Что такое ВЕРИФИКАЦИЯ

Представьте, что вам поручили строить дом для кого-то. С чего начать? Вы начинаете, выбирая двери и окна, выбирая краски и ковер цветов, или выбор сантехники? Конечно, нет!

Во=Первых

Вы должны рассмотреть, как владельцы будут использовать пространство, их бюджет, так что вы должны решить, какой тип дома построить. Перед тем как начать, изучать подробности SystemVerilog вы должны понять, как вы планируете верифицировать устройство и, как это влияет на структуры testbench.

Так же, как во всех домах есть кухня, спальни, и ванные комнаты, все тесты имеют некоторые общие структуры стимулов генерации и проверки реагирования

Важнейший принцип. который вы должны усвоить, как инженер -тестировщик: "Ошибки хороши." Не уклоняться от поиска следующей ошибки, и не звонить в колокола каждый раз, когда вы обнаружили новую, но всегда отслеживать каждую найденную ошибку. Вы должны внимательно насколько это возможно, осмотреть проект с разных сторон, чтобы извлечь все возможные ошибки сейчас, пока они еще легко исправимы .

Характерные черты SystemVerilog как Языка Проверка оборудования (HVL) которые отличают его от Языков Описание оборудования, таких как Verilog или VHDL являются

-случайные воздействия с ограничениями
- Функциональные покрытия
-СТРУКТУРЫ высшего уровня , особенно объектно-ориентированного программирование. динамические структуры данных
многопоточность и взаимодействия процессов
Поддержка HDL типов, таких как 4-значные состояния в Verilog
-- Тесная интеграция с событие-симулятор
Есть множество других полезных функций, но это позволит вам создать тест на более высоком уровне абстракции, чем вы сможете достичь с помощью HDL или языка программирования, таких как C.

www/accelera.org /System verilog 3.1 Language reference manual.

Процесс верификации

Какова цель верификации? ответ: «Поиск ошибок", будет лишь отчасти верным.

Цель аппаратного обеспечения является создание устройства, которое выполняет конкретную задачу, такую как DVD-плеер, сетевой маршрутизатор или процессор радиолокационного сигнала, на основе проектной спецификации. цель инженера-

тестировщика, состоит в том чтобы убедиться, что устройство может выполнить эту задачу успешно - то есть, проект есть точное представление спецификации. Ошибка это то, что вы получаете, при обнаружении расхождения. **Процесс верификации параллелен процессу создания устройства**.

дизайнер читает аппаратные спецификации блока, интерпретирует человеческий язык описания, и создает **соответствующее логическое описание** в машиночитаемой форме, как правило, RTL кода. Чтобы сделать это, он должен понимать формат ввода, функцию преобразования и формат вывода.

Всегда возможна двусмысленность в этой интерпретации, в том числе из-за неясностей в оригинальном документе, недостающих деталей , или противоречивого описания.

инженер-тестировщик, также должен прочитать аппаратные спецификации, создать план верификации, а затем выполнить его, чтобы провести испытания, показывающие что RTL код **правильно реализует функции**. Назначив более чем одного человека выполнять ту же интерпретацию, вы добавили избыточность процесса проектирования.

Работа инженера-тестировщика также состоит в чтении аппаратных спецификаций, чтобы сделать независимую оценку того, что они означают. Ваши тесты, а затем эксперименты RTL должны показать, что это соответствует вашей интерпретации.

Какие типы ошибок скрываются в проекте? Самый простой для обнаружения из них находятся на уровне блоков, модулей, созданных одним человеком.

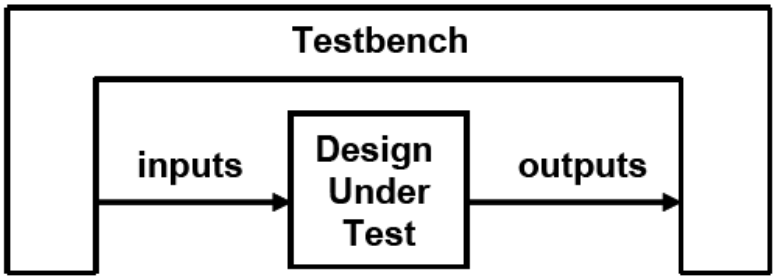
Правильно ли ALU складывают два числа? Разве каждый успешно ли шина завершает транзакцию ? Все ли пакеты прошли через часть сетевого коммутатора?

