УДК 004.315 DOI: 10.22184/NanoRus.2019.12.89.93.98

МЕТОДИКА СОЗДАНИЯ МОДЕЛЕЙ АРИФМЕТИЧЕСКИХ СОПРОЦЕССОРОВ

METHODS OF CREATING ARITHMETIC COPROCESSORS MODELS

ЗУБКОВСКИЙ ПАВЕЛ СЕРГЕЕВИЧ
zubkovsky@niisi.ras.ru

ФГУ ФНЦ НИИСИ РАН
117218, Москва, Нахимовский просп., 36, к. 1

ZUBKOVSKIY PAVEL S.
zubkovsky@niisi.ras.ru

SRISA RAS
bld. 1, 36 Nakhimovskiy Ave., Moscow, 117218

Предложена методика создания моделей арифметических сопроцессоров, разобраны основные этапы проектирования арифметических сопроцессоров, приводятся примеры арифметических сопроцессоров, разработанных по данной методике.

Ключевые слова: арифметические сопроцессоры; RTL-модель; доступ к памяти; тестирование.

The paper highlights a method of creating models of arithmetic coprocessors, analyzes the main stages of designing arithmetic coprocessors and gives examples of arithmetic coprocessors developed by this method.

Keywords: arithmetic coprocessors; RTL-model; memory access; testing.

В настоящее время развитие ядерной и термоядерной энергетики, электроники, авиа- и ракетостроения, биотехнологий и так далее стало невозможным без проведения полномасштабных инженерных расчетов, требующих использования супер-ЭВМ эксафлопного класса (10^{18} оп/сек). Существующие универсальные микропроцессоры не позволяют достичь требуемой производительности. Можно выделить два основных направления создания высокопроизводительных микропроцессоров: это создание многоядерных микропроцессоров со встроенным графическим процессором [1] и использование специализированных сопроцессоров, ориентированных на выделенные математические функции [2]. В данной работе предлагается методика создания моделей арифметических сопроцессоров, ориентированных на решение выделенного класса задач.

Методика создания моделей арифметических сопроцессоров состоит из следующих этапов:

1. Определение области применения (разработка ТЗ). На этом этапе формируются основные требования к арифметическому устройству, а именно: ориентированность под выделенный класс задач, требования по производительности, потреблению.
2. Разработка архитектуры арифметического устройства. На этом этапе определяются базовые арифметические и логические операции, организация работы с памятью, модель взаимодействия с ядром микропроцессора.
3. Создание поведенческой модели арифметического устройства (эталонного эмулятора, написанного на языке высокого уровня).
4. Разработка библиотек (адаптация существующих библиотек) для решения арифметических задач; реализация задач, которые должны решаться с помощью разрабатываемого арифметического устройства.
5. Разработка логической модели арифметического устройства (RTL-модели) в составе микропроцессора.
6. Создание тестового программного обеспечения.

ОПРЕДЕЛЕНИЕ ОБЛАСТИ ПРИМЕНЕНИЯ (РАЗРАБОТКА ТЗ)

На этом этапе изучается круг задач, которые будут решаться с помощью разрабатываемого арифметического устройства, формируются требования по пиковой и реальной производительности устройства. Изучаются существующие аналоги в целях заимствования архитектурных решений.

В зависимости от решаемых задач определяются типы данных, с которыми будет работать арифметическое устройство (целые, вещественные или комплексные числа, разрядность данных), и точность вычислений (одинарная или двойная). Разные задачи используют разные типы данных. Изображение, криптография, видео, RAID-массивы — примеры задач, использующих целочисленные данные. Газодинамика, инженерные расчеты, гидроакустика — задачи, работающие с вещественными данными разной точности. Архитектура вычислительного устройства будет разрабатываться с учетом типов данных, определенных в ТЗ.

