ГЛАВА 23

FPNA — ПРОГРАММИРУЕМЫЙ ПОЛЬЗОВАТЕЛЕМ МАССИВ УЗЛОВ

Введение

Прежде, чем перейти к рассмотрению этой темы, справедливости ради следует заметить, что термин *программируемый пользователем массив узлов* (или *FPNA — field-programmable node array*) введен автором и не относится к стандартной промышленной терминологии. Пока не относится.

Мелко-, средне- и крупномодульные архитектуры

Когда дело доходит до классификации структур различных интегральных микросхем, заказные интегральные микросхемы (ASIC), как правило, относят к *мелкомодульным*, так как инженеры-разработчики могут точно задавать их функциональность вплоть до уровня отдельных логических вентилей. По сравнению с ними, большинство современных ПЛИС могут классифицироваться как *среднемодульные*, поскольку они состоят из небольших блоков («островков») программируемой логики (где каждый логический блок состоит из нескольких логических вентилей и регистров) в «море» программируемых внутренних соединений. Такая классификация справедлива даже в том случае, когда ПЛИС включают в себя микропроцессорные ядра, блоки памяти и встроенные функции, такие как умножители.

По-правде говоря, многие инженеры на самом деле склонны относить ПЛИС к *крупномодульным* устройствам, но при рассмотрении устройства FPNA гораздо больше смысла имеет классификация ПЛИС как *среднемодульных,* поскольку FPNA построены по настоящей крупномодульной архитектуре. Основополагающая концепция этих устройств состоит в том, что они сформированы из массива *узлов*, каждый из которых является сложным элементом обработки данных (**Рис. 23.1**).

Разумеется, на **Рис. 23.1** показано довольно упрощенное представление FPNA-устройства, не только потому что на нём не приведены элементы ввода/вывода и показано небольшое количество узлов обработки, хотя потенциально такое устройство может содержать сотни или тысячи подобных узлов. В зависимости от поставщика FPNA каждый узел может представлять собой арифметико-логическое устройство АЛУ, целый микропроцессор или элемент алгоритмической обработки (последний из них более подробно будет описан в этой главе). Во время написания этой книги от 30 до 50 компаний проводили серьёзные эксперименты с различными типами устройств FPNA; перечень наиболее интересных из них представлен в **Табл. 23.1.**

1960 г. *NASA и Bell Labs запустили первый коммерческий спутник связи.*

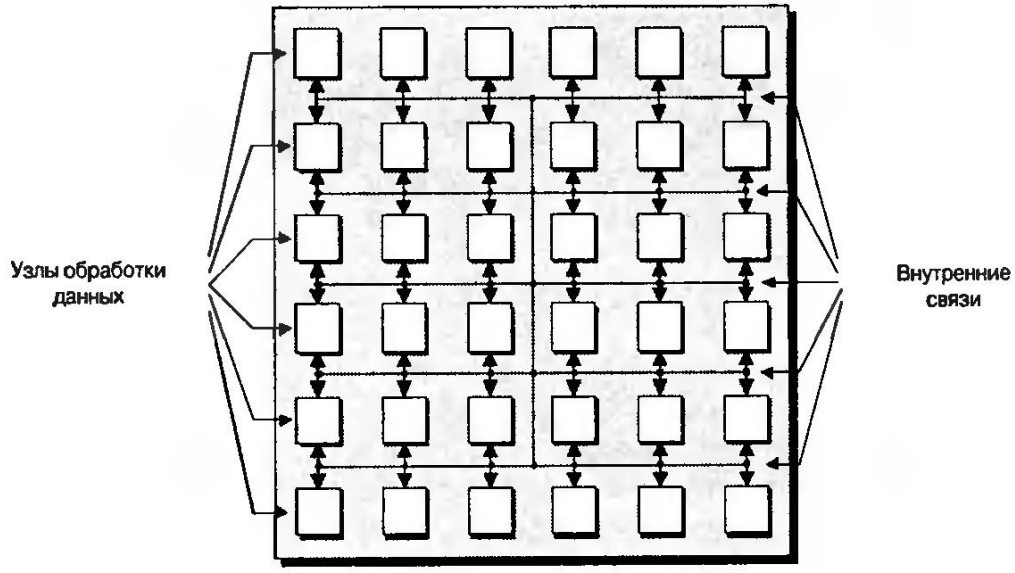


Рис. 23.1. Общее представление устройства FPNA

Таблица 23.1. **Основные разработчики FPNA**

Компания	Web-сайт	Комментарий
Elixent Ltd	www.elixent.com	Узлы на основе АЛУ
IPflex Inc.	www.ipflex.com	Операционные узлы
Motorola	www.motorola.com	Узлы на основе микропроцессоров
PACT XPP Technologies AG	www.pactxpp.com	Узлы на основе АЛУ
picoChip Design Ltd.	www.picochip.com	Узлы на основе микропроцессоров
QuickSilver Technology Inc.	www.qstech.com	Узлы алгоритмических элементов

При рассмотрении материала этой главы мы сосредоточим своё внимание только на двух компаниях — picoChip и QuickSilver — чьи концепции диаметрально противоположны. Устройства picoArray компании picoChip формируются из массивов процессоров и, в основном, предназначены для больших, не критичных к величине энергопотребления, стационарных систем, к которым относятся, например, базовые станции для беспроводных сетей. Кроме того, эти устройства при необходимости в любой момент могут быть реконфигурированы (например, ежечасно или подобно тому, как сменяются профили в сотовом телефоне в течение дня).

