УДК 004.052.42 DOI: 10.31114/2078-7707-2022-4-37-43

Функциональная верификация микропроцессоров с применением методов машинного обучения

Н.А. Гревцев

ФГУ ФНЦ НИИСИ РАН, г. Москва, ngrevcev@cs.niisi.ras.ru

***Аннотация* — Применимость методов машинного обучения для тестирования моделей процессора в настоящее время исследуется в крупнейших иностранных технологических компаниях (исследовательские центры ARM, Intel, IBM и другие) и институтах. Однако исследования проводятся только с точки зрения машинного обучения в области формальной верификации, генерации тестов с использованием символического выполнения и решения ограничений, а также для поиска нерегулярных ошибок в уже изготовленном кристалле СБИС микропроцессора. Новизна предлагаемого решения в применении машинного обучения для имитации поведения приложений пользователя с целью повышения качества тестирования RTL-модели микропроцессора направленными псевдослучайными методами генерации тестов. В рамках данной работы планируется показать применимость инструментов машинного обучения для функциональной верификации RTL-модели микропроцессора на системном уровне. Основным результатом проведенного исследования является возможность имитировать поведение набора пользовательских приложений на уровне машинного кода, а также автоматизация процесса анализа труднодостижимых в рамках классического маршрута верификации ситуаций с целью повышения тестового покрытия.**

***Ключевые слова* — функциональная верификация, машинное обучение, генерация тестов на основе покрытия, deep learning.**

I. Введение

Сложность разрабатываемых микропроцессоров непрерывно растёт: усложняется микроархитектура, наращивается число ядер с целью повышения производительности, что приводит к необходимости развивать фундаментальные научные подходы и методы для последующего применения в маршруте тестирования. Поэтому вопрос получения нового знания о том, как направить случайный тестовый код в сторону реальных задач является актуальным и одним из ключевых при разработке новых методов верификации.

Несмотря на то, что стохастическое псевдослучайное тестирование является одним из ключевых инструментов в маршруте верификации моделей микропроцессоров, полностью случайные тесты не позволяют качественно протестировать «сложно-достижимые» ситуации (так как без инженерного знания и специального нацеливания тестов на те или иные аспекты микроархитектуры тесты либо не являются валидными, либо содержат нереалистичный код) [1].

В то же время запуск пользовательских задач в качестве инструмента тестирования RTL-модели неприменим из-за чрезвычайно большой длины тестов, большого количества повторяющихся участков кода, а также неравномерно распределенной «сложности» кода программы (интересные с точки зрения тестирования ситуации могут отстоять во времени на миллионы тактов, что приводит к медленному росту покрытия и к бесполезной трате машинных ресурсов при моделировании однотипных последовательностей инструкций).

Зачастую критические ошибки обнаруживаются в уже выпущенном кристалле СБИС вследствие невозможности полноценно запустить операционную систему и приложения пользователя на RTL-модели. Основной причиной этого является низкая скорость моделирования RTL-кода.

Несмотря на постоянно совершенствующуюся систему стохастического тестирования, запуск параллельной симуляции тестовой системы на многопоточном кластере, а также направление тестовых сценариев с помощью эвристик, покрыть все возникающие в реальных пользовательских приложениях ситуации не представляется возможным. Полноценный запуск операционной системы на рабочих частотах может позволить обнаружить большинство оставшихся в проекте ошибок до выпуска готового изделия и передаче его конечным пользователям. По этой причине даже крупные компании-разработчики МП вынуждены делать дорогостоящий «тестовый» запуск перед полноценным выходом процессора на рынок. Поэтому основным направлением при верификации ядра микропроцессора является направленное стохастическое тестирование. Его суть состоит в том, что инженер-верификатор, настраивая поведение случайных тестов, направляет процесс стохастического тестирования в сторону наиболее интересных областей.

Предлагаемая идея позволит использовать преимущества запуска приложений пользователя с точки зрения обнаружения ошибок в микропроцессоре на ранних этап разработки его RTL-модели (когда стоимость исправления ошибки минимальна).

