DOI: 10.15514/ISPRAS-2022-34(5)-2

Практика и перспективы применения открытых и собственных программных решений в маршруте верификации систем на кристалле

[1,2]*А.В. Гаращенко, ORCID: 0000-0002-5147-6186 <agarashchenko@elvees.com>*
[1,3]*Д.С. Лашина, ORCID: 0000-0003-1252-228X <dlashina@elvees.com>*
[1,2]*С.А. Никитин, ORCID: 0000-0001-7307-5976 <sanikitin@elvees.com>*
[1]*А.В. Николаев, ORCID: 0000-0001-5141-2521 <anikolaev@elvees.com>*
[1,2]*Е.А. Прокопьев, ORCID: 0000-0002-7362-061X <eprokopev@elvees.com>*
[1]*Ф.М. Путря, ORCID: 0000-0002-3127-291X <fputrya@elvees.com>*
[1,2]*Б.Н. Цыренжапов, ORCID: 0000-0002-1212-0275 <btsyrenzhapov@elvees.com>*
[1]*АО НПЦ «ЭЛВИС»,*
124460, Россия, Москва, Зеленоград, ул. Конструктора Лукина, д. 14, стр. 142
[2] *Национальный исследовательский университет «МИЭТ»*
124498, Россия, Москва, Зеленоград, Площадь Шокина, д. 1
[3]*Национальный исследовательский ядерный университет «МИФИ»,*
115409, Россия, Москва, Каширское ш., 31

Аннотация. В статье рассматривается опыт и оценивается возможность применения свободного, открытого и собственного САПР в маршруте верификации СнК со степенью интеграции в миллиарды транзисторов, изначально базирующемся на коммерческих пакетах от "большой тройки". Предлагается подход к оценке пригодности конкретного САПР для заданного этапа маршрута верификации, основанный на формальном описании этапа и требований к средствам автоматизации для выбранного этапа. Приводятся собственные решения, внедренные в компании, являющиеся альтернативой коммерческим решениям или уникальными разработками. На основе предложенного подхода проведен анализ существующих средств автоматизации с точки зрения применимости в маршруте верификации современных СнК.

Ключевые слова: функциональная верификация; маршрут; система на кристалле; средства автоматизации; САПР; ПО с открытым исходным кодом; свободное ПО

Для цитирования: Гаращенко А.В., Лашина Д.С., Никитин С. А., Николаев А. В., Прокопьев Е. А., Путря Ф. М., Цыренжапов Б.Н. Практика и перспективы применения открытых и собственных программных решений в маршруте верификации систем на кристалле. Труды ИСП РАН, том 34, вып. 5, 2022 г., стр. 23-42. DOI: 10.15514/ISPRAS-2022-34(5)-2

The practice and prospects of using open and proprietary software solutions in the verification route of SoC

[1,2]*A.V. Garashchenko, ORCID: 0000-0002-5147-6186 <agarashchenko@elvees.com>*
[1,3]*D.S. Lashina, ORCID: 0000-0003-1252-228X <dlashina@elvees.com>*
[1,2]*S.A. Nikitin, ORCID: 0000-0001-7307-5976 <sanikitin@elvees.com>*
[1]*A.V. Nikolaev, ORCID: 0000-0001-5141-2521 <anikolaev@elvees.com>*
[1,2]*E.A. Prokopev, ORCID: 0000-0002-7362-061X <eprokopev@elvees.com>*
[1]*F.M. Putrya, ORCID: 0000-0002-3127-291X <fputrya@elvees.com>*
[1,2]*B.N. Tsyrenzhapov, ORCID: 0000-0002-1212-0275 <btsyrenzhapov@elvees.com>*
[1] *ELVEES Research and Development Center,*
14, stroenie 14, Konstruktora Lukina st., Zelenograd, Moscow, 124460, Russia
[2] *National Research University of Electronic Technology (MIET)*
1, Shokin Square, Zelenograd, Moscow, Russia, 124498
[3] *National Research Nuclear University «MEPhI»*
115409, Russian Federation, Moscow, Kashirskoe shosse, 31

Abstract. This article discusses the experience and evaluates the capability of using free, open and internal proprietary EDA tools in the billon gates SoC verification flow initially based on the commercial EDA of the “Big 3”. Article suggests an approach to assessing the suitability of a particular EDA tools for a certain stage of the verification flow based on a formal stage description and requirements for the tool. It presents our internal tools implemented in the company, which are used as an alternative to commercial tools or as unique solutions. Based on the proposed approach, the analysis of the existing automation tools from the point of view of the applicability in the SoC verification flow was carried out.

Keywords: functional verification; verification route; SoC; system-on-a-chip; automation tools; ECAD; EDA; open source software; free software

For citation: Garashchenko A.V., Lashina D.S., Nikitin S. A., Nikolaev A. V., Prokopev E. A., Putrya F. M., Tsyrenzhapov B.N. The practice and prospects of using open and proprietary software solutions in the SoC verification flow. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 5, 2022. pp. 23-42 (in Russian). DOI: 10.15514/ISPRAS-2022-34(5)-2

1. Введение

Функциональная верификация является частью маршрута верификации СнК, пронизывающая его от архитектуры до изготовления опытных образцов. Верификация является одной из самых затратных стадий разработки. При этом критическим является как время инженеров, необходимое на разработку и отладку окружений с тестами, так и время работы серверного парка и специализированной аппаратуры, необходимое на итеративные запуски моделирования и формальной верификации. По этой причине наличие специализированных САПР и аппаратного обеспечения, а также автоматизация процессов являются критическими в маршруте верификации.

Маршрут разработки (и, в частности, верификации) СнК с большой степенью интеграции, применяемый в подавляющем большинстве центров проектирования, сильно зависит от пакетов коммерческих САПР и аппаратуры, поставляемых "большой тройкой" (Cadence, Synopsys, Siemens (Mentor)).

Вопрос диверсификации применяемых в маршруте разработки СнК САПР неоднократно ставился на открытых площадках [1] и в рамках коллективов компаний. Однако ответ на вопрос о необходимости замены большинства применяемых САПР на собственные и открытые решения связан с оценкой экономической целесообразности. Проблемой альтернативных САПР является качество и ограниченный набор поддерживаемых свойств, негативно влияющих на сроки разработки и риски получения нерабочей микросхемы. Сроки

и затраты на разработку и поддержку альтернативных решений требуемого качества также оказываются крайне высокими.

Однако при сохранении в маршруте верификации стержня из САПР большой тройки, в ряде компаний с целью диверсификации и повышения эффективности процесса разработки все же разрабатываются собственные решения в области автоматизации, совместимые с маршрутом, построенным на коммерческих САПР. Кроме того, развиваются и открытые САПР. Преимущественно такие САПР применяются в академической среде, однако есть и примеры интереса к открытым продуктам со стороны больших коммерческих игроков, поддерживающих открытые решения. В таких начинаниях отметились компании типа Google [2], Siemens. Открытые решения также вызывают интерес у небольших стартапов и коллективов, занимающихся разработкой в ПЛИС, для которых задача верификации тоже стоит, хоть и менее актуальна, чем для заказной разработки.

С одной стороны, интерес к открытому ПО со стороны крупных игроков даёт толчок в развитии открытых решений, с другой стороны, остаются риски, что проекты рано или поздно все равно уйдут в коммерческий сегмент и закроются. Поэтому ситуацию с открытыми инструментами важно анализировать на постоянной основе (свежий пример – закрытие сайта SysWIP). Далее в статье детализируется маршрут верификации и проводится анализ доступных инструментов, решающих задачи автоматизации на каждом из этапов.

