УДК 004.942

Верификация поведения цифровых устройств с помощью моделей высокого уровня

А.С. Щербаков

ЗАО “Интел А/О”, andreas.s.scherbakov@intel.com

Аннотация **— Рассматриваются проблемы доказательства соответствия поведения электронных логических блоков заданной абстрактой модели высокого уровня (МВУ). Рассматриваются требования к языкам, применяемым для описания таких моделей на примере языка «Murphi». Указаны основные сложности, возникающие при формальной верификации схем относительно МВУ. Приведен обзор метода, использующего детализацию (refinement) на основе описания событий.**

Ключевые слова **— модели высокого уровня, логические схемы, Murphi, формальная верификация.**

I. Проблемы повторного использования блоков

Значительная часть современных больших электронных схем, таких как процессоры, обычно состоит из набора блоков высокой сложности, которые, как правило, в целом повторяют решения, использованные ранее (в предыдущих версиях проекта или в схожих по функциональности блоках других изделий). Заметим, что проектирование и отладка таких блоков “с нуля” в условиях динамичного конкурентного рынка не представляется возможной. В то же время в каждую последующую версию необходимо добавлять новые возможности, а также оптимизировать ее характеристики для конкретной архитектуры. Хотя усилия, которые затрачивают разработчики на внесение подобных изменений, могут быть невелики для относительно небольших доработок, этого нельзя сказать об усилиях, затрачиваемых на верификацию (валидацию) функциональной корректности получившейся схемы. Такие затраты могут быть весьма существенны, поскольку область влияния изменений значительно превосходит саму область изменений, и в результате зачастую затрагивает существенную часть всей схемы блока, что требует ее повторной верификации. Для такой верификации составляются вспомогательные спецификации с утверждениями, описывающими поведение схемы. Даже если такие спецификации для предыдущей версии и использующие их средства доступны при проверке модифицированного блока, затраты времени на редактирование вспомогательных файлов могут быть зачастую сравнимы с затратами на их создание. Причина в том, что такие файлы для верификации (валидации) пишутся инженерами “поверх” уже разработанной логической схемы, оперируют с конкретными сигналами (как внешними, так и внутренними), временными сдвигами, наборами операций и т. п., что делает их существенно зависимыми от реализации, даже если авторы стремились этого избегать. Такие спецификации могут подразумевать в качестве необходимых такие особенности поведения схем, которые нигде не выражены в формальном виде, а зачастую вообще не документированы. При дальнейших изменениях эти предположения могут оказаться неверными, что, разумеется, требует внимательного пересмотра всех спецификациий, используемых при верификации блока.

Понятно, что подобная проблема возникает и при проектировании новых узлов, с той лишь разницей, что большие затраты усилий на верификацию в этом случае кажутся «естественными». Тем не менее, даже в этом случае зависимость спецификаций для проверки функциональности от деталей реализации не позволяет создать списки утверждений (assertions) для проверки блока до окончания разработки его реализации, что отодвигает сроки начала верификации и удлиняет весь цикл проекта. Создание независимых моделей поведения для блоков позволило бы осуществлять раннюю верификацию как первоначальных версий, так и, в еще большей степени, всех последующих [1].

II. Языки представления МВУ

Теоретически, любой язык, понимаемый машиной, мог бы служить для описания моделей высокого уровня (МВУ). Как правило, так и происходит – используются имеющиеся языки, созданные для других применений, например процедурные (как C) или ориентированные на описание логических схем (VHDL, Verilog). Поскольку такие языки могут (в значительной мере неявно) передавать представление об абстрагированном поведении устройства, оперируя с переменными, действиями, событиями и т. п., они на практике могут быть использованы для

абстрактного описания функциональности. Однако, опыт их использования для высокоуровнего моделирования выявил ряд недостатков. Можно сказать, что традиционный язык, с одной стороны, слишком богат для абстрактного описания, а с другой – слишком беден. Первое выражается в том, что развитый язык, созданный для описания реализаций, изобилует разнообразными конструкциями, описывающими детали этих реализаций, что для абстрактного описания несущественно. Подобное многообразие возможностей и видимое сходство инструментария с используемым для описания конкретных реализаций устройств и программ приводит к соблазну прямой привязки декларируемых свойств к имеющемуся низкоуровневому представлению вместо описания существенных абстрагированных понятий. К примеру, интерфейс модуля может содержать оределенный набор сигналов, их компоновку в шины, службные и тактовые входы и прочее. Все это, с точки зрения абстрактого описания поведения, – совершенно излишняя информация, так как все перечисленное не является ее неотъемлемыми атрибутами в высокоуровневом представлении. Язык не дает возможности отделить абстрактное представление от конкретного. Такая модель часто содержит комментарии, объясняющие общий смысл содержащихся в ней высказываний, но эта информация никак не может быть использована в автоматизированных приложениях.