Раннее обнаружение микроархитектурных ошибок достигается путем включения в состав псевдослучайный тестов эквивалентного «экстракта» пользовательских программ, построенного с помощью методов машинного обучения.

Предлагаемый в данной работе подход позволит применить накопленные техники сбора и анализа функциональных метрик и автоматизировать процесс инжектирования полученных знаний в процесс генерации тестов. По сравнению с стандартным маршрутом стохастического тестирования предлагаемый метод перекладывает обязанности по анализу и применению тестовых метрик, а также по направлению процесса генерации тестов с инженера верификатора на нейронную сеть (Рис. 1).

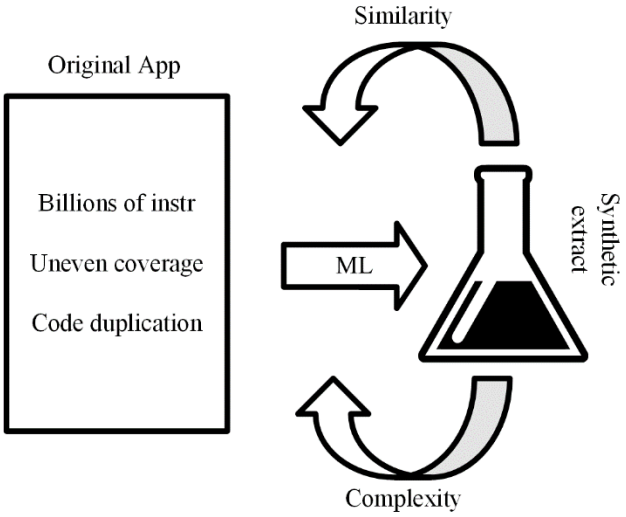

Рис. 1. Получение синтетического экстракта из приложения пользователя

II. Применимость машинного обучения в маршруте верификации микропроцессора

Машинное обучение (ML) в наши дни широко используется во многих сценариях. Методы ML, включая традиционные алгоритмы и алгоритмы глубокого обучения, позволяют достичь высокой производительности при решении задач классификации, обнаружения и размещения в различных сферах проектирования микропроцессоров. Более того, методы ML обладают большим потенциалом для предоставления высококачественных решений для многих NP-полных задач, которые распространены в области функциональной верификации микропроцессорных ядер, в то время как традиционные методы обычно приводят к огромным затратам времени и ресурсов на решение этих проблем. Традиционные методы, как правило, решают все проблемы постепенно, накапливая знаний по мере продвижения стадии верификации проекта. Вместо этого алгоритмы ML фокусируются на извлечении высокоуровневых особенностей или шаблонов, которые могут быть повторно использованы в других связанных или аналогичных ситуациях, избегая повторного сложного анализа. Поэтому применение методов машинного обучения является перспективным направлением для ускорения решения задач верификации и тестирования микропроцессоров [3].

В этой главе представлены исследования, в которых основное внимание уделяется различным успешным применениям ML для верификации дизайна микропроцессора на различных этапах проектирования и тестирования (например, верификация pre-silicon прототипа [3], post-silicon валидизация [4], сокращение количества стимулов [5], тесты фильтрации [6], направленная генерация тестов [7] и изучение паттернов доступа к памяти [8]). Хорошо известно, что сложные конструкции микропроцессоров в значительной степени зависят от верификации на основе моделирования. При этом примерно 70% усилий по проектированию тратится на верификацию [9], и машинное обучение успешно применяется для ускорения процесса верификации проекта. Например, в работе [10] предпринята попытка предоставить более интеллектуальную модель генерации тестов для сокращения времени моделирования. Методы генерации ограниченно-случайного теста модулируется данными покрытия. System Verilog testbench с фреймворком Universal Verification Methodology (UVM) также интегрирован с моделями машинного обучения.

В основном машинное обучение применяется для решения двух наиболее важных отраслевых проблем: 1) ускорение процесса верификации путем оптимизации ограничений таким образом, чтобы 100% покрытия DUT (Device Under Test) достигалось как можно быстрее; 2) поиск наименьшего набора стимулов, который приводит к полному покрытию для DUT. Предполагается, что такой набор стимулов идеально подходит для регрессионного тестирования.