В отличие от них *адаптивные вычислительные машины* компании QuickSilver сформированы из кластеров узлов алгоритмических элементов. Применяются эти устройства в основном в небольших, неэнергоёмких портативных изделиях, например в фотоаппаратах и сотовых телефонах (хотя они также представляют интерес и для многих других приложений). Кроме того, эти устройства могут быть реконфигурированы (компания QuickSilver предпочитает использовать термин «*адаптированы*») сотни тысяч раз в секунду.

1961 г. *Разработан метод вычислений с разделением времени.*

Алгоритмическая оценка

Устройства FPNA главным образом предназначены для реализации сложных алгоритмов, требующих значительных вычислительных ресурсов. Поэтому, прежде чем двинуться дальше, давайте уделим немного времени технологиям, использующим эти алгоритмы, чтобы подготовить почву для восприятия последующего материала.

С одной стороны, существуют технологии, использующие алгоритмы пословной обработки, к числу которых, например, относится *метод множественного доступа с разделением по времени* (или *TDMA — time division multiple access*), применяемый в системах беспроводной цифровой передачи данных. Варианты этой технологии, такие как Sirius, XM Radio, EDGE и другие, образуют подгруппу, использующую TDMA. Неудивительно, что архитектура, которая способна поддержать высокотехнологичные стандарты, использующие метод множественного доступа с разделением по времени (TDMA), также справится и с менее сложными своими собратьями (**Рис. 23.2**).

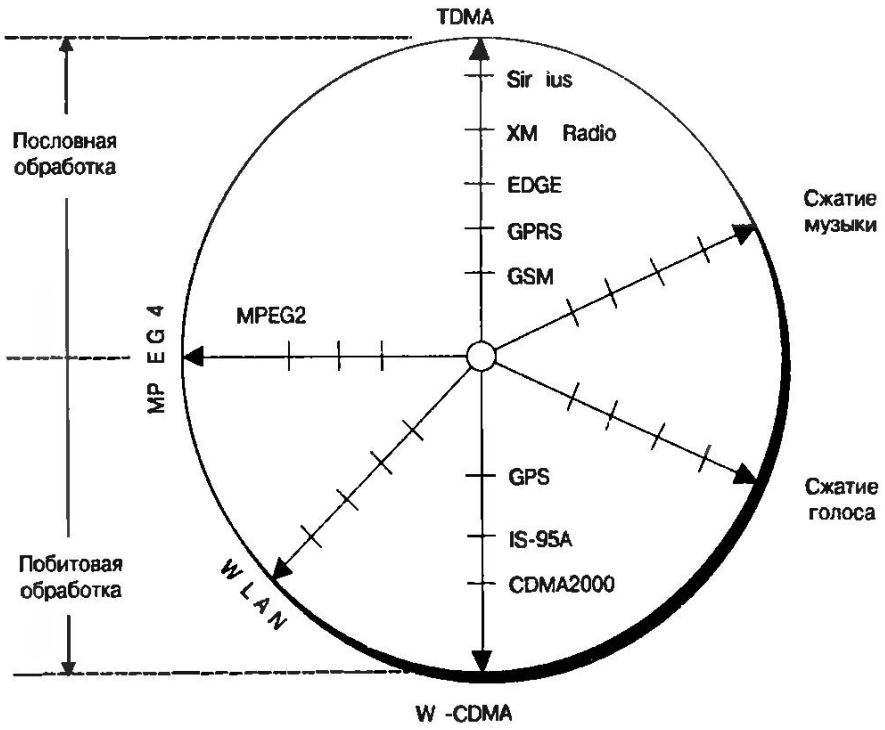


Рис. 23.2. Общее представление устройства FPNA

С другой стороны, существуют технологии, использующие алгоритмы побитовой обработки, такие как *метод широкополосного множественного доступа с кодовым разделением каналов* (или *W-CDMA — Wideband Code Division Multiple Access*) и его подвиды CDMA2000, IS-95A и другие. (W-CDMA используется в широкополосной цифровой радиосвязи, применяемой для передачи данных сети Интернет, мультимедийных, видео и других приложений с изменяемой пропускной способностью канала связи.)

