ОАО «Мультиклет»

Мультиклеточная архитектура: особенности, реализация и перспективы развития

г.Екатеринбург, 2014

Оглавление

1. Введение

Результат выполнения процессором любой команды выражается в изменении состояния ряда устройств процессора и/или системы (регистров, шин, ячеек памяти, каналов ввода/вывода и т.д.). Использование этого нового состояния последующими исполняемыми командами, образует информационную связь между командой–источником результата и командами–приемниками этого результата. Эта информационная связь может быть как непосредственной (прямой), так и опосредованной (косвенной).

В случае прямой информационной связи команды именуются и при этом имена должны идентифицировать собственно команду, а не ее местоположение или другие особенности реализации. Доступ к результатам может осуществляться по именам как команд–источников, так команд–приемников. В первом случае используется широковещательная рассылка с последующим поименным отбором, при которой в поле операнда команды–приемника задают имя команды–источника. Во втором – выполняется поименная рассылка, при которой в команде–источнике задают имя команды–приемника результата.

При косвенной информационной связи между командами результат доступен только после его отчуждения. А именно, после записи его командой–источником в общедоступные регистры и/или память, имя которых при использовании задают в поле операнда команды–приемника, или в какие-то другие устройства процессора и/или вычислительной системы, доступ к которым имеют только конкретные команды–приемники. Например, при адресной рассылке результат записывается по указанным в команде–источнике адресам памяти, в которых находятся команды–приемники и в указанное поле этих команд.

Используемые виды информационной связи между командами и характер доступа к результатам являются ключевыми факторами, определяющими архитектурную модель процессора. Именно они определяют не только способы кодирования программ, но также способы и устройства их исполнения. Они также

ограничивают количество возможных процессорных моделей четырьмя видами (фон-неймановская, потоковая и две мультиклеточные модели).

2. Процессорные архитектуры с косвенной информационной связью

На базе фон-неймановской модели построены все коммерчески успешные процессоры. В ее основе лежит общедоступность результатов и, как следствие, решение процессором определенной задачи достигается пошаговым целенаправленным изменением состояния вычислительной системы. Этот процесс изменения состояний является результатом исполнения последовательности команд строго в заданном порядке, который определяют в процессе программирования.

Как правило, все варианты реализации фон-неймановской модели направлены на решение одной и той же задачи, а именно, повышение производительности процессора, но решают они ее разными способами.

Например, в конвейерных процессорах совмещают выполнение команд во времени при использовании одного исполнительного устройства. В RISC процессорах – сокращают количество функций реализуемых командой и, следовательно, время выполнения одной команды. В CISC процессорах увеличивают объем функций выполняемых одной командой и, соответственно, время, но при этом уменьшают общее количество выполняемых команд.

За счет совмещения выполнения команд в пространстве, т.е. на нескольких исполнительных устройствах, обеспечивают повышение производительности суперскалярные процессоры и процессоры со сверхдлинным командным словом (VLIW процессоры). При этом в суперскалярных процессорах задача распределения потока команд по нескольким исполнительным устройствам решается преимущественно аппаратным способом, а в VLIW процессорах – программным, во время компиляции.

В конкретных реализациях эти и другие способы могут сочетаться. В частности, RISC и CISC процессоры обычно имеют конвейерную организацию, но

отличаются друг от друга длиной конвейера. Конвейер также широко используется для сокращения времени выполнения потока команд поступающих к исполнительному устройству в суперскалярных процессорах. Наиболее совершенные способы организации конвейера не только реализуют совмещение выполнения потока команд во времени, но и обеспечивают переупорядочивание команд в конвейере для устранения конфликтов по данным между ступенями конвейера. В этом случае команда в конвейере выполняется не по очередности, а по готовности.

Известны две схемы, осуществляющие такое переупорядочивание команд. Это схема централизованного окна команд и распределенная схема Томасуло.

Следует отметить, что ни использование методов распараллеливания в суперскалярных или в VLIW процессорах, ни методов переупорядочивания команд в конвейере, которые меняют очередность исполнения команд – не отменяют ключевого принципа фон-неймановской модели – упорядоченного изменения состояния процессора. Вне зависимости от очередности исполнения команд, формируемые ими состояния процессора меняются в той последовательности, в которой эти команды были выбраны из памяти, а не исполнены.

