Содержание

Вступление

Задача функциональной верификации в той или иной мере встает перед каждым инженером, занимающимся проектированием электронных систем. Доказательство того, что система функционирует так, как того требует спецификация, с ростом сложности системы требует все больших ресурсов. Причем рост сложности далеко не линеен.

Современные тенденции в микроэлектронике определяют переход к процессорно-ориентированным системам и совместной разработке аппаратной части и программного обеспечения. Разработка программного обеспечения до готовности аппаратной части требует наличия виртуальных прототипов, обеспечивающих переход на системный уровень проектирования.

Требования к росту производительности систем подталкивают к использованию нескольких процессоров и более сложных шинных архитектур. Появление аппаратных ускорителей перекладывает верификацию функционала с программистов на разработчиков аппаратуры.

Некачественная или неполная верификация может привести к некорректной работе программы уже после начала ее использования потребителем. В этом случае пострадает в первую очередь репутация разработчика.

Одновременно с указанными проблемами время выхода на рынок в условиях конкуренции постоянно сокращается, что накладывает дополнительные ограничения на процесс верификации. Чтобы справиться с возникающими сложностями, требуются разработка и внедрение новых методов верификации, более производительных, абстрактных и автоматизированных. Повышенная производительность может достигаться за счет совершенствования вычислительных систем, использования более мощных серверов, оптимизации алгоритмов, появления новых методик верификации и комбинирования существующих. Абстрактность достигается за счет использования языка C и его библиотек (таких как SystemC) с последующей автоматизированной конвертацией в языки описания аппаратуры SystemVerilog и VHDL. Автоматизацию следует проводить в области генерации тестов.

Попытка описать методы решения указанных проблем находит отражение в книге, которую Вы держите в руках. Она является переводом англоязычной публикации и ориентирована на русскоговорящих инженеров, занимающихся верификацией систем-на-кристалле, и может быть полезна для понимания сложностей, возникающих в процессе верификации сложных электронных систем, и методов их решения. Надеемся, что каждый найдет для себя в этой книге что-нибудь полезное.

Генеральный директор ЗАО «СКАН»
Ланцев А.Н.

Посвящается нашим семьям

Предисловие

В повседневной жизни мы окружены компьютерами. Мы осознаем сложность компьютерных систем, поскольку пользуемся настольными компьютерами, ноутбуками или серверами. Во многих жизненных ситуациях мы взаимодействуем с системами, которые имеют встроенные компьютеры, например, сотовые телефоны, гаджеты и системы мониторинга. В конструкции самолета или автомобиля много вычислительных устройств, совместная работа которых гарантирует, что наше путешествие будет приятным и безопасным. Можем ли мы с уверенностью сказать, что эти встроенные вычислительные системы сконструированы и изготовлены правильно и качественно, чтобы смело полагаться на их функции? Простой ответ заключается в том, что никто не может доказать их абсолютную непогрешимость. Однако, поскольку у нас нет выбора, нам остается только расслабиться и наслаждаться жизнью!

Рассмотрим простой *сумматор*, чтобы понять трудности верификации современных компьютерных систем. Сумматор является одним из простейших вычислительных устройств в калькуляторе. Он складывает два входных значения и выдает результат. Как правило, входными значениями являются 32-разрядные целые числа. Поэтому, чтобы проверить такой сумматор, мы должны смоделировать несколько триллионов ($2^{32} \times 2^{32}$) тестовых векторов. Очевидно, что слишком дорого применять триллионы входных векторов для проверки сумматора. Фактически, невозможно имитировать сумматор с использованием всех возможных входных последовательностей, когда входными значениями выступают 64-битные целые числа. Если мы не способны полностью верифицировать простой сумматор, то как можно надеяться, что мы сможем проверить современные встроенные и гибридные системы, которые состоят из сложного программного обеспечения и аппаратных средств, включая мульти- и многоядерные процессоры, память, шины, контроллеры, интерфейсы и т.д.? Функциональная валидация широко признана как основное узкое место в методологии проектирования систем-на-кристалле (SoC—System-on-Chip), поэтому на нее затрачивается до 70% от общего времени разработки и ресурсов. Несмотря на столь значительные усилия, многие конструкции СнК отказывают уже при первых тестах (отказ кремниевого кристалла) в связи с функциональными ошибками. Сложность функциональной валидации, как ожидается, вырастет еще больше в связи с совокупным влиянием возрастающей сложности проектирования и сокращением сроков выхода на рынок.

Существующие подходы к валидации используют комбинацию методов моделирования и формальной верификации. Моделирование является наиболее широко используемой формой валидации с применением генерации случайных и ограничено-случайных тестов. Например, в случае с сумматором вместо того, чтобы проверять все возможные комбинации входных данных, инженеры-валидаторы могут генерировать тесты для нужных им функциональных сценариев на основе критериев тестового покрытия и случаев предельного превышения интересующих их параметров. Так как нецелесообразно создавать и применять все возможные тесты, валидация на основе моделирования не гарантирует качество дизайна. С другой стороны, формальные методы валидации (такие как верификация моделей) не используют входных векторов, но исследуют пространство состояний для обеспечения правильности функционирования. К сожалению, это исследование может привести к расширению пространства состояний в случае валидации больших и сложных проектов. В современных методиках валидации используется эффективное сочетание обоих подходов. Для примера, полный дизайн СнК моделируется с помощью миллиардов тестов, в то время как очень важные компоненты, такие как контроллеры или конкретные протоколы, полностью верифицируются формальными методами.

Существуют различные перспективные направления, позволяющие значительно сократить общую трудоемкость валидации за счет совместного использования моделирования и формальной верификации. Они полезны для верификации, насколько ее возможно проводить на уровне спецификации (которая представляет собой оценку порядка величины и потому проще, чем верификация на уровне реализации, но позволяет исследовать все функциональные детали) и автоматического повторного использования валидации высокого уровня для верификации реализации систем низкого уровня. Аналогично, они используют направленные тесты для валидации дизайна, так как подобные тесты позволяют достичь того же целевого тестового покрытия с помощью оценки по порядку величины при меньшем количестве тестов по сравнению с использованием случайных или ограничено-случайных тестов. В результате моделирование с использованием эффективных направленных тестов может существенно снизить общую трудоемкость валидации. Однако генерация направленных тестов в основном выполняется при вмешательстве человека. Написанные от руки тесты влекут за собой трудоемкую и отнимающую много времени работу инженеров-валидаторов, которые имеют глубокие знания о конструкциях, проходящих верификацию. Для сложных конструкций невозможно вручную создать все направленные тесты для достижения всеобъемлющего тестового покрытия. Таким образом, необходимо разработать инструменты и методы для автоматической генерации направленных тестов.

Эта книга описывает инновационные пути объединения моделирования и формальных методов валидации сложных систем. В ней представлен всесторонний обзор моделирования и методов валидации на системном уровне, направленных на снижение общей трудоемкости валидации и улучшение качества продукции. Она описывает проблемы, связанные с проверкой сложных систем и представляет эффективные методы решения этих проблем. В книге рассматриваются различные методы моделирования на системном уровне, использующие языки проектирования SystemC, UML (Unified Modeling Language — унифицированный язык моделирования) и моделирование на уровне транзакций. Она представляет состояние современных методов для автоматической генерации направленных тестов с упором на сокращение времени на генерацию тестов и трудоемкости валидации посредством применения методов кластеризации, декомпозиции и обучения. Она также описывает новые пути повторного использования валидации высокого уровня на различных уровнях абстракции.

Читатель этой книги получит полное понимание задач верификации проектируемых систем и узнает, как валидация на системном уровне может значительно улучшить качество проектирования и снизить общую стоимость изделий. В главе 1 представлены преимущества валидации на системном уровне. Глава 2 описывает, как задать сложные системы, использующие высокоуровневые модели. Глава 3 содержит основные схемы автоматической генерации направленных тестов. В следующей главе рассказывается, как значительно уменьшить число тестов без ущерба для целевого тестового покрытия. В главах 5—9 описаны эффективные методы генерации направленных тестов. В следующих двух главах представлены методы генерации тестов с упором на валидацию многоядерных архитектур. Глава 12 описывает инновационные способы повторного использования сгенерированных тестов и утверждений высокого уровня для валидации систем на уровне реализации. Наконец, глава 13 завершает книгу кратким обсуждением перспективных направлений исследований. Мы надеемся, что вам понравится наша книга, в которой вы найдете полезную информацию.

