ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

СОГЛАСОВАНО
Доцент базовой кафедры
«Системное программирование»
Института системного программирования
им В.П. Иванникова РАН (ИСП РАН)
факультета компьютерных наук,
кандидат физико-математических наук

_____ А. С. Камкин
«01» апреля 2023 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы факультета
компьютерных наук
«Программная инженерия»
профессор департамента программной
инженерии, кандидат технических наук

_____ В. В. Шилов
«7» апреля 2023 г.

ПРОТОТИП ОНЛАЙН-ГЕНЕРАТОРА ТЕСТОВЫХ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРОВ

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.04-01 81 01-1-ЛУ

Исполнитель
студент группы БПИ201
_____ / М. Ю. Литвинов /
«01» апреля 2023 г.

Москва, 2023

УТВЕРЖДЕН

RU.17701729.04.04-01 81 01-1-ЛУ

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

ПРОТОТИП ОНЛАЙН-ГЕНЕРАТОРА ТЕСТОВЫХ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРОВ

Пояснительная записка

RU.17701729.04.04-01 81 01-1-ЛУ

Листов 27

Москва, 2023

СОДЕРЖАНИЕ

Москва, 2023

Москва, 2023

1. ВВЕДЕНИЕ

Наименование программы

Прототип онлайн-генератора тестовых программ для микропроцессоров.

Наименование программы на английском языке

Prototype of Online Test Program Generator for Microprocessors.

Документы, на основании которых ведется разработка

Учебный план подготовки бакалавров по направлению 09.03.04 «Программная Инженерия» и утвержденная академическим руководителем программы тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Функциональное назначение

Программа предоставляет возможности генерации и исполнения тестовых последовательностей из инструкций целевой микропроцессорной архитектуры (Instruction Set Architecture, ISA) верифицируемой системы (по умолчанию предоставляется реализация под RISC-V RV32I-совместимой модели микропроцессора), а также вынесения вердикта о корректности работы использующихся в тестах инструкций на основе сравнения результатов выполнения двух функционально эквивалентных в рамках заданных правил последовательностей.

Эксплуатационное назначение

Программа может являться компонентом более обширного комплекса программных средств для верификации микропроцессоров, ускоряя работу разработчиков микропроцессоров посредством автоматической генерации тестовых последовательностей из инструкций целевой ISA (с их непосредственным исполнением) и соответствующих им входных данных по установленным заранее принципам.

В частности, программа может применяться в отношении программируемых логических интегральных схем (ПЛИС), настроенной на работу с целевой ISA, при этом самостоятельно реализуя необходимые ей сервисы операционной системы по работе с памятью, если таковые понадобятся.

Краткая характеристика области применения

Программа, предоставляющая возможность автоматической генерации тестовых последовательностей из инструкций, составляющих целевую ISA некоторого семейства микропроцессоров, а также возможности непосредственного исполнения сгенерированных тестовых последовательностей (динамическое/функциональное онлайн-тестирование) и сравнения результатов выполнения каждой тестовой последовательности с ее несколько измененной в рамках задаваемых правил версией.

Поставленная задача выполняется в рамках верификации как одного из этапов проектирования микропроцессоров и решение поставленной задачи имеет применение в автоматизации процесса системной верификации микропроцессоров[6], модели которых могут быть реализованы в сверхбольших интегральных схемах (СБИС), в упомянутых ранее ПЛИС, а также запущены через программные эмуляторы. В решении задачи применяются методы псевдослучайной генерации входных данных, использующихся в инструкциях микропроцессоров в рамках генерируемых тестовых последовательностей.

В силу сложности масштабирования поставленной задачи по отношению к произвольно-задаваемым архитектурам набора команд, итоговое программное решение должно обеспечивать корректную работу как минимум в рамках микропроцессорной архитектуры RISC-V с базовым набором команд RV32I[1], однако в процессе разработки рассматривались архитектурные решения с учетом упомянутого масштабирования задачи в перспективе.

Поскольку процесс верификации микропроцессоров не обязан включать в себя избыточную настройку вспомогательных программных средств, взаимодействующих с верифицируемой реализацией (прототипом) модели микропроцессора (в частности, настройку операционной системы и средств компиляции пользовательских программ), и раз архитектуры набора команд могут варьироваться от реализации к реализации, то под каждую тестируемую реализацию необходимо предварительно осуществить сборку специфической версии разрабатываемого генератора тестовых последовательностей с помощью средств кросс-компиляции перед ее непосредственной выгрузкой на верифицируемый прототип, где итоговый генератор не должен опираться на код, взаимодействующий с операционной системой – актуальная на момент написания настоящего документа версия разрабатываемого прототипа онлайн-генератора не позволяет полностью абстрагироваться от системных вызовов операционной системы и предоставление такой возможности рассматривается в перспективе.

[1] Подразумевается базовый набор команд RV32I, спецификация которого актуальна на момент разработки[1].