Несмотря на разнообразие реализаций, все процессорные устройства, использующие фон-неймановскую модель архитектуры, имеют один общий недостаток.

Это необходимость решения сложной задачи распараллеливания для организации параллельной работы как внутри процессора, при наличии нескольких однотипных исполнительных устройств, так и при использовании многоядерных процессоров или многопроцессорной вычислительной системы.

В потоковой модели используется адресный способ связи. Это либо рассылка результатов в потоковых процессорах, либо указание на адрес источника результата в редукционных процессорах. Типичная команда потокового процессора, например, двухместная арифметическая операция, содержит, как правило, код операции, поля операндов, поле адреса команды, которой передается результат выполненной операции, и поле номера операнда, в котором он будет размещен. Команда

выбирается и выполняется «по готовности операндов», т.е. тогда, когда ею будут получены все результаты выполнения других команд, необходимые для ее выполнения. Таким образом, команды в программе могут быть размещены произвольным образом. Принцип исполнения команд потокового процессора «по готовности операндов» позволяет реализовать параллелизм «естественным» образом, т.е. без необходимости решения задачи распараллеливания, но требует для этого использования специальных языков программирования (языков с однократным присваиванием), а также, по сравнению с фон-неймановской моделью, приводит к резкому увеличению затрат на пересылку операндов и выполнению лишних вычислений.

Многочисленные попытки модифицирования архитектуры потоковых процессоров приводили к решению локальных проблем, не изменяя сути данного метода. Например, ввод тегов для обеспечения реентерабельности подпрограмм или для параллельного выполнения различных итераций цикла является виртуальным расширением адреса («адрес» & «тег»), но не сменой парадигмы.

Учитывая большой объем существующего в настоящее время программного обеспечения, написанного на традиционных императивных языках, необходимость использования языков с однократным присваиванием для программирования потоковых процессоров делает их применение экономически нецелесообразным.

Редукционная модель процессора также использует адресную связь между командами, но она задается указанием в поле операнда команды–приемника адреса команды–источника, которую необходимо выполнить для получения требуемого результата. В этом случае команды выбираются из памяти и исполняются «по запросу команды» Такая модель принципиально не имеет лишних вычислений и также позволяет отказаться от упорядоченности команд, которые могут быть размещены в памяти произвольным образом.

Как и потоковая, редукционная модель обеспечивает «естественную» реализацию параллелизма, но тоже требует применения для программирования специальных языков, а именно языков функционального программирования. Это требование, как и аналогичное требование к потоковой модели, не позволяет

использовать существующее программное обеспечение, что делает использование редукционных процессоров экономически невыгодным. Следует также отметить, что выборка команд «по запросу» ограничивает возможности совмещения по времени процессов выборки и исполнения команд и, как следствие, их параллельную реализацию.

Разноплановость преимуществ, присущих разным моделям, привела к созданию процессорных моделей, сочетающих различные способы реализации параллелизма и управления вычислительным процессом. Наиболее известные из них – это макропотоковые, и их развитие – гиперпотоковые процессоры.

В макропотоковых моделях исполнение «по готовности» относится не к одной команде, а к последовательности из нескольких команд, называемых нитью (thread). При этом последовательность команд в пределах нити статична и определена при компиляции, при этом характер ее выполнения фон-неймановский. Как правило, для этого используется конвейерная организация. Порядок обработки нитей зависит от их готовности и меняется динамически в процессе вычислений.

Известны две формы макропотоковой обработки: без блокирования и с блокированием. В модели без блокирования выполнение нити начинается только после получения всех необходимых данных. При этом выполнение нити не прерывается, она выполняется до конца без остановки. В варианте с блокированием выполнение нити может быть начато до получения всех операндов, при этом, когда требуется отсутствующий операнд, выполнение нити приостанавливается (блокируется). Процессор запоминает всю необходимую информацию о состоянии такой заблокированной нити и начинает выполнение другой готовой нити. После получения отсутствующего операнда и завершения выполнения текущей нити, выполнение блокированной нити возобновляется. Модель с блокированием обеспечивает более полную загрузку исполнительных устройств за счет использования механизма блокировки и, как следствие, возможности формирования длинных нитей.

