DOI: 10.22184/1992-4178.2019.186.5.120.125

Создание многофункциональной и мультипроектной платформы прототипирования на базе комплекта HAPS компании Synopsys

Часть 2

С. Фролова[1], Ф. Путря, к. т. н.[2]

УДК 658.512 | ВАК 05.13.12

Во второй части статьи рассматривается программная инфраструктура для использования тестов функциональной верификации и пользовательских сценариев на прототипе.

ФУНКЦИОНАЛЬНАЯ ВЕРИФИКАЦИЯ И ПРОТОТИПИРОВАНИЕ В ОДНОМ МАРШРУТЕ

Анализ стадии прототипирования и ее особенностей

Требования к конфигурации прототипа зависят от сценариев его использования. Чаще всего прототипирование воспринимается как стадия подтверждения функциональности программно-аппаратного комплекса и ранней разработки программного обеспечения. Подмножество программного обеспечения, работающего на прототипе, может быть определено как критерий готовности проекта. Высокая скорость прототипа используется для запуска приложений, библиотек и операционных систем для проверки функциональности и производительности проекта. На этом этапе могут быть найдены проблемы как в функциональности, так и в производительности. В некоторых случаях обратная связь от этой стадии может привести к перепроектированию SoC или IP и вызвать сдвиг сроков проекта.

Чтобы уменьшить вероятность обнаружения критических проблем на заключительном этапе проектирования, мы использовали прототип для более раннего выявления проблем проекта, подготавливая различные конфигурации прототипа и начиная разработку программного обеспечения до появления полной модели SoC, на этапе активной функциональной верификации IP и SoC. Также мы использовали прототип для ускорения задач верификации путем явного разделения проверяемых свойств между платформами цифрового моделирования и прототипирования.

На ранней стадии проекта ряд блоков еще находится на стадии верификации, верхний уровень проекта не верифицирован полностью и до конца не собран. Таким образом, мы должны решить следующие проблемы:

- определить классы типичных ошибок, которые с наибольшей вероятностью могут быть обнаружены именно на стадии прототипирования, чтобы сосредоточиться в первую очередь на выявлении подобных классов ошибок и повысить эффективность прототипа на ранней стадии разработки SoC;
- составить сценарии, которые можно более эффективно проверить на прототипе по сравнению с цифровым моделированием;
- создать прототип с простыми и гибкими возможностями отладки;
- создать инфраструктуре для удобного переноса коды тестовых сценариев между этапами моделирования и прототипирования для ускорения процесса отладки.

Относительно анализа ошибок и сценариев: на первый взгляд выявление ошибок при разработке ПО на прототипе носит не систематический, а скорее случайный характер. Выявление некоторых типов ошибок становится более вероятным на прототипе из-за другого стиля мышления программистов. Они концентрируются на задаче оптимизации производительности, а не проверки оборудования и, таким образом, иногда создают новые ситуации. Характер реальных данных, обрабатываемых SoC, может также отличаться от случайных и синтетических шаблонов, используемых при верификации IP, и создавать новые ситуации для аппаратного обеспечения.

Тем не менее статистика ошибок, обнаруженных в прототипе, позволяет нам составить список свойств и подсистем проекта, для которых обнаружение ошибок будет наиболее вероятным на этапе создания прототипа.

1. Проверка блока управления в случае, если устройство неоднократно перенастраивается. Зачастую более

[1] АО НПЦ «ЭЛВИС» г. Москва, г. Зеленоград, начальник лаборатории.

[2] АО НПЦ «ЭЛВИС» г. Москва, г. Зеленоград, начальник отдела.