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

Функциональные характеристики

Состав выполняемых функций

- Использование пользовательских описаний ISA тестируемой модели микропроцессора, логики изменения (мутаций) генерируемых последовательностей инструкций, логики сравнения результатов выполнения оригинальной и измененной последовательностей инструкций для определения специфики конкретного генератора, а также моделей (шаблонов) частично заданных тестов и строго зафиксированных тестов;
- Последовательная генерация последовательностей из инструкций целевой архитектуры набора команд по предоставленным моделям частично заданных тестов (шаблонов) с подготовкой требуемых для их полного определения входных данных;
- Последовательная генерация последовательностей из инструкций целевой архитектуры набора команд по описаниям строго зафиксированных тестов;
- Использование пользовательского описания логики мутаций последовательностей генерируемых инструкций для их изменения в границах исходного уровня функциональной эквивалентности и дальнейшего сравнения результатов выполнения каждой сгенерированной последовательности инструкций целевой архитектуры команд с ее измененной версией;
- Подготовка и вывод итоговых вердиктов об успешности выполнения тестируемым прототипом микропроцессора каждой сгенерированной последовательности инструкций и ее измененной версии по заданным правилам сравнения.

Описание и обоснование формата входных данных

Так как онлайн-генератор тестовых программ с указанным предварительно описанием форматов генерируемых тестовых последовательностей, то в качестве пользовательских входных данных используется совокупность относительных расположений[2] файлов в файловой системе, а также вспомогательных флагов, подаваемых на компиляцию самого генератора.

[2] Относительные пути рассчитываются относительно точки вызова (директории), из которых запускается процесс компиляции.

Так как работа с генератором в перспективе может производиться на bare-metal системах, каждая из которых обладает собственной ISA и моделью памяти, то уже на этапе компиляции требуется указать все те параметры и пользовательские определения, которые будут использоваться в скомпилированном онлайн-генераторе, при этом исключив необходимость обращений к файловой системе уже в процессе работы программы.

Список пользовательских аргументов, рассматриваемых в процессе компиляции онлайн-генератора тестовых программ:

- Относительный или абсолютный путь в файловой системе до файла (кросс-)компилятора языка Си с поддержкой стандарта C99, который будет использоваться в процессе компиляции;
- Совокупность флагов, применяемых к выбранному ранее компилятору;
- Относительный или абсолютный путь в файловой системе до файлов описания ISA, использующейся в процессе генерации и исполнения тестовых цепочек;
- Относительный или абсолютный путь в файловой системе до файлов описания тестов, необходимых для генерации тестовых последовательностей по заданным шаблонам и для исполнения строго зафиксированных тестов;
- Относительный или абсолютный путь в файловой системе до файла описания логики функционально эквивалентного преобразования одной последовательности тестовых инструкций в другую;
- Относительный или абсолютный путь в файловой системе до директории вывода итогового результата статической компиляции – исполняемого файла онлайн-генератора тестовых программ;
- Опциональный флаг вывода сопроводительных сообщений и сообщений отладки в процессе работы генератора.

В первую очередь, пользователю необходимо передать информацию о кросс-компиляторе языка Си (стандарт C99) и передаваемых ему параметрах, которые будут использоваться в процессе сборки исполняемого модуля генератора тестовых программ непосредственно под тестируемую модель микропроцессора.

Кроме того, на этапе компиляции программы пользователю требуется предоставить четыре относительных или абсолютных пути в рамках файловой системы, описывающих расположение файлов

с описанием ISA для тестируемой модели микропроцессора, файлов с описанием моделей частично заданных и строго зафиксированных тестов, файлов с описанием логики мутаций генерируемых последовательностей инструкций и расположение директории для размещения скомпилированного онлайн-генератора соответственно.

Дополнительно пользователь может использовать флаг, сигнализирующий об использовании функции «printf», объявленной в заголовке «stdio.h» из стандартной библиотеки языка Си, для передачи итоговых вердиктов об исполнении тестовых последовательностей непосредственно в стандартный поток вывода, также используемый для вывода сопроводительных и отладочных сообщений в процессе работы онлайн-генератора.

За исключением упомянутого дополнительного флага, все остальные параметры обязательно должны быть указаны с учетом определенного порядка их включения в общий список аргументов, и для стандартной реализации ISA RISC-V RV32I используется отдельный скрипт сборки, автоматизирующий передачу большинства необходимых данных в общий скрипт автоматизации процесса сборки онлайн-генератора – в перспективе планируется замена подобного подхода на именованные аргументы, порядок указания которых не повлияет на процесс компиляции генератора.

Детальное описание процесса указания аргументов компиляции онлайн-генератора, а также общие инструкции по работе с онлайн-генератором описываются в документе «Прототип онлайн-генератора тестовых программ для микропроцессоров». Руководство программиста (ГОСТ 19.505-79).

После непосредственной компиляции исполняемого модуля генератора тестовых программ он может быть загружен на исполнение в системе, подлежащей тестированию – на данном этапе никакие пользовательские входные данные уже не требуются.