Для решения этих проблем в [11] предложен новый метод оптимизации ограничений, основанный на рекуррентной нейронной сети (RNN). В [12] предпринята попытка разработать автоматизированный процесс для определения высокоприоритетных аспектов проектирования для верификации. Кроме того, авторы создают компактные тестовые программы, которые на порядок меньше, чем используемые на начальных стадиях верификации. Классификатор машинного обучения используется для изучения корреляции между расположением ошибок и паттернами их проявления - это решение упрощает аппроксимацию процесса локализации ошибок.

Традиционные методы генерации случайных тестов (RTG) или даже автоматизированные методы генерации тестов не являются оптимальными из-за того, что они содержат много повторяющихся или нерелевантных ситуаций (тестовых примеров). Таким образом, становится очевидным, почему проводится много исследований, в которых применяются методы ML для оптимизации процесса генерации тестов для верификации в качестве дополнительных подходов к существующему маршруту верификации.

Вэнь Чен в работе [13] ставит цель исследовать тестовое знания (testing knowledge) для повышения эффективности тестов с точки зрения достижения удовлетворительного уровня покрытия. Основная идея состоит в том, чтобы предоставить инженерам по верификации интерпретируемые и применимые знания. Предложены методологии анализа данных для трех сценариев применения, где такие знания необходимы для принятия обоснованных решений [14]:

• система обнаружения тестов для принятия решения о том, какие тесты моделировать, чтобы определить новые тесты, которые с наибольшей вероятностью будут способствовать росту покрытия;

• методология изучения правил для уточнения шаблона теста, чтобы увеличить шансы попадания в определенные события (точку) покрытия.;

• метод расширения плана тестирования для анализа набора существующих тестовых элементов и устранения недостающих точек покрытия.

Помимо традиционных стратегий функциональной верификации и тестирования, в некоторых исследованиях используются модели, основанные на машинном обучении для руководства процессом повышения покрытия. В недавних исследованиях было продемонстрировано превосходство алгоритмов машинного обучения: подход, основанный на машинном обучении, может значительно лучше работать с функциональным покрытием и достигать сложно достижимых состояний быстрее, чем случайные или ограниченно случайные методы [14], [15].

Труднодоступные точки можно покрывать чаще и быстрее, чем с помощью простых случайных шаблонов, применяя некоторые методы ML к входным данным генерации шаблонов при верификации, основанной на покрытии [15]. Однако следует подчеркнуть, что готовых к производству рецептов применения ML для функциональной верификации промышленных образцов пока не существует. Но в работе [6] объясняются некоторые основные концепции машинного обучения и ключевые идеи этих методологий, иллюстрируются проблемы реализации этих методологий на практике на основе примеров промышленной верификации.

III. Работа модели

В отличие от описанных работ данное исследование сосредоточено на функциональной верификации на *системном* уровне. В первую очередь это связано с применением высокоразвитой системой стохастического тестирования и является продолжением предыдущих исследований.

В отличии от UVM, в котором объектом тестирования является отдельный блок микропроцессора, а тестовыми воздействиями - наборы сигналов и ограничений, в функциональной верификации на системном уровне тестовыми последовательностями являются группы ассемблерных инструкций, а весь дизайн микропроцессора рассматривается как единое целое. Основным плюсом данного подхода является поиск ошибок, находящихся на стыке различных блоков микропроцессора и верификация их взаимодействия в общей системе.

На рисунке 2 представлен общий вид разработанного генератора.

