Проектирование в условиях временных ограничений: верификация проектов

Статья продолжает рассмотрение вопросов создания проектов на основе ПЛИС с условием максимального ускорения процедуры проектирования. Если в предшествующих выпусках рассматривались средства и методы, применяемые на заключительных этапах проектирования (этапе отладки), то в последующих статьях преимущественное внимание будет уделено начальным этапам проектирования. Именно на них закладывается не только сам проект, но и большинство его ошибок. Чем раньше будет обнаружена ошибка, тем меньше времени в целом будет затрачено на проектирование. Один из вариантов раннего обнаружения заключается в применении средств и методов верификации на всех этапах проектирования. Этому направлению в отечественных изданиях и на страницах данного журнала уделяется явно мало внимания. Последняя обзорная публикация на эту тему в журнале была в конце 2004 года [1].

Ростислав ГРУШВИЦКИЙ
RIGrushvitsky@mail.eltech.ru
Максим МИХАЙЛОВ
yamaksya@yandex.ru

Общие вопросы верификации проектов

Как уже отмечалось в первой статье цикла [2], непрекращающееся усложнение создаваемых систем приводит к неравномерному распределению появления и удаления логических и других ошибок проектов по различным этапам проектирования. На рис. 1 приведены статистические данные о распределении количеств вносимых и устраненных ошибок на отдельных этапах проектирования. На рис. 1 видно, что при сохранении существующих методик проектирования до 70% ошибок возникает на начальных этапах проектирования и только на этапе отладки готового прибора удается устранить порядка 70% ранее внесенных ошибок. Естественно, столь позднее выяснение причин неработоспособности не способствует быстрому появлению разработок на рынке.

Значительные усилия прилагаются для изменения отмеченной тенденции. Однако старые представления проблемы верификации еще очень сильны. Даже терминология в области тестирования до настоящего времени еще окончательно не утвердилась. Существующие стандарты относятся к тестированию программного продукта (например, стандарт IEEE Std 829-1998 IEEE Standard for Software Test Documentation). И хотя большинство терминов для тестирования программного и аппаратного продукта близки или даже совпадают, трактовки различных авторов для одних и тех же понятий существенно отличаются. Как справедливо отмечает один из сотрудников фирмы Cadence, говоря о верификационной методологии: «Проблема верификации настолько сложна, что не может быть предметом обсуждения ни в 30-минутном докладе, ни на трех-четырех страницах журнальной статьи». В целом соглашаясь с ним, авторы настоящей серии статей не могут оставить без внимания проблемы верификации и будут пытаться изложить собственную точку зрения на эти проблемы. В современных изданиях только глоссарий по данной тематике занимает несколько страниц текста. Далее авторы пользуются термином «верификация», определяющим как саму процедуру проверки совпадения проекта и требований к нему, так и используемую при этом методологию. Мы предполагаем, что термины «верификационные подходы» и «верификационные средства» — более общие, они поглощают или включают частные варианты организации или целевого назначения (тестирование, валидацию, эмуляцию, моделирование, симуляцию, аттестацию и т. д.).

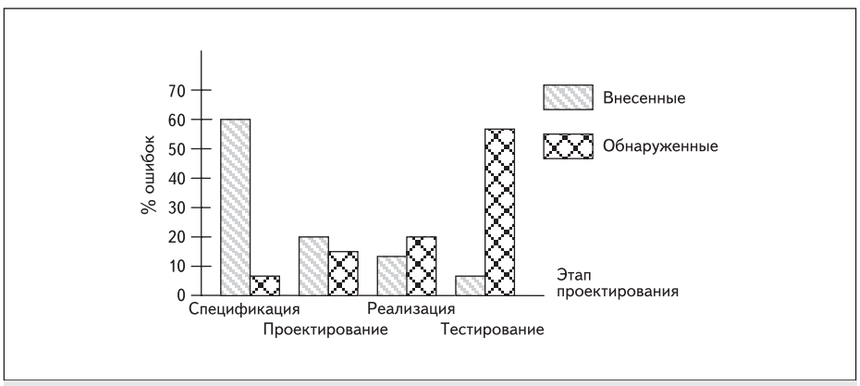

Рис. 1. Распределение вносимых и обнаруживаемых ошибок проектирования

Учитывая бытующую еще тенденцию, когда о проблемах верификации разработчики вспоминают, когда устройство уже готово, авторы серии статей начали в прошлом году именно с проблем отладки современных, сложных проектов и только теперь возвращаются к вопросам методологии верификации разработок аппаратуры на всех этапах проектирования.

Подходы к верификации проектов

Существуют различные методики и средства верификации. Они отличаются как эффективностью работы, так и требуемыми