2. Этапы маршрута верификации

Для оценки применимости САПР из различных источников в маршруте верификации стоит разбить маршрут верификации на этапы, для каждого из которых будет характерен свой подход к автоматизации и свои нишевые САПР. Перечень этапов разработки СнК, в которых с различной степенью вовлечённости задействованы верификаторы, представлен ниже:

- разработка стратегии верификации;
- разработка верификационного плана;
- планирование и отслеживание статуса работ;
- управление исходными кодами проекта;
- разработка эталонных моделей блоков и системы;
- разработка или поиск верификационных компонент (VIP);
- разработка верификационных окружений;
- разработка тестовых сценариев (уровень СФ-блоков);
- линт и семантический анализ;
- формальная верификация RTL;
- генерация тестов для полной проверки автомата или перебора сценариев использования СФ-блока или системы;
- разработка тестовых сценариев (уровень подсистем и системный уровень);
- проверка RTL прикладным кодом;
- моделирование RTL и списков цепей (netlist);
- отладка RTL с использованием результатов моделирования;
- отладка окружения и тестов;
- пазработка тестовых сценариев с возможностью запуска на различных этапах маршрута разработки СнК;
- анализ производительности системы, поиск узких мест в архитектуре и реализации системы;
- автоматизация запуска регрессий и построения отчётов о статусе верификации;
- контроль достигнутого качества верификации;
- ускорение моделирования RTL; эмуляция;

- прототипирование СФ-блоков или системы;
- запуск тестов на опытных образцах ИС (bring-up);
- валидация ИС;
- поддержка пользователей в процессе эксплуатации микросхемы.

С формальной точки зрения линт и семантический анализ являются задачей разработчиков, передающих RTL на верификацию. Однако качество данного этапа напрямую связано с трудозатратами верификаторов на отладку полученного RTL, поскольку без линта и семантических проверок RTL может содержать большое число механических и логических ошибок, отнимающих время на отладку на первоначальном этапе запуска тестов. Проблемы метастабильности при переходе доменов тактового сигнала и сигнала сброса находятся за рамками цифровой симуляции и без применения специальных инструментов могут быть пропущены при выпуске СнК. Примером инструмента решающего задачу описанных выше проверок перед передачей RTL на верификацию является SpyGlass [3].

Прототипирование и стадии, следующие за tape-out микросхемы, напрямую не относятся к верификации, однако часто требуют привлечения специалистов, занимавшихся верификацией. Вопросы оптимизации маршрута, касающиеся данных стадий, рассмотрены в разделе, посвящённом созданию сценариев с возможностью запуска на различных этапах маршрута разработки СнК.

Стадии, напрямую касаюшиеся маршрута верификации, с различной степенью подробности будут рассмотрены далее в статье.

3. Способ сравнения САПР для этапа маршрута верификации на основе матрицы требований

Для каждого этапа верификации предлагается создание его формального описания, обязательно включающего перечень входных данных для этапа, описание действий, выполняемых над входными данными и описание формируемых по окончании этапа выходных данных. Данное описание используется для выделения перечня действий, которые требуется автоматизировать. Необходимость автоматизации обосновывается требованием соблюдения сроков и экономических затрат, отводимых на выполнение этапа верификации при достижении требуемого качества получаемых результатов.

Исходя из полученного перечня действий, подлежащих автоматизации, формируется перечень требований к САПР. Требования формулируются таким образом, что ответ на вопрос о соответствии оцениваемого САПР требованию может быть только бинарный (либо удовлетворяет, либо нет). Требования могут иметь приоритеты, определяющие степень важности для выбора САПР. Требование может быть принципиально необходимо для выполнения действия, либо необходимо для достижения лучших экономических показателей. В последнем случае указывается степень влияния на экономические показатели в баллах. Перечень требований и их ранжирование производится по методу экспертных оценок. Для получения более объективной оценки работа должна выполняться экспертами из нескольких компаний, представителей отрасли. На основе перечня требований проводится сравнение существующих коммерческих, свободных, открытых САПР и инструментов, созданных внутри компании. Для оценки применимости САПР в маршруте верификации выполнятся проверка закрытия всех принципиально необходимых свойств и оценивается полученная сумма баллов по всем закрытым требованиям для упрощения сравнения различных вариантов САПР.

При выборе САПР, а особенно в случае оценки открытого и свободного ПО, важно учитывать и факторы, связанные с доступностью и поддержкой инструмента в процессе эксплуатации. Требуется оценить объем вовлечённой в разработку инструмента команды, возможность прямого контакта с представителями команды разработки, интенсивность выпуска релизов,

дату выхода последнего релиза, число потребителей продукта. Данная информация критически важна для понимания, что использование продукта не приведёт к появлению блокирующих проблем в процессе работы над проектом (например, связанных, с ошибками в ПО САПР, которые не могут быть исправлены в короткие сроки). Приведённый набор требований по умолчанию должен входить в матрицы сравнения для выбора САПР.

В последующих разделах приводятся примеры описания этапов маршрута верификации и требований к применяемым на них САПР. Сравниваются известные САПР с точки зрения применимости в маршруте верификации, а также приводятся примеры использования сторонних и собственных решений, на различных этапах верификации.

4. Пример сравнения САПР для моделирования RTL модели на базе этапа моделирования RTL и списка цепей

4.1. Краткое описание этапа верификации

В табл. 1 представлено краткое описание этапа верификации.

Табл. 1. Описание этапа верификации
Table 1. Stage of verification description

Входные данные	Действия	Выходные данные
• RTL или нетлист (Verilog, VHDL, SV) • описание задержек (SDF) • описание доменов питания (UPF) • верификационное окружение • тесты • (verilog, VHDL, SV, UVM [4], C++) • скрипты управления моделированием	• моделирование цифровой схемы	• логи моделирования • временные диаграммы (например, VCD) • трассы транзакций и событий • дополнительная база данных для отладки • база данных покрытия (кода и функциональное)

4.2 Сравнение доступных САПР с учётом требования к этапу верификации

Сравнение доступных САПР представлено в табл. 2.

*Табл. 2. Сравнение доступных САПР (X/V/Q = XCELIUM, VCS, QUESTA) * - обязательные требования*
*Table. 2. Compare of available EDA (X/V/Q = XCELIUM, VCS, QUESTA) * - mandatory requirement*

Требования	Приоритет /баллы	BIG 3	Другие коммер-ческие (МИР)	Другие коммер-ческие (РФ)	Свободное	Открытое	Собственное
Verilog	*	X/V/Q		eremex [5]		Icarus Verilator	
VHDL	*	X/V/Q		eremex			
SystemC	*	X/V/Q				Verilator	
SystemVerilog (HW)	*	X/V/Q				Verilator	
SystemVerilog (VER)	*	X/V/Q					
SDF	*	X/V/Q					
UPF	*	X/V/Q					
Смешанное моделирование нескольких языков	*	X/V/Q		eremex			
нет ограничения по объёму проектов	*	X/V/Q					
Сбор покрытия кода	*	X/V/Q					
Сбор функционального покрытия	*	X/V/Q					
Сбор трасс транзакций	90	X/V/Q					
Скриптовое управление моделированием	80	X/V/Q					
Скорость моделирования позволяет достичь килогерцового эквивалента для верхнего уровня проектов СнК	100	X/V/Q				Verilator	

В таблице приведены только наиболее важные требования к САПР для демонстрации формы самой таблицы. В последующих разделах для экономии места таблицы тоже могут быть сокращёнными или опускаться.

4.3 Наличие открытых и собственных решений

Существует несколько открытых проектов цифровых симуляторов. Наиболее известные Verilator [6] и Icarus [7]. Данные проекты активно используются академическим сообществом. Verilator применяется и в коммерческих разработках для ускорения моделирования (однако при этом теряется детализация моделирования, например, потеря возможности хранить 4 состояния). Для отдельных СФ-блоков, особенно элементов вычислительного конвейера или алгоритмических ускорителей Verilator может применяться

с достаточной степенью эффективности при условии решения проблемы сбора покрытия альтернативными методами (например, по трассам).

