УДК 004.415.538, 004.434

# Метод создания и отладки комплексных тестов для функциональной верификации СнК, ориентированный на их повторное использование на всех этапах проектирования

Е.К. Головина, М.А. Макеева, А.В. Николаев, Ф.М. Путря, А.А. Смирнов

ОАО НПЦ «ЭЛВИС», fputrya@elvees.com

***Аннотация* — Системы на кристалле (СнК) — сложные устройства, состоящие из множества СФ-блоков, связанных между собой коммутационной логикой. Даже при условии интеграции в СнК СФ-блока, прошедшего всеобъемлющую проверку автономными тестами (например, с использованием методологии UVM), высока вероятность ошибок, возникающих при взаимодействии СФ-блока с другими элементами системы на фоне сочетания множества факторов, обусловленных параллельной работой других блоков и процессов. Высокая нагрузка на все узлы системы может влиять на реально достигаемые значения функциональных характеристик и параметров интегрированного в СнК СФ-блока. Для выявления данных проблем на этапе проектирования СнК необходимо создание множества комплексных тестов, обеспечивающих различные типы нагрузок на узлы верифицируемой системы. Алгоритмы и программный код таких тестов сильно привязаны к составу и архитектуре СнК, сложны в разработке и отладке. Это, зачастую, приводит к невозможности должным образом проверить модель СнК в отведённые сроки. В данной статье предлагается метод создания и отладки комплексных тестов, упрощающий как их разработку, так и повторное использование на всех этапах проектирования СнК и между проектами СнК.**

***Ключевые слова* — СнК, верификация, комплексные тесты, тесты системного уровня, тесты интеграции, ООП.**

## I. Введение

В составе СнК СФ-блок не является независимой единицей, он неразрывно связан с коммутационной подсистемой (обладающей сложной системой буферизации, арбитража и маршрутизации) и другими СФ-блоками, обеспечивающими обмен данными и событиями с периферийными устройствами, между элементами СнК или управляющими системой. СФ-блок в совокупности с перечисленными элементами системы образует сложную подсистему, в составе которой функциональные характеристики СФ-блока могут отличаться от подтвержденных при автономном тестировании, а некоторые особенности взаимодействия элементов такой подсистемы могут даже приводить к потенциальным ошибкам, иногда имеющим фатальный характер. Таким образом, принципиально необходимыми являются комплексные тесты СФ-блоков, выполняемые на уровне всей СнК, в которых задействуются все СФ-блоки, входящие в состав системы, и имитируются обменные процессы, как характерные для реальных приложений, так и имеющие нагрузочный характер для различных узлов и подсистем СнК, а в особенности, подсистемы коммутации. Такие тесты позволяют подтвердить функциональные характеристики СФ-блоков в составе СнК при различных вариантах нагрузки на СнК, выявить архитектурные просчёты и ошибки, обнаруживающиеся только при одновременном стечении множества событий, а также оценить энергопотребление системы в различных режимах работы СнК.

Разработка тестов такого уровня крайне сложна, как с точки зрения создания алгоритмов и верификационного кода, охватывающих огромное число СФ-блоков различного типа, так и с точки зрения отладки такого рода тестов. Отладка тестов отягощается большим временем моделирования, требуемого для каждого запуска теста на модели СнК.

После окончания одного проекта СнК (а чаще всего ещё до окончания текущего проекта), начинается новый проект СнК, отличающийся по составу СФ блоков от предыдущего, в котором, однако, существенная доля СФ-блоков может быть повторно использована из прошлых проектов. К сожалению, код комплексных тестов сильно привязан к составу и архитектуре конкретной СнК, что приводит к сложностям его повторного использования и необходимости разработки практически с нуля для новой версии СнК, что, как уже было сказано, приводит к существенным временным затратам.

Таким образом, актуальным является решение следующих проблем:

1) Ускорение процесса создания и отладки комплексных тестов.