Гиперпотоковые модели эмулируют работу нескольких логических процессоров на одном вычислительном ядре. При этом в процессе компиляции

формируются такие последовательности команд, которые процессор мог бы обрабатывать параллельно, занимая исполнительные устройства, не занятые одним из потоков, подходящими командами из другого независимого потока. Это принципиальное отличие гиперпотоковых моделей от макропотоковых. При этом характер выполнения потока логическим процессором – фон-неймановский, а управление исполнением потоков осуществляется с помощью операционной системы.

Сочетанием применяемых методов отличается и проект процессора TRIPS [1], архитектуру которого авторы относят к EDGE-архитектуре – явного исполнения графа потока данных. Процессор TRIPS имеет два функциональных блока, каждый из которых является фактически VLIW процессором, содержащим 16 (4x4) 64-разрядных арифметико-логических устройств (АЛУ) с плавающей точкой и управляемым потоком данных. От традиционного потокового процессора они отличаются тем, что команды загружаются во все АЛУ одновременно, но процесс их выполнения такой же как в потоковом процессоре – асинхронный, по готовности операндов.

Разные методы используют и в синергической вычислительной системе – синпьютере, который как и процессор TRIPS имеет асинхронную организацию, но в отличие от него состоит из множества идентичных, функционально полных вычислительных устройств, объединенных коммутационной средой, обеспечивающей обмен результатами между ними [2]. Для программирования синпьютера алгоритм рассматривается как совокупность формул, представленных в ярусно-параллельной форме, а именно, в виде ациклического ориентированного графа, вершины которого взаимно-однозначно соответствуют используемым в формуле величинам и операциям, а дуги соответствуют и отражают направление информационных связей между ними. При этом вершины распределены по ярусам (подмножествам) так, что вершины одного яруса непосредственно не связаны друг с другом, а ярусы пронумерованы таким образом, что вершины-источники всегда размещены на вышележащем ярусе (имеющем номер меньше).

Местоположение любого узла ярусно-параллельной формы задается парой чисел (i,k), где: i – номер яруса (строки); k – номер узла в ярусе (столбца).

В общем случае, для выполнения операций, расположенных на i-ом ярусе, могут использоваться результаты, полученные на ярусах от 1-го до (i-1)-го. В синергической вычислительной системе эта возможность ограничена следующим образом:

для выполнения операций на i-ом ярусе используются только результаты (i-1)-го яруса;

если для выполнения операции на i-ом ярусе нужен результат с яруса j, где $1 \leq j < (i-1)$, то на ярусе j+1 этот результат запоминается в регистровой или оперативной памяти, а на ярусе i-1 читается.

Это ограничение позволяет исключить из указания источника результата номер яруса. Источник однозначно задается номером узла в предшествующем ярусе и, таким образом, для кодирования двухместных операций достаточно двух полей операндов, содержащих номера узлов источников, результаты которых используются в качестве аргументов при выполнении данной операции. Например, в таблице 1 приведена программа синпьютера для вычисления формулы g=e*(a+b)+(a-c)/f.

Таблица 1

Номер яруса	**Номер узла**			
	1	2	3	4
1	чт a	чт b	чт c	
2	+ 1,2	чт e	- 1,3	чт f
3	* 1,2	/ 3,4		
4	+ 1,2			
5	зп 1,g			

В синпьютере, последовательность описаний операций одного узла является упорядоченной последовательностью команд, выполняемых вычислительным устройством с соответствующим номером. При этом операции одного яруса в принципе могут размещаться произвольно – в любом узле. На них не накладывается требование упорядоченности. Возможность неупорядоченного размещения команд в одном ярусе упрощает решение задачи распараллеливания, но не позволяет реализовать параллелизм «естественным» путем.

Каждое вычислительное устройство состоит из памяти программ, памяти данных, блока регистров общего назначения и процессора, включающего устройство управления и несколько попарно связанных буферных и исполнительных устройств.

