Для цитирования: Фролов П.В. Система интеграции инженерных тестов на основе статического распределения ресурсов // Вопросы радиоэлектроники. 2018. № 2. С. 76–80.
УДК 004.414.23

П.В. Фролов[1, 2, 3]

[1] ПАО «ИНЭУМ им. И.С. Брука», [2] АО «МЦСТ», [3] МФТИ (ГУ)

СИСТЕМА ИНТЕГРАЦИИ ИНЖЕНЕРНЫХ ТЕСТОВ НА ОСНОВЕ СТАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ РЕСУРСОВ

Существует практическая необходимость интеграции отдельных решений для системной логической верификации микропроцессоров. В статье представлен подход, позволяющий формировать системные тесты из отдельных подтестов путем распараллеливания их исполнения на разных вычислительных ядрах верифицируемого многоядерного микропроцессора. При этом распределение доступных системных ресурсов осуществляется статически на основе заранее сформированных ресурсных требований подтестов. Полученные значения ресурсных параметров передаются в подтесты на стадии компоновки системного теста. Программная среда исполнения при запуске моделирования осуществляет инициализацию системных ресурсов и передает управление подтестам. В статье описаны структура программного комплекса, реализующего представленный подход, алгоритм формирования системного теста и используемый для этого инструментарий. Приведены основные сценарии применения на примере маршрута логической верификации микропроцессора «Эльбрус-8С2».

Ключевые слова: *системная верификация, генерация тестов, многоядерность.*

Введение

В настоящее время задачи верификации и тестирования вычислительных систем требуют разработки тестовых программ, исполняющихся на нескольких процессорных ядрах и реализующих различные сценарии – это вычислительные задачи, работа с кэш-памятью, контроллерами памяти, графическими сопроцессорами, контроллерами ввода-вывода, модулями управления питанием и другим компьютерным оборудованием.

Как правило, разработка средств системной верификации (направленных тестов и генераторов псевдослучайных тестов) для тестирования отдельных устройств ведется несколькими инженерами [1, 2]. При этом, естественно, возникает необходимость интеграции отдельных решений, в совокупности позволяющих проверять одновременную работу разных модулей системы на базе общих наличных ресурсов. В частности, такое требование ставит проблему эффективного использования аппаратных ресурсов при исполнении тестовых программ на ПЛИС-прототипе верифицируемого микропроцессора [3]. Например, тесты, предназначенные для верификации устройств вычислительного ядра, как правило, задействуют только одно ядро, т.е. при их исполнении на прототипе, реализующем многоядерную модель микропроцессора, все вычислительные ядра, кроме одного, простаивают. Такого рода ситуации возникают и относительно других ресурсов тестируемого микропроцессора. В статье описан практически опробованный подход, дающий возможность в существенной степени решить подобного рода проблемы.

Система интеграции инженерных тестов

В [4] было предложено решение, позволяющее интегрировать различные инструменты системной верификации путем внедрения общей структуры генераторов случайных тестов. Однако оно фактически предполагает разработку новых генераторов в рамках предложенной концепции, ибо адаптация уже внедренных генераторов достаточно трудоемка и требует высокой квалификации от инженеров-верификаторов.

Более практичной представляется возможность реализации одного системного теста из подтестов, параллельно запускаемых на разных вычислительных ядрах. Для корректного исполнения подтестов необходимо обеспечить отсутствие конфликтов между ними по доступу к общим ресурсам. Чтобы не усложнять логику исполнения теста, это целесообразно осуществлять статически, т.е. на этапе сборки системного теста, для чего необходимо описывать объем доступных системных ресурсов и ресурсные требования каждого подтеста.

В работе предлагается система интеграции отдельных тестов в один системный тест, в которой поддерживается этот принцип. С точки зрения реализации эта система состоит из трех структурных элементов:

1. Статический распределитель доступных системных ресурсов между подтестами.
2. Среда исполнения подтестов – вспомогательные подпрограммы системного теста, осуществляющие инициализацию тестируемого системного компонента, передачу управления подтестам и корректное завершение их исполнения.
3. Компоновщик – набор скриптов, формирующих итоговый системный тест из подтестов и кода среды исполнения.

Передача значений ресурсов в подтесты осуществляется через отладочные символы. Неопределенные символы в исходном коде подтеста объявляются как внешние (extern) – финальное значение они получают уже при компоновке системного теста. Таким образом, система интеграции позволяет компоновать вместе подтесты, реализованные как на языке ассемблера, так и на языках программирования высокого уровня, таких как C или C++.

Статический распределитель системных ресурсов

Этот компонент принимает на вход описание доступных ресурсов системы и набор описаний ресурсных требований для каждого подтеста в json-формате.