Существуют также технологии, использующие алгоритмы, представляющие смесь пословной и побитовой обработки данных, например, различные виды MPEG, сжатие речи, музыки и так далее.

При оценке этих технологий быстро осознаешь, что традиционный подход к системам с перестраиваемой архитектурой не в состоянии решить проблему на должном уровне (концепция систем с перестраиваемой архитектурой была изложена в гл. 22). Например, некоторые из концепций пытаются рассмотреть системы с перестраиваемой архи-

тектурой и работают на очень низком уровне, а именно на уровне отдельных логических вентилей или блоков ПЛИС. В сочетании с огромнейшей сложностью программирования, этот подход в результате приводит к большим временным затратам на переконфигурирование, что делает его непригодным для некоторых приложений. В отличие от него, другие подходы решают проблему на слишком высоком уровне, например на уровне целых приложений или алгоритмов, что приводит к неэффективному использованию ресурсов.

Неудивительно, если в скором времени выясниться, что эти технологии являются гетерогенными (разнородными) по своей природе, т. е. их разновидности будут существенно отличаться по своему составу. Исходя из этого, очевидно, что для реализации гетерогенных технологий необходимо использовать и гетерогенные структуры. Но что они собой представляют?

1962 г. *Америка. Стив Хофштейн (Steve Hofstein) и Фредерик Хайман (Fredric Heiman) разработали первый полевой транзистор.*

Технология picoArray компании picoChip

Для удовлетворения требований к обработке описанных выше алгоритмов, компания picoChip использует устройство под названием picoArray. Его гетерогенная узловая архитектура представляет собой матрицу различных 16-битных RISC (Reduced Instruction Set Computing — технология вычислений с сокращённым набором команд) микропроцессоров, каждый тип которых оптимизирован для реализации определенных вычислений. Например, один из них может содержать много памяти, а другой будет поддерживать специальные алгоритмические команды типа «расширение» и «сужение» для беспроводного стандарта CDMA, используя для этого один такт (в отличие от 40 тактов при использовании универсального процессора).

В первых версиях этих устройств каждый процессорный узел был приблизительно эквивалентен (по производительности, но не по архитектуре) процессорам ARM9 для функций управления, и TI C54xx для цифровой обработки сигналов (ЦОС). Учитывая тот факт, что каждое устройство picoArray может содержать сотни таких узлов, в результате его применения можно получить огромную вычислительную мощность.

Например, где-то в декабре 2002 года, когда я впервые узнал о технологии picoArray, лидирующее положение в мире цифровой обработки сигналов занимали микросхемы TMS320C6415 компании Texas Instruments. Эти устройства обеспечивали огромный объем вычислений с такой скоростью, что глаза начинали слезиться. Однако picoChip заявила, что один picoArray, работающий на частоте лишь 160 МГц, может достичь вычислительной мощности почти в 20 раз больше (по показателям для 16-битного АЛУ), чем TMS320C6415, работающий при 600 МГц. Вот это да!

Идеальные приложения для picoArray: беспроводные базовые станции

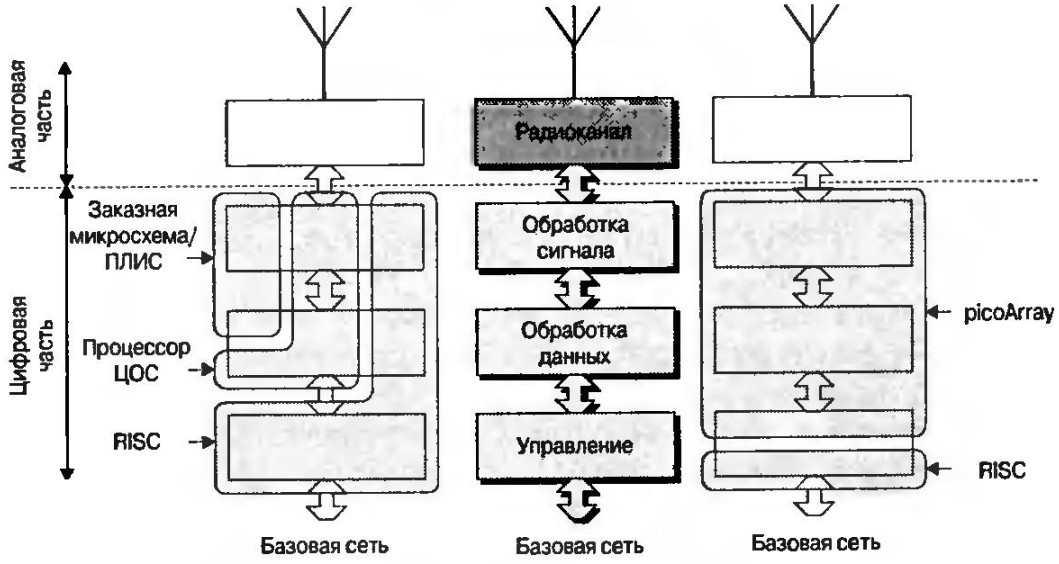
Операторы сотовой связи тратят миллиарды долларов ежегодно на развитие своих беспроводных инфраструктур, причём большая часть этих сумм уходит на разработку новых модулей цифровой обработки, устанавливаемых на базовых станциях. В зависимости от расположения каждая базовая станция должна предусматривать обработку десятков или сотен каналов одновременно.