Устройство управления реализует классическую фон-неймановскую схему выбора команд, т.е. их последовательность определяется счетчиком команд. Буферные устройства обеспечивают хранение выбранных команд до получения всех операндов и передачу готовых команд на исполнение. Исполнительные устройства выполняют готовые команды и передают их результат в коммутационную среду.

В процессе выборки команд устройства управления всех вычислительных устройств согласованно выбирают команды очередного яруса и присваивают им значения тегов, равные последовательному номеру выбранного яруса. Декодируют выбранные команды и, при необходимости, формируют заявку коммутационной среде на необходимые им результаты вычислительных устройств, полученные при выполнении команд предыдущего яруса, указывая коммутационной среде номер тега предыдущего яруса и номера вычислительных устройств, после чего записывают выбранную команду, в буферное устройство. Выбор буферного устройства определяется выполняемой операцией. Затем формируют новое значение счетчика команд, и процесс выборки повторяется.

В буферном устройстве команда будет находиться до тех пор, пока она не будет сформирована, а именно, пока коммутационная среда не получит и не передаст в буфер все запрошенные при выборке данной команды результаты.

Процесс формирования команд инициализируется поступлением в коммутатор очередного результата. При этом коммутационная среда по тегу результата проверяет наличие заявок на данный результат от данного вычислительного устройства. Если заявки есть, то результат передается в буферные устройства соответствующих вычислительных устройств, которые записывают его в поля операндов тех команд, которые его запрашивали.

Процесс исполнения команды и рассылки результатов инициализируется, если в буферном устройстве есть хотя бы одна команда, у которой получены все операнды (исполнение «по готовности»), и которая не является последней выбранной командой (кроме команды остановки). При этом, если готовых к исполнению команд несколько, то на исполнение передается команда, которая из них была выбрана первой. Исполнительное устройство выполняет команду и выдает в коммутатор её результат вместе с тегом команды.

Завершается или приостанавливается этот процесс тогда, когда в буферных устройствах нет готовых к исполнению команд.

Использование принципа исполнения команд «по готовности» в синпьютере и в процессоре TRIPS – не позволяет отнести их к потоковым процессорам. Характер размещения команд и их выборки на исполнение – фон-неймановский. Принцип «по готовности» ограничен только этапом исполнения команд. Соответственно, указанный недостаток фон-неймановской архитектуры присущ обоим процессорам.

Следует также отметить, что ряд общих недостатков имеют все процессорные устройства, использующие косвенную информационную связь с любым принципом доступа к результатам и, соответственно, с любой архитектурной моделью.

Это, во-первых, полная машинная зависимость программ. При изменении конфигурации процессора или вычислительной системы, например, при изменении количества используемых исполнительных устройств, как правило, требуется перекомпиляция программ, а иногда и полное перепрограммирование.

Во-вторых, непрерывное увеличение семантического разрыва между постоянно развиваемыми языками программирования высокого уровня и машинным

кодом. В результате повышается сложность компиляторов и, следовательно, затраты на их разработку.

Указанные недостатки, как и недостаток фон-неймановской архитектуры, связанный с необходимостью решения задачи распараллеливания, возникли из-за использования косвенной информационной связи между командами и, следовательно, они не устранимы в процессорных устройствах, использующих эту связь.

3. Организация непосредственной информационной связи между командами

Принципиально иной вид информационной связи между командами – это непосредственная связь. Примером кодирования программы с использованием этой связи между командами является язык триад, используемый в качестве промежуточного представления исходной программы в процессе компиляции [3].

Промежуточное представление в виде триад образуется записями с тремя полями:

<op><arg1><arg2>,

где: op – поле кода операции; arg1 – поле первого операнда; arg2 – поле второго операнда.

В качестве операндов используются это либо указатели на таблицу символов (для имен, определенных программистом, или констант), либо указатели на триады–источники используемых результатов.

Для следующей последовательности операторов, написанной на языке высокого уровня и образующей линейный участок:

```
L1:   a:=b+c;
      i:=a*f–g;
      a:=i /h;
      goto L2;
```

промежуточное представление на языке триад будет выглядеть так, как показано в таблице 2.