На основе анализа разработанных генераторов был составлен перечень общих ресурсов, минимально необходимый для учета при компоновке подтестов:

- Множество доступных вычислительных ядер.
- Множество физических адресов оперативной памяти системы.
- Множество векторов прерываний.
- Внешние по отношению к вычислительному ядру устройства системы.

Описание ресурсных требований подтеста должно содержать количество используемых им вычислительных ядер и точки входа для каждого из них. Точки входа задаются отладочными символами, соответствующими стартовым адресам подтестов. Задача распределителя – поставить в соответствие каждой из точек входа номер доступного вычислительного ядра.

Множество доступных физических адресов оперативной памяти в общем случае состоит из нескольких диапазонов, определяющих маршрутизацию запросов в системе. Задача распределителя состоит в размещении в доступных диапазонах исполняемого кода среды исполнения и подтестов, а также областей с рабочими данными с учетом дополнительных требований, указанных в описании ресурсных требований. Исполняемый код располагается в секциях объектных файлов, размер которых определяется при предварительной компиляции подтеста или среды исполнения. Области рабочих данных описываются отладочными символами нижней границы и размером; возможно наложение дополнительных требований на размещение областей, таких как ограничение целевого адресного диапазона, выровненность границы, взаимное расположение отдельных областей и др. Таким образом, в качестве результата распределения адресного пространства оперативной памяти выступают два набора пар: {(подтест, секция) → физический адрес} для объектного кода и {(подтест, отладочный символ старта области) → физический адрес} для рабочих данных.

Векторы прерываний – это целые числа в определенном диапазоне. В случае использования подтестом прерываний в описании ресурсных требований указывается необходимое количество векторов. Распределение множества V векторов прерываний между подтестами заключается в получении пар соответствия (подтест → подмножество векторов V_i), где V_i – попарно непересекающиеся подмножества V.

Для разрешения конфликта по использованию внешних устройств необходимо, во-первых, запретить компоновку тестов, работающих с одним устройством, во-вторых, при динамическом отображении внутренних регистров устройств в программно-видимую область памяти (Memory Mapped Input/Output) обеспечить отсутствие пересечений соответствующих диапазонов (как и с оперативной памятью).

Среда исполнения подтестов

Основное назначение среды – это начальная инициализация тестируемой системы в соответствии с описанием доступных системных ресурсов: настройка системных маршрутизаторов в соответствии с заданными диапазонами физических адресов оперативной памяти и областей, предназначенных для отображения внутренних регистров устройств (например, MMIO-диапазон для PCI-устройств). При инициализации системы запускаются требуемые системным тестом вычислительные ядра, а среда исполнения производит дополнительную настройку запрошенных подтестами специфических ресурсов (например, программного стека), осуществляет синхронизацию ядер, после которой передает управление на точки входа подтестов. Для вынесения вердикта об успешном завершении системного теста (означающего отсутствие найденных аппаратных ошибок) среда исполнения дожидается завершения всех подтестов. В случае обнаружения ошибки подтестом среда исполнения предоставляет

средства передачи диагностической информации пользователю (код завершения, отладочную печать при исполнении на прототипе, реализованном на ПЛИС).

Окружение среды исполнения предоставляет дополнительные возможности для реализации специальных тестовых сценариев, связанных с нарушением непрерывного исполнения подтестов или с работой подтеста на фоне внешней по отношению к нему активности (например, потока DMA-транзакций). Эти сценарии реализуются с помощью библиотеки стандартных подтестов, входящих в состав среды исполнения. Например, реализована посылка прерываний вычислительному ядру, исполняющему подтест, и предоставлен специальный код обработки этих прерываний с возвратом в вызывающий контекст. В подтесте должен быть поддержан вызов этого кода.

Помимо стандартных подтестов в состав среды исполнения входит расширяемая библиотека стандартных функций, используемых в первую очередь при отладке аппаратуры. Для получения доступа к ним в исходный код подтеста нужно добавить директиву включения соответствующего заголовочного файла.

Конфигурация дополнительных возможностей среды исполнения осуществляется в одном json-файле вместе с описанием доступных системных ресурсов. Таким образом, при распределении ресурсов учитываются требования самой среды исполнения – память, необходимая для размещения ее исполняемого кода, а также ресурсные требования стандартных подтестов при их включении в системный тест.

Компоновщик системного теста

Задача компоновщика – собрать системный тест, обеспечив передачу в подтесты и среду исполнения значений ресурсов, выделенных согласно требованиям. Для компоновки системного теста используются стандартные утилиты из пакета binutils [5], позволяющие осуществлять разнообразные манипуляции с отладочными символами и предоставляющие возможность гибкого управления процессом сборки. Компоновщик обеспечивает заданную последовательность преобразований объектных файлов подтестов, а также формирование вспомогательных файлов конфигурации этих преобразований, таких как списки символов и скрипт редактора связей.