сложные сценарии запуска и работы, предполагающие множественные изменения режимов и конфигураций устройства, могут выявить критическую ошибку в автомате управления устройством. Создание таких сценариев является нетривиальной задачей (из-за сложности их идентификации и полного перечисления) и чаще всего опирается на опыт и интуицию программиста-верификатора. На этапе моделирования RTL типичная последовательность запуска следующая: сброс оборудования, инициализация, запуск тестового сценария, анализ результатов. Для прототипа в FPGA ситуация иная – программист запускает, отлаживает и анализирует несколько тестов за один сеанс, применяя сброс только в крайних случаях. Такой подход к запуску тестов на ПЛИС часто позволяет выявить множество проблем при реконфигурации и вариациях в последовательности управления устройством.

2. Подсистема отладки. Взаимодействие программистов с прототипом во время запуска и отладки программного обеспечения увеличивает нагрузку на подсистему отладки. На этом этапе выявляется ряд как функциональных, так и архитектурных проблем подсистемы отладки (например, сложность или невозможность для программиста понять текущее состояние системы или суть проблемы в программе). Подсистема отладки является первым инструментом, используемым для определения источника проблемы с программным или аппаратным обеспечением на прототипе. Таким образом, число сценариев в подсистеме отладки, созданной на стадии прототипирования, может превысить число сценариев, созданных во время моделирования.

3. Работа с реальным физическим уровнем или периферийными устройствами. На стадии верификации периферийных IP-блоков верификационные СФ-блоки (VIP) должны помочь решить задачу проверки соответствия протокола на этапе моделирования. Но VIP – это всего лишь инструмент в руках человека, что-то все же может быть пропущено при проверке из-за человеческого фактора. Более того, VIP – это всего лишь инструмент проверки протокола, который не всегда с достаточной точностью эмулирует конкретные периферийные устройства. Прототип помогает выявить проблемы в части протокола (несоответствие в интерпретации пакетов данных на верхнем уровне) и проблемы на физическом уровне (перекос, нарушение целостности сигнала, влияние помех на работоспособность устройства и т. д.). Для некоторых периферийных устройств, таких как USB или SD, важно проверить взаимодействие с огромным количеством периферийных элементов различных провайдеров. В частности, эта задача относится к устройствам первичной загрузки.

4. Сброс, инициализация, первый запуск. Динамика событий сброса, последовательности включения тактовой частоты, процедуры конфигурирования и запуска устройства в прототипе обычно сильно отличаются от того, что имеет место при функциональной проверке. Это позволяет обнаруживать ошибки, связанные с начальным состоянием после сброса и процедурой начальной настройки.

5. Случайные и прикладные тесты. Высокая частота прототипа дает возможность выполнять огромное количество циклов случайных и прикладных тестов по сравнению с моделированием. С увеличением количества выполняемых тестовых строк кода вероятность обнаружения конкретных ошибок возрастает. Это могут быть проблемы самого конечного автомата IP или проблемы со взаимодействием различных IP.

Большинство описанных свойств могут и должны быть проверены на этапе функциональной верификации, однако прототип позволяет выявить нетривиальные проблемы в частях аппаратного обеспечения, описанных выше. К сожалению, отсутствие системы сбора показателей тестового и кодового покрытия от прототипа в ПЛИС не позволяет использовать его в качестве альтернативного средства функциональной проверки, поскольку трудно оценить, какой кумулятивный эффект с точки зрения покрытия дает прототип ПЛИС относительно симуляции. Прототип на этом этапе скорее выступает в качестве дополнительного детектора сценариев, выявляющего ошибки и сценарии, которые должны быть воспроизведены при моделировании RTL и введены в набор регрессионных тестов. С другой стороны, из-за возможности взаимодействия с периферийными устройствами и частотой прототипа (время выполнения на 2–3 порядка меньше, чем при моделировании RTL), некоторые типы сценариев проверки устройства могут быть выполнены только на прототипе. При таком подходе прототипирование больше не является только средством совместного проектирования аппаратной и программной части, а дополняет этап функциональной проверки. Он (этап) ускоряет поиск проблем в оборудовании и обеспечивает своевременную обратную связь как для процесса проектирования, так и для процесса верификации. Он также дополняет стадию верификации за счет распределения проверяемых свойств и сценариев между динамическим моделированием и прототипом. Такое распределение, учитывающее особенности устройства и возможности прототипа, должно осуществляться уже на этапе планирования верификации. В соответствии с вышеизложенным прототип должен появляться на насколько возможно ранних этапах разработки для:

- запуска прикладного программного обеспечения и анализа потенциальных функциональных проблем и проблем с производительностью;
- запуска прикладного программного обеспечения, прикладных и синтетических сценариев для большого количества реальных и синтетических данных

в различных режимах работы с огромным количеством реконфигураций для агрессивной проверки состояния конечного автомата;
- выполнения большого количества случайных тестов;
- анализа функциональности и эффективности подсистемы отладки;
- обмена данными и проверки взаимодействия с реальными периферийными устройствами.

Для ускорения процедуры создания прототипа чипа необходимо иметь инфраструктуру, позволяющую выполнять прототипирование IP-блоков, подсистем и системы (в полной и частичной конфигурации). Помимо этого, эти конфигурации прототипа должны подходить для миграции программного обеспечения между ними.

Сравнение способов организации взаимодействия тестового сценария и прототипа устройства

В зависимости от конфигурации прототипа сценарий тестирования может быть выполнен на прототипе несколькими способами. Рассматривая задачу прототипирования IP-ядра или подсистемы, мы должны эмулировать исполнителя основного сценария (в большинстве случаев для SoC это CPU), который выполняет управляющее ПО (драйвер IP и ПО для тестирования IP или приложение) и генерирует действия для целевого IP или подсистемы в прототипе. Было проведено сравнение семи конфигураций прототипа, которые определяют взаимодействие тестового сценария и аппаратного взаимодействия с прототипом.

1. **Управляющий компьютер – PCI-e – прототип.** Программное обеспечение, управляющее сценарием, работает на рабочей станции под управлением ОС Linux. Для запуска сценариев может использоваться как user- так и kernel-уровень (например, для разработки драйверов низкого уровня), но это усложняет ИТ-поддержку конфигурации. Обмен транзакциями между исполнителем и прототипом осуществляется по шине PCIE. На стороне прототипа необходима реализация моста PCIE-AXI.
2. **Управляющий компьютер – PCIE Synopsys UMR BUS – прототип.** Метод похож на предыдущий, но для обмена между рабочей станцией и прототипом используется шина Synopsys UMR BUS.
3. **Управляющий компьютер – ETH – прототип.** Конфигурация аналогична предыдущей, но с обменом данными через Ethernet.
4. **EMBEDDED CPU в прототипе.** Для управления сценариями используется CPU, синтезированный в FPGA. По возможности используется тот же процессор, что и в SoC.
5. **EMBEDDED CPU ext board.** Метод такой же, как и в предыдущем случае, но используется более быстрая реализация CPU на отдельной плате расширения.
6. **VP – ПЛИС Ко-симуляция (Гибридная виртуальная платформа).** Обычно вариант 1 или 2 используется для соединения между рабочей станцией и прототипом, но ПО работает на виртуальной платформе.
7. **RTL_SIM – ПЛИС Ко-симуляция.** Конфигурация такая же, как и предыдущая, но для запуска программного обеспечения используется RTL-симуляция.

Мы также использовали следующие параметры сравнения реализаций:

Трудозатраты на адаптацию теста или программного обеспечения. Необходимо адаптировать тест RTL к платформе прототипирования или программное обеспечение, отлаженное на прототипе, к реальному SoC. Числа даны для простого случая в человеко-часах. Эти цифры зависят от сложности устройства, организации кода и инфраструктуры для переноса тестов.

Скорость выполнения программ, миллионы инструкций в секунду. Приведенные числа в первую очередь определяются производительностью CPU, но также зависят от соотношения объема кода, который включает управление прототипом и код обработки данных, не связанный с управлением. Эти цифры показывают порядок величины в каждом конкретном случае.