Однако особенностью окружений проектов СнК является активное использование верификационных СФ-блоков (чаще всего имеющих API UVM) для проверки периферийных интерфейсов, число и сложность которых растёт с каждым проектом. Применение доменов частот и питания создаёт дополнительные ситуации, отладка которых слабо закрывается возможностями указанных симуляторов. Проект современной СнК, собираемый из IP разных поставщиков в большинстве случаев оказывается смешанным дизайном, содержащим код на нескольких HDL и указанными симуляторами не может быть промоделирован. Поэтому пока можно говорить только о нишевом использовании открытых инструментов типа Verilator, но для проектов СнК замены коммерческим симуляторам на данный момент нет.

5. Стратегия и планирование

Разработка стратегии требует инструмента, обеспечивающего многопользовательское редактирование документов. Решения для этой задачи широко представлены на рынке коммерческих и свободных инструментов. Типовой пример Confluence [8] Направления планирования и отслеживания задач закрываются инструментами с функциональностью MSProject [9] и Jira. Аналоги упомянутых инструментов широко представлены на рынке, в том числе имеются и свободные.

Задача создания верификационного плана является уже более критической. ПО для создания плана должно позволять трассировать свойства от ТЗ и спецификации до тестов и сценариев, которые должны быть исполнены в процессе моделирования. Для поддержания плана в актуальном состоянии важна интерактивность. Изменение ТЗ и спецификации должно сразу прослеживаться в редакторе плана. Описание тестов и покрываемые свойства должны быть привязаны к реальным тестам, регрессии и инструментам анализа покрытия. Поставщики САПР для верификации предоставляют подобные средства создания плана и контроля его исполнения. Как правило, они тесно интегрированы с САПР моделирования и запуска регрессий. Примером такого инструмента является VManager [10] от Cadence (в этом также году произошёл переход на платформу Verisium, объединяющей все инструменты верификации и внедряющей инструменты ИИ для оптимизации запуска и отладки регрессий [11]). Полноценных открытых альтернатив, сочетающих в себе весь перечисленный функционал, на данный момент нет.

6. Управление исходными кодами проекта

В настоящее время наиболее популярные системы контроля версий это Git [12] и SVN [13]. Позволяют сохранять историю изменений файлов исходных кодов, откатывать изменения, разграничивать права доступа к исходным кодам. Git в дополнении к вышеупомянутому позволяет настраивать политику доступа (раздельный доступ на чтение модификацию и т.д.), работать с отдельными ответвлениями, не затрагивая основной репозиторий, проводить минимальные обязательные проверки перед обновлением файлов репозитория, ставить метки на определённые срезы репозитория. Также существует множество коммерческих проектов вроде Perforce [14], Mercurial [15], предоставляющие расширенный функционал, вроде масштабируемости в Perforce, или большей скорости в Mercurial.

Проекты современных СнК разрабатываются несколькими командами, которые могут быть разнесены по разным локациям. Проект СнК разбивается на множество подпроектов, например, проекты СФ-блоков, эталонные модели, инструментальные средства разработки. Данные проекты могут иметь собственные репозитории, причём для большой компании может иметь место применение различных СКВ (систем контроля версий) для подпроектов. Проект всей системы собирается из множества подпроектов и, соответственно, репозиториев

подпроектов, в которых идут регулярные правки. Актуальным является наличие системы контроля исходных кодов, обеспечивающей стабилизацию проекта на всех уровнях иерархии и с учётом всех возможных зависимостей, включая инструментальное ПО, версии САПР и библиотек. Требуется регулярная фиксация иерархического среза проекта с возможностью гибкой настройки критериев для фиксации такого среза.

С учетом вышеперечисленного в нашей копании было принято решение о разработке проприетарной надстройки над системами git и svn, базирующейся на языке python и xml-описаниях зависимостейю Данный инструмент позволяет в одном проекте совместно использовать не только git и svn репозитории, но и задавать внешние зависимости от инструментария, например, компиляторов, библиотек или САПР, для которых используется собственный подход к хранению версий продуктов и выпуска релизов.

7. Разработка эталонных моделей, верификционных окружений и VIP

Высокоуровневая модель разрабатывается на этапе архитектурного проектирования и впоследствии может использоваться при верификации в качестве эталонной. Существуют коммерческие решения для построения виртуальных прототипов и моделей, однако базируются они в первую очередь на SystemC [16] и TLM [17] в качестве стандарта взаимодействия компонент. Разработка SystemC и C моделей возможна и без использования стороннего САПР (предполагается, что модели применяемых в проекте СФ-блоков доступны). Коммерческие решения позволяют создавать модели СнК быстрее и имеют интеграцию с САПР для моделирования, однако данные свойства не являются критическими для маршрута разработки моделей.

Существенные затраты времени уходят на разработку и отладку верификационного окружения. Одни из наиболее затратных компонент с точки зрения разработки – верификационные СФ-блоки (VIP). На разработку VIP для сложного интерфейса требуется несколько человеко-лет, а таких VIP в проекте требуется десятки типов. Компании из "большой тройки" предлагают свои VIP для решения данной проблемы. На рынке VIP имеются компании не входящие большую тройку, например [18], есть открытые решения VIP [19]. В стенах нашей компании разработана библиотека VIP для наиболее часто используемых накристальных интерфейсов [20] и ряда периферийных интерфейсов.

Для ускорения разработки верификационных окружений существует продукты, предоставляющие возможность генерации UVM окружений (есть коммерческие решения, например, SYSVIP от Cadence) и фреймворки с открытым исходным кодом для построения верификационных окружений и генерации тестов. В рамках верификации крупных СнК окружениям необходимо удовлетворять таким требованиям, как возможность использования VIP третьих сторон, скорость моделирования, удобство разработки и отладки большим коллективом разработчиков, а также удобство повторного использования кода между окружениями на различных уровнях иерархии проекта.

Подход к тестированию с использованием SystemVerilog и UVM является наиболее эффективным, но обладает повышенным порогом вхождения. В данный момент активно развивается подход к написанию тестов при помощи языка высокого уровня Python с использованием фреймворка CocoTb, обеспечивающего взаимодействие с RTL моделью [21] CocoTB применим для верификации автономных IP-блоков, однако данный фреймворк обладает целым рядом недостатков. К ним относятся сложность переноса тестов на системный уровень, прототипы и чипы, низкая скорость моделирования, невозможность интеграции UVM VIP третьих сторон, недостаток средств отладки.

Существуют фрейморки, ориентированные на оптимизацию процесса сборки окружений и запуска тестов. VUnit – фреймворк с открытым исходным кодом, созданный для целей автоматизированного тестирования RTL-кода, написанного на языке Verilog. Данный

фреймворк не заменяет, а призван дополнить традиционные методы тестирования посредством автоматизации. VUnit нацелен на минимизацию временных затрат на верификацию благодаря автоматическому обнаружению иерархии модулей, порядка компиляции, а также порядка включения библиотек. Скорость разработки повышается за счёт инкрементальной компиляции и возможности декомпозиции тестбенчей. Качество проектов при этом повышается, позволяя запустить большие регрессионные наборы на сервере непрерывной интеграции[22]

SVUnit – Фреймворк, аналогичный предыдущему, однако обладающий расширенной функциональностью и поддержкой языка SystemVerilog [23]. Оба упомянутых фреймворка не решают задачи собственно генерации окружений

Для ускорения процесса создания окружения в компании был разработан генератор UVM окружений. В качестве входных данных принимает метаинформацию о проекте (например, число и тип интерфейсов, карту памяти устройства, IPXACT описание блока) и тип проекта (автономное окружение, окружение коммутатора, окружение системы). Генерируемое окружение удовлетворяет перечисленным выше требованиям и содержит дополнительные свойства, связанные с переносимостью тестов между стадиями проектирования (см. далее).