В результате распределения системных ресурсов согласно требованиям для каждого из подтестов и среды исполнения формируются списки пар «отладочный символ → значение» и «секция объектного кода → физический адрес». После компиляции отладочные символы объектных файлов подтестов, описывающие значения выделенных ресурсов, остаются неопределенными – компоновщик преобразует их в глобальные и добавляет префикс, уникальный для подтеста. Остальные отладочные символы должны быть определенными, они преобразуются в локальные. Такое преобразование сохраняет символьную разметку целевого объектного файла системного теста, что важно для отладки

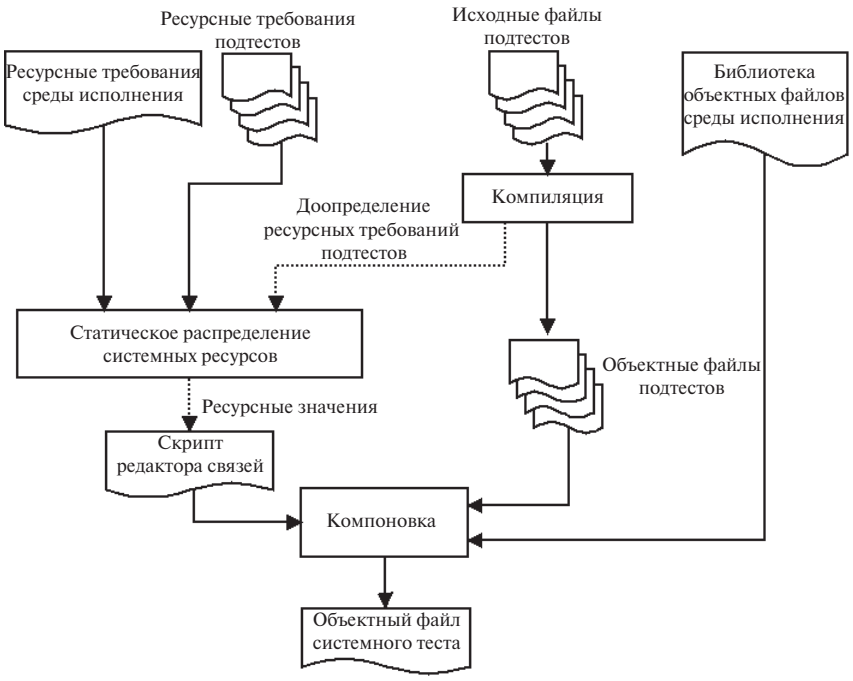

Рисунок. Структура системы интеграции системных тестов на основе статического распределения ресурсов

возможных аппаратных ошибок и в то же время обеспечивает отсутствие конфликтов по именам символов при итоговой компоновке.

Итоговая сборка, приводящая к результату, представленному на рисунке, осуществляется редактором связей, на вход которому вместе с объектными файлами подтестов и среды исполнения передается формируемый компоновщиком скрипт [6], в котором:

- через явное указание адреса в разделе SECTIONS определяется размещение секций объектного кода;
- полученные при распределении системных ресурсов значения присваиваются соответствующим отладочным символам как абсолютные.

В полученном объектном файле системного теста формата .elf адреса соответствуют физическим адресам тестируемой системы; для дальнейшего исполнения в целевом тестовом окружении (на RTL-модели или прототипе, реализованном на ПЛИС) используется бинарный формат.

Типовые сценарии применения

Предложенная система допускает несколько типовых сценариев применения в маршруте логической верификации микропроцесорных систем.

Очевидный подход – интеграция отдельных тестов – неудобен ввиду относительно высокой трудоемкости: каждый подтест в общем случае требует своего описания ресурсных требований. С другой стороны, привнесение случайного характера в алгоритм статического распределения ресурсов позволяет получить целый спектр тестовых сценариев из одного набора подтестов.

Основной сценарий применения системы интеграции – это компоновка случайных тестов, полученных с помощью генераторов, когда для определенного сценария генерации разрабатывается свое описание ресурсных требований, удовлетворяющих всем тестам данного сценария. Подобная схема была успешно применена при верификации микропроцессора «Эльбрус-8С2» с помощью прототипа, реализованного на ПЛИС.

Еще один возможный сценарий применения заключается в совместном формировании генератором как теста, так и описания ресурсных требований, однако подразумевает дополнительные усилия по доработке генератора.

При верификации микропроцессора «Эльбрус-8С2» с использованием системы интеграции было модифицировано несколько генераторов случайных ассемблерных тестов для проверки отдельных устройств вычислительного ядра. Доработки оказались минимальными, практически потребовалось заменить включаемые в тест стандартные файлы с процедурой инициализации системы на заголовочные файлы среды исполнения и перевести работу с памятью на адреса, определяемые метками. Для поддержки тестовых сценариев, связанных с получением вычислительным ядром внешних прерываний, потребовалась дополнительная доработка процедуры обработки прерываний.

