Перенос тестовых сценариев между этапами моделирования СБИС «Система-на-кристалле» и этапом функционального контроля

Андрей Андрианов (andrianov@module.ru)

В статье рассмотрена проблема переноса и запуска программных тестовых сценариев, разработанных на этапах моделирования СБИС- и ПЛИС-прототипирования на реальную микросхему. Рассматриваются особенности последующего использования этих тестов в ходе испытаний готовых микросхем, дальнейшей разработки программного обеспечения, а также дальнейшей поддержки в виде набора для разработчика (SDK).

Введение

В связи с ростом сложности современных СБИС класса «Система на кристалле» встаёт вопрос сокращения временных затрат на всех этапах проектирования. Одним из таких этапов является тестирование изготовленной микросхемы. После производства образцы микросхем должны пройти ряд проверок, прежде чем могут быть отгружены потребителям. Тестовая последовательность может включать в себя проверки, использующие встроенную логику тестирования (DFT) в тестовом режиме и тесты, проводимые в функциональном режиме, программные или смешанные тесты, использующие специализированную аппаратную логику, но запускаемые программно, например BISR-тесты внутренней памяти [1, 2].

Основная масса программных тестов – это тесты, предназначенные для работы без операционной системы, разработанные и запускавшиеся на этапе моделирования СБИС для её верификации. Часть этих проверок, имитирующих работу микросхемы в реальных сценариях применения, может быть использована далее для проверки работоспособности микросхемы в различных условиях эксплуатации и в качестве примеров, демонстрирующих работу с различными устройствами СБИС для разработчиков.

В статье описан опыт обеспечения переносимости программных тестовых сценариев между окружением моделирования СБИС и реальной микросхемой, который был применён при разработке СБИС 1888ТХ018 [3], 1888ВС048 и 1888ВС058. Отдельная часть статьи описывает особенности реализации начального загрузчика, предоставляющего возможность пакетного запуска тестов, как под управлением внешней системы контроля, так и без неё.

Обеспечение переносимости тестовых сценариев между верификационным окружением и реальной микросхемой

Задача разработки и отладки программных сценариев на модели СБИС обладает особой спецификой. Рабочее окружение характеризуется очень небольшой скоростью выполнения, которая может варьироваться в зависимости от особенностей верификационного окружения, используемых версий ПО для моделирования, а также размера моделируемой СБИС. Зачастую в этом случае отсутствуют полноценные средства отладки, что затрудняет проектирование и отладку сложных программных абстракций.

При разработке и верификации СБИС 1888ВС048 и 1888ВС058 для обеспечения переносимости верификационных сценариев и упрощения разработки были использованы следующие основные подходы:

- набор программных тестов был выделен в отдельный подпроект, который допускает сборку и поддержку отдельно от проектов СБИС и в дальнейшем может быть передан разработчикам для работы с готовой микросхемой [4];
- набор программных тестов предусматривает сборку в нескольких оптимизированных под разные окружения конфигурациях (реальная микросхема, модель СБИС, модель IP блока);
- из всех программных тестов была выделена библиотечная часть, которая может быть использована для верификации всех последующих микросхем, содержащих этот IP-блок, а также при разработке ПО для данной микросхемы;
- для обеспечения работы некоторых функций (обработки прерываний и исключений, работы с TLB, GPIO, таймерами, реализации задержек) были разработаны «легковесные» абстракции, позволяющие разрабатывать код, который может переноситься между различными микросхемами и процессорными архитектурами путём перекомпиляции, без внесения каких-либо других изменений (см. рис. 1);
- для тестирования программных абстракций и независимых от аппаратуры алгоритмов была добавлена возможность компиляции набора библиотек и независимых от аппаратного обеспечения тестов в виде приложений для ПК. В этом случае вместо кода для работы непосредственно с аппаратурой используются специальные «заглушки». Этот подход позволяет существенно ускорить разработку и тестирование ряда критических компонентов, не взаимодействующих с аппаратурой;
- для динамического управления памятью используется специализированный аллокатор памяти [5], обеспечивающий поддержку нескольких «куч», из которых производится выделение памяти;
- для обеспечения возможности пошаговой отладки на RTL-модели СБИС [3] был задействован gdb-сервер OpenOCD совместно с разработанным верификационным компонентом, работающим по протоколу remote bitbang.

