Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"Национальный Исследовательский Ядерный Университет "МИФИ""
(НИЯУ МИФИ)
Кафедра Электроники

Реферат на тему "Логическая модели уровня RTL. Методы тестирования модели."

Выполнил
студент С19-401
Высоцкий Матвей Дмитриевич

Принял
Доцент кафедры электроники
Петров Константин Александрович

Москва 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Введение

С момента своего появления и до нашего времени цифровая электроника проделала большой путь развития. Из схем с маленьким функционалом, большими и медленными транзисторами и ручным процессом разработки, она превратилась в сверхбольшие интегральные схемы (СБИС), с огромным набором функциональных подблоков, над которыми работают десятки и сотни людей.

Изменился и подход к разработке: первые цифровые схемы разрабатывались вручную, путем расчета уравнений булевой алгебры, создания типовых наборов элементов. Разработчик при таком подходе синтезировал схему на бумаге, что затрудняло процесс оптимизации, внесения правок, верификации.

Решением данных проблем стало внедрение и использование нового подхода в разработке, который называется Register Transfer Level(RTL). RTL - уровень регистровых передач, представляет собой подход к описанию электронной аппаратуры, как абстрактной модели, состоящей из последовательных комбинаций логических операций, результат которых передается между регистрами. Описание производится с помощью специальных языков описания аппаратуры Hardware Description Language (HDL), без указания электронно-компонентной базы (ЭКБ) или логических вентилей, из которых состоит схема.

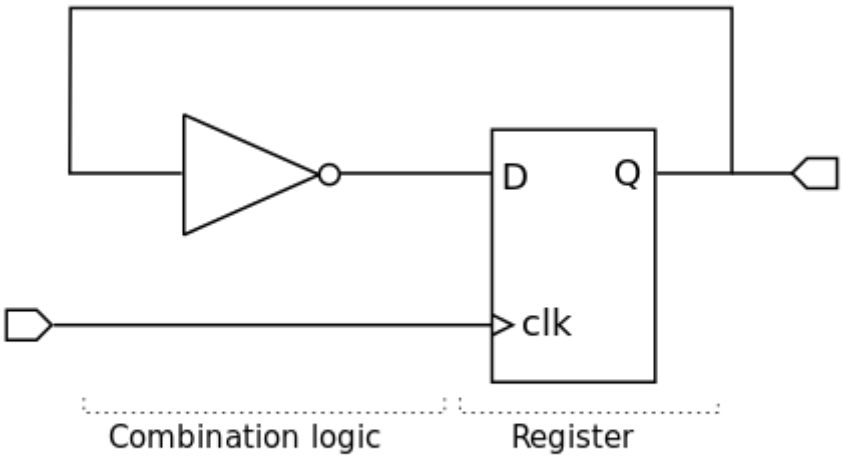

Рисунок 1 — RTL-модель

РОЛЬ RTL В ПРОЦЕССЕ РАЗРАБОТКИ

Процесс разработки цифровой электроники

Современные цифровые устройства проходят много стадий разработки, но глобально можно разделить процесс разработки на четыре больших этапа (Рассматривается разработка ASIC). Каждый этап разработки прямо или косвенно связан с RTL моделью разрабатываемого устройства.

Процесс разработки начинается с создания RTL модели устройства. На языке описания аппаратуры описываются составляющие цифрового устройства: память, регистры, вычислительные блоки, блоки управления. Описывается алгоритм работы каждого элемента внутри модели. Разработанные подблоки собираются в виде одного большого модуля, который является поведенческой моделью устройства.

После окончания разработки некоторых подблоков начинается процесс верификации. Инженеры-верификаторы разрабатывают тестовые модели, которые должны проверить работу модели устройства в целом или конкретных ее функций. В случае обнаружения ошибок RTL модель возвращается на доработку к инженеру-разработчику.

