|  |  |
| --- | --- |
| ДИСЦИЛИНА | **Схемотехника устройств компьютерных систем Часть 2** |
|  |  |
| ИНСТИТУТ | **ИТ** |
| КАФЕДРА | **вычислительной техники** |
|  |  |
| ВИД УЧЕБНОГО | **Лекция** |
| МАТЕРИАЛА |  |
| ПРЕПОДАВАТЕЛЬ | **Тарасов И.Е.** |
|  |  |
| СЕМЕСТР | 3 |
|  |  |

2

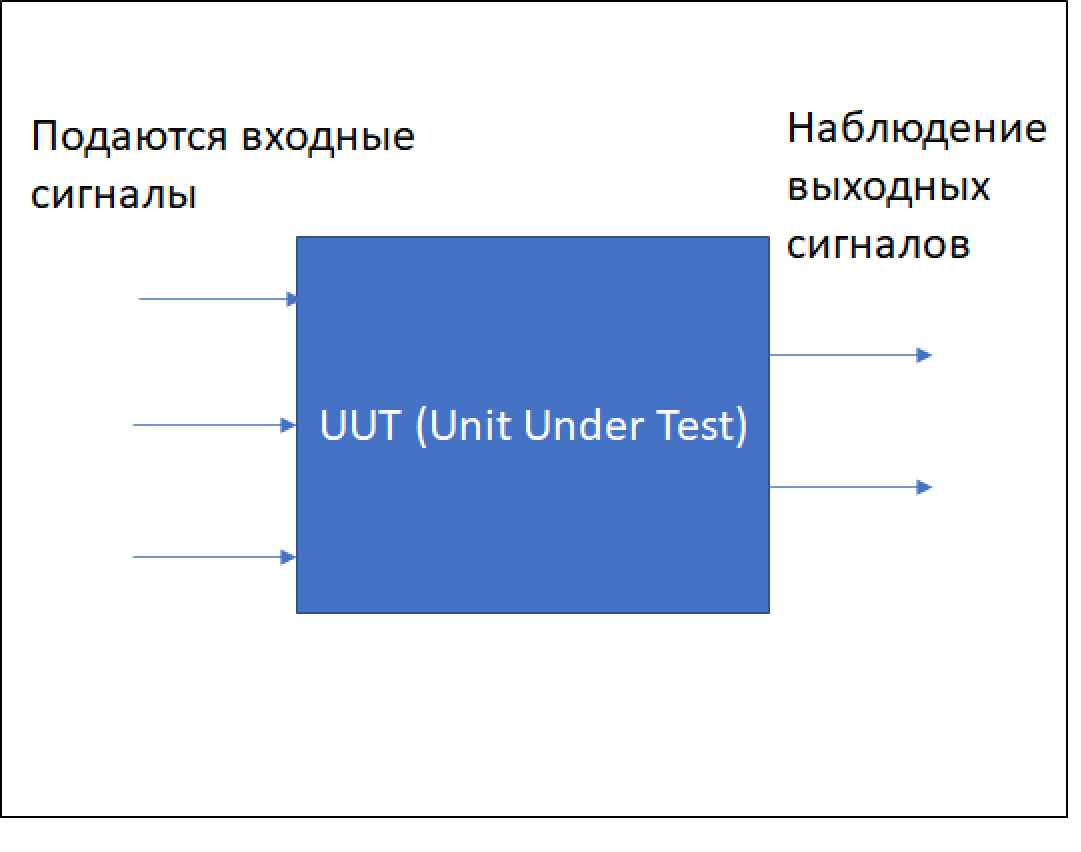
15. Стратегии моделирования и верификации компьютерных систем

15.1. Поведенческое и физическое моделирование, их отличия и место в маршруте проектирования.

