ФУНКЦИОНАЛЬНАЯ ВЕРИФИКАЦИЯ ЦИФРОВЫХ HDL-ПРОЕКТОВ:

МЕТОДОЛОГИЯ НА ОСНОВЕ АССЕРТОВ

Механизм ассертов предоставляет много новых методов и подходов к верификации, существенно повышающих производительность труда разработчиков тестов для цифровых устройств. В статье [1] дан общий обзор методов расширенной функциональной верификации. На сей раз мы более детально продемонстрируем преимущества одного из них, основывающегося на использовании ассертов и библиотек ассертов для контроля за "горячими" точками проекта.

А.Рабоволюк
al1@megratec.ru

В ходе развития методов проектирования цифровых систем разработчики сначала использовали только схемотехнику и проектировали цифровые блоки из элементарных вентилей, накапливая библиотеки отработанных блоков. Затем, с ростом числа элементарных вентилей, задействованных в проекте, постепенно произошел качественный переход от схемотехники к языкам описания аппаратуры – HDL (Hardware Description Language). Соответственно, современный маршрут проектирования включает такие основные этапы, как создание HDL-описания аппаратуры, ее моделирование и верификация, синтез схемы, размещение и трассировка элементов. По мере необходимости добавляются и дополнительные этапы. Однако с ростом объема и функциональной сложности проектов постоянно возрастает время, требуемое для верификации. В отдельных случаях оно занимает до 90% и более от всего времени разработки проекта. Поэтому все чаще возникают проекты, в которых "стандартный" маршрут не работает. Например, функциональная сложность проекта может возрасти настолько, что команда инженеров по верификации окажется не в состоянии предусмотреть все возможные потенциальные проблемы и их проверифицировать, 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 привел Intel к банкротству.

Для разрешения этой новой проблемы потребовалось усовершенствовать "классический" маршрут проектирования, используя новые подходы и концепции. В частности, к ним относится методология Assertion Based Verification (ABV) [2] и библиотеки OVL и QVL, которые мы и рассмотрим подробнее.

ФОРМАЛЬНАЯ МОДЕЛЬ. АССЕРТЫ

Для верификации сложных цифровых проектов уже недостаточно просто создать HDL-описание поведения системы и приступить к написанию тестов для верификации, так как с большой вероятностью можно пропустить внутреннюю ошибку, которая либо не будет воспроизведена тестами, либо не приведет к появлению неправильных данных на наблюдаемых сигналах. Для вылавливания таких трудно диагностируемых ошибок ведущими разработчиками оборудования, включая компанию 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 эти языка реализуют схожую идеологию ассертов.

Ассерт (Assertion) дословно можно перевести как утверждение. Ассерт – это некоторое описание правила работы сигналов. Основная идея методологии верификации на основе ассертов (Assertion Based Verification) заключается в том, чтобы информацию, т.е. знания о работе проекта, которые раньше оставались у разработчика в голове, записать формальными методами и использовать при моделировании в автоматическом режиме.

Для примера рассмотрим буфер FIFO. Текст его тестового окружения 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 состояний 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 не описывается в проекте, а требуемые ограничения (запись при переполнении или чтение из пустого буфера) менеджер по верификации должен проверять либо вручную, путем наблюдения за диаграммами сигналов, либо писать дополнительный блок, который будет автоматически отслеживать соблюдение условия. Применяя методологию Assertion Based Verification, разработчик проекта рядом с вхождением буфера формулирует дополнительные знания, которые раньше оставались не задокументированными, а именно, вставляет блок ассертов (выделенная часть текста), 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 двух важных блоков.

В целом, ассерты – это достаточно мощный и гибкий инструмент верификации, в задачи которого входит:

- отслеживание выполнения утверждений;
- поиск заданных последовательностей;
- генерация тестовых последовательностей;
- сбор статистической информации, например о покрытии.