При компиляции пакета тестового ПО для системы сборки должны быть заданы две входные переменные:

- платформа (микросхема, для которой будет производиться компиляция тестов, ПЛИС-прототип(ы) или ПК);
- тип сборки (сборка для моделирования или сборка для реальной микросхемы).

На основе анализа их значений производится выбор набора низкоуровневых драйверов, которые будут включены в данную сборку.

Для повышения производительности моделирования при работе на модели СБИС были применены следующие подходы:

- медленные функции стандартной библиотеки языка C (printf, memcpy, memset и т.п.), не генерирующие непосредственно тестовое воздействие, были заменены реализацией, выполняющей запрошенное действие на стороне верификационного окружения при моделировании [6];
- для разработки и отладки ряда тестовых сценариев использовалось гибридное верификационное окружение [7];
- часть тестов разрабатывалась и отлаживалась на ПЛИС-прототипе, после чего переносилась на модель СБИС. Часть тестов разрабатывалась с использованием высокоуровневого скриптового языка lua [8, 9, 10];
- активно задействован механизм контрольных точек моделирования (для пакета ПО Cadence Insisive/Xcellium) [7, 11];
- при сборке модели СБИС использовалась многопоточная компиляция проекта, а также функционал MSIE пакета Cadence Incisive/Xcellium [12, 13, 14]. Данный подход позволяет сократить время на подготовку модели СБИС, разбивая проект на отдельные части, обрабатывающиеся параллельно. Это позволило получить значительное ускорение на начальном этапе сборки при полноценном использовании ресурсов современных многоядерных процессоров. В случае изменения исходного кода требуется повторная сборка только той части, куда было внесено изменение, что позволяет дополнительно сократить время инкрементальных сборок проекта.

Для некоторых блоков последовательность инициализации при работе на модели СБИС и реальной микросхеме различается (например, PCI Express, DDR-память и т.п.). Для проверки работоспособности ряда блоков используются упрощённые модели оконечных устройств (например, модели SPI Flash, SD-карт и т.п.). Они не обладают полной функциональностью или имеют иную последовательность инициализации.

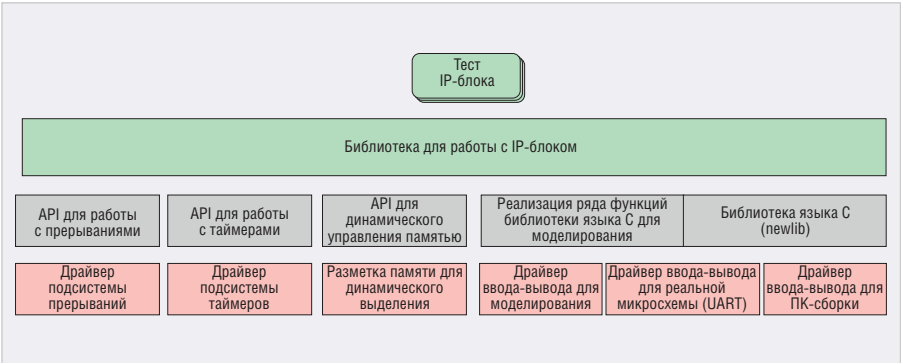

Рис. 1. Архитектура пакета тестового ПО

Код для работы с такими устройствами разрабатывался и отлаживался на ПЛИС-прототипе, после чего проверялся на RTL-модели, и на основе этого вносились изменения как в верификационное окружение, так и в сам код для обеспечения его работоспособности во всех сценариях применения.