8. Разработка тестовых сценариев, запускаемых на различных стадиях маршрута разработки СнК

Разработка и отладка тестов является одним из основных этапов и занимает большую часть времени инженеров верификаторов. Типовая ситуация для компаний, занимающихся разработкой СнК – наличие нескольких проектов СнК, как разработанных ранее, так и проектируемых в параллель. Очевидным решением с точки зрения оптимизации является повторное использование кода тестов, написанных для блоков, повторно применяемых в проекте. Однако, чтобы обеспечить максимальную простоту повторного использования, требуется соблюдение целого ряда правил написания и хранения кода тестов, позволяющую абстрагировать код теста СФ-блока от особенностей конкретного проекта, таких как архитектура процессора, особенности организации управления системой тактирования, контроллером прерываний, памятью и т.п.

От проекта к проекту конфигурация и версия СФ-блока, выполняющего схожую функцию (например, контроллер SPI, блок таймеров), может меняться. При этом большая часть тестовых сценариев, необходимых для проверки интеграции блока с точки зрения алгоритма теста остаётся одинаковой, при этом прямой перенос кода алгоритма с проекта на проект невозможен из-за отличий в регистровой модели СФ-блока.

Ещё одним направлением для повторного использования кода тестовых сценариев является необходимость их воспроизведения на различных уровнях иерархии проекта и на различных этапах разработки СнК. Например, перенос теста с автономного окружения на системный уровень для проверки интеграции блока или на прототип для проверки корректности функционирования прототипа.

Один из подходов к решению описанных проблем является стандарт PSS. Стандарт PSS (Portable Test and Stimulus Standard) позволяет описывать сценарии на самом высоком уровне, которые являются источником при генерации тестов под конкретный проект. В PSS также можно описать все ограничения блока и то, какие он действия выполняет. Все это описывается математическими графами, которые более понятны и удобны при написании сценария теста. PSS описывает намерение теста, которое должно одинаково интерпретироваться на всех уровнях (системы и подсистемы) и во всех других проектах, где будет использоваться данный блок [24].

Для генерации конечных тестов используется инструмент Perspec. Perspec – инструмент для автоматической генерации теста, который основан на модели. В Perspec можно составить

варианты использования (сценарии) и рандомизировать их. Также Perspec позволяет собирать данные о сценарном покрытии [25]. Обычно покрытие анализируются после запуска симуляций тестов. Механизм perspec позволяет собрать данные о необходимом покрытии до того, как верификатор переходит к генерации и запуску теста. Данный функционал помогает сократить число итераций при написании теста, который должен полностью покрывать заданную группу сценариев.

Однако для обеспечения портируемости сценариев, созданных с применением PSS, требуется наличие дополнительного кода, обеспечивающего запуск сценария на каждой из платформ. Кроме того, переход на создание кода на чистом PSS делает код тестов радикально отличающимися от кода целевого ПО, что усложняет повторное использования кода между стадиями верификации и разработки ПО. Негативным эффектом тут будет снижение скорости получения работоспособного ПО на микросхеме после её изготовления и затруднения при использовании кода прикладного ПО на этапе верификации.

В компании была предложена другая концепция создания переносимых тестовых сценариев на базе создания абстрактных классов тестовых сценариев для различных типов устройств. Обеспечить наилучшие условия разработки портируемых тестов позволяет кроссплатформенная библиотека тестовых функций libcpptest, содержащая драйвера устройств. Такие драйвера специалист создаёт и разрабатывает самостоятельно. При этом драйвер обращается к регистрам и макросам доступа к регистрам, которые генерируются исходя из IPXACT [26] описания блока, а также использует ядро упомянутой библиотеки для абстрагирования типовых операций взаимодействия с системой.

Библиотека тестовых сценариев предполагает создание сценариев на высоком уровне абстракции для групп устройств. Такой подход позволяет расширять код сценариев и настраивать их запуск на системном уровне независимо от конкретной реализации СФ-блока. Для обеспечения такой возможности все СФ-блоки делятся на группы, для каждой из которых создается абстрактный драйвер, содержащий в себе задекларированные методы, которые имеются во всех похожих блоках. Впоследствии эти методы реализуются в конкретных драйверах. К примеру, можно рассмотреть абстрактный блок таймера. Каждый таймер предназначен для отчета времени, которое в него заранее заложено. Поэтому один из сценариев, который можно написать для каждого таймера - это фиксация времени, что отчитывается от запуска таймера и до выработки прерывания (или иного события). Для абстрагирования таймера можно выделить следующие методы общие для каждого таймера: конфигурация и запуск таймера, остановка таймера, сброс таймера, сброс прерывания. Исходя из этих методов можно написать следующий алгоритм универсального теста таймеров:

- конфигурация таймера (параметры должны задаваться случайно);
- запуск таймера;
- фиксация времени перед запуском таймера;
- ожидание срабатывания таймера;
- фиксация времени останова таймера;
- сброс и остановка таймера;
- сравнение полученного интервала с эталонным значением.

При этом абстрактный сценарий может требовать определения как функций в драйвере конкретного СФ-блока, применяемого в системе, так и функций, зависящих от проекта системы или окружения; в приведенном примере это функция фиксации времени.

При конструировании тестовых сценариев для верификации подсистемы или системы достаточно важную роль играет разработка интеграционных тестов, таких как тесты регистров, памяти, прерываний и создание комплексных тестов, проверяющих корректность подключения устройств и их функционирование, в том числе при параллельной работе устройств в составе системы. Каждый из тестов имеет свои цели, объекты и способы реализации, и вместе формируют достаточно большой фронт работ, поэтому для повышения эффективности труда инженера-верификатора были разработаны специальные инструменты, позволяющие автоматизировать процесс создания драйверов, окружений и их сборку [27], [28] на базе платформенного подхода. С использованием упомянутых инструментов и библиотек для ускорения процесса создания тестов системного уровня был разработан специальный инструмент – конструктор типовых тестовых сценариев для проверки регистров, доступа к памяти, прерываний, комплексных сценариев проверки параллельной работы блоков.

Главной целью комплексного теста является проверка корректности взаимодействия СФ-блоков между собой путём одновременного или поочерёдного запуска набора тестовых сценариев верификации СФ-блоков и их взаимодействия в системе. Для его конструирования в библиотеке libcpptest создан ряд виртуальных классов, позволяющих разрабатывать потоки тестовых сценариев и конструировать из них комплексные тесты системного уровня [29]. При этом процесс разработки потоков тестового сценария и управление тестом на верхнем уровне позволяет разнести эти процессы. Так, интсрумент автоматизированного распределения памяти для тестовых сценариев разрабатывается независимо от тестовых потоков.

Ядро библиотеки libcpptest обеспечивает переносимость тестовых сценариев на автономные, системные окружения прототипы и на чипы и делает возможным применение всех упомянутых ранее подходов на всех этапах проектирования чипа.

Ядро библиотеки подразумевает разделение теста на два уровня: низкий – уровень драйверов, и высокий – уровень тестового приложения. В ядро введены 4 абстрактных класса, предоставляющими уровень абстракции от аппаратуры для взаимодействия с контроллером прерываний, доступа к памяти и регистрам (включая как прямой, так и backdoor доступ), а также механизмы отладочной печати.