БИБЛИОТЕКИ АССЕРТОВ

Ассерты удобно использовать для отслеживания правильности работы проекта в так называемых "горячих точках". "Отслеживание" может заключаться не только в проверке правильности работы конкретных сигналов и переменных, но и в накоплении дополнительной статистической информации, например о полноте покрытия. Отчет может быть представлен в виде одной или нескольких таблиц, в частности – 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 потенциальной возможностью допустить ошибку при составлении ассертов. Чтобы их преодолеть, были разработаны библиотеки ассертов, которые позволяют снизить требования к знанию языка ассертов (SystemVerilog Asserions или PSL), т.е. разработчику достаточно изучить только основные принципы работы с ассертами, не вникая в детали. Существуют различные варианты реализации этих библиотек, причем как свободно распространяемых, например OVL, так и коммерческих, таких как QVL и CheckerWare.

В общем случае, библиотечный элемент можно представить в виде блока (рис.1). Как правило, библиотечный элемент содержит следующие элементы:

Data in – входные порты. Среди них data inputs – входные порты для подключения к сигналам и переменным, которые необходимо проверифицировать, constants – порты для задания константных значений (параметров блока), active – активация блока, clock – синхросигнал, reset/areset – сброс.

Data out – выходные порты. Они включают выходы firing signals – индикаторы, которые "загораются", т.е. принимают значение "true" при нарушении проверяемых блоком условий; corner cases – счетчики, показывающие, сколько раз блок зафиксировал заданные пограничные условия; statistics variables – счетчики, показывающие различную статистическую информацию.

БИБЛИОТЕКИ OVL И QVL

OVL (Accellera's Open Verification Library) и QVL (Questa Verification Checkers Library) – это набор библиотечных ас-

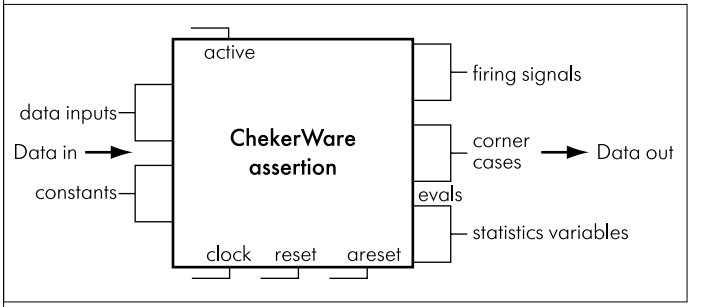

Рис.1. Условное представление библиотечного элемента

сертов, которые проверяют заданные параметры проекта. Данные библиотечные элементы подключаются к проекту посредством единого унифицированного интерфейса. Они содержат как элементы для тестирования простых структур, таких как счетчики, буферы, константы и т.д. (далее – чекеры), так и элементы для тестирования сложных протоколов, таких как протоколы шин AMBA, PCI, USB интерфейсов DDR и др. (далее – мониторы).

Чекеры являются компонентами модулей, цель которых – гарантировать, что определенные условия всегда выполняются. Чекеры состоят из одного или нескольких элементов четырех типов – property, message, severity и cover points [3].

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, времнной property – в разных тактах. Причем вполне возможно, что два отслеживаемых события будет разделять бесконечно большое число тактов.

Message – это строка, которая будет выводиться в окно протокола работы системы, если чекер зафиксирует нарушение.