2) Обеспечение гибкости настройки комплексного теста для создания множества нагрузочных сценариев на базе одного отлаженного исходного кода комплексного теста.

3) Обеспечение возможности вертикального (между этапами проектирования СнК) и горизонтального (между различными проектами СнК) повторного использования кода комплексного теста.

## II. Подходы к ускорению процесса верификации СнК на системном уровне

Как правило, в силу простоты отладки тестов и скорости моделирования автономных тестовых окружений СФ-блоков (когда моделируется только аппаратура одного СФ-блока), именно на автономном уровне выполняется всеобъемлющая верификация СФ-блока и подтверждаются его функциональные характеристики. В силу вышесказанного один из очевидных путей ускорения процесса отладки тестов системного уровня - использование автономных окружений СФ-блоков для отладки частей кода комплексных тестов. Другим путём ускорения процесса разработки являются прототипы в ПЛИС и эмуляторы. Однако в случае ПЛИС ухудшается наблюдаемость внутренних событий и усложняется отладка. Кроме того, полностью эквивалентную модель СнК в ПЛИС, как правило, реализовать не удаётся.

На текущей момент автономные тестовые окружения и тесты СФ-блоков создаются преимущественно с применением методологии UVM [1]. В рамках подходов, описанных в UVM, предлагается использовать тестовые последовательности, созданные на SystemVerilog, как основу и для тестов системного уровня. Однако можно выделить ряд особенностей, препятствующих эффективному повторному использованию UVM кода на всех этапах проектирования СнК:

1) Для полноценной проверки СнК необходимо создавать тестовые программы, исполняемые непосредственно на модели CPU (как правило, C/C++ код). Это особенно важно при подходе к верификации программного аппаратного обеспечения как единого целого и при разработке драйверов.

2) Для полноценного тестирования топологического netlist и особенно прототипа в кремнии необходимы тесты, исполняемые на CPU в составе СнК.

3) UVM сложно использовать для задач прототипирования СФ-блоков или частей СнК в обычных FPGA без использования дорогостоящего САПР.

4) Восстановление аварийной ситуации, обнаруженной на готовой ИС в рамках автономного окружения СФ-блока, требует переноса кода на UVM.

5) Перенос на UVM сложных алгоритмов, реализация которых уже имеется на C/C++ в качестве тестов для будущих аппаратных ускорителей для этих же алгоритмов, трудоёмок и нецелесообразен.

6) Количество специалистов, умеющих использовать UVM, все ещё значительно меньше C/C++ программистов для встраиваемых систем.

В этом контексте использование языков программирования встраиваемых систем, например, C/C++ для создания тестов более выгодно. Однако для данных языков необходимо решить проблему повторного использования кода между этапами разработки СнК. В патенте US6539522 B1 [2] предлагается разделение тестового программного обеспечения на два уровня — низкий уровень драйверов и высокий уровень тестового приложения. Данный подход, похожий на подход построения ОС (например, Linux), позволяет создавать тестовые приложения на более высоком уроне абстракции, что упрощает как процесс создания приложений, так и их перенос между проектами. Тестовые приложения взаимодействуют с низкоуровневыми драйверами СФ-блоков посредством строго определённого интерфейса драйвера (driver API). Такая реализация позволяет создавать код тестового приложения, который может быть повторно использован на разных проектах СнК.

Однако процесс отладки при таком подходе по-прежнему сопряжён с целым рядом сложностей. Для запуска тестового приложения нужна операционная система (ОС), запуск которой на полной модели СнК крайне длительный процесс.