9. Формальная верификация

На полный разбор средств и подходов к формальной верификации цифровых схем не хватит объёма одной статьи. В рамках формата текущей статьи считаем важным отметить, что изначально наиболее интенсивно инструменты для формальной верификации развивались в академической среде и существует большое число открытых и академических проектов разной степени проработанности, посвящённых формальной верификации. Изначально коммерческие инструменты были в роли догоняющих, однако за последнее десятилетие коммерческие инструменты сделали большой шаг вперёд и такие продукты как Cadence Jasper Gold от Cadence позволяют выполнять проверку свойств для дизайнов в миллионы транзисторов предлагают широкий набор дополнительных возможностей и адаптивный выбор решателей. Порог входа для использования коммерческих решений для верификаторов ниже чем у открытых альтернатив, что делает их использование выгодней с точки зрения сроков разработки. Однако в направлении открытых альтернатив работа ведётся достаточно интенсивно, в частности есть проекты формальной верификации на основе инструмента Yosys [30].

В рамках задачи поиска решений в области формальный верификации инженерами нашей компании был повторен результат, описанный в [31]. Как и в первоисточнике задача формальной верификации решалась для арифметических блоков АЛУ процессора. В итоге задача формальной верификации была успешно решена. Недостатком инструментов

оказалась не полная поддержка *Verilog* 2001 (*IEEE* 1364-2001), приведенная к необходимости адаптации проверяемого RTL под ограничения применяемого инструмент.

10. Генерация тестов для полной проверки автомата или перебора сценариев использования СФ-блока или системы

В рамках функциональной верификации СнК на этапе разработки тестов выделяют две глобальные задачи: полная проверка автоматов СФ-блоков и перебор сценариев их использования. Основными метриками оценки степени проверки выступают: значения тестового покрытия кода, блоков, выражений и сигналов, полученные при помощи САПР. Для максимизации полноты покрытий, а также проверки отсутствия ошибок и клинчей в архитектуре, реализации СФ-блоков и протоколов их взаимодействия помимо ручного написания применяются подходы автоматизированной генерации тестов.

Часто для автоматизации формирования тестов применяют методы на основе математических моделей [32] и методы на основе псевдослучайного дополнения шаблонов [33]. Генерация тестовых сценариев на основе методов с применением математических моделей явным образом не привязана к языкам программирования или верификации. Например, технология UniTESK и поддерживающие ее инструменты основаны на использовании моделей конечных автоматов [34]. Однако для построения математических моделей такого класса существует проблема экстрагирования пространства состояний автомата из формальных спецификаций и исходного кода RTL, что обусловлено высокой комбинаторной сложностью как отдельных СФ-блоков, так и системы в целом. Комбинирование различных алгоритмов генерации тестов и математических моделей СФ-блоков позволяет создавать как ожидаемые, так и критические ситуации для системы. На практике используют комбинации [35] в зависимости от имеющихся ресурсов.

Помимо этого, для генерации тестов применяются высокоуровневые модели блоков и подсистем, среди средств реализации которых используются следующие языки: C++, SystemC/TLM, SystemVerilog. Для проектов типа процессоров или акселераторов наличие полноценной высокоуровневой модели всего блока или устройства является обязательным, поскольку только при наличии такой модели возможно построить полноценный маршрут верификации на базе случайных и направленных тестов. Сгенерированный код представляет собой программные последовательности, написанные на языках ассемблера или типичных языках программирования встраиваемых систем (C/C++), компилируемых в базис целевой архитектуры системы и исполняемые на RTL. Данный подход активно используется для проверки вычислительных ядер, блоков программного управления, подсистемы памяти, предсказаний переходов и т.п. Код запускается на высокоуровневой модели, рассчитываются значения регистров и памяти, создаются их дампы, которые в дальнейшем используются при сравнении значений, полученных из соответствующего запуска кода на RTL.

Отдельной проблемой верификации на основе методов генерации тестов является продолжительность их моделирования, т.к. формирование тестов относится к классу задач, сложность (количество сгенерированных тестов) которых растет экспоненциально с ростом числа входных данных. Следовательно, для конструирования меньшего количества тестов, а соответственно и уменьшения общего времени моделирования, необходимо использовать такие методы искусственного ограничения пространства математической модели (например, вершин и ребер графа), как обобщение и унификация её состояний. Т.е. рассматривать структурные единицы модели на более абстрактном уровне без учета конкретных значений. При этом основной акцент генерации смещается на стрессовые для системы ситуации, приводящие её к различным блокировкам.

Для решения проблемы генерации тестов на основе описания автомата состояний и переходов существуют коммерческие инструменты, такие как Cadence Perspec,

базирующийся на стандарте PSS. Имеются также альтернативные инструменты, упомянутые выше. В рамках задачи верификации ядер DSP и CPU в компании были разработаны собственные инструменты генерации тестов для проверки автоматов datapath и кэшей данных процессоров.

11. Отладка RTL, тестовых окружений и тестов

Описание этапа верификации представлено в табл. 4.

Табл. 3. Описание этапа верификации
Table. 3. Description of verification stage

Входные данные	Действия	Выходные данные
• логи моделирования • временные диаграммы • трассы транзакций, инструкций и событий • дополнительная база данных для отладки • база данных покрытия	• анализ временных диаграмм и состояний регистров и памяти • анализ исходных кодов и схемы с привязкой к состоянию в заданный момент • анализ логов моделирования • анализ трасс транзакций и событий • сравнение временных диаграмм • сравнение трасс • анализ состояния динамических объектов окружения • пошаговая отладка RTL и тестбенча • подключения программного отладчика для анализа состояния процессора с привязкой к исходным кодам программы • поиск причины ошибки • составление инструкции для воспроизведения ошибки	• описание ситуации, в которой возникла ошибка, описание ожидаемого и фактического поведения • описание предполагаемой причины ошибки • инструкции и шаги для воспроизведения ошибок

В контексте маршрута верификации проектов СнК этап отладки, включающий применение большого количества САПР, один из самых ресурсоемких и продолжительных. На данном этапе итерационно производится поиск, анализ и локализация проблем, которые связаны не только с ошибками в самом RTL, но и с ошибками в окружении и тестах, поэтому задачу отладки формально возможно разделить на две смежных и часто пересекающихся по используемому инструментарию задачи: отладка RTL и отладка тестовых окружений и тестов.

Существуют разные подходы к отладке RTL, большинство из которых связаны с использованием результатов моделирования. Каждая итерация этапа часто требует перезапуска процесса моделирования с новыми параметрами, что резко ограничивает временные ресурсы. Основными признаками возникающих проблем корректности работы системы могут выступать: пропуск или расхождения в трассах транзакций и исполнения программ; дополнительная печать в логах прохождения тестов; недостаточное тестовое или функциональное покрытие; сработавшие assertions. На практике для минимизации времени моделирования RTL в тестовые последовательности добавляются блоки самопроверки и дополнительной печати, которые позволяют фиксировать ошибки, не покрытые при помощи assertions за счёт сравнения с эталонной моделью. Также разрабатываются специальные трассировочные блоки, встраиваемые в RTL и позволяющие в динамике «на лету» логировать дополнительную информацию, например, программную трассу вычислительных ядер.

Соответственно, первоочередным после фиксации проблемной ситуации и визуального просмотра логов при наличии эталонной модели проверяемого СФ-блока является сравнение

трасс. Часто простого сравнения недостаточно, инженерам-верификаторам необходимо разрабатывать дополнительные «парсеры», средства сведения логов и интеллектуального сравнения трасс, учитывающие недетерминированные ситуации с точки зрения несходимости моделей. Производится попытка локализации проблемы и осуществляется поиск возможных причин. После фиксации ситуации встаёт вопрос её воспроизводимости с возможностью восстановления контекста из сохранных SNAPSHOT для «длинных» тестовых последовательностей. Если на данной итерации не удаётся установить причину проблемы, необходимо произвести более глубокий анализ с использованием временных диаграмм, дампов конфигурационных регистров и памяти.