Заключение

В статье описан принцип и основные детали реализации системы интеграции, дающей возможность оптимизировать одновременное тестирование базовых компонентов моделируемых вычислительных средств за счет более полного и эффективного использования их ресурсов. Применение системы позволило увеличить число случайных тестов, исполненных прототипом на ПЛИС, за счет полного использования доступных вычислительных ядер. Были найдены логические ошибки, связанные с сохранением/восстановлением контекста исполнения подтеста при обработке внешних прерываний.

СПИСОК ЛИТЕРАТУРЫ

1. Мешков А. Н., Рыжов М. П., Шмелев В. А. Развитие средств верификации микропроцессора «Эльбрус-2S» // Вопросы радиоэлектроники. 2014. Т. 4. № 3. С. 5–17.
2. Средства функциональной верификации микропроцессоров / А. С. Камкин, А. М. Коцыняк, С. А. Смолов, А. Д. Татарников, М. М. Чупилко, А. А Сортов // Труды Института системного программирования РАН. 2014. Т. 26. № 1. С. 149–200.
3. Юрлин С. В. Бычков И. Н. Прототипирование на основе ПЛИС для верификации многоядерных микропроцессоров // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2014. № 4. С. 45–50.
4. Фролов П. В. Генерация случайных тестов системного уровня для микропроцессоров с архитектурой «Эльбрус» // Вопросы радиоэлектроники. 2014. Т. 4. № 3. С. 38–46.
5. GNU Binutils. Available at: https://sourceware.org/binutils/ (accessed 07.11.2017)
6. Документация редактора связей ld. Скрипты редактора связей [Электронный ресурс]. URL: https://sourceware.org/binutils/docs/ld/Scripts.html#Scripts (дата обращения: 07.11.2017)

ИНФОРМАЦИЯ ОБ АВТОРЕ

Фролов Павел Викторович, начальник сектора, АО «МЦСТ», ПАО «ИНЭУМ им. И. С. Брука»; ассистент, МФТИ (ГУ), 119334, Москва, ул. Вавилова, д. 24, тел: 8 (499) 135-70-89, e-mail: pavel.v.frolov@mcst.ru.

For citation: Frolov P. V. System-level test integration based on static resource allocation. Voprosy radioelektroniki, 2018, no. 2, pp. 76–80.

P. V. Frolov

SYSTEM–LEVEL TEST INTEGRATION BASED ON STATIC RESOURCE ALLOCATION

There is a practical need in integration of different tools for system-level functional verificaton of microprocessors. An approach to creation of system-level multicore tests from subtests is proposed. The subtests are executed in parallel on different cores of the microprocessor verified. Allocation of system resources is made statically based on specifications of subtests resources requirements. Assignment of the resources is made during subtests mutual linking into the final system test. Special runtime environment provides resources initialization and executes the subtests. The software that implements the considered approach and the algorithm of the system test creation are described. Typical application scenarios of the approach are presented with an example of functional verification of the Elbrus-8C2 microprocessor.

Keywords: system-level verification, test generation, many-core.

REFERENCES

1. Meshkov A., Ryzhov M., Shmelyov V. The development of the verification tools of the Elbrus-2S microprocessor. *Voprosy radioelektroniki,* 2014, vol. 4, no. 3, pp. 5–17 (in Russian).
2. Kamkin A. S., Kotsynyak A. M., Smolov S. A., Tatarnikov A. D., Chupilko M. M., Sortov A. A. Tools for Functional Verification of Microprocessors. *Trudy Instituta sistemnogo programmirovaniya RAN,* 2014, vol. 26, no. 1, pp.149–200.
3. Yurlin S. V., Bychkov I. N. FPGA prototyping for functional verification of multi-core processors. *Problemy razrabotki perspektivnykh mikro- i nanoelektronnykh sistem (MES),* 2014, no. 4, pp. 45–50 (In Russian).
4. Frolov P. Random system-level test generation for Elbrus architecture microprocessors. *Voprosy radioelektroniki,* 2014, vol. 4, no. 3, pp. 38–46 (in Russian).
5. GNU Binutils. Available at: https://sourceware.org/binutils/ (accessed 07.11.2017).
6. Documentation for ld. Linker Scripts. Available at: https://sourceware.org/binutils/docs/ld/Scripts.html#Scripts (accessed 07.11.2017).

AUTHOR

Frolov Pavel, head of Sector, JSC MCST, PJSC Brook INEUM; assistant, MIPT, 24, ulitsa Vavilova, Moscow, 119334, Russian Federation, tel.: +7 (499) 135-70-89, e-mail: pavel.v.frolov@mcst.ru.