Прошедшая верификацию модель уходит на этап синтеза. С помощью специальных САПР (Например, Cadence Genus) RTL код синтезируется в дерево логических элементов по заданным правилам. Качество написанного кода и алгоритм работы определяют важнейшие характеристики, такие как энергопотребление, площадь и максимальная частота (При условии, что ЭКБ остается постоянной).

Синтезированную модель превращают в прототим реальной микросхемы на этапе разработки топологии. Инженер-тополог преобразует синтехированные логические вентили в их эквивалент из реальных транзисторов.

Процесс разработки аналоговой электроники

Аналоговая электроника не может быть абстрактно описана на языке описания аппаратуры и синтезирована в готовую схему, таким же образом как

цифровая. Но и здесь языки описания аппаратуры выполняют полезную функцию. Для цифро-аналоговых устройств аналоговая часть поведенчески описывается, чтобы в последствии можно было сделать верификацию модели. В случае чисто аналоговых схем возможно написание поведенческой модели и тестовых воздействий. Поведенческая модель может быть легко заменена SPICE-моделью(нетлистом) и тестироваться теми же тестами.

Роль RTL в процессе разработки

RTL модель - важная часть в разработке устройств, она является тем, на что опираются и с чем взаиможействуют на всех этапах разработки. В наше время разработка цифровой электроники почти не возможна без данного подхода, разработка СБИС фактически не возможна, поэтому разработка с использованием RTL применяется повсеместно, а с учетом тренда на замену аналоговых схем на цифровые (там где это возможно), применение данных практик будет все шире.

ИСТОРИЯ RTL ЯЗЫКОВ И СИТУАЦИЯ НА ДАННЫЙ МОМЕНТ

Появление RTL и языков описания аппаратуры

Первые вычислительные устройства стали появлять еще в годы нового времени. Это были специальные механические машины, которые выполняли арифметические операции и применялись в задачах, связанных с финансами или переписью населения. Со временем машины перестраивались под новые требования, получали новые реализации и в начале 20 века стали электрическими. Переход к электричеству позволил масштабировать микроархитектуру устройств, сделать ее более функциональной, но при росте функционала неизбежно растет количество элементов, которые используются при создании системы. Алгоритм работы требовал все больше и больше логических вентилей. Создание сложных систем вручную, используя логические уравнения требует много человеко-часов работы, а оптимизация таких решений представляется почти нереальной. Решением данной проблемы стал подход RTL и разработка HDL языков, которые создавали бы уровень абстракции между человеком и логикой микросхемы.

Языки описания аппаратуры

Идея создания языков описания аппаратуры появилась еще в 70-е годы, в результате чего в конце 70-х годов в Бредфордском университете был создан язык HILO, а в 1983 году появился язык ABEL. Позже в 1984 году появился язык Verilog HDL, который перенял многие концепции HILO. Также в 80-е годы появляется язык VHDL, который был основан на принципах языка программирования Ada. Несмотря на последующее развитие других языков описания аппаратуры, Verilog HDL и VHDL стали основными языками в разработке аппаратуры.

Verilog HDL

Язык Verilog HDL был разработан сотрудниками компании Automated Integrated Design Systems (с 1985 года Gateway Design Automation) Филом Мурби и Прабхом Гоелом. Компания Gateway Design Automation создала

первый симулятор данного языка под назанием Gateway’s Verilog. В 1990 году Gateway Design Automation была выкуплена компанией Cadence Design Systems. С момента разработки и до 1995 года язык являлся закрытым проприетарным инструментом.

Verilog-95

Во время увеличивающейся популярности языка VHDL, Cadence приняла решение добиться стандартизации языка. Cadence передала Verilog в общественное достояние. Verilog был послан в IEEE и принят как стандарт IEEE 1364—1995 (часто называемый Verilog-95).