В зависимости от принципа и автомата работы СФ-блока к поиску проблем возможно зайти с разных сторон. Однако с точки зрения требований, предъявляемых к САПР, необходима конвергенция программных решений: визуализации временных диаграмм; менеджеров значений регистров и памяти; трассировщика связей блоков с возможностью отслеживания передачи значений шин и сигналов; редактирования исходного RTL-кода. Помимо этого, при моделировании необходима возможность фиксации, перебора и рандомизации задержек на интерфейсах. Также удобным и порой важным инструментом для воспроизведения и анализа проблем, возникающих при моделировании списка цепей, на RTL является средство сведения и сравнения временных диаграмм, полученных на разных запусках симуляции.

Для отладки тестовых окружений и тестовых воздействий применяется как подход, описанный выше, так и специфические именно для окружения шаги. Для окружений и тестов чаще применяются методы отладки, характерные для отладки ПО, возможно использование встроенных средств в связке с САПР: GDB, пошаговой отладки кода, анализа состояний объектов и переменных, дизассемблера, анализа секций объектного файла и т.д.

В части инструментов анализа трасс, в частности анализа трасс процессорных ядер или трасс транзакций в маршруте отдела верификации, используются как коммерческие решения, так и собственные решения, не уступающие коммерческим по функциональности. Однако в части анализа временных диаграмм в связке со схемой и исходными кодами альтернативы коммерческим инструментам в маршруте нет, кроме того, в инструментах типа SimVision от Cadence есть целый ряд дополнительных функций, ускоряющих процесс отладки.

12. Анализ производительности системы, поиск узких мест в архитектуре и реализации системы

Для обеспечения сходимости проекта СнК по критерию эффективности на реальных задачах весь маршрут разработки СнК, начиная с архитектурного проектирования и заканчивая квалификационными тестами топологического списка цепей, должен быть ориентирован на раннюю локализацию проблем с производительностью и подтверждение потребительских характеристик СнК перед её запуском на фабрику (tapeout). На начальных этапах для оценки производительности используются высокоуровневые модели, как правило TLM SystemC, которые в первом приближении позволяют оценить эффективность предлагаемой архитектуры для решения целевых задач. Для решения задач построения высокоуровневых моделей существует ряд коммерческих САПР, в частности, Cadence предлагает инструмент Helium Virtual and Hybrid Studio. Ранее отмечалось, что именно в части моделирования высокоуровневых прототипов критической зависимости от коммерческого САПР нет, хотя есть зависимость от поставщиков моделей IP. Однако экосистему для удобного построения моделей большая тройка предлагает в достаточной мере проработанную. В работе [36] подробно приводятся причины расхождения результатов оценки производительности на TLM и RTL моделях и необходимость оценки характеристик СнК на RTL. В этом плане преимуществом коммерческих решений является и возможность смешанного моделирования с использованием симуляторов и эмуляторов для уточнения параметров модели в процессе

детализации отдельных компонент, а также встроенных инструментов для анализа узких мест.

В компании был разработан ряд инструментов, базирующийся на подходе к формированию трасс транзакций и событий в процессе моделирования RTL, их анализа, автоматической проверки метрик производительности и визуализации результатов [37]. Инструменты разработанные внутри компании позволяют закрыть задачу анализа производительности системы на фазе разработки RTL, за счёт применения её декомпозиции и построения инфраструктуры для запуска тестов производительности и анализа их результатов на автономных окружениях вычислительных ядер, окружениях подсистем памяти, коммутаторов, а также подтверждения параметров на выборочном наборе тестов на RTL системного уровня [38].

13. Автоматизация запуска регрессий и контроль достигнутого качества верификации

Действия, требующие автоматизации со стороны комплекса управления регрессиями и контроля качества верификации, приведены в перечне ниже:

- запуск теста вручную с заданными параметрами;
- запуск регрессии по событию (расписание, при изменении исходных кодов в репозитории);
- запуск квалификационных тестов для кандидата в релиз (RTL, среда верификации или обе сущности);
- распараллеливание запуска регрессии с использованием доступных вычислительных ресурсов;
- хранение результатов запусков для всех перечисленных возможных вариантов запусков тестов;
- визуализация результатов запусков пакетов тестов для текущего среза проекта или релизов проекта;
- возможность просмотра текущего состояния верификации в режиме онлайн;
- оповещение заинтересованных лиц о резких изменениях состояния верификации проекта;
- накопление данных по достигнутому покрытию кода и функциональному покрытию;
- визуализация достигнутого статуса по покрытию по результатам прохождения пакета тестов;
- визуализация достигнутого статуса по покрытию по результатам накопления данных от серии запусков пакетов тестов;
- визуализация данных о закрытии позиций верификационного плана по результатам запуска пакета тестов;
- построение отчетов о статусе верификации по расписанию по результатам запуска регрессии;
- накопление и визуализация дополнительных метрик тестов для отладки регрессии (время исполнения теста, число предупреждений и ошибок в логах и т.п.)’
- возможность настройки отчета о статусе верификации и метриках тестов (определение отображаемых параметров и способа их визуализации для ускорения анализа статуса регрессии, возможность настройки разных типов отчетов, например, для инженеров и для менеджеров).

Коммерческие инструменты, например, Vmanager либо самостоятельно решают все перечисленные задачи, либо предоставляет интерфейс для python-скриптов для настройки

формируемых отчётов или поведения регрессии проекта. Однако, построение системы запуска регрессии и получения отчётности возможно и на открытых и свободных инструментах.

В процессе верификации проекта уровня СнК может возникнуть множество проблем, влияющих на общий ход событий и, таким образом, спровоцировать изменения в планах, сформированных изначально. В случае возникновения проблем в процессе верификации скорость локализации проблемы представляет наибольшую важность, поскольку в случае позднего обнаружения проблемы зачастую бывает затруднительно понять, с какого момента началась проблема, а главное — после каких изменений. В связи с этим важен контроль достигнутого качества верификации на текущий момент с возможностью отслеживания исторических изменений.

Наблюдаемые метрики и характеристики теста определяются ответственными за верификацию проекта и механизмы сбора, как правило, носят проприетарный характер. В компании такой механизм реализован в рамках ПО Project Compiler (ProjectCompiler - проприетарное ПО АО НПЦ "ЭЛВИС" впервые упомянуто в [39]). Экстрагированная из лог-файлов информация хранится в базе данных в специальном формате. В нашем случае информация хранится в формате JSON в базе данных на СУБД PostgreSQL, поскольку данная СУБД предоставляет гибкие возможности работы с JSON-структурами внутри полей. Формат JSON выбран исходя из соображения минимизации трудозатрат по добавлению новых метрик и характеристик теста: хранение в виде JSON позволит добавить новую информацию, не прибегая к модификации DDL-описания сущности базы данных. Оптимизация процесса контроля статуса верификации проектов достигается за счёт автоматизированного непрерывного анализа состояния регрессий.

В настоящее время для построения полноценной системы контроля качества верификации на всем её протяжении какого-либо одного инструмента недостаточно. Используют комбинации и проприетарных и свободных инструментов, таких как Vmanager, Gitlab, Jenkins, Grafana и практически всегда есть необходимость использования собственного проприетарного ПО, объединяющего перечисленные инструменты в маршруте и закрывающего отдельные функции, специфичные для внутреннего маршрута компании.

14. Ускорение моделирования RTL. Эмуляция

Компании из "большой тройки" поставляют специализированную аппаратуру для ускорения моделирования [40] и прототипирования цифровых схем. Без использования специализированной аппаратуры, позволяющей запускать большие объёмы тестов и прикладного кода (вплоть до ОС) представить себе верификацию СБИС СнК с должной степенью проверки сегодня нельзя. К сожалению, аналогов эмуляторов нет даже за пределами "большой тройки". Инструменты для прототипирования представлены более разнообразно, есть даже примеры их создания компаниями в России [41], однако на текущий момент остаётся зависимость от элементной базы и САПР для синтеза в ПЛИС.