Описание используемого математического аппарата

В контексте поставленной задачи разделяются две основные группы тестов: частично заданные (шаблоны) и строго зафиксированные. В рамках динамического тестирования гораздо эффективнее иметь большой объем разнообразных тестов, где тестовые последовательности инструкций используют разные входные данные. Существует возможность описать некоторым образом набор тестов, что в совокупности с подстановкой псевдослучайных входных данных (если такое требуется) и большим количеством их итераций предоставляет возможность получить существенное количество рассмотренных случаев поведения тестируемого прототипа микропроцессора, требующих корректного выполнения используемых операций – подобные наборы тестов представляется шаблонами.

С другой стороны, некоторые отдельные случаи ожидаемого поведения микропроцессора достаточно трудно сгенерировать средствами генератора псевдослучайных последовательностей чисел за счет малой вероятности создания их точных копий, а потому некоторые случаи достаточно рассмотреть ровно один раз – именно в такую категорию тестов попадают строго зафиксированные тесты, задающие четкую последовательность инструкций без пропущенных входных данных.

Заметим некоторую систематичность, что любая тестовая последовательность инструкций представима в виде некоторой перестановки над элементами большей совокупности инструкций с определенными правилами извлечения инструкций для перестановки, а та совокупность, в свою очередь, определяется на основе обработки других совокупностей инструкций – в рамках выработанной терминологии взаимодействия с частично заданными тестами подобные совокупности инструкций получили название *шаблонных блоков*, а процесс генерации перестановки инструкций блока с определенными правилами их извлечения получил название *обработки*.

Таким образом, можно прийти к основной идее для описания шаблонов генерации тестовых последовательностей инструкций, которая используется по аналогии, например, в программном комплексе MicroTESK: шаблон генерации тестовых инструкций описывается через иерархическую структуру специфических шаблонных блоков (*дерево шаблона*), где для обработки конкретного блока в первую очередь требуется обработать все его вложенные блоки (или осуществить непосредственную обработку инструкций самого конкретного блока, если какие-либо вложенные блоки отсутствуют), на основе которых будет генерироваться последовательность инструкций для текущего блока – это означает, что при обработке самого внешнего блока (для которого все остальные блоки будут считаться вложенными и что никакой блок не имеет самый внешний блок в числе своих вложенных блоков) все остальные блоки будут уже обработаны.

На уровне обработки блока считается, что если блок не обладает вложенными блоками, а обладает непосредственным набором инструкций (в древовидной структуре сверху вниз от самого внешнего блока такие блоки будут располагаться в самом низу) для обработки, то этот набор инструкций будет считаться неявно принадлежащим некоторому (единственному) вложенному блоку.

На момент написания настоящего документа шаблонные блоки могут соответствовать одному из двух типов: *sequence* или *pool* соответственно.

Тип шаблонного блока sequence предписывает, что инструкции, находящиеся непосредственно на уровне самого sequence-блока или в любом обработанном блоке из числа вложенных, будут

зафиксированы в итоговой перестановке инструкций точно так же, как они и были зафиксированы на момент начала обработки sequence-блока с сохранением исходной очередности. Неформально говоря, тип sequence описывает упорядоченное (мульти[3])множество инструкций. Любые вспомогательные флаги и параметры, которыми может обладать sequence-блок, игнорируются.

Тип шаблонного блока pool предписывает, что инструкции, находящиеся непосредственно на уровне самого pool-блока или в любом обработанном блоке из числа вложенных, будут включены в итоговую перестановку в определенном количестве на основе определенных правил, зафиксированных на уровне блока вспомогательными флагами.

К таким вспомогательным флагам относятся:

- Выбор готовых блоков – вместо того, чтобы включать отдельные инструкции в перестановку, будут выбираться сразу обработанные блоки инструкций;
- Возможность повторно выбирать элементы (блоки/инструкции) – предоставляется возможность включать одну и ту же инструкцию или блок в итоговую перестановку два и более раз;
- Отключение учета относительного порядка (блоков/инструкций) – вместо того, чтобы выбирать элементы с учетом их относительного расположения, они будут выбираться независимо от него (выбрав элемент, никакой элемент до него не будет выбран впоследствии).

Все вышеупомянутые флаги по умолчанию не используются, а значит, в таком случае будут выбираться инструкции без повторов и с учетом их относительного следования. Можно утверждать, что выбирая столько же инструкций блоков или самих блоков, сколько и было всего из числа вложенных блоков, с сохранением относительного порядка и отсутствием повторов обработка будет происходить аналогично sequence-блоку.

Неформально говоря, тип pool описывает (упорядоченное) (мульти)множество инструкций. За исключением параметров количества выбираемых элементов и используемых флагов любые другие параметры игнорируются.

При обработке вложенных блоков с общим внешним блоком порядок их обработке соответствует тому порядку, который использовался при описании блоков изначально.

Следующее представление демонстрирует пример описания некоторого тестового шаблона (число в скобках обозначает некоторый уникальный номер вершины, сокращения «seq» и «pool» соответствуют

[3] Зависит от того, были ли в изначальном блоке полностью идентичные инструкции без незаполненных полей.