Скорость обмена данными с основным процессором, МБ/с. Сравнение скорости обмена данными между исполнителем сценария и прототипом. Скорость обмена в данном случае – это приблизительные числа, действительные зависят от накладных расходов на трансляцию транзакций, скорости прототипа оборудования и частоты транзакций с устройством.

Затраты на настройку высокоскоростной периферии и сетевых устройств (в человеко-часах). Во время выполнения сценария может использоваться внешнее периферийное устройство, например внешнее хранилище данных, сетевые устройства (камеры и т. д.). Здесь мы сравниваем простоту подключения и программирования таких устройств из инфраструктуры прототипа. (Рассматривается возможность управления такими устройствами из основного потока управления теста / ПО.)

Повторное использование инфраструктуры на других платах. Определяет возможность простого повторного использования программной инфраструктуры в других решениях для создания прототипов.

Результаты этого сравнения приведены в табл. 1.

При работе в ОС Linux, на рабочей станции или встроенном процессоре трудозатраты по модификации теста более или менее одинаковы. В случае подхода на основе C для переносимых тестов (см. следующий подраздел

Таблица 1. Сравнение конфигураций прототипа

Platforms/Features	Трудозатраты на адаптацию теста или программного обеспечения	Скорость выполнения программ, в миллионах инструкций в секунду	Скорость обмена данными с основным процессором, МБ/с	Затраты на настройку высокоскоростной периферии и сетевых устройств	Повторное использование инфраструктуры на других платах
PC-PCI-FPGA	4	1000	4000 (800)	5	1
PC-PCI-FPGA(UMR)	4	1000	4000 (10)	5	0
PC-ETH-FPGA	4	1000	100	5	1
EMBEDDED CPU in FPGA	1	10	160	160	1
EMBEDDED CPU ext board	4	300	800	160	0
VP-FPGA Co-simulation	1	0,1	800	80	1
RTL SIM-FPGACo-simulation	1	0,001	800	160	0

Примечание. Числа показывают порядок величины и в общем случае зависят от множества факторов (например, для PCIE, теоретически дающего высокие скорости, фактическая скорость будет ограничиваться возможностями IP и транзакторов, синтезированных в ПЛИС)

«Перенос и отладка тестового сценария») затраты по переносу теста на такие платформы прототипирования не очень высоки. Единственная проблема с программным обеспечением, работающим в пользовательском пространстве Linux, – это переносимость обработчика прерываний (необходима реализация специального уровня). В случае виртуальной платформы или совместного моделирования с тестом RTL усилие переноса близко к нулю.

PCI и Ethernet обычно присутствуют во многих решениях для создания прототипов и изначально поддерживаются в рабочих местах, поэтому ПО, созданное для таких целей, может быть повторно использовано на многих платформах для создания прототипов.

В случае адаптации драйвера к прототипу идеальным вариантом является ЦПУ с той же архитектурой, что и в будущем СнК, и той же ОС, что и в реальной системе. В этом случае RTL CPU, синтезированный в ПЛИС, является предпочтительным. Точная модель в виртуальной платформе также может быть вариантом. Для решений, которые используют рабочую станцию для выполнения кода, разработчик должен всегда думать о прототипе как об одной из целевых платформ для сборки драйвера, чтобы сделать код переносимым.

С точки зрения контроля скорости выполнения программного обеспечения рабочая станция из вариантов 1–3 является наиболее предпочтительной. Предпочтительной она является и с точки зрения обмена данными, но встроенный ЦП будет конкурентоспособен здесь из-за высокой пропускной способности внутренней шины. Для гибридного моделирования скорость обмена данными зависит от типа обмена (сигнал, транзакция) и по умолчанию имеет много накладных расходов, вызванных временем выполнения моделирования, упаковкой / распаковкой данных.

Сетевые периферийные устройства могут быть легко доступными с рабочих станций под Linux.