Почти тривиально писать направленные(=прямые) тесты, чтобы найти эти ошибки, если они содержатся полностью в **пределах одного блока** конструкции.

После уровня блоков, следующее место для поиска расхождений - **границы между** блоками. Интересные проблемы возникают, когда два или больше дизайнеров читая ТО же описание имеют различные толкования. Для данного протокола, какие сигналы изменяются и когда? Первый дизайнер строит драйвер шины и имеет один взгляд на спецификации, а второй строит приемник с немного иной точки зрения. Ваша **задача состоит в нахождении в спорных районах общей логики** и, возможно, даже помочь примирить эти две различные точки зрения.

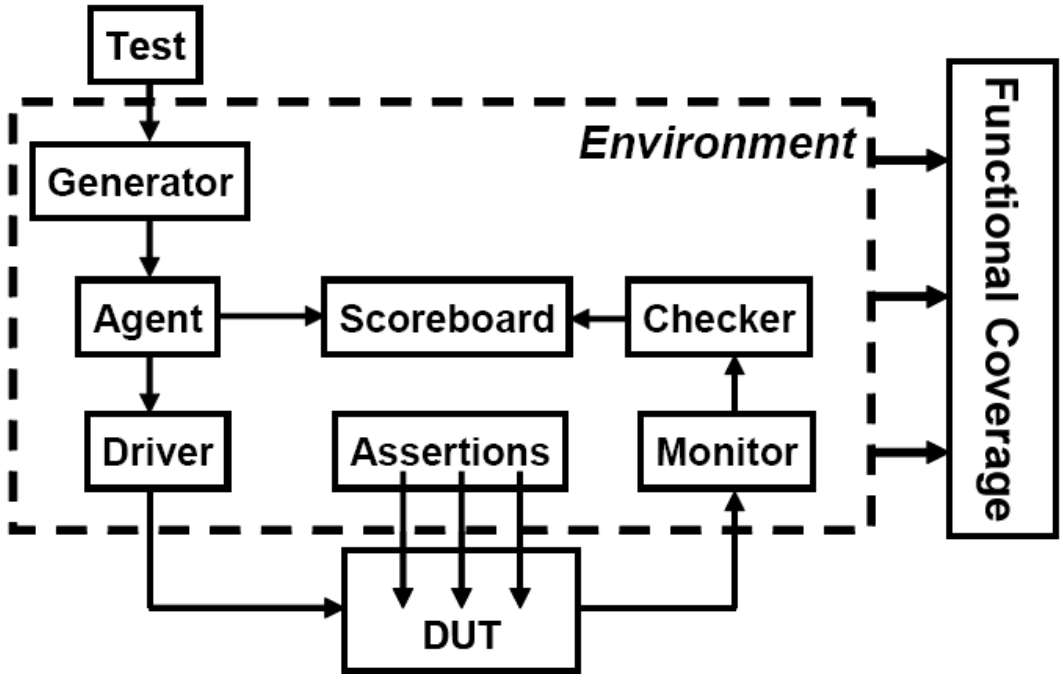
Для имитации одного блока, необходимо создать тесты, которые генерируют стимулы из всех блоков, что довольно муторно. Преимуществом является то, эти низкоуровневые модели исполняются очень быстро.

The testbench — design environment



К тому же вы можете обнаружить ошибки как в DUTтак и в testbench ПРИЧЕМ часто требуется БОЛЕЕ длинный код чтобы обеспечить стимуляции от отсутствующих блоков. Когда вы начинаете интегрировать блоки системы , они могут стимулировать друг друга, сокращая рабочую нагрузку. многоблочные модели позволяют раскрыть больше ошибок, но они и работают медленнее. На самом высоком уровне тестируемого устройства вся система тестируется, но производительность симуляции значительно снижается.

Ключевое понятие для любой современной методологии верификации это слоистые testbench. Хотя может показаться, что testbench в этой постановке более сложен, на самом деле это помогает реализовать Вашу задачу путем деления кода на более мелкие части, которые могут быть разработаны отдельно. Не пытайтесь писать одну процедуру, которая может случайным образом генерировать все виды стимулов, в том числе нелегальных, а также вводить ошибки многослойного протокола. Программа быстро становится сложной и не поддерживаемой.



Блоки testbench (внутри пунктирной линии) записываются в начале разработки. В ходе проекта они могут развиваться и вы можете добавить функциональности, но эти блоки не должны меняться для отдельных тестов.

