БЕЛОРУСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Курсовая работа

по дисциплине: “Архитектура вычислительных систем”

по теме “Фреймворки для симуляции архитектуры микропроцессора”

Выполнил: студент гр. 853505

Мелешко В.М.

Проверил: руководитель работы

Леченко А.В.

Минск 2020

Оглавление

Микропроцессор……2

Зачем симулировать микропроцессор……4

Симулятор ASIM 7

Основные компоненты 8

Модули 9

Выражение альтернативных дизайнов 10

Управление выбором дизайна 11

Преимущества 12

Порты 13

Фидеры 14

Фидеры инструкций 15

Файлы конфигурации 18

Обзор HAsim 19

Высокоуровневые Языки Описания Оборудования 20

Верстак архитектора и модульность 21

Виртуальная платформа LEAP 23

Удаленный запрос-ответ (RRR) 24

Подводя итоги 24

Литература 25

Микропроцессор

Микропроце́ссор — устройство, отвечающее за выполнение арифметических, логических операций и операций управления, записанных в машинном коде, реализованный в виде одной микросхемы или комплекта из нескольких специализированных микросхем (в отличие от реализации процессора в виде электрической схемы на элементной базе общего назначения или в виде программной модели).

Первый электронный компьютер ЭНИАК был изготовлен в США в1945 г. ЭВМ получили широкое распространение,начиная с50-х годов.Прежде это были очень большие и дорогие устройства,используемые лишь в государственных учреждениях и крупных фирмах.Размеры и форма цифровых ЭВМ неузнаваемо изменились в результате разработки новых устройств,называемых микропроцессорами. Микропроцессор (МП) -это программно-управляемое электронное цифровое устройство, предназначенное для обработки цифровой информации и управления процессом этой обработки, выполненное на одной или нескольких интегральных схемах с высокой степенью интеграции электронных элементов. В1970году Маршиан Эдвард Хофф из фирмы Intel сконструировал интегральную схему, аналогичную по своим функциям центральному процессору большой ЭВМ - первый микропроцессор Intel-4004, который уже в1971году был выпущен в продажу.Кристалл представлял собой 4-разрядный процессор,электрическая схема прибора насчитывала 2300 транзисторов.МП работал на тактовой частоте 750 кГц при длительности цикла команд 10,8 мкс. В систему его команд входило всего 46 инструкций.

В начале 70-х годов фирма Motorola в рамках расширения сферы своей деятельности (до этого фирма занималась,в основном,производством автомобильной радиоэлектроники,в соответствии со своим названием), начала разработку своего первого 8-разрядного микропроцессора - 6800.

Motorola и Intel выпустили свои первые □классические□модели МП - 6800 и8080 - приблизительно в одно и то же время,в1974году. 6800 быстро стал основным конкурентом 8080 благодаря некоторым реализованным в нем важным новшествам, например, наличию только одного питающего напряжения и одновременному выпуску вспомогательных периферийных контроллеров для данного процессора.

Аналогично истории фирмы Intel, уйдя из которой основной разработчик МП 8080 создал процессор Z-80, захвативший основную часть рынка 8080, разработчик 6800 также покинул фирму Motorola и создал новый МП на базе архитектуры 6800, названный 6502, и быстро обогнавший по популярности базовую модель. В отличие отZ-80, который содержал 83новые команды, система команд 6502 осталась почти без изменения, однако 6502 имел ряд схемотехнических усовершенствований, снижавших стоимость построенной на базе него системы.

- МП с архитектурой 8080 (с доминированием Z-80) поддерживались единственной стандартной операционной системой (CP/M), что позволяло разрабатывать программное обеспечение, совместимое для аппаратного обеспечения различных производителей,и ускоряло распространение и того и другого;

- МП семейства 6800 (с доминированием 6502) были проще в программировании и имели более высокое быстродействие.

В результате,каждое семейство получило большее распространение в своей области:

- семейство 8080 победило на рынке коммерческих приложений благодаря наличию стандартной операционной системы CP/M,позволяющей запускать разнообразные прикладные программы на платформах разных производителей;

- семейство 6800 захватило рынок персональных компьютеров с повышенными требованиями к скорости обработки графических изображений и звука (производительность МП 6502 с тактовой частотой 2 МГц была в среднем на 20% выше чем уZ-80 с тактовой частотой 4 МГц). Специфика данного рынка приводила к тому,что каждый производитель фактически разрабатывал свою операционную систему, оптимизированную по быстродействию для решения его специфических задач.Поэтому количество производителей аппаратного обеспечения на базе МП этого семейства было значительно меньшим (наиболее известные из них - Apple, Commodore иAtari).

Спустя 4 года постепенно приходит 16-разрядная,представленная МП 8086 фирмы Intel (1978 год)и МП 68000 фирмы Motorola (1979 год). И опять области использования МП двух фирм разделились похожим образом: Intel сохранил лидерство на рынке коммерческих систем благодаря стандартной операционной системе MS-DOS. Наиболее крупным пользователем процессоров Intel стала фирма IBM; Motorola лидировала в областях с повышенными требованиями к быстродействию, как для рабочих станций, так и для персональных компьютеров с повышенными графическими возможностями (таких,например, как Macintosh фирмы Apple). Несмотря на преобладающее использование ОС UNIX, наличие большого количества других несовместимых операционных систем ограничивало более широкое распространение данного семейства.

В1980 году был анонсирован математический сопроцессор 8087, который добавил к архитектуре 8086 почти 60команд вещественной арифметики. В середине 80-х годов Motorola и Intel выпустили полностью 32-разрядные МП, 68020 (68030) и80386, соответственно. Наметившиеся ранее тенденции разделения областей применения МП этих фирм сохранились - семейство процессоров 80x86фирмы Intel продолжает лидировать в коммерческих приложениях за счет большого объема совместимого программного обеспечения,а семейство 680x0фирмы Motorola - в устройствах,требующих повышенного быстродействия при обработке больших объемов информации (например, Multimedia, издательские системы,научные приложения), а также в устройствах скоростного управления в реальном масштабе времени (например,контроллерах VME) и некоторых других.Появившиеся затем усовершенствованные модели обоих семейств (80486 иPentium уIntel, 68040 и68060 уMotorola) практически не изменили общей картины. Подсемейство процессоров i486 (i486SX, i486DX, i486DX2 иi486DX4), в котором сохранились система команд и методы адресации процессора i386, уже имеет некоторые свойства RISC-микропроцессоров.Например,наиболее употребительные команды выполняются за один такт. Появившийся в1993году процессор Pentium ознаменовал собой новый этап в

развитии архитектуры x86, связанный с адаптацией многих свойств процессоров с архитектурой RISC. Именно с процессоров Pentium фирмы Intel началось пятое поколение процессоров семейства x86. По базовой регистровой архитектуре и системе команд процессоры Pentium являются 32- разрядными процессорами, но имеют 64-битную шину данных, шина адреса позволяет адресовать 4Гбайт физической памяти.

К шестому поколению процессоров Intel относятся Pentium Pro, все разновидности процессоров Pentium II/III, а также Celeron. Процессоры Pentium III появились в1999году и являются дальнейшим развитием Pentium Кодовое название до выхода - Katmai. Их главное отличие - раширение набора SIMD- инструкций SSE (Streaming SIMD Extensions, другое название - KNI - Katmai New Insructions), основанное на новом блоке 128-разрядных регистров XMM. Для малобюджетных компьютеров весной 1998 года выпустили облегченный вариант процессора Pentium II, названный Celeron. Процессор Pentium 4 является 32-разрядным представителем семейства x86,по микроархитектуре принадлежащей к новому, седьмому (по классификации Intel) поколению. С программной точки зрения он представляет собой процессор x86с очередным расширением системы команд - SSE2. По набору программно-доступных регистров Pentium 4 повторяет процессор Pentium III.

Зачем симулировать микропроцессор?

