Часть 7. Несколько слов об отладке проекта

Что такое “testbench”?

Этим термином обычно обозначают код для проверки проекта пользователя при симуляции. Код создан для того, чтобы описать, какие действия будут осуществлены с проверяемым модулем, и увидеть его ответную реакцию.

Как было сказано в предыдущих разделах, процесс проектирования начинается с того, что уточняется задание на проект и вырабатываются проектные спецификации. Далее идет разработка файлов проекта. И одновременно с этим начинается верификация, то есть проверка проекта. Весь процесс верификации сводится к тому, что разработчик сравнивает то, что он должен получить от изделия, с тем, что он наблюдает в поведении проверяемого проекта (рис. 1). На основании сравнения разработчик определяет, соответствует ли ожидаемое поведение проекта полученному.

Всякий раз, когда описывается концепция тестирования или описывается методика работы тестбенча, необходимо ясно отличать опорный проект и тестируемый проект (DUT — design under test).

DUT представляет собой проект в некоторой законченной форме, подходящей для воплощения в продукции. Опорный проект — это идеальный вариант, «мечта» проектировщика, или то, что должен выполнять данный проект в соответствии с техническим заданием. Компонент для сравнения может быть выполнен во множестве форм, например, это может быть документ, описывающий работу DUT, некоторая опорная модель, которая содержит уникальный алгоритм, или это могут быть различные протоколы обмена данных.

Создание проекта — это отдельная проблема, и она должна быть хорошо понятна разработчикам. В этом разделе мы ограничимся обсуждением проблем по фиксации того, что данный проект создан в соответствии с выданными техническими требованиями.

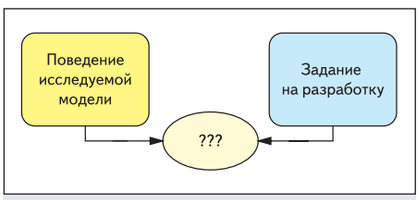

Рис. 1. Сравнение поведения проверяемого проекта с тем, что должно получиться по заданию на проектирование

Два основных вопроса

Завершением любого проекта можно считать тот момент, кода будут получены положительные ответы на два основных вопроса: «Это уже работает?» и «Мы это уже сделали?» Это довольно очевидные вопросы, но они — основа для любой методики по проверке проекта. Ответ на первый вопрос появляется в соответствии с теми идеями относительно проверки проекта, которые мы обсуждали в предыдущем разделе. Положительный ответ соответствует тому, что выполненный нами проект соответствует выданному заданию на проектируемое изделие. Положительный ответ на второй вопрос означает, что мы сравнили результаты проверки с заданием и пришли к выводу, что разработанный нами проект полностью соответствует заданию, или, в противном случае, почему он ему не соответствует. На рис. 2 показана блок-схема, отражающая обобщенную методику проверки проекта. На данной диаграмме кружочки с цифрами представляют собой основные пункты, по которым должно проходить тестирование проекта.

Каждый проект проверки начинается со спецификации проекта (п. 1 на диаграмме). Спецификация содержит все детали относительно того, как именно должен быть выполнен данный проект. В мировой практике принято, чтобы проект вели две независимых группы разработчиков: одна группа занимается собственно разработкой функциональных узлов проекта, а другая — проверкой проектов. Однако в небольших фирмах разработкой проектов и проверкой занимаются одни и те же люди, но, в ряде случаев, это может привести к росту числа ошибок.