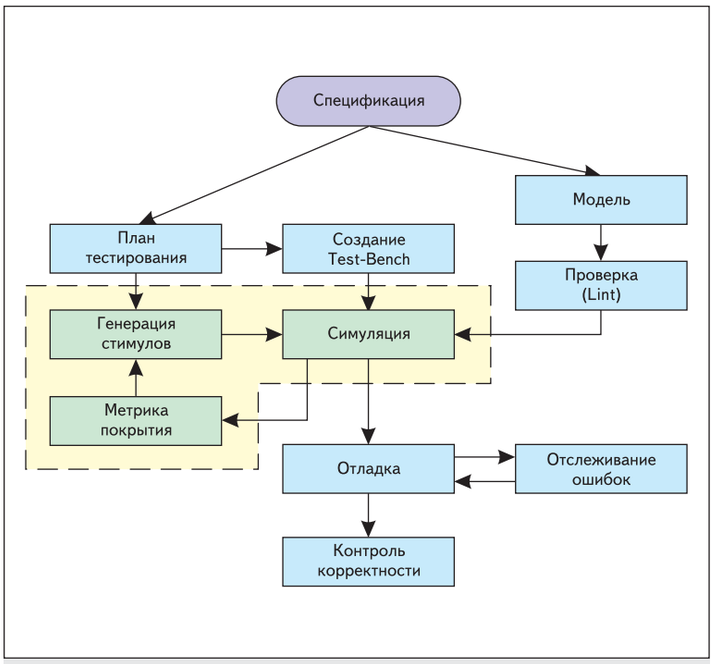

Рис. 2. Проектный поток для симуляционной верификации

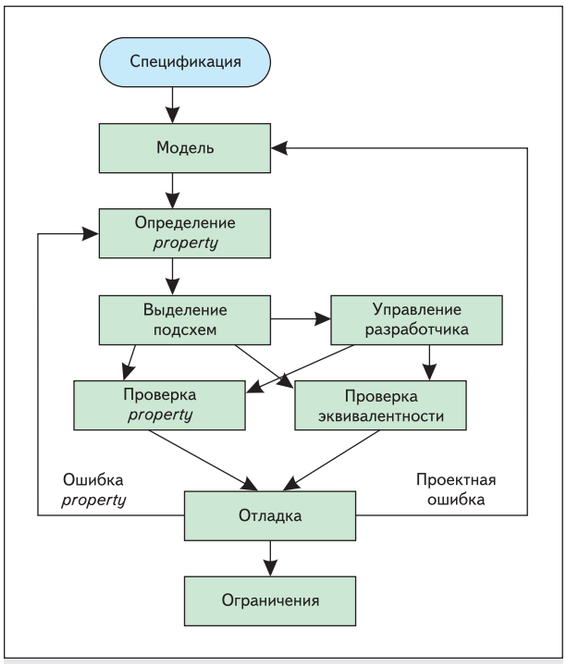

Рис. 3. Фрагмент проектного потока при формальных методах верификации

ресурсами (временными, интеллектуальными, финансовыми и т. д.). Первый вопрос, который возникает при рассмотрении верификации, связан с самостоятельностью данного направления в рамках общего процесса проектирования. В работе [3] H. Foster относит ее к самостоятельному направлению. В работе [4] P. Wilcox отводит даже целый параграф для подтверждения аналогичного утверждения. Если в предшествующие годы верификационная методика разрабатывалась после того, как проект был практически закончен (созданы схемы, написаны программы и т. д.), то в современных условиях такой подход оказывается неприемлемым. Окончательный проект может содержать логические ошибки или не соответствовать временным или ресурсным требованиям. Обнаружение этих фактов на последних этапах разработки обходится слишком дорого.

Очень важной для современных проектировщиков следует считать задачу систематизации различных вариантов верификации. Можно предложить три основных типа организации проверки соответствия проектов заданным требованиям.

Первый вариант (симуляционный) базируется на проведении серии (динамических) экспериментов над проверяемым проектом. На основании предварительно составленного плана тестирования формируется требуемое множество входных тестовых воздействий (стимулов). Эти воздействия подаются на входы тестируемого проекта. Модель системы после создания проходит этап проверки специальной программой (linter program, «программный доктор» и т. д.) на наличие статических синтаксических ошибок, «подозрительных» конструкций и отклонений от «хорошего» стиля. Одновременно создается требуемое тестовое окружение, которое используется для процедуры симуляции поведения модели системы при воздействии входных стимулов. В процессе отладки полученные отклики системы сверяются с предполагаемыми «эталонными». Проект может быть моделью будущего устройства или самим устройством. Физическое устройство в свою очередь может быть как окончательным (конечным) продуктом, так и прототипом будущей системы. Использование при тестировании модели (а не самого устройства) позволяет производить не только полное, но и частичное тестирование. В последнем случае проверке подвергается лишь часть свойств системы. Проектировщик игнорирует не существенные для данного эксперимента требования исходной спецификации. Проектный поток, соответствующий симуляционному подходу к верификации [5], приведен на рис. 2.

Существует большое количество типов моделей. Среди большого множества моделей особо выделяют два типа: функциональные и временные. В первом типе игнорируются временные характеристики проекта, во втором упор делается на получение оценок временных свойств проекта. Важнейшей характеристикой подхода является полнота охвата проекта тестированием. Методы оценки полноты охвата определяются метрикой покрытия исследуемого устройства тестами. Тестовые последовательности для моделей, саму процедуру тестирования и используемое для этого оборудование называют Test-Bench. Верификацию этого типа можно отнести к классу динамической верификации. Часто динамическую верификацию отождествляют с тестированием с помощью Test-Bench, хотя область применения методов Test-Bench несколько уже.