типам шаблонным блоков sequence и pool; параметр «n» описывает количество элементов (блоков/инструкций), выбираемых в блоке pool; «BLOCKS», «WITHREPEATS» и «NOORDER» соответствуют флагам выбора блоков вместо инструкций, разрешения повторов при выборе элементов и отсутствии следования относительному порядку элементов при обработке pool-блока; многоточие показывает, наличие непосредственного набора инструкций блока вместо вложенных для него блоков).

$$(1)seq\left\{\begin{array}{l}(2)seq\{...\\(3)pool\,(n=1,\,BLOCKS)\left\{\begin{array}{l}(4)seq\{...\\(5)pool\,(n=3,\,WITHREPEATS,NOORDER)\{...\\(6)seq\{...\end{array}\right.\end{array}\right.$$

Таким образом, для обработки вершины 1 требуется обработать вершины 2, 3 и 6, а для обработки вершины 3 требуется обработать вершины 4 и 5, тогда итоговым порядок обработки вершин будет 2, 4, 5, 3, 6, 1.

Описание и обоснование выбора общего алгоритма работы

Все основные параметры для работы с онлайн-генератором задаются на этапе его компиляции и на этапе его исполнения дополнительной обработки входных параметров не производится.

Онлайн-генератор начинает выполнение своих задач с инициализации тех рабочих областей оперативной памяти, с которыми будет происходить взаимодействие при генерации и выполнении тестов — для этого вызываются определенные процедуры, спецификация которых задается на пользовательском уровне (подгружаются инструкции ISA, сами тесты и иные вспомогательные данные).

Далее, вызывается процедура генерации тестов по описанным шаблонам: каждый шаблон имеет определенное количество итераций, которые для него необходимо провести, на каждой итерации заново формируя тестовую последовательность из заданного шаблона.

Так как для обработки конкретного шаблонного блока требуется провести обработку всех вложенных для него блоков, в иерархической структуре шаблона подобная обработка схожа со стандартным алгоритмом обхода графа данных в глубину.

Для каждой сгенерированной в итоге инструкции вызывается процедура, также определяемая на пользовательском уровне и определенная либо для конкретного шаблона, либо определенная по умолчанию для шаблонных тестов, позволяющая заполнить пропущенные данные в инструкциях с использованием логики генерации псевдослучайных последовательностей данных.

Далее, все инструкции транслируются в их двоичное представление и размещаются с учетом их выравнивания по длине самих инструкций в заготовленной заранее области памяти для инструкций.

Вызываются предварительно описанные на пользовательском уровне с помощью препроцессорных средств языка Си процедуры выгрузки информации о содержании регистров на момент вызова процедуры в заготовленную заранее область памяти для регистров – так как при оформлении и непосредственном вызове подобных процедур с помощью функций языка Си могут быть перезаписаны и утеряны данные отдельных регистров, то требуется использовать именно пользовательские макроопределения языка.

Упомянутая область памяти для инструкций трактуется на уровне программы как область инструкций самой программы, которые должны быть исполнены, и управление передается на нее – по завершении исполнения инструкций содержимое регистров выгружается в отдельную область памяти регистров, а также восстанавливается сохраненное до выполнения инструкций содержимое регистров.

Далее, происходит функционально эквивалентное преобразование изначальной тестовой последовательности, которые также описывается на пользовательском уровне, и для новой последовательности выполняются шаги из трех предыдущих абзацев настоящего документа.

С использованием объявленного на пользовательском уровне функционала определяются затронутые в рамках выполнения двух тестовых последовательностей регистры процессора и производится сравнение их сохраненного содержимого и выносится вердикт об отсутствии и наличии различий среди данных затронутых регистров – в случае наличия расхождений необходимая контекстная информация о тесте выгружается в заранее подготовленную для этого область памяти, также определенную на пользовательском уровне, а также при указании соответствующего флага на этапе компиляции генератора производится подробный вывод информации о текущем тесте в стандартный поток вывода языка Си.

В случае со строго зафиксированными тестами производятся практически все вышеупомянутые шаги, только уже не потребуются процедуры генерации тестовой последовательности по заданному шаблону и заполнения недостающих параметров инструкций, а также появляется возможность задавать на пользовательском уровне специфические процедуры проверки корректности выполнения тестов без расчета затронутых в рамках выполнения тестовых последовательностей регистров.

При возникновении расхождений среди данных затронутых регистров выполнение как шаблонных, так и строго зафиксированных тестов прекращается.

Условия эксплуатации

Условия эксплуатации описываются в составе документа «Прототип онлайн-генератора тестовых программ для микропроцессоров». Руководство программиста (ГОСТ 19.505-79).