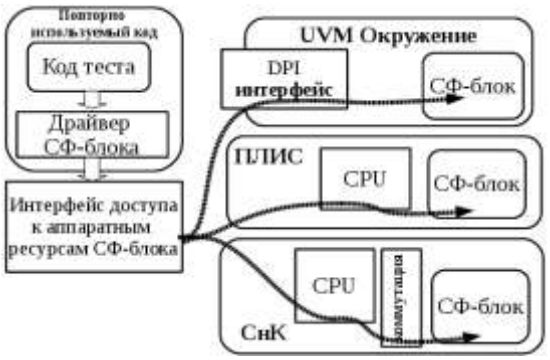


**Рис. 1. Повторное использование кода драйвера и тестового приложения для автономных тестов СФ-блока, прототипа в FPGA и тестов уровня СнК**

В работе [3] показано, что упомянутый выше подход имеет ряд ограничений. Для обеспечения возможности запуска теста на основе ОС на автономном окружении СФ-блока используется симулятор ОС, который может потребовать доработки или разработки с нуля. Ещё один минус использования тестов на основе ОС — ограничения планировщика, который не позволяет, с требуемой для тестов на пиковую загрузку системы точностью, контролировать запуск и останов процессов, управляющих СФ-блоками, а также полностью контролировать разделение физической памяти между процессами. Также имеются ограничения для тонкой настройки устройств при создании тестов (ограничения API драйверов).

Для решения этой проблемы предлагается модификация драйвера с использованием ООП и способа взаимодействия с ним таким образом, что снимаются оба описанных ограничения и становится возможной отладка кода и драйвера, и теста на

быстром автономном окружении СФ-блока без использования дополнительных инструментов (Рис. 1).

Упомянутые выше подходы позволяют упростить разработку, отладку на автономных окружениях и перенос между проектами СнК тестов отдельных СФ-блоков, однако для каждого из указанных действий остаются проблемы в случае комплексных тестов, в которых одновременно задействуются различные типы СФ-блоков, зачастую принципиально отличающиеся по организации и функциональному назначению.

## III. Тестовые сценарии и комплексные тесты

Особенностью комплексных тестов является привязанность тестового кода к составу СнК, что усложняет их повторное использование на проектах различных СнК. Для решения данной проблемы было выполнено два шага:

1) От теста требуется не полное воссоздание алгоритма реального приложения, а имитация обменных процессов и процессов обработки данных, создающих требуемую нагрузку на все элементы системы. Поэтому для комплексных тестов можно ввести упрощение — взаимодействие между СФ-блоками явным образом не программируется. Это упрощение позволяет создавать тест по частям таким образом, что каждая часть задействует только один или небольшую группу СФ-блоков, программируя их для организации имитационного процесса обмена или обработки данных.

2) Введены понятия **тестового сценария**, представляющего собой независимый тест одного или группы СФ-блоков, который управляет СФ-блоками исключительно посредством объекта драйвера СФ-блока, и **комплексного теста** — объекта, оперирующего тестовыми сценариями и способного выполнять конфигурацию, последовательный или параллельный запуск тестовых сценариев и анализ результатов исполнения каждого сценария.

Тестовый сценарий, таким образом, становится привязан не к конкретной СнК, а к СФ-блоку, что даёт возможность отлаживать его код на автономном окружении СФ-блока. Комплексный тест СнК, в свою очередь, составляется из уже отлаженных тестовых потоков СФ-блоков по аналогии с процессом проектирования самой СнК, реализуя платформенный подход и в процессе разработки комплексных тестов.

Для обеспечения возможности интеграции тестовых сценариев различных СФ-блоков в рамках одного комплексного теста необходима унификация тестовых сценариев. Идеальным механизмом для этого является заложенная в ООП возможность наследования. В рамках данной работы предлагается использовать принципы ООП как для создания драйверов, так и для создания тестов всех уровней, поскольку это упрощает создание кода, ориентированного на повторное использование, при этом накладные расходы, связанные с ООП, у современных компиляторов несущественны.

В базовый класс тестового потока заложены функции конфигурации, запуска, останова и контроля результатов теста, переопределяемые при реализации каждого тестового потока. Используя такой интерфейс комплексный тест может в едином стиле работать с любым сочетанием тестовых сценариев и выполнять их параллельный запуск.