Моделирование цифровых систем является важным шагом в маршруте их разработки. Возрастание сложности проектируемых устройств заставляет разработчиков тратить все больше времени на их моделирование. Целями моделирования могут быть как исследование алгоритмов работы проектируемого устройства, так и верификация характеристик, получаемых при его аппаратной реализации. В первом случае производится моделирование на верхних уровнях абстрагирования (т.е. преимущественно на поведенческом, и, возможно, RTL), а моделирование на физическом уровне призвано проверить возможность работы созданного устройства в заданных условиях эксплуатации (т.е. проверяется возможность работы на заданной тактовой частоте, с требуемыми длительностями сигналов, в заданном температурном диапазоне и т.д.).

При моделировании используется подход, основанный на «испытательном стенде» (*testbench*). Моделируемое устройство (в англоязычной литературе *UUT, Unit Under Test*) представляется своим синтезируемым кодом, а для проверки его поведения в различных условиях создаются описания тестовых воздействий («моделируемый код»).

На рис. 15.1 показана общая иллюстрация к организации процесса моделирования.



*Рис. 15.1. Иллюстрация к организации процесса моделирования*

Задание тестовых входных воздействий может быть выполнено на Verilog с помощью модуля типа *Verilog Test Fixture*. Для этого в диалоговом окне добавления компонента к проекту (Add Sources) выбирается пункт Add or Create Simulation Source. Необходимо выбрать тип файла Verilog и задать его имя.

Для унификации обозначений обычно применяется сочетание символов tb (от *testbench*, «испытательный стенд»). Так, если модуль имеет название my\_and (двухвходовый элемент И), то с ним удобно сопоставить тестовый файл my\_and\_tb.

Пример шаблона файла с описанием тестовых воздействий показан ниже.

`timescale 1ns / 1ps

module my\_and\_tb;

// Inputs

reg a;

reg b;

// Outputs

wire c;

// Instantiate the Unit Under Test (UUT)

my\_and uut (

.a(a),

.b(b),

.c(c)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

// Wait 100 ns for global reset

#100;

// Add stimulus here

end

endmodule

Это описание необходимо модифицировать для получения требуемого эффекта от моделирования. Предположим, что требуется проверить поведение модуля при последовательной установке в логическую единицу сначала входа a, а затем входа b. Тогда раздел «инициализация входов» (Initialize Inputs) будет выглядеть так.

initial begin

// Initialize Inputs

a = 0; # 10; a = 1;

b = 0; # 20; b = 1;

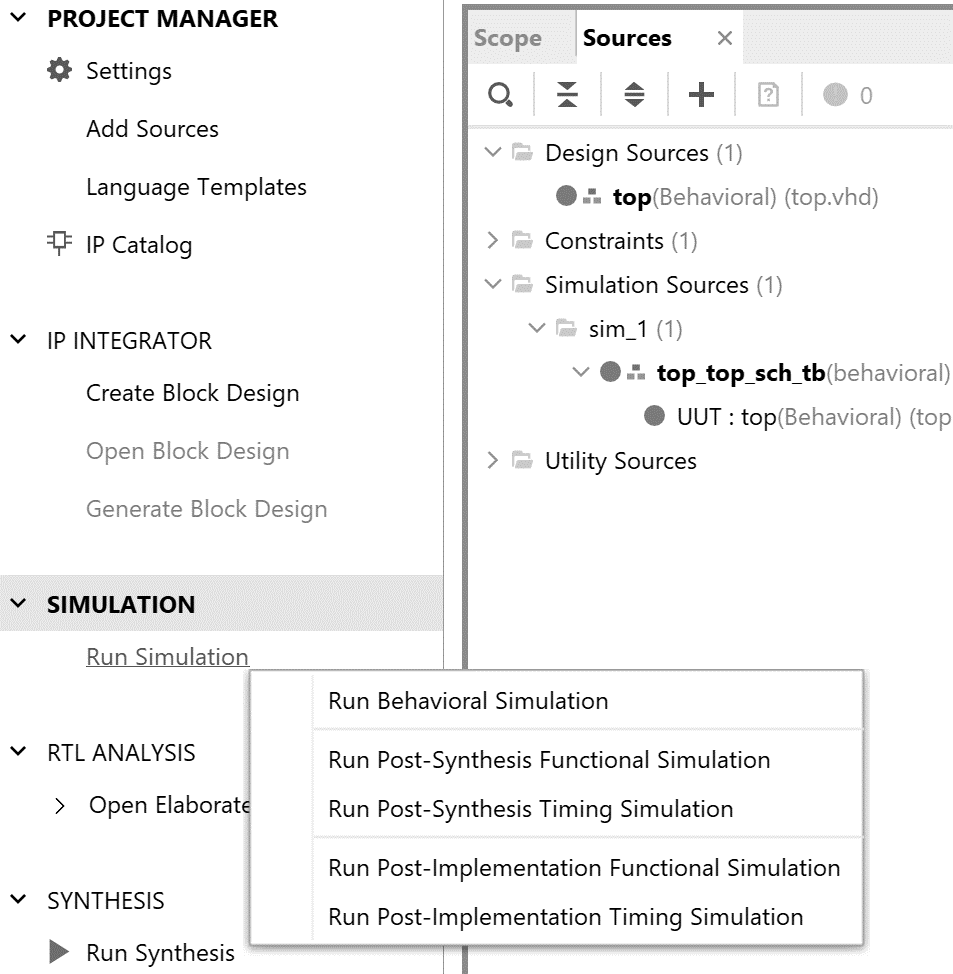
// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