Verilog-2001

В 2001 году произошло масштабное обновление стандарта языка. Синтаксис претерпел большое влияние со стороны языка C, благодаря чему язык стал доступнее для изучения (В последствие, C-подобный синтаксис стал одной из причин роста популярности Verilog и причиной победы в популярности над языком VHDL). В новом стандарте было добавлено много крайне полезных элементов, таких как арифметические операции +, -, /, *. До этого инженеры были вынуждены использовать большое количество побитовых операций. На данный момент стандарт 2001 является самым используемым диалектом языка и является обязательной часть почти всех коммерческих EDA для электроники.

Verilog-2005

Стандарт языка претерпел небольшие изменения в виде добавления нескольких новых конструкций. Основным изменением стало создание отдельного направления: языка Verilog-AMS. Благодаря новой части языка появилась возможность моделирования аналого-цифровых схем, где аналоговая часть написана на языке Verilog-A, а цифровая часть на Verilog HDL (Verilog-D).

SystemVerilog HDL

В 2009 году происходит слияние ранее разработанного Accelera языка SystemVerilog и стандарта Verilog-2005 в стандарт SystemVerilog-2009. Новый

стандарт привнес моного нового в язык: появилась возможность ООП тестиоования, разработка интерфейсов, большое количество новых конструкций, поведенческое описание аналоговых устройств. Глобально язык стал еще больше похож на язык C и перенял модель ООП языка C++.

Настоящее время

На данный момент Verilog является мощным языком описания аппаратуры, который применяется почти во всех крупных проектах по микроэлектронике. Язык поддерживается почти всеми коммерческими и некоммерческими САПР и инструментами, а также выпускаются новые стандарты языка. Verilog является наиболее часто используемым языком при разработке ASIC.

- Плюсы
  - удобство работы
  - прост в изучении
  - конструкционно-читаем (В описании схемы четко прослеживаются комбинационные и последовательностные элементы)
  - большое количество инструментов и обширное комъюнити инженеров-пользователей, помогающих с решением проблем
- Минусы
  - требует статического анализа кода из-за слабой системы типов
  - не все конструкции языка синтезируемы, что ограничивает разработчика в функционале языка
  - наибольшая оптимизация достигается уменьшением степеней свободы САПР
  - нет перехвата ошибок на этапе написания кода

VHDL

Язык VHDL был разработан в 1983 году по заказу министерства обороны США, в целях осуществления проектирования и верификации широкого

спектра логических систем, начиная с маленьких, заканчивая СБИС. Язык имеет большое количество стандартов: 1987, 1991, 1993, 1996, 1997, 1999, 2000, 2002 и 2008 гг. Каждый стандарт маленькими шагами расширял язык, таким образом, в 2000 году VHDL получил основы ООП. Также язык получил часть, описывающую смешанные сигналы VHDLAMS.

Особенности языка

При создании VHDL, язык испытал большое влияние со стороны языков Ada и Pascal, что стало причиной появления принципа наилучшей читаемости кода. Создатели стремились сделать язык самодокументируемым. Также язык является строго типизированным и радикально не допускает работы разных типов друг с другом.

Настоящее время

В настоящее время язык является обязательной частью всех коммерческих и некоммерческих САПР для электроники, а также получает новые стандарты. На языке VHDL созданы описания открытых микропроцессоров ERC32 (SPARC V7) и LEON (SPARC V8). VHDL является наиболее используемым языком при разработке FPGA устройств, например все IP блоки Xillinx для FPGA написаны на VHDL.

- Плюсы
    - Большое количество САПР и инструментов, поддерживающих язык
    - Большое количество библиотек разработки
    - Большое комъюнити разработчиков
    - Возможность перехвата ошибок на этапе написания кода
    - Четко выведенное синтезируемое подмножество
    - Самодокументируемость
- Минусы
    - Требует написания избыточного и нефункционального кода (например преобразование типов), что служит причиной большого

количества опечаток

- Тяжелый процесс изучения
- Тяжелый процесс разработки
- Крайне неудобный процесс верификации
- Нет инструемнтов автогенерации кода

Другие языки описания аппаратуры

Кроме популярных Verilog и VHDL существуют менее популярные языки, призванные решить некоторые проблемы основных языков. Хорошими примерами альтарнативы являются: SystemC, PyHDL(MyHDL), VeriL-lang и AHDL.