Проектирование процессора начинается тогда, когда архитектору предъявляется ряд требований , например, построить высокопроизводительный процессор x86, вышедший из строя, или маломощный процессор in-order процессор ARM. Затем архитектор использует интуицию и знание существующих систем , чтобы определить исходную целевую архитектуру. Эта интуиция должна быть подкреплена подробными количественными исследованиями репрезентативных исходных данных до завершения разработки архитектуры . Этот процесс является итеративным, поскольку каждое исследование приводит к настройке критических параметров архитектуры. В конечном счете целевая архитектура завершена, и может начаться дорогостоящий и трудоемкий этап описания аппаратного обеспечения уровня передачи регистров (RTL). Эти ранние исследования проводятся с использованием имитаторов, называемых моделями производительности, названными так потому, что они дают архитектору детальное представление о динамическом поведении целевой системы от одного такта к следующему. Такая низкая производительность может ограничить разнообразие и длину тестовых пробегов, тем самым снижая уверенность в архитектурных выводах. Столкнувшись с этим, исследователи изучили три взаимодополняющих подхода.

(A) Transform the benchmarks:

SMARTS	Wunderlich et al. [60]	Systematic sampling of execution runs alternating between detailed and functional modes.
SimPoint	Perelman et al. [49]	Automatic extraction of representative regions and weighting of their frequency.
Trace-Graph Workloads	Isshiki et al. [27]	Transform benchmark into optimized trace-graph based on branches.

(B) Abstract the model:

QEMU	Bellard [4]	Functional emulator: no timing details of target.
Regression Models	Lee et al. [34]	Represent cores and contention as functions and explore space using regression modeling.
Interval simulation	Genbrugge et al. [23]	Characterize architectural performance based on intervals between major events.
Adaptive Models	Jones et al. [28]	Use JIT compilation and code-caching to amortize simulation overhead.

(C) Parallelize the model:

SlackSim	Chen et al. [12]	Allows slack synchronization between host threads to tradeoff accuracy and performance.
Graphite	Miller et al. [39]	Scales slack synchronization technique across multiple host machines.
DARSIM	Lis et al. [37]	Multi-threaded simulator with configurable slack synchronization.

(D) Accelerate the model using FPGAs:

Protoflex	Chung et al. [15]	Time-multiplexed functional emulator to accelerate functional modes of SMARTS-based simulation.
UT-FAST	Chiou et al. [13, 14]	Uses FPGA to add timing information to trace generated by modified QEMU that is able to rollback.
RAMP Gold	Tan et al. [55]	Time-multiplexed model with detailed caches but no core pipeline timings or OCN.
LI-BDN PowerPC	Vijayaraghavan et al. [57]	Automatic transformation of implementation to model.
HAsim	Pellauer et al. [46, 47]	General framework for time-multiplexed modeling with emphasis on core detail and OCN, as well as ease-of-use.

Первый заключается в преобразовании контрольных показателей таким образом, чтобы детальная модель выполнялась только на репрезентативной части динамического выполнения. Второй - трансформировать модель, уменьшая детализацию, чтобы облегчить исследование более широкого пространства. Эти подходы полезны для первоначального поиска архитектурного пути, или если изучаемое явление является независимым от циклического поведения ядер. С практической точки зрения, такого рода исследования все еще должны подкрепляться высокодетализированными моделями, если архитекторы собираются убедить своих скептически настроенных менеджеров, особенно в отношении более радикальных предложений. Это приводит нас к третьему подходу: ускорение детализации модели производительности путем распараллеливания самого симулятора. Хотя этот вид распараллеливания может в некоторой степени помочь, растущая популярность многоядерных архитектур фактически расширит разрыв между скоростью симулятора и скоростью цели. Это связано с целым рядом факторов.

Во-первых, моделирование четырех ядер принципиально в четыре раза превосходит работу по моделированию одного ядра, но запуск симулятора на четырехъядерной хост-машине на практике не приводит к четырехкратному ускорению из-за накладных расходов на связь. Во-вторых, многоядерные процессоры следующего поколения обычно

увеличивают число ядер, так что архитекторы могут найти себя моделирующими восемь или шестнадцать целевых машин на четырехъядерном хосте. В-третьих, внутрикристальная сеть (OCN) усложняется по мере увеличения числа ядер, что требует моделирования более сложных топологий и маршрутизаторов. Некоторые изменения необходимы для того, чтобы скорость моделирования с высокой детализацией оставалась достаточно высокой, чтобы архитекторы оставались во главе процесса проектирования. В последнее время несколько компаний начали выпускать продукты, которые позволяют программировать ПЛИС, добавляемые к компьютеру общего назначения по быстрому каналу связи, например PCIe, Hypertransport или Intel Front-Side Bus. Эти продукты привели к появлению интереса в сообществе моделирования процессоров к изучению того, можно ли ускорить модели производительности, как современная видеокарта ускоряет 3D-моделирование. Такие методы были успешно применены в Asim.

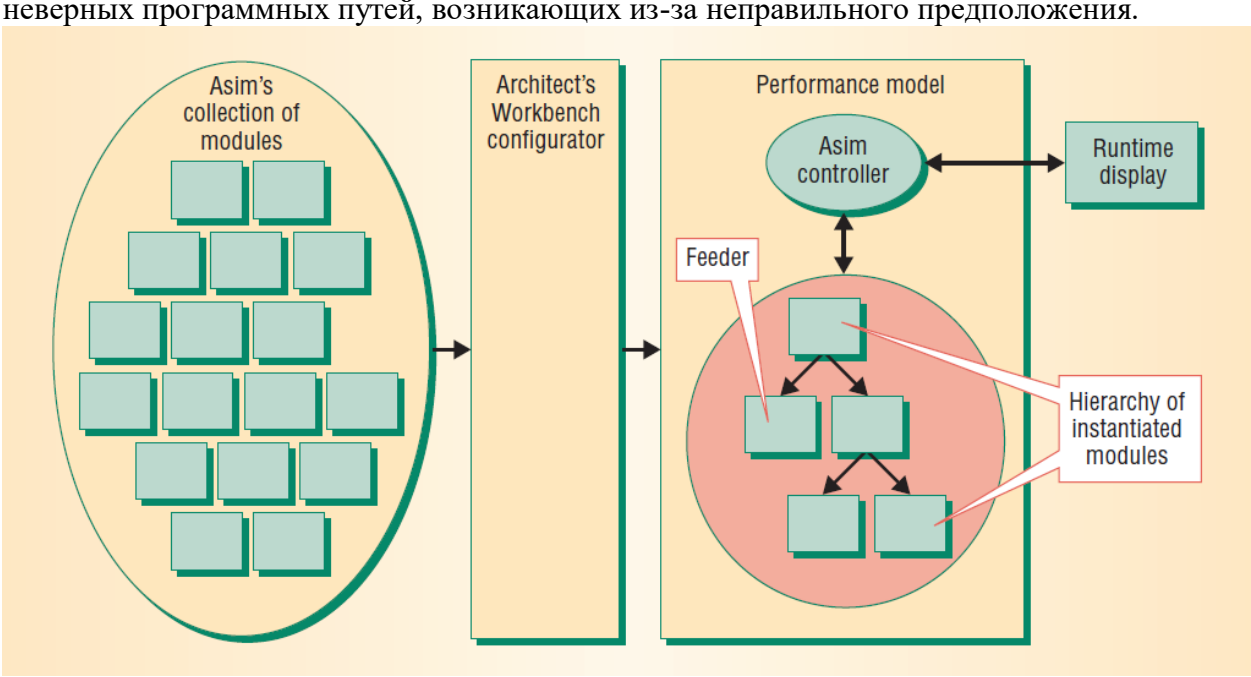
Симулятор ASIM

Микропроцессоры сейчас состоят из нескольких сотен миллионов транзисторов. Взаимодействие между аппаратными структурами, построенными из этих миллионов транзисторов, трудно предсказать с помощью простого моделирования или аналитических моделей. Наличие такого большого количества транзисторов делает также современные микропроцессоры чрезвычайно сложными, существенно расширяя цикл разработки новых микроархитектур. Более того, поскольку на архитектурные аспекты проекта все больше влияют задержки распространения сигнала, систематическое моделирование временных задержек стало более важным. В этом контексте модели производительности играют решающую роль, поскольку разработчики используют их для изучения альтернативных вариантов проектирования и прогнозирования производительности процессоров и систем задолго до их создания. Эти факторы привели к существенному увеличению сложности самих моделей производительности. В прошлом долговечность и полезность модели в основном зависела от навыков и дисциплины авторов моделей. В Compaq данные модели все еще становились чрезвычайно сложными и неуправляемыми, потому что не хватало структурированного способа их разработки. Чтобы справиться с этими сложностями, в конце 1998 года была начата разработка Asim, чтобы можно было работать над созданием