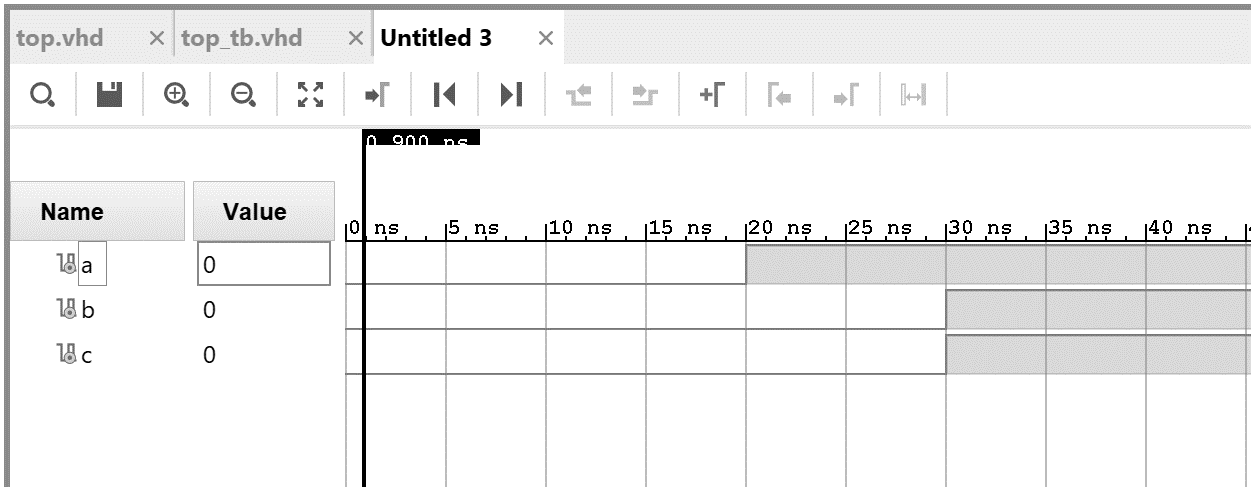
end

Запуск моделирования в Vivado производится в панели управления, как показано на рис. 15.2.