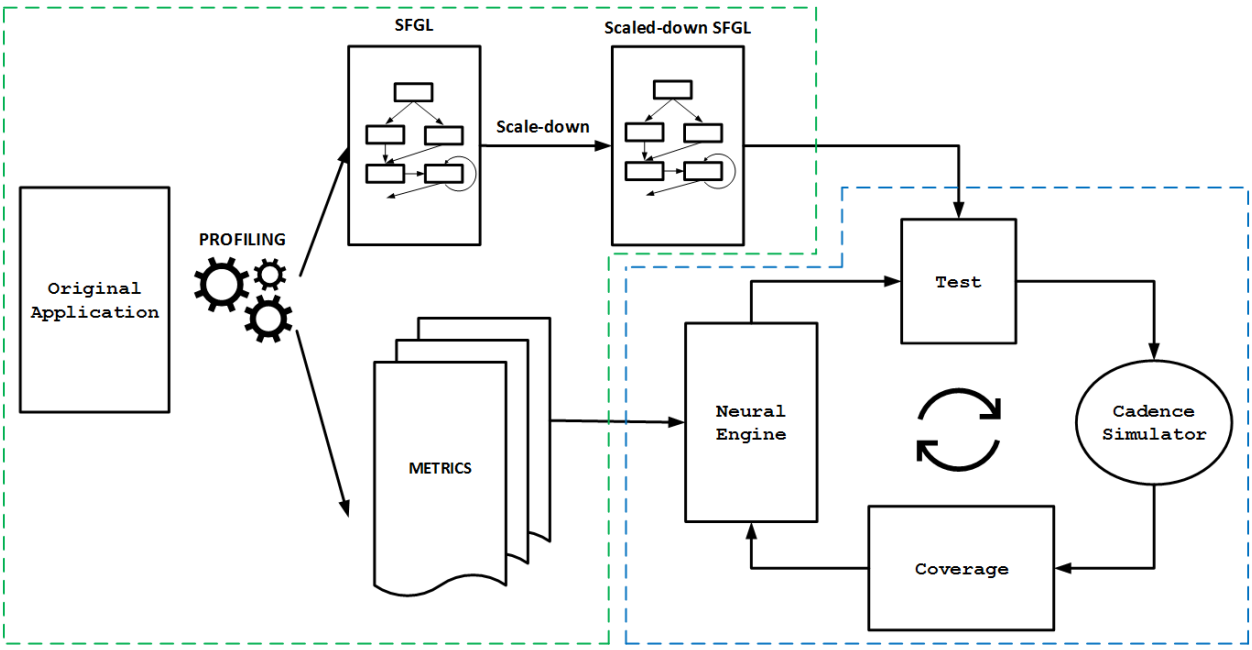

Рис 2. Общий вид разработанного генератора тестов

Его работу можно разделить на две стадии. На первой стадии (на рисунке отмечено зеленым) трасса оригинального приложения профилируется и происходит сокращение времени выполнения приложения. Конечным результатом работы этой части генератора являются граф потока управления, описывающий структуру приложения, таблица метрик, описывающих наполнение приложения и слепок работы приложения с памятью. По этим данным можно построить синтетический клон приложения, то есть псевдослучайный тест на языке ассемблер, структурно повторяющий работу оригинального приложения, но при этом выполняющийся на несколько порядков быстрее. На второй стадии (на рисунке отмечено синим) с помощью инструментов машинного обучения из синтетического клона приложения итеративно создается его синтетический экстракт.

A. Профилирование приложения

Первая стадия подробно описана в статье "Генерация синтетических клонов приложений пользователя для функциональной верификации" [17], поэтому в рамках данной статьи будет приведено только ее краткое описание.

На первой стадии происходит считывание трассы исполнения программы и построение по ней графа переходов. Базовым блоком (вершиной графа) в данном случае является блок инструкций, заканчивающийся инструкцией перехода (условного или безусловного). Кроме того, собирается и сохраняется информация о считанных инструкциях для каждого блока и количество осуществлённых в трассе переходов для каждого отдельного перехода. Сокращение времени исполнения программы достигается за счет пропорционального уменьшения числа итераций циклов в определённое число раз, определяемое заданным фактором сокращения R. Результатом работы первой стадии является слепок программы, который представляет из себя граф потока управления SFGL (Statistical Flow Graph with Loop annotaion) и таблицы архитектурных метрик, описывающие наполнение каждого блока.

Таблицы наполнения, описывающие микроархитектурные особенности участков кода с помощью стохастического генератора тестов, могут быть обратно преобразованы в тестовые последовательности. Такие синтетические тестовые последовательности будут воспроизводить то же поведение, что и оригинальный фрагмент. На последней стадии производится итоговый обход графа и запись получаемого кода в виде набора ассемблерных инструкций. Они в свою очередь генерируются исходя из наполнения базовых блоков по типам инструкций. Каждое ветвление во время непосредственного исполнения теста сопровождается предварительным считыванием из памяти значения, соответствующего выбору определённой ветви в каждом отдельном случае.