Ваши тесты должны стремиться, чтобы все Блоки выполняли интересующие действия одновременно. Например Все порты ввода / вывода являются активными, Процессоры перемалывает данные а кэши пополняются. В совокупности таких действий, несомненно, происходят ошибки выравнивания данных и временные несогласования,.

После того как вы убедились, что DUT правильно, выполняет назначенные функции нужно посмотреть, **как оно работает, когда есть ошибки.** Может ли DUT поддерживать частичные транзакции, или транзакции с поврежденными данными или контрольным полем? Трудно просто перечислить все возможные проблемы, так же трудно определить как дизайнер должен справиться с ними .

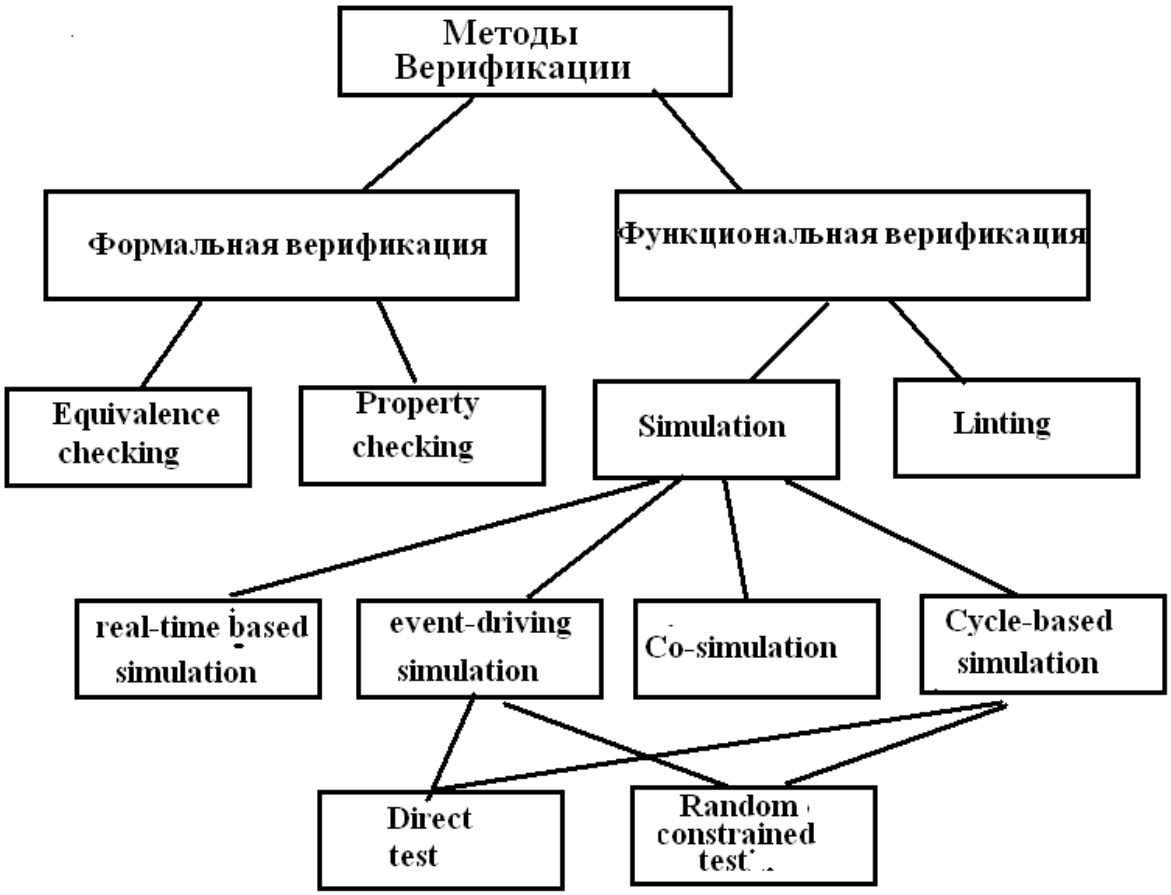
Ошибки вмешательств и обработки может быть самой сложной частью верификации .

По мере роста уровня абстракции абстракция, верификация бросает новые вызовы.