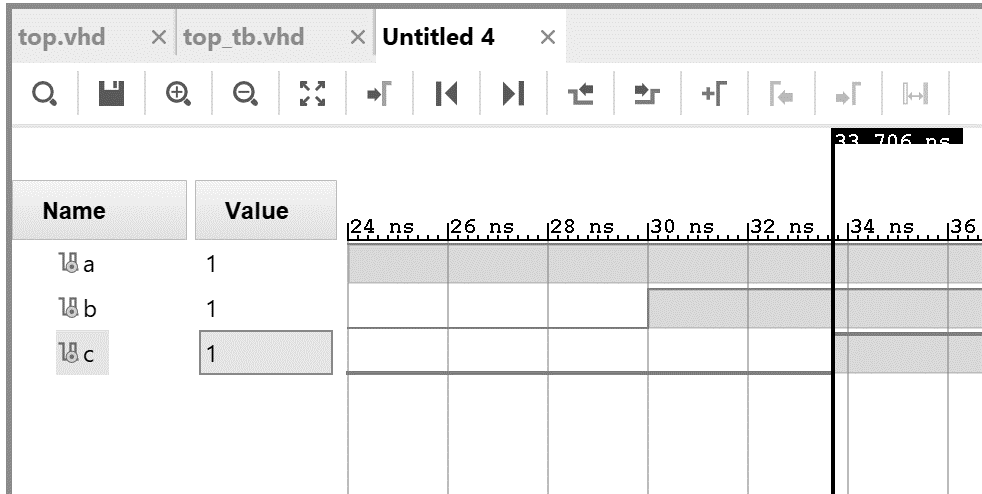
*Рис. 15.2. Запуск моделирования в САПР ПЛИС*

Запуск поведенческого моделирования (Behavioral Simulation) приведет к построению временных диаграмм работы модуля. На рис. 15.3 показано, что для заданных входных воздействий симулятор автоматически определил правильное состояние выхода c.



*Рис. 15.3. Пример временных диаграмм, создаваемых программой моделирования в САПР ПЛИС (поведенческое моделирование)*

В САПР ПЛИС возможно проведение моделирования на физическом уровне, когда задержки распространения сигналов не принимаются равными тем, которые указаны в поведенческом описании, а рассчитываются, исходя из физических моделей компонентов ПЛИС и трассировки конкретного проекта. Для моделирования в таком режиме необходимо выбрать пункт Run Post-Implementation Timing Simulation. Результаты моделирования показаны на рис. 15.4. С учетом задержки распространения сигнала логическая единица на выходе c появляется не в момент времени 30 нс, а в момент 33,.706 нс.



*Рис. 15.4. Пример временных диаграмм, создаваемых программой моделирования в САПР ПЛИС (физическое моделирование)*

Дополнительная величина 3,706 нс не является задержкой распространения через единственный логический вентиль, что было бы чрезмерно завышенным результатом для 28-нм элементной базы. Данная задержка определена с учетом влияния входных буферов для сигналов a и b и выходного буфера выхода c, подключенных к выводам ПЛИС.

Для получения результатов Post-Route моделирования необходимо, как следует из названия, выполнить трассировку (Routing) проекта. Таким образом, результат оказывается привязан к конкретной элементной базе. Кроме того, его получение связано с необходимостью анализа физических моделей компонентов, что требует существенно большего времени по сравнению с выполнением поведенческого моделирования. Однако такой результат существенно точнее, поскольку времена распространения сигналов рассчитываются по реальной трассировке, а не вводятся в модель из субъективных соображений.

При проектировании цифровых систем моделирование на физическом уровне обычно используется на завершающих этапах верификации проекта, когда требуется подтвердить не просто правильность выполнения преобразований, а соблюдение временных характеристик установления и распространения сигналов. В процессе отладки удобнее использовать моделирование на поведенческом уровне, поскольку оно производится существенно быстрее. В этом случае даже ориентировочные величины задержек позволяют иллюстрировать временной сдвиг сигналов друг относительно друга, что позволяет отлаживать архитектуру устройства.

15.2. Входные воздействия и наблюдение реакции системы.

Более сложные моделирующие конструкции можно создавать с помощью процедурных блоков initial и always. Процедурный блок initial выполняется однократно в процессе моделирования, в момент времени 0, а блок always – каждый раз, когда изменяется любой из сигналов, перечисленных в списке чувствительности.