Целевая аудитория

Эта книга предназначена для студентов старших курсов, аспирантов, исследователей, разработчиков инструментов САПР, проектировщиков и менеджеров, заинтересованных в развитии эффективных инструментов и методов проектирования и валидации на системном уровне, генерации направленных тестов и функциональной валидации гетерогенных конструкций СнК.

Благодарности

Эта книга является результатом десятилетних научных исследований и промышленного сотрудничества многих коллективов. Книга включает в себя описание методов и представлений о валидации на системном уровне, которые стали результатом диссертаций профессора Минсон Чэня, доктора Хун-Мо Ку и доктора Ксяоке Циня. Мы хотели бы поблагодарить наших спонсоров за оказанную финансовую поддержку в этом исследовании. Эта работа была частично профинансирована Национальным научным фондом (National Science Foundation (CAREER Award 0746261, CCF-0903430. and CNS 0905308), Semiconductor Research Corporation (2009-HJ-1979), Intel Corporation, Фондом докторантуры Министерства образования Китая 20110076120025 и средствами Государственного плана Китая по высокотехнологичным разработкам 2011AA010101.

Эта книга несет печать сотрудничества с многочисленными коллективами. Мы хотели бы отметить вклад доктора Магди Абадира (Freescale), доктора Джаянта Бхадра (Freescale), профессора Никила Датта (UC Irvine), профессора Масахиро Фуджита (Токийский университет), доктора Хуан Чжо, доктора Друбайоти Калита (Intel), профессора Тулика Митра (NUS Сингапур) и профессора Абхика Ройчудхури (NUS Сингапур). Мы также благодарны всем членам *CISE Embedded Systems Lab* Университета Флориды. В список входят доктор Канад Басу, Хади Хаджимири, Камран Рахмани, Камик Шривастава, Четан Мурти, Сеок-Фан Сеон и доктор Вейксун Ван.

Об авторах

Минсон Чэнь является адъюнкт-профессором в Software Engineering Institute при Восточно-Китайском педагогическом университете. Его исследования направлены на автоматизацию проектирования встроенных систем, разработку кибер-физических систем, методов моделирования и проверки программного обеспечения. Он получил степени бакалавра и магистра в Нанкинском университете в 2003 и 2006 годах, соответственно, и доктора наук — в Университете штата Флорида в 2010 году, все в области компьютерных наук. Он опубликовал множество научных статей в ведущих международных журналах и материалах международных конференций. Одна из его работ была номинирована на Best Paper Award Международной конференции по VLSI Design 2009. В настоящее время его исследования финансируются Национальным научным фондом Китая, Министерством образования и включены в Государственный план по высокотехнологичным разработкам. Доктор Чэнь работал в качестве члена оргкомитетов по разработке программ ряда конференций ACM и IEEE, включая SAC и ICFEM. Он является членом обеих ACM и IEEE.

Ксяоке Цинь получил степени бакалавра и магистра на факультете автоматизации Университета Цинхуа в Пекине в 2004 и 2007 годах, соответственно. Докторскую диссертацию он защищал на факультете компьютерных и информационных наук и инженерии Университета штата Флориды в 2012 году. Его исследовательские интересы лежат в области сжатия кода, верификации моделирования, валидации на системном уровне многоядерных архитектур и планировании производства встроенных систем в реальном времени. Он опубликовал множество статей в ведущих мировых журналах и материалах международных конференций. Его работа по валидации согласованности кэша была номинирована на Best Paper Award в области автоматизации проектирования и испытаний в Европе (DATE), 2012. Доктор Цинь выступал в качестве рецензента и обозревателя нескольких конференций и журналов ACM и IEEE.

Хеон-Мо Ку является специалистом по медиаархитектуре в группе параллельной обработки визуальных данных корпорации Intel. Его исследовательские интересы лежат в области проектирования и верификации встраиваемых систем для медиа-технологий, включая видеокодеки, обработку видео и машинное распознавание образов. Он получил степени бакалавра и магистра на факультете электроники и электротехники Национального университета Kyungpook в Корее в 1993 и 1995 годах, соответственно. До начала работы над докторской диссертацией он работал в компании LG Electronics Research Center в течение 8 лет. Он получил докторскую степень по вычислительной технике в Университете Флориды в 2007 году.

Прабхат Мишра является адъюнкт-профессором на кафедре компьютерных и информационных наук в Университете штата Флорида. Его исследовательские интересы включают автоматизацию проектирования встроенных систем, вычисление энергопотребления, верификацию аппаратных средств и программного обеспечения и проектирование безопасных систем. Он получил степень бакалавра в Университете Джанавпур в Калькутте в 1994 году, степень магистра по технологии в Индийском технологическом институте в Кхарагпуре в 1996 году и доктора наук в Университете штата Калифорния, Ирвин, в 2004 году, все в области компьютерных наук. До прихода в Университет штата Флорида он несколько лет работал в различных компаниях, специализирующихся в разработке полупроводниковых приборов и автоматизации проектирования, включая Intel, Motorola, Synopsys и Texas Instruments. Он опубликовал четыре книги, был соавтором десяти книг и написал более 100 научных статей в ведущих мировых журналах и материалах различных конференций. Его исследования получили несколько наград, включая премию NSF CAREER от Национального научного фонда, две премии за лучшую статью (VLSI Design 2011 и CODES+ISSS 2003), он имеет несколько номинаций на лучшую статью и премию EDAA 2004 года за выдающуюся диссертационную работу от Европейской ассоциации автоматизации проектирования (European Design Automation Association). В 2007 году он получил премию как Международный педагог года от инженерного колледжа Университета штата Флорида за значительный вклад в международные научные исследования и преподавание компьютерных наук. В настоящее время он работает в качестве помощника редактора в журналах ACM Transactions on Design Automation of Electronic Systems (TODAES), IEEE Design & Test of Computers (D&T) и Joumal of Electronic Testing (JETTA), внештатным редактором в журнале IEEE Transactions on Computers, а также выступал членом оргкомитета ряда конференций ACM и IEEE, в том числе ICCAD, DATE, ASPDAC, CODES+ISSS и VLSI Design. Он занимал должность генерального председателя на конференции IEEE High-Level Design Validation and Test (HLDVT) 2010, был руководителем группы по разработке программы конференции HLDVT 2009 года, информационным директором ACM TODAES и работал внештатным редактором в издательствах IEEE D&T, Springer JETTA и IJPP. Он является старшим членом в ACM и IEEE.

Список сокращений

ABV — Валидация на основе утверждений
ADL — Язык описания архитектуры
ATE — Автоматическое тестовое оборудование
ATM — Банкомат (автоматизированный кассовый терминал)
ATPG — Автоматический генератор тестовых векторов
BCP — Булево распространение ограничений
BDD — Двоичные диаграммы решений
BIST — Встроенное самотестирование
BMC — Проверка ограниченной модели
CNF — Конъюнктивная нормальная форма
CSP — Проблема (задача) удовлетворения ограничений
DAG — Направленный ациклический граф
DSE — Исследование пространства проектных параметров
DUV — Валидируемый дизайн
ESL — Уровень электронных систем
FSM — Конечный автомат (автомат с конечным числом состояний)
IC — Интегральная схема
LTL — Линейная временная логика
MoC — Модель вычислений
MPSoC — Мультипроцессорная система на кристалле
PSL — Язык описания свойств
RTL — Уровень регистровых передач
SAT — Выполнимость
SoC — Система на кристалле
SVA — Утверждение SystemVerilog
TLM — Моделирование на уровне транзакций
TRS — Спецификация уточняющих тестов
UMC — Проверка неограниченной модели
UML — Унифицированный язык моделирования
VSIDS — Уменьшение суммы, независимое от состояния переменной
XMI — Обмен метаданными с помощью XML

ГЛАВА 1

ВВЕДЕНИЕ

1.1. Растущая сложность валидации СнК

Система на кристалле (SoC) включает в себя все необходимые аппаратные компоненты (например, микропроцессоры, блоки памяти и периферийные устройства) и программные модули (операционные системы, работающие в реальном времени, и программы контроля), объединяя их в единую интегральную схему с требуемой функциональностью. Она может выполнять различные вычисления, в том числе цифровые, аналоговые, а также смешанные цифро-аналоговые. Рис. 1.1 представляет блок-схему архитектуры типичной многопроцессорной СнК (MPSoC — Multiprocessor System-on-Chip) [1]. Например, дизайн СнК состоит из четырех RISC-процессоров и различных общих устройств ввода-вывода, которые соединены высокопроизводительной шиной (AHB — Advanced High-Performance Bus) и периферийной шиной (APB — Advanced Peripheral Bus). СнК широко используются во встраиваемых системах, применяемых в таких областях, как автомобильная электроника, авионика, телекоммуникации, создание интеллектуальных зданий, портативных устройств и т.д.