Второй вариант верификации базируется на формальных методах преобразования спецификации проекта в модель этого проекта. Такой подход можно определить как дедуктивный. Методы базируются на представлении основных характеристик (property) будущего устройства в терминологии некоторого формализма (в рамках специализированных алгебр, теории множеств, того или иного представления графов и т. д.). Далее осуществляется преобразование этого описания обычно автоматизированными средствами в модель системы, позволяющую, в конечном счете, практически реализовать систему. При создании модели применяются методы строгого математического доказательства соответствия обладания моделью запланированных свойств будущей системы. Проектировщики могут использовать алгоритмы, базирующиеся на представлении проекта в форме двоичной диаграммы решений (Binary Decision Diagram — BDD), на доказательстве булевской выполнимости (Boolean Satisfiability — SAT), автоматической генерации тестов (ATPG), символьной симуляции и т. д. При применении дедуктивных методов окончательную работоспособность модели напрямую определяет точность спецификации проекта или его частей.

Третий вариант сводится к доказательству эквивалентности моделей (Equivalence Checking of Models). Утверждение о работоспособности той или иной модели проекта основано

на эквивалентности с другой моделью. Как доказана правильность работы исходной модели при таком подходе, не имеет значения. Одним из вариантов проверки эквивалентной модели является ранее проведенное ее тестирование. В наибольшей степени метод разработан для доказательства эквивалентности арифметических цепей и для различных вариантов RTL-представлений. Например, доказывается эквивалентность списка соединений вентилей (gate netlist) и RTL-представления. Фрагмент проектного потока, базирующегося на формальных методах верификации, приведен на рис. 3. Фрагмент можно использовать как автономно (вместо симуляционного метода — фрагмент на рис. 2, выделенный пунктирными линиями), так и в дополнение к симуляционному подходу.

Последние два варианта относят к классу формальной верификации. Более подробное представление о методах формальной верификации можно получить, ознакомившись с работой [6]. Чаще всего формальная верификация использует статический вариант, но может быть и динамической.

На рис. 4 схематически изображены различные виды верификации. Один из эффективных приемов раннего и наиболее полного выявления ошибок — это использование методов верификации разработки на всех этапах проектирования. Теперь даже на одном и том же этапе редко используется только один подход. Верификационная методология старается объединить достоинства различных подходов.

Выбор того или иного метода верификации осуществляется с учетом различия затрат на верификацию и достигаемого результата. К затратам на верификацию можно отнести сложность ее организации, большую стоимость необходимого для этого оборудования, а также требуемое на ее проведение время. К достигаемому результату и полученному эффекту необходимо отнести, прежде всего, полноту проверки. С этих точек зрения и стоит сравнивать вышеперечисленные методы.

Методы формальной верификации позволяют получить практически полную проверку при сравнительно небольших временных затратах. Другое дело, что, как правило, каждая модель отражает лишь часть свойств будущей системы. Наиболее проработанным вариантом считается формальная верификация RTL-моделей. Анализ RTL-моделей, заключающийся в контроле стиля программирования на соответствие стандартам (Linting), используется уже достаточно давно. При более высоком уровне абстракции автоматизация методов формальной верификации требует привлечения сложных САПР, а само выполнение — высококвалифицированного персонала. Поэтому использование САПР, ориентированных на формальную верификацию, может позволить себе далеко не любая организация. Разработку и выпуск САПР, включающих в свой состав средства формальной верификации как дополнение к традиционным возможностям проверки правильности, сегодня смогли реализовать только такие фирмы, как Synopsys и Cadence. Органичное включение этих методов в повседневные традиционные маршруты проектирования могут позволить себе такие гиганты, как фирмы IBM, Motorola или HP. Большинство логических ошибок в таких сложных проектах, как, например, разработка процессоров, удается найти с помощью методов формальной верификации. Стоит отметить, что многие интересные решения (малогабаритные и доступные широкому кругу разработчиков), полученные сравнительно маленькими фирмами, после их поглощения крупными фирмами либо раство-