множества новых моделей. Точнее говоря, Asim достигает этих целей за счет модульности и возможности многократного использования. Модульность помогает разбить проблему моделирования производительности на отдельные части, которые можно смоделировать отдельно, в то время как возможность повторного использования позволяет многократно использовать программный компонент в разных контекстах. Многократное использование увеличивает производительность и уверенность в надежности самого программного компонента. Asim предоставляет набор инструментов, которые могут эффективно управлять этими компонентами, чтобы помочь разработчикам моделей справляться со сложностью **большой базы программного обеспечения.**

Основные компоненты

В Asim базовый программный компонент или модуль обычно представляет физический компонент проекта, такой как кеш, или фиксирует работу аппаратного алгоритма, например, политику замены кеша. Конкретная модель будет представлена в виде выбранной пользователем иерархии модулей. Каждый модуль Asimmodule предоставляет четко определенный интерфейс, который позволяет разработчикам повторно использовать модули в разных контекстах или заменять их другими модулями, реализующими другой алгоритм для той же функции. Asim определяет интерфейсы модулей с точки зрения набора вызовов методов или портов. Интерфейсы вызова методов обеспечивают связь между модулем и субвстроенные в него модули. Порты обеспечивают абстракцию связи, которая явно позволяет модели представлять каналы связи и временные характеристики между модулями. Asim также позволяет использовать модули, представляющие инфраструктуру, не основанную на аппаратном обеспечении. Интерфейсы к этим неаппаратным модулям также используют интерфейс вызова метода. Один из таких неаппаратных модулей - фидер предоставляет входные данные, необходимые для управления моделями производительности. Программисты Asim разрабатывают модели независимо от фидеров. В этом подходе устройства подачи инструкций управляют функциональными аспектами выполнения программы, в то время как модели производительности управляют предсказаниями времени, в том числе синхронизацией неверных программных путей, возникающих из-за неправильного предположения.



Разработчики могут использовать Architect'sWorkbench для настройки модели производительности путем выбора определенного набора модулей, просмотра существующих моделей и тестов, запуска модели производительности с конкретным тестом производительности. Asim обеспечивает отображение цикла выполнения, что позволяет разработчикам визуализировать активность в конвейерах процессора по циклу. Asim легко поддерживает как временные, так и автономные модели производительности. Временные модели - модели полного процессора или системы - предсказывают время, необходимое для выполнения части или всей программы. Напротив, с помощью автономных моделей мы можем эффективно изучать поведение отдельных аппаратных компонентов по отдельности. Например, мы могли бы использовать автономную модель для изучения точности предсказателя переходов. Однако, чтобы понять временное поведение предсказателя переходов, мы должны погрузить тот же модуль в полную модель процессора. Используя эту структуру, были разработаны несколько моделей производительности, включая модели для унипроцессоров, процессоров, векторных процессоров и мультипроцессоров, а также для многопоточных, отказоустойчивых, мощных процессоров и тесно связанных сетевых архитектур. Asim также используется для изучения нескольких подкомпонентов процессора, включая предикторы линий и ветвлений, а также компоненты памяти. Asim - бесценная среда для моделирования, поддержки, расширения и управления моделями архитектуры процессора и системы.

Модули

Для эффективного моделирования производительности процессора и системы требуется среда моделирования производительности, которая справляется со сложностью современных машин и программного обеспечения, которое их моделирует. Среда моделирования производительности должна предоставлять язык для выражения структур оборудования, библиотеки, обеспечивающие функциональность, общую для всех моделей, и набор инструментов для построения этих структур оборудования в моделях. Большинство современных языков могут выражать описания оборудования, используемые моделями производительности. Например, Asim в основном написан на C ++.

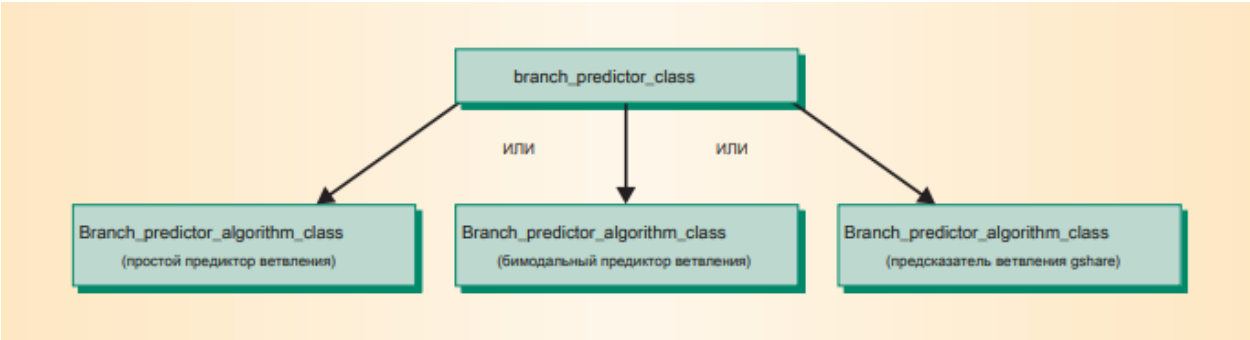
Ключом к достижению этих целей является предоставление пользователю удобных средств для выбора альтернатив и использования четко определенных интерфейсов для

определения того, как взаимодействовать с другими компонентами программного обеспечения. Более ранние модели производительности затрудняли предоставление альтернативных конструкций для отдельных аппаратных компонентов, таких как предикторы ветвлений, или повторное использование кода конкретного компонента в различных моделях. Например, условная компиляция была популярна для предоставления альтернатив модели во время компиляции. Поскольку в условной компиляции отсутствуют четко определенные границы кода, она не ведет к модульному дизайну и, естественно, не определяет четко определенный интерфейс для предоставления или извлечения определенного компонента для повторного использования.

Выражение альтернативных дизайнов

В Asim модуль представляет собой программный компонент, который разработчики могут использовать в различных контекстах, чтобы либо инкапсулировать поведение аппаратной структуры, либо представить программный компонент, который необходимо разделить на модули. Пользователь явно выбирает такие модули при создании модели. Однако внутри мы представляем такие модули как классы C++. Например, на рисунке 2 модуль branch_predictor_class фиксирует высокоуровневое описание универсального предсказателя ветвления. В стиле C++ разработчики наследуют класс предсказателя ветвления от более общего asim_module_class. Использование выбранного класса C++ дает разработчикам гибкость для создания экземпляра модуля по мере необходимости. Таким образом, например, модуль, представляющий ЦП, может быть создан несколько раз для создания мультипроцессора. Asim требует, чтобы модули имели хорошо определенные интерфейсы, чтобы их можно было либо повторно использовать в

различных контекстах, либо заменить другими модулями, которые кодируют другой алгоритм для той же функции. Таким образом, алгоритм gshare может легко заменить простые или бимодальные алгоритмы прогнозирования ветвлений, показанные на рисунке 2.