В связи с увеличением сложности программных и аппаратных компонентов сложность проектирования современных СнК растет с угрожающей скоростью. Для размещения новых приложений и удовлетворения вычислительных требований современные СнК используют множество сложных архитектурных особенностей для поддержки синхронизации, обмена данными, динамического регулирования частоты и напряжения, когерентности кэшей, конвейеризации

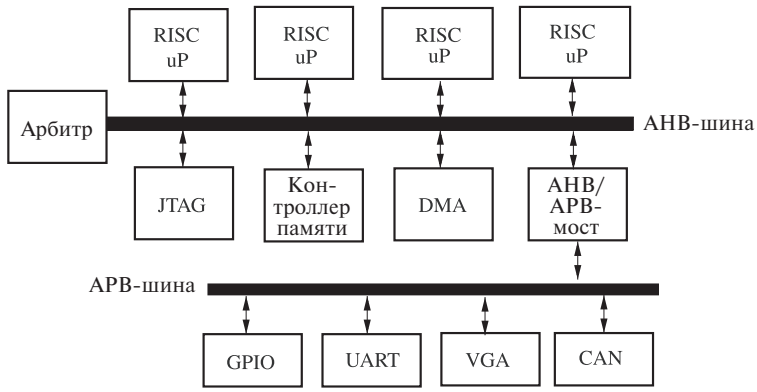

Рис. 1.1. Пример дизайна MPSoC: арбитр — устройство разрешения конфликтов

межсоединений и т.д. Увеличение сложности дизайна вызывает рост сложности верификации. Быстрое развитие технологий совместно с жесткой конкуренцией на рынке поставило разработку СнК перед проблемой: как быстро гарантировать функциональную правильность СнК? Функциональная верификация СнК широко признана в качестве основного узкого места. Недопустимо моделировать даже простой двухвходовой сумматор с использованием всех возможных входных последовательностей, так как это потребует $2^{64} \times 2^{64}$ входных (тестовых) векторов, когда каждый вход составляет 64 бита. Верификация комплексных СнК аналогична верификации миллионов и миллиардов комплексных сумматоров, взаимодействующих друг с другом в виде программного обеспечения и/или аппаратных средств. Серьезными проблемами являются обнаружение всех функциональных ошибок, а также электротехнических дефектов из-за комбинированного воздействия возрастающей сложности и ограничений, накладываемых сокращением времени выхода на рынок.

В соответствии с результатами исследований в области функциональной верификации, проведенных Ron Collett International (в 2002 и 2004 годах) и Far-West Research (в 2007 году), функциональные ошибки становятся ведущими причинами отказов СнК. Около трех четвертей ошибок при проектировании СнК связаны с функциональными ошибками (багами). Статистика показывает, что такие ошибки вводятся из-за различных факторов, в том числе небрежного кодирования, неправильной интерпретации спецификации, микроархитектурной сложности дизайна, случаев превышения допустимых параметров и так далее. Обнаружение ошибок на низком уровне реализации обходится значительно дороже по сравнению с выполнением верификации на более высоком уровне абстракции. Например, выявить ошибки легче во время предварительной валидации кремниевого кристалла по сравнению с посткремниевой валидацией, поскольку в уже изготовленном кристалле наблюдаемость внутренних сигналов будет ограничена. Как только причина ошибки будет проанализирована, кристалл придется изготавливать заново, что стоит очень дорого. В ином же случае ошибка может быть исправлена или скомпенсирована за счет существующих механизмов. Если ошибки программного обеспечения или аппаратных средств были выявлены уже на уровне потребителей изделий, это может привести к различным неприятным сценариям в зависимости от области их применения: как значительным финансовым потерям товарной продукции, так даже катастрофическим последствиям при нарушении работы критически важных систем безопасности. Например, в 1994 году процессор Intel Pentium совершил ошибку, которую называют FDIV bug[1], и компании пришлось заплатить огромную сумму (около полумиллиарда долларов), чтобы заменить дефектные процессоры. Точ-

[1] Ошибка Pentium FDIV была одной из самых позорных ошибок микропроцессоров Intel. Из-за ошибки в справочной таблице определенные операции деления с плавающей точкой давали неправильные результаты.

но так же, взрыв ракеты-носителя Ariane 5[1] был вызван ошибкой преобразования данных в управляющей программе. Поэтому так важно гарантировать функциональную правильность перед поставкой изделий.

Функциональная валидация (или верификация[2]) способна выявить логические/функциональные ошибки в готовых программах и аппаратном обеспечении, которые не соответствуют своим спецификациям.

Графики на рис. 1.2 представляют собой сравнение результатов исследования, проведенного Far West Research (пунктир) в 2007 году, и данных исследования Wilson Research Group в 2010 году (сплошная линия), они демонстрируют важность верификации в осуществлении различных промышленных проектов с точки зрения процента от общего времени проектирования, затраченного на верификацию. Можно отметить, что с 2007 по 2010 год доля верификации в общем времени проектирования значительно возросла в нескольких проектах. Исследование также показывает, что процент времени, затраченного на верификацию, продолжает расти. Таким образом, функциональная валидация, как

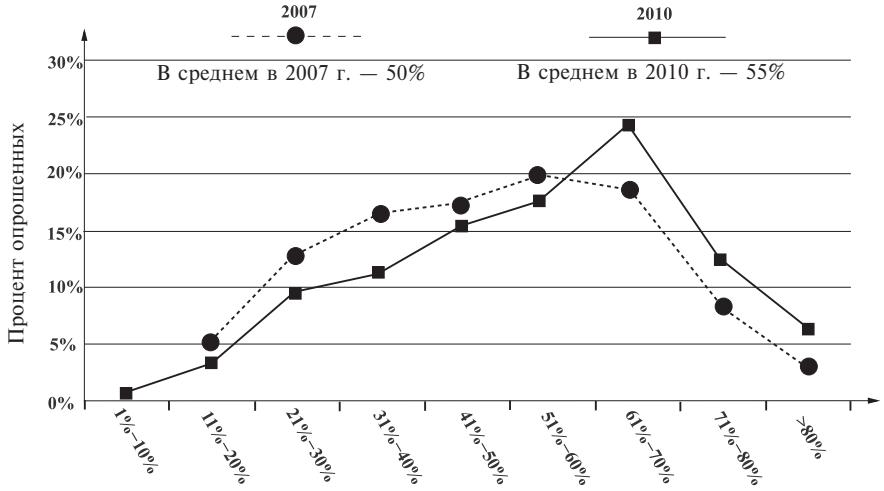

Рис. 1.2. Доля общего времени проектирования, затраченного на верификацию изделий (источник Wilson Research Group and Mentor Graphics, 2010)

[1] 4 июня 1996 года беспилотная ракета Ariane 5, запущенная Европейским космическим агентством, взорвалась через несколько секунд после старта.

[2] «Валидация» выполняется на разных уровнях абстракции, чтобы выявить два типа причин отказов: недостатки спецификации и ошибки реализации. Термин «формальная верификация» относится к использованию формальных методов для обеспечения того, чтобы изделия удовлетворяли всем требованиям спецификации. В этой книге термин «верификация» относится к использованию как валидации на основе моделирования, так и формальных методов для обнаружения ошибок реализации. Пожалуйста, обратите внимание, что термины «валидация» и «верификация» имеют различные применение и интерпретацию в различных областях техники.

ожидается, останется серьезной проблемой в методологии проектирования СнК в обозримом будущем.

Несмотря на значительные усилия по проведению функциональной валидации, большинство конструкций СнК дают отказы уже в самом начале работы (отказ кремниевого кристалла). Из результатов исследований в области функциональной верификации, проведенных в Collett International Research в 2002 и 2004 годах, а FarWest Research в 2007 году, которые показаны на рис. 1.3, можно увидеть постоянную тенденцию роста количества повторных сборок (ре-дизайн и повторное изготовление) изделий после обнаружения ошибок. Другими словами, повторная сборка становится все более частым явлением. Очевидно, что подобные операции повторной сборки стоят очень дорого и сильно влияют на сроки выхода изделий на рынок. Таким образом, необходимо развивать и использовать эффективные методы функциональной валидации для уменьшения общей трудоемкости валидации и улучшения качества продукции.