Помимо перечисленных выше требований на стадии разработки ТЗ важно определить способ работы с арифметическим устройством. Первый способ предполагает программирование задач на языке ассемблера с использованием специальных команд сопроцессора. Последовательность команд раскладывается вручную с учетом особенностей архитектуры арифметического устройства и микропроцессора в целом. Второй способ опирается на использование библиотечных функций, ядра которых переписываются под конкретную архитектуру арифметического устройства и вызываются из программ, написанных на языке более высокого уровня. И третий способ возлагает функцию взаимодействия с арифметическим устройством на компилятор.

Первый способ характеризуется большими временными затратами на программирование и оправдан, если с помощью разрабатываемого арифметического устройства планируется решать ограниченное количество задач, которые можно запрограммировать единожды. Третий способ повышает сложность разработки компилятора. Наиболее оптимальным с точки зрения автора является второй способ взаимодействия, требующий умеренных временных затрат на адаптацию библиотечных функций, а также позволяющих использовать общеизвестные тесты производительности, такие как LINPACK.

РАЗРАБОТКА АРХИТЕКТУРЫ АРИФМЕТИЧЕСКОГО УСТРОЙСТВА

Базовые арифметические и логические операции

К базовым арифметическим операциям относятся как универсальные операции, такие как сложение, вычитание, умножение

и деление, так и специальные операции для повышения эффективности решения выделенного класса задач. Использование специальных операций может повысить производительность микропроцессора при решении определенных задач до 90 %, но ведет к усложнению программирования. Поэтому, определяя базовые арифметические операции, необходимо учитывать, какой доступен уровень программирования (будут ли разрабатываться и использоваться вручную адаптированные библиотеки, которые можно в дальнейшем применять в компиляторе). Арифметические операции могут быть выполнены с использованием синхронных схем, а также в базисе самосинхронной логики. Использование самосинхронных схем позволяет повысить энергоэффективность и надежность разрабатываемого устройства. Тем не менее самосинхронные схемы имеют и ряд недостатков: сложность временной модели, нехватка автоматизации проектирования, малое число наработок и, как следствие, опыта.

НИИСИ РАН совместно с ИПИ РАН провели работы по проектированию функционального блока, ответственного за деление с плавающей точкой. Блок выполнен полностью по самосинхронному маршруту и используется параллельно со своим синхронным аналогом. При примерно равном быстродействии синхронного и самосинхронного решений зафиксировано небольшое снижение потребляемой мощности [8].

Помимо арифметических операций вычислительное устройство может выполнять логические операции или операции декомпозиции.

Организация работы с памятью

Выбор эффективного механизма доступа к данным арифметическими сопроцессорами должен определяться исходя из выделенного класса задач. Если алгоритмы обработки данных запрограммированы в виде ассемблерных ядер и оптимизированы под конкретную архитектуру микропроцессора, то целесообразно использовать локальную память с каналами ПДП (прямого доступа к памяти) для обмена данными между локальной и внешней памятью. К преимуществам данного подхода относится отсутствие ограничений максимальной ширины вектора, гарантированный доступ с минимальной длительностью и периодичностью в один такт. В качестве локальной памяти может использоваться отдельная накристальная память или кеш-память 2-го уровня, отключенная от ядра процессора. Преимуществом отдельной накристальной памяти является то, что при работе с ней кеш-память 2-го уровня остается доступной процессору. Но не всегда есть возможность расположить такую дополнительную память на кристалле. В то же время отключение внутренней памяти данных негативно сказывается на производительности операционной системы и пользовательских приложений, скомпилированных стандартными средствами.

Заполнение внутренней памяти и выгрузка данных из нее производятся по каналу прямого доступа к памяти из внешней памяти процессора. Дополнительное повышение эффективности программирования можно получить, если данные организованы нужным образом уже на стадии загрузки во внутреннюю память. Например, в ряде алгоритмов работы с матрицами возникает необходимость транспонировать отдельные участки этих матриц произвольного размера перед загрузкой в сопроцессор.