Чтобы удовлетворить различные способы использования модулей разработчиками, Asim предоставляет два стиля интерфейса: порты и вызов метода. Как правило, субмодули, встроенные в родительские модули, только с внутрицикловой связью, а не аппаратные компоненты используют такой интерфейс. Например, разработчики могут реализовать предсказатель ветвления, кодируя соответствующий тип модуля. Как показано на рисунке 2, модуль branch_predictor_ class может реализовывать предсказатель ветвления путем взаимодействия с модулем класса branch_predictor_algorithm. Любой совместимый предсказатель ветвления в Asim должен соответствовать этому стилю интерфейса. В этом случае интерфейс состоит из трех методов:

• GetPrediction, который получает предсказание ветвления от предсказателя;
• UpdateBranchPredictor, который обновляет предиктор ветвления после разрешения ветвления; а также
• HandleMispredictedBranch, который обрабатывает неверно предсказанные ветви.

Управление выбором дизайна

Управление всеми вариантами дизайна, выраженными программистом, может быть сложной задачей. Процессор или вся система могут иметь сотни модулей, и каждый модуль может иметь несколько вариантов конструкции. Среда моделирования производительности должна обеспечивать основу, позволяющую легко управлять этими вариантами и предоставлять их. Зависимость между модулями еще больше усложняет управление этим выбором дизайна. Например, для класса branch_predictor_class, показанного на рисунке 2, разработчик должен определить связанный алгоритм предсказателя ветвления. Asim управляет этими зависимостями с помощью двух механизмов. Во-первых, у каждого модуля есть тип Asimmodule, который определяет, какие возможности он предоставляет, и неявно определяет его интерфейс. Он также определяет, какие модули и типы требуются. Таким образом, для предсказателя ветвления, показанного на рисунке 3, требуется BranchPredictor_Algorithm.

```
/*****************************************************************************
* Awb definitions
*
* %AWB_START
*
* %name FiveStagePipeline: BranchPredictor
* %desc Branch Predictor for FiveStagePipeline
* %attributes FiveStagePipeline
* %provides BranchPredictor
* %requires BranchPredictor_Algorithm
* %private branchpredictor.h
* %param BP_PROBE_LATENCY 1 "branch prediction latency in cycles"
*
* %AWB_END
*****************************************************************************/
```

Каждый из простых, бимодальных и gshare предикторов, в свою очередь, предоставляет BranchPredictor_Algorithm. Таким образом, интерфейс требует-предоставляет возможность правильно выбирать зависимые модули Asim. Во-вторых, Asim хранит информацию о каждом модуле в отдельном файле .awb. Если разработчики определяют предиктор ветвления gshare в двух файлах - gshare_branch_predictor.h и gshare_branch_ predictor.cpp - эта информация, тип Asimmodule и вся необходимая информация о зависимости обычно находится в ghsare_branch_predictor.awb. Платформа Asim обрабатывает эти файлы .awb и выявляет зависимости между модулями.

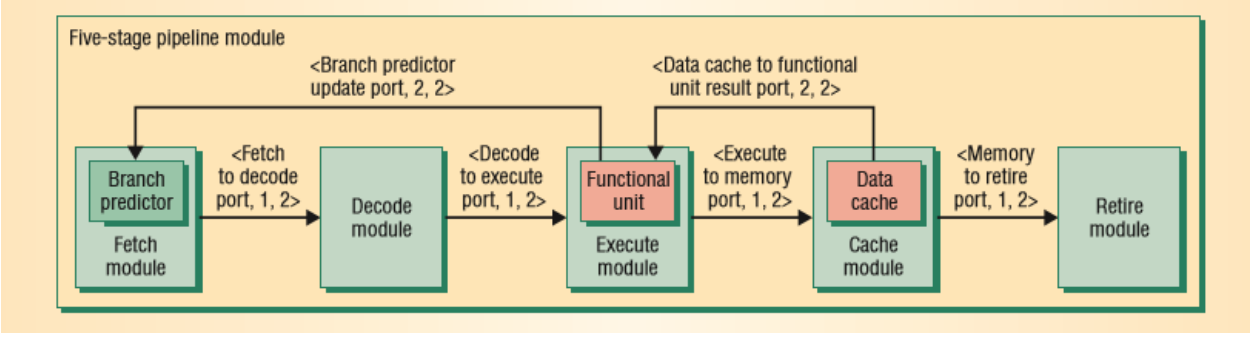
Преимущества

Модули выражают альтернативные варианты дизайна аппаратной структуры, идентифицируя модуль как член класса модуля Asim, который характеризует интерфейс к этому компоненту. Код, реализующий эту модель, изолирован в отдельные файлы. Разработчики могут повторно использовать модули в полноценном конвейере процессора или в простой автономной модели, которая считывает поток инструкций и отправляет запросы модулям. Этот гибкий обмен кодом между простыми и сложными моделями экономит время на кодирование и позволяет разработчикам изучать взаимодействия между модулями. Кроме того, наличие нескольких модулей, реализующих один компонент процессора, упрощает эксперименты с альтернативными вариантами дизайна. Было проведено несколько успешных запусков полных факторных экспериментов с модулями. Эта возможность сочетания и сопоставления также делает сложность тестирования модели производительности очевидной. Обычно модуль тестируется только

в небольшом количестве контекстов, а не во всех контекстах, в которых он может появиться. В случае Asim более частое повторное использование кода модуля в различных контекстах и, как следствие, зрелость кода модуля дают большую уверенность в точности отдельных модулей. Чем более детализированной становится модель, тем медленнее она работает. Однако исследования моделей производительности часто не требуют детальной точности определенных аппаратных структур. В таких случаях разработчики могут заменить модуль более простым и менее подробным модулем, чтобы ускорить выполнение модели производительности

Порты

Текущие тенденции в масштабировании полупроводников показывают, что проволочная задержка будет играть решающую роль в производительности процессора. По этой причине, а также для того, чтобы интерфейсы между модулями были чище и лучше определены, была создана парадигма связи портов Asim. В Asim порты выполняют несколько функций. Во-первых, модули, представляющие аппаратные структуры, обмениваются информацией через порты. Реальный процессор или система обычно не имеют никакой глобальной информации, мгновенно доступной в любом месте микросхемы. Вместо этого состояние процессора распределяется между несколькими аппаратными компонентами, для связи которых может потребоваться несколько циклов. Компоненты, которым требуется доступ к состоянию других компонентов, должны делать это посредством явной связи. Порты Asim представляют собой каналы связи, которые позволяют модулям, представляющим компоненты оборудования, передавать друг другу такую информацию о состоянии. Чтобы передать информацию от одного модуля другому, отправляющий модуль должен записать соответствующую информацию в порт. Приемный модуль на другой стороне соединения порта считывает информацию из порта. Например, в конвейере микропроцессора, показанном на рисунке 4, модуль, который выполняет инструкции, может отправлять правильный результат ветвления в предсказатель ветвления через порт, который соединяет модули выполнения и предсказателя ветвления.