Поэтому неудивительно, что у всех заинтересованных лиц есть огромное желание уменьшить стоимость реализации каждого канала. Так как один picoArray может заменить несколько традиционных за-

1962 г. *Америка. Компания Unimation представила первый промышленный робот.*

казных микросхем, ПЛИС и процессоров ЦОС (DSP), его использование как раз и позволит существенно снизить цену канала базовой станции.

Одна из проблем, связанных с использованием традиционных решений заключается в том, что для них требуется, по крайней мере, три среды проектирования: для заказных микросхем (ASIC) и (или) ПЛИС, процессоров ЦОС (DSP) и RISC (последняя из них относится к функциональности микропроцессора). При этом все сложности разработки и тестирования, к сожалению, замедляют выход таких базовых станций на рынок (**Рис. 23.3, а**). Вследствие этого значительным преимуществом устройств picoArray является возможность создать законченное изделие в одной среде проектирования. (**Рис. 23.3, б**).



а) Традиционный способ ***б)*** С применением picoArray

Рис. 23.3. Применение традиционных устройств и picoArray

При обычном подходе заказные микросхемы позволяют достичь очень высокой производительности, но они отличаются высокой стоимостью и длительными сроками разработки. К тому же алгоритмы, реализованные в заказных микросхемах, жестко прошиваются в схему. Это, пожалуй, самая главная проблема заказных микросхем, так как стандарты беспроводной связи меняются настолько быстро, что к моменту завершения разработки заказной микросхемы она уже может устареть. Между нами говоря, подобное явление происходит гораздо чаще, чем можно себе представить.

В отличие от заказных микросхем у picoArray каждый микропроцессорный узел полностью программируем, а это значит, что каждый канал может быть легко реконфигурирован для адаптации к ежечасным изменениям пользовательских профилей, к еженедельным расширениям и исправлениям ошибок, и к ежемесячным этапам развития беспроводных протоколов. Следовательно, базовая станция, построенная по технологии picoArray, будет иметь более долгий срок эксплуатации, что позволит снизить затраты на работу системы связи.

Среда проектирования picoArray

Функциональность микропроцессорных узлов устройства picoArray описывается на чистой версии языка C или на Ассемблере. Как уже отмечалось в гл. 11, язык C относится к классу последовательных языков, поэтому нам необходимо каким-либо образом описывать

все параллельные вычисления. В отличие от использования одного из методов расширений языка C/C++, рассмотренного в гл. 11, инженеры компании picoChip пошли другим путём. Они используют язык VHDL для описания структур устройства, в том числе и блоков параллельной обработки, а также для объединения модулей на блочном уровне. После этого с помощью языка C или Ассемблера производится описание внутренней структуры каждого блока.

Интересно заметить, что компания picoChip поставляет полную библиотеку программируемых/реконфигурированных модулей, которые могут быть соединены вместе для реализации полнофункциональной базовой станции (пользователи также могут подстроить отдельные модули для реализации своих собственных алгоритмов). Примерно в мае 2003 года компания picoChip заявила о своём лидерстве, реализовав с помощью этой библиотеки базовую станцию стандарта 3GPP и произведя с неё первый звонок! С тех пор picoChip продолжает активно развиваться, поэтому чтобы быть в курсе текущего положения дел, лучше посетить её сайт по адресу www.picochip.com.

1962 г. *Начал работать первый коммерческий спутник связи Telstar.*

Технология адаптивных вычислительных машин компании QuickSilver

В течение нескольких последних лет сотрудники компании QuickSilver в *режиме повышенной секретности* работали над своей версией устройств FPNA. Представляю себе, как они начнут кряхтеть, услышав это название. Исходя из того, что я знаю (а это больше, чем они думают что я знаю ..., по крайней мере, мне так кажется), можно утверждать, что технология компании QuickSilver, которую они называют *адаптивной вычислительной машиной (АВМ)*, может похвастаться в полном смысле этого слова революционной гетерогенной узловой архитектурой и структурой внутренних соединений (**Рис. 23.4**).