Итак, группа проверки использует спецификацию проекта, чтобы построить план проверки. Это список всех вопросов, на которые нужно ответить в процессе проверки, и описание того, каким способом будут выдаваться ответы. Таким образом, на данном этапе создается контрольный список всех вопросов, на которые необходимо ответить, этот же список вопросов ложится в основу для спецификаций по тестбенчу (п. 2 на диа-

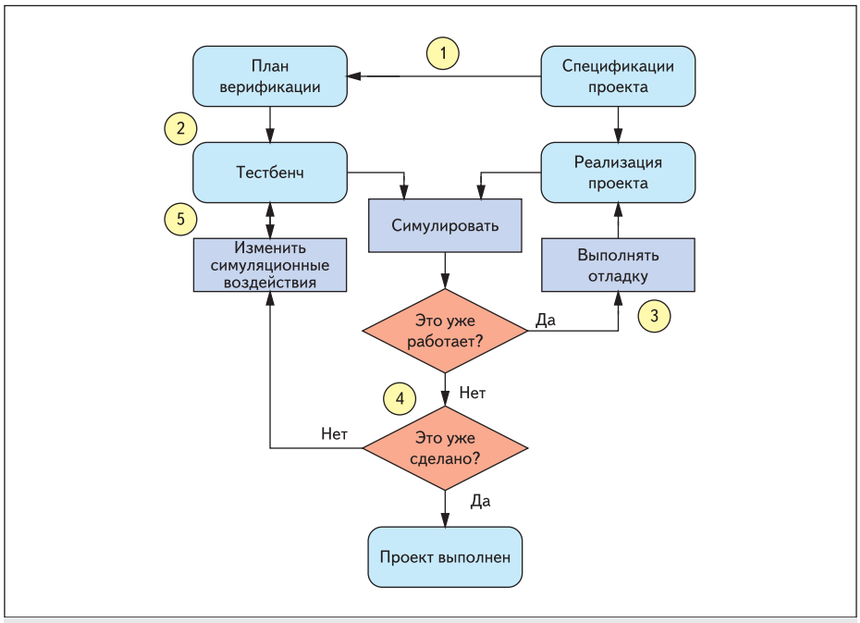

Рис. 2. Обобщенная блок-схема проведения верификации проекта

Таблица. Примеры вопросов из раздела Timing Specification по проверке соответствия стандарту PCI

№ вопроса	Описание вопроса	Да	Нет
CE45	Все трехстабильные сигналы переходят в третье состояние не позже чем за 28 нс для шины, работающей на частоте 33 Мгц, и не позднее, чем за 14 нс для шины, работающей на частоте 66 Мгц?	+	
CE46	Все входные сигналы на шинах требуют время предустановки не менее чем 7 нс для шины, работающей на частоте 33 Мгц, и не менее чем 3 нс для шины, работающей на частоте 66 Мгц?	+	
CE47	Сигнал REQ# требует время предустановки не менее чем 12 нс для шины, работающей на частоте 33 Мгц, и не менее чем 5 нс для шины, работающей на частоте 66 Мгц?	+	

грамме). В качестве примера можно привести часть стандартного списка вопросов для проверки соответствия разрабатываемого устройства на стандарт шины PCI. Данный список вопросов разработан комитетом по стандарту PCI. В таблице приведены только 3 вопроса из 52. Полный список вопросов можно посмотреть в документах комитета, а также, например, в [2].

После того, как тестбенч будет создан, начинается этап моделирования проекта совместно с тестбенчем. Результаты моделирования и дадут нам ту информацию, на основании которой мы можем ответить на два заданных вопроса. Первый вопрос — «Это уже работает?» Если ответ «Нет», то проект должен быть заново отлажен (что соответствует п. 3 на диаграмме). Отрицательный ответ на вопрос также приводит к тому, что проект должен быть доработан, для того чтобы исправить все выявленные ошибки, недостатки или неточности, которые удалось найти при тестировании.

Когда нам кажется, что проект функционирует должным образом, значит, тогда пришло время задавать следующий вопрос: «Мы это уже сделали?» На диаграмме этому состоянию соответствует п. 4. Сначала необходимо определить, какая часть проекта была охвачена тестированием. Если охват тестированием недостаточен, то объем работ по симуляции и тестбенч должны быть увеличены. Этому состоянию соответствует п. 5.