Рис. 1.3. Отраслевые тенденции в достижении функционально работающих кристаллов (источники: данные исследований функциональной верификации за 2002 и 2004 годы — Ron Collet Research, за 2007 год — FarWest Research)

1.2. Валидация на системном уровне: возможности и проблемы

В этом разделе в первую очередь акцентирована важность валидации на системном уровне и введение понятия традиционного потока нисходящих процессов проектирования и валидации. Далее дан обзор существующих методов функциональной валидации. В конце раздела обсуждаются проблемы исследований в этой области и связанные с ними возможности валидации на системном уровне.

1.2.1. Нисходящий маршрут проектирования и валидации

Предварительное исследование вариантов возможных реализаций (DSE — Design Space Exploration) требуется для создания наилучшей архитектуры СнК, которая имеет множество различных конструктивных ограничений, таких как стоимость, площадь, мощность, энергия, производительность и т.п. Автоматизированное исследование является необходимым из-за ограничений сроков вывода на рынок и тенденции добавления более сложных функций в новые приложения СнК. Поскольку уровень регистровых передач (RTL — Register Transfer Level) содержит более подробную информацию, RTL моделирование является слишком медленным для исследования больших и сложных СнК. Для повышения производительности за счет повышения скорости моделирования последние инновации в области проектирования СнК перенесли проектирование на более высокий уровень — уровень электронных систем (ESL)[1]. Проектирование на ESL специализируется на создании различных спецификаций системного уровня, которые выполняются на языках высокого уровня (например, SystemC [3],MATLAB [4], Esterel [5]) и моделировании (например, Simulink [6], SysML [7], Petri-нетто [8]) на уровне абстракции. Этот уровень выше знакомого RTL в области аппаратных средств и исходного кода в программном обеспечении[2]. На таком высоком уровне спецификации представляют определенную перспективу в описании поведения на системном уровне и широко используются в исследованиях архитектуры, совместном проектировании и совместной верификации аппаратных средств и программного обеспечения.

Традиционно проектирование СнК начинается с исследований на системном уровне и поиска подходящей спецификации архитектуры, а затем спецификация превращается в реализацию на низком уровне. Для ускорения проектирования СнК были разработаны различные программы автоматизированного проектирования электронных приборов (EDA — Electronic Design Automation) (инструменты), которые имеют большую эффективность и производительность. Как правило, методология нисходящего проектирования аппаратных средств состоит из следующих уровней абстракции:

системный/архитектурный уровень. Системный уровень сфокусирован на описании общей спецификации поведения системы и архитектурных исследованиях;

[1] В книге, посвященной проектированию и верификации на ESL [2], термин ESL определяется следующим образом: «Использование соответствующих абстракций для увеличения понимания системы и повышения вероятности успешной реализации функциональности экономически эффективным образом».

[2] В зависимости от области приложений и уровней абстракции могут применяться различные типы языков описания спецификаций, например, языки описания архитектуры (ADL) [9] очень популярны для предварительного исследования и валидации процессора на кристалле.

уровень регистровых передач. Реализация функций, полученных из спецификаций системного уровня на уровне регистровых передач, использует язык описания аппаратуры (HDL — Hardware Description Language);
уровень логических элементов. Реализация функциональной конструкции на уровне логических элементов с использованием САПР логического синтеза;
транзисторный/физический уровень. Реализация логических конструкций с использованием физических компонентов (например, транзисторов, проводников и т.д.) и инструментов их размещения и трассировки.

Проектирование программного обеспечения также имеет аналогичный нисходящий поток. Для обеих конструкций необходима значительная работа по валидации на каждом уровне проектирования СнК, чтобы гарантировать правильность реализации проекта. Поскольку эта книга фокусируется на системном уровне валидации, мы будем рассматривать только два самых высоких уровня проектирования СнК для программного обеспечения и аппаратных средств.

Рис. 1.4 представляет собой упрощенную схему проектирования и валидации на первых двух (из четырех рассмотренных выше) уровнях. Различные парадигмы моделирования аппаратных средств и программного обеспечения используются для создания спецификации СнК. Двумя наиболее широко используемыми спецификациями являются моделирование на уровне транзакций (TLM — Transaction Level Modeling) [10, 11] и унифицированный язык моделирования (UML — Unified Modeling Language) [12]. Они устанавливают стандарт высокой скорости моделирования и легкую совместимость моделей для совместного проектирования аппаратных средств и программного обеспечения. В целом, TLM является перспективным языком для моделирования аппаратных средств, а UML фокусируется на моделировании программного обеспечения. TLM в основном позволяет моделировать транзакции между раз-

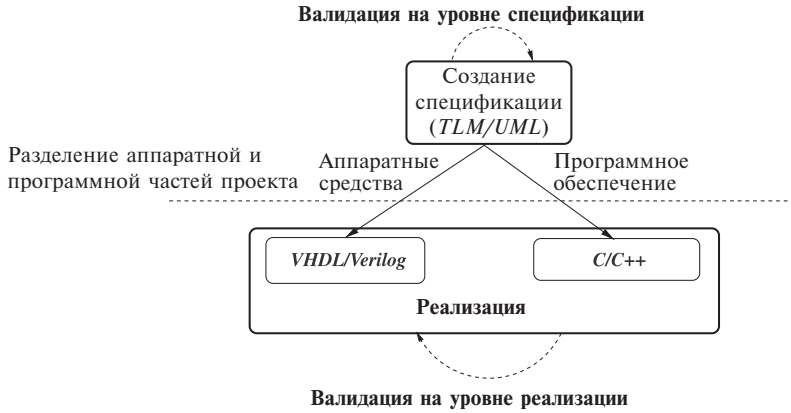

Рис. 1.4. Нисходящая последовательность проектирования и валидации СнК

личными аппаратными компонентами системы и обработку данных в каждом компоненте. UML-модели могут включать как структурную, так и поведенческую информацию о системе программного обеспечения. Утвержденные спецификации могут быть использованы в качестве эталонной модели для валидации реализованных программного обеспечения и аппаратных средств. В зависимости от стиля моделирования спецификации могут включать различный уровень детализации функциональной и временной информации. Например, TLM предоставляет два вида стилей моделирования: модели, не содержащие информации о времени исполнения транзакций (loosely timed models), могут быть использованы для моделирования поведения системы с меньшей временной информацией, а модели, содержащие примерную информацию о времени, позволяют провести временной анализ поведения системы. Хотя TLM является перспективным языком для системного уровня моделирования, все еще трудно точно описать различные реализации и дать информацию о времени их выполнения. После предварительного исследования и проведения моделирования на системном уровне RTL подходит для моделирования реализации на поведенческом уровне. В схеме, представленной на рис. 1.4, аппаратная часть реализована с использованием RTL языков, таких как VHDL или Verilog, а программное обеспечение — с использованием языков программирования, таких как C или C ++.

В то время как спецификация на системном уровне имеет преимущество для ранних исследований и валидации, введение нового уровня абстракции также вносит потенциальные источники ошибок. В соответствии с выводами, сделанными в работе [13], в настоящее время есть два ключевых источника отказов СнК (приводящих к перевыпуску кристалла): ошибки реализации и спецификации. Было обнаружено, что 82% проектов с перевыпуском в результате функциональных отказов имели ошибки реализации. Интересно отметить, что почти 47% проектов, перевыпущенных в результате функциональных отказов, имели также неправильные или неполные спецификации [13]. Во время пошагового уточнения от системного уровня до уровня транзисторов также были выявлены ошибки системного уровня, которые ранее было труднее обнаружить. Кроме того, очень дорого выявлять и исправлять ошибки на поздних стадиях цикла разработки по сравнению с выявлением, проводимым на ранних стадиях проектирования. Таким образом, важно гарантировать правильность спецификаций на системном уровне. В последнее десятилетие проводятся многочисленные исследования как на производстве, так и в научных кругах по разработке методологий валидации на системном уровне [14—16]. Спецификации на системном уровне позволяют повысить скорость моделирования и упростить интерфейс обмена данными. Путем выполнения максимального количества циклов валидации на ранних стадиях проектирования можно уменьшить общую стоимость валидации и повысить качество реализации. Диапазон

проводимых исследований включает работы от изучения методов моделирования до использования формальных методов. В этой книге мы сосредотачиваемся на валидации на основе моделирования как спецификаций, так и реализации с использованием эффективных направленных тестов. Особенно подробно в этой книге рассматриваются различные методы моделирования на высоком уровне и методы генерации направленных тестов для уменьшения общей трудоемкости валидации. Следовательно, стоимость генерации тестов и их качество представляют собой две основные задачи, которые будут рассмотрены в следующих главах.