Пример 1:

initial

begin

clk = 1b’0;

forever #10 clk = ~clk;

end

В этом примере с помощью ключевого слова forever организуется бесконечный цикл – через каждые 10 нс значение сигнала clk меняется на противоположное (подразумевается, что единицы времени для моделирования установлены равными 1 нс, как это обычно бывает).

Пример 2:

initial

begin

q = 1b’0;

end

always @ (a,b)

begin

q = a & b;

end

В примере выходному сигналу q присваивается значение, равное нулю. Далее следует процедурный блок always, который описывает логический вентиль И.

Основным преимуществом моделируемого подмножества Verilog является возможность создания моделей, описывающих задержки распространения сигналов. Это позволяет применять такие задержки к элементам, основываясь только на информации, указанной разработчиком модели, что существенно быстрее, чем получение этой информации путем анализа физической модели этого компонента. Таким образом, разработчик модели обязан обеспечить правильные величины задержек, но это компенсируется увеличением скорости моделирования.

Задержка указывается с помощью символа #. Например, для непрерывного присваивания.

assign #3 q = a & b;

#3 показывает, что задержка распространения сигнала составляет 3 нс (точнее, 3 «единицы времени», величина которых определяется директивой `timescale, и обычно равна 1 нс).

Для логических вентилей могут указываться три величины задержек, соответствующие следующим величинам:

– время перехода в высокий уровень (rise time);

– время перехода в низкий уровень (fall time);

– время отключения (turn off time).

Эти времена указываются после символа # в скобках, в порядке, приведенном в списке.

assign #(2,3,4) q = a & b;

Необходимо еще раз подчеркнуть, что приводимые таким образом задержки используются только при моделировании. Они игнорируются средствами синтеза, которые вместо этого могут рассчитать реальные задержки распространения, учитывающие используемые компоненты, соединяющие их проводники, условия работы и т.д.

Следующие конструкции не являются синтезируемыми и предназначены только для повышения удобства описания моделей.

Конструкция wait предназначена для синхронизации работы процедурных блоков. Например, в одном процедурном блоке может моделироваться установка сигнала, который используется в другом блоке.

wait (changed\_signal)

a = b;

Указанный блок (в примере состоящий из оператора a = b) будет выполняться каждый раз, когда изменится значение сигнала changed\_signal.

Конструкция while представляет собой цикл с проверкой условия. Она имеет следующий синтаксис:

while(<условие>)

<оператор>;

При необходимости исполнять в цикле несколько операторов, как обычно в подобных случаях, используются «операторные скобки» begin/end.

Команда forever представляет собой бесконечный цикл.

initial

begin

clk = 1’b0;

forever #10 clk = ~clk;

end

Команда repeat повторяет цикл заданное число раз.

initial

begin

counter = 0;

repeat (256)

begin

$display(“Counter = %d”, counter);

#10 counter = counter + 1;

end

end

15.3. Понятие стратегии моделирования.

В процессе проектирования системы разработчик может потратить практически неограниченное время на редактирование файла входных воздействий и запуск моделирования. Если набранные им тексты синтаксически корректны, при всех запусках будут строиться какие-то временные диаграммы, которые, однако, не обязательно означают, что устройство работает правильно.

Для того, чтобы моделирование не превратилось в бесполезный процесс, необходимо определить, с какой степенью детализации будут проводиться проверки отдельных систем, сколько будет таких проверок, какие специальные условия и сочетания сигналов необходимо проверить и что является признаком успешного окончания проверки. Все это образует понятие стратегии моделирования.

Важность выбора стратегии моделирования является следствием того, что выбрать единственную «оптимальную стратегию» на практике невозможно. Например, моделирование путем полного перебора входных значений выглядит привлекательным, поскольку на первый взгляд позволяет проверить все возможные варианты работы устройства. Однако даже для 32-разрядного сумматора такой перебор означает подачу 264 сочетаний входных значений, т.е. 1,8\*1019. Такой перебор займет сотни лет, хотя речь идет о проверке одного небольшого цифрового узла.