15. Выводы

В маршруте функциональной верификации современных СнК полностью отказаться от использования САПР большой тройки в данный момент невозможно. Так, симуляторы и средства отладки только от большой тройки закрывают ряд критических свойств (например в части поддержки стандартов SystemVerilog, SDF, UPF, возможностей по сбору функционального покрытия и покрытия кода). Аппаратные средства ускорения моделирования в данный момент не имеют аналогов. Усугубляется ситуация тем, что многие средства автоматизации выстраиваются на основе возможностей, предоставляемых симулятором и, таким образом привязаны к решениям от большой тройки.

Тем не менее с большинством этапов верификации, за пределами обозначенных выше, ситуация обстоит не так критично, а это почти десяток инструментов и утилит помимо симулятора. Существуют как альтернативные коммерческие предложения, так и открытые решения либо, внутренние решения компаний, занимающихся разработкой СнК (в частности упомянутые в данной статье решения на примере одной компании). Использование альтернативных решений позволяет как диверсифицировать маршрут верификации в части зависимости от поставщиков САПР и поддержки получаемой от них, так и получить новые свойства, позволяющие повысить эффективность решения задач верификации.

В частности, в нашей компании были разработаны следующие собственные решения, закрывающие часть этапов верификации:

- генераторы верификационных окружений;
- библиотека собственных VIP (UVM SystemVerilog);
- инструменты для анализа производительности сетей на кристалле;
- методология разработки тестов, переносимых между стадиями верификации;
- система конфигурируемых генераторов тестов для вычислительных ядер;
- универсальное средство сборки и запуска окружений и тестов.

В компании успешно применяется открытое и свободное ПО для следующих этапов:

- система управления исходными кодами и выпуска релизов;
- система запуска регрессионного тестирования и анализа результатов таких запусков;
- отдельные VIP.

Также был получен опыт применения открытого и свободного ПО для следующих этапов:

- создание стратегии верификации;
- планирование работ;
- формальная верификация (арифметические блоки);
- создание верификационного плана;
- отслеживание задач и ошибок.

ПО из последнего списка было впоследствии заменено на коммерческое, поскольку последнее позволяет решать описанные задачи с большей эффективностью. Однако опыт показал принципиальную возможность использования свободного и открытого ПО на данных этапах, и в случае необходимости, с учётом готовности пойти на определённое снижение эффективности работы, возможно использовать и альтернативные решения.

Упомянутые в статье альтернативные (как открытые, так и собственные) решения не являются продуктами, сопоставимыми по потребительским качествам с коммерческими. Т.е. имеется задел по САПР для многих этапов, однако по каждому из перечисленных этапов доведение САПР до продуктового качества потребует усилий команд разработчиков.

Для более оптимального построения работы по разработке и поиску альтернативного САПР предлагается использовать описанный в статье подход по выработке и ранжированию требований к альтернативным решениям. При этом к составлению перечня требований должны привлекаться специалисты из большинства центров проектирования, являющихся потребителями САПР.

Кроме того, центры проектирования СнК могут и сами в ряде случаев являться поставщиками инструментов разработки, поскольку имеют внутренние решения, поддрживающие этапы маршрута проектирования.

Список литературы / References

[1] Обзор конференции МЭС-2018 / Overview of the MES-2018 conference. Available at: http://www.mes-conference.ru/index.php?prev=mesPrev&yp=2018&mpd=mpd_srv#sv, accessed 20.10.2022 (in Russian).

[2] Build Custom Silicon with Google. Available at: https://developers.google.com/silicon, accessed 20.10.2022.
[3] SpyGlass. Available at: www.syposys.com, accessed 20.10.2022
[4] Harshitha N.B., Praveen Kumar Y.G., Kurian M.Z. An Introduction to Universal Verification Methodology for the digital design of Integrated circuits (IC's): A Review. In Proc. of the International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 1710-1713.
[5] Компания ЭРЕМЕКС. Официальный сайт / EREMEX Company. Official site. Available at: https://www.eremex.ru/, accessed 20.10.2022 (in Russian).
[6] Verilator. Available at: https://www.veripool.org/verilator/, accessed 20.10.2022.
[7] Icarus simulator. Available at: http://iverilog.icarus.com/, accessed 20.10.2022.
[8] Atlassian. Available at: https://www.atlassian.com, accessed 27.10.2022
[9] MSProject. Available at: https://www.microsoft.com/, accessed 27.10.2022
[10] Zhang D. Smarter Verification Management with vManager Platform. In Proc. of the DVCon China, 2021, 40 p.
[11] Cadence Revolutionizes Verification Productivity with the Verisium AI-Driven Verification Platform. Available at: https://www.businesswire.com/news/home/20220913005378/en/Cadence-Revolutionizes-Verification-Productivity-with-the-Verisium-AI-Driven-Verification-Platform, accessed 27.10.2022.
[12] Git. Available at: https://git-scm.com, accessed: 21.11.2022.
[13] Apache Subversion. Available at: https://subversion.apache.org, accessed: 21.11.2022.
[14] Perforce Software. Available at: www.perforce.com, accessed 27.10.22.
[15] Mercurial. Available at: www.mercurial-scm.org, accessed 27.10.22.
[16] SystemC. Available at: https://systemc.org/, access 27.10.22.
[17] OSCI TLM-2.0 Language Reference Manual. Available at: https://www.accellera.org/images/downloads/standards/systemc/TLM_2_0_LRM.pdf, accessed 27.10.2022.
[18] SmartDV. Available at: https://www.smart-dv.com/, accessed 27.10.22.
[19] OpenCores. Available at: https://opencores.org/, accessed 27.10.22.
[20] AMBA Specifications. Available at: https://www.arm.com/architecture/system-architectures/amba/amba-specifications, accessed 27.10.22.
[21] What is cocotb. Available at: https://docs.cocotb.org/en/stable/index.html accessed 27.10.22.
[22] What is Vunit. Available at: https://vunit.github.io/about.html, accessed 31.10.2022.
[23] SVUnit User Guide. Available at: http://agilesoc.com/open-source-projects/svunit/svunit-user-guide, accessed 27.10.2022.
[24] Balance M. Automating test from IP to SoC levels with portable stimulus, Available at: https://www.techdesignforums.com/practice/technique/automating-test-from-ip-to-soc-levels-with-portable-stimulus/, accessed 28.10.2022.
[25] Cadence Perspec System Verifier Supports New Accellera Portable Test and Stimulus Specification 1.0, Available at: https://www.design-reuse.com/news/44368/cadence-perspec-system-verifier-accellera-portable-test-and-stimulus-specification-1-0.html, accessed 31.10.2022
[26] Никитин С.А., Николаев А.В и др. Автоматизация маршрута функциональной верификации на основе стандарта IP-XACT. Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС), вып. 4, 2020 г., стр. 90-94 / Nikitin S.A. Nikolaev A.V. et al. Automation of functional verification flow based on IP-XACT standard. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 4, 2020, pp. 90-94 (in Russian).
[27] Жезлов К.А., Колбасов Я.С. и др. Автоматизация процесса создания тестовых окружений, обеспечивающая сквозной маршрут разработки, верификации и исследования СФ-блоков и СнК. Проблемы разработки перспективных микро-и наноэлектронных систем (МЭС), вып 2, 2016 г., стр. 46-53 / Zhezlov K.A., Kolbasov Y.S. et al. Automation of verification environments development process providing a through design flow for design, verification and research of IP-BLOCKS and SOC. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 2, 2016, pp. 46-53 (in Russian).
[28] Фролова С.Е., Путря Ф.М. Создание многофункциональной и мультипроектной платформы прототипирования на базе комплекта HAPS компании Synopsys. Часть 2. Электроника: наука, технология, бизнес, вып. 5, 2019, стр. 120-125 / Frolova S.E., Putrya F.M.. Creation of multi-functional and multi-project prototyping platform based on synopsys haps kit. Part 2. Electronics: Science Technology Business, вып. 5, 2019, pp. 120-125 (in Russian).