Состав описания ISA тестируемого прототипа

В рамках описания ISA предполагается следующее разделение: существуют группы инструкций, включающие в себя набор инструкций с одинаковым форматом кодирования в двоичное представление, которое также описывается на уровне группы; любая инструкция без параметров содержит в себе некоторую ссылку на ту группу, к которой она принадлежит, а также ту информацию, которая остается неизменной при любых входных данных инструкции (например, код самой операции); каждая инструкция может иметь свое сокращенное представление – псевдо-инструкцию, в рамках которой одно или несколько полей инструкции-прообраза, на которую в псевдо-инструкции хранится ссылка, уже имеет зафиксированные данные (например, что инструкция прибавления значения к одному и тому же регистру имеет сокращенное представление с двумя операндами вместо трех); любая инструкция в рамках тестов представляется параметризованной инструкцией, которая может ссылаться как на псевдо-инструкцию, так и напрямую на инструкцию без параметров и которая может обладать своими собственными зафиксированными данными (оставшиеся данные должны быть сгенерированы в рамках обработки шаблонов).

Файлы с описанием ISA для верифицируемого прототипа должны включать в себя следующий минимальный набор аспектов:

- Описание количеств использующихся регистров: регистров общего назначения, а также регистров для работы с вещественными числами, если те используются;
- Описание количеств присутствующих в описании обычных инструкций и групп, на которые их можно поделить, а также псевдо-инструкций, реализуемых через имеющиеся обычные инструкции;
- Описание реализации зафиксированного интерфейса для способов выгрузки значений регистров из/в статически выделенные области оперативной памяти без изменения до/во время/после операции выгрузки соответствующих регистрам значений;
- Описание представлений групп инструкций, не-параметризованных инструкций, псевдо-инструкций и параметризованных инструкций;

- Описание реализации зафиксированного интерфейса для получения длины конкретной инструкции в байтах;
- Описание реализации зафиксированного интерфейса для кодирования (ассемблирования) инструкции по их имеющимся данным;
- Описание реализации зафиксированного интерфейса для включения тех или иных регистров, изменяемых в рамках выполнения инструкции, в список таких регистров в рамках проведения одной итерации некоторого теста;
- Описание реализаций зафиксированных интерфейсов для инициализации областей памяти регистров, групп инструкций, инструкций, псевдо-инструкций.

Состав описания частично заданных и зафиксированных тестов

При описании строго зафиксированных тестов требуется вручную задавать инструкции на уровне той области памяти, которая была под них выделена. В случае частично заданных тестов помимо этого требуется также задавать информацию о шаблонных блоках, на основе которых будет генерироваться итоговая тестовая последовательность.

Важно: любой тест должен включать в себя хотя бы одну достижимую инструкцию возврата (*return*) для передачи управления для корректного функционирования программы, и в любом шаблоне самый внешний блок должен иметь ID 0.

Файлы с описанием частично заданных и строго зафиксированных тестов для верифицируемого прототипа должны включать в себя следующий минимальный набор аспектов:

- Описание максимальных допустимых количеств инструкций в рамках одной генерируемой последовательности (обязательный параметр) и инструкций, добавляемых в процессе мутаций последовательностей (необязательный параметр – при его отсутствии количество генерируемых дополнительно инструкций будет зависеть от первого упомянутого параметра);
- Описание количеств частично заданных и зафиксированных тестов;
- Описание максимально допустимого количества блоков в рамках одного шаблона;
- Описание частично заданных тестов через конкретный для всех таких тестов интерфейс;
- Описание строго зафиксированных тестов через конкретный для всех таких тестов интерфейс;

- Описание реализации зафиксированного интерфейса для заполнения инструкций по умолчанию в рамках одной тестовой последовательности, если для обрабатываемого шаблона не задавалась специфическая функция;
- Описание реализаций зафиксированных интерфейсов для инициализации областей памяти под частично заданные тесты, количества их последовательных запусков и пользовательские функции заполнения сгенерированных тестовых последовательностей, строго зафиксированные тесты и их пользовательские функции сравнения эквивалентности результатов выполнения двух тестовых последовательностей, иерархическую структуру шаблонных блоков, стеки обхода шаблонных блоков и размещения сгенерированных инструкций, а также вывод вердиктов об ошибках во время выполнения тестов и иные буферные области.

Состав описания мутаций созданных последовательностей инструкций

Файлы с описанием мутаций (с сохранением функциональной эквивалентности) созданных последовательностей инструкций должны включать в себя реализацию интерфейса для добавления конкретного количества дополнительных строк в выделенную под это область памяти на основании сгенерированной ранее тестовой последовательности инструкций.

Описание и обоснование формата выходных данных

На этапе компиляции выходными данными считаются статически-скомпилированный исполняемый файл самого онлайн-генератора тестовых программ, а также набор файлов использовавшихся при сборке.

На этапе исполнения в качестве основных выходных данных предоставляются вердикты об успешности/безуспешности корректности работы использовавшихся в тестах инструкций на основании сравнения результатов выполнения каждой сгенерированной тестовой последовательности инструкций и ее измененной версии, а также должны предоставляться некоторые дополнительные данные, необходимые для воспроизведения тестов с некорректным завершением работы – все вердикты и необходимые для их описания данные должны соответствовать некоторому общему интерфейсу и располагаться в оперативной памяти, связанной с верифицируемым прототипом.