С другой стороны, проверка 2-3 очевидных случаев оставляет открытыми множество вопросов. Например, как поведет себя устройство при переполнении разрядной сетки результата? Если проверена передача байтов 0 и 255, состоящие из одинаковых битов, то в правильном ли порядке передаются отдельные биты? Как поведет себя системная шина, если адресуемое устройство не выдает сигнал подтверждения готовности? Все эти вопросы требуют специального внимания к организации проверок.

Одним из практических подходов является сочетание *направленного тестирования* (directed testing) и *псевдослучайного тестирования* (pseudo-random testing). Например, в случае сумматора могут быть проверены ситуации, для которых разработчик потенциально может ожидать некорректного поведения – например, сложение чисел, при котором выполняется перенос бита в старшие разряды. Эти проверки можно дополнить подачей приемлемо большого количества чисел, выбираемых генератором псевдослучайных чисел.

Формально, используемые в программировании генераторы являются псевдослучайными, поскольку используют алгоритмы генерации, основанные на получении следующего значения последовательности из предыдущего по детерминированной формуле. Случайными являются, например, аппаратные генераторы на основе измерения тепловых шумов полупроводникового диода, которые редко используются в технике. В то же время, использование функции random с учетом неконтролируемого начального значения последовательности для практики можно отнести к генераторам случайных чисел.

Тем не менее, при создании тестовых воздействий рекомендуется использовать именно контролируемые последовательности. Это необходимо для того, чтобы после обнаружения ошибки ее можно было воспроизвести.

Таким образом, в качестве отправной точки для организации моделирования можно выбирать сочетание проверок сценариев с очевидно понятными результатами и подачу достаточно большой последовательности псевдослучайных значений, которые могут способствовать выявлению неявных проблем в проекте.

С практической точки зрения полезно организовывать автоматизированное тестирование. На рис. 15.5 показан простой пример проверки периферийного устройства.



*Рис. 15.5. Организация автоматизированного тестирования*

Тестируемое устройство, показанное как UUT, подключено к генератору тестовых последовательностей. Генератор формирует, например, сигнал rx интерфейса UART. При этом ожидаемый эталонный отклик (т.е. передаваемы байт) подается внутри модели через блок имитации задержки на устройство сравнения. Имитация задержки требуется для того, чтобы воспроизвести работу контроллера UART, который записывает принятый байт через определенное время после начала передачи. Сравнение, реализованное в модели, определяет, совпадает ли выход UART с эталонным значения передаваемого байта.

В такой модели легко реализовать проверки по разным сценариям. Например, описывая правила генерирования разных значений, можно реализовать как направленное тестирование, так и псевдослучайное. Конечным результатом такой проверки можно считать отчет «проведено N тестов, закончились успешно N, ошибок 0». При необходимости можно подробно изучить правила генерации входных значений, использованные алгоритмы и правила проверки.

При разработке цифровых систем применяются следующие виды тестирования.

**Моделирование** представляет собой запуск программы-симулятора, демонстрирующей реакцию разрабатываемого устройства на входные воздействия в виде диаграмм сигналов и текстовых файлов. Моделирование интенсивно используется на ранних стадиях разработки, когда требуется уточнить поведение системы, проверить корректность разработанных описаний и убедиться, что реакция на входные воздействия соответствует техническому заданию, а система в целом действительно решает поставленную задачу.

**Стендовые испытания** (испытания в лабораторных условиях) подразумевают проверку работы устройства в контролируемой среде. При этом используются лабораторные генераторы, контрольно-измерительное оборудование, а специальные воздействия ограничиваются. Поскольку испытания проводятся на реальном экземпляре разрабатываемого изделия, становится возможным проверить адекватность функциональных моделей и убедиться, что использованные схемотехнические решения действительно работоспособны на реальной микросхеме. На этом этапе могут быть выявлены отклонения от синхронного стиля проектирования (симулятор не может автоматически проверить уход фазы тактового сигнала, влияние джиттера, возникновение метастабильных состояний и прочие отклонения реального кристалла от идеализированной математической модели), проблемы в организации питания, отвода тепла от микросхемы, влияние эффектов дискретизации по уровню и времени при обработке аналоговых сигналов.