Конечно, при идеальных условиях проект не будет иметь каких-либо ошибок, и, таким образом, охват тестированием в этом случае всегда будет достаточен. Необходимо пройти только один цикл проверки, чтобы сразу получить положительные ответы на оба вопроса. В реальности же может потребоваться довольно много итераций. Правильная методология проверки проекта как раз и позволяет значительно уменьшить число итераций, чтобы закончить выполнение проверки проекта в кратчайшее время и использовать наименьшее количество ресурсов.

Чтобы читателю были более понятны эти, кажущиеся очевидными, рассуждения, давайте рассмотрим небольшой пример из практики.

Итак, несколько слов о том, как надо и как не надо отлаживать проекты

Представим себе «классический пример». Допустим, у нас есть проект пользователя, представляющий собой UART. UART имеет следующие характеристики: данные на прием и на передачу 8 разрядов, бит проверки на четность и стоповый бит. Частота приема и передачи — фиксированная. Тактовая частота системная, сброс — системный асинхронный. Пример этот довольно часто встречается при разработке проектов в FPGA. Вроде бы ничего сложного, обычный проект, достойный студента 3-го или 4-го курса... А теперь давайте составим план проверки проекта.

Первый вариант — облегченный

Этим вариантом заканчивают свою работу студенты и так же поступают оптимисты, которые свято верят в то, что проект обязан заработать сам по себе. Причем они с большим удивлением отвечают на вопрос: «А как именно вы проводили проверку проекта?» Они всегда очень уверенно добавляют: «При симуляции у меня проект идеально работал, без сбоев!»

Итак, вот их методика проверки.

Шаг № 1. Делаем файл testbench с UART'ом. Создаем тактовую частоту из системной, путем деления системной частоты на счетчике. Делаем заглушку «передача на прием».

Шаг № 2. Передаем байт данных, принимаем этот же байт, визуально сравниваем переданный и принятый байты. Если они совпали, то на этом проверка завершается.

А как это положено делать?

Для этого надо тщательно проследить, как именно работает наше устройство, и выбрать «крайние» режимы работы.

Шаг № 1. Для начала введем в testbench две системных тактовых частоты: одну для приема, а другую для передачи. Соответственно сделаем и две бодовых частоты: одну для передатчика, другую для приемника. Теперь сформируем массив данных для передачи в виде файла. В testbench сделаем выдачу данных в передатчик из файла, а в приемнике сделаем запись данных в файл принимаемых данных. Дополнительно можно и в testbench сформировать функцию проверки переданных и принятых данных.

Шаг № 2. Делаем системные тактовые частоты равными. Проводим проверку тракта передача/прием. Результат сравниваем в файлах. Если совпали, переходим к следующему шагу.

Шаги № 3 и 4. Моделируем разбег частот генераторов системной тактовой частоты. Генератор «перекашиваем» в разные стороны. Уход частоты генератора — статический, временной и температурный. Здесь также необходимо учесть то, что делитель бодовой частоты тоже может давать неточное приближение к стандартной бодовой частоте.

Шаги № 5 и 6. В этом разделе мы будем проверять, формирует ли приемник бит ошибки по паритету. Выполняем проверку, аналогичную пунктам 3 и 4, но на стороне передачи делаем ошибку по паритету. Эту ошибку приемник должен показать в регистре состояний.

Шаги № 7 и 8. Теперь проверим тем же образом стоповый бит. На стороне передачи создаем неправильную посылку по стоповому биту. Эту ошибку приемник тоже должен показать в регистре состояний.

Шаг № 9. Мы проверили работу передатчика и приемника при идеальной линии связи. Теперь необходимо проверить работу приемника при сбойных импульсах в линии. Формируем «короткий» стартовый импульс в линии. Выберем одно из «крайних» значений тактового генератора и проводим тест. Приемник не должен принимать данные, но в регистре состояний может появиться соответствующий бит.

Шаг № 10. Осталось самое легкое: введем сбойные биты «внутрь» битового интервала данных на передаче. Если в приемнике сделан мажоритар, то он должен восстановить правильное значение данных.