Решить эти задачи призван контроллер прямого доступа в память (ПДП) с возможностью трехмерной выборки данных во внешней памяти и независимого трехмерного размещения данных во внутренней памяти.

Внутреннюю память можно разделить на две половины, которые могут работать одновременно независимо друг от друга с контроллером ПДП и инструкциями загрузки/сохранения сопроцессора. Во время выполнения вычислительного цикла обмен данными производится между регистровым файлом сопроцессора и одной половиной внутренней памяти. В другой половине в это время может происходить выгрузка результатов предыдущего вычислительного цикла и загрузка данных для следующего.

Если планируется использовать алгоритмы, запрограммированные на языках более высокого уровня и скомпилированные стандартными средствами с подключением математических библиотек, то целесообразно использовать механизм загрузки/сохранения с использованием кеш-памяти данных. Тут возможны два варианта работы с памятью: с использованием всех уровней кеш-памяти данных и с использованием только кеш-памяти данных второго уровня с обходом кеш-памяти данных первого уровня. Выбор варианта работы можно осуществить с помощью выбора политики кеширования данных. Обращение в кеш-память второго уровня при промахе в кеш-память первого уровня или при использовании кеш-памяти только второго уровня приводит к остановке конвейера на некоторое количество тактов — от 5 до 20, в зависимости от реализации. При использовании кеш-памяти данных первого уровня недостатком является ограничение максимальной ширины вектора шириной обращений в кеш-память первого уровня. Для использования данного механизма необходима разработка эффективной подкачки данных в кеш-память.

Выбор наиболее оптимального алгоритма подкачки данных основывается на анализе алгоритмов, которые будут выполняться на сопроцессоре, особенности доступа к памяти в их программной реализации. Типичными задачами для арифметических сопроцессоров являются задачи линейной алгебры, а также преобразование Фурье. К задачам линейной алгебры относятся операции над векторами и матрицами, получение LU-разложения матрицы, вычисление определителя матрицы, решение систем линейных уравнений и др. Для таких задач характерно расположение значений в памяти с некоторым шагом. Данные могут следовать друг за другом, если шаг равен 1 (такое расположение данных будем называть непрерывным), или располагаться в памяти с шагом, большим 1. Примерами непрерывного расположения данных могут служить строки матрицы в Си-программе, а примерами с шагом, большим 1, — ее столбцы.

Для ускорения команд загрузки используют «потоковые» буферы, являющиеся эффективными благодаря возможности определять потоки данных и заранее подгружать необходимые данные. Используют потоковые буферы как с фиксированным шагом, так и с предсказанием адреса загружаемых данных. «Потоковые» буферы были предложены Jouppi [8] для предварительной выборки данных при промахах в кеш-память. При промахе в кеш-память данные по последующим адресам записываются в «потоковый» буфер. Затем загружаются данные по следующим адресам, пока буфер не будет заполнен. В дальнейшем «потоковый» буфер был усовершенствован и шаг, с которым буфер начитывал данные, хранился в таблице, адресуемой PC. Сейчас для формирования адреса, по которому будет осуществлена предвыборка, используют предсказатель, который генерирует адрес для запроса в память и чтения данных в буфер. Предсказание осуществляется на основе истории последних

обращений. Для анализа могут быть задействованы от двух до четырех обращений в память. Разница между адресами называется шагом и записывается в таблицу истории. Этот шаг используется в качестве индекса к другой таблице, которая хранит предсказанный базовый адрес. Адрес для запроса формируется путем сложения базового адреса и шага.

Еще одна модель, которая используется для предвыборки данных, — это предвыборка на основе востребованности данных. Ранним примером такой предвыборки является предвыборка следующей строки кеш-памяти, где каждый блок кеш-памяти помечен тэг-битом, показывающим, когда необходимо загрузить следующий блок. Когда происходит предвыборка блока, его значению тег-бита присваивается 0. При обращении к этим данным при тег-бите, равном 0, происходит предвыборка данных по последующему предсказанному адресу и тег-бит устанавливается равным 1.