Роль системы сборки и тестирования в обеспечении переносимости верификационных сценариев

Система сборки играет важную роль в обеспечении переносимости верификационных сценариев. К тому же именно она должна обеспечить максимально типовой порядок работы с проектом во всех окружениях. При моделировании СБИС и разработке IP-блоков, система сборки должна обеспечивать поддержку многопоточной компиляции и элаборации проекта, а также обеспечивать поддержку использования MSIE для сокращения времени повторной сборки проекта [15]. Так как один и тот же блок может применяться в различных проектах, то также необходима поддержка разбиения большого проекта на подпроекты на уровне системы сборки. Дополнительно снизить время повторных сборок проекта позволяет использование системы кэширования результатов компиляции (ccache).

При работе с итоговой микросхемой система сборки должна поддерживать интеграцию с современными интегрированными средами разработки, такими как Eclipse, Code::Blocks и т.п, и при этом не зависеть от пакета ПО для моделирования СБИС, который может отсутствовать у конечных разработчиков решений на базе данной микросхемы. Для обеспечения регулярного автоматизированного тестирования проекта и сбора метрик о состоянии проекта система сборки и тестирования проекта должна иметь интеграцию с современными системами непрерывной интеграции, такими как Jenkins.

Для решения вышеобозначенных задач был выбран набор инструментов cmake [12, 15], а для запуска тестов – ctest. Это позволило обеспечить унификацию процесса запуска тестов как на модели, так и на готовой микросхеме, а также использовать существующие решения для анализа результатов автоматизированных тестовых прогонов: Jenkins xUnit plugin, Jenkins test results analyzer plugin, Jenkins code coverage plugin и другие.

На уровне системы сборки проекта реализован запуск окружения для пошаговой отладки при работе на модели СБИС [3], дополнительные сценарии для сбора покрытия и профилирования кода [16], подстановка соответствующих опций компилятора и вызов соответствующих инструментов после окончания тестового прогона, а также сборка документации при помощи системы документирования исходного кода doxygen.

ПО начальной загрузки СБИС 1888ВС048 и 1888ВС058

ПО начальной загрузки СБИС (ROM-загрузчик) – ещё один важный компонент, которому уделяется неоправданно мало внимания в публикациях. ПО начальной загрузки СБИС 1888ВС048 и 1888ВС058 может не только обеспечивать начальную инициализацию,