1.2.2. Методы валидации СнК

Методы валидации СнК могут быть разделены на две категории: формальная верификация и моделирование. На рис. 1.5 показаны некоторые из широко используемых методов в этих двух категориях. Далее в этом разделе дается обзор этих методов валидации проектирования.

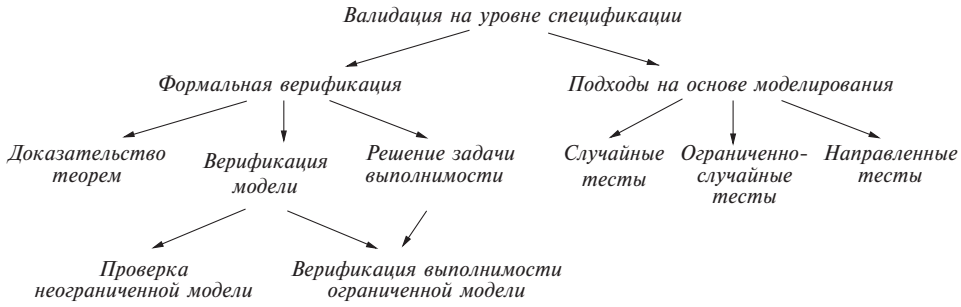

Рис. 1.5. Методы валидации СнК

1.2.2.1. Валидация на основе моделирования

Валидация на основе моделирования [17] позволяет обнаружить ошибки проектирования благодаря использованию векторов тестовых последовательностей, состоящих из входных воздействий и ожидаемых выходных результатов [18—21]. На рис. 1.6 показана схема традиционной валидации на основе моделирования. Валидация состоит из трех основных процедур: 1) генерации тестовых векторов, 2) моделирования конструкций СнК с использованием тестовых векторов, 3) сравнения сгенерированных выходных сигналов с ожидаемыми результатами. В процессе валидации тестируемые параметры показывают, какая часть дизайна не была верифицирована и какие тесты должны быть добавлены. Для обнаружения различных типов ошибок проектирования на различных уровнях абстракции дизайна предлагается тестировать различные параметры. Генерация направленных тестов с управляемым тестовым покры-

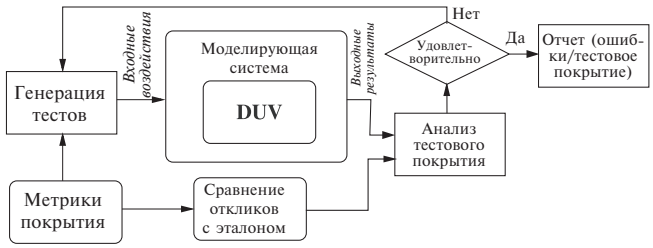

Рис. 1.6. Последовательность валидации на основе моделирования

тием предназначена для активации точки целевого покрытия, что эффективно сокращает количество тестов по сравнению со случаем применения случайных тестов. Посредством моделирования эффективность тестового покрытия анализируется на предмет того, покрыты ли целевые функции тестами или нет, таким образом, можно измерить ход валидации. Если найдены пробелы в покрытии (т.е. тестирование оказалось недостаточным), создаются дополнительные тесты для их валидации. Если требуется высокая степень доверия, мы можем улучшить метрику покрытия или использовать дополнительные методы тестового покрытия. Инженеры-тестировщики могут изменять область и глубину покрытия в течение процесса валидации. Для примера, они могут начать с простых метрик покрытия на ранней стадии валидации и использовать более сложные метрики покрытия позже.

Генерация тестов играет важную роль в валидации, управляемой моделированием. Она определяет не только качество тестирования, но также сильно влияет на время моделирования. Есть три типа методов генерации тестов: генерация случайных, ограниченно-случайных и направленных тестов. В текущей производственной практике [22, 23] наиболее широко используются методы генерации случайных и ограниченно-случайных тестов, потому что тестовые векторы могут быть получены автоматически, следовательно, большая часть ошибок дизайна может быть обнаружена в начале цикла разработки. Тем не менее, необходимо огромное количество тестов для достижения высокой степени уверенности в правильности дизайна, поскольку можно легко пропустить состояния дизайна, в которых наступает отказ (например, состояния конвейера, соотношения тактовых сигналов). По сравнению со случайными методами тестирования, которые используют миллиарды и триллионы случайных и псевдослучайных тестов, в традиционном направленном тестировании дизайна применяется меньше тестов для получения требуемого тестового покрытия функций, поскольку в данном случае используется информация о структуре дизайна [24, 25]. Путем применения направленных тестов общая трудоемкость валидации может быть уменьшена без сокращения целевого тестового покрытия. Большинство методов генерации направленных тестов требуют эксперт-

ных *знаний валидируемого дизайна* (DUV — Design Under Validation). Как правило, методы генерации направленных тестов являются трудоемкими и подвержены ошибкам. Из-за неизбежного вмешательства человека невозможна генерация всех направленных тестов для достижения всеобъемлющего целевого покрытия в разумные сроки. Эта книга описывает несколько эффективных методов выполнения автоматической генерации направленных тестов.

Методы моделирования достаточно хорошо работают при валидации сложных конструкций. Легко проверить, активируется ли функциональный сценарий данным тестом при валидации дизайна с многомиллионным количеством соединений. Тем не менее, достаточно трудно гарантировать правильность проектирования, используя валидацию на основе моделирования. Например, чтобы обеспечить правильность дизайна СнК, требуется проверить все возможные входные стимулы. Однако невозможно сгенерировать и моделировать все возможные входные последовательности, так как их число может быть чрезмерно большим (потенциально бесконечным!). Как правило, используются триллионы случайных или ограниченно-случайных тестов при моделировании из-за ограничений сроков выхода на рынок и в силу других причин. Так как применяются только выборочные входные воздействия (вместо валидации всех возможных входных последовательностей), валидация не дает уверенности в правильности дизайна. Нельзя считать, что непротестированный процесс (функциональность) является корректным или не достижим. Кроме того, существует нехватка хороших метрик покрытия для количественной оценки степени доверия и квалификации набора тестов. Поэтому трудно ответить на вопрос: *«Когда верификация завершена?»* из-за сложностей оценки прогресса верификации и эффективности тестов.

1.2.2.2. Формальная верификация

Формальные методы верификации [26, 27] обеспечивают полноту верификации путем математических доказательств правильности дизайна. Некоторые из широко используемых формальных методов включают в себя верификацию модели [28, 29], доказательство теорем [30, 31], проверку эквивалентности [32] и *решение применимости* (SAT — Satisfiability Solving) [33, 34]. Подходы на основе верификации моделей широко приняты при валидации СнК. На рис. 1.7 показаны основные представления о процессе верификации модели. Во-первых, он переводит DUV в формальную модель, а цель валидации (VT — Validation Target) в свойство в виде временной логики [29]. Свойство описывает функциональность конкретного процесса (например, экземпляр функционального покрытия). Верификатор модели проверяет, удовлетворяет ли DUV свойству путем изучения всех достижимых состояний DUV.

Если свойство не выполняется (т.е. существует некоторое состояние, которое нарушает свойство), верификатор модели предлагает контрпример для

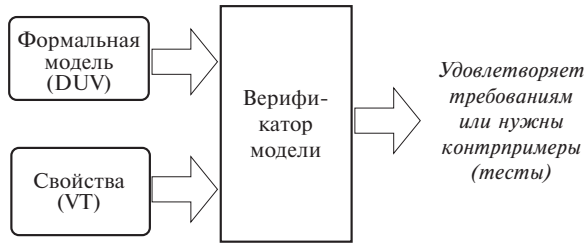

Рис. 1.7. Последовательность валидации с использованием верификации модели

свойства. Как описано в главе 3, фальсификация свойства может быть использована для получения контрпримеров, которые в свою очередь могут быть применены в качестве направленных тестов [35]. Обратите внимание, что сложность верификации свойства значительно выше, чем моделирование дизайна с использованием теста. Это связано с тем, что моделирование подтверждает влияние теста в конкретные сроки, в то время как верификация модели (свойства) требует обеспечения правильности (реализация удовлетворяет требованиям к свойству) за весь срок службы системы. Эта сложность поиска формальных методов может привести к значительному росту пространства состояний[1] в случае больших и сложных конструкций. В связи с ограничениями пространства состояний формальные методы, как правило, используются для верификации небольших, но важных компонентов, в то время как методы моделирования более удобны для валидации больших конструкций, в которых приходится жертвовать всей полнотой валидации.