По аналогии с ОС типа Linux можно дополнительно упростить разработку тестовых сценариев, введя классификацию СФ-блоков (например, последовательные порты, сетевые интерфейсы и т.п.). Для каждого класса устройств характерен определённый способ управления таким устройством и тип создаваемой им нагрузки на систему. Общие для каждого типа устройств части алгоритма управления могут быть вынесены в базовые классы тестовых сценариев, что дополнительно сократит трудозатраты на разработку новых сценариев.

Однако, несмотря на предложенное упрощение, от комплексного теста все же требуется имитация взаимодействия между СФ-блоками. Так, с одной стороны, для различных СФ-блоков должен имитироваться межблочный обмен данными, для организации которого может потребоваться и доступ к памяти одного СФ-блока со стороны другого, и передача событий или прерываний между процессами. С другой стороны, код тестового потока должен оставаться независимым от других СФ-блоков и системной организации СнК. При этом код тестового сценария должен динамически подстраивать алгоритм под ограничения, которые были переданы в процессе конфигурации или обусловлены составом СнК.

## IV. Конфигурация тестов и способы ее ускорения

В задачу конфигурации комплексного теста, таким образом, входят, во-первых, передача информации о системной организации СнК, достаточной для работы тестового сценария, не привязанного к конкретной СнК, а во-вторых, настройка тестовых сценариев для генерации определённой нагрузки на систему.

Чтобы понять, какую именно информацию о структуре СнК нужно передать сценарию, необходимо выявить различия между проектами СнК, влияющие на код теста. Проведённый анализ нескольких поколений СнК позволил выявить типовые отличия СнК:

1) Архитектура и количество CPU.
2) Подсистема обработки прерываний и событий.
3) Набор типов используемых СФ-блоков.
4) Число и базовые адреса СФ-блоков каждого типа.
5) Аппаратная реализация одного или нескольких СФ-блоков при сохранении выполняемых им функций.
6) Способ организации коммутации в СнК, карта адресуемой внутренней и внешней физической памяти.
7) Подсистема управления доменами частот и питания.

Использование ООП и языков высокого уровня снимает проблему, связанную с п. 1 данного списка. П. 2 требует введения и передачи тестовому сценарию в процессе конфигурации унифицированного интерфейса управления контроллером прерываний и событий. Реализация функций интерфейса выполняется отдельно для каждого типа контроллера.

Проблема п.3 решается путём модульной организации библиотеки драйверов СФ-блоков и тестовых сценариев, а также унификацией классов тестовых сценариев. В этом случае для компиляции комплексного теста необходимо подключить и скомпилировать только модули с драйверами и сценариями, требующимися для теста конкретной СнК.

Проблемы п.п. 4 и 5 решаются за счёт того факта, что тестовый сценарий взаимодействует с СФ-блоком посредством драйвера. При конфигурации сценария необходимо лишь передать ему указатель на объект драйвера, в котором уже учтены все особенности аппаратной реализации конкретного СФ-блока и его положение в карте памяти системы.

Для решения проблемы п. 6 каждому тестовому сценарию необходима информация о диапазонах адресов, с которыми он имеет право работать. Данная информация может быть передана в процессе конфигурации через объект, содержащий список диапазонов доступных адресов. Поскольку указанная проблема характерна практически для всех сценариев, функция конфигурации карты памяти вынесена в базовый класс тестового сценария (Рис. 2).

П.7 предполагает выполнение некоторой предварительной настройки СнК перед запуском тестов и на структуре самих тестов не сказывается. Тесты самой системы управления системными частотами и питанием могут быть построены на основе предложенного в данной работе подхода, однако имеют целый ряд особенностей и в данной статье не рассматриваются.

После передачи сценариям информации об аппаратных ограничениях СнК, каждый из сценариев должен, во-первых, выполнить адаптацию под переданные ограничения, а во-вторых, выполнить рандомизацию параметров, характерных для самого алгоритма сценария. Использование случайных сценариев делает задачу обнаружения ошибок более эффективной [4].