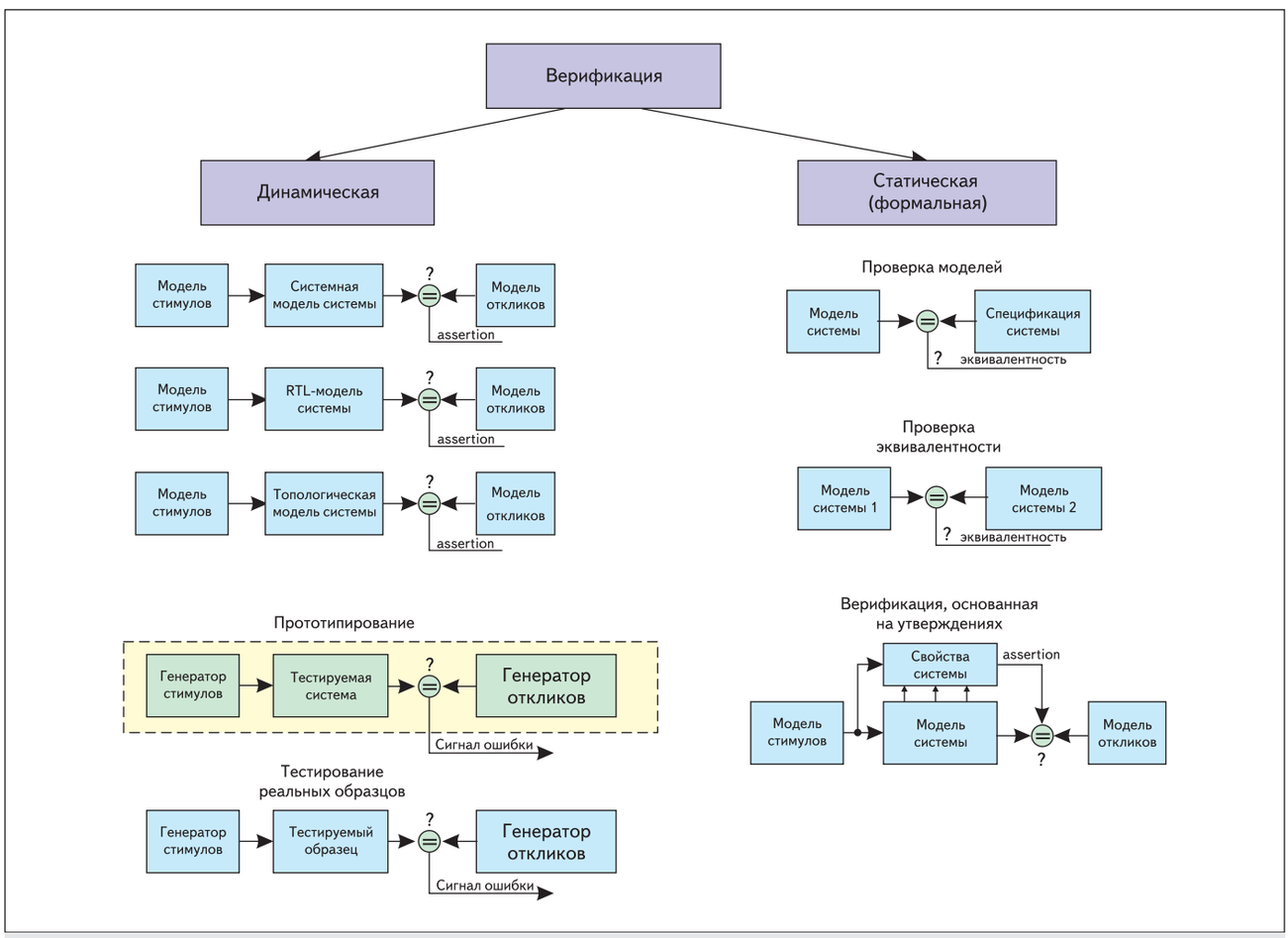

Рис. 4. Варианты верификации проектов и устройств

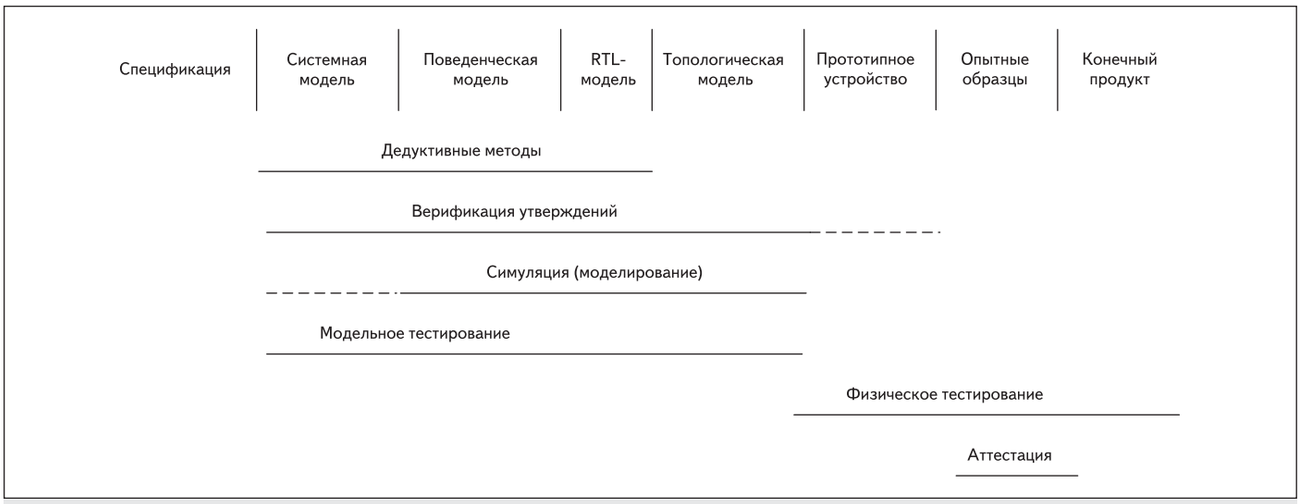

Рис. 5. Виды верификации на различных этапах проектирования

ряются в больших, сложных и дорогих САПР, либо «кладутся под сукно». Примером могут служить разработки фирмы O-In Design Automation в области верификации после лицензирования в 2004 году фирмой Cadence основных библиотек и приобретения самой O-In Design Automation фирмой Mentor Graphics.

Ведущие фирмы производители ПЛИС (такие как Xilinx и Altera) стараются включить в свои САПР стыковку со средствами формальной верификации других фирм (прежде всего Synopsys и Cadence). Средства формальной верификации применимы только к моделям будущих устройств, поэтому на конечных этапах разработки приходится привлекать методы динамической верификации для прототипных или опытных образцов.