Итак, 10 шагов, вместо 2 шагов в предыдущем случае.

Какой из этого можно сделать вывод? Еще раз обратимся к классику (см. [3] во 2-й части данной статьи). Вот что пишет Жаник Бергерон: «До 70% трудозатрат в проекте приходится на верификацию проекта».

Несколько слов об «отладке проекта» до «отладки проекта»

Прежде чем переходить к описаниям тестбенчей и к примерам их реализации, необходимо сказать несколько слов о том, что под термином «отладка проекта» необходимо понимать не только обычную отладку при помощи программ симуляторов аппаратных средств, но можно и должно понимать отладку проекта при помощи обычных программных средств, таких как Си или MathCAD.

Вот пример из реальной практики. Автору необходимо было отладить многоканальный HDLC-контроллер. Проект представлял собой специализированный вычислитель, который на каждом из 32 тайм-слотов перезагружался, производил свои вычисления для одного байта данных из текущего тайм-слота и результаты вычислений сбрасывал в память. Для того чтобы отладить такой проект, необходимо было либо выполнить его по частям, то есть сделать только одноканальный контроллер, пропустить через него достаточно большой пакет данных и убедиться, что проект работает верно. А затем дополнить данный проект до полного 32-канального варианта. И тогда представим себе диаграмму: 32 канала — тайм-слота, во время каждого тайм-слота вычислитель делает 10–20 дейст-