Если мы суммируем все аспекты, перечисленные выше, мы обнаружим, что вариант с рабочей станцией, подключенной к прототипу через PCI-E, является наиболее подходящим для нас.

Перенос тестовых сценариев и отладка

Чтобы начать этап создания прототипа на ранней стадии проектирования (когда верификация все еще находится в плохом состоянии), важно упростить миграцию тестового кода между несколькими целевыми объектами: прототипами в ПЛИС, окружениями систем / подсистем и окружениями уровня IP. Это необходимо как минимум для двух основных действий:

- проверки прототипа после синтеза. Часть верификационных тестов может использоваться для быстрого запуска и отладки прототипа;
- быстрого воспроизведения ошибок на симуляции RTL. Воспроизведение на уровне IP предпочтительно из-за времени выполнения тестов. В сложных случаях могут использоваться RTL-системы, наиболее близкие к синтезированному в ПЛИС. Возможности по отладке в ПЛИС меньше, чем в RTL, и ее можно использовать только для первого этапа локализации ошибок. В этом случае важна возможность простого повторного запуска сценария

в наиболее подходящей среде RTL, поскольку это ускоряет разработку прототипа и проверку устройства в целом.

Для создания тестов мы создали специальную библиотеку libcpptest (рис. 8), которая имеет специальный аппаратный уровень абстракции (доступ к регистру/памяти, обработка прерываний и т.д.), а также набор правил и примитивов, которые облегчают создание переносимых тестов.

Также мы создали библиотеку агентов, JTAG, адаптеры Telnet и специальную версию GDB, которые поддерживают все платформы, используемые во время моделирования и прототипирования. Таким образом, мы можем воспроизводить сценарии GDB в прототипах системы или средах уровня IP. Это важно, потому что программисты обычно применяют GDB и среды разработки, использующие GDB для локализации и анализа проблем. Если один инструмент можно применять для воспроизведения одной и той же проблемы на разных платформах, это ускоряет работу программистов и верификаторов при отладке (рис. 9).

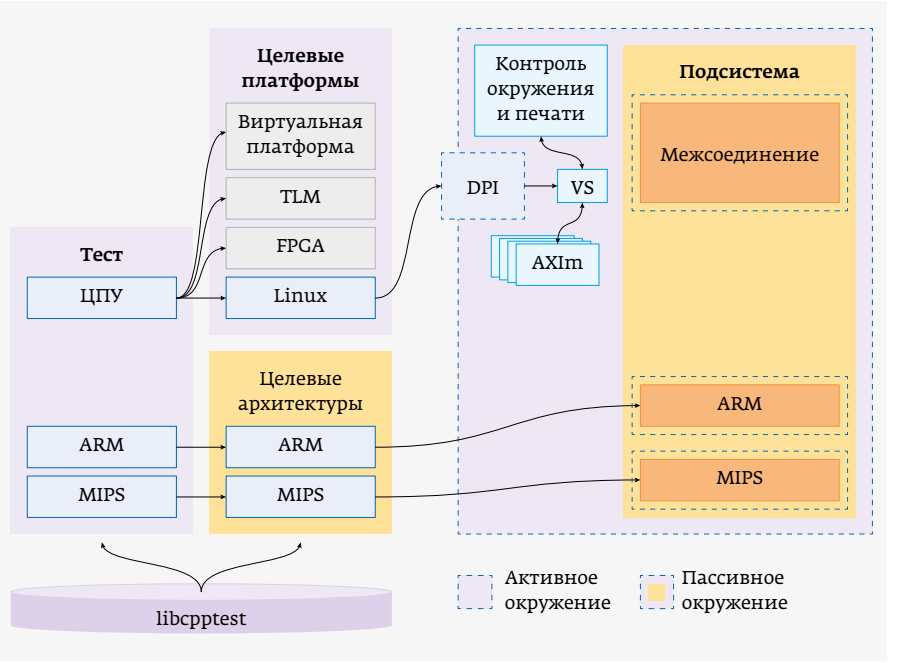

Рис. 8. Библиотека для создания переносимых тестовых сценариев

РЕЗУЛЬТАТЫ: КАКИЕ ЗАДАЧИ БЫЛИ РЕШЕНЫ НА ПЛАТФОРМЕ ПРОТОТИПИРОВАНИЯ HAPS

На данный момент мы используем наш HAPS для двух крупных проектов: первый из них практически закончен; второй – в стадии реализации.