Легко автоматизируются фрагменты проектов, имеющих регулярную структуру. С другой стороны, современные, даже не очень сложные (а соответственно и более дешевые) САПР позволяют автоматизировать не только и не столько создание Test-Bench, но и оценку полноты созданного теста. Широкое привлечение средств автоматизации для создания Test-Bench стало возможным после распространения текстовых средств описаний проектов. Языки описания аппаратуры, такие как Verilog, VHDL и SystemC, стали стандартами для редакторов САПР. Текстовое описание тестовой последовательности в программе Test-Bench облегчает автоматизацию не только процедуры создания этих программ, но и позволяет оценить полноту тестирования. Например, достаточно широко распространены у отечественных проектировщиков программные пакеты фирмы Mentor Graphics под названиями ModelSim и QuestaSim. Эти САПР графически отображают покрытие кодом (CodeCoverage) отдельных частей проекта.

Как уже отмечалось выше, динамическую верификацию можно применять на этапах проверки готового устройства и даже в процессе его эксплуатации. Целесообразность и возможность применения по отношению к одному проекту того или иного вида верификации существенно зависят от фазы проектного потока. Применение на различных этапах разработки как симуляционной, так и формальной верификации позволяет совместить достоинства обоих подходов. Наиболее распространенные виды верификации, применяемые на различных этапах проектировании, приведены на рис. 5.

Нужно четко различать, какой вид и когда следует (и выгоднее) использовать.

Динамическое тестирование позволяет проверять не только работоспособность моделей, но и конечное устройство. При этом для всех ситуаций можно применять одни и те же тестовые данные. Тесты, как правило, включают проверку функциональности, «крайние ситуации» и случайные входные данные. Следует различать динамическое тестирование, когда проверяемое устройство рассматривается как черный ящик, внутренность которого невидима проектировщику, и случай, когда разработчик может наблюдать за поведением внутренних фрагментов. Для удобства дальнейшего рассмотрения первый случай будем называть внешним динамическим тестированием, а второй — внутренним. Несколько проблем сопровождает возможности применения динамического тестирования.

Во-первых, к сожалению, даже достаточно простые проекты могут требовать для своей проверки чрезвычайно большого количества тестовых векторов, поэтому время симуляции ограничивает возможности такого подхода. Например [1], для простого 32-битного компаратора в общем случае требуется 264 тестовых вектора для исчерпывающей проверки. Поэтому сокращение избыточных тестовых векторов — это существенная и сложная задача. Тесты, имеющие сходное воздействие на тестируемый объект, объединяют в отдельные классы, называемые классами эквивалентности. Это позволяет уменьшить общее количество требуемых тестов. Однако приводимые обычно утверждения о полноте тестов при анализе только отдельных экземпляров классов эквивалентности очевидно не совсем корректны. Автоматизация создания тестов на настоящий момент — вопрос, еще требующий своего разрешения.

Вторая проблема, вызываемая внешним тестовым подходом, заключается в том, что некоторые внутренние ошибки невозможно наблюдать. Даже если внутренние ошибки искажают выходные значения, бывает очень сложно установить (проследить обратно), что именно послужило источником искажений. Наконец, определенные типы ошибок реализации очень трудно обнаружить с помощью функциональных тестов, что усугубляется недостаточностью управления (то есть невозможностью задать значения и отслеживать реакции) на внутренних цепях проекта. Следствием этого являются повторные изготовления чипов (стоимостью до миллиона долларов), если при верификации не были обнаружены существенные ошибки, а выявились они только на этапе тестирования или аттестации готового изделия. По данным Collet International Research, 70% всех проектов (изготовленных по технологии ASIC) потребовали неоднократных повторных изготовлений, поскольку содержали логические ошибки.

Методы тестирования моделей потому и получили широкое распространение, что позволяли разработчику при моделировании «проникать» внутрь проекта и там диагностировать дефект. Ранее достижение аналогичной гибкости при физическом тестировании устройств было затруднено. Положение несколько улучшилось после широкого внедрения (с 1980-х годов) в современные ИС методов граничного сканирования. К настоящему моменту, благодаря широкому внедрению

принципов внутрикристальной отладки (в частности, переноса идей граничного сканирования внутрь кристалла), уменьшилось влияние указанных выше проблем. Определенную информацию о современных методах отладки проектов можно получить из предыдущих выпусков цикла статей авторов в данном журнале [2].

Основное внимание разработчиков приковано к современным методам и средствам верификации электронной продукции. Меньше всего ошибок возникает в тех ситуациях, когда удается на данном этапе проектирования использовать средства вычислительной техники. К сожалению, далеко не все этапы проектирования поддаются автоматизации. В местах разрыва непрерывного процесса автоматизации сильнее всего проявляется человеческий фактор, и именно в этот момент в проекте возникают неточности, которые со временем могут создавать серьезные проблемы.

Почему проблемы верификации стали столь значимыми?

Вне зависимости от варианта реализации заказного или полузаказного проекта ИС возрастает цена ошибки. Сложность и стоимость проверяющего их оборудования зачастую превышает сложность тестируемого оборудования. Конечно, дороже всего обходятся ошибки при технологии ASIC, далее следует технология стандартных ячеек SC. Самые крупные проекты вообще не могут быть верифицированы за разумное время. При отладке процессора Pentium IV моделировалось 200 млрд циклов, для реального процессора это соответствовало всего двум минутам работы при тактовой частоте 1 ГГц [1]. В каждой статье упоминается тот факт, что 70% времени работы над проектом тратится именно на этап верификации [4].