Следует сказать, что, поскольку полноценное создание и использование машиночитаемых МВУ в результате затруднено, они, как правило, не создаются вообще или создаются по мере надобности, имея при этом локальную область применения, ограниченную рамками текущей активности группы инженеров. За основу же при первоначальном проектировании применяются неформализованные текстовые документы. Такая ситуация приводит к увеличению затрат труда и росту числа ошибок.

Идеальная форма модели, оставаясь абстрактной, сочетала бы в себе свойства доступного для понимания документа и строгой формальной спецификации: наглядность алгоритмов и процессов; первичность по отношению к реализации в виде электроной схемы; роль эталонного описания в автоматизированных процессах верификации.

Чтобы обеспечить создание таких высокоуровневых моделей и их стандартизацию, необходимы специальные языки представления.

Еще одной их важной особенностью должно являться выделение в основную структурную единицу элементов функциональности. Этим они должны отличаться от традиционных процедурных языков и языков описания логических электронных схем (RTL), где структура основана на выделении упорядоченных во времени операций, блоков данных или компонентов и узлов схемы. Элемент функциональности (пожалуй, наиболее точно соотетствующий прижившемуся английскому слову “feature”) не обязательно имеет непосредственную проекцию на фрагмент логики или отрезок времени и может, в принципе, произвольным образом разделять ресурсы с другими элементами функциональности. Такой подход к абстракции модели, конечно, затрудняет ее детализацию (что, как показано далее, возможно компенсировать применением вспомогательных описаний без потери абстрактности), но дает возможность достаточно естественного ее построения и наращивания.

III. Асинхронность

Одним из наиболее важных (и создающих существенные сложности при верификации) свойств абстрактных моделей высокого уровня является отсутствие в них определенной привязки выполнения каких-либо действий к моментам времени (синхронизации). Описывая многообразную функциональность, мы в то же время ничего не говорим о том, когда именно происходит то или иное событие. Таким образом, модель является асинхронной.

В такой модели отсутствует не только привязка выполнения действий к временным циклам реализующей ее логики, но и заданная упорядоченность выполнения множественных действий [2]. Это можно сравнить с многопоточным выполнением программы без явно заданных семафоров. Организация необходимого разделения ресурсов сознательно выносится за рамки абстрактной модели. Целью такого подхода является создание базовых функциональных моделей, не зависящих от реализации.

Чтобы понять, почему синхронизацию целесообразно полностью вынести из МВУ, а не, например, параметризовать задержки во времени, заметим, что проявления асинхронности могут быть весьма разнообразны. Сюда надо включить как «статическую» неопределенность, порождаемую различными версиями реализации, так и динамическую, связанную со сложным алгоритмом исполнения разнообразных команд в арифметико-логических блоках, а также такие влияющие на порядок выполнеия решения, как динамическое распараллеливание, конвейеризация команд. Перечислить все подобные проявления, пожалуй, невозможно, поэтому необходимо особое внимание к обеспечению покрытия всех возможных асинхронных сценариев средствами верификации [3].

IV. Недетерминистичность МВУ

Вообще говоря, недетерминистичность является наиболее универсальным свойством МВУ, позволяющим им соответствовать многообразным конкретным реализациям. Асинхронность является одним из классов недетерминизма – неопределенностью моментов времени, в который произойдет транзакция. Хотя этот класс представляет наибольший практический интерес и наибольшую сложность в реализации средств верификации, можно выделить и другие классы недетерминистического поведения, например, не совсем полное определение функции в МВУ. Это может быть и наличиие каких-то служебных дополнений к основной функциональности, и неопределенность базиса для последней. Простейший пример второго: пусть известно, что некий блок осуществляет сложение двух чисел с переполнением, но (в рамках абстрактой модели) неизвестно, какой длины эти числа. В этом случае логика формирования сигнала переполнения не может быть заранее “привязана” в МВУ к какому-то заданному номеру бита результата, т. к. неизвестно, какой бит окажется старшим. Если при реализации традиционных арифметико-логических устройств задать соответствующий параметр несложно, то, например, в декодерах для обработки аудио- и видеопотока, где числовые представления максимально сжаты и могут иметь статически непредсказуемую переменную разрядность, необходим механизм сопоставления абстрактной и конкретной модели даже для простых арифметических действий.