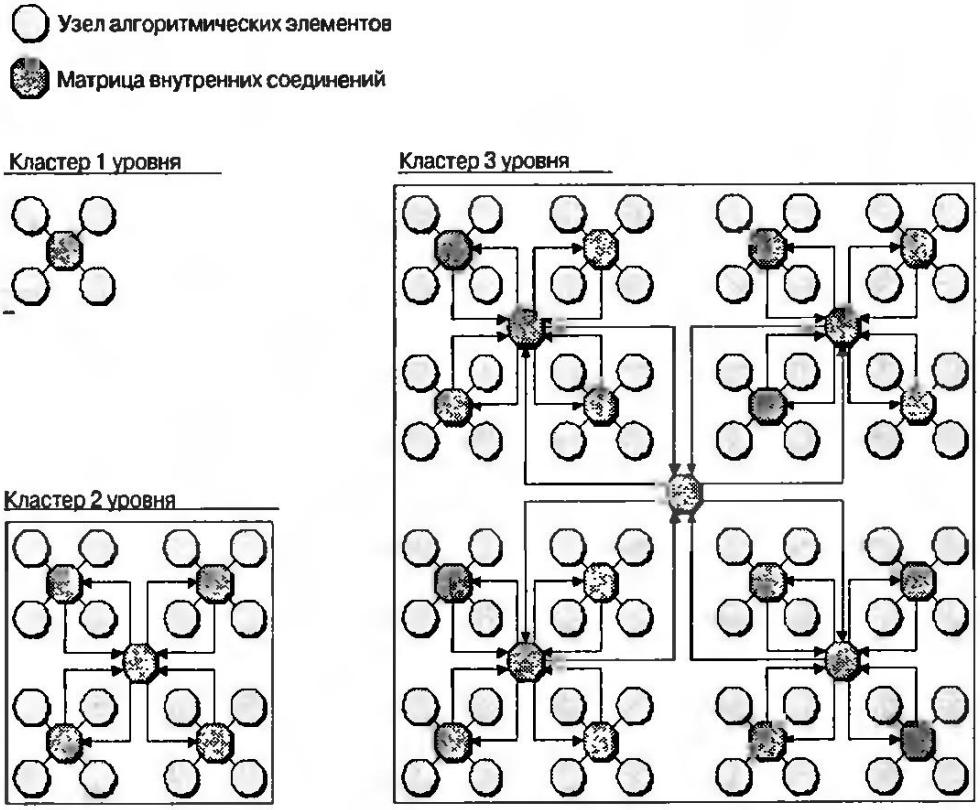


Рис. 23.4. Архитектура адаптивной вычислительной машины

1962 г. *Начала работу первая коммерческая телефонная система с тоновым набором номера.*

На самом нижнем уровне находится *узел алгоритмических элементов*. Четыре таких узла, составляющих квадрант, соединённый *матрицей внутренних соединений*, формируют структуру, которую можно назвать кластером первого уровня. Четыре таких кластера первого уровня могут быть сгруппированы в кластер второго уровня и так далее.

Во время написания этой книги существовало много разных типов алгоритмических узлов (несколько позже мы поговорим о том, какие из них формируют квадрант). Я не буду рассматривать во всех подробностях все узлы, но важно понимать, что каждый их них выполняет задачи на уровне целых алгоритмических элементов. Например, арифметический узел может применяться для реализации различных линейных арифметических функций, например фильтров с конечной импульсной характеристикой (КИХ-фильтров), дискретного косинусного преобразования (ДКП), быстрого преобразования Фурье (БПФ) и т. д. Также такой узел может применяться для реализации нелинейных арифметических функций, таких как (1/sin A)·(1/x) и полученное произведение возвести в 13 степень.

Аналогично *узел обработки битов* может применяться для реализации различных функций побитной обработки, например *линейного сдвигового регистра с обратной связью*, *кодового генератора Уолша*, *генератора кода общей диагностики*, дешифратора TCP/IP-пакетов и т. д.

Каждый узел окружен *оболочкой*, которая позволяет со стороны внешнего мира рассматривать все узлы совершенно одинаковыми. Эта оболочка принимает входящие пакеты информации (команды, необработанные данные, конфигурационные данные и т. д.) из внешнего мира, распаковывает их, распределяет по узлу, управляет задачами обработки, собирает результаты и возвращает их во внешний мир.

Особый интерес представляет концепция оболочки, позволяющая унифицировать внешние интерфейсы узлов и отделить их от внешнего мира, особенно тогда, когда мы понимаем, что каждый узел представляет собой «множество Тьюринга». Это означает, что перед любым узлом, например перед арифметическим узлом побитной обработки, можно поставить любую задачу, и этот узел решит её, хотя и менее эффективно, чем это мог бы сделать узел специализированного типа. Кроме того, компания QuickSilver также позволяет создавать свои собственные типы узлов, в которых вы определяете их начинку и окружаете её оболочкой QuickSilver.