Среди отечественных разработчиков наибольшее распространение получают приборы с программируемыми пользователем поведением, свойствами и характеристиками. Сюда относятся не столько микроконтроллерные и микропроцессорные системы с уже хорошо отработанной методологией проектирования и отладки, сколько СБИС сложной логики, объединяющие в себе и изменяемую архитектуру, и программируемые фрагменты (это ПЛИС, FPGA, CPLD и, конечно, SoPC). Ошибки при имплементации в FPGA или CPLD тоже не обходятся бесплатно. Возрастание стоимости ошибки в этом случае определяется чаще всего временными потерями, поскольку имплементация, учитывающая обнаруженные ошибки, обходится значительно дешевле, чем при других технологиях.

Причем чем раньше обнаружено расхождение между требуемым и созданным описанием, тем лучше: больше будет сэкономлено времени и сил. Поэтому желательно начать верифицировать проекты с самых верхних (абстрактных) уровней описания. При этом также желательно распределять планируемые проверки среди вычислительных средств. Вместе с тем, большинство проектировщиков на различных этапах проектирования прибегают к средствам натуральной языковой спецификации проектов и их отдельных свойств, и поэтому важно избежать неопределенностей (двойных трактовок) при их интерпретации для средств в форме фрагментов из элементов вычислительной техники. Адекватность трактовок описаний, составленных на натуральном языке, — важнейшая задача любого этапа проектирования. Так как языковая спецификация допускает множество интерпретаций, а формальная нет, то современным разработчикам целесообразно направлять свои усилия на формализацию описаний и спецификаций их проектов.

Средства верификации, применяемые разработчиками

Большинство разработчиков проектов небольшой и средней сложности используют в своей повседневной деятельности традиционный подход к верификации и типовой набор средств. На начальном этапе проектирования модели отдельных блоков либо приобретаются как IP предшествующих разработок, либо у сторонних фирм, а при необходимости разрабатываются самостоятельно. Практически во всех случаях осуществляется моделирование их работы. Верификационные средства, привлекаемые на этом этапе, определяются видом задания блока, а метод и средства тестирования чаще всего зависят от возможностей САПР, на которой разработчик собирается компоновать проект. На этапе сборки модели проекта создается набор тестовых последовательностей. Учитывая длины таких последовательностей и итерационность процедуры сборки, чаще всего применяют Test-Bench. На этапе отладки опытных образцов используются средства тестирования, специфика которых определяется свойствами проектируемой системы, тестовым инструментарием и опытом разработчика. При этом используемые ранее подходы (особенно методы динамического тестирования) практически одинаково хорошо подходили для всех этапов проектирования и для абсолютно разных проектов.

Для больших и сложных проектов наиболее «болезненным» является первый этап проектирования — системный. Добавление возможностей моделирования на уровне транзакции (применение исполняемых спецификаций) расширило набор верификационных средств. Но внедрение средств проектирования и тестирования на базе языка SystemC еще только началось. Стремление поддержать и этот уровень проектирования средствами вычислительной техники можно только приветствовать, но эти попытки пока, похоже, не дают желаемого перелома.

Многие современные средства и методы верификации явно расширили возможности проектировщиков всех уровней. Но сегодня целесообразность применения тех или иных методов верификации существенно зависит от выполняемой фазы проектного потока. Смена верификационных средств и методов при переходе с этапа на этап, естественно, не повышает эффективность тестирования. Сохранение определенного единства подхода при смене уровней проектирования и обеспечение более тесной стыковки проектов на различных этапах их разработки и верификационных средств возможно только при увеличении степени формализма у тех и у других и при органичном соединении ключевых свойств проекта с верификационными средствами.

Этому соединению соответствует современная тенденция распространения методов верификации, основанных на утверждениях, на большинство этапов проектирования: от разработки спецификации до выпуска готовой продукции.

Верификации, основанные на утверждениях

Этот подход требует относительно малых затрат, достаточно эффективен и поэтому бóльшая часть излагаемого далее материала посвящена детальному анализу именно этого подхода. Пара «утверждение» (*assertion*) и «свойство» (*property*) является краеугольным камнем многих современных методологий верификации. Метод подключения утверждений (о тех или иных свойствах проекта) к самому проекту находит применение не только при тестовом подходе, но и в методах формальной верификации. Способ проверки таких утверждений (симуляционный или аналитический) определяет их принадлежность к тому или иному верификационному подходу.

Когда проектировщик пользуется **assertion** как элементом верификационной методологии, то значительный процент проектных ошибок удается найти на достаточно ранних этапах проектирования. При этом самым важным является то, что они находятся с малыми затратами (в том числе временными). Например, при проверке функционирования процессора достаточно трудно проверить взаимное влияние различных групп команд друг на друга.

Благодаря применению **assertion**, по данным, приведенным в [6], из всех ошибок удалось найти:

- 34% при отладке проекта Alpha 21164 (1996);
- 17% при отладке проекта Cyrix M3(p1) (1998);
- 25% при отладке проекта DEC Alpha 21264 (1998);
- 25% при отладке проекта Cyrix M3(p2) (1999).

Благодаря применению **OVL assertion** из всех ошибок удалось найти 85% при отладке проекта HP (2001).

Верификация не возникла на пустом месте. «Проектные ловушки» для обнаружения тех или иных непредвиденных ситуаций в поведении проекта использовались всегда. Вид проектного задания на тот или иной блок определял и формальный вид ловушки. Примером (на условном языке описания аппаратуры) может служить конструкция:

```
ЕСЛИ FIFO_empty ='0'
        ТОГДА print(«FIFO not clear»)
КОНЕЦ ЕСЛИ;
```

Фрагмент позволяет при моделировании проверить, является ли в интересующем месте проекта буфер FIFO пустым.