Для минимизации загрузок на шине предвыборка ограничивается механизмом, основанным на точности предсказаний. В этом механизме используется двухбитный счетчик для каждого предсказанного адреса. Это необходимо для того, чтобы предотвратить предвыборку с плохой историей предсказаний в прошлом. Если данные вытесняются из буфера и не были востребованы, соответствующий счетчик инкрементируется. Если загруженный в буфер блок используется, то соответствующий данной записи счетчик декрементируется. Когда старший разряд счетчика установлен, соответствующая запись в таблице предсказаний запрещена, но запросы по этим адресам все равно отслеживаются; таким образом, запись может вновь стать разрешенной, когда начнет делать правильные предсказания.

Помимо буферов для предварительного вычитывания данных в современных микропроцессорах используются буферы для сохраняемых данных, которые способствуют ускорению функции копирования, так как запись происходит не сразу в память, а в буфер, что позволит на этом такте начать обработку следующего запроса. При освобождении памяти буфер будет выдавать хранящуюся в нем информацию. Если к моменту прихода второго запроса на запись по последующему адресу запись в память еще не произошла, происходит склейка и обращение в память на запись объединяется в одно. Это приводит к уменьшению количества запросов на запись в память и, следовательно, к уменьшению общей загруженности тракта ОЗУ.

Буферизация данных может проводиться на разных уровнях подсистемы памяти: в системном контроллере, а также на всех уровнях кеш-памяти.

Для эффективной реализации алгоритма быстрого преобразования Фурье (БПФ) полезно предусмотреть режим так называемой бит-реверсной автоинкрементной адресации, а также вычитывание элементов вектора с переменным шагом. При выполнении БПФ с прореживанием по времени результирующий вектор получается на месте исходного путем многократного поэтапного выполнения над различными его парами элементов одной и той же базовой операции, так называемой бабочки Фурье [9]. Алгоритм можно разделить на два этапа, на первом из которых выполняется бит-реверсная перестановка элементов исходного вектора. На втором этапе действия выполняются над парами элементов вектора, отстающих друг от друга на определенном расстоянии, измеряемом в числе элементов вектора. По мере выполнения алгоритма шаг следования элементов возрастает от 1 (соседние элементы вектора) до величины половины общей длины вектора.

Модель взаимодействия с ядром микропроцессора

Определяется степень автономности арифметического устройства от ядра микропроцессора, а именно:

- прохождение каждой команды через конвейер инструкций микропроцессора или запуск одной командой микропроцессора обработки алгоритма вычислений;
- использование виртуальной адресации;
- использование кеш-памяти;
- использование исключительных ситуаций, прерывания.

Для загрузки и сохранения данных в арифметическом сопроцессоре могут использоваться кешируемые обращения в память по виртуальным адресам. Трансляцию виртуального адреса в физический осуществляет буфер трансляции адресов TLB, а для ускорения доступа к данным может использоваться кеш-память 1-го и 2-го уровня.

Стандарт IEEE754 [7] разработан ассоциацией IEEE (Institute of Electrical and Electronics Engineers) и используется для представления действительных чисел (чисел с плавающей точкой) в двоичном коде. Наиболее используемый стандарт для вычислений с плавающей точкой используется многими микропроцессорами и логическими устройствами, а также программными средствами.

Если работа арифметического устройства удовлетворяет стандарту IEEE754, то предполагается обработка пяти исключительных ситуаций:

- Invalid operation (некорректная операция);
- Division by Zero (деление на ноль)
- Overflow (переполнение);
- Underflow (потеря значимости);
- Inexact operation (потеря точности).

Исключительная ситуация «потеря точности» является наиболее трудоемкой с точки зрения предсказания, поскольку зачастую ее признак вычисляется на последней стадии — стадии округления. Описание алгоритма предсказания исключительной ситуации «потеря точности» можно найти в работе [6].