Порты Asimus моделируют задержки связи между модулями. Asimmodels представляет синхронизированную систему, которая синхронизирует каждый модуль один раз за цикл. С точки зрения реального кода, каждый модуль имеет Часовой метод, который планировщик вызывает в определенное время, в течение которого модуль должен быть

активен. В методе Clock модуль определяет все логические действия, которые ему необходимо выполнять в этом цикле, такие как отправка информации и получение ее от других модулей через порты. Как и у реального оборудования, порты имеют фиксированную задержку и максимальную пропускную способность. Информация, которую модуль отправляет через порт, не появляется в принимающем модуле до того, как истечет фиксированная задержка порта. Точно так же модуль не может отправить больше информации, чем позволяет пропускная способность порта. Таким образом, Asim может точно моделировать задержки проводов и пропускную способность между модулями. Asim также использует порты для моделирования задержек в структуре оборудования. На рисунке 4 задержка порта от кеша данных до функциональных блоков включает задержку в один цикл для учета времени, необходимого для доступа к кешу данных. Asim допускает такие представления времени, потому что он создает четкое разделение между аппаратным алгоритмом, независимым от времени, и самим компонентом синхронизации. Порты определяют интерфейсы между модулями, чтобы облегчить кодирование повторно используемых модулей. Хорошо определенный интерфейс запрещает неявные побочные эффекты, которые могут затруднять точность, возможность повторного использования и переносимость кода модели. Традиционно модели производительности используют вызовы методов и глобальные переменные для передачи информации между аппаратными структурами. К сожалению, эти методы допускают неограниченный поток информации, который допускает неожиданные побочные эффекты, что затрудняет повторное использование. Наличие четко определенного интерфейса порта также вынуждает разработчиков явно указывать задержки моделирования между модулями. Эта функция очень важна, потому что программисты часто стараются приблизить время. Чтобы получить предсказания ветвления в Asim, модуль определяет два порта для предсказателя ветвления: один, который запрашивает предсказание, и другой, который его возвращает. У каждого порта есть необходимая задержка. Модуль, запрашивающий прогнозы, даже не увидит их, пока они не станут действительно доступными. Asim реализует порты через очереди «первым пришел - первым вышел». Каждый модуль объявляет один конец порта. Отправляющий модуль объявляет выходной порт со строкой идентификатора, а принимающий модуль объявляет соответствующий входной порт с той же строкой идентификатора. Код утилиты Asim обрабатывает фактическое соединение между портами вывода и ввода. Код утилиты запускается при инициализации и соединяет порты вывода и ввода с помощью строк идентификатора. Этот метод позволяет субмодулям одного модуля подключаться к субмодулям других модулей без ведома родительских модулей о соединении. Автоматическое соединение также позволяет разработчикам автоматически подключать реплицированные модули и соответствующие им порты, что может быть полезно для мультипроцессоров, которые могут создавать экземпляры нескольких процессоров, подключенных к сети через порты.

Фидеры

Для повышения эффективности модели производительности разработчики фокусируются на моделировании только тех машинных операций, которые влияют на

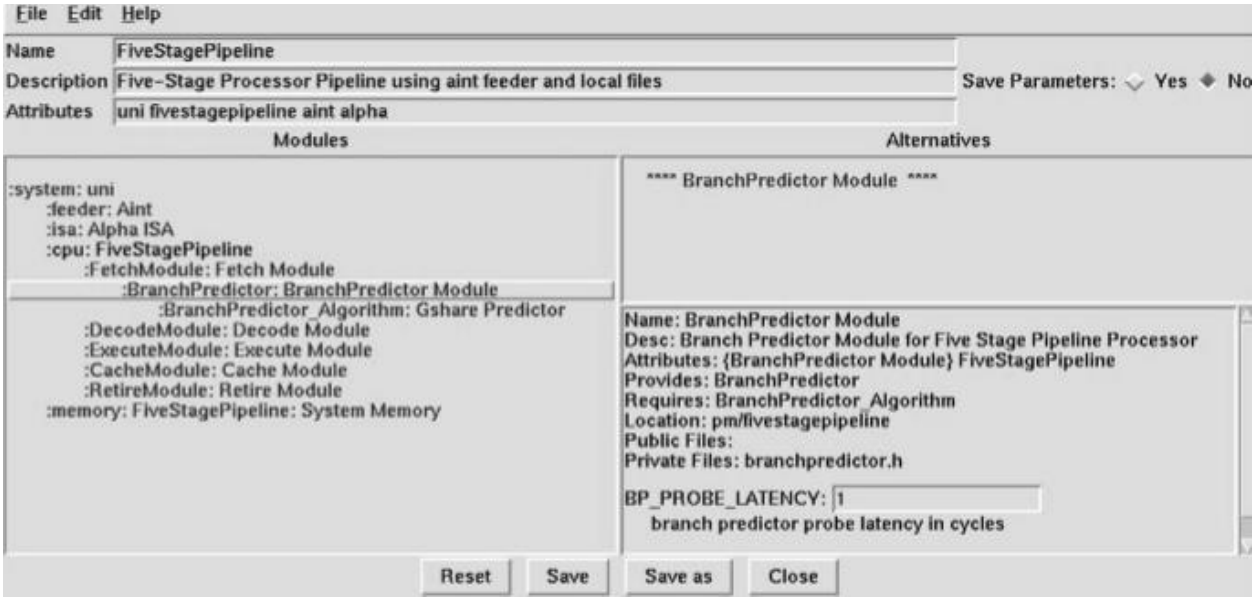
производительность. В частности, разработчики часто могут использовать трассировки инструкций или эмуляции абстрактной архитектуры, чтобы избежать подробностей выполнения архитектуры. Asim поддерживает эту возможность с помощью специально назначенных модулей, называемых фидерами, которые предоставляют входные данные для управления моделью производительности. Поскольку программа или группа программ в конечном итоге управляет реальной компьютерной системой, ключевой источник для модели производительности должен предоставлять инструкции, составляющие программу. Для изучения аппаратных компонентов изолированно с использованием автономных моделей можно использовать несколько фидеров. Например, для управления сетевой моделью нам может потребоваться фидер, который подает сетевые пакеты либо из предыдущей трассировки, либо из аналитической модели. В качестве альтернативы, чтобы изучать предиктор ветвления изолированно, нам нужен только поток инструкций перехода или подмножество инструкций программы. Asim осознает необходимость таких фидеров и для создания наборов сменных фидеров использует абстракцию своего модуля для определения стандартных интерфейсов фидеров.

Фидеры инструкций

В настоящее время Asim поддерживает три механизма подачи инструкций. Статический фидер читает трассировки инструкций, интерпретирует их и передает в модель производительности. Либо реальная машина, либо другая модель производительности может генерировать такую трассировку. Простота статического фидера трассировки, присущая ему повторяемость и соответствующая скорость выполнения делают его привлекательным. Однако, поскольку трассировка имеет только правильно выполненный путь, она не предоставляет инструкции из неправильно выполненных путей. В динамическом фидере трассировки запускается эмулятор инструкций для генерации трассировки на лету и передачи ее в модель производительности. По сравнению со статическим фидером трассировки динамический фидер трассировки экономит дисковое пространство, поскольку Asim не нужно хранить трассы, размер которых может составлять несколько гигабайт. Asim использует динамический механизм трассировки для

чтения инструкций из системной модели SimOS, который имитирует полноценные компьютерные системы, на которых работает множество приложений, включая базы данных. У динамического фидера трассировки также есть проблемы с предоставлением инструкций из неправильно выполненных путей.

Разделение фидера и модели производительности Asim отделяет выполнение архитектурных инструкций в фидере от прогнозов синхронизации в модели производительности, что упрощает модели, избегая полной архитектурной эмуляции. Следующий фрагмент кода (рисунок 5) помогает проиллюстрировать это разделение функций: но в это время фидер не выполняет никаких действий для соответствующей инструкции загрузки.



После устранения промаха и правильного моделирования его времени модель производительности предписывает устройству подачи прочитать значение, возвращаемое промахом, из области памяти устройства подачи. Модель производительности взаимодействует с механизмом подачи через фиксированный интерфейс, который состоит из набора вызова методов, включая вызовы для декодирования, выполнения операций с памятью, уничтожения и фиксации инструкции в устройстве подачи. Модель производительности может иметь несколько этапов конвейера для учета фактической работы длинного конвейера микропроцессора и должна вызывать эти вызовы методов в правильном порядке. В качестве альтернативы, модель должна обнаруживать нарушение порядка сохранения-загрузки, избегать вызова метода фиксации в инструкции загрузки и требовать, чтобы средство подачи уничтожило инструкцию загрузки, чтобы очистить внутреннее состояние средства подачи, соответствующее этим инструкциям. ПК1: R1 ← (R2) ПК2: R3 ← R1 + R5 PC3: R3 → (R2) Этот код показывает последовательность из трех инструкций по адресам PC1, PC2 и PC3. Первая инструкция загружает значение из места в памяти, вторая добавляет его, а третья сохраняет добавленное значение обратно в то же место. Таблица 1 показывает возможный сценарий выполнения в модели подачи и производительности, в которой модель производительности эмулирует конвейер с пятью циклами, который включает функции Fetch, Decode, Execute, Memory и Commit. Напротив, устройство подачи выполняет только фиксированный и меньший набор задач

для правильного выполнения программы. В таблице 1 модель производительности управляет работой подающего устройства.