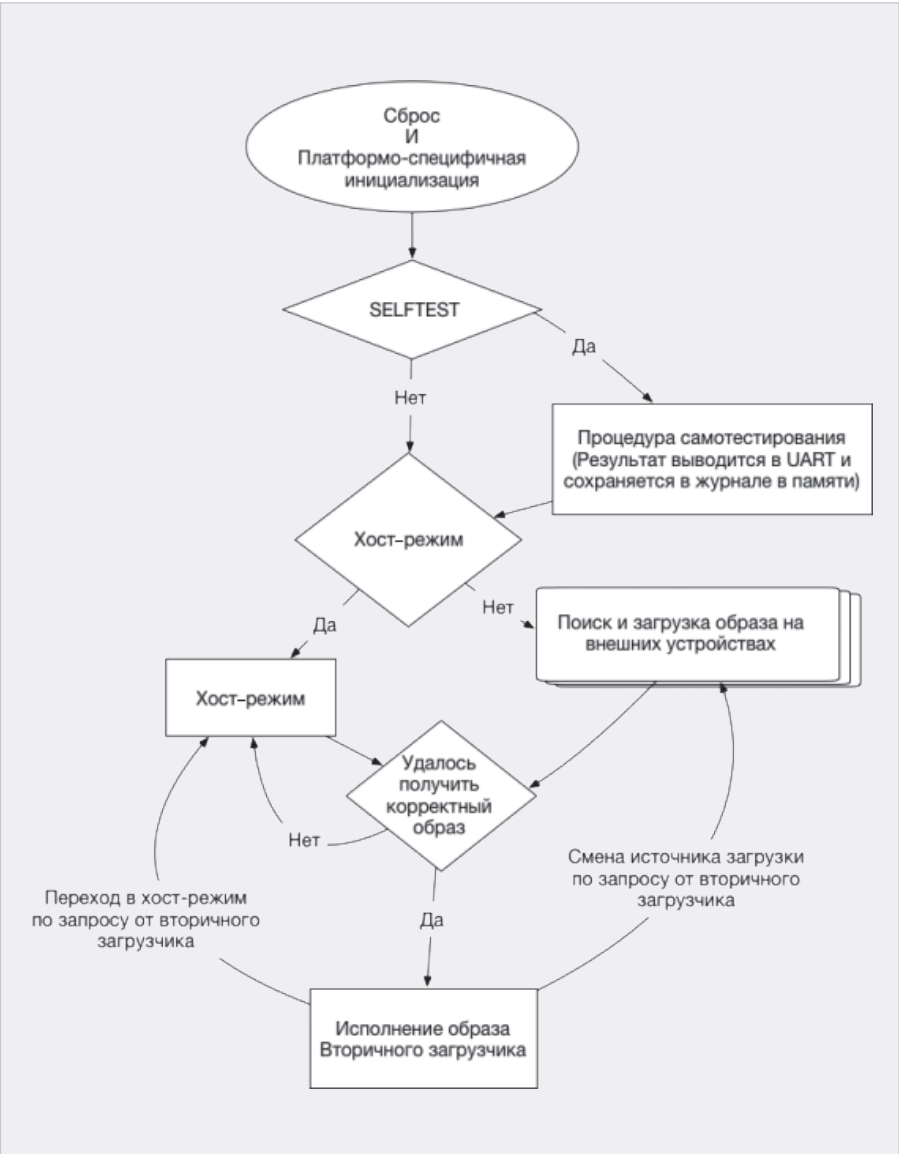

Рис. 2. Алгоритм работы ROM-загрузчика

загрузку, проверку целостности и выполнение пользовательского кода на микросхеме, но и упрощает диагностику ошибок на этапе загрузки и в разрабатываемом пользователями коде. Дополнительно ROM-загрузчик обеспечивает разработчиков решением для внутрисхемного программирования внешней (по отношению к микросхеме) памяти, предоставляет программный интерфейс для пользовательских программ, что позволяет сократить объём необходимой встроенной SRAM-памяти микросхемы. В ROM-загрузчик 1888ВС048 включена также процедура самотестирования и обеспечен интерфейс взаимодействия с внешней системой для пакетного запуска тестовых сценариев.

Процесс загрузки СБИС 1888ВС048 и 1888ВС058 происходит в несколько этапов:

- при сбросе микросхемы начинает работать встроенный ROM-загрузчик. Он определяет внешнее устройство, с которого будет производиться загрузка, на основе внутренней логики или на основании состояния внешних выводов, производит копирование образа загружаемой программы во внутреннюю SRAM-память и проверку её целостности, после чего передаёт управление на точку входа;
- загруженная программа может являться сама по себе простой программой, использующей данную микросхему (например, тесты), или частью загрузчика операционной системы, например u-boot spl. В этом случае задача данной программы – инициализировать внешнюю динамическую память, установленную на плате, после чего загрузить туда из внешней ПЗУ образ u-boot и передать управление на точку входа. Типичный размер spl – несколько десятков килобайт, в зависимости от объёма кода;
- загрузчик операционной системы u-boot производит чтение сжатого образа ядра операционной системы из файловой системы во внешней ПЗУ, производит проверку целостности и передаёт управление в ядро операционной системы. В случае использования OS Linux, помимо ядра необходим также скомпилированный файл описания устройств (Device Tree Blob), который также загружается в динамическую память прежде, чем управление передаётся в код ядра. Также может присутствовать файл initrd (initial ramdisk), содержащий временную корневую файловую систему, которая используется до момента монтирования основной корневой файловой системы;
- ядро операционной системы производит инициализацию оборудования и монтирует корневую файловую систему.

Основная масса тестов компилируется в виде вторичного загрузчика, который загружается при помощи встроенного ROM-загрузчика. Схематично алгоритм работы ROM-загрузчика представлен на рисунке 2.