Для обоих проектов мы использовали HAPS в качестве платформы для совместного проектирования программной и аппаратной части и ранней разработки программного обеспечения. Также у нас есть планы предоставить нашим клиентам доступ к прототипу.

Главной особенностью использования прототипов на основе HAPS в нашем потоке было раннее начало стадии прототипирования и его тесная связь со стадией верификации.

Ниже приведен список задач и проблем, которые были решены с помощью HAPS в двух проектах.

Проект 1:

- обнаружены проблемы в управляющем автомате и подсистеме отладки;
- обнаружены проблемы в производительности для конкретных задач (проблема архитектуры, решенная с помощью редизайна RTL без критического влияния на установленные сроки благодаря старту задачи прототипирования на ранней стадии разработки ИС);
- отладка, оптимизация, регрессия компилятора (gcc/clang);
- отладка ядра библиотеки OpenVX, отладка и оптимизация ядер CNN, регрессия;
- отладка и оптимизация библиотеки распознавания лиц, регрессия;
- разработка драйвера для Linux.

Проект 2:

- обнаружены проблемы в подсистеме отладки, межсоединения, подсистемы безопасности;
- загрузка Linux;
- разработка драйвера для Linux.

ЗАКЛЮЧЕНИЕ

В этой статье было объяснено, как создать платформу на базе комплекта прототипирования HAPS для целей отладки аппаратуры и ПО.

Мы создали унифицированную конфигурацию оборудования HAPS. Она может быть использована для прототипирования нескольких типов проектов, а также она применима для прототипов системного уровня, подсистемы и уровня IP (менее 5 мин на смену между подготовленными проектами). Это позволяет быстро менять различные типы проектов в условиях интенсивной работы разных команд.

Некоторые функциональные проблемы, которые могут проявиться на этапе функциональной верификации, были обнаружены и исследованы на прототипе. Это стало возможно потому, что этап прототипирования начался параллельно с функциональной верификацией. В этой