Если при компиляции генератора использовался флаг использования стандартного потока вывода, описанный в пункте 4.1.2 «Организация входных данных», то информация о вердиктах выполнения тестовых последовательностей инструкций будет дополнительно выводиться в упомянутый стандартный поток вывода.

Требования к составу и параметрам технических средств

Для успешной компиляции программы требуется следующий состав технических средств:

- Компьютер с одной из установленных операционных систем – Linux (x86-64) или Linux (ARM64);
- Наличие не менее 100МБ свободного места в памяти компьютера.

Иные требования совпадают с системными требованиями указанных операционных систем.

Поскольку скомпилированный прототип генератора будет использовать статическую память в глобальной области видимости в зависимости от требований самого пользователя (описывается набором использующихся регистров в рамках тестируемой системы, количеством тестов и итераций для отдельных тестов, объемом памяти для выгрузки результатов отдельных тестов и так далее), итоговое максимальное ограничение на объем оперативной памяти не может быть сформулировано однозначно.

В случае запуска генератора на уровне bare-metal системы (например, с использованием ПЛИС), могут потребоваться условия совместимости аппаратного обеспечения с существующими программными решениями для выделения регионов оперативной памяти, используемой в генераторе, а также для возможности изучить выгруженные генератором данные вердиктов о выполненных тестах.

Требования к составу и параметрам программных средств

Для успешной компиляции программы требуется следующий состав программных средств:

- Кросс-компилятор языка Си (стандарт C99) с генерацией кода, совместимого с тестируемым прототипом микропроцессора на уровне его ISA;
- Система CMake[2] с открытым исходным кодом для кроссплатформенного описания процесса компиляции исходного кода на языке Си версии 3.16 или более поздней.

Если же запуск генератора будет производиться в рамках эмулятора микропроцессора с опциональным флагом вывода сопровождающих и отладочных сообщений, то вдобавок к вышеупомянутым требованиям добавится и требование совместимости с набором интерфейсов системных вызовов в рамках стандарта C99.

4. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Ориентировочная экономическая эффективность

Использование автоматической генерации последовательностей тестовых инструкций для микропроцессоров позволит с некоторой точностью покрыть большинство возможных входных данных существующих инструкций, тем самым повышая итоговую надежность разрабатываемой модели микропроцессора.

Кроме того, использование подхода онлайн-генерации тестов в сравнении с оффлайн-генерацией позволит гораздо быстрее перейти к этапу непосредственного исполнения сгенерированных тестов.

В дополнение к этому утверждению, возможное использование разрабатываемого генератора в совокупности с ПЛИС позволит достичь таких скоростей исполнения отдельных инструкций, которые были бы сравнимы с аналогичными показателями на готовой СБИС на полупроводниковой подложке, что позволит получить наиболее точную характеристику разрабатываемой модели.

Предполагаемая потребность

На момент начала разработки не было выявлено примеров использования подхода онлайн-генерации последовательностей тестовых инструкций с целью верификации проектируемой RTL-модели микропроцессора. В дополнение к этому, проект разрабатывается по принципу открытого исходного кода, что предоставит возможности ручной адаптации существующего решения под специфические нужды в рамках верификации микропроцессоров.

Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

С момента начала разработки известно о существовании несколько относительно схожих аналогов решения поставленной задачи (отечественных и зарубежных):

- ИСП РАН MicroTESK[4];
- IBM Threadmill[3];
- Common Hardware for Interfaces, Processors and Systems (CHIPS) RISC-V Generator[5].

Все представленные аналоги реализуют подход оффлайн-тестирования, а потому не могут быть полностью сопоставлены разрабатываемому проектному решению, однако все же являются наиболее схожими по назначению.

Конкретно разработка Threadmill от IBM является закрытым программным решением, что противопоставляется текущему программному решению с открытым исходным кодом. Генератор тестовых программ от CHIPS реализован исключительно под RISC-V в отличие от подхода к обобщению под различные архитектуры в текущем программном решении. К тому же, данный аналог использует промежуточную симуляцию, что также сказывается на производительности в процессе выполнения последовательностей тестовых инструкций.

5. СПИСОК ИСПОЛЬЗОВАВШИХСЯ ИСТОЧНИКОВ

1) Andrew Waterman, Krste Asanovic. The RISC-V Instruction Set Manual [Статья в Интернет] Режим доступа: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf , свободный. (дата обращения: 14.11.2022)
2) CMake cross-platform build-process managing system. [Интернет-ресурс] Режим доступа: https://cmake.org/ , свободный. (дата обращения: 14.12.2022)
3) IBM Threadmill Post-Silicon Exercoser [Статья в Интернет] Режим доступа: https://researcher.watson.ibm.com/researcher/view_group.php?id=1347 , свободный. (дата обращения: 13.11.2022)
4) MicroTESK configurable test program generator [Статья в Интернет] Режим доступа: http://www.microtesk.org/ , свободный. (дата обращения: 13.11.2022)
5) Random instruction generator for RISC-V processor verification [Интернет-ресурс] Режим доступа: https://github.com/chipsalliance/riscv-dv , свободный (дата обращения: 09.01.2023)
6) А.С. Камкин, А.М. Коцыняк, С.А. Смолов, А.Д. Татарников, М.М. Чупилко, А.А. Сортов. Средства функциональной верификации микропроцессоров [Статья в Интернет] Режим доступа: Сборник трудов ИСП РАН, https://ispranproceedings.elpub.ru/jour/article/view/771, свободный. (дата обращения: 13.10.2022)

