УДК 004.415.538 DOI: 10.31114/2078-7707-2020-4-90-94

Автоматизация маршрута функциональной верификации на основе стандарта IP-XACT

С.А. Никитин, А.В. Николаев, Ф.М. Путря, И.А. Неклюдов

АО “НПЦ “ЭЛВИС”, г. Москва sanikitin@elvees.com, anikolaev@elvees.com, fputrya@elvees.com, inekluydov@elvees.com

***Аннотация* — В настоящее время особую актуальность приобретает автоматизация процесса верификации. Этому способствует в первую очередь, усложнение электронных схем с каждым годом, а также различные нестандартные варианты верификации, такие, как верификация систем на кристалле, в которых число вычислительных ядер и специализированных блоков растёт с каждым годом. В описанном подходе рассмотрена идея автоматизации сборки тестов и тестового окружения, а также запуска тестов, основываясь на стандарте IP-XACT.**

***Ключевые слова* — функциональная верификация, системы на кристалле, IP-XACT, автоматизация.**

I. Введение

В настоящее время функциональная верификация является наиболее критическим этапом в маршруте проектирования систем на кристалле. На сегодняшний день рост сложности цифровых устройств продолжает следовать закону Мура, а из этого следует рост сложности верификации. Исследования Cadence показали, что в большинстве проектов процедура верификации занимает большую часть времени разработки, что отображено на рис. 1 [1].

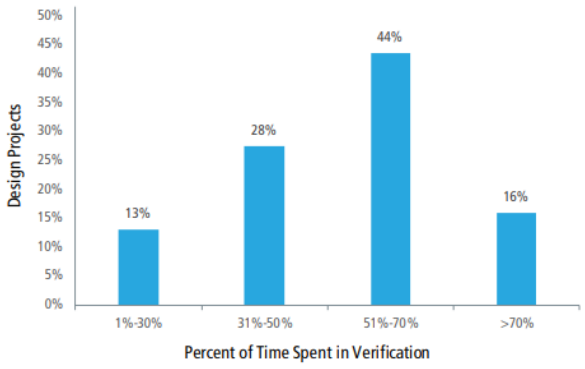

Рис. 1. Доля времени верификации в проектах по разработке систем на кристалле

Такие затраты времени экономически обоснованы тем, что недостаточно тщательная процедура верификации в ряде случаев может привести к сбою работы всей системы и предприятие может понести серьёзные убытки. Возвращаясь к закону Мура, следует отметить, что по состоянию на 2017 год количество транзисторов в современных процессорах достигло отметки 19.2 млрд. На рис. 2 отображён график роста числа транзисторов [2]. В текущем году этот показатель достиг отметки 32 млрд. Такое количество транзисторов содержится в процессоре Epyc Rome, выпущенном компанией AMD в 2019 году на техпроцессе 7 nm (TSMC) [3].

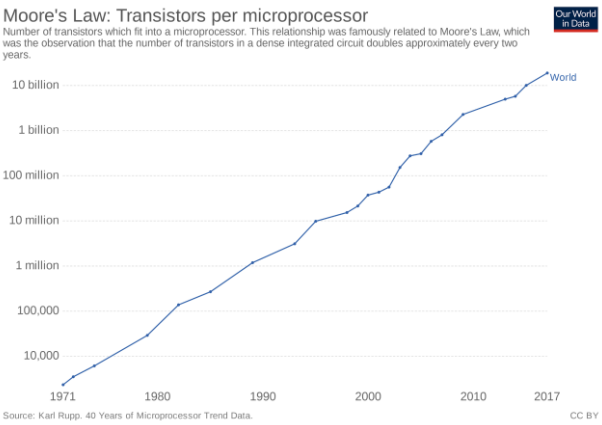

Рис. 2. Рост числа транзисторов с 1971 по 2017 г.