1.2.2.3. Полуформальная валидация

Методы формальной верификации способны обеспечить гарантию правильности функционирования, но могут привести к значительному росту пространства состояний при работе со сложными проектами. С другой стороны, с помощью методов моделирования можно обрабатывать большие сложные конструкции, но невозможно полностью доказать правильность системы за счет сложности входного пространства состояний[2]. Для объединения преимуществ обоих видов верификации — формальной и на основе моделирования — при валидации СнК все чаще используются полуформальные методы валидации, которые представляют собой компромисс между моделированием и формальными методами валидации. Например, валидация на основе утверждений (ABV — Assertion Based Verification) [36] является популярным полуформаль-

[1] Размер пространства состояний растет экспоненциально с ростом количества переменных состояния системы.

[2] Входное пространство состояний (все возможные входные последовательности) может оказаться слишком большим для сложных конструкций с большим числом входов.

ным методом, который расширяет понятие утверждения при применении метода моделирования с помощью семантики линейной временной логики (linear temporal logic semantics). На рис. 1.8 показано, что применение полуформальной верификации с использованием утверждений/свойств выросло на 53% с 2007 по 2010 год. Благодаря утверждениям ABV не только увеличивает наблюдаемость дизайна при моделировании, но также имеет определенные преимущества перед формальными методами, улучшая общее качество верификации и ее результаты.

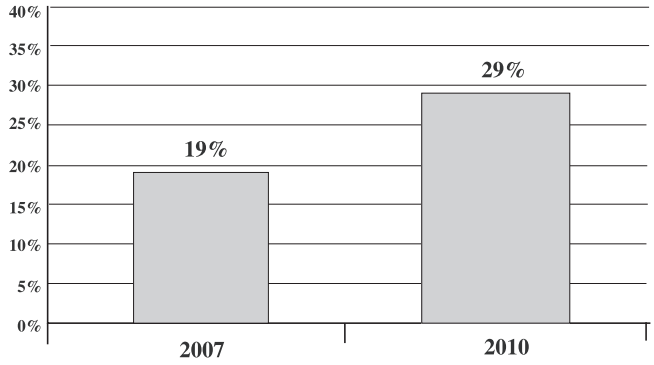

Рис. 1.8. Рост применения полуформальной верификации утверждения/свойства (источник: Wilson Research Group and Synopsys, 2011)

В настоящее время существует два самых популярных языка ABV: язык описания спецификаций свойств (PSL — Property Specification Language) [37] и язык утверждений SyslemVerilog (SVA — SyslemVerilog Assertion) [38]. Язык PSL является платформенно-независимым и может быть использован в многоуровневой конструкции. Язык SVA похож на PSL, но был адаптирован для дизайнов SystemVerilog. Несмотря на то, что полуформальная верификация расширяет временные возможности утверждений, она по-прежнему в основном базируется на моделировании. Аналогично методам моделирования полнота валидации так и не может быть гарантирована. Инженеры-верификаторы вынуждены решать различные проблемы, в том числе определять, сколько утверждений достаточно для валидации, как активировать данное утверждение, насколько часто следует проверять утверждение и т.д.

Табл. 1.1 представляет сравнение трех описанных методов валидации. Вполне очевидно, что как моделирование, так и формальные методы имеют свои плюсы и минусы. Следовательно, в начале проекта разработчики должны определить надлежащий план валидации, который подходит для их проекта. Как правило, наиболее подходящий метод зависит от проекта, а также от функциональности и полноты охвата методов валидации.

Таблица 1.1. Сравнение трех методов верификации

Методы	Моделирование	Полуформальный метод	Формальный метод
Полнота покрытия	Низкая	Средняя	Высокая
Сходимость тестового покрытия	Медленная	Медленная	Быстрая
Автоматизация генерации направленных тестов	Низкая	Средняя	Высокая
Масштабируемость	Высокая	Средняя	Низкая

1.2.3. Возможности валидации на системном уровне

Так как спецификация на системном уровне используется в качестве эталонной модели в нисходящем проектировании СнК, функциональная ошибка в спецификации приведет к ошибкам в реализации. Найти ошибки в реализации — более трудоемкий процесс, поскольку валидация на уровне реализации более сложна, чем верификация спецификаций на системном уровне. Таким образом, во время валидации спецификаций на системном уровне имеет смысл проверить максимальное количество функциональных сценариев. Интересно отметить, что один и тот же набор функциональных сценариев должен быть подтвержден как на уровнях спецификации, так и реализации. Благодаря повторному использованию одинаковых функциональных сценариев на различных уровнях абстракции общая трудоемкость валидации может быть значительно снижена.

На рис. 1.9 сравниваются различные задачи, относящиеся к уровням спецификации и реализации. Предположим, что дизайн D имеет общее число функциональных сценариев F_{Total}, которые должны быть проверены. Допустим, что имеются F_{Spec} функциональных сценариев, которые можно активировать на уровне спецификации. Также предположим, что для генерации тестов для каждого функционального сценария на уровне спецификации нужно затратить среднее время T_{Spec}. В дополнение к F_{Spec} функциональных сценариев есть еще F_{Imp} функциональных сценариев, которые могут быть ак-

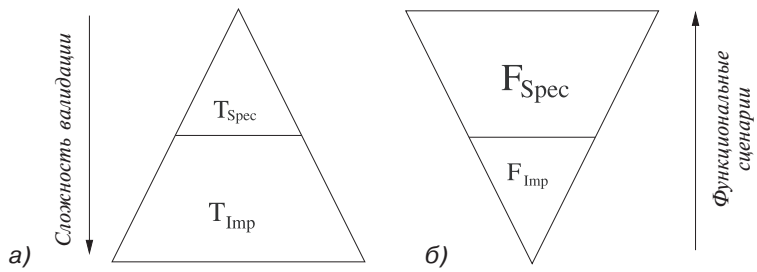

Рис. 1.9. Функциональная валидация на уровне спецификации и реализации

тивированы только на уровне реализации. Предположим, что активация каждого сценария реализации (т.е. время генерации тестов) занимает среднее время T_{Imp}. На рис. 1.9, *а* показано, что при проверке функционального сценария валидация на уровне реализации существенно дороже, чем валидация на уровне спецификации. Рис. 1.9, *б* в свою очередь свидетельствует, что большая часть общих функциональных сценариев системы может быть активирована на уровне спецификации, а реализация унаследует все эти сценарии, добавив свои собственные новые дополнительные функциональные сценарии в связи с введением деталей реализации. В этой книге сложность валидации функционального сценария относится к валидации СнК при использовании направленных тестов. Таким образом, трудоемкость валидации функционального сценария включает в себя время генерации тестов, время моделирования, а также связанное с ними время отладки и удаления ошибок (если таковые имеются).

$$\text{Минимизировать: } F_{\text{Spec}} \times T_{\text{Spec}} + \{F_{\text{Spec}} + F_{\text{Imp}}\} \times F_{\text{Imp}}$$

$$\text{при условии, что: } \begin{cases} F_{\text{Spec}} + F_{\text{Imp}} = F_{\text{Total}} \\ T_{\text{Spec}} << T_{\text{Imp}} \\ F_{\text{Spec}} > F_{\text{Imp}} \end{cases} \tag{1.1}$$

Для достижения цели функционального покрытия, а также сведения к минимуму общей трудоемкости валидации необходимо найти метод оптимизации процесса, изображенного на рис. 1.1. Для генерации направленных тестов существует четыре возможных варианта:

без оптимизации. Генерация тестов на уровне спецификации и реализации представляет собой два независимых процесса, на каждом уровне не производится никакой оптимизации;

оптимизация на уровне спецификаций. Генерация тестов на уровне спецификации и реализации представляет собой два независимых процесса. Общее время генерации тестов на уровне спецификации может быть уменьшено с помощью определенных методов оптимизации;

повторное использование тестов на уровнях спецификации и реализации. При генерации тестов на уровнях спецификации и реализации оптимизация не проводится, но тесты, сгенерированные на уровне спецификации, могут быть использованы для осуществления валидации на уровне реализации;

оптимизация генерации тестов на уровне спецификации + повторное использование тестов на уровнях спецификации и реализации. С помощью оптимизации уменьшается общее время генерации тестов на уровне спецификации, а затем тесты, сгенерированные на уровне спецификации, могут быть повторно использованы для осуществления валидации на уровне реализации.