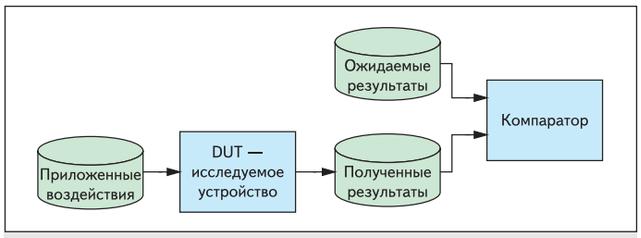

Рис. 3. Обобщенная модель тестбенча

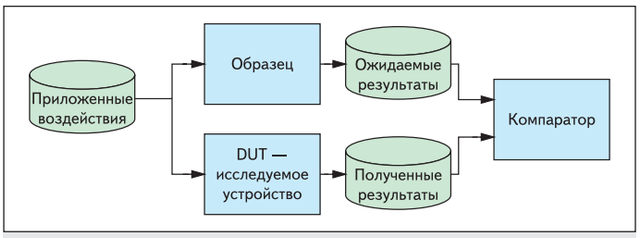

Рис. 4. Основной вариант тестбенча с опорной моделью

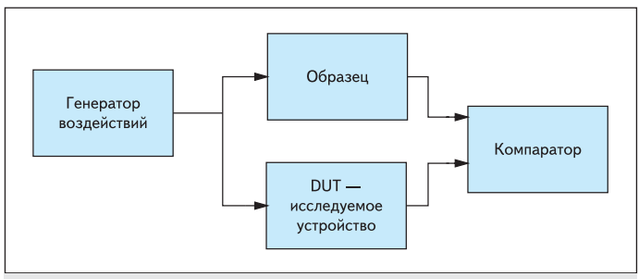

Рис 5. Автоматизированный тестбенч

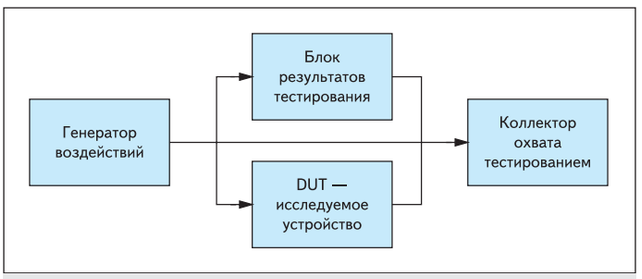

Рис. 6. Обобщенный вариант тестбенча

вий по одному такту. Всего получим до 600 тактов. Теперь представим, что мы хотим промоделировать прием последовательности из 20 байт. Следовательно, получим до 1200 тактов. Что тут можно сказать? Процесс долгий, кропотливый и неблагодарный. Автор в этом случае избрал совершенно другой путь. К разработке был привлечен программист: ему впоследствии надо было писать программы для прибора, в который входил и этот контроллер. Следовательно, для проверки прибора все равно необходимо было сделать конвертер данных, который бы из входной HDLC-последовательности мог бы извлекать кадры данных. Такая программная модель контроллера и была сделана. Именно на ней были опробованы все алгоритмы работы контроллера. Причем при создании этой модели разработчики старались написать код программы на Си так, чтобы он был более всего похож на «железо». Потом осталось только заменить переменные языка Си на регистры и триггеры в FPGA.

И еще одно замечание о пользе программных инструментов. Применение программных инструментов позволяет значительно сократить число ошибок при разработке. В качестве примеров автор может привести свои статьи [2] (см. также [1] — в 6-й части данной статьи), в которых описаны программные инструменты, примененные им в своих разработках. И конечно, не надо забывать, что существует достаточно много документов, в которых описано применение различных программ, позволяющих ускорить проектирование (см. [3] — в 6-й части данной статьи). В любом случае полезно проверить наличие такой документации на сайтах фирм-производителей и фирм, разработчиков программных инструментов, даже если вы не планируете работать с продукцией данной фирмы.

Тестбенч (испытательный стенд)

В этом разделе мы уже ввели термин тестбенч, который здесь можно рассматривать как испытательный стенд. В наиболее общей канонической форме тестбенч сравнивает результаты, полученные при симуляции работы DUT, с ожидаемыми результатами (рис. 3). Благодаря такому тестбенчу можно дать ответ на вопрос: «Это уже работает?» Если фактические результаты соответствуют ожидаемым результатам, то проверяемое устройство действительно работает, по крайней мере, с той точки зрения, что полученный набор ожидаемых результатов соответствует требуемым.

Базовый тестбенч имеет: файл стимуляционных воздействий на DUT, средство для того, чтобы применить стимуляционные воздействия к проекту, и средства для того, чтобы собрать полученные от DUT результаты. После проведения тестирования и получения образцовых результатов их можно будет сравнить с ожидаемыми результатами, полученными от работы проверяемого проекта. Поскольку простой проект с небольшим количеством входов имеет обычно и немного состояний, то набор ожидаемых результатов может быть просто построен вручную. Для любого проекта даже умеренной сложности подобная задача может быть достаточно трудоемкой. В таком случае более продуктивный способ состоит в том, чтобы создать образцовую модель, которая будет генерировать ожидаемые результаты. На рис. 4 показана структура тестбенча, где ожидаемые результаты и результаты, полученные от DUT, будут сгенерированы из того же самого набора стимулирующих воздействий. Если DUT будет работать правильно, то компаратор покажет, что работа, выполненная образцовой моделью и DUT, идентична.

Точно так же как и в варианте с образцовой моделью, есть еще одна возможность не выполнять непосредственно набор стимулирующих воздействий как некоторую последовательность векторов. Обычно гораздо проще написать программу, которая и будет генерировать стимулирующие воздействия. Поэтому, возможно, не будет необходимости хранить в файле набор стимулирующих воздействий или результаты работы теста. Вместо этого генератор стимулирующих воздействий сможет генерировать набор таких воздействий непосредственно в темпе выполнения тестирования, и компаратор может сравнить результаты работы по мере того, как они вырабатываются. Это обеспечивает более автоматизированный процесс, где процесс выработки набора стимулирующих воздействий и процесс сравнения результатов будет выполнен как часть тестбенча (рис. 5).