Table 1. Possible execution scenario in performance model and feeder.

Cycle	Performance model			Feeder		
	PC1	PC2	PC3	PC1	PC2	PC3
1	Fetch			Decode		
2	Decode	Fetch			Decode	
3	Execute	Decode	Fetch	Calculate effective address		Decode
4	Memory cache miss		Decode			
...						
				Load value		
22	Commit	Execute		Commit	Execute	
23			Execute			Calculate effective address
24		Commit	Store		Commit	Store value
25			Commit			Commit

Фидер отслеживает все атрибуты архитектурного состояния, такие как значения регистров и памяти. Как правило, модель производительности не имеет представления об этих значениях. Следовательно, модель получает эффективный адрес инструкции загрузки от фидера и не вычисляет его явно. Устройство подачи не имеет понятия кеш-памяти или иерархии памяти, но вместо этого имеет плоское пространство памяти. Модель производительности поддерживает кеш, но не отслеживает данные кеша. Вместо этого он сохраняет теги кеша только для целей синхронизации, чтобы проверить попадания или промахи кеша. Как показано в таблице 1, модель производительности испытывает и моделирует время промаха кеша. Architect's Workbench - это набор инструментов для использования, управления и отладки моделей производительности и тестов Asim. AWB состоит из двух основных функций: файлов конфигурации и инструментов для работы с этими файлами. Файлы конфигурации определяют архитектурную модель и эталонный тест для конкретного эксперимента. Конфигуратор - это интерактивный инструмент, который манипулирует этими файлами конфигурации и предоставляет пользователю Asim доступ к альтернативным модулям. Конфигуратор использует графический интерфейс

пользователя, но разработчики так же могут использовать все функции графического интерфейса пользователя с помощью неинтерактивных инструментов командной строки.

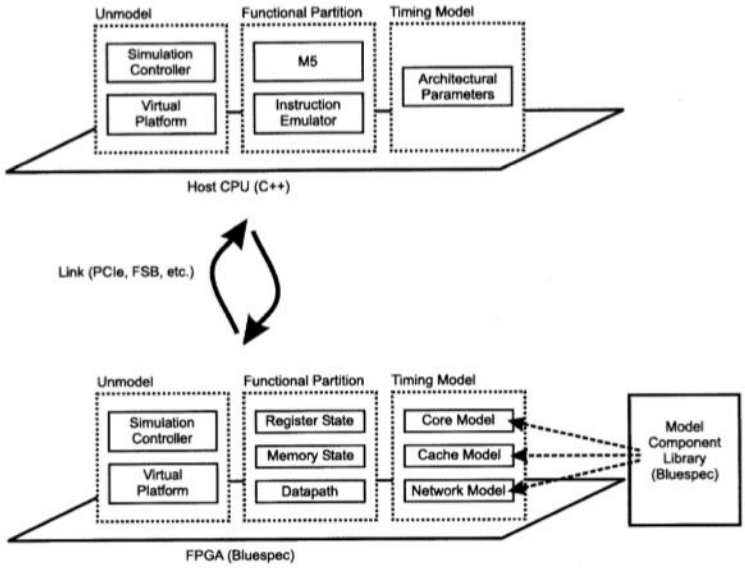
Файлы конфигурации

Чтобы провести эксперимент в Asim, разработчикам нужна как модель производительности, так и программа или тест. Asim фиксирует как модель производительности, так и конфигурации тестов в структурированных текстовых файлах конфигурации. Эти файлы конфигурации позволяют пользователю запускать одну и ту же модель с разными тестами или разные модели с одним и тем же тестом, тем самым обеспечивая больший контроль над экспериментальной настройкой. Файл конфигурации модели производительности Asim предоставляет полную спецификацию. Каждый файл содержит все модули в иерархии "обеспечивает-требует", необходимые для создания полной модели и соответствующих параметров модели. Asim записывает значения

параметров по умолчанию в файлы .awb, поэтому в файл конфигурации нужно записывать только значения параметров, которые отличаются от значений по умолчанию. Файл конфигурации содержит информацию о тесте и о том, как его запустить. Информация о тесте определяет название теста, каталог, источник и т.д. Управляющая часть файла конфигурации направляет AWB выполнение теста, например, чтобы указать интервалы пропуска или выборки. Интерпретатор внутри модели обрабатывает команды.

Обзор HAsim

Учитывая важность проблемы усилий по разработке, цель проекта HAsim состоит в том, что бы создать основу для построения эффективных симуляторов из библиотеки многоразовых компонентов, а не акцентировать внимание на каком-либо конкретном целевом процессоре. Там, где это возможно, HAsim использует методы, разработанные исторически для симулятора Asim. В других местах HAsim разработал новые методы, которые отличают его от современных моделей производительности с ускорением FPGA. На рисунке показан обзор модели, написанной на HAsim.



Подробностей много аспекты этой картины будут объяснены в ходе этого документа. Самое главное, что следует отметить, это то, что HAsim-это гибридная модель, состоящая из кода, работающего на процессоре общего назначения, а также на ПЛИС. В этой схеме мы можем использовать мощь каждой физической платформы: FPGA для мелкозернистого параллелизма и CPU для редких, но труднореализуемых событий, таких как системные вызовы. Ключевым способом сокращения усилий по разработке является сокращение объема кода, который архитектор должен изменить, чтобы построить свое проектное исследовательское пространство. В HAsim – все делится на четыре основных компонента: Функциональный раздел отвечает за корректное выполнение потока команд на уровне ISA; Временная секция (или временная модель) отвечает за отслеживание специфичных для микроархитектуры таймингов, таких как предсказатели ветвей и промахи кэша; Библиотека предопределенных компонентов моделирования, таких как предсказатели ветвей и кэш; "Немодельный компонент относится ко всем функциональным возможностям, не связанным, непосредственно, с моделированием , включая возможность отслеживать статистику и параметры, а также виртуальную платформу, необходимую для взаимодействия с центральным процессором хоста. В большинстве сценариев компьютерный архитектор, использующий HAsim, требуется только для изменения временной модели. Кроме того, библиотека предопределенных компонентов моделирования может еще больше уменьшить это, позволяя архитекторам адаптировать уже существующие модули для своих экспериментов.