SystemC

SystemC - язык проектирования и верификации моделей системного уровня, разработанный в виде открытых библиотек на языке C. Язык позволяет использовать синтаксис C и компиляторы языков C и C для реализации модели устройства. Разработчики SystemC предполагали сделать удобный в изучении и доступный большому количеству разработчиков язык. На данный момент язык поддерживается большим количеством САПР для электроники, но не пользуется большой популярностью из-за сложного процесса отладки (В случае возникновения ошибки, компилятор выдает сообщения об ошибках в классах C++, что не позволяет понять местонахождение и тип ошибки).

MyHDL

MyHDL - библиотека языка python, которая позволяет имитировать описание аппаратуры с помощью функций и методов, подобных блокам Verilog HDL. Библиотека легко интегрируется с другими модулями python, что позволяет использовать тестирование с помощью pytest. MyHDL запускается с помощью интерпретатора python, но для использования в разработке необходима трансляция в Verilog или VHDL, что доступно в качестве функционала библиотеки. Библиотека является новым подходом к разработке, но не является популярным способом среди инженеров. К тому же

оригинальный код не позволяет полноценно контроллировать результат трансляции, что является одной из причин не использовать язык в пользу классических HDL языков.

Veril-lang

Veril - новый open-source проект для разработки описания аппратуры, написанный на языке Rust. Проект приобрел некоторую известность в конце 2023 года. Veril представляет собой не просто язык HDL, а целую среду разработки. Плюсами данного языка является удобная сборка проекта (Полноценная система на Verilog требует написания большого количества сторонних скриптов на TCL, языках командной оболочки, утилит make и cmake), Veril пользуется удобством сборки проектов языка Rust. Минусами языка является его новизна, как следствие отсутсвие большого количества примеров использования, возможные внутренние проблемы компилятора и необходимость трансляции в SystemVerilog.

AHDL

AHDL - проприетарный язык описания аппаратуры от компании Altera. Язык предназначен для программирования ПЛИС фирмы Altera. Язык не имеет большой популярности среди разработчиков.Недостаток AHDL в его проприетарности. Преимущество AHDL в том, что все конструкции языка синтезируемы. В сравнении с Verilog AHDL как язык ассемблера в сравнении с языком высокого уровня: в AHDL больше контроля, но меньше высокоуровневой поддержки. На данный момент язык поддерживается компиляторами только от компании Altera.

ВЕРИФИКАЦИЯ МОДЕЛИ

Процесс верификации модели является важной и неотъемлемой частью разработки любой современной СБИС. Финальной частью разработки микросхемы является изготовление на специальной фабрике (TSMC, например). В отличие от программного обеспечения, изготовленная микросхема не может быть изменена, что сильно увеличивает цену ошибки. На данный момент верификация делится на несколько типов и имеет свои методологии.

Верификация в процессе разработки

Функциональная верификация

Основная задача функциональной верификации - проверка функционала системы, проверка каждого состаяния и по возможности достижение функционального покрытия 100% (каждый бит должен пройти переходы из 0 в 1 и наоборот). Результатом функциональной верификации является заключение о работоспособности ситемы, в особенности возможность выдержать рандомизированные тесты.

Функциональная верификация имеет несколько методологий, каждая из которых позволяет использовать сильные стороны различных инструментов либо упрощает тестирование сложных объектов.

UVM

UVM(Universal Verification Methodology) - универальная методология верификации. Данная метология подразумевает создание универсального тестового окружения для конкретных тестовых задач(например, измерить частоту изменения сигнала в определенном узле). Созданное тестовое окружение может использоваться много раз, подключаясь к разным моделям в разных проектах. Большим преимуществом такого подхода является возможность быстрой верификации (при условии, что уже существуют все готовые тестовые блоки окружения).