V. Проблема кванторов

Принципиальная дополнительная трудность, возникающая при переходе от верификации логических схем относительно детальных утверждений об их поведении к верификации относительно МВУ, состоит в появлении универсального квантора в выражениях для поиска контрпримеров. В «традиционной» верификации мы стремимся найти контрпример, показывающий, что при каких-либо значениях во времени на входах (более корректно, на свободных переменных) заявленные свойства для выходных сигналов не выполняются. Слова «для каких-либо» соответствуют экзистенциальному квантору, который будет входить в систему уравнений, результат решения которой будет составлять контрпример, демонстрирующий несоблюдение заявленных свойств, а отсутствие решения – корректность работы схемы относительно данных свойств. (Отметим, что это верно для LTL темпоральной логики, которая, как правило, применяется при задании свойств).

Теперь предположим, что мы используем абстрактную МВУ вместо детального описания свойств. Чтобы найти контрпример для соответствия МВУ конкретной реализации, надо показать, что при какой-то допустимой детализации МВУ (или абстракции конкретной модели), обеспечивающей сравнимость наших двух моделей, и при каких-либо значениях сигналов на входах контролируемые состояния или переходы моделей не соответствуют друг другу. Здесь мы видим уже два квантора: универсальный – “при всех допустимых детализациях” и экзистенциальный – “при каких либо значениях на входах”, которые будут входить в систему уравнений для поиска контрпримера.

Заметим, что возможности недетерминистической детализации содержатся как в сопоставлении состояний и переходов двух моделей, так и в неопределенности выбора пути внутри самой МВУ (как показано ниже на примере языка Murphi).

На практике решение системы логических уравнений с квантором одного вида (экзистенциальным) выполнить сравнительно легко, так как для этого существует хорошо развитый инструментарий SAT/SMT приложений. Для разнородных же кванторов эффективные решения практически отсутствуют, что является наиболее серьезной сложностью, стоящей на пути практической автоматизации сравнения МВУ с логической схемой. Эксперименты, проведенные автором, показывают, что прямое перечисление всех вариантов недетерминистической детализации в создаваемых для поиска контрпримера системах уравнений не приводит к неприемлемому росту времени нахождения решения (при несущественных для общности ограничениях), что вполне позволяет работать с разнородными кванторами.

VI. Язык Murphi

Язык Murphi был разработан в 1992 г. в университете Стэнфорда и был предназначен для верификации сложных протоколов ввода-вывода [4], например, для доказательсва когерентности кэш-памяти. Этот лаконичный недетерминистический язык, основанный на циклически исполняемых множествах конкурирующих пар “правило – условие”, выглядит весьма подходящим для описания МВУ. Хотя поддержка языка прекратилась, модели с его использованием для исследовательских целей продолжают создаваться [5]. Продуктизация таких приложений и их использование в практике проектирования в перспективе может привести к

включению в эту практику и языка Murphi как формы представление МВУ. Язык Murphi, в силу своей выразительности и универсальности, может стать удобным стандартом представления.

Рассмотрим вкратце основные особенности Murphi:

1. Язык описывает абстрактную машину состояний.
2. Содержит принципиальный нон-детерминизм переходов: условия, при которых переход может осуществиться заданы, но ничего не говорится о выборе из нескольких возможных переходов.
3. Компактен: множество возможных переходов в машине состояний зависит только от ее состояния.
4. Асинхронен: временные границы циклов и последоватьльность совершения конкурирующих переходов никак не задаются.