Исполнение кода такой дополнительной настройки сценария на модели CPU приводит к ситуации, когда более 90% времени моделирования теста уходит на конфигурацию и проверку результатов, что недопустимо в условиях ограничения времени на верификацию. Для решения этой проблемы использован подход, который обеспечивает быстрый доступ ко всем моделям используемых памятей напрямую, без необходимости моделировать обращения к ним через интерфейсы СнК [5]. Во-первых, это позволяет ускорить все операции с большими массивами данных, а во-вторых, подгружать заранее выполненную конфигурацию для каждого из сценариев. Для упрощения процедуры загрузки конфигурации к коду сценариев предъявляется лишь одно дополнительное требование: все параметры, необходимые для запуска и контроля результатов теста, должны храниться в диапазоне адресов, переданных сценарию при конфигурации.

Так, например, для теста СФ-блока DMA набор диапазонов может быть следующий: системный диапазон, достаточный для хранения переменных сценария (например, параметры для конфигурации устройства, число обменов, и т.п.), диапазон, достаточный для хранения микропрограммы или задания для DMA и произвольный набор диапазонов, между которыми должен быть организован обмен данными. После конфигурации и старта СФ-блок DMA непосредственно выполняет обмен данными со всеми диапазонами, кроме системного. Содержание всех приведённых диапазонов может быть быстро загружено в память перед стартом теста.

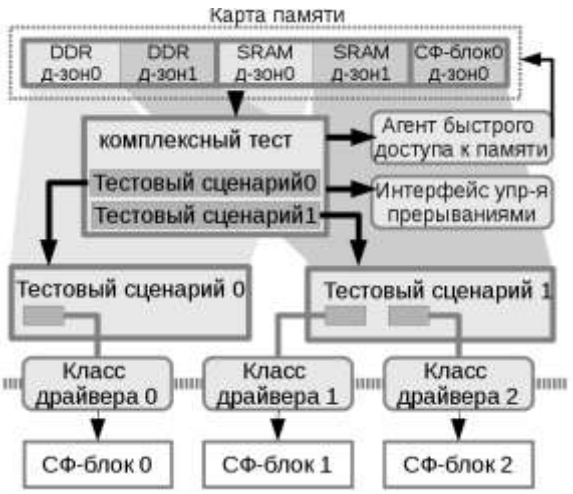


**Рис. 2. Связи, между компонентами тестового кода, конфигурация которых необходима для работы комплексного теста**

В результате, процесс полного запуска комплексного теста выглядит следующим образом:

1) Создание объектов тестовых сценариев и передача им указателей на драйверы СФ-блоков, и ограничений, характерных для каждого конкретного потока.

2) Формирование карты памяти для всех сценариев (разделение физической памяти).

3) Регистрация объектов сценариев в рамках комплексного теста.

4) Передача комплексному тесту карты памяти, распределяемой между сценариями.

5) Передача комплексному тесту (и далее всем сценариям) интерфейса работы с контроллером прерываний и интерфейса доступа к памяти.

6) Адаптация каждого сценария под переданные ограничения и рандомизация.

7) Конфигурация устройств со стороны каждого сценария для подготовки к запуску.

8) Параллельный запуск сценариев.

9) Ожидание окончания работы сценариев.

10) Проверка результатов тестов.

Для ускорения инициализации комплексного теста еще до запуска моделирования выполняются действия по конфигурации тестов с учётом переданных ограничений, результат такой настройки напрямую подгружается в память модели. Таким образом, модифицируются или исключаются следующие пункты приведённого выше списка:

2) Загрузка карты памяти с предварительно сохранёнными в ее диапазонах адресов данными для каждого из сценариев.

5) Передача комплексному тесту (и далее всем сценариям) интерфейса работы с контроллером прерываний и интерфейса *быстрого* доступа к памяти.

П 6. исключается из списка действий, выполняемых на модели CPU.

Эксперименты показали, что ускоренная инициализация вместе с использованием интерфейса быстрого доступа к памяти сокращают время на настройку и проверку теста вплоть до 5-10% от общего времени прохождения теста.