После определения ТЗ и разработки архитектуры арифметического устройства все остальные этапы проектирования могут идти параллельно, тем самым экономя время разработки. По результатам этих этапов проектирования корректируется архитектура арифметического устройства.

СОЗДАНИЕ ПОВЕДЕНЧЕСКОЙ МОДЕЛИ АРИФМЕТИЧЕСКОГО УСТРОЙСТВА

Создание поведенческой модели (также называемой образцовым эмулятором или С-моделью) является важным этапом разработки и несет насколько функций. Так как разработка поведенческой модели занимает немного времени, ее можно использовать для оценки архитектурных решений еще до того, как будет готова RTL-модель. Поведенческая модель сохраняет все изменения состояния микропроцессора после каждой инструкции или после каждого такта работы микропроцессора. На протяжении всего времени разработки модели арифметического устройства происходит сравнение состояний поведенческой модели и RTL-модели при тестировании. Кроме того, с помощью поведенческой модели существует возможность сделать «срез» состояния микропроцессора в произвольный момент времени и запустить выполнение программы на RTL-модели уже с нужного момента, а не с самого начала. Это необходимо, если была найдена ошибка на аппаратном ускорителе или ПЛИС-прототипе.

РАЗРАБОТКА БИБЛИОТЕК

На данном этапе разрабатывается библиотека подпрограмм, которая будет использоваться для выделенного класса задач.

Целесообразно использовать уже существующие библиотеки, адаптировав их под архитектуру разрабатываемого арифметического устройства (например библиотеку подпрограмм линейной алгебры GotoBLAS) [9]. В НИИСИ РАН проводятся работы по адаптации библиотеки GotoBLAS для эффективной работы на разрабатываемых арифметических устройствах [4]. Наиболее затратные по времени исполнения фрагменты алгоритма реализации библиотечных функций запрограммированы в виде ассемблерных ядер и оптимизированы под конкретную архитектуру процессора. Библиотека оптимизирована под различные архитектуры, в том числе и под архитектуру MIPS64. Библиотека распространяется в рамках программы открытого математического обеспечения, что позволяет расширять функциональные возможности библиотеки, оптимизируя ее под другие архитектуры процессоров. Подход к выполнению адаптации библиотеки BLAS заключается в перепрограммировании ассемблерных ядер с учетом специфики системы команд арифметического сопроцессора, при этом сохраняется совместимость перепрограммированных ядер со структурой библиотеки.

При адаптации библиотеки к архитектуре арифметического сопроцессора в ассемблерные программы ядер, которые оптимизированы под архитектуру MIPS64, добавляются ветви, реализующие выполнение программы ядра на арифметическом сопроцессоре. Программирование этих ветвей выполняется в системе команд арифметического сопроцессора и с учетом реализованного в программе узла алгоритма.

Для оценки эффективности выполненной адаптации библиотеки сравниваются время выполнения программ исходного и адаптированного ядра и определяется коэффициент ускорения работы ядра. Аналогичные оценки выполняются и для функции, которая при выполнении вызывает необходимый набор адаптированных ассемблерных ядер.

РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ АРИФМЕТИЧЕСКОГО УСТРОЙСТВА (RTL-МОДЕЛИ) В СОСТАВЕ МИКРОПРОЦЕССОРА

Для проведения успешного функционального тестирования модель арифметического устройства интегрируется в состав модели микропроцессора. Это позволяет запускать на арифметическом устройстве тестовые программы: рукописные тесты разработчика, псевдослучайные тесты, а также реальные вычислительные задачи, скомпилированные с использованием математических библиотек. RTL-модель, так же как и поведенческая модель, снабжена механизмом сохранения всех изменений состояния. Это позволяет использовать тесты без самопроверки, а также лучше локализовать ошибки, сравнив состояния поведенческой и RTL-моделей. Для увеличения скорости обработки данных могут использоваться ускорители программного моделирования (Palladium фирмы Cadance, VStation PRO и Veloce фирмы Mentor Graphics), а также ПЛИС-прототипы.