Какие страсти, ужас! Попытка выбрать наилучший путь решения задачи вызвала у меня головную боль. Один из ключевых моментов технологии АВМ заключается в том, что любая часть устройства, от нескольких узлов до целой микросхемы, может быть быстро адаптирована для решения определённой задачи в большинстве случаев за один такт. Также любопытно, что примерно 75% каждого узла реализуется в локальной памяти. Эти особенности позволяют вносить радикальные изменения в методы реализации алгоритмов. В отличие от обычного способа передачи данных от функции к функции при таком подходе данные могут оставаться в узле, в то время как функция узла может меняться с каждым тактом. Это также значит, что в отличие от заказных микросхем, в которых для каждого алгоритма требуется отдельный кристалл, возможность адаптировать АВМ десятки или сотни тысяч раз в секунду подразумевает, что только те части алгоритма, которые действительно выполняются, должны оставаться в устройстве в текущий момент времени. Этот подход позволяет существенно снизить величину потребляемой мощности и занимаемое на кристалле место.

Конфигурирование состава узлов

1963 г. *Америка. В Массачусетском технологическом институте разработан компьютер LINC.*

Я точно не знаю, когда наступит время этой темы, поэтому рассмотрим её прямо сейчас. Итак, мы уже знаем, что существуют разные типы алгоритмических узлов, и что каждый кластер состоит из квадранта этих узлов, соединённых вместе матрицей внутренних соединений. Исходя из этого, я уверен, читателю интересно будет узнать, каким образом разные типы узлов распределяются по кластерам.

Все дело в том, что компания QuickSilver сама не производит и не продает микросхемы, не считая, конечно, опытно-экспериментальных образцов и средств тестирования. Она предоставляет лицензии на использование своих технологий адаптивных вычислительных машин (АВМ) всем желающим, тем самым, позволяя конечному пользователю самому определить оптимальный состав узлов для требуемого конкретного приложения и получить готовые микросхемы, изготовленные в соответствии с его спецификацией. Поскольку все узлы оснащены оболочками, которые позволяют рассматривать их как идентичные блоки, у пользователя появляется возможность легко производить замену одного типа узла другим!

Разновидности узлов

В дополнение к структуре, изображенной на **Рис. 23.4**, любая адаптивная вычислительная машина включает в себя группу узлов специального назначения, таких как системный контроллер, контроллер внешней и контроллер внутренней памяти, а также узлы ввода/вывода. Каждый узел ввода/вывода может использоваться для реализации задач ввода/вывода данных в форме универсального асинхронного приёмопередатчика (УАПП) или в виде шинного интерфейса, например, PCI, USB, Firewire и им подобных (как и алгоритмические узлы, узлы ввода/вывода при необходимости могут быть реконфигурированы в течение одного такта). Кроме того, эти узлы используются для импорта конфигурационных данных, так как при необходимости у каждой адаптивной машины ширина шины конфигурации может быть равна количеству входных контактов.

Теперь рассмотрим кратко процесс разработки и выполнения приложений на АВМ. В данном случае необходимо отметить, что почти все трудности, возникающие при использовании других технологий, ложатся на плечи разработчика АВМ. Например, каждая АВМ содержит встроенную операционную систему ОС, которая распределена по узлу системного контроллера и оболочкам, связанных с каждым алгоритмическим узлом. Алгоритмические узлы также планируют свои задачи и все межузловые соединения. Это позволяет разгрузить работу узла системного контроллера, основная обязанность которого заключается в отслеживании свободных в текущий момент узлов и распределении между ними новых заданий.

Из **Рис. 23.4** следует, что ядро архитектуры АВМ может легко масштабироваться. При этом желательно, чтобы установленные на печатной плате микросхемы различных АВМ взаимодействовали между собой и с остальной частью схемы на уровне операционных систем. В этом случае их работу можно рассматривать как работу отдельных устройств.

1963 г. *Начал работу первый популярный компьютер серии PDP-8.*

Пространственная и временная сегментация

Наиболее важной особенностью архитектуры ABM является возможность реконфигурировать её сотни тысяч раз в секунду, затрачивая на это минимум энергии. Это достоинство позволяет ABM поддерживать *концепцию пространственной и временной сегментации*.

Во многих случаях различные алгоритмы и даже разные части одного и того же алгоритма могут выполняться в разные периоды времени. Концепция пространственной и временной сегментации относится к процессу реконфигурирования динамических аппаратных средств для быстрого выполнения различных частей алгоритма в разные моменты времени и на разных узлах ABM.

Рассмотрим простой пример. Предположим, что в беспроводном телефоне некоторые операции являются модальными, то есть они должны выполняться в течение некоторого отрезка времени. При этом существует три основных режима работы — **обнаружение**, **ожидание** и **обмен данными**. Режим обнаружения относится к сотовым телефонам и подразумевает обнаружение ближайшей базовой станции. В режиме ожидания телефон следит за базовой станцией и за каналом персонального вызова, отслеживая сигнал, который скажет: «Проснись и слушай, тебе звонят». Режим обмена делится на два варианта: прием и передача. Хотя нам может показаться, что мы говорим и слушаем одновременно, но на самом деле, в конкретный момент времени телефон работает только в одном из этих режимов.