Учитывая такие показатели легко понять, что процедура синтеза и верификации не может существовать без средств автоматизации. На сегодняшний день есть множество методологий и средств, предоставляющих возможность автоматизации процесса верификации СнК, однако следует учитывать, что немаловажную роль играют комплексные средства, предоставляющие возможность автоматизации в рамках всего маршрута проектирования и верификации СнК. В каждой организации маршрут верификации построен по-своему, и как правило, средства автоматизации являются проприетарным программным обеспечением компании. Также важную роль в работе компании играет взаимодействие с другими организациями, будь то компании-интеграторы или конечные потребители. В процессе взаимодействия компании-производителя с другими компаниями может возникнуть необходимость передачи проекта. В случае если компания-потребитель является, например, интегратором, у неё может возникнуть потребность в программных средствах, которыми пользуется компания-производитель. К этим средствам относится проприетарное ПО, предоставляющее возможность

автоматизации. В случае, если различные программные средства являются сугубо проприетарными, то у компании-потребителя может возникнуть ряд проблем, связанных с адаптацией к поставленным средствам, так как компания-потребитель не имеет представления об этих средствах. Чтобы такие проблемы свести к минимуму, компания-производитель может создать своё проприетарное ПО, основываясь на общих стандартах, которые используются повсеместно. В этом случае, ею не будут разрабатываться механизмы, уже изобретенные ранее, а компания-потребитель сможет быстрее адаптироваться к приобретенным средствам. В данной статье речь пойдет о средствах автоматизации процесса верификации, в основе которых лежит стандарт IP-XACT.

Поставщики САПР также нередко сталкиваются с задачей автоматизации процессов и прикладывают усилия к тому, чтобы в рамках своего программного обеспечения предложить решения задач автоматизации. Однако на практике в подавляющем большинстве случаев получается так, что в маршруте проектирования используется программное обеспечение нескольких поставщиков, а также один поставщик, скорее всего, не покрывает задачи автоматизации всего маршрута верификации, так как на каждом предприятии маршрут верификации может сильно отличаться. Например, при поставке IP-блока разработчику СнК могут возникнуть задачи, требующие автоматизации, такие как компиляция RTL-модели, интеграция RTL-модели блока в RTL-модель системы, создание/генерация интеграционных тестов, генерация заголовочных файлов. Имеет место быть и поставка регистровой модели СнК разработчикам аппаратуры. В этом случае возникают задачи автоматизации генерации тестов регистров.

Существует также ряд проблем, решаемых самими пользователями. Например, проблемы интеграции со стороны ПО: выполнение на тестируемом процессоре кода, написанном на языке C/C++. Такая сцепка средствами САПР на текущий момент не проработана до конца. В связи с этим возникает необходимость создания набора собственных проприетарных средств, которые будут предоставлять возможность автоматизации.

II. IP-XACT

Разработка стандарта IP-XACT началась в компании NXP Semiconductors (Нидерланды), с помощью которой была предпринята попытка автоматизации разработки систем на кристалле. Эта компания работала с большой группой инженеров, производящей сотни чипов с небольшими модификациями, а проектировщики должны были интегрировать каждую систему на кристалле заново. Отсюда появилась задача, которая заключалась в создании способа описания IP-блока для использования в дальнейшем без необходимости проводить полное реконструирование системы на кристалле. На тот момент в качестве решения был выбран продукт Platform Express (Mentor Graphics), который в дальнейшем был включён в IP-XACT. Ранние версии стандарта разрабатывались в SPIRIT Consorcium, который объединился с Accellera в 2009 году. В июне 2009 года спецификация IP-XACT была передана IEEE-SA для утверждения в качестве стандарта IEEE 1685. Актуальной версией на сегодняшний день является стандарт IEEE 1685-2014.

IP-XACT описывает XML-схему для документирования метаданных IP-блоков, которые используются в проектировании и верификации электронных систем. Схема обеспечивает стандартный метод документирования IP-блоков, который совместим с технологиями автоматизированной интеграции SoC. Данная схема поддерживается многими САПР, нацеленными на разработку электроники [4].