Диагностический вывод. Стандартным средством отладки для большинства СБИС является интерфейс асинхронного приёмника-передатчика (UART). Наиболее широко применяемые во встраиваемых решениях загрузчики операционной системы, такие как u-boot, barebox и redboot, ориентированы на использование терминала, подключённого через интерфейс асинхронного приёмника-передатчика как основного интерфейса взаимодействия с пользователем. Ядро операционной системы Linux при работе во встраиваемых системах аналогичным образом чаще всего использует последовательный порт для отображения отладочных сообщений и дальнейшего входа в систему.

Подробная диагностика на ранних этапах работы ROM-загрузчика может серьёзно облегчить выявление проблем с загрузкой. Дополнительно ROM-загрузчик может устанавливать в самом начале своей работы обработчики векторов исключений процессора. Таким образом, если произойдёт исключение в пользовательской программе, можно предоставить пользователю подробную информацию об ошибке, вплоть до

программной трассировки стека вызовов, которые привели к этой ошибке. Пример вывода загрузчика 1888ВС048 в последовательный порт представлен на рисунке 3.

Проверка целостности и совместимости загружаемого кода. Образ вторичного загрузчика содержит заголовок, в котором указана служебная информация о загружаемой программе: порядок байт, адрес точки входа, размер, идентификационный номер микросхемы, для которой скомпилирована программа, и контрольные суммы CRC32 для заголовка и данных. Это позволяет гарантировать, что загружаемый код не содержит ошибок, а также предназначается именно для этой микросхемы. В заголовке также зарезервировано место, куда ROM-загрузчик поместит указатель на структуру, содержащую информацию о том, с какого устройства была произведена загрузка.

Инструменты для аварийной загрузки. В случае, если загрузиться не удалось ни с одного устройства, или необходимо произвести обновление ПО, записанного во внешнее ПЗУ, в ROM-загрузчике предусмотрен аварийный (хост) режим. Этот режим позволяет передать образ вторичного загрузчика, используя внешний интерфейс. Для СБИС 1888ВС048 и 1888ВС058 этим интерфейсом является UART, который используется также для отладочного вывода. Для 1888ВС058 этим каналом дополнительно может быть интерфейс EMI или канал Ethernet (EDCL). Для передачи бинарных данных по каналу UART используется протокол xmodem. Наличие этого режима позволяет загрузить вторичный загрузчик, реализующий процесс программирования внешней ПЗУ.

Данный режим также позволяет производить автоматизированное тестирование на платах, при условии наличия программируемого источника питания (APC) или при наличии на отладочной плате схемы управления питанием. Этот сценарий загрузки используется при проведении автоматизированного тестирования на реальной микросхеме.

ROM API для вторичного загрузчика. Вторичный загрузчик (spl) может получать от ROM-загрузчика информацию о том, какие компоненты процедуры самотестирования завершились на данной микросхеме с ошибкой, информацию о версии микросхемы и версии ROM-загрузчика. Наиболее полезной и

```
    RC Module's                 __               __
   _______  ______ ___  / /_  ____  ____  / /_
  / ___/ / / / __ `__ \/ __ \/ __ \/ __ \/ __/
 / /  / /_/ / / / / / / /_/ / /_/ / /_/ / /_
/_/   \__,_/_/ /_/ /_/_.___/\____/\____/\__/
basis | Production | HEAD-c13a6a3
--- RumBoot Configuration ---
Force Host Mode: disabled
Selftest:        disabled
EDCL/RMAP:       enabled
UART speed:      115200 bps
Max SPL size:    131072 bytes
---          ---          ---
boot: SDIO0 (CD: GPIO0_3): Failed to enable or media not present
boot: SPI0 (CS: internal): Reading image from offset 0
boot: SPI0 (CS: internal): Error: Bad Magic (1)
boot: SPI0 (CS: GPIO0_5): Reading image from offset 0
boot: SPI0 (CS: GPIO0_5): --- Boot Image Header ---
boot: SPI0 (CS: GPIO0_5): Magic:            0xb01dface
boot: SPI0 (CS: GPIO0_5): Header version:   2
boot: SPI0 (CS: GPIO0_5): Chip Id:          3
boot: SPI0 (CS: GPIO0_5): Chip Revision:    1
boot: SPI0 (CS: GPIO0_5): Data length:      39140
boot: SPI0 (CS: GPIO0_5): Header CRC32:     0x65c22a5f
boot: SPI0 (CS: GPIO0_5): Data CRC32:       0xcf79bf84
boot: SPI0 (CS: GPIO0_5): ---         ---         ---
boot: SPI0 (CS: GPIO0_5): Image validated, executing...
This is the first test. Arming GPIOs!
boot: SPI0 (CS: GPIO0_5): Back in rom, code 0, will now boot from next device
```