Вы можете показать, что отдельные ячейки в последовательности блоков маршрутизатора ATM правильны, но что если есть потоки различных приоритетов? Какая следующая ячейка должна быть выбрана не всегда очевидно на самом высоком уровне. Возможно, вам придется анализировать статистику из тысячи ячеек, чтобы убедиться, что совокупное поведение является правильным. И последнее замечание: вы никогда не сможете доказать, остались ли еще ошибки

И

Fron Janic Bergeron, Sinopsis Inc
"Writing testbenches using System Verilog

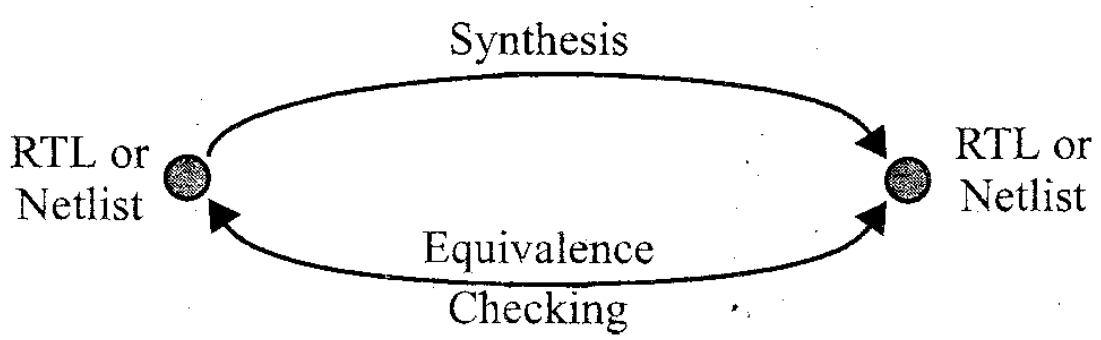


ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ математическое сравнение имплементации и спецификации или иных требований к устройству для определения не содержит имплементация нарушений спецификации

Примеры- некорректная таблица истинности, наличие недостижимых или тупиковых состояний автоматов, отсутствие некоторых соединений в синтезированной схеме

Применение формальной проверки подпадает под две широкие категориями : проверка эквивалентности и проверки property (свойства).

Проверка эквивалентности



В наиболее общем применении , проверка эквивалентности сравнивает два нетлиста чтобы некоторые элементы пост-обработки, такие как список соединений, цепочки сканирования, деревья тактовых сигналов или ручная модификация , не изменили функционирования схемы.

Другое популярное использование проверки эквивалентности – верифицировать , что список соединений правильно реализует оригинальный RTL код. Если полностью доверять инструменту синтеза, такая проверка не будет необходима . Тем не менее, инструменты синтеза это большие программные системы, которые зависят от правильности алгоритмов и библиотек . История показала, что такие системы подвержены ошибкам. В некоторых редких случаях, эта форма проверки эквивалентности используется для проверки вручную написанного RTL кода – правильно ли он представляет вентильный уровень

Проверки Эквивалентности может использоваться также для доказательства, что два RTL

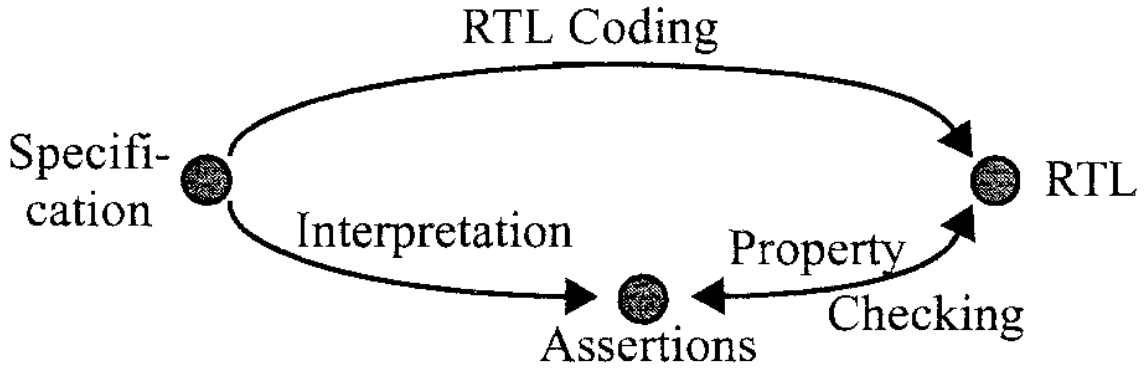
описания логически идентичны. Доказательство их эквивалентности позволяет избежать длительной работы моделирования регрессии, когда лишь незначительные нефункциональные изменений внесены в исходный код для получения лучших результатов синтеза..