6. ПРИЛОЖЕНИЯ

Приложение 1: Словарь использующихся терминов и понятий

Таблица 1 – словарь использующихся терминов и понятий

Термин	**Значение**
Функциональная верификация	Процесс проверки соответствия разрабатываемой RTL-модели микропроцессора документам, описывающих требования к ней.
Динамическое (функциональное) тестирование	Способ проведения верификации, подразумевающий проверку соответствия реализации требованиям путем исполнения тестов на целевой (тестируемой) системе.
Оффлайн-тестирование	Процесс генерации набора тестовых ситуаций без использования системы, подлежащей тестированию.
Онлайн-тестирование	Процесс генерации набора тестовых ситуаций и их непосредственного исполнения на системе, подлежащей тестированию.
Архитектура набора команд (Instruction Set Architecture, ISA)	Совокупность правил и свойств, по которым происходят вычисления на уровне микропроцессора. ISA регламентирует использование регистров микропроцессора, а также существующие типы данных, модель памяти и другие аспекты.
Bare-metal система	Комплекс аппаратных средств без выстроенного программного окружения, ответственного за контроль используемой памяти, планируемых задач и других взаимодействий в рамках системы и необходимых для этого ресурсов – операционной системы.
Граф	Математический объект, описывающий пару из множеств вершин и ребер, где множество ребер задается как некоторое подмножество возможных пар вершин.

Приложение 2: Описание и функциональное назначение структур данных программных объектов

Таблица 2 – описание и функциональное назначение структур данных программных объектов

Структура данных	**Описание**
InstructionGroup	[Задается пользователем] Описывает формат кодирования совокупности инструкциий в двоичное представление.
Instruction	[Задается пользователем] Описывает отдельную инструкцию (без параметров) с не зависящей от параметров информацией инструкции (например, код операции).
PseudoInstruction	[Задается пользователем] Описывает укороченное представление существующей инструкции с частично-заполненными полями инструкции.
ParametrizedInstruction	[Задается пользователем] Описывает одну частично или полностью заполненную инструкцию.
GeneratorContext	Инкапсулирует информацию, необходимую для генерации тестовой последовательности по заданному шаблону – указатели используемые области памяти, а также максимальные ограничения в рамках генератора.
TemplateBlock	Инкапсулирует информацию об одном шаблонном блоке частично заданного теста – собственный IDссылки на внешние, соседние и вложенные блоки, указатель на расположение его инструкции на конкретный момент на стеке обхода дерева шаблонов, а также указатель на непосредственные инструкции блока, если тот не имеет вложенных блоков.
BlockProcessInfo	Инкапсулирует информацию, необходимую для обработки одной вершины – контекст генератора, указатели на стеки блоков и инструкции для обхода дерева шаблонов, а также используемые флаги и параметры самого блока.
BlockProcessResult	Инкапсулирует информацию о том, сколько инструкций вложенных блоков было задействовано и сколько инструкций было сформировано в результате обработки.

Приложение 3: Описание свойств программных объектов (переменных, функций и так далее)

Таблица 3 – описание и функциональное назначение глобальных переменных пользовательского уровня, используемых в рамках ядра программы

Программный объект	**Тип**	**Описание**
GEN_INS_GROUPS	struct InstructionGroup *	Используется для хранения данных об имеющихся группах инструкций.
GEN_INS	struct Instruction *	Используется для хранения данных об имеющихся инструкциях.
GEN_PSEUDO_INS	struct PseudoInstruction *	Используется для хранения данных об имеющихся псевдо-инструкциях.
GEN_FIRST_REG_DUMP	uint32_t * или uint64_t *	Используется для хранения содержимого регистров после проведения первой тестовой последовательности.
GEN_SECOND_REG_DUMP	uint32_t * или uint64_t *	Используется для хранения содержимого регистров после проведения второй тестовой последовательности.
GEN_BUF_REG_DUMP	uint32_t * или uint64_t *	Используется для хранения исходных значений регистров микропроцессора до исполнения тестовой последовательности.
GEN_REG_INCLUDE_IN_DUMP	uint8_t *	Используется для определения необходимости включения тех или иных регистров в итоговый список сравнения при выполнении тестов.
GEN_TEST_TEMPLATES	void (**)(struct TemplateBlock *, uint8_t, struct ParametrizedInstruction *, uint16_t)	Используется для определения частично заданных тестов.
GEN_TEST_TEMPLATES_ITERS	uint8_t *	Используется для определения количеств итераций запуска обработки конкретных шаблонов.
GEN_TEST_TEMPLATES_FILL_INS	void (**)(struct ParametrizedInstruction *)	Используется для определения пользовательских функций заполнения пропущенных входных данных