Недостатком «проектных ловушек» являлась их уникальность. Каждый проектировщик разрабатывал собственный набор ловушек и, как правило, в новом проекте создавал новый набор.

Типичный пример пары утверждение/свойство — это вариант решения проблемы правильной передачи и хранения многобитовых данных (его с успехом продолжают использовать и сейчас). Традиционность применения не меняет сути. Речь идет о бите четности кода. Для самого проекта, чаще всего, бит четности не является информационным, он является *свойством*, присущим коду при передаче, обработке или хранении данных. Однако его можно использовать в специальных ловушках (утверждениях) для контроля получения запланированного результата. Путем добавления к передаваемому коду небольшой избыточности (бита четности или битов специальных типов кодирования) вероятность пропуска неверной кодовой комбинации сводится к минимальной и заранее определенной величине. Простота определения четности кода (процедура сложения по модулю два) привела к повсеместному распространению метода и его встраиванию и в модели, и в аппаратуру.

Поэтому, если для виртуального проекта объем затрат на верификацию не играет доминирующей роли (при реализации «в кремнии» они не превышают единиц процентов временных и аппаратных затрат), то можно ожидать, что удачные пары утверждений/свойств могут существовать не только в модели системы, но и останутся в конечном приборе.

В языке VHDL уже много лет с успехом используется оператор **assert**. Конструкция **assert** может содержать только булевское выражение, индикацией выполнения условия может быть только выводимое сообщение типа литеральной строки, а варианты реакции на выполнение условия весьма ограничены. Поэтому практическое использование этих конструкций в языке VHDL представляет собой задачу не менее, а зачастую и более сложную, чем реализация самого проекта. Для проектов со сложным поведением написание **assertions**, при использовании только стандартных возможностей Verilog или VHDL, является достаточно трудным. Существовавшие до сих пор технические средства (системы моделирования) практически позволяли применять конструкцию **assert** только при написании программ Test-Bench.

Расширение возможностей операторов **assert** при их стандартизации и строгая формализация правил их создания давно назрели. Утверждения призваны улучшить верификационную методологию и в результате должны обеспечить более быструю реализацию проектов. Современные реализации Assertion-Based Verification (ABV), а тем более методология, расширенная до понятия Assertion-Based Design (ABD), являются более строго формализованными, структурированными приложениями утверждений, хорошо приспособленными и ориентированными на задачи верификации. Assertions позволяют легко контролировать корректное поведение внутренних сигналов тестируемого устройства. Их можно вставлять в модули Verilog программ или в архитектурные пары entity-architecture VHDL-описаний, аналогично можно использовать контрольные модули (checkers) из библиотеки — Open Verification library (OVL) фирмы Accellera. Современные языки описания аппаратуры (SystemVerilog) уже включают утверждения в свой синтаксис. Следует ожидать, что в будущем любой язык описания аппаратуры будет содержать средства описания утверждений. Наибольшего успеха при ориентации на средства проектирования, ориентированные на утверждения, можно получить только при полном (охватывающем все этапы проектирования) процессе. Сейчас уже многие средства моделирования поддерживают Assertion Based Design, чего, к сожалению, нельзя сказать о средствах синтеза и имплементации.

Выбор свойств и утверждений для проектов

Идеальным вариантом ABD являлось бы добавление к проекту (в его виртуальном и реализованном вариантах) средств, проверяющих правильную работу проекта.

Как и в приведенном выше примере с кодированием посылки, основная сложность Assertion Based Design — как раз в правильном выборе верифицируемых свойств проекта (узко специфичных, но точно выраженных). Выполняемые в проекте специфические функции, как правило, наиболее точно описываются на уровне регистровых передач (RTL). Традиционные средства RTL фиксируют поведение проекта шаг за шагом (цикл за циклом) и делают это (включая реакции на ошибки) излишне пунктуально для анализа поведения проекта на более высоком уровне. Задача **assertion** — описание поведения во времени контролируемой системы в краткой, формальной и строгой форме, при необходимости «утверждения» должны выполнять действия, позволяющие зафиксировать сложное проектное поведение в лаконичной форме. Утверждения должны описывать как правильное (разрешенное) поведение, так и неправильное (запрещенное). И, наконец, **assertions** должны легко создаваться проектировщиком, просто анализироваться инженером, ответственным за верификацию, и приниматься к реализации САПР.

Очевидно, что удовлетворение всех перечисленных свойств — задача не тривиальная. При разработке утверждений их набор должен обеспечить всесторонность контроля. **Assertions** должны связываться с системой на различных уровнях ее рассмотрения, тем самым они могут выполнять так называемую *повторную верификацию* (верификацию на любом из уровней). Хотя в основе **property** лежат обыкновенные булевские выражения, ABV должны добавлять средства, которые позволят сжать временные взаимоотношения между этими выражениями. Они открывают широчайшие перспективы для описания интенсивного управления проектным поведением, конвейерами, защелками и т. д.