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

Cover points – это флаги, счетчики 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 чекера. Например, число тактов, в которых выполнялось измерение. Флаги крайних случаев отслеживают разного рода пограничные состояния. Это могут быть редкие или необычные состояния, "сложно достижимые" состояния или различные нетипичные состояния контролируемого модуля. "Правильный" тест должен покрыть все эти состояния, так как 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_preload)
      );
endmodule
```

Для проверки правильности работы этого буфера, помимо вставки в проект самого элемента, необходимо вставить экземпляр чекера 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 corresponding enqueued value. : Test

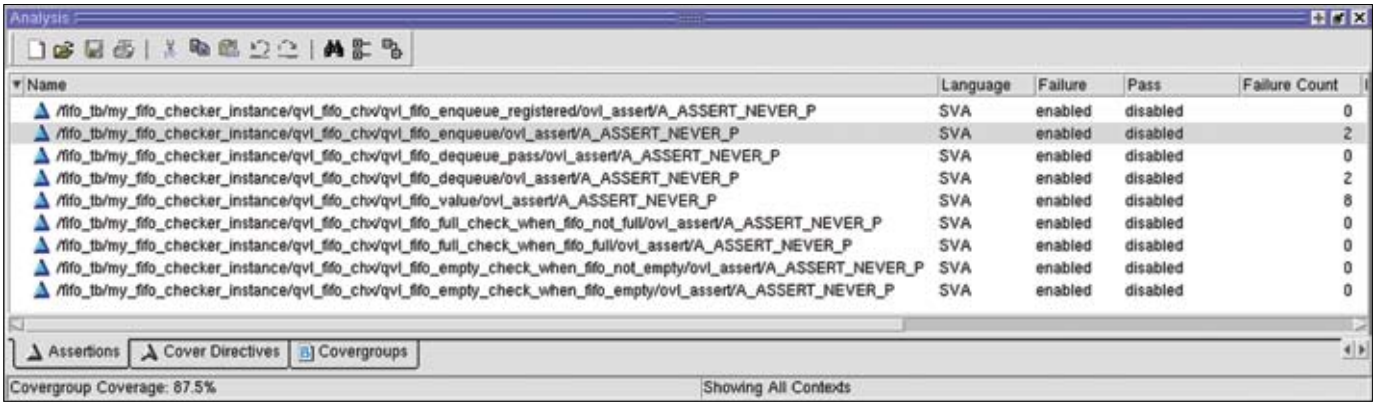

Рис.2. Ассерты, порожденные чекером

expression is not FALSE : severity 1 : time 2450 ns : fifo_tb.my_fifo_checker_instance.qvl_fifo_chx.qvl_fifo_value.ovl_error_t

Как видно из приведенного выше примера вывода диагностики, чекеры не только проверяют правильность входных и выходных сигналов, но и контролируют сложное функциональное поведение элемента, например эквивалентность записанных и считанных данных. Принципиальное отличие библиотеки QVL от OVL заключается в наличии в QVL так называемых мониторов, т.е. элементов для диагностики таких сложных устройств, как шины PCI или AMBA, интерфейс USB и т.д. [4].

Для удобного анализа всех статистических данных можно воспользоваться средствами анализа ассертов, сводящими все ассерты в таблицы. Например, на рис.2 показана таблица, из которой видно, что события qvl_fifo_enqueue (попытка записи в полный буфер) и qvl_fifo_dequeue (попытка чтения из пустого буфера) возникали по 2 раза, а событие qvl_fifo_value (прочитанное значение из буфера не соответствует ожидаемому) возникало 8 раз.

Механизм ассертов позволяет описывать и выполнять достаточно сложную аналитику в автоматическом режиме. Например, чекер FIFO при помощи ассертов формирует аналитическую структуру, показанную на рис.3. При этом разработчик тестов может оценить, насколько полон созданный им тест. Так, на рис.3 видно, что не был оттестирован один "крайний" случай, а именно – Simulatneous_Enqueue_and_Dequeue, т.е. одновременное чтение и запись данных.

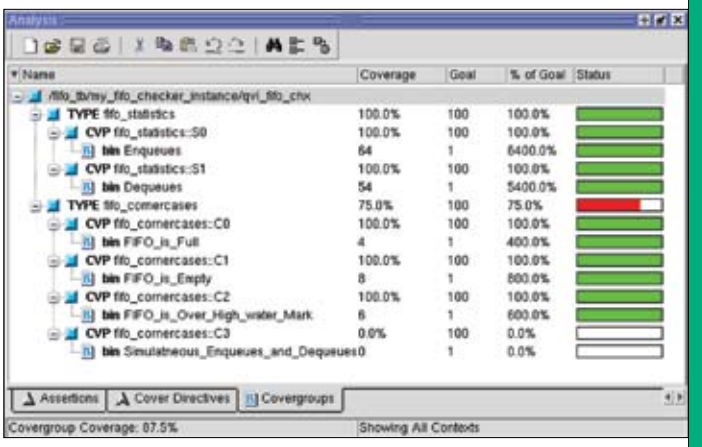

Рис.3. Кавергруппы, порожденные чекером FIFO

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

ЛИТЕРАТУРА

1. **Лохов А., Рабоволюк А.** Комплексная функциональная верификация СБИС. Система Questa компании Mentor Graphics. – ЭЛЕКТРОНИКА: НТБ, 2007, №3, с.102–109.
2. Questa SV/AFV User's Manual.
3. Questa Verification Library Checkers Data Book.
4. Questa Verification Library Monitors Data Book.