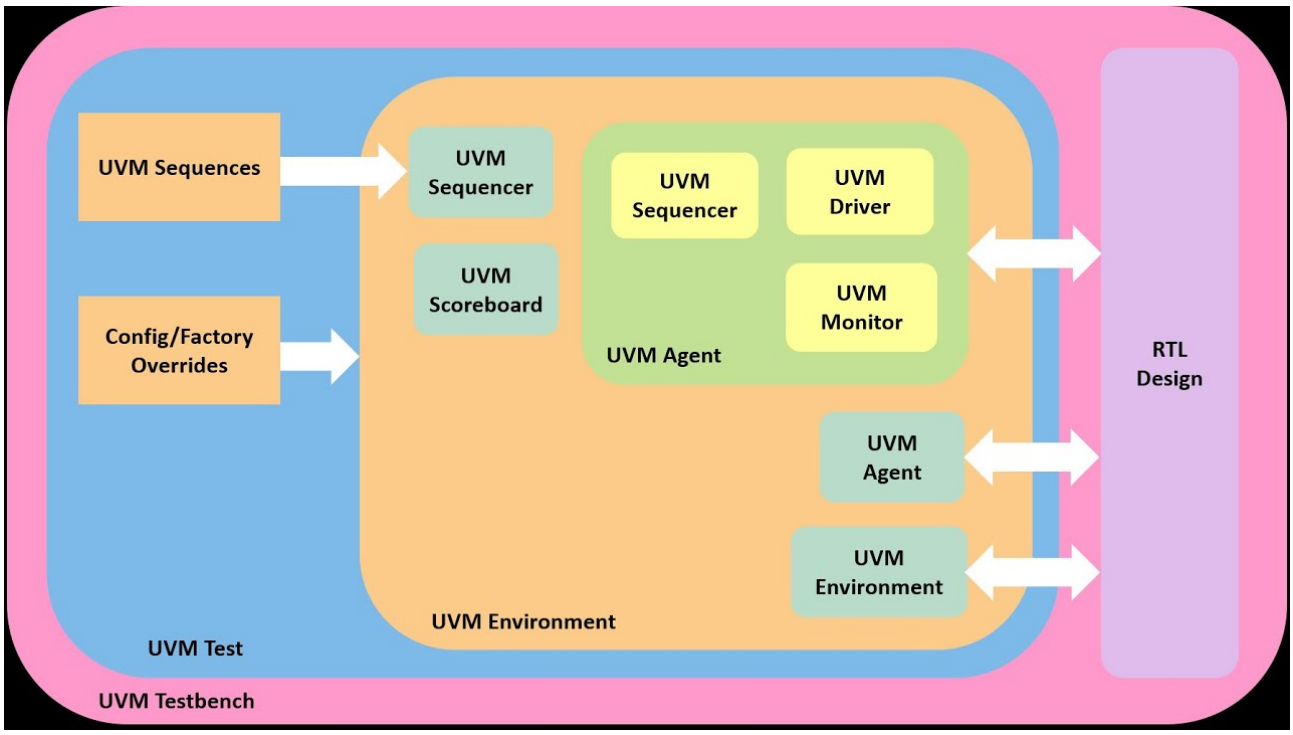

Рисунок 2 — Схема UVM тестирования

Cocotb

Cocotb представляет собой фреймворк на языке python, который позволяет использовать код на python для тестирования RTL модели. Основная идея данного метода: перенос верификации из плоскости электроники в плоскость программного обеспечения. Фреймворк позволяет пользоваться всеми инструментами языка, что делает тестирование более гибким процессом, а также существенно снижает порог входа для инженеров и программистов.