Табл. 1.2 сравнивает эти подходы. Предположим, что во время валидации на системном уровне мы можем провести оптимизацию генерации тестов на уровне спецификации, которые способны в α раз ($\alpha > 1$) снизить время по сравнению с традиционными методами генерации тестов. Кроме того, предположим, что мы можем получить ускорение в β раз ($\beta > 1$) за счет повторного использования тестов при валидации. Согласно сравнению, представленному в табл. 1.2, последний вариант является наиболее выгодным. Эта книга описывает эффективную генерацию тестов и методы их повторного использования для снижения общей трудоемкости валидации с помощью четвертого (последнего) варианта.

Таблица 1.2. Сравнение четырех вариантов генерации направленных тестов

Оптимизация	Затраченное время
Без оптимизации	$F_{Spec} \times T_{Spec} + F_{Total} \times T_{Imp}$
Оптимизация на уровне спецификаций	$F_{Spec} \times T_{Spec}/\alpha + F_{Total} \times T_{Imp}$
Повторное использование тестов на уровнях спецификации и реализации	$F_{Spec} \times T_{Spec} + F_{Spec} \times T_{Imp}/\beta + F_{Imp} \times T_{Imp}$
Оптимизация генерации тестов на уровне спецификации + повторное использование тестов на уровнях спецификации и реализации	$F_{Spec} \times T_{Spec}/\alpha + F_{Spec} \times T_{Imp}/\beta + F_{Imp} \times T_{Imp}$

1.2.4. Проблемы валидации на системном уровне

Нисходящие процессы проектирования и валидации представляются перспективными, поскольку спецификацию существенно проще проанализировать, изучить, оптимизировать и проверить по сравнению со сложной реализацией. В связи с постоянно растущим потребительским спросом на сложные приложения в сочетании с другими ограничениями дизайна сложность спецификаций на системном уровне также интенсивно возрастает. Во время валидации на системном уровне каждый из компонентов (таких как IP-ядра, процессоры и память) может быть верифицирован с использованием существующих методов валидации. Тем не менее, валидация всей системы в целом является чрезвычайно сложной задачей из-за экспоненциально большого числа возможных взаимодействий, которые чрезвычайно сложно смоделировать, проанализировать и проверить. Возрастающее число разнородных компонентов (например, гетерогенных ядер и GPU) делает интеграцию СнК на системном уровне все усложняющейся проблемой. Потребуются значительные усилия по валидации для проверки указанных функциональных сценариев спецификаций на системном уровне, а также верификации соответствия между спецификацией

и реализацией. Чтобы использовать валидацию, управляемую спецификацией, должны быть решены следующие проблемы:

моделирование и спецификации сложных СнК. На системном уровне спецификации с формальной однозначной семантикой можно подвергнуть автоматическуой верификации с самого начала процесса проектирования. Прошедшая полную валидацию спецификация может использоваться в качестве золотой эталонной модели для последующей детализации (например, генерации RTL моделей), а также автоматизированного повторного использования при валидации высокого уровня на разных уровнях абстракции. Здесь возникают две конкретных проблемы:

- *полуформальная семантика*: большинство языков описания спецификаций СнК являются полуформальными, что препятствует автоматизации процесса валидации. Поэтому сложной проблемой представляется извлечение формальных моделей из спецификаций высокого уровня, что необходимо для осуществления нисходящего процесса валидации систем на кристалле;
- *отсутствие комплексных метрик покрытия:* традиционные методы валидации базируются на покрытии кода, переходов и т.д., чтобы оценить адекватность валидации. К сожалению, таких метрик покрытия недостаточно, чтобы показать прогресс валидации. Основная задача заключается в определении набора всесторонних моделей функциональных отказов и связанных с ними метрик покрытия для валидации на системном уровне.

Валидация с использованием направленных тестов. Из-за ограничения сроков выхода на рынок абсолютно необходимо уменьшить общую трудоемкость валидации путем использования эффективных направленных тестов. В этом контексте существуют две проблемы:

- *эффективная генерация направленных тестов:* использование направленных тестов является перспективным методом, поскольку они требуют меньшего числа тестов по сравнению с использованием случайных тестов при том же целевом функциональном покрытии. В настоящее время большинство методов генерации направленных тестов предполагают эксперт-знание, которое отнимает много времени и подвержено ошибкам. Верификация модели на основе генерации контрпримера (направленного теста) может быть полностью автоматизирована. Тем не менее, в связи с проблемой значительного роста пространства состояний затраты на генерацию большого количества тестов очень велики, кроме того ее просто невозможно выполнить для многих сценариев. Важно существенно снизить общую сложность генерации тестов, чтобы сделать ее примени-

мой для сложных конструкций СнК, включая многоядерные архитектуры с протоколами когерентности кэшей;

- *уплотнение направленных тестов*: одного и того же целевого тестового покрытия можно достигнуть с помощью меньшего числа различных направленных тестов с разными порядками величин по сравнению с числом случайных тестов, необходимых для тех же целей. Тем не менее, число направленных тестов все еще может быть очень большим (порядка миллионов) для сложных современных СнК. Важной задачей является уменьшение избыточности сгенерированных тестов, не жертвуя при этом функциональным покрытием цели.

Соответствие между спецификацией и реализацией. Типичный нисходящий процесс проектирования включает в себя несколько референсных моделей на различных уровнях абстракции. Наличие нескольких референсных моделей поднимает важный вопрос: как мы можем поддерживать согласованность между большим количеством этих моделей? При преобразовании спецификации на системном уровне для более низких уровней абстракции важно обеспечить согласованность между различными уровнями абстракции. Одним из перспективных направлений является использование того же набора тестов, утверждений и свойств для спецификации и реализации. Основная задача заключается в том, как повторно использовать свойства и тесты на уровне спецификации для валидации СнК на уровне реализации.

1.3. Комплексный подход к валидации на системном уровне

В центр внимания этой книги мы поставили существующие методы валидации на системном уровне, которые позволяют сократить общую трудоемкость валидации СнК. Рис. 1.10 представляет обзор общей структуры валидации на системном уровне, которая состоит из четырех основных компонентов:

моделирование и спецификации дизайна СнК. Моделирование играет важную роль в валидации на системном уровне. В главе 2 представлены различные широко используемые языки описания спецификаций СнК и способы их перевода для формальных моделей для обеспечения автоматизированного анализа и валидации;

генерация свойств, управляемая тестовым покрытием. Функциональное тестовое покрытие играет большую роль в определении адекватности функциональной валидации. В главе 3 определены различные модели отказов для спецификаций СнК. Основываясь на этих моделях отказов, описаны методы автоматической генерации свойств для валидации конкретных функциональных сценариев;

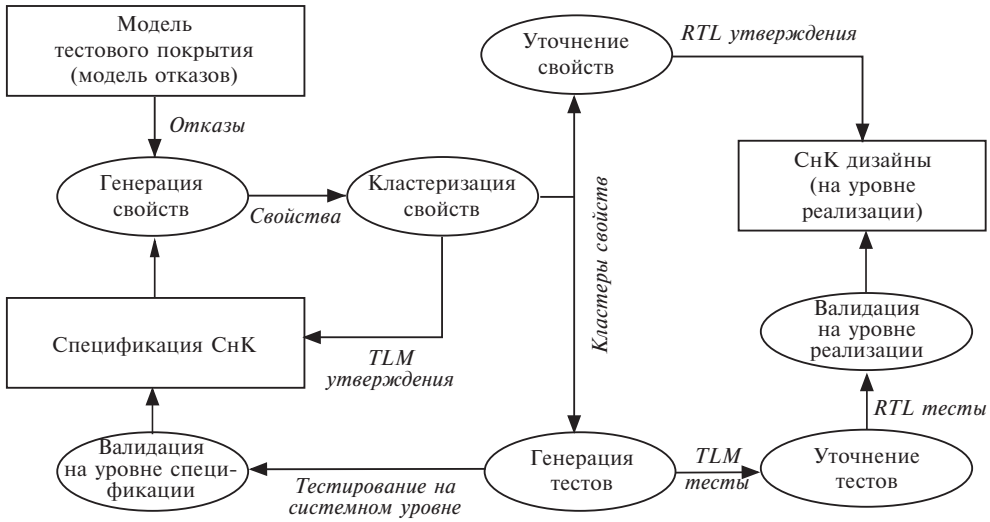

Рис. 1.10. Нисходящая последовательность валидации дизайнов СнК

генерация направленных тестов. Основное внимание в этой книге направлено на рассмотрение множества эффективных методов автоматической генерации направленных тестов для самых разнообразных конструкций СнК. Чтобы разрешить проблему роста пространства состояний при существующих способах верификации моделей, предлагаются эффективные методы генерации тестов для уменьшения времени генерации и снижения требований к памяти, которые описаны в главах 4—11;

автоматизированное повторное использование тестов на различных уровнях для снижения трудоемкости валидации. Глава 12 представляет собой основу для автоматического повторного использования тестов и утверждений высокого уровня для осуществления валидации на уровне реализации.