В результате создание комплексного теста становится возможным просто путём сборки из сценариев для отдельных блоков (которые уже были отлажены на автономных окружениях) и их конфигурации. А изменение характера нагрузки на систему обеспечивается путём перенастройки или рандомизации внешними инструментами карты памяти для каждого из потоков и передаче дополнительных ограничений каждому сценарию. Так, например, если каждому сценарию при конфигурации передать диапазоны адресов из различных физических блоков памяти, получится вариант теста с минимальным числом конфликтов по памяти между процессами. Напротив, если всем сценариям передать диапазоны из одного физического блока памяти, получится тест, создающий максимальную нагрузку на логику арбитража данного блока и существенное число конфликтов по памяти между процессами. Интерфейс быстрого доступа к памяти позволяет ускорить процедуру конфигурации, делая ее время несущественным на фоне исполнения самого теста.

## V. Тесты целостности модели СнК

На начальном этапе разработки СнК выявляется огромное число тривиальных ошибок как в коде теста, так и в модели СнК (как правило, ошибки соединения СФ-блоков). В рамках отладки комплексных тестов на локализацию такой ошибки требуются часы, а иногда даже дни. В этом контексте отладку сложных комплексных тестов нет смысла начинать, пока нет уверенности, что все СФ-блоки в составе СнК корректно проинтегрированы. Для выполнения проверки интеграции используются тесты целостности СнК, которые можно разделить на три типа:

1) Проверка доступности регистров и памяти СФ-блока со стороны системы.

2) Проверка доступности ресурсов системы со стороны СФ-блока.

3) Проверка корректности отработки прерываний со стороны СФ-блока.

Поскольку время на разработку данных тестов также весьма велико, и они являются отправной точкой для запуска более сложных тестов, то процесс их создания также необходимо упростить. Все тесты данной группы предлагается унифицировать и создавать на основе описанного выше подхода (Рис. 3).

Базовый сценарий для тестирования регистров СФ-блоков реализует функции проверки значений регистров по умолчанию и различные последовательности чтения/записи. Для реализации описанных алгоритмов для конкретных СФ-блоков требуются только таблицы масок и значений по умолчанию, которые автоматически генерируются из описания СФ-блоков на основе IPXACT [6].


**Рис. 3. Структура теста целостности, создаваемого в полуавтоматическом режиме**

Базовый сценарий для тестирования доступности памяти со стороны каждого СФ-блока реализует алгоритмы тестовых проходов по памяти. Реализация тестового сценария для конкретного СФ-блока дополняется функциями управления СФ-блоком, достаточными для поддержки алгоритмов, описанных в базом классе. Таким образом, тесты памяти могут быть выполнены единообразно для устройств принципиально разных типов, что весьма актуально для современных СнК содержащих множество устройств инициирующих обменные процессы (CPU, DMA, DSP, GPU, VPU). При этом тест автоматически перебирает сценарии проверки всей карты памяти СнК со стороны каждого из СФ-блоков.

Тест прерываний также является унифицированным. Для его реализации со стороны драйвера СФ-блока нужны только функции

формирования и сброса прерывания, регистрируемые в комплексном тесте обработки прерываний СнК.

Предложенные выше решения для организации тестов целостности СнК позволяют ускорить процесс проверки взаимосвязей между СФ-блоками и памятью в составе СнК, избежать траты времени на отладку интеграционных ошибок при запуске более сложных комплексных тестов и раньше приступить к подтверждению соответствия модели всей СнК заявленным характеристикам путём исполнения на ней набора комплексных тестов.

## VI. Заключение

Проблема разработки комплексных тестов сопряжена с поиском баланса между скоростью разработки таких тестов, простотой их отладки, гибкостью настройки, обеспечивающей возможность создания специфических тестовых ситуаций, и скоростью исполнения теста.