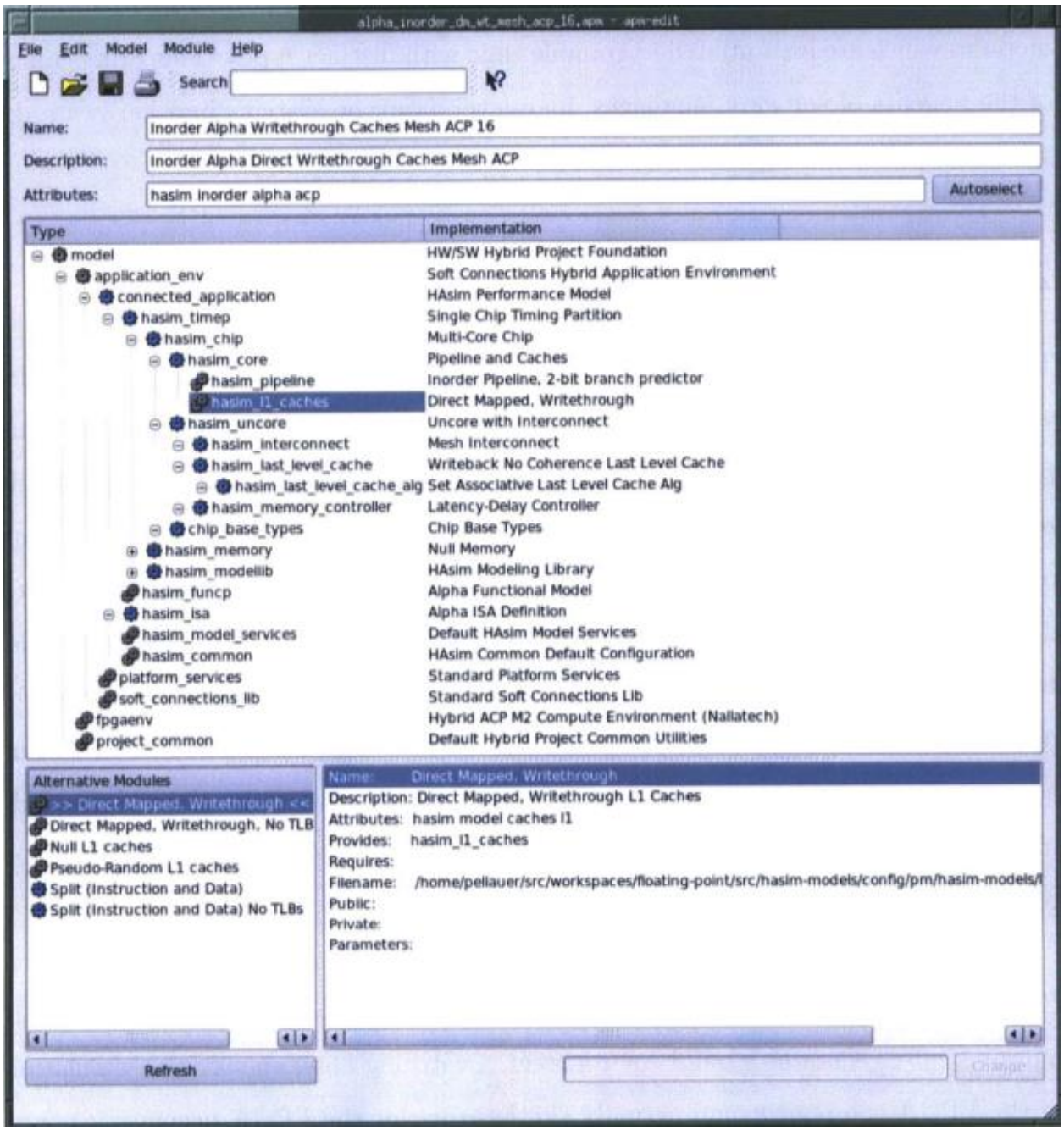
Высокоуровневые Языки Описания Оборудования

Языки описания структурного оборудования, такие как VHDL и Verilog, дают разработчикам точный контроль над своими микроархитектурами. Однако этот контроль часто осуществляется за счет сложного низкоуровневого кода, который не переносится и его трудно поддерживать. При использовании ПЛИС в качестве симулятора архитектор не описывает конечный продукт, и поэтому не нуждается в таком требовательном контроле. Таким образом, языки описания аппаратного обеспечения высокого уровня , такие как Bluespec SystemVerilog , HandelC или SystemC , идеально подходят для поставленных целей. Имитаторы Плис HAsim, UT-FAST и ProtoFlex - это все написано в Bluespec

SystemVerilog. Преимущества Bluespec аналогичны преимуществам использования языков высокого уровня в разработке программного обеспечения: повышение уровня абстракции увеличивает время разработки кода и потенциал повторного использования, одновременно устраняя множество низкоуровневых ошибок, вызванных неправильным взаимодействием блоков. Bluespec также оснащен мощным статическим разработчиком, который позволяет разработчику писать полиморфные аппаратные модули, экземпляры которых создаются во время компиляции с различными типами. Это приносит много преимуществ языкам высокого уровня.

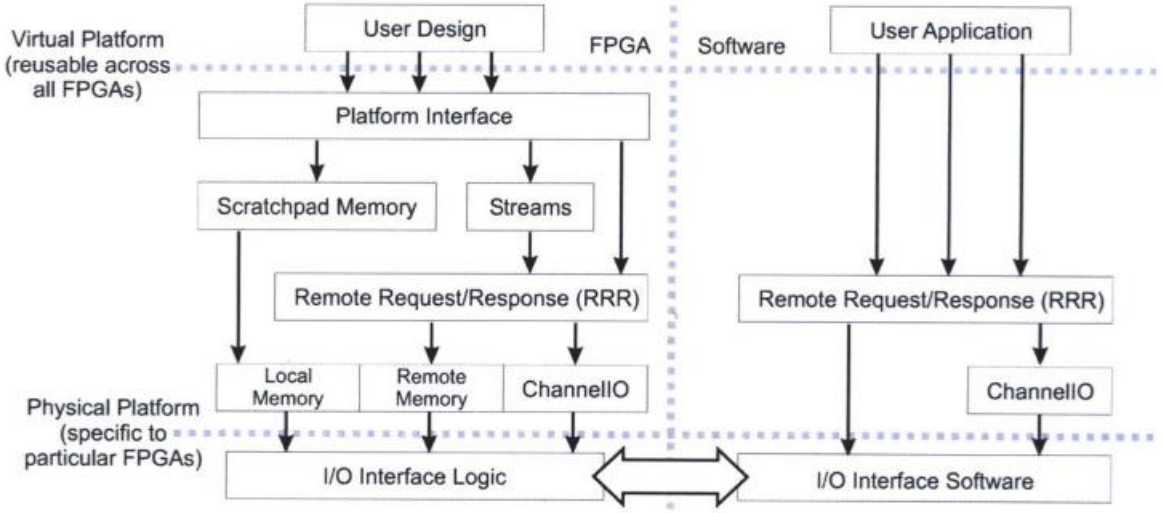
Верстак архитектора и модульность

Верстак архитектора Asim (AWB) - это существующая структура для разработки моделей производительности. Он направлен на улучшение процесса моделирования производительности , особенно на ранней стадии исследования, путем поддержки модульности и повторного использования кода. Эта поддержка обеспечивается на двух уровнях: во-первых, AWB поддерживает представление модели в виде иерархического дерева модулей, где каждый модуль может быть заменен альтернативными реализациями, удовлетворяющими требованиям интерфейса модуля. На самом деле, эти замены позволяют полностью контролировать структуру дерева в течение некоторого времени. Это позволяет спроектировать большое разнообразие различных моделей из общего пула модулей. На втором уровне модульности AWB позволяет получать эти модули из произвольного набора независимо поддерживаемых репозиториев исходного кода. HAsim интегрирован в AWB. Это позволяет компьютерным архитекторам использовать Графический интерфейс для настройки своих модулей. После выбора конфигурации модели AWB автоматически создает среду сборки.


alpha_inorder_dn_wt_mesh_acp_16.apm - apm-edit
File Edit Model Module Help
Search
Name: Inorder Alpha Writethrough Caches Mesh ACP 16
Description: Inorder Alpha Direct Writethrough Caches Mesh ACP
Attributes: hasim inorder alpha acp
Autoselect
Type
Implementation
model
HW/SW Hybrid Project Foundation
application_env
Soft Connections Hybrid Application Environment
connected_application
HAsim Performance Model
hasim_timep
Single Chip Timing Partition
hasim_chip
Multi-Core Chip
hasim_core
Pipeline and Caches
hasim_pipeline
Inorder Pipeline, 2-bit branch predictor
hasim_l1_caches
Direct Mapped, Writethrough
hasim_uncore
Uncore with Interconnect
hasim_interconnect
Mesh Interconnect
hasim_last_level_cache
Writeback No Coherence Last Level Cache
hasim_last_level_cache_alg
Set Associative Last Level Cache Alg
hasim_memory_controller
Latency-Delay Controller
chip_base_types
Chip Base Types
hasim_memory
Null Memory
hasim_modellib
HAsim Modeling Library
hasim_funcp
Alpha Functional Model
hasim_isa
Alpha ISA Definition
hasim_model_services
Default HAsim Model Services
hasim_common
HAsim Common Default Configuration
platform_services
Standard Platform Services
soft_connections_lib
Standard Soft Connections Lib
fpgaenv
Hybrid ACP M2 Compute Environment (Nallatech)
project_common
Default Hybrid Project Common Utilities
Alternative Modules
>> Direct Mapped, Writethrough <<
Direct Mapped, Writethrough, No TLB
Null L1 caches
Pseudo-Random L1 caches
Split (Instruction and Data)
Split (Instruction and Data) No TLBs
Name: Direct Mapped, Writethrough
Description: Direct Mapped, Writethrough L1 Caches
Attributes: hasim model caches l1
Provides: hasim_l1_caches
Requires:
Filename: /home/pellauer/src/workspaces/floating-point/src/hasim-models/config/pm/hasim-models/
Public:
Private:
Parameters:
Refresh
Change