На сегодняшний день стандарт имеет следующие основные поля:

1. Interface. Используется для описания интерфейсов. Схема разделена на две части: определение шины и определение абстракции. Эти описания содержат ссылки на шины и абстракции, соответственно, которые описываются отдельно. Описание шины содержит высокоуровневые атрибуты интерфейса (например, способ подключения), а описание абстракции содержит низкоуровневые атрибуты (имя порта, ширина порта, направление и т.д.). При этом, одно определение шинного соединения может быть связано с несколькими абстракциями.
2. Component. Используется для описания метаданных, связанных с конкретным IP-блоком, который может быть интегрирован в систему. Компонент может в себя включать ядра процессора, периферийные устройства, соединения между ними (шинные интерфейсы), списки файлов, относящихся к компоненту, описание компиляторов и параметры компиляции, карты памяти и описания адресных пространств и т.д.
3. Design. Используется для описания всей системы в целом. Здесь описываются компоненты системы, а также их конфигурация и взаимосвязь друг с другом.
4. Abstractor. Используется для соединения между двумя различными абстракциями шины одного типа (например, APB_RTL и APB_TLM). Абстрактор содержит два интерфейса, имеющих одно определение шины, но разные определения абстракций.

Помимо стандартных полей, стандарт IP-XACT позволяет разработчику добавлять свои расширения через специальное поле VendorExtensions. Следует отметить, что стандарт не зависит от маршрута проектирования и не привязан к каким-либо поведенческим характеристикам IP-блока. Из этого следует, что стандарт гарантирует совместимость IP-блоков различных производителей. Примеры описаний IP-XACT приведены по ссылкам [7]-[10].

В настоящее время с помощью IP-XACT в основном решается задача интеграции IP-блоков в проект системы средствами САПР высокоуровневого проектирования. Например, с помощью IP-XACT можно описать определенные пины и распознать интерфейс AXI, а затем соединить блоки не по пинам, а целиком по интерфейсам. Такое применение IP-XACT существует в САПР Cadence и Synopsys [5].

IP-XACT обладает достаточно широкими возможностями. В рамках данной статьи покрывается лишь малая часть стандарта, которая задействует лишь одно основное поле - ipxact:component.

III. Автоматизация маршрута верификации

На сегодняшний день существует множество методик и видов верификации. Применяемый метод верификации в каждом конкретном случае в основном зависит от самого верифицируемого объекта. Например, если выполняется верификация IP-блока, реализующего конкретную функцию (протокол передачи данных, вычисления и т.д.), то верификация будет проводиться с помощью языков SystemVerilog. Однако, если объектом верификации является процессорное ядро, то в этом случае требуется совершенно иной подход к верификации, который предполагает не только подачу тестовых воздействий на DUT, но и исполнение тестов на самом DUT. Из этого следует, что тесты на основе прикладного кода должны описываться не на SystemVerilog, а на компилируемых языках высокого уровня (ЯВУ) (в рамках данной статьи речь идет о языках C/C++), или на языке ассемблера, если есть такая необходимость. Чаще всего в рамках одного SoC тестируется мультипроцессорная архитектура (как гомогенная, так и гетерогенная), требующая сборки исполняемых файлов под различные архитектуры в рамках одного теста. Это требование накладывает свои особенности на тест и на автоматизацию при сборке.

Ещё одна причина, по которой имеется необходимость написания тестов на ЯВУ заключается в обеспечении переносимости тестов. Если верифицируется IP-блок в связке с CPU (например, в составе СнК), то возможны два сценария:

1. Для верификации IP-блока можно заменить CPU на агента, и код, написанный на SystemVerilog запустить через SystemVerilog-агент.
2. Код, написанный на ЯВУ скомпилировать на самом процессоре и исполнить программу посредством самого процессора.

Второй сценарий является более медленным, так как моделируется работа процессора, но в то же время этот сценарий - более правильный с точки зрения проверки в силу того, что в первом случае мы проверяем абстракцию, а во втором случае возникают транзакции, генерируемые IP-блоком. Скомпилировав и исполнив программу управления IP-блоком, можно осуществить более полноценную проверку интеграции двух сущностей на уровне СнК. Также имеют место быть ситуации, где SystemVerilog неприменим и проведение верификации возможно только с использованием ЯВУ. Такими ситуациями являются прототипирование и тестирование готового чипа: средствами SystemVerilog невозможно протестировать программируемый блок. Здесь также можно применить перенос тестов, написанных ранее на ЯВУ, что также может сэкономить время верификации. Итого, если есть тесты, написанные на SystemVerilog, то в состав СнК их можно перенести через агенты и получить недостаточно полноценный способ проверки. А чтобы обеспечить полноценную проверку, необходимо запускать тесты на CPU. Переносимость предъявляет дополнительные требования к коду тестов, иначе говоря - код должен быть написан по определенным правилам. Это также влечёт временные затраты, однако выигрыш заключается в переносе и адаптации тестов к различным платформам.