Тесты, не использующие никаких уровней абстракции и рандомизации, позволяют наиболее точно создать интересующую ситуацию и, как правило, требуют наименьшего времени моделирования (меньше накладных расходов), однако время разработки требуемого набора таких тестов может превышать сроки, отведённые на разработку СнК, а их перенос на новый проект требует переписывания и повторной отладки более половины кода теста. Тесты на основе ОС, необходимые для полноценного тестирования СнК и проверки взаимодействия ПО и аппаратуры, как правило, проще всего переносятся между проектами, однако для создания тестов целостности системы, нагрузочных тестов и тестов, подтверждающих функциональные параметры СФ-блоков в составе СнК, необходим другой подход, позволяющий более точно управлять запуском параллельных процессов. Кроме того, запуск ОС на модели СнК – это процесс, крайне затратный с точки зрения времени моделирования и отладки

Предложенная в данной работе структура комплексного теста, предполагающая разбиение теста на тестовые сценарии, создание тестов на высоком уровне абстракции с использованием ООП и прослойки драйверов СФ-блоков, позволяет упростить процесс создания комплексного теста и обеспечить возможность отладки частей теста на автономных окружениях СФ-блоков или их прототипах в ПЛИС.

Проведённый в данной работе анализ позволил выявить для тестового сценария набор интерфейсов и параметров, достаточный для обеспечения независимости кода сценария от структуры СнК и простоты его повторного использования на новых проектах СнК, с одновременной возможностью гибкой настройки характера создаваемой тестом нагрузки на систему. Ускорение фазы конфигурации на основе быстрого доступа к памяти позволило достичь для комплексных тестов времени моделирования сравнимого с тестами, не использующими дополнительных уровней абстракции и рандомизации. В Таблице 1 указаны средние сроки переноса комплексного теста в зависимости от его способа реализации на новый проект СнК, отличающийся составом, числом и картой адресов СФ-блоков и памятей, системой обработки прерываний, архитектурой CPU.

Таблица 1

*Сроки переноса комплексного теста на новый проект*

| Непосредственное программирование | Использование интерфейса драйверов | Библиотека тестовых сценариев |
|---|---|---|
| ~1 месяц<br>> 1 для asm кода | 1.5-2 недели | 2-3 дня |

Библиотека драйверов СФ-блоков и тестовых сценариев, создаваемых на языках программирования встраиваемых систем C/C++, позволяет также ускорить и процесс разработки ПО под проектируемую СнК, предоставляя примеры программирования СнК и шаблоны для будущих драйверов встраиваемых ОС, разработанные в процессе верификации.

Использование стандартных языков программирования встраиваемых систем (C/C++) на всех этапах верификации СнК от автономных тестов СФ-блоков до комплексных тестов уровня СнК снижает входной порог знаний для программиста, занимающегося верификацией, поскольку в этом случае знания языков описания и верификации аппаратуры (HDVL) не обязательны, что расширяет круг инженеров способных подключиться к задачам верификации и является немаловажным фактором для организации успешного маршрута верификации СнК.

## Литература


[1] Rosenberg S., Meade K. A Practical Guide to Adopting the Universal Verification Methodology (UVM) // Cadence Design Systems. 2013. ISBN 978-1-300-53593-5.

[2] Devins R.J., Ferro P.J. Method of developing re-usable software for efficient verification system-on-chip integrated circuit designs // U.S. Patent US6539522 B1. Mar. 25, 2003.

[3] Putrya F. Method of free C++ code migration between SoC level tests and stand-alone IP-Core UVM environments // IEEE EWDTS. Rostov-on-Don, Russia. September 27-30, 2013.

[4] Wilcox P. Professional Verification: A Guide to Advanced Functional Verification. Kluwer Academic Publishers. 2004. 225 c.

[5] Lungu P., Zhu B.A. Fully Reusable Register/Memory Access Solution: Using VMM RAL // *vmmcentral.org.*

[6] Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows. IEEE Std 1685-2009. Published 18 February, 2010.