1.4. Структура книги

Структура книги представлена следующим образом.

Глава 2. *Моделирование и спецификации дизайнов СнК*. В этой главе рассказывается, как извлечь формальные модели из спецификаций и моделей отказов, опираясь на метрики функционального покрытия для проведения высокоуровневой валидации.

Глава 3. *Автоматизированная генерация направленных тестов*. В этой главе описывается, как автоматически получить тесты, используя различные методы верификации моделей, в том числе методы неограниченной и ограниченной SAT верификации моделей (SAT-Based Bounded Model Checking — BMC).

Глава 4. *Уплотнение функциональных тестов*. В этой главе предлагается метод уплотнения тестов, который позволяет значительно сократить количество направленных тестов без ущерба для целей функционального покрытия.

Глава 5. *Кластеризация свойств и методы обучения*. В этой главе представлены различные формулировки кластеризации и конфликтов на основе обучающих методов, чтобы улучшить общее время генерации тестов.

Глава 6. *Порядок принятия решений под управлением методов обучения*. Эта глава описывает порядок принятия решений на основе методов обучения, которые позволяют эффективно сократить время генерации тестов для одного свойства, а также для кластера аналогичных свойств.

Глава 7. *Синхронизированная генерация направленных тестов*. В этой главе рассматривается эффективный BMC-метод генерации тестов, который одновременно позволяет исследовать несколько свойств для того, чтобы максимально использовать полученные знания.

Глава 8. *Генерация тестов с использованием декомпозиции дизайна и свойств*. Для решения задачи относительно значительного роста пространства состояний в этой главе представлены перспективные методы декомпозиции дизайна и свойств, которые помогут сократить время генерации тестов.

Глава 9. *Методы декомпозиции свойств, управляемой обучением*. Для упрощения процесса композиции субтестов, описанного в главе 8, в этой главе рассматривается полностью автоматизированный подход к генерации тестов, основанный на декомпозиции свойств и установлении порядка принятия решений на основе методов обучения.

Глава 10. *Генерации направленных тестов для многоядерных архитектур*. За счет повторного использования информации, полученной в ходе исследования одного ядра, для валидации других оставшихся ядер предлагается новый BMC-метод генерации тестов для многоядерных архитектур.

Глава 11. *Генерации тестов для валидации согласованности кэша*. В этой главе рассматривается метод генерации тестов в реальном времени для валидации протоколов когерентности кэшей за счет эффективного перемещения структуры пространства состояний соответствующих глобальных FSM.

Глава 12. *Повторное использование тестов для снижения трудоемкости валидации на системном уровне*. В этой главе представлена схема валидации, которая включает в себя повторное использование сгенерированных тестов и свойств для осуществления валидации на уровне реализации.

Глава 13. *Заключение* завершает книгу, в последней главе также предлагаются интересные направления для дальнейших исследований в области валидации на системном уровне.

Литература

1. Wolf W, Jerraya A, Martin G (2008) Multiprocessor system-on-chip (MPSoC) technology. IEEE Trans Comput Aided Des Integr Circuits Syst (TCAD) 27(10):1701-1713.
2. Bailey B, Martin G, Piziali A (2007) ESL design and verification: a prescription for electronic system level methodology. Morgan Kaufmann/Elsevier, Burlington.
3. Open SystemC Initiative (OSCI) (2006) SystemC. http://www.systemc.org.
4. MATLAB. http://www.mathworks.com/products/matlab/.
5. Berry G, Gonthier G (1992) The esterel synchronous programming language: design, semantics, implementation. Sci Comput Program 19(2):87-152.
6. Simulink. http://www.mathworks.com/products/simulink/.
7. SysML. http://www.sysml.org/.
8. Peterson J (1981) Petri nets theory and the modeling of systems. Prentice-Hall, Englewood Cliffs.
9. Mishra P, Dutt N (2008) Processor description languages: applications and methodlogies. Morgan Kaufmann, San Fransisco.
10. Cai L, Gajski D (2003) Transaction level modeling: an overview. In: Proceedings of international conference on hardware/software codesign and system, synthesis (CODES+ISSS), pp 19-24.
11. Rose A, Swan S, Pierce J, Fernandez J (2005) Transaction level modeling in systemC. http:// www.systemc.org.
12. Object Management Group (2007) UML superstructure V2.1.2. http://www.omg.org/docs/formal/07-11-02.pdf. Accessed Nov 2007.
13. Schutten R, Fitzpatrick T (2003) Design for verification methodology allows silicon success. EETIMES, Number: 16500856.
14. Bentley B (2002) High level validation of next generation microprocessors. In: Proceedings of high level design validation and test (HLDVT), pp 31-35.
15. Schubert T (2003) High level formal verification of next generation microprocessors. In: Proceedings of design automation conference (DAC), pp 1-6.
16. Fine S, Ziv A (2003) Coverage directed test generation for functional verification using Bayesian networks. In: Proceedings of design automation conference (DAC), pp 286-291.
17. Bryant R (1991) A methodology for hardware verification based on logic simulation. J ACM 38(2):299-328.
18. Adir A, Asaf S, Fournier L, Jaeger I, Peled O (2007) A framework for the validation of processor architecture compliance. In: Proceedings of design automation conference (DAC), pp 902-905.
19. Ezer S, Johnson S (2005) Smart diagnostics for configurable processor verification. In: Proceedings of design automation conference (DAC), pp 789-794.
20. Puig-Medina M, Ezer G, Konas P (2000) Verification of configurable processor cores. In: Proceedings of design automation conference (DAC), pp 426-431.
21. Roy A, Panda S, Kumar R, Chakrabarti P (2005) A framework for systematic validation and debugging of pipeline simulators. ACM Trans Des Autom Electron Syst (TODAES) 10(3):462-491.
22. Adir A, Almog E, Fournier L, Marcus E, Rimon M, Vinov M, Ziv A (2004) Genesys-Pro: innovations in test program generation for functional processor verification. IEEE Des Test Comput 21(2):84-93.
23. Shimizu K, Gupta S, Koyama T, Omizo T, Abdulhafiz J, McConville L, Swanson T (2006) Verification of the cell broadband engine processor. In: Proceedings of design automation conference (DAC), pp 338-343.
24. Koo H, Mishra P (2006) Functional coverage-driven test generation for microprocessor verification. In: Proceedings of US-Korea conference (UKC), pp 19-24.
25. Mishra P, Dutt N (2005) Functional verification of programmable embedded architectures: a top-down approach. Springer, Berlin.

26. Camurati P, Prinetto P (1988) Formal verification of hardware correctness: introduction and survey of current research. IEEE Comput 21(7):8-19.
27. Kern C, Greenstreet M (1999) Formal verification in hardware design: a survey. ACM Trans Des Autom Electron Syst (TODAES) 4(2):123-193.
28. McMillan K (1993) Symbolic model checking: an approach to the state explosion problem. Kluwer Academic Publishers, Boston.
29. Clarke E, Grumberg O, Peled D (2000) Model checking. The MIT press, Cambridge.
30. Srivas M, Bickford M (1990) Formal verification of a pipelined microprocessor. IEEE Softw 7(5):52-64.
31. Wilding M, Greve D, Hardin D (2001) Efficient simulation of formal processor models. Formal Methods Syst Des 18(3):233-248.
32. Kuehlmann A, Eijk C (2001) Combinational and sequential equivalence checking. In: Logic synthesis and verification, Kluwer Academic Publishers, Norwell.
33. Clarke E, Biere A, Ramimi R, Zhu Y (2001) Bounded model checking using satisfiability solving. Formal Methods Syst Des 19(1):7-34.
34. Prasad M, Biere A, Gupta A (2005) A survey of recent advances in SAT-based formal verification. Int J Softw Tools Technol Transfer (STTT) 7(2):156-173.
35. Ammann P, Black P, Majurski W (1998) Using model checking to generate tests from specifications. In: Proceedings of international conference on formal engineering methods (ICFEM), pp 46-54.
36. Foster H, Krolnik A, Lacey D (2004) Assertion-based design, 2nd edn. Kluwer Academic Publishers, Boston.
37. PSL working group (2005) Property Specification Language. http://www.eda.org/ieee-1850/.
38. SVA committee (2004) SystemVerilog Assertion. http://www.eda.org/sv-ac/.