**Интеграционные тесты** проводятся для исследования влияния системных эффектов. Они проводятся при сопряжении прибора с реальными устройствами, причем на этом этапе может быть обнаружено, что синтетические тесты неадекватно отразили реальную ситуацию. Например, блок питания системы может иметь повышенный уровень шумов по сравнению с лабораторным источником питания, который использовался при отладке. Дополнительные шумы могут также генерироваться источниками, которые не были учтены при лабораторных испытаниях. Реальная коммуникационная среда может иметь другой уровень нагрузки (количество пакетов, их состав, уровень потерь и т.п.). Это может привести к тому, что результаты лабораторного моделирования будут признаны неадекватно отражающими реальную картину.

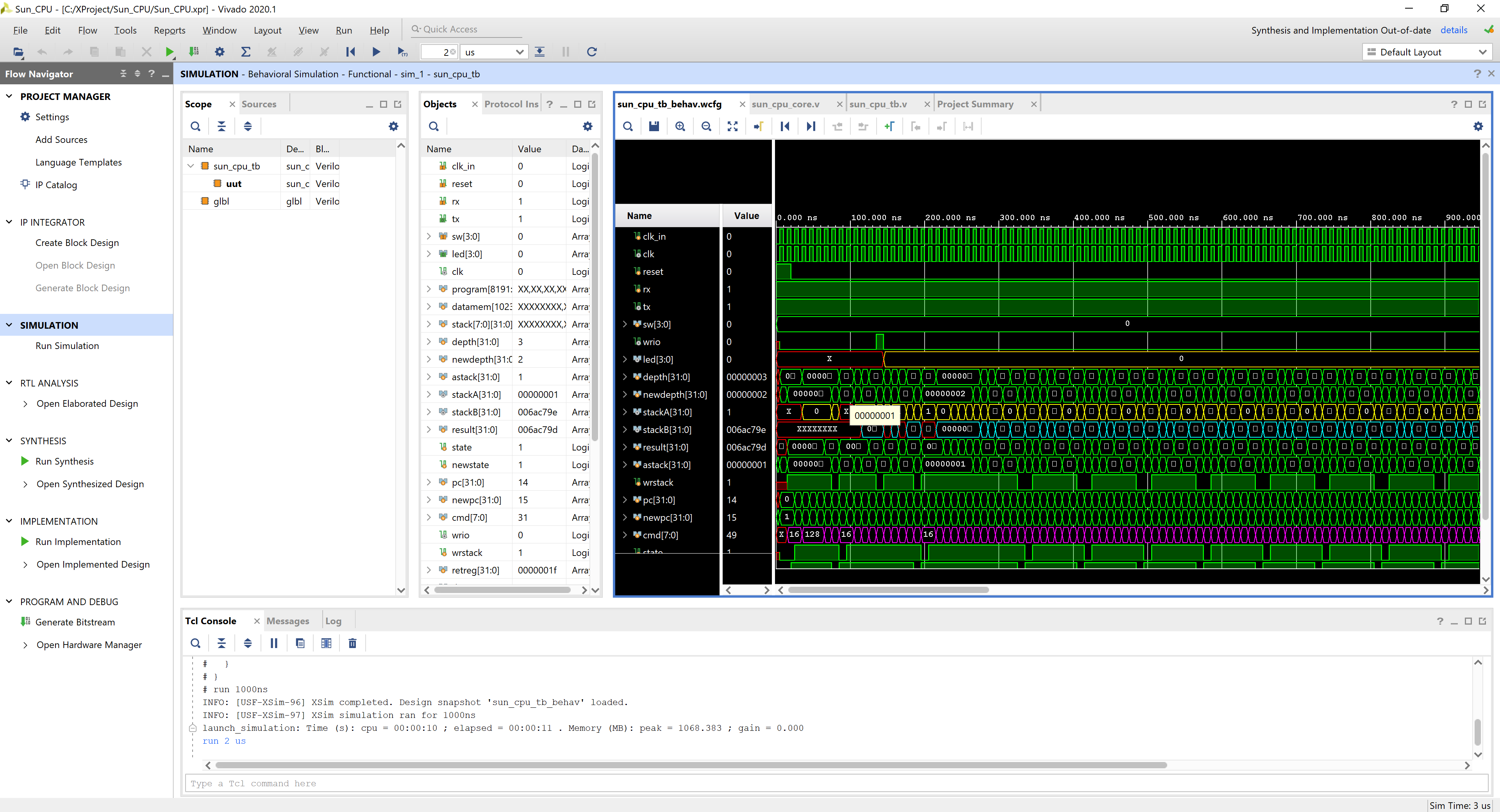
**Испытания в полевых условиях** подразумевает проведение тестовой эксплуатации устройства в реальных условиях. На этом этапе могут быть выявлены такие эффекты, как неработоспособность при реальных сочетаниях температуры эксплуатации, уровня помех, механических и электромагнитных воздействий, низкая эргономичность или ремонтопригодность.