Рис. 3. Пример отладочного вывода ROM-загрузчика в последовательный порт

необходимой является информация о том, с какого устройства была произведена загрузка, а также набор функций для чтения дополнительных данных (например, u-boot) с этого устройства, которое выполняется кодом, находящимся в ROM.

Для обеспечения этого взаимодействия ROM-загрузчик использует для хранения переменных только область стека, а при компиляции генерируется специальный заголовочный файл, включение которого в пользовательскую программу позволяет вызывать имеющиеся в масочном ПЗУ функции.

Поддержка объединения тестов в цепочки. Этот режим позволяет записать несколько образов вторичного загрузчика во внешнюю флеш-память и выполнять последовательно без сброса. После выполнения загруженной программы из цепочки, происходит возврат в ROM-загрузчик. В зависимости от кода возврата будет выполнен переход в хост-режим загрузки, попытка загрузиться с другого источника, или будет считан следующий образ вторичного загрузчика из той внешней памяти, откуда была произведена загрузка.

Такой подход позволяет записать набор существующих тестов без ручного объединения их в один тест и загружать их последовательно из внешней памяти. Количество тестов в такой цепочке ограничено только объёмом подключенной внешней памяти. Дополнительно это позволяет разделять готовую прошивку платы на несколько независимых компонентов для удобства разработки и отладки. Например, первый образ вторичного загрузчика инициализирует динамическую память, которая установлена на плате и может отличаться от платы к плате, и записывает уникальный MAC-адрес в регистры Ethernet контроллера. Следующий за ним вторичный загрузчик является spl-компонентом загрузчика u-boot, который в этом случае не содержит кода инициализации динамической памяти и может являться общим для всех плат на базе этой микросхемы.

Этот сценарий отлично подходит для верификации уже изготовленных микросхем при испытаниях.

Заключение

Описанный выше подход оправдал себя при разработке и верификации СБИС 1888ТХ018, 1888ВС048 и 1888ВС058, так как позволил сформировать к моменту прихода микросхемы предварительную версию набора раз-

работчика, а также обеспечил повторное использование кода при разработке и поддержке нескольких проектов. Дополнительным преимуществом стало создание единого инструментария для отладки и начального программирования ПЗУ, что позволяет инженерам работать с целым рядом микросхем.