С другой стороны, SystemVerilog обладает рядом преимуществ при автономном тестировании. Например, SystemVerilog предоставляет удобные средства создания контролируемых параллельных случайных воздействий разного уровня сложности и, адаптированность под воспроизводимость результатов (как частный случай можно рассмотреть воспроизводимость потоков - явное преимущество над ЯВУ в силу того, что при работе с ЯВУ управлением потоками занимается ОС и могут возникнуть проблемы воспроизводимости). Также SystemVerilog будет удобен при разработке тестов, направленных на проверку определенных свойств, которые очень сложно протестировать на системном уровне. Например, тесты краевых ситуаций, связанных с динамикой прихода событий. В таких тестах необходимо программировать объекты внутри системы, чтобы создавать синхронизированные во времени события.

Следует также рассмотреть ситуации, связанные с тестированием программируемых блоков (CPU, в частности):

1. Тестирование блока в составе СнК. Как ранее было описано, здесь применяются тесты на ЯВУ. Внутренняя работа блока остается прозрачной.
2. Тестирование блока как системы. Здесь речь идёт о необходимости проверки внутренних блоков процессора. При таком тестировании применяются генераторы тестов. Эти тесты могут быть на ЯВУ или на языке ассемблера.

Тесты на ЯВУ может писать программист, работающий только с программным обеспечением, что существенно влияет на скорость разработки.

Для целей верификации в организации используется специальная библиотека драйверов СФ-блоков и тестовых сценариев, позволяющая ускорить процесс разработки ПО под проектируемую СнК, подставляя примеры программирования СнК и шаблоны для будущих драйверов встраиваемых ОС, разработанные в процессе верификации [6].

Процесс автоматизации, описываемый в рамках данной статьи, заключается реализации автоматической сборки всех частей тестового сценария.

IV. Использование IP-XACT как средства автоматизации верификации процессорных ядер

Рассмотрим типовой маршрут разработки верификационного модуля (теста) на языке высокого уровня в АО НПЦ «ЭЛВИС», представленного на рис. 3.

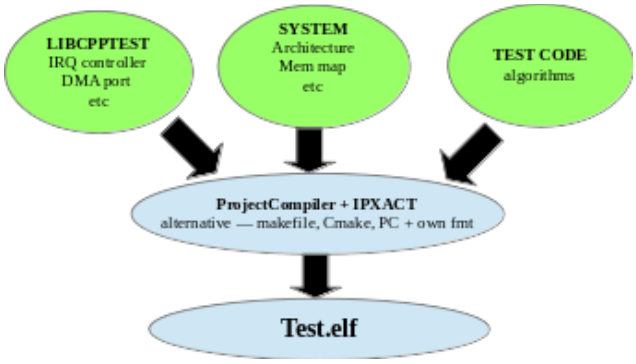

Рис. 3. Маршрут разработки теста

Для создания законченного теста программисту нужно собрать все необходимые драйверы устройств из библиотеки, при необходимости дополнив её (проектно-независимая часть), объединить их в систему, определив конкретные реализации интерфейсов, инструментов сборки и т.п. (проектно-зависимая часть) и запустить в этой системе алгоритм работы проверяемого устройства (тестовая часть). Очевидным решением для реализации подобной сборки является makefile (Cmake). Но такое решение крайне неудобно с точки зрения отладки из-за большой глубины иерархии скриптов makefile и необходимости задавать огромное количество параметров на всех уровнях иерархии. Поэтому была разработана система сборки на основе модульного средства Project Compiler, описанного в наших предыдущих работах [11]. Это позволило вынести алгоритмическую часть сборки в отдельный инструмент, но для настройки использовался проприетарный формат описания структуры файлов и инструментария. Это создавало неудобства при необходимости передачи кода третьим сторонам и по-прежнему сохранялась необходимость отдельного описания файлов кода от описания IP блока. Поэтому было решено совместить описания IP блоков и описания программной части верификационных компонент. Один из немногих стандартов, которые сегодня позволяют это сделать является IP-XACT. В нем есть возможность при описании компонента IP блока описывать также наборы файлов, служащие для сборки программного модуля, а также набора инструментов со всеми необходимыми параметрами.