СОЗДАНИЕ ТЕСТОВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Верификация арифметического устройства производится на разных этапах его разработки (см. табл. 1). При разработке отдельных арифметических блоков основу верификации составляют тесты, направленные на проверку правильности выполнения арифметических действий над числами. В процессе дальнейшей разработки арифметического сопроцессора производится объединение отдельных арифметических модулей в общий сопроцессор и подключение вспомогательных и управляющих устройств, таких как декодер команд, регистровый файл, управляющие регистры и регистры состояния. На этом этапе основное внимание уделяется тестам, предоставляющим возможность проверить сопряжение отдельных блоков в общую систему, установить правильность декодирования инструкций и обмена данными с регистровым файлом. Отдельным этапом разработки является установка и настройка соединений арифметического сопроцессора с основным ядром микропроцессора. Здесь в основном используются тесты, содержащие арифметические инструкции сопроцессора, а также инструкции обмена данными между ядром, сопроцессором и памятью. Отдельное место занимают тестовые программы, позволяющие оценить не только производительность арифметического сопроцессора, но и производительность системы, состоящей из ядра микропроцессора и арифметического сопроцессора.

На первом этапе разработки арифметического сопроцессора для проверки отдельных арифметических модулей применяются тестовые наборы воздействий (так называемое модульное тестирование). Эти наборы создаются индивидуально для каждого модуля с учетом входных и выходных шин данных и управляющих сигналов. Недостатком такого метода верификации является значительная трудоемкость написания тестовых воздействий, невозможность создания универсальных наборов, применимых для разных арифметических модулей.

После соединения отдельных арифметических модулей в рамках арифметического сопроцессора и подключения регистрового файла и декодера команд возможности тестирования сравнительно ограничены. Единственной возможностью является моделирование входных сигналов арифметического сопроцессора при работе ядра микропроцессора. Этот подход позволяет поверить сопряжение арифметических блоков, правильность декодирования инструкций и подключения регистрового файла.

Таблица 1. Верификация арифметического сопроцессора на разных этапах разработки

	Этапы разработки арифметического сопроцессора	Способы тестирования
1	Разработка отдельных арифметических блоков	Индивидуальные тестовые воздействия для каждого арифметического блока
2	Объединение отдельных арифметических блоков в общий арифметический сопроцессор	Моделирование сигналов интерфейса арифметического сопроцессора и ядра микропроцессора
3	Соединение арифметического сопроцессора с ядром микропроцессора	Тестовые программы
4	Оптимизация работы арифметического сопроцессора в составе микропроцессора	Тесты производительности

Основным этапом верификации арифметического сопроцессора является использование тестовых программ, применение которых становится возможным только после подключения арифметического сопроцессора к ядру микропроцессора. Можно рассматривать три основных типа тестовых программ, применяемых на данном этапе.

К первому типу относятся программы, направленные на проверку правильности выполнения одной вполне определенной инструкции арифметического сопроцессора. Такие программы, как правило, пишутся на языке Ассемблера. Например, программа может состоять из последовательных вызовов одиночных инструкций с различными операндами с последующим контролем полученных результатов. Подобные программы создаются с минимальным использованием других инструкций сопроцессора для того, чтобы избежать наложения ошибок и затруднений при диагностике. Для исключения ошибок, связанных с работой блока обратных связей по данным, учитывая количество тактов, необходимых для выполнения одной инструкции арифметического сопроцессора, требуется разделить тестируемые инструкции несколькими инструкциями nop. Очевидно, что из-за невозможности протестировать все возможные инструкции арифметического сопроцессора со всеми возможными их операндами необходимо разбить все множество значений операндов на группы. Для чисел с плавающей запятой можно выделить следующие группы: ±0; денормализованные числа; нормализованные числа, близкие к минимуму, ±1,0, максимуму; ±∞; QNaN; SNaN. В этом случае тесты для 32-разрядного (одинарная точность) и 64-разрядного (двойная точность) формата записи числа будут использовать одни и те же числовые значения, записанные в своем формате.