B. Максимизация покрытия

Полученные на этом этапе результаты уже можно использовать в виде тестов для верификации микропроцессора, однако из-за случайной природы генератора часть уникальных тестовых ситуаций, которые могут произойти в реальном приложении, может быть упущена. Поэтому цель данной работы состоит в том, чтобы сосредоточить максимально возможное количество тестовых ситуаций из приложения пользователя в рамках одного теста. То есть создавать концентрированный с точки зрения покрытия синтетический аналог реального приложения – его экстракт. Именно для максимизации покрытия, то есть создания наиболее емкого теста, используются алгоритмы машинного обучения. Меняя коэффициенты в таблицах наполнения, описывающих структуру теста, генератор будет создавать отличающийся от предыдущей итерации тест. Для того, чтобы оценить, насколько данные изменения были полезны, тесты запускаются в RTL-симуляторе со сбором покрытия. Покрытие оценивается по следующим параметрам:

- Code
  - Block
  - Statement
  - Expression
  - Toggle
- FSM
- Functional
  - Assertions
  - CoverGroup

Таким образом, для каждого изменения шаблона, то есть изменения значения метрик оригинального приложения, мы получаем обратную связь в виде оценки влияния этих изменений на покрытие модели. Эти данные могут быть использованы для обучения с подкреплением для поиска максимума функции Cov: test => coverage, отображающей тесты в покрытие.

Q-learning - это разновидность off-Policy метода. Во время обучения агент может использовать стратегию, которая отличается от той, которую он изучает. Функция состояния-действия $Q(S,A)$ оценивает действие A в состоянии S. Для этого агент обучения использует следующее уравнение.

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)],$$

где S_t – состояние на шаге t, A_t –действие на шаге t, R_{t+1} – награда на шаге $t + 1$, $\alpha \in (0,1]$ - размер шага, $\gamma \in [0,1]$ – влияние будущих шагов на оценку пары текущее состояние-действие.

Поскольку количество пар состояние-действие огромно, функция $Q(S,A)$ аппроксимируется нейронной сетью. Этот алгоритм называется Deep Q-learning. Во время обучения агент использует второй член в приведенном выше уравнении в качестве ошибки, чтобы соответствовать весам нейронной сети. Обычно агент использует жадную стратегию при

запуске, т.е. выбирает случайное действие. Вероятность выбора случайного действия медленно снижается, заставляя агента выбирать действия с наибольшим значением Q.

Определенная точка в многомерном пространстве тестовых параметров в данном случае рассматривается как состояние. Действия инициируют изменение теста, а качестве вознаграждения R агент получает информацию о текущей настройке теста. Выполнив ряд шагов, агент узнает, какая настройка теста обеспечивает максимальный охват. Задача заключается в максимизации данной дискретной функции.

Поскольку структура приложения также сильно влияет на его микроархитектурные особенности, сгенерированные фрагменты синтетического кода упорядочиваются в финальном тесте в соответствии со структурой приложения, полученной на последнем этапе.

При динамическом изменении шаблона в ходе обучении модели покрытие нового теста может не измениться в численном выражении или стать меньше такового на предыдущей итерации даже если были достигнуты ненайденные ранее уникальные точки покрытия. Это может происходит за счет того, что часть точек покрытия из предыдущей итерации была замещена отличающимися новыми. При этом в численном выражении сумма точек покрытия не изменилась. Поэтому для принятия решения необходимо сравнивать не две последующие стадии, а суммарное покрытие всех предыдущих стадий с покрытием текущей стадии.

В связи с этим итоговый алгоритм принятия решений для модели обучения будет сформулирован в следующей формуле:

$$K = \sum_0^N Fn - \sum_0^{N-1} Fn \ ,$$

где $\sum_0^{N-1} Fn$ - сумма всех уникальных точек покрытия на предыдущих стадиях, а $\sum_0^N Fn$ – сумма всех уникальных точек покрытия на всех стадиях. При K > 0 принимается решение о том, что последнее изменение теста было успешным.