Проверка properties (сущностных свойств)

Проверка сущностей это более современные путь технологии формальной верификации .В ней АССЕРЦИИ характеристик устройства формально доказываются или опровергаются. Например, все автоматы в конструкции могут быть проверены для недоступные или изолированные состояния . Более мощные средства Проверки сущностей могут , определять наличия тупиковых ситуаций при некоторых условиях

Другой тип Ассерций , что которые могут быть формально проверены относится к интерфейсам. В Спецификации понятия propertyes языка SystemVerilog указано что assertions об интерфейсах устройства фиксируются и интерпретатор пытается доказать или опровергнуть их. Например, assertion Может потребовать , что если сигнал ALE будет установлен , то либо сигнал DTACK либо ABORT в конце концов.

будут установлены ,



Reconvergence модель для контроля сущности показана на рисунке . Самым большим препятствием для технологии проверки свойств является определение спецификации доказываемой ассерции в интерпретации устройства . Только часть Ассерций , может быть доказана реально. Кроме того, для того чтобы были доказательства полезны, АССЕРЦИИ не должны быть тривиальными выдержками из поведения уже перехваченными RTL кодом. Они должны быть основаны на внешних требования, которым должен отвечать проект.

Функциональная Верификация

Функциональные Верификации проверяют проектной замысел.

Главной целью функциональной верификации является убеждение в том ,что устройство **реализует предназначенные функции**. Как показано на реконструктивной модель на рисунке , функциональной верификации связывает устройство с его спецификацией. Без функциональной верификации, пользователь вынужден верить, что преобразование специфицирующего документа в RTL код была выполнена без неправильного толкования назначений спецификации.

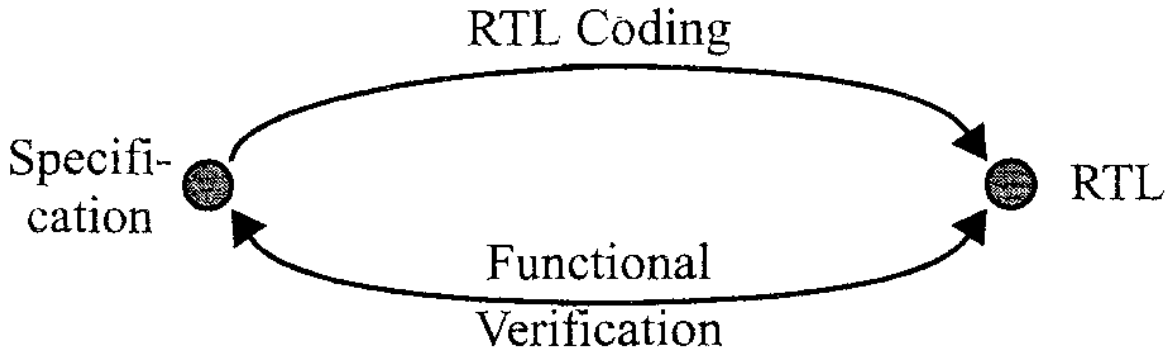


Рисунок 1-8.Функциональная проверка

Вы можете доказать наличие ошибок, но вы не можете доказать их лтсутствие

МОДЕЛИРОВАНИЕ

Моделирование является наиболее распространенным средством проверки технологии. Она называется "моделирование", поскольку она ограничена аппроксимацией реальности. Моделирование никогда не конечная цель проекта. Цель всех проектов, разработка аппаратной части, чтобы создать реальные физические проекты, которые Имитируют Ваш проекта до его реализации.

К тому же вы можете обнаружить ошибки как в DUTтак и в testbench ПРИЧЕМ часто требуется БОЛЕЕ длинный код чтобы обеспечить стимуляции от отсутствующих блоков. Когда вы начинаете интегрировать блоки системы , они могут стимулировать друг друга, сокращая рабочую нагрузку. многоблочные модели позволяют раскрыть больше ошибок, но они и работают медленнее. На самом высоком уровне тестируемого устройства вся система тестируется, но производительность симуляции значительно снижается.