Виртуальная платформа LEAP

Усилия по разработке могут быть дополнительно облегчены путем принятия стндартизированного набора интерфейсов для FPGA. Эта виртуальная платформа предоставляет разработчикам ПЛИС набор виртуализированных абстракций, позволяя им сосредоточиться на расширении основных функциональных возможностей, не тратя время и энергию на отладку низкоуровневых драйверов. Кроме того, большинство симуляторов на базе ПЛИС, скорее всего, будут размещены на гибридных вычислительных платформах, содержащих одну или несколько Плис, и один или несколько процессоров. Расширение хорошо понятных коммуникационных протоколов, таких как удаленный вызов процедур (RPC) и общая память для гибридной среды CPU/FPGA делают форму более доступной для разделения обязанностей между FPGA и CPU. На рисунке ниже показана структура виртуальной платформы HAsim.



Основными интерфейсами между симулятором и платформой являются набор виртуальных устройств и RPC-подобный коммуникационный протокол, называемый Remote Request-Response (RRR), который позволяет нескольким распределенным службам на ЦП и ПЛИС взаимодействовать друг с другом. Основным преимуществом такого подхода является переносимость - виртуальная платформа может быть портирована на новую физическую платформу FPGA без изменения приложения. Только низкоуровневые драйверы устройств должны быть переписаны. Виртуальная платформа HAsim является общей абстракцией для программирования ПЛИС и полностью независима от архитектурного моделирования.

Традиционно разработчики, ускоряющие приложения на ПЛИС, не имеют ничего, кроме необработанных устройств памяти в своих стандартных наборах инструментов. Каждый проект обычно включает в себя утомительную разработку специфичного для приложения управления памятью, которая не используется повторно в разных проектах. Разработчики программного обеспечения ожидают, что среда программирования будет включать автоматическое управление памятью. Виртуальная память обеспечивает иллюзию очень больших массивов, а кэш процессора уменьшает задержку доступа без явных инструкций программиста. LEAP scratchpads - это абстракция, которая динамически выделяет и управляет независимыми массивами памяти в большом резервном хранилище. Доступ к Scratchpad автоматически кэшируется на нескольких уровнях, начиная от общих встроенных, основанных на оперативной памяти, ассоциативных кэшей до частных кэшей, хранящихся в блоках оперативной памяти FPGA. В рамках LEAP scratchpads имеют тот же интерфейс, что и блоки оперативной памяти on-die, и являются заменой подключаемых модулей.

Удаленный запрос-ответ (RRR)

LEAP предоставляет типизированный асинхронный протокол запроса-ответа, называемый RRR (для удаленного ответа на запрос), чтобы обеспечить типизированную связь типа вызова метода между ПЛИС и программным процессом. Подобно удаленным вызовам процедур, пользователь определяет службы, серверы которых находятся на ПЛИС или в программном обеспечении, а клиент - на противоположном конце. Пользователь определяет интерфейс, экспортируемый каждым сервером. Ниже приведен пример такого интерфейса.

```
service ISA EMULATOR
{
    server fpga <- cpu
    {
        method UpdateRegister(in REGINFO rinfo);
    };
    server cpu <- fpga
    {
        method Sync(in REGINFO rinfo);
        method Emulate(in IINFO iinfo, out IADDR newPc);
    };
};
```

Во время компиляции компиляторы заглушек RRR генерируют мультиплексирование кода, который погружает пользовательский код в базовые каналы связи LEAP. Такая система, как RRR, абстрагирует почти всю головную боль между модулем FPGA и программным модулем. Большинство виртуальных сервисов LEAP , описанных ранее, а также скретч-пэдов расположены поверх RRR.

Подводя итоги

HAsim подчеркивает модульность plug-and-play и повторное использование кода, что обеспечивается AWB. Виртуальная платформа LEAP-это общая абстракция, и она нашла применение не только в архитектурном моделировании. Теперь проблема заключается в сборке модели таким образом, чтобы она могла использовать преимущества мелкозернистого параллелизма, присущего ПЛИС. С этой целью мы имеем создал абстракцию под названием A-Ports, которая позволяет проводить распределенное моделирование без необходимости централизованного контроллера.

Литература

[1] **Основы программного моделирования ЭВМ**: Учебное пособие / Речистов Г.С., Юлюгин Е.А., Иванов А.А., Шишпор П.Л., Щелкунов Н.Н., Гаврилов Д.А. — 2-е изд., испр. и доп. — Издательство МФТИ, 2013.

[2] Intel Corporation. **Intel® Itanium® Architecture Software Developer's Manual** Rev. 2.3

[3] ARM Limited. **ARM Architecture Reference Manual** — 2005

[4] IBM Corporation. **PowerPC® Microprocessor Family: The Programming Environments Manual for 64-bit Microprocessors**. Version 3.0

[5] J.C.S. Adrian et al. **Systems, Apparatuses, and Methods for Blending Two Source Operands into a Single Destination Using a Writemask**. US Patent Application Publication. № 2012/0254588 A1

[6] D. Mihoka, S. Shwartsman. **Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure** bochs.sourceforge.net

[7] Yair Lifshitz, Robert Cohn, Inbal Livni, Omer Tabach, Mark Charney, Kim Hazelwood. **Zsim: A Fast Architectural Simulator for ISA Design-Space Exploration** www.cs.virginia.edu/kim/docs/wish11zsim.pdf

[8] **SoftSDV: A Presilicon Software Development Environment for the IA-64 Architecture** / Richard Uhlig, Roman Fishtein, Oren Gershon, Israel Hirsh, Hong Wang // Intel Technology Journal. — 1999. — № 14. — ISSN: 1535-766X. — noggin.intel.com/content /softsdv-a-pre-silicon-software-development-environment-for-the-ia-64-architecture/

[9] Fabrice Bellard. **QEMU, a Fast and Portable Dynamic Translator** // FREENIX Track: 2005 USENIX Annual Technical Conference. — 2005. — www.usenix.org/publications/library/proceedings/usenix05/tech/freenix/full_papers/bellard/bellard.pdf

[10] James E.Smith, Ravi Nair. **Virtual machines — Versatile Platforms for Systems and Processes**. — Elsevier, 2005. — ISBN: 978-1-55860-910-5.

[11] М. Розенблюм и др., «Полное компьютерное моделирование: подход SimOS», Параллельная и распределенная технология IEEE, т. 3, вып. 4, Winter 1995, pp. 34-43.

[12]. А. Пайтханкар, «AINT: инструмент для моделирования мультипроцессоров с общей памятью», кандидатская диссертация, Univ. Колорадо, Боулдер, 1996.

[13]. М. Рейли и Дж. Эдмондсон, «Моделирование производительности микропроцессора Alpha», Компьютер, май 1998, стр. 50-58.

[14]. С.С. Мукерджи и др., «WisconsinWind Tunnel II: быстрый портативный симулятор параллельной архитектуры». Параллелизм IEEE, Октябрь-декабрь. 2000, с. 12-20.