г. Ортвин Хесс (Ortwin Hess) из Университета Сюррея (University of Surrey) в Гилфорде, Великобритания, вместе с коллегами предложил технологию заключения света в сужающемся волноводе, который является структурой, направляющей излучение по своей длине. В его состав входят метаматериалы. Идея состоит в том, что по мере сужения волновода компоненты света должны поочерёдно останавливаться во всё более узких точках, поскольку никакой компонент не может пройти через открытое пространство, меньшее его длины волны. Таким образом и получается «пойманная радуга». Многочисленные модели показывают, что подобные «конусообразные» волноводы должны действовать, но до сих пор их изготовление из метаматериалов остаётся неразрешимой задачей. Однако Вера Смолянинова (Vera Smolyaninova) из Таусонского университета (Towson University) в Балтиморе совместно с другими исследователями использовала выпуклую линзу, чтобы создать требуемую структуру волновода и локализовать радугу.




Учёные покрыли одну из сторон линзы диаметром 4,5 мм золотой плёнкой толщиной 30 нм и поместили её на плоскую стеклянную пластину, также покрытую плёнкой из благородного металла, позолоченной стороной вниз. Если посмотреть на эту систему сбоку, то воздушное пространство между изогнутой поверхностью линзы и пластиной постепенно становится меньше вплоть до нулевой толщины в точке, где линза касается стекла, – практически тот же сужающийся волновод. Когда учёные направили в него многоволновое лазерное излучение, внутри сформировалась радуга. Она имела вид серии цветных колец при рассматривании линзы сверху через микроскоп.

Зелёный свет с более короткой длиной волны оказался в точке пространства, слишком узкого для преодоления этим излучением. Красный свет с большей длиной волны был пойман в более широкой части волновода. Между ними расположились остальные цвета. По словам исследователей, тот факт, что столь комплексное явление удалось воспроизвести с помощью очень простой конфигурации, удивителен.

***newscientist.com***