Verilator

Verilator является одновременно и инструментом и методом тестирования моделей. Программа переводит синтезируемый RTL код в классы на языке C. Дальнейшее тестирование осуществляется также на языке C. Данный подход позволяет пользоваться всеми возможностями языка, но при этом отсутствует переход в область программного обеспечения, так как управление сигналами осуществялется вручную, либо из тестовых окружений, написанных на Verilog. Плюсами данного подхода является очень маленькое время обработки и симуляции. Верификация с помощью Verilator является самым быстрым способом симулировать большой проект описания аппаратуры (некоторые тесты показывают выигрышь во времени в 2 раза или более по сравнению с

лучшими проприетарными САПР). В качестве минусов данного способа можно выдилить необходимость глубокого понимания работы языка C++, а также высокий порог входа в сборку проектов и доступность ПО только для ОС Linux.

Assertion based verification

Верификация с помощью конструкций assertion может являться как вспомогающим, так и основным способом верификации в зависимости от сложности и объема проекта. В данном подходе создаются точечные ограничения, либо целый набор правил работы устройства. В случае нарушения любого из правил симуляция останавливается и выдается сообщщение об ошибке, которое заранее прописано разработчиком или верификатором. Данный способ помогает в отладке, потому что сразу показывает где произошла ошибка, в какой момент, и что к ней привело, но при этом написание крупных наборов правил довольно сложный процесс, что делает данный подход не самым популярным.

Формальная и статический анализ

Часто источниками ошибок может быть сам код, и несмотря на компилируемость, неправильно транслироваться в логические функции. Для того чтобы избежать таких ошибок применяется формальная верификация и статичекий анализ кода. Перед началом разработки, разработчики договариваются о правилах написания кода, а также пишут специальные правила для программы linter, которая следит за исполнением этих договоренностей. Данный подход позволяет избежать ошибок еще до попадания модели к верификатору. Сейчас в качестве программ linter применяются мощные программные средства, например, Cadence Hal или JasperGold.

Прототипирование

В качестве одного из способов верификации может выступать прототипирование микросхемы с помощью FPGA. В ПЛИС загружается прошивка крупного блока или всей системы в целом (например микропроцессорная система, микроконтроллер), после чего создается программа на языке C или языке ассемблера, которая полностью проверяет

функционал системы. Такой тип верификации полезен, чтобы на практике оценить возможные отклонения системы от идеальности, также такой подход помогает верифицировать схемы без четкого разделения на программную и аппаратную части(например, USB).

Физическая верификация

Во время разработки часто возникает вопрос о синтезируемости и реализуемости модели при заданных параметрах. Чтобы удостовериться или опровергнуть реализуемость применяется физическая верификация. Данный тип верификации проводится на этапе синтеза. Разработанная модель синтезируется в логические вентили, после чего на созданные вентили накладываются задержки, случайные неидеальности, условия асинхронного расхождения и возможные гонки сигналов на линиях. На этапе физического моделирования создается netlist логических вентилей и SDF файл условий. Также на данном этапе возникаает понятие угла (corner), которое дополняет моделирование различными внешними факторами, например, температурными условиями и напряжением питания. В случае особых требований к параметрам работы устройства осуществляется проверка на просадки питания и максимальное количество срабатывания битов.

Производственная верификация

Производственная верификация (DFT - design for test) служит способом проверки работоспособности устройства после его изготовления. Для проверки готовой микросхемы на этапе разработки и тестирования в модель вносится большое количество входов и выходов. Цель данного этапа заключается в нахождении производственных ошибок технологии. На этапе верификации через специальные входы и выходы прозваниваются все регистры и логические блоки, чтобы найти битые транзисторы и ячейки памяти и определить работоспособность схемы.

ЗАКЛЮЧЕНИЕ

В наше время RTL подход к проектированию цифровой электроники является основным доминирующим способом разработки. Данный подход позволяет эффективно разрабатывать СБИС, проводить сложные процессы верификации, поддерживать существующие проекты и быстро вносить изменения. RTL имеет большое комъюнити разработчиков, а также обширное количество стандартов, инструментов, решений и новых идей. RTL занял основную роль в разработке электроники несколько десятилетий назад, и , возможно, будет занимать ее еще долгое время, в силу современных тенденций цифровизации и развития программного обеспечения.