Предлагаемый подход был применен для приложений из набора тестов SPEC CPU2000 INT. В качестве рассматриваемой модели был взят одноядерный микропроцессор MIPS-подобной архитектуры. Для каждого отдельного теста был создан синтетический клон приложения, который затем был итеративно реорганизован нейронной сетью. Клоны и экстракты приложений (до и после работы нейронной сети) вместе с оригинальными приложениями были повторно запущены на симуляторе с анализом покрытия с использованием параметров, упомянутых ранее. Результаты приведены в таблице 1. Столбец “Клон” соответствует тестам, сгенерированным из оригинальных приложений по их исходным метрикам до работы нейронной сети. Полученное покрытие коррелирует с результатами, полученными на оригинальных задачах. Столбец «Экстракт» соответствует результатам применения нейронной сети к сгенерированным тестам. Как видно из таблицы, полученный экстракт приложения на несколько порядков меньше оригинального приложения, а значит время, затраченное на его симуляцию, значительно сокращается. Однако его покрытие, как правило, больше, чем у оригинального приложения.

Таблица 1

Результат замеров покрытия у оригинального приложения, синтетического клона приложения и синтетического экстракта приложения

SPEC2000 *INT TESTS*	***Покрытие RTL-модели***				
	Оригинальное приложение, длинна: 10^6	Клон		Экстракт	
	Покрытие, %	Покрытие, %	Длина, N*10^3	Покрытие, %	Коэффициент сокращения
164.gzip	68,3	62,6	90	77.1	11,1
175.vpr	65,9	59,2	7	72.8	142,9
176.gcc	69,4	62,8	9	74.2	111,1
181.mcf	60,7	55,9	12	68.7	83,3
186.crafty	70,2	62,1	8	69.4	125,0
197.parser	70,9	59,2	5	72.3	200,0
252.eon	73,2	67,4	8	77.0	125,0
254.gap	70,3	63,0	15	74.8	66,6
255.vortex	39,9	41,5	272	53.4	3,7
256.bzip	70,0	61,1	6	75.4	166,6

Конечный результат, в котором нейронная сеть не может увеличить охват исходной программы, будет использоваться в качестве высоко компактного набора шаблонов, используемого как для быстрой оценки качества изменений, внесенных в RTL-проект на pre-silicon стадии, так и в качестве альтернативы выполнению реальных задач для обнаружения неисправностей на post-silicon стадии.

Сам процесс обучения нейронной сети также используется для верификации RTL-модели микропроцессора. Так как для обучения нейронной сети необходимо собирать данные о покрытии RTL-модели, то сгенерированный на каждой итерации тест запускается на симуляторе. Результаты запуска теста могут быть сравнены с результатами, полученными на эталонной модели (Instruction Set Simulator) для поиска ошибок.

В процессе обучения также создается регрессионная база тестирования. В случае, если суммарное покрытие будет увеличено при добавлении нового теста, то последнее изменение шаблона было успешным и последний тест должен быть сохранен в базу тестов. Таким образом, конечным результатом – т.е. итоговой тестовой базой будет являться набор всех тестов на тех итерациях, на которых повышалось покрытие.

База тестов также может быть сокращена без потерь итогового покрытия. Это является следствием того, что последующие итерации тестовой генерации могут включать в себя все тестовые ситуации из предыдущих итераций. Для того, чтобы сократить число тестов необходимо по очереди (с начала списка тестов) начать исключать тесты и следить за тем, изменилось ли тестовое покрытие.

Следует отметить, что не все приложения могут быть свернуты таким образом. Например, программы, использующие механизмы JIT, не могут быть сведены к минимуму на данном этапе разработки проекта из-за их организации.