Теперь давайте поднимемся еще на одну ступень выше. Если мы заменим блок под названием «компаратор» на другой блок, который можно образно назвать так: блок, «отвечающий за то, что проверка выполнена», то в этом случае мы сможем обрабатывать более сложные наборы данных. И этот новый блок будет отвечать на ряд вопросов, среди которых наиболее важным будет следующий: «Действительно ли функционирование DUT соответствует функционированию опорной модели?» Таким образом, новый блок позволит более эффективно производить проверку проекта по сравнению с предыдущим случаем, когда мы лишь сравнивали фактически полученные результаты с ожидаемыми. И такой блок мы будем называть «блоком результатов тестирования» (scoreboard).

При проверке блок результатов тестирования получает информацию от референсных компонентов, которые выдают образцовые

значения данных, сгенерированных в течение моделирования при проверке DUT. Вся информация, которая при этом собирается, должна помочь выработать решение о проверке. Другой вопрос, на который также необходимо получить ответ: «Мы это уже сделали?» Для того чтобы принять это решение, мы можем использовать информацию, которая будет собрана в блоке — коллекторе охвата (coverage collector). Он получает информацию от генераторов стимулирующих воздействий, из блока результатов тестирования, и от DUT — обо всем том, что произошло в течение моделирования. Вариант такого тестбенча с блоком результатов тестирования и коллектором охвата показан на рис. 6.

Выводы:

- Генераторы стимулирующих воздействий инициализируют тестирование проекта. Те же самые стимулирующие воздействия посылаются как проекту при тесте, так и блоку принятия решений и/или опорной модели.
- Блок результатов тестирования оценивает поведение DUT и отвечает на вопрос «Это уже работает?» Блок результатов тестирования может включать в себя опорную модель.
- Коллектор охвата собирает информацию о работе DUT во время моделирования. Информацию в коллекторе охвата можно использовать для того, чтобы ответить на вопрос: «Мы это уже сделали?»

О стиле написания тестбенча

Здесь необходимо привести один пример (см. [3] из 2-й части данной статьи). Представим, что нам надо написать небольшой тестбенч для проверки некоторого узла. Как это будет выглядеть? Есть две точки зрения на этот вопрос. Разделим инженеров на две группы в соответствии с их навыками в написании тестбенчей. Первая группа — это опытные инженеры-разработчики, вторая — инженеры-тестировщики.

Инженеры-разработчики легко справляются с заданиями, в которых нужно выполнять синтезируемые модели. Такие модели содержат набор VHDL- или Verilog-файлов, и в них соблюдается один или только несколько стилей RTL-программирования. Есть довольно много публикаций по стилям RTL-программирования с рекомендациями по оптимизации проекта по занимаемой площади, скорости или потребляемой мощности.

Здесь еще раз приведены основные правила кодирования, которые помогают избежать нежелательных аппаратных компонентов, таких как триггеры-защелки, лишние внутренние шины или трехстабильные буферы.

Вот эти правила RTL-кодирования:

- Чтобы избежать триггеров-защелок, необходимо устанавливать в начале блока все выходы комбинаторных блоков в состояние по умолчанию.
- Чтобы избежать внутренних шин, не выполняйте назначения регистрам (assign reg) из двух различных блоков always.
- Чтобы избежать трехстабильных буферов, не присваивайте значения Z (например, 1'bz).

Дополним эти правила:

- Все входы блока, состоящего только из комбинационной логики, должны быть перечислены в списке чувствительности.
- Синхрочастота и асинхронный сброс должны быть в списке чувствительности блока, содержащего последовательные компоненты — триггеры и регистры.
- Используйте неблокирующие назначения для переменных типа reg, которые в проекте будут установлены как триггеры или регистры.