[29] Головина Е.К., Макеева М.А. и др. 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 (МЭС), вып. 2, 2014 г., стр. 45-50 / Golovina E.K., Makeeva et al. The method of building and debugging complex tests for the SoC functional verification, oriented on their reuse at all stages of the development. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 2, 2014, pp. 45-50 (in Russian).

[30] Symbiyosys. Available at: https://symbiyosys.readthedocs.io/en/latest/index.html#symbiyosys-sby-documentation, accessed 20.10.2022.

[31] David M. Russinoff. Formal Verification of Floating-Point RTL with ACL2, 2019. Available at: https://www.cl.cam.ac.uk/~jrh13/spisa19/paper_06.pdf, accessed 27.10.2022

[32] Garashchenko A.V., Putrya F.M. et al. Automatic Test Generation Methodology for Verification of a Cache Memory Based on the Graph Model of Cache Hierarchy. In Proc. of the IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, 2019, pp. 1876-1879.

[33] Chupilko M., Kamkin A. et al. MicroTESK: Specification-Based Tool for Constructing Test Program Generators. Lecture Notes in Computer Science, vol. 10629, 2017, pp. 217-220.

[34] Иванников В.П., Камкин А.С. и др. Применение технологии UniTesK для функционального тестирования моделей аппаратного обеспечения. Препринт ИСП РАН, вып. 8, 2005 г., стр. 1-26 / Ivannikov V.P., Kamkin A.S. Application of UniTesK technology for functional testing of hardware models. ISP RAS preprints, issue 8, 2005, pp. 1-26 (in Russin).

[35] Garashchenko A., Nikolaev A.V. et al. System of Combined Specialized Test Generators for the New Generation of VLIW DSP Processors with Elcore50 Architecture. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 2, 2019, pp. 2-7.

[36] Путря Ф.М. Особенности верификации современных систем на кристалле. Труды международной конференции «Микроэлектроника-2015», 2016 г., стр. 49-57 / Putrya F.M. System on chip verification issues. In Proc. of the International Conference «Microelectronics-2015», 2016, pp. 49-57 (in Russian).

[37] Жезлов К.А., Колбасов Я.С. и др. Этапы маршрута верификации и валидации системы, ориентированные на получение модели СнК, способной гарантированно исполнять целевые задачи и алгоритмы с заданными характеристиками. Вопросы радиоэлектроники, вып. 8. 2018 г., стр 64-72. / Zhezlov K.A., Kolbasov Ya. S. et al. Steps of verification and validation route aimed at obtaining the SOC model able to perform target tasks and algorithms with specified characteristics. Issues of Radio Electronics, issue 8. 2018, pp. 64-72 (in Russian).

[38] Жезлов К.А. Методика автоматизированного анализа производительности коммутационных сред с учетом структуры СнК и прикладных. Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС), вып. 3, 2020 г., стр. 94-99 / Zhezlov K.A. Methodology of automated performance analysis of SOC interconnect subsystems, according to SOC structure and application. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 3, 2020, pp. 94-99 (in Russian).

[39] Жезлов К.А., Колбасов Я.С. и др. Автоматизация процесса создания тестовых окружений, обеспечивающая сквозной маршрут разработки, верификации и исследования СФ-блоков и СнК. Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС), вып. 2, 2016 г., стр. 46-53. / Zhezlov K.A., Kolbasov Ya.S. et al. Automation of verification environments development process providing a through design flow for design, verification and research of IP-blocks and SOC. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 2, 2016, pp. 46-53 (in Russian).

[40] Flaherty N. Cadence boosts its emulation and verification systems, Available at: https://www.eenewseurope.com/en/cadence-boosts-its-emulation-and-verification-systems/, accessed 20.10.2022

[41] Юрлин С.В., Бычков И.Н. Прототипирование на основе ПЛИС для верификации многоядерных микропроцессоров. Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС), вып. 4, 2014 г., стр. 45-50. / Yurlin S.V., Bychkov I.N. FPGA prototyping for functional verification of multi-core processors. Problems of development of advanced micro-and nanoelectronic systems (MES), issue 4, 2014, pp. 45-50 (in Russian).

Информация об авторах / Information about authors

Антон Витальевич ГАРАЩЕНКО – кандидат технических наук, ведущий инженер АО НПЦ "ЭЛВИС", доцент института Института системной и программной инженерии и информационных технологий НИУ МИЭТ. Сфера научных интересов: автоматизация верификации гетерогенных многоядерных СнК.

Anton Vitalievich GARASHCHENKO – Candidate of Technical Sciences, Staff Engineer at JSC ELVEES RnD Center, Assistant Professor of SSEIT NRU MIET. Research interests: automation of heterogeneous SoC verification.

Дарья Сергеевна ЛАШИНА – техник АО НПЦ "ЭЛВИС", студентка магистратуры кафедры ИИКС НИЯУ МИФИ. Сфера научных интересов: функциональная верификация СнК, информационные технологии.

Daria Sergeevna LASHINA – Technician at ELVEES RnD Center, JSC, Master's Student of ICIS MEPHI. Research interests: functional verification, computer science.

Святослав Александрович НИКИТИН – старший инженер АО НПЦ "ЭЛВИС", аспирант кафедры ПКИМС НИУ МИЭТ. Сфера научных интересов: автоматизация верификации цифровых СФ-блоков и СнК, машинное обучение.

Svyatoslav Aleksandrovich NIKITIN – Senior Engineer of ELVEES RnD Center, JSC, postgraduate student of DICD NRU MIET. Research interests: Digital IP and SoC verification automation, Machine Learning.

Артём Валерьевич НИКОЛАЕВ – кандидат технических наук, руководитель группы автоматизации верификации СФ-блоков и СнК. Сфера научных интересов: автоматизация верификации цифровых СФ-блоков и СнК.

Artyom Valerievich NIKOLAEV – Candidate of Technical Sciences, Head of the Verification Automation Group. Research interests: digital IP and SoC verification automation.

Евгений Андреевич ПРОКОПЬЕВ – техник АО НПЦ "ЭЛВИС", студент бакалавриата кафедры МПСУ НИУ МИЭТ. Сфера научных интересов: функциональная верификация СнК, информационные технологии.

Evgeny Andreevich PROKOPEV – technician at ELVEES RnD Center, JSC, student-bachelor of NRU MIET. Research interests: functional verification, computer science.

Федор Михайлович ПУТРЯ – кандидат технических наук, начальник отдела верификации. Сфера научных интересов: Многоядерные системы на кристалле (СнК), функциональная верификация, автоматизация, анализ производительности.

Fedor Mikhailovich PUTRYA – Candidate of Technical Sciences, Head of the Verification Department. Research interests: Multicore SoC, functional verification, automation, performance analysis.

Булат Намсараевич ЦЫРЕНЖАПОВ – техник АО НПЦ "ЭЛВИС", студент бакалавриата кафедры НМСТ НИУ МИЭТ. Сфера научных интересов: функциональная верификация СнК, информационные технологии.

Bulat Namsaraevich TSYRENZHAPOV – technician at ELVEES RnD Center, JSC, student-bachelor of NRU MIET. Research interests: functional verification, computer science.