Обычно стандарт IP-XACT используется только для описания регистровой архитектуры IP блока, мы же впервые начали создавать описания IP блоков содержащие информацию не только о карте памяти и регистрах устройства, но и всей информацией, необходимой для верификации блока, как то набор инструментов, параметры компиляции, структура и местоположение файлов с кодом.

Основная идея автоматизации при помощи IP-XACT заключается в том, что в конфигурационные файлы помещается метаинформация, в которой содержатся ссылки на такие же конфигурации других частей тестируемой системы. Таким образом, вся тестируемая система и окружение приводится к единому целому.

В основе решения для автоматизации маршрута верификации была задействована IP-XACT конструкция “component”, отвечающая за одну конкретную сущность (компонент) (драйвер, тест и т. д.). Внутри представления компонента для указания файлов различных типов использованы конструкции наборов файлов “fileSet”, внутри которых каждый файл является конструкцией “file”. Каждому файлу соответствует свой тип (source-файл, include-файл, файл зависимости). Поскольку работа ведется в основном с многоядерными гетерогенными системами, помимо файлов внутри компонента реализовано описание средств компиляции и сборки, а также пути и параметры под конкретные цели сборки.

Итого, для целей автоматизации используются следующие возможности IP-XACT:

1. Component - общая конфигурация (схема верхнего уровня);
2. FileSet - список файлов данной конфигурации, включая список файлов и зависимостей
3. File (файлы исходных кодов, includes);
4. DefaultFileBuilder - опции сборки (shell-команды, флаги, наименования средств сборки);

Проведенная работа направлена на автоматизацию маршрута верификации. В процессе проведения работы была разработана система инкрементальной компиляции проектов, базирующаяся на стандарте IP-XACT. Ранее задачи автоматизации верификации строились на проприетарных методах, основывающихся на XML-файлах. Используя такие методы, нередко возникают проблемы и дополнительные задачи, связанные с подготовкой проектов к использованию сторонними компаниями.

V. Заключение

В процессе проведения данной работы было выполнено:

1. Проведено сравнительное исследование методов автоматизации.
2. Создана собственная система компиляции тестов высокого уровня на базе стандарта IP-XACT, в чём и состоит новизна работы.

В данной работе также заложены возможности для проведения дальнейшей работы над автоматической сборкой RTL.

Литература

[1] URL: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/cadence-cloud/cadence-cloud-verification-wp.pdf (дата обращения: 07.10.2019).

[2] URL: https://ourworldindata.org/search?q=Transistors (дата обращения: 07.10.2019).
[3] URL: https://www.tomshardware.co.uk/amd-epyc-rome-7000-series-data-center-processor-zen-2-7nm,news-61389.html (дата обращения: 07.10.2019).
[4] URL: https://www.osp.ru/os/2012/03/13015149/ (дата обращения: 10.10.2019).
[5] URL: https://community.cadence.com/cadence_blogs_8/b/sd/posts/creating-systemc-tlm-2-0-peripheral-models (дата обращения: 27.01.2020).
[6] Головина Е., Макеева М., Николаев А.В., Путря Ф.М., Смирнов А.А. Метод создания и отладки комплексных тестов для функциональной верификации СнК, ориентированный на их повторное использование на всех этапах проектирования // Проблемы разработки перспективных микро- и наноэлектронных систем - 2014. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2014. Ч. 2. С. 45-50.
[7] URL https://github.com/tudortimi/ipxact/blob/master/tests/ieee/xml/SampleComponent.xml (дата обращения: 06.02.2020).
[8] URL: https://github.com/tudortimi/ipxact/blob/master/tests/Leon2/xml/NXP.com/serial_device/serial_device.xml (дата обращения: 06.02.2020).
[9] URL: https://github.com/tudortimi/ipxact/blob/master/tests/Leon2/xml/NXP.com/adaptor_scml/scmladaptor.xml (дата обращения: 06.02.2020).
[10] URL: https://github.com/tudortimi/ipxact/blob/master/tests/Leon2/xml/amba.com/AMBA3/APB/r1p0/APB_rtl.xml (дата обращения: 06.02.2020).
[11] Жезлов К.А., Колбасов Я.С., Козлов А.О., Николаев А.В., Путря Ф.М., Фролова С.Е. Автоматизация процесса создания тестовых окружений, обеспечивающая сквозной маршрут разработки, верификации и исследования СФ-блоков и СНК // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 2. С. 46-53.