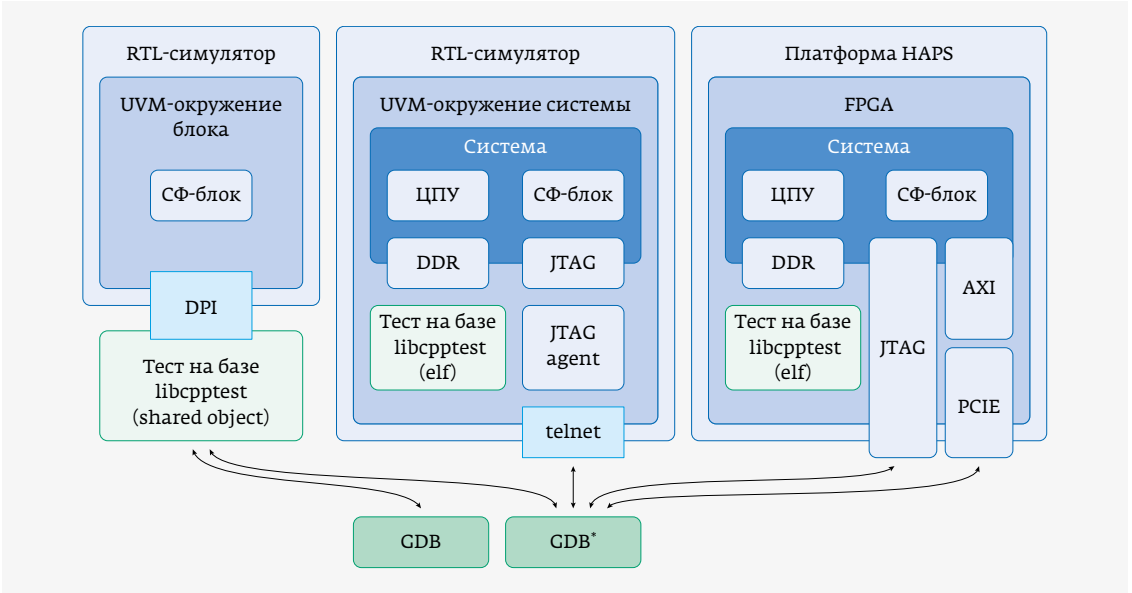

Рис. 9. Повторное использование скриптов GDB на прототипе и верификационном окружении (как для IP, так и для системы)

ситуации тесное взаимодействие с группами верификации и программного обеспечения помогает быстро локализовать проблемы и создавать новые сценарии для регрессионного тестирования RTL.

Раннее начало стадии прототипирования выявило ряд функциональных и архитектурных ошибок (ускорение поиска ошибок параллельно с функциональной верификацией).

Разница не менее чем на три порядка между симуляцией RTL и скоростью прототипирования (даже в случае интенсивного применения black boxes на верхнем уровне).

Время прохождения регрессии измеряется в днях (на прототипе), что делает невозможным их завершение на RTL в разумные сроки, даже в случае распараллеливания и применения замены частей проекта на «черные ящики». Таким образом, прототип также становится важной частью стадии верификации.

Раннее начало стадии прототипирования меняет распределение задач и приоритеты на стадии верификации. Например, в приоритетном режиме нам нужно закончить проверку функций, которые важны для запуска прототипа. На прототипе мы сосредоточены на сценариях, которые с более высокой вероятностью могут выявить ошибки, пропущенные на этапе верификации. Инфраструктура разработана для обеспечения переносимости тестов (моделирование ↔ прототип) и упрощения отладки.

Созданы как аппаратные, так и программные части инфраструктуры для упрощения миграции тестов между средами и прототипами RTL на уровне IP, подсистемы и системного уровня. То же самое относится и к программному обеспечению, используемому в целях отладки (например, GDB). Например, контрольную часть верхнего уровня теста нужно только перекомпилировать для выполнения на другой платформе, бинарные файлы для ядер и сценарии GDB можно использовать повторно, как есть. Небольшие усилия, необходимые для переноса тестов (минуты для перекомпиляции), облегчают взаимодействие между группами разработки ПО, прототипирования и верификации.

Одним из основных преимуществ раннего начала стадии прототипирования является снижение риска перепроектирования на поздней стадии проекта. Например, мы нашли проблему в архитектуре проекта и исправили ее на ранней стадии проекта, и она была решена без влияния на график выполнения проектов.

Предполагаются следующие шаги:

- расширение платформы HAPS для более глубокой проверки сценария варианта использования с внешними интерфейсами. Для таких задач Synopsys HW дает много возможностей;
- добавление к прототипу мониторов отладки RTL для мониторинга транзакций AXI и других событий проекта.

ЛИТЕРАТУРА

1. Hybrid Prototyping Guide February 2017. Copyright © 2017 Synopsys, Inc.
2. DDR3_SODIMM2R_HT3DDR3_SODIMM2R_HT3 Reference Manual© 2017 Synopsys, Inc. March 2017.
3. GPIO_HT3 Reference Manual© 2016 Synopsys. Inc. June 2016.
4. HAPS MGB Interface Card FamilyMGB Interface CardsMGB Interface Cards Reference Manual© 2016 Synopsys. Inc. June 2016.
5. **Putrya F.** Method of free C++ code migration between SoC level tests and stand-alone IP-Core UVM environments // Design & Test Symposium (EWDTS). 2014. East-West. P. 1–4.