В том случае, когда телефон реализован на основе обычных микросхем, то каждая из рассмотренных выше функций обработки полезного сигнала потребует своего собственного кремниевого кристалла или некоторой области на общем кристалле. Это значит, что даже если функция не востребована, она все равно будет занимать место, что отразится в виде высокой стоимости телефона и высокого энергопотребления, а это в свою очередь приведет к быстрому разряду аккумулятора. Если же телефон будет изготовлен на базе технологии ABM, для этих функций потребуется только один кристалл, который при необходимости можно будет в процессе работы телефона адаптировать для выполнения определённой функции.

Но это лишь начало. Во многих случаях каждая из этих основных функций состоит из набора алгоритмов, которые могут выполняться в разные моменты времени. Например, рассмотрим сильно упрощенное представление беспроводного телефона, принимающего и обрабатывающего сигнал (**Рис. 23.5**).

Входящий сигнал состоит из последовательности сильно сжатых блоков данных, передача каждого такого блока занимает крошечный отрезок времени. Затем эта информация проходит через ряд алгоритмов, каждый из которых выполняет некоторую обработку данных и переводит их на более низкую частоту.

Основная особенность этого процесса заключается в том, что каждый алгоритм выполняется в разные моменты времени. При традиционном использовании заказных микросхем каждая функция занимает свой собственный кристалл или часть кристалла на общем устройстве. Поскольку в отдельные моменты времени выполняется только ограниченное количество функций, то в результате подобный подход приводит к значительным потерям доступных ресурсов (затрачивается время и потребляется лишняя энергия).

Еще раз подчеркну, что решение этих проблем заключается в использовании адаптивных вычислительных машин, которые при необ-

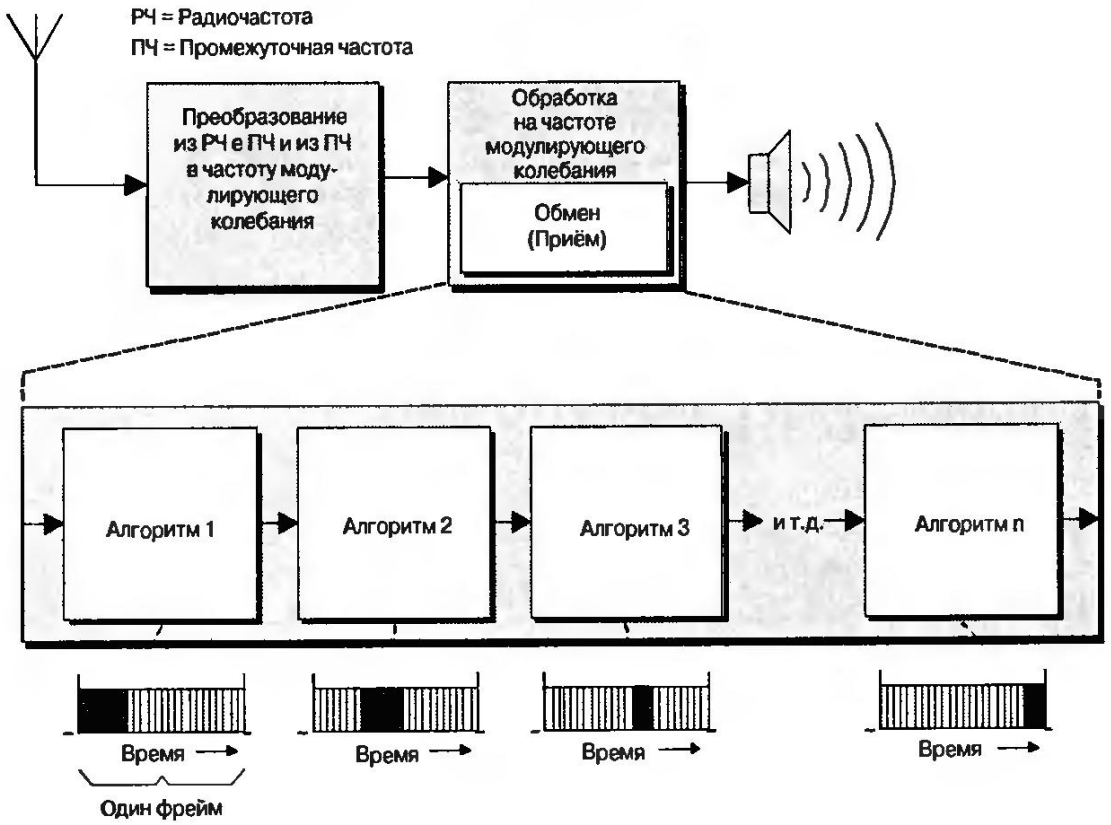


Рис. 23.5. Сильно упрощенное представление сотового телефона, принимающего и обрабатывающего сигнал

1963 г. *Компания Philips представила первую аудиокассету.*

1965 г. *Джон Кемени (John Kemeny) и Томас Курц (Thomas Kurtz) разработали язык программирования Бейсик (BASIC).*

ходимости на лету могут быть адаптированы для выполнения любого алгоритма. Предоставление *аппаратного обеспечения по запросу* приводит в результате к наиболее эффективному использованию аппаратной части в смысле стоимости, размера, производительности и потребления энергии (ABM обещает увеличить производительность в десятки, сотни и более раз по сравнению с аналогичными решениями, потребляя при этом в 2...20 раз меньше энергии).