Automation of Functional Verification Flow Based on IP-XACT Standard

S.A. Nikitin, A.V. Nikolaev, F.M. Putrya, I.A. Nekluydov

JSC “ELVEES”, Moscow, sanikitin@elvees.com, anikolaev@elvees.com, fputrya@elvees.com, inekluydov@elvees.com

***Abstract* — Currently, verification of SoC and IP takes a lot of time in the RnD process. So automation of verification has a growing role in reducing that time. Tests written on high abstraction level languages (such as C/C++) are very helpful in terms of reducing code writing time in case of big SoC or IP (such as processors or accelerators). But such tests require great efforts for compilation and assembling, so that process should be automated. This paper is about IPXACT based approach to automaton of compilation and assembling high abstraction level languages (such as C/C++) tests. This approach is powered by Python in a range of flows described in previous articles. Target executable file is assembled from library part (driver like code library, helpful in terms of code reusing for same IP), project specific part (memory maps, IRQ control, periphery interaction etc.) and test algorithm part. And it is possible to assemble many executables (according to target cores, maybe different type cores) and include it all in simulation. For that structure, our approach is much more flexible and ready to reuse than using common makefile. Also, we use IPXACT standard to store assembling and compilation settings to make our approach more transparent and compatible with other CADs and RnD flow. As a result, we achieve significant time benefit for test development that allow us to reduce verification time of SoC and large IPs.**

***Keywords* — Functional verification, System-on-a-Chip, IP-XACT, automation.**

REFERENCES

[1] URL: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/cadence-cloud/cadence-cloud-verification-wp.pdf (access date: 07.10.2019).
[2] URL: https://ourworldindata.org/search?q=Transistors (access date 07.10.2019).
[3] URL: https://www.tomshardware.co.uk/amd-epyc-rome-7000-series-data-center-processor-zen-2-7nm,news-61389.html (access date: 07.10.2019).
[4] URL: https://www.osp.ru/os/2012/03/13015149/ (access date: 10.10.2019).
[5] URL: https://community.cadence.com/cadence_blogs_8/b/sd/posts/creating-systemc-tlm-2-0-peripheral-models (access date: 27.01.2020).
[6] Golovina E., Makeeva M., Nikolaev A.V., Putrya F.M., Smirnov A.A. Reusable complex Soc level tests creating and debugging method // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2014. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2014. Part 2. P. 45-50 (in Russian).
[7] URL https://github.com/tudortimi/ipxact/blob/master/tests/ieee/xml/SampleComponent.xml (access data: 06.02.2020).
[8] URL: https://github.com/tudortimi/ipxact/blob/master/tests/Leon2/xml/NXP.com/serial_device/serial_device.xml (access date: 06.02.2020).
[9] URL: https://github.com/tudortimi/ipxact/blob/master/tests/Leon2/xml/NXP.com/adaptor_scml/scmladaptor.xml (access date: 06.02.2020).
[10] URL: https://github.com/tudortimi/ipxact/blob/master/tests/Leon2/xml/amba.com/AMBA3/APB/r1p0/APB_rtl.xml (access date: 06.02.2020).
[11] Жезлов К. А., Колбасов Я. С., Козлов А. О., Николаев А. В., Путря Ф. М., Фролова С. Е. Automation of verification environments development process providing a through design flow for design, verification and research of IP-blocks and SoC // Problems of advanced micro- and nanoelectronic systems development (MES) 2016. № 2. P. 46-53 (in Russian).