IV. Заключение

Инструменты машинного обучения повсеместно внедряются во все стадии разработки и тестирования микропроцессоров. Основная цель данной работы - дополнить маршрут стохастического тестирования за счет перевода задач по анализу приложений пользователя и направления процесса верификации с инженера верификатора на нейронную сеть. То есть вместо непрерывного выполнения псевдослучайных направленных тестов на основе шаблонов, созданных инженером вручную, цель в том, чтобы автоматизировать процесс поиска нового тестового знания. Поскольку каждая последующая итерация тестов, созданных данным генератором, будет отличаться от предыдущей, процесс нахождения максимального покрытия также будет являться процессом тестирования.

Поддержка

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме FNEF-2022-0004.

Литература

[1] N.A. Grevtsev, P.A. Chibisov, "A novel technique for atomic instructions functional verification using lock contention analysis," 2019 IEEE East-West Design & Test Symposium (EWDTS)

[2] Multicore Processor Models Verification in the Early Stages N.A. Grevtsev, P.A. Chibisov SELECTED ARTICLES of the VIII All-Russia Science&Technology Conference MES-2018 Part 1

[3] G. Huang, J. Hu, Y. He, J. Liu, M. Mingyuan, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, X. Ning, Y. Ma, H. Yang, B. Yu, H. Yang, Y. Wang, Machine learning for electronic design automation: A survey, ACM Transactions on Design Automation of Electronic Systems 26 (2021) 1–46.

[4] H. A.-A. Robbie Sadre, Machine learning applications in digital system verification, 2019,pp. 21–27.

[5] Q. Guo, T. Chen, H. Shen, Y. Chen, W. Hu, On-the-fly reduction of stimuli for functional verification, in: 2010 19th IEEE Asian Test Symposium, 2010, pp. 448–454.

[6] L.-C. Wang, Data mining in functional test content optimization, in: The 20th Asia and South Pacific Design Automation Conference, 2015, pp. 308–315.

[7] N. Pfeifer, B. V. Zimpel, G. A. G. Andrade, L. C. V. dos Santos, A reinforcement learning approach to directed test generation for shared memory verification, in: 2020 Design, Automation Test in Europe Conference Exhibition (DATE), 2020, pp. 538–543.

[8] L. Peled, U. C. Weiser, Y. Etsion, A neural network prefetcher for arbitrary memory access patterns, ACM Transactions on Architecture and Code Optimization (TACO) (2020) 1 – 27.

[9] J. Bergeron, Writing Testbenches: Functional Verification of HDL Models, Springer US, 2012.

[10] D. A. Anisi, Accelerating Coverage Closure For Hardware Verification Using Machine Learning, Master’s thesis, 2019.

[11] M. Fajcik, P. Smrz, M. Zachariasova, Automation of processor verification using recurrent neural networks, in: 2017 18th International Workshop on Microprocessor and SOC Test and Verification (MTV), 2017, pp. 15–20..

[12] B. W. Mammo, Reining in the functional verification of complex processor designs with automation, prioritization, and approximation, Ph.d. dissertation, University of Michigan, 2017

[13] W. Chen, Data Learning Methodologies for Improving the Efficiency of Constrained Random Verification, Ph.d. dissertation, UC Santa Barbara, 2014.

[14] W. Hughes, S. Srinivasan, R. Suvarna, M. Kulkarni, Optimizing design verification usingmachine learning: Doing better than random, 2019.

[15] K. H. Mami Miyamoto, Finding effective simulation patterns for coverage-driven verification using deep learning, in: SASIMI 2016 Proceedings, 2016, pp. 335–340.

[16]Гревцев Н.А., Хисамбеев И.Ш., Чибисов П.А. Исследование способов повышения эффективности стохастического тестирования моделей микропроцессоров // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 2. С. 8-15.

[17] Гревцев Н.А., Краснюк А.А., Орлов Д.О., Чибисов П.А. Генерация синтетических клонов приложений пользователя для функциональной верификации // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2021. Выпуск 3. С. 50-56. doi:10.31114/2078-7707-2021-3-50-56.