Язык Murhi содержит:

1. Типы: диапазоны, булевские, скаляры, массивы, структуры, объединения.
2. Присваиваемые переменные.
3. Правила переходов с необходимыми условиями и присвоением значений произвольному набору переменных.
4. Параметризацию однородных множеств правил свободными переменными.
5. Инвариантные свойства для верификации.
6. Стандартные арифметические и логические операции.
7. Дополнительные конструкции для удобства манипулирования данными, в т. ч. функции, переименования, неопределенные состояния и др.

Murphi **не** содержит:

1. Интерфейса: входов, выходов.
2. Операторов, выполняемых безусловно (исключение – стартовая инициализация переменных).
3. Синхросигналов, сигналов сброса и т. п.
4. Иерархии модулей.

VII. Детализация модели

Особенности сравнеия МВУ с подробной моделью логики существенно снижают возможности традиционной верификации (валидации). Рассмотрим верификацию некоторой модели **M** относительно спецификации **S**. Фактически, модель, как и спецификация, задает некоторое множество допустимых поведений в некотором базисе. Если обозначим эти множества теми же символами **M** и **S**, то задача верификации состоит в доказательстве того, что пересечение **M** с дополнением **S** не пустое:

$$\mathbf{C} = \mathbf{M} \setminus \mathbf{S} \neq \emptyset \qquad (1)$$

В этом случае любой член множества **C** является контрпримером. В обычном случае верификации спецификация **S** задана в том же базисе, что и **M**, т.к. утвержения для **S** выписаны на основе уже имеющейся модели реализации. В этом случае нахождение хотя бы одного члена **C**, в принципе, несложно. В случае, если **S** – это МВУ, мы сталкиваемся с множествами, заданными в принципиально неэквивалентных базисах. Прямое рассмотрение пересечения таких множеств привело бы к сложности порядка произведения их объемов, что на практике означало бы невозможность решения задачи. Заметим, что недетерминизм взаимных преобразований базисов при этом должен быть учтен дополнительно.

В качестве простого подхода базис для **M** может быть выражен через базис для **S**. Тогда мы имеем детализацию (*refinement*) абстрактной модели. Также, наоборот, можно использовать базис абстрактной модели, тогда говорят об абстракции множества **M**. Фактор усложнения задачи в этом случае определяется сложностью взаимозависимости базисов.

На практике часто применяется способ абстракции, связанный с непосредственной привязкой активации переходов в МВУ с соответствующими транзакциями в конкретной логике [6]. При этом требуется предварительное детальное сопоставление моделей в ручном режиме, а полученный результат не может быть безопасно использован без пересмотра для новых версий. Кроме того, прямое сопоставление переходов накладывает ограничение на абстрактность МВУ.

В данной работе предлагается использовать единый исходный базис для **M** и **S**, основанный на абстрактных событиях. При этом подразумевается, что, хотя каждое из этих абстрактых событий может не иметь прямых проекций на операции в МВУ или RTL, существенные для проверки свойств элементы поведения как в МВУ, так и в RTL могут отождествляться с выполнением соответствующих последовательностей абстрактных событий. При этом RTL в простейшем случае может быть инструментирован вызовами процедур для обозначения границ интересующих элементов поведения (чтобы избежать привязки к специфическим сигналам устройства). В МВУ такими элементами являются правила и их последовательности.

Такой подход (рис.1) помогает значительно уменьшить сложность поведения абстрактных и конкретных моделей, выраженного в едином базисе. В результате верификация модели может начаться значительно раньше, что сократило бы общее время проектирования.

Заметим, что в общем случае, заданной последовательности переходов в RTL может

соответствовать произвольное число последовательностей событий. Впрочем, на практике возможно сравнительно простое ручное назначение правильного выбора варианта, поэтому детальное исследование его автоматизации оставлено на будущее.

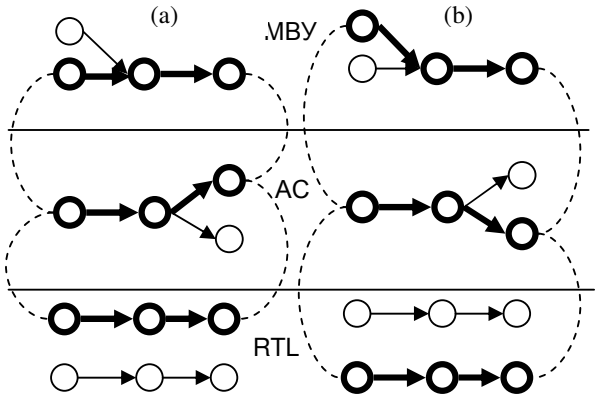

Рис. 1. Пример: Две возможных подпоследовательности из сценария, записанного в абстрактных событиях (АС), ставятся в соответствие как последовательностям правил в МВУ, так и последовательностям транзакций в логической схеме (RTL). Соответствия находятся на основании шаблонов, заданных в дополнительных файлах "сценариев".