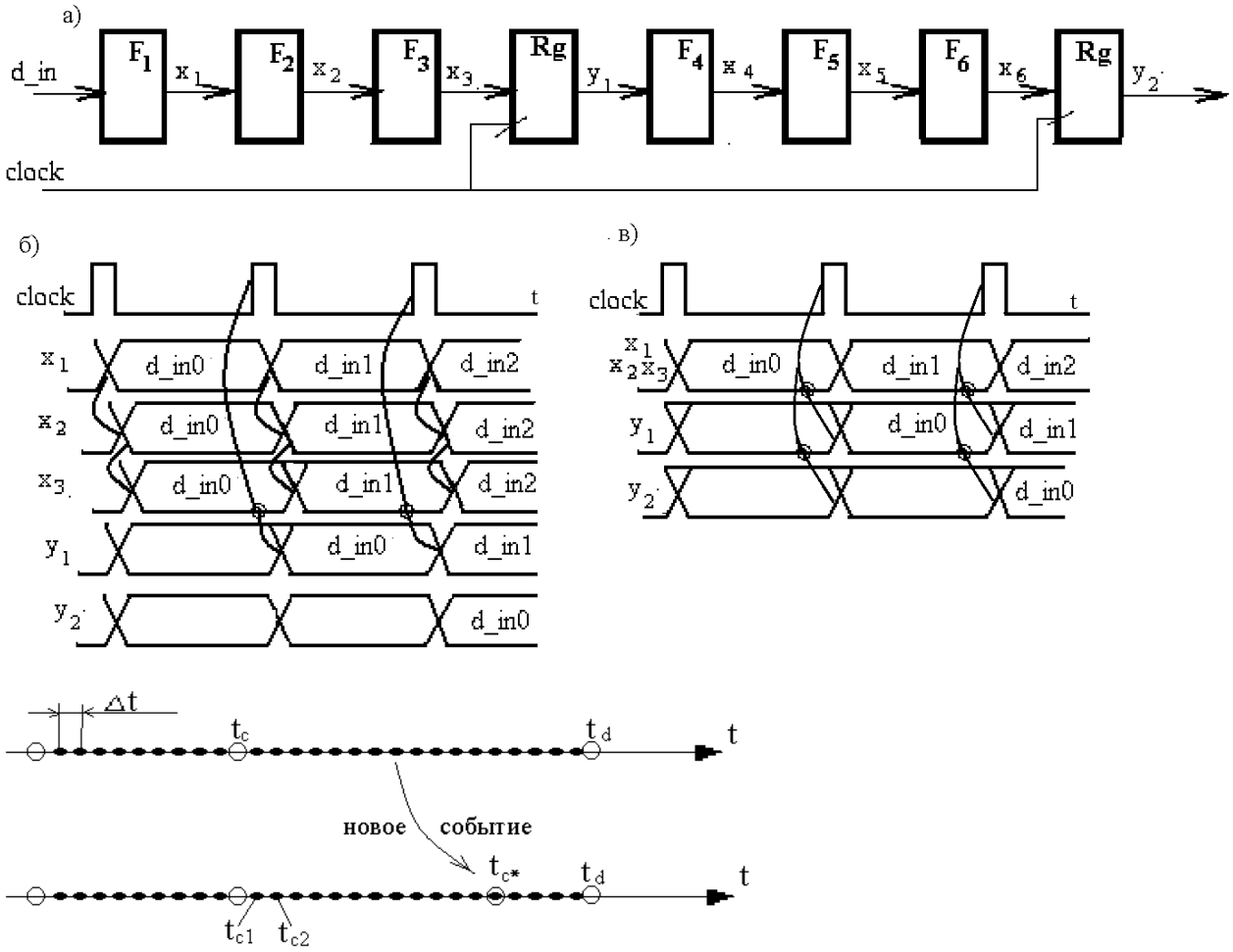
Написание тестов с использованием SystemVerilog дает возможность продавать их и получать прибыль. Эта технология проверки позволяет проектировщикам взаимодействовать с устройством , пока оно еще не производятся, и исправить недостатки и проблемы раньше.

Вы никогда не должны забывать, что моделирование является приближение к действительности

Цель testbench заключается в определении правильности организации тестируемого устройства DUT (ТсУ). Это достигается за счет следующих шагов.

- Создание стимулов
- Подать стимул на ТсУ
- Зафиксировать реакцию
- Проверить правильность

Различают Событийное моделирование, цикло-базированное М, со-моделирование



Понятия прямого тестирования и квазислучайного.