Таблица 2.

Номер триады	**op**	**arg1**	**arg2**
0	+	b	c
1	*	(0)	f
2	–	(1)	g
3	/	(2)	h
4	:=	i	(2)
5	:=	a	(3)
6	goto	L2	

Имена – это указатели на таблицу символов, а числа в скобках – это указатели (ссылки) на триады, причем допустима только ссылка на триады данного линейного участка. Информационная связь между различными линейными участками – косвенная – через память.

Использование непосредственных информационных связей между командами и широковещательная рассылка их результатов является принципиальной основой существующей мультиклеточной архитектуры. Система команд мультиклеточного процессора подобна триадам и, фактически, является аппаратной реализацией входного языка программирования.

Как известно, моделью вычислений в императивных языках высокого уровня является выполнение упорядоченной последовательности операторов. Каждый оператор представляет собой неделимую и целостную языковую конструкцию, описывающую процесс преобразования данных. Порядок выполнения операций внутри оператора задается путем их ранжирования и расстановки скобок, т.е. указанием непосредственных информационных связей между операциями. Промежуточные результаты вычислений внутри оператора не отчуждаются, а программисту виден только результат выполнения оператора. Следовательно, для

абстрактной машины, непосредственно реализующий некоторый язык высокого уровня, оператор языка является командой.

Традиционно, под командой понимается то неделимое и целостное действие вычислительной машины, которое видимо и доступно программисту; оно не может быть задано или выполнено частично (неделимость и целостность); результат данного действия фиксируется путем изменения состояния машины, которое далее может использоваться другими командами (видимость); программист может задавать это действие в процессе программирования (доступность).

В машинах с фон-неймановской архитектурой система команд машины, которая используется при программировании, полностью эквивалентна системе команд, которую реализуют исполнительные устройства данной машины. Для виртуальной машины, реализующий некоторый язык высокого уровня на аппаратном уровне, ситуация принципиально отличается.

Так, исходное множество операций любого алгоритмического языка изначально зафиксировано и конечно. Множество операторов, которые теоретически могут быть сконструированы с использованием данных операций, является потенциально бесконечным. Поэтому машина с архитектурой, непосредственно реализующей язык высокого уровня, будет иметь конечный набор команд исполнительных устройств, реализующих множество операций языка, но не будет иметь фиксированной системы команд на уровне машины. Например, команда сложения, реализующая соответствующую операцию, является таковой только внутри машины, но она не является оператором и, соответственно, командой при программировании, где она может быть только составной частью арифметического выражения, используемого в условных операторах или операторах присваивания.

Устранение этого семантического разрыва между уровнем программирования (оператор) и уровнем исполнения (последовательность команд) для фон-неймановских машин традиционно решается в процессе компиляции. Однако решение этой задачи аппаратными средствами в процессе исполнения более эффективно, поскольку появляется степень свободы в формировании потока

команд, поступающего на исполнение. Это позволяет сформировать поток оптимальным образом, исходя из текущего состояния машины.

Программа мультиклеточного процессора представляет собой неупорядоченную совокупность параграфов, каждый из которых является линейным участком. Информационная связь между командами внутри параграфа – прямая, с широковещательной рассылкой и последующим поименным отбором результатов, а между параграфами связь косвенная, через память. Команды параграфа образуют частично упорядоченную и информационно замкнутую последовательность, а именно, команды–источники предшествуют своим командам-приемникам, а команды–приемники имеют ссылки на команды-источники только своего параграфа.

Состояние мультиклеточного процессора, видимое программисту, формируется по завершению параграфа. В этом смысле, параграф эквивалентен команде в фон-неймановском процессоре или оператору (линейной последовательности операторов) в виртуальной машине, непосредственно реализующей язык высокого уровня. Параграфы в программе не упорядочены. Переход от одного параграфа к другому выполняется только командами передачи управления. Поскольку состояние процессора меняется по завершению параграфа, то команда, формирующая адрес следующего параграфа, может занимать любое место в последовательности команд, образующих параграф.

Ссылки кодируются числом, значение которого равно разнице между номерами команды-приемника и команды-источника, полученными при последовательной нумерации команд параграфа. Так, например, фрагмент программы, приведенный в таблице 2, будет выглядеть следующим образом (см. таблицу 3).