Создание и выполнение приложений на ABM

Разумеется, правомерен следующий вопрос: как создать приложения, которые могли бы работать на этих маленьких штучках? Ну, во общем-то, средства проектирования компании QuickSilver представляют собой C-подобный язык, названный SilverC. Концепция этого языка аналогична расширениям языка C/C++ (гл. 11).

SilverC сохраняет традиционный синтаксис и структуры управления языка C. Это упрощает C-программистам и проектировщиками систем ЦОС использовать и преобразовывать существующий C-код. SilverC также поддерживает специальные ключевые слова, например module (модуль), pipe (канал) и process (обработка), которые способствуют представлению потока данных и поддерживают параллельное программирование. Более того, SilverC поддерживает специальные расширения для цифровой обработки сигналов, например циркулярный указатель для эффективного использования ресурсов ориентированного ациклического графа, целые числа с фиксированной шириной и фиксированной точкой и так далее.

Представления на языке SilverC могут описываться и моделироваться гораздо быстрее, чем при использовании языка описания аппаратных средств (Verilog или VHDL), которые применяются при традиционном проектировании систем на основе заказных микросхем и

1967 г. *Америка. Компания Fairchild разработала интегральную микросхему под названием Micromosaic, которая стала предшественницей современных заказных и полузаказных микросхем.*

1967 г. *Разработана система шумоподавления Долби (Dolby).*

ПЛИС. После моделирования и проверки SilverC-представления производится его компиляция в исполняемое (двоичное) приложение Silverware. Внутренняя операционная система ABM при необходимости лишь загружает его определённые части, и в текущий момент времени на ABM может работать множество таких приложений.

Важно, что при разработке приложения Silverware нет необходимости знать тип кристалла используемой ABM, включая состав узлов и так далее, или сколько ABM-устройств доступно на плате. Внутренняя операционная система ABM сама распределяет все необходимые задачи по соответствующим узлам.

Подождите, ещё не вечер

В процессе обсуждения средств разработки для устройств цифровой обработки сигналов (гл. 12) мы рассмотрели концепцию проектирования на системном уровне и среду моделирования Simulink компании MathWorks (www.mathworks.com). Это популярное средство поддерживает потоковое проектирование и прекрасно подходит для архитектуры ABM.

Сотрудники QuickSilver усердно поработали над интеграцией языка SilverC в Simulink. На простейшем уровне можно использовать Simulink для описания различных блоков и потоков данных между ними, и автоматически получить структуру, скелет, всего устройства, состоящую из модулей и соединяющих их вместе каналов. Затем можно снова уйти «внутрь скелета» и вручную создавать код на SilverC.

Также QuickSilver разработала библиотеку SilverC-модулей, которые могут загружаться в существующие Simulink-блоки. Эта библиотека включает в себя широко используемые компоненты цифровой обработки сигналов, фильтры, шифраторы, дешифраторы, манипуляторы битов и кодовых групп. Эти модули используются для функционального и потактного моделирования и после компиляции в двоичный Silverware-код могут быть напрямую загружены в ABM.

Это кристалл, Джим, но не такой, каким мы его знаем!

Как вы, вероятно, догадались, возможности FPNA, вообще, и предложения компании QuickSilver в частности произвели на меня неизгладимое впечатление! Итак, означает ли это конец заказным интегральным микросхемам и ПЛИС? Конечно же, нет!

FPNA очень хорошо подходят для разных областей, но нет такой универсальной структуры кристалла, которая хорошо бы подходила для всех случаев, а еще бы и кофе варила. На практике микросхемы FPNA представляют собой лишь одно из средств в арсенале системного разработчика.

С одной стороны, учитывая вышесказанное, я не очень бы удивился, если в недалёком будущем появились бы заказные микросхемы и ПЛИС со встроенными ядрами FPNA. С другой стороны, как уже было сказано, компания QuickSilver позволяет создавать свои собственные типы узлов и окружать их оболочками этой же компании. Поэтому в качестве альтернативы можно использовать главную ABM-структуру, как рекомендует QuickSilver, но включить в неё еще и несколько узлов в виде ПЛИС.

И если однажды это вдруг случиться, ставлю против Ваших, читатель, хлопчатобумажных носков, что я непременно буду в самой гуще событий, яростно жестикулируя и восклицая: «Я же говорил вам!»