		инструкций для конкретных шаблонов
GEN_ASM_TESTS	uint16_t (**) struct ParametrizedInstruction *, uint16_t)	Используется для определения строго зафиксированных тестов.
GEN_ASM_TESTS_DUMP_EQ	uint8_t (**)(uint32_t * или uint64_t *, uint32_t * или uint64_t *, uint8_t)	Используется для определения пользовательских функций сравнения корректности работы конкретных строго зафиксированных тестов.
GEN_USED_ELEMENTS	uint8_t *	Используется для обработки pool-блока без учета относительного расположения блоков или инструкций.
GEN_TEST_INSTRUCTIONS	struct ParametrizedInstruction *	Используется для хранения изначальных инструкций шаблонных блоков, а также для хранения инструкций строго зафиксированных тестов.
GEN_BLOCK_INSTRUCTIONS_STACK	struct ParametrizedInstruction *	Используется для построения обхода конкретного дерева шаблона.
GEN_INSTRUCTIONS_BUF	struct ParametrizedInstruction *	Используется для промежуточной выгрузки выбранных в процессе обработки инструкций (как в рамках шаблонов, так и извне).
GEN_BLOCK_ORDER_STACK	uint8_t *	Используется для построения порядка обхода конкретного дерева шаблона.
GEN_TEMPLATE_TREE	struct TemplateBlock *	Используется для загрузки конкретного дерева шаблона.

Таблица 4 – описание и функциональное назначение используемых в рамках программы операций (функций)

Функция	**Значение**
uint8_t get_ins_len(struct InstructionGroup *, struct Instruction *, struct PseudoInstruction *, struct ParametrizedInstruction *)	[Задается пользователем] Получает длину конкретной инструкции из структуры данных.
uint32_t или uint64_t encode_ins(struct ParametrizedInstruction *)	[Задается пользователем] Кодирует одну инструкцию в ее двоичное представление.
uint16_t mutate(struct ParametrizedInstruction *, uint16_t ins_count, uint16_t, struct ParametrizedInstruction *)	[Задается пользователем] Добавляет инструкции в сгенерированную тестовую последовательность, сохраняя изначальную функциональную эквивалентность.
void include_regs(struct ParametrizedInstruction *, uint8_t *to_include)	[Задается пользователем] Рассматривает инструкцию и определяет, какие регистры были ей изменены для итогового сравнения результатов выполнений двух тестовых последовательностей.
void default_fill_ins(struct ParametrizedInstruction *)	[Задается пользователем] Используется по умолчанию для заполнения пропущенных входных данных инструкций, если для конкретного шаблона не была задана специфическая функция.
int main()	Стартовая точка онлайн-генератора.
void print_configuration()	[Флаг для stdio.h] Выводит информацию о генераторе.
init_reg_dumps()	Инициализация областей памяти, хранящих информацию о данных регистров.
init_groups()	Инициализация областей памяти, хранящих информацию о группах инструкций.
init_instructions ()	Инициализация областей памяти, хранящих информацию об инструкциях.
init_stacks()	Инициализация областей памяти, необходимых для обработки дерева шаблонов и выполнения строго зафиксированных тестов.
init_tree()	Инициализация областей памяти, необходимых для работы дерева шаблонов.

init_test_patterns()	Инициализация областей памяти, необходимых для работы частично заданных тестов.
init_asm_tests()	Инициализация областей памяти, необходимых для работы строго зафиксированных тестов.
init_misc()	Инициализация вспомогательных областей памяти.
void run_pattern_tests()	Загрузка и выполнение частично заданных тестов.
void run_asm_tests()	Загрузка и выполнение строго зафиксированных тестов.
generate(struct GeneratorContext *)	Генерация тестовой последовательности по заданному шаблону.
generate_and_fill(struct GeneratorContext *, void (*)(struct ParametrizedInstruction *))	Генерация тестовой последовательности по заданному шаблону с заполнением пропущенных входных данных инструкций через специфицированную функцию.
struct BlockProcessResult process_block(struct GeneratorContext *, uint8_t, uint16_t, uint16_t)	Обработка вершины, на которую имеется указатель в контексте генератора на данный момент.
void encode_ins_dump(struct ParametrizedInstruction *, uint16_t, uint32_t * или uint64_t *)	Кодирует набор инструкций в их двоичное представление и размещает впритык друг к другу в памяти с учетом длины каждой из инструкций.
void core_include_regs(struct ParametrizedInstruction *, uint16_t, uint8_t *to_include, uint8_t reg_count)	Рассматривает каждую инструкцию и определяет, какие регистры были ей изменены для итогового сравнения результатов выполнений двух тестовых последовательностей.