Functional Verification of Microprocessors using Machine-Learning Methods

N.A. Grevtsev

SRISA RAS, Moscow, ngrevcev@cs.niisi.ras.ru

***Abstract* — Nowadays, the major technology companies (such as ARM, Intel, IBM, and others) and universities are studying the practical applicability of machine learning techniques for microprocessor model functional verification. However, recent scientific research and experiments on machine learning are performed in the fields of formal verification, random test generation with CSP solvers, and symbolic execution for post-silicon debugging or bug localization only. The novelty of the proposed approach comes from the usage of machine learning (ML) for user applications behavior imitation in random test generation flow. Its primary goal is to improve the quality of RTL-model verification by directed test generation. In this work, the basic concept of machine learning techniques for functional verification at the system level and base framework of the proposed methodology are presented. The key research findings are to provide the possibility of user applications behavior imitation on the executable machine code level to increase the coverage and automation of the hard-to-find bugs analysis process during the random test generation flow.**

***Keywords* — Functional verification, machine learning, coverage-driven test generation, executable machine code analysis, deep learning.**

REFERENCES

[1] N.A. Grevtsev, P.A. Chibisov, "A novel technique for atomic instructions functional verification using lock contention analysis," 2019 IEEE East-West Design & Test Symposium (EWDTS)

[2] Multicore Processor Models Verification in the Early Stages N.A. Grevtsev, P.A. Chibisov SELECTED ARTICLES of the VIII All-Russia Science&Technology Conference MES-2018 Part 1

[3] G. Huang, J. Hu, Y. He, J. Liu, M. Mingyuan, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, X. Ning, Y. Ma, H. Yang, B. Yu, H. Yang, Y. Wang, Machine learning for electronic design automation: A survey, ACM Transactions on Design Automation of Electronic Systems 26 (2021) 1–46.

[4] H. A.-A. Robbie Sadre, Machine learning applications in digital system verification, 2019,pp. 21–27.

[5] Q. Guo, T. Chen, H. Shen, Y. Chen, W. Hu, On-the-fly reduction of stimuli for functional verification, in: 2010 19th IEEE Asian Test Symposium, 2010, pp. 448–454.

[6] L.-C. Wang, Data mining in functional test content optimization, in: The 20th Asia and South Pacific Design Automation Conference, 2015, pp. 308–315.

[7] N. Pfeifer, B. V. Zimpel, G. A. G. Andrade, L. C. V. dos Santos, A reinforcement learning approach to directed test generation for shared memory verification, in: 2020 Design, Automation Test in Europe Conference Exhibition (DATE), 2020, pp. 538–543.

[8] L. Peled, U. C. Weiser, Y. Etsion, A neural network prefetcher for arbitrary memory access patterns, ACM Transactions on Architecture and Code Optimization (TACO) (2020) 1 – 27.

[9] J. Bergeron, Writing Testbenches: Functional Verification of HDL Models, Springer US, 2012.

[10] D. A. Anisi, Accelerating Coverage Closure For Hardware Verification Using Machine Learning, Master's thesis, 2019.

[11] M. Fajcik, P. Smrz, M. Zachariasova, Automation of processor verification using recurrent neural networks, in: 2017 18th International Workshop on Microprocessor and SOC Test and Verification (MTV), 2017, pp. 15–20..

[12] B. W. Mammo, Reining in the functional verification of complex processor designs with automation, prioritization, and approximation, Ph.d. dissertation, University of Michigan, 2017

[13] W. Chen, Data Learning Methodologies for Improving the Efficiency of Constrained Random Verification, Ph.d. dissertation, UC Santa Barbara, 2014.

[14] W. Hughes, S. Srinivasan, R. Suvarna, M. Kulkarni, Optimizing design verification usingmachine learning: Doing better than random, 2019.

[15] K. H. Mami Miyamoto, Finding effective simulation patterns for coverage-driven verification using deep learning, in: SASIMI 2016 Proceedings, 2016, pp. 335–340.

[16] Grevtsev N.A., Khisambeev I.Sh., Chibisov P.A. Methods to improve efficiency of microprocessor model stochastic tests // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2016. Part 2. P. 8-15.

[17] Grevtsev N.A., Krasnyuk A.A., Orlov D.O., Chibisov P.A. User applications synthetic clones generation for functional verification // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2021. Issue 3. P. 50-56. doi:10.31114/2078-7707-2021-3-50-56