В целом можно отметить, что использование программ-симуляторов и даже стендовых испытаний не является абсолютной гарантией работоспособности изделия в реальных условиях эксплуатации. Поэтому необходимо планировать дополнительное время на проведение испытаний в реальных условиях с последующей коррекцией схемы или даже конструкции изделия в целом.

15.4. Системное моделирование.

Под системным моделированием понимается тестирование поведения функционально завершенной системы, в отличие от проверки отдельных узлов и сценариев их взаимодействия. Например, для системы цифровой обработки сигналов такой проверкой может быть фильтрация зашумленного сигнала с правильным измерением параметров полезной информационной составляющей. Для процессорного устройства это может быть моделирование выполнения прикладной программы с получением итогового результата.

Системное моделирование процессора может быть основано на загрузке в память процессора кодов программы и запуске модели с поданным тактовым сигналом. При этом записанная в памяти программа будет самостоятельно инициировать выполнение определенных действий. При необходимости следует сформировать входные воздействия на периферийные устройства процессора.



*Рис. 15.6. Пример временных диаграмм при системном моделировании процессора*

Системная модель генерирует существенно больше данных для проверки по сравнению с тестированием отдельных узлов процессора. Возможно выявление скрытых дефектов проекта, связанных с неверной архитектурой или ошибками в проектировании логики управления.

15.5. Выводы по разделу.

Моделирование позволяет разработать и отладить цифровую систему существенно быстрее и с меньшими затратами по сравнению с экспериментальными проверками изготовленного образца устройства. Основой моделирования является описание входных воздействий на цифровую схему с наблюдением результатов в программе-симуляторе.

Важным вопросом является выбор стратегии моделирования – правил задания входных воздействий, количества проверок и правил сопоставления выходов с ожидаемыми значениями. Важность выбора стратегии моделирования обусловлена тем, что полный перебор сочетаний входных сигналов занимает неприемлемо большое время.

Моделирование необходимо сочетать с лабораторными испытаниями, поскольку в модели могут оказаться неучтенными существенные эффекты – например, шумы, импульсные помехи, отклонения частоты тактового генератора и т.п., которые являются предметом экспериментальной проверки.

Процессы моделирования следует автоматизировать, чтобы избегать непроизводительных ручных запусков САПР с наблюдением отдельных сценариев. Например, для процессора эффективным приемом является заполнение памяти процессора программой и запуск моделирования, при котором команды программы будет самостоятельно инициировать выполнение тех или иных операций.

Контрольные вопросы:

1. Как организовать моделирование цифрового устройства?

2. Можно ли использовать в описании входных воздействий операторы, управляющие задержкой сигнала? Как описать задержку на 10 нс?

3. Как систематизировать процесс моделирования, организовав автоматическую проверку и выявление ошибок при проектировании?

4. Что произойдет, если память процессора заполнить программой и запустить процесс моделирования?