Инженеры-аппаратчики мыслят категориями статических автоматов, мультиплексоров, декодеров, триггеров-защелок, тактовых частот и т. д. Но не используйте стиль RTL-кодирования, когда вы пишете тестбенчи. Для того чтобы сделать конечный проект, который будет загружаться в изделие, у HDL-языков мало «выразительных средств». Эти средства определяются применяемой технологией и аппаратной платформой. А в том случае, когда нужно написать тестбенч, никаких ограничений, связанных с аппаратной платформой, нет. Тогда можно пользоваться всей мощью HDL-языка. Если же вы, создавая тестбенч, все еще пользуетесь приемами и стилем RTL-кодирования, то работа по тестированию может стать трудоемкой и занудной.

Разницу двух подходов можно показать на примере описания для простейшего протокола установления соединения (handshake). На рис. 7 приведена диаграмма работы такого протокола. Он начинается с выставления сигнала запроса (REQ). В ответ на сигнал запроса должен быть выставлен сигнал подтверждения (ACK). После получения сигнала подтверждения снимается сигнал запроса. Далее ожидается снятие сигнала запроса и снимается сигнал подтверждения.

Инженеры-аппаратчики, которые ориентированы на RTL-кодирование, быстро изобразят статический автомат, описание которого на VHDL показано в примере 1. Такой довольно простой алгоритм требует написания 28 строк кода и описания двух процессов. И еще необходимо добавить два состояния в такой автомат.

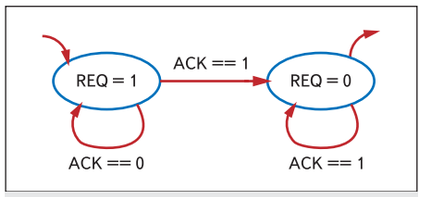

Рис. 7. Диаграмма работы протокола установления соединения (handshake)

```
Type STATE_TYP is (…, MAKE_REQ, RELEASE, …);
Signal STATE, NEXT_STATE: STATE_TYP;
…
COMB: process (STATE, ACK)
begin
   NEXT_STATE <= STATE;
   case STATE is
…
   when MAKE_REQ =>
      REQ <= '1';
      if ACK = '1' then
         NEXT_STATE <= RELEASE;
      end if;
   when RELEASE =>
      REQ <= '0';
      if ACK = '0' then
         NEXT_STATE <= …;
      end if;
   end case;
end process COMB;

SEQ: process (CLK)
begin
   if CLK'event and CLK = '1' then
      if RESET = '1' then
         STATE <= …;
      else
         STATE <= NEXT_STATE;
      …
      end if;
   end if;
end process SEQ;
```

Пример 1. Стиль RTL-кодирования

Инженеры-тестировщики, которые ориентированы на поведенческое описание тестбенча, сконцентрируют свои усилия не на аппаратной реализации в стиле RTL-кодирования, не на описании статического автомата, а на описании его поведения. Этот вариант тестбенча на языке VHDL показан в примере 2. Описание выполнено при использовании только 4 утверждений.

```
process
   begin
         …
      REQ <= '1';
         wait untill ACK = '1'
      REQ <= '0';
         wait untill ACK = '0'
   …
end process;
```

Пример 2. Стиль поведенческого кодирования

Сравнивая два приведенных примера, можно сделать следующий вывод: поведенческое описание тестбенча выполнить легче. Оно легче сопровождается и легче модернизируется. Мало того, такое описание симулируется гораздо быстрее.

В следующем разделе мы рассмотрим вопросы, связанные с моделированием проекта в ModelSim.

Литература

1. Xilinx PCI Data Book: www.xilinx.com/pci
2. Каршенбойм И. Г. Между ISE и ViewDraw // Компоненты и технологии. 2005. № 6. http://iosifk.narod.ru/ise_and_wd.pdf.
3. Glasser M., Rose A., Fitzpatrick T., Rich D., Foster H. Advanced Verification Methodology Cookbook. Version 2.0. Mentor Graphics Corporation. www.mentor.com
4. Writing Efficient Testbenches. Mujtaba Hamid. XAPP199 (v1.0) June 11, 2001.
5. Programmable Development and Test. WP276 (v1.0) December 17, 2007: www.xilinx.com