Так, опосредованно, происходит связывание последовательностей событий в абстрактной и конкретной моделях. Пунктирные дуги обозначают одновременность начала/окончания последовательностей; стрелки – возможности перехода; кружки – правила МВУ, абстрактные события (АС), группы операторов в RTL, соответсвенно. Жирным выделены активные в данном найденном сопоставлении цепочки состояний и переходов из числа допускаемых моделями. Заметим, что сопоставления (а) и (б) могут иметь место как в различные моменты времени, так и одновременно (несмотря на то, что им соответствуют общие переходы и состояния МВУ и последовательностей АС).

VIII. Проведенные эксперименты

В данной работе изучались возможные подходы к решению проблемы формальной верификации относительно модели высокого уровня на примере блока контекстно-зависимого декодирования видеопроцессора. В качестве МВУ использовалась специально созданное на основе документации описание на Murphi. Ряд деталей при этом искусственно абстрагирован. Целью исследования было нахождение эффективных способов проверки взаимного соответствия состояний двух моделей одного устройства, различающихся уронем абстракции (МВУ против детализировнного описания на языке System Verilog). Приведение базиса и комбинирование моделей осуществлялось вручную.

При этом изучались как тесты на соответствие текущих состояний двух моделей в повторяющиеся моменты времени, так и тесты на совместное достижение моделями эквивалентных конечных состояний в будущем.

Показано, что для верификации на постоянство соответствия могут использоваться существующие средства верификации без серьезной потери производительности. Проверка на достижение тождественных целевых состояний, напротив, весьма ресурсоемка, так что для практически интересных многоступенчатых процессов (например, инициализации блоков памяти) верификация не достигает результата за разумное время. Это позволяет сделать вывод о предпочтительности проверки эквивалентности по сравнению с проверкой отдаленных по числу циклов целевых свойств, несмотря на то, что теоретическая сложность задачи во втором случае меньше. Это объясняется тем, что задача в первом случае разбивается на последовательность «коротких» подзадач и эвристики современных SAT solvers способны их выделять.

Исследовались возможности учета асинхронности между моделями без знания относительного расположения их фаз. Создан механизм для комбинирования моделей с недетерминированным (те описываемым переменной во времени с неизвестным заранее значением) стробированием фаз каждой модели относительно "единого" синхросигнала (реализующий "скачущую" (jittering) синхронизацию с применением обработки промежуточного побитового представления логики). В качестве вариантов возможно как представление асинхронности в виде свободных переменных, так и рассмотрение всех допустимых сценариев асинхронности. Установлено, что при ограничении (сверху и снизу) соотношения числа тактов синхронизации, происходящих в абстракной и конкретной модели вида 1:**N**..**N**:1. где **N** – числа в несколько единиц (до 10), эти подходы примерно эквивалентны по производительности, с небольшим преимуществом первого. При этом комбинированная модель для верификации дополняется вспомогательными линиями задержек и матрицей компараторов размера **N**×**N**, где **N** – рассматриваемый максимально допустимый разброс циклов синхонизации двух моделей между точками эквивалентности их состояний (рис. 2). Следует отметить, что полученный результат отличается от ожидаемого из опыта обычной формальной верификации, где введение новых свободных переменных в большинстве случаев намного менее предпочтительно, чем конъюнкция альтернатив. По-

видимому, в нашем случае сильно связанные уравнения, порождаемые матрицей компараторов, существенно снижают эффективность эвристических подходов решающих их SAT-приложений (что может являться интересным полем для исследования).

Также исследовалась сложная микроархитектурная реализация динамического распределения ресурсов. Была использована имеющаяся независимо созданная для других экспериментов абстрактная модель на Murphi (~2200 строк). Создан прототип транслятора из Murphi в System Verilog, полностью поддерживающий конструкции языка и имеющий гибкий механизм настройки. Модель переведена в синтезируемую форму и успешно верифицирована. Механизм детализации на основе последовательностей абстрактых событий в настоящее время интегрируется в экспериментальное решение.