Литература

1. *Щигорев Л.А.* Организация саморемонта блоков статической оперативной памяти с резервными элементами// Проблемы разработки перспективных микро- и наноэлектронных систем-2016 Сб. трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2016. Часть III. С. 178-185.
2. *Щигорев Л.А.* Применение шины диагностики в задаче саморемонта блоков статической оперативной памяти // Нано- и микросистемная техника. 2018, Т. 20, № 2. С. 98-106.
3. *Андрианов А.В.* Реализация возможности пошаговой отладки при отладке тестовых сценариев на модели СБИС СнК. Труды НИИСИ РАН, т. 8, № 3, 2018, С. 56-60.
4. *Андрианов А.В.* Методы обеспечения переносмости тестовых сценариев между различными верификационными окружениями. Сборник трудов конференции МЭС-2018, т. 2 С. 79-84.
5. *Андрианов А.В., Мушкаев С.В.* Гибридый метод аллокации массивов памяти в аппаратных платформах с разветвленной структурой памяти на базе процессора NeuroMatrix® DSP, Сборник трудов конференции МЭС-2016, т. 2 С. 233-236.
6. *Андрианов А.В.* Практические способы оптимизации процесса регрессионного тестирования СБИС СнК. Сборник трудов 4-й международной конференции “Микроэлектроника – 2018”.
7. *Андрианов А.В., Шагурин И.И.* Методика гибридной верификации СБИС “Система-на-Кристалле. «Датчики и системы». 2018 г., № 2, С. 14-18.
8. *Андрианов А.В.* Верификция IP-блоков RTL-модели и ПЛИС-прототипе при помощи высокоуровневого языка lua. Тезисы докладов международной научно-технической конференции “Электроника – 2015”, С. 102-103
9. *Андрианов А.В.* Протоипирование кода драйверов OS Linux в пространстве пользователя с использованием языка высокого уровня lua. Сборник трудов конференции МЭС-2014, т. 2 С. 25-28.
10. *Андрианов А.В.* Использование скриптового языка lua при ПЛИС-прототипировании аппаратных блоков при работе без операционной системы. Труды XIV Росиийской научно-технической конференции “Новые иинформационные технологии в устройствах связи и управления”, 2015, С. 268-269.
11. *Андрианов А.В.* Использование семейства инструментов CMake для моделирования проектов сложных СБИС в среде Cadence Incisive. Труды НИИСИ РАН, т. 7, № 4, 2017.
12. Cadence Incisive Enterprise Simulator Refererence Manual, Cadence 2015.
13. Reducing Snapshot Creation Turnaround for UVM/SV Based TB Using MSIE Approach, Cadence User Conference 2015, https://www.cadence.com/downloads/cdnlive/in/2015/VER2.pdf.
14. CMake: The Cross Platform Build System, Tanner Lovelace, Linux Magazine, Июль 2006 http://clubjuggler.livejournal.com/138364.html.
15. *Андрианов А.В.* Измерение покрытия кода при работе без операционной системы встроенными средствами компилятора gcc. Сборник докладов конференции DSPA-2018, т. 2. С. 657-661.

19-я МЕЖДУНАРОДНАЯ ВЫСТАВКА ЭЛЕКТРОНИКИ

ChipEXPO-2021

КОМПОНЕНТЫ | ОБОРУДОВАНИЕ | ТЕХНОЛОГИИ

ВЫСТАВКА ПРОЙДЕТ

14-16.09

В ТЕХНОПАРКЕ ИННОВАЦИОННОГО ЦЕНТРА

СКОЛКОВО

ТЕМАТИЧЕСКИЕ ЭКСПОЗИЦИИ:

- Экспозиция Департамента радиоэлектронной промышленности Минпромторга России, включая:
  - экспозицию предприятий, являющихся изготовителями изделий, включенных в единый реестр российской радиоэлектронной продукции (Постановление Правительства РФ №878)
  - экспозицию разработок, созданных в рамках государственной программы «Развитие электронной и радиоэлектронной промышленности на 2013-2025 годы» (Постановление Правительства РФ №109)
  - экспозицию разработок, обеспечивающих выполение приоритетных национальных проектов.
- Дивизионы кластера «Радиоэлектроника» ГК «Ростех»
- Квалифицированные поставщики ЭКБ
- Участники конкурса «Золотой Чип»
- Стартапы в электронике
- Консорциумы и дизайн-центры по электронике
- Корпорация развития Зеленограда

ОФИЦИАЛЬНАЯ ПОДДЕРЖКА:

Sk Сколково · Государственная Дума · Минпромторг России · Ростех · Росэл · Российская академия наук

ОРГАНИЗАТОРЫ:

ЗАО «ЧипЭКСПО» Москва,121351, ул. Ярцевская, д.4. Тел.: +7 (495) 221-50-15
E-mail: info@chipexpo.ru http://www.chipexpo.ru