Ко второму типу можно отнести программы, предназначенные для выявления ошибок при вызове и обработке исключительных ситуаций, выполнения вычислений с использованием специальных операндов, предусмотренных стандартом [2]. В данном случае, в отличие от предыдущей группы тестов, является оправданным использование в программе комбинаций команд, относящихся к разным типам. Сюда же можно отнести короткие программы, написанные на языке высокого уровня. Для верификации RTL-модели эти тесты представляют особый интерес, так как они очень близки к реальным пользовательским приложениям по набору и сочетанию выполняемых в них моделью инструкций; в большинстве случаев они являются самопроверяющимися, что делает их отличным материалом для тестирования модели в целом и арифметического сопроцессора в частности; с их помощью можно оценить текущее состояние модели, подсчитав суммарное количество правильно работающих тестов. Наборы таких тестов являются доступными, не зависят от архитектуры и типа процессора. Эти тестовые программы позволяют затронуть практически все модули арифметического сопроцессора, найти достаточно редкие ошибки в модели, а также, при том условии что весь набор этих тестов будет запускаться на модели всякий раз, когда были внесены какие-либо изменения в модель, тесты способны предотвратить появление старых, уже исправленных ошибок, а также появление некоторых новых. Также эта группа ценна количеством входящих в нее тестов (несколько сотен программ различной сложности), а также тем, что, изменив условия компиляции исходных текстов программ (скажем, включив оптимизацию по размеру выполняемого кода или по скорости), мы получаем иной набор инструкций, зачастую совершенно не похожий на первоначальный. Поскольку суперскалярные микропроцессоры читают из кеш-памяти одновременно несколько инструкций, то существенным может оказаться расположение этих инструкций друг относительно друга в программе. Поэтому разные условия компиляции тестов действительно приводят к нахождению в модели и в арифметическом сопроцессоре совершенно других типов ошибок.

К третьей группе относятся программы, включающие в себя максимально разнообразные последовательности команд разных типов. Для создания таких программ целесообразно применять генератор псевдослучайных тестов как источник случайных последовательностей команд, которые используют в качестве операндов случайные числа.

Процесс разработки и использования псевдослучайных тестов блока вещественной арифметики можно условно разделить на следующие этапы.

1. Создание шаблона теста. Создается шаблон, который на условном языке описывает последовательность инструкций будущего теста, значения различных настроек генератора, используемые области памяти и начальные состояния регистров. В это описание включаются различные вероятностные распределения: вероятности выбора определенной инструкции из группы инструкций, вероятности для выбора константы-операнда и пр. Значения многих настроек также не фиксированы, а принимаются с определенной вероятностью. Большинство числовых параметров выбирается случайно из некоторого интервала. Таким образом, одному шаблону может соответствовать целый класс (более или менее широкий) тестовых программ. Кроме того, шаблон может включать в себя пользовательские макросы.
2. Генерация тестовой программы, удовлетворяющей шаблону. Созданный шаблон подается на вход генератора псевдослучайных тестов; результатом его работы является ассемблерная программа и модифицированный шаблон, содержащий специальное число — зерно теста. Оно позволяет при необходимости повторно получить эту программу, задав генератору шаблон с этим числом. Заметим, что в процессе верификации такая потребность возникает часто. Таким образом, при помощи определенных средств автоматизации можно быстро получить и запустить большое количество различающихся тестов, которые в то же время могут быть направлены на верификацию сколь угодно узкого или широкого фрагмента арифметического устройства (от одной команды в рамках арифметического модуля до всего набора команд и взаимодействия с ядром микропроцессора).
3. Запуск теста на эталонном эмуляторе и RTL-модели, получение лог-файлов. Полученная тестовая программа запускается на RTL-модели и на образцовом эмуляторе. Трассы инструкций, представляющие собой последовательность выполненных моделью инструкций, сохраняются в текстовый файл.
4. Сравнение логов и поиск расхождения. Если на некоторой инструкции в состояниях моделей появятся различия, то по крайней мере одна из них выполнила некоторое действие некорректно. Это несоответствие отразится в одном или нескольких из сохраненных лог-файлов и может быть выявлено посимвольным сравнением этих файлов. В этом случае тестовая программа, а также шаблон с зерном и файлы результатов сравнения сохраняются для последующего анализа расхождения.