Профессиональная верификация

Как уже неоднократно отмечалось, современное проектирование любых типов ИС отличает возрастание сложности проектов под постоянным прессингом сроков завершения. Проекты имеют сложную иерархическую структуру, содержат большое число файлов, много объектов-модулей и сигналов, разные блоки используют отличающиеся языки описания, разработку ведет несколько человек и т. д. Обеспечение качественной верификации при этом становится сложной задачей. Как решить данную проблему?

Просто увеличить количество верификационных процедур и разнообразить сам набор верификационных средств? По мнению большинства исследователей и руководителей больших проектов, простое увеличение нагрузки на разработчиков путем добавления к их обязанностям обеспечение полноты верификационных процедур не дает положительного результата. Разработчик зачастую закладывает в тестирующее средство те же решения, которые он уже реализовал в самом проекте. Поэтому предлагается новая верификационная методология (иногда называемая Advanced Verification), в соответствии с которой процесс верификации должен не зависеть от проектного потока, начинаться одновременно с самим проектированием и осуществляться параллельно с ним от начала до окончания разработки. Более того, поскольку действия требуют большого профессионального опыта и организации, то следует к про-

ектному потоку подключать отдельных специалистов: инженеров-проектировщиков и инженеров-тестировщиков. Подобное разделение функций уже достаточно давно используется при разработке программного обеспечения, теперь это осознали и руководители разработок аппаратных средств. Параллелизм выполнения работ разработчика и инженера по верификации аппаратуры должен обязательно совмещаться с разделением функциональных обязанностей этих специалистов. Реально и тот, и другой занимаются верификацией, правда, с разным подходом к включению этих средств.

Инженер-проектировщик должен овладеть ABD и добавлять в текст проекта утверждения, прежде всего для того, чтобы сократить время, необходимое для завершения проекта; кроме того, он уменьшит число прерываний и вопросов со стороны инженера-тестировщика о цели проекта и неправильных трактовок принятых решений. Если утверждения включены в текст, основные расхождения с тестировщиком будут касаться действительно проектных проблем, и обратная связь от обсуждения может оказать существенную помощь для обнаружения проектных дефектов. Проектный инженер, пренебрегающий использованием системы утверждений в исходных текстах, потратит в результате больше времени на обсуждение с тестировщиком вопросов, решение которых позволило бы провести тестирование (функциональность проекта, интерфейсные требования и т. д.).

Инженер-тестировщик также должен владеть техникой утверждений. Специалист по верификации должен помочь проектному инженеру в освоении техники внедрения утверждений в проектный код. Инженер-тестировщик — именно тот, кто может способствовать пониманию проектировщиками преимуществ использования ABD.

Различие конечных результатов и требований к инженерам двух рассматриваемых групп показывает настоятельную необходимость резкого увеличения числа специалистов, занимающихся верификацией. Профессиональная функциональная верификация требует специфических знаний, обучения и тренировки. К сожалению, в отличие от зарубежных стран, в отечественной практике пока не видно движения в этом направлении.

Заключение

Как уже отмечалось, наиболее важным моментом для Assertion Based Design является правильный выбор свойств проекта, которые предполагается контролировать. Несмотря на короткий срок развития дисциплины, уже возникли разные направления и стандарты (в данном контексте имеются в виду языки и библиотеки).

В дальнейшем предполагается произвести обзор языковых средств описания утверждений/свойств на основе следующей классификации [7]:

- Специальные языки — в этом случае используются формальные языки, которые разработаны специально для максимально эффективного описания утверждений/свойств. Языки этого типа, к которым относятся Sugar, PSL (Property Specification Language) и OVA (OpenVera Assertions), отличаются широкими возможностями при создании сложных регулярных и временных выражений и с помощью очень компактного кода позволяют описывать сложные поведенческие алгоритмы.
- Модели, написанные на HDL и вызываемые из HDL. Эти модели представляют собой утверждения/свойства, использующие стандартные выражения языка HDL, и могут быть представлены в устройстве как любые другие блоки. Однако эти блоки будут свернуты программами синтеза вида «Вкл./Выкл.», чтобы быть уверенным, что они не будут реализованы физически. Хорошим примером этого метода может служить открытая библиотека средств проверки (OVL), разработанная компанией Accellera.
- Специальные операторы языка HDL. Изначально язык VHDL поддерживал простые операторы контроля, которые проверяли значения булевых выражений и отображали определенные пользователем текстовые строки, если выражение принимало значение false (ложь). Изначально Verilog не поддерживал операторов проверки, но его последующая реализация — SystemVerilog — была дополнена такой возможностью.

Каждый из вариантов имеет свои выгоды (и ограничения), различные формы утверждений и применяемые в них модели, поскольку несколько отличаются принципы создания их фундаментов.

Продолжение следует

Литература

1. Долинский М. Assertion Based Verification — верификация, основанная на утверждениях // Компоненты и технологии. 2004. № 9.
2. Грушвицкий Р., Михайлов М. Проектирование в условиях временных ограничений: отладка проектов // Компоненты и технологии. 2007. № 6.
3. Foster H., Krolnic A., Lacey D. Assertion-Based Design. Kluwer Academic Publishers, 2003.
4. Wilcox P. Professional Verification. A Guide to Advanced Functional Verification. Kluwer Academic Publishers, 2004.
5. Lam W. K. Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall PTR, 2005.
6. Drechsler R. Advanced Formal Verification. Kluwer Academic Publishers, 2004.
7. Максфилд К. Проектирование на ПЛИС. Архитектура, средства и методы. Курс молодого бойца. М.: Додека-XXI, 2007.