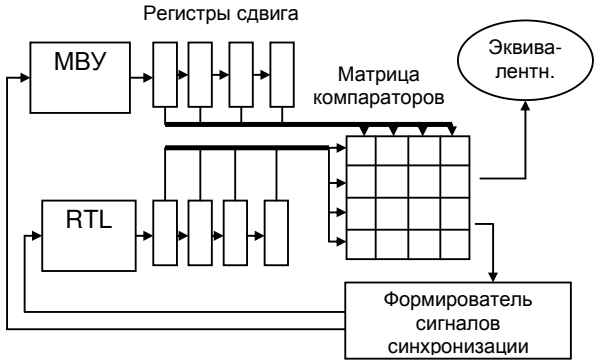

Рис. 2. Асинхронное сравнение моделей с ограниченным сдвигом циклов с помощью дополнительной логики, реализующей матрицу сравнений последовательных состояний и формирование коррекции циклов

IX. Заключение

Выделение описаний абстрактного поведения в отдельный формат и использование единого невисимого базиса «элементарных событий» (не связанных непосредственно с характерными границами транзакций одной из моделей, а выбираемыми из удобства описания) для спецификаций взаимного соответствия событий, происходящих в абстракной и конкретной моделях, является перспективной отправной точкой для построения эффективных приложений верификации поведения моделей-реализаций относительно абстрактных моделей. Основой для разработки соответствующих приложений (во всяком случае, на начальных стадиях) вполне могут служить имеющиеся средства верификации, на вход которым могут подаваться сгенерированные сравнительно несложными алгоритмами комбинированные модели, включающие проверяемый логический блок и абстрактную модель. Однако, при этом необходим учет специфических особенностей производительности приложений, применяемых для решения получающихся систем уравнений.

Применение специализированных языков описания абстрактных моделей (например, Murphi) повышает универсальность модели и упрощает создание вспомогательных спецификаций для проверки ее реализаций. На примере прототипа компилятора языка Murphi, созданного в рамках настоящего исследования, показано, как специализированные средства трансляции способны преобразовывать абстрактные модели к виду, сравнимому с моделями реализации.

Для обеспечения такого сравнения рекомендуется представление последовательностей переходов как в абстрактной, так и в конкретной модели в терминах обобщенных событий в едином базисе.

Тестирование предложенного подхода на реальном примере микроархитектурной реализации распределения ресурсов показало достаточную для практических применений производительность при использовании стандартных средств верификации. Разработка специализированных приложений обеспечит существенное ускорение процесса верификации.

Литература

[1] Mahajan, Yogesh; Chan, Carven; Bayazit, Ali; Malik, Sharad; Qin, Wei. Verification Driven Formal Architecture and Microarchitecture Modeling // Formal Methods and Models for Codesign, 2007. MEMOCODE 2007. 5th IEEE/ACM International Conference. May 30 - June 2, 2007. - P. 123–132.

[2] Nicolas Halbwachs, Louis Mandel. Simulation and Verification of Asynchronous Systems by means of a Synchronous Model // Sixth International Conference on Application of Concurrency to System Design (ACSD'06), 2006. - P. 3–14.

[3] J. Cortadella. Tools for automatic synthesis and verification of asynchronous interfaces. Invited lecture at the workshop // Asynchronous Interfaces: Tools, techniques, and implementations (AINT'2000), Delft, The Netherlands, July 2000.

[4] David L. Dill, Andreas J. Drexler, Alan J. Hu and C. Han Yang. Protocol Verification as a Hardware Design Aid // IEEE International Conference on Computer Design: VLSI in Computers and Processors, IEEE Computer Society. 1992. - P. 522–525.

[5] Murali Talupur and Mark Tuttle. Going with the flow: Parameterized verification using message flows // Proceedings of the 8th International Conference on Formal Methods in Computer-Aided Design (FMCAD), November 2008.

[6] J. Bingham, J. Erickson, and G. Singh, F. Andersen. Industrial Strength Refinement Checking // to appear at Formal Methods in Computer Aided Design (FMCAD), 2009.