5. Автоматизированное повторение п. 2–4. Вышеописанный процесс автоматизируется с помощью управляющих программ, позволяющих запускать генератор тестов заданное количество раз с одним и тем же шаблоном.

Для оценки быстродействия арифметического сопроцессора и эффективности его работы в составе микропроцессора применяют тесты производительности — специальные программы, измеряющие пиковую и среднюю производительность. Под этими терминами подразумевается количество действий над числами за единицу времени, соответственно в идеальных и реальных условиях.

Автор считает, что в данной работе новыми являются следующие положения и результаты: методика создания моделей арифметических сопроцессоров, позволяющая разработать специализированный сопроцессор, ориентированный на выделенные математические функции. В соответствии с данной методикой в НИИСИ РАН били разработаны: сопроцессор вещественной арифметики, сопроцессор комплексной арифметики и векторный сопроцессор. Разработанные сопроцессоры входят в состав нескольких поколений микросхем, созданных в НИИСИ РАН.

ЛИТЕРАТУРА

1. Nickolls J., Dally W.J. *The GPU computing era* // IEEE Micro. 2010. V. 30. № 2. P. 56–69.
2. Расширения набора команд для архитектуры Intel. http://software.intel.com/ru-ru/intel-isa-extensions (дата обращения: 01.08.2018).
3. Аряшев С.И., Зубковский П.С. Особенности доступа к ОЗУ арифметическими сопроцессорами в микропроцессорах разработки НИИСИ РАН // Международный форум «Микроэлектроника-2016». 2-я научная конференция «Интегральные схемы и микроэлектронные модули». Республика Крым, г. Алушта, 26–30 сентября 2016. — С. 264–269.
4. Аряшев С.И., Зубковский П.С., Кулешов А.С., Цветков В.В. Адаптация библиотеки подпрограмм линейной алгебры GotoBLAS к архитектуре векторного сопроцессора // Суперкомпьютерные технологии (СКТ-2014). Материалы 3-й Всероссийской научно-технической конференции: в 2 т. — Ростов-на-Дону. Издательство Южного федерального университета, 2014. Т. 1. — С. 90–95.
5. Зубковская Н.В., Зубковский П.С., Чибисов П.А. Сопроцессоры вещественной и комплексной арифметики и их тестирование // Проблемы разработки перспективных микро-и наноэлектронных систем — 2010. Сборник трудов / под общ.ред. академика РАН А.Л. Стемпковского. — М.: ИППМ РАН, 2010. — С. 360–363.
6. Зубковский П.С., Ивасюк Е.В. Схема предсказания исключительной ситуации «потеря точности» в модуле операции «умножение с накоплением» // Проблемы разработки перспективных микро- и наноэлектронных систем — 2012. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. — М.: ИППМ РАН, 2012. — С. 521–524.
7. IEEE Standard for Binary Floating Point Arithmetic ANSI IEEE Std. 754-1985.
8. Бобков С.Г., Горбунов М.С., Дьяченко Ю.Г., Рождественский Ю.В., Степченков Ю.А., Сурков А.В. Использование самосинхронной логики для снижения потребляемой мощности и повышения надежности микропроцессоров // Проблемы разработки перспективных микро- и наноэлектронных систем — 2014. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. — М.: ИППМ РАН, 2014. Часть 1. — С. 43–48.
9. Krivutsenko A. 2008. GotoBLAS — Anatomy of a fast matrix multiplication. Computational Science and Engineering Technical University Munich.