Таблица 3.

Номер команды	op	arg1	arg2
0	rd		b
1	rd		c
2	rd		f
3	rd		g
4	rd		h
5	+	(–5)	(–4)
6	*	(–1)	(–4)
7	–	(–1)	(–4)
8	/	(–1)	(–4)
9	:=	(–2)	i
10	:=	(–2)	a
11	goto	L2	

Последнюю команду параграфа отмечают управляющим признаком «конец параграфа». Максимальное значение ссылки (окно видимости результата исполненной команды) определяется размером ее поля в команде и равно 2**p–1, где p – размер поля ссылки в битах.

4. Мультиклеточная архитектура

Контекстно-зависимая программа выполняется мультиклеточным процессором (см. рис.1), состоящим из N идентичных клеток с номерами от 0 до n–1, объединенных коммутационной средой. Каждая клетка содержит блок памяти программ (PM), устройство управления (CU), буферные устройства (BUF), а также каждой клетке соответствует коммутационное устройство (SU), совокупность которых образует коммутационную среду (SB).

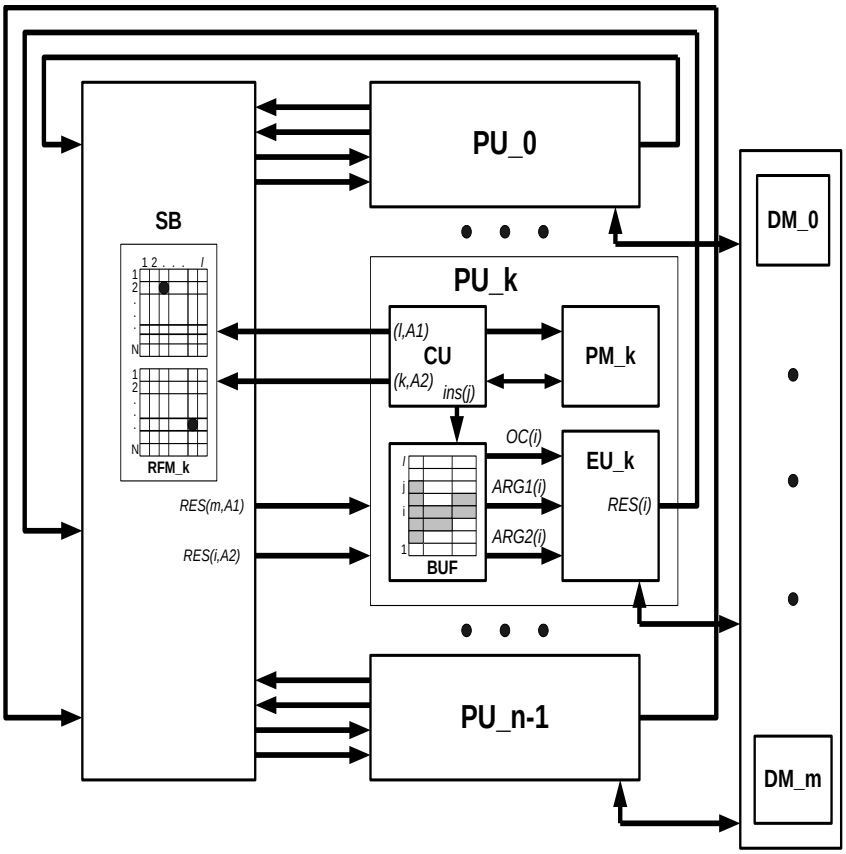

Рис.1. Структура мультиклеточного процессора

Процесс исполнения программы мультиклеточным процессором аналогичен процессу исполнения программы в синпьютере. Основные отличия заключаются в размещении программы в памяти программ, определении по ссылке индивидуальных номеров для каждого запрошенного результата и использовании общих блока памяти данных и блока регистров общего назначения.

Для исполнения контекстно-зависимой программы, команды каждого параграфа последовательно, начиная с одного и того же адреса, являющегося адресом данного параграфа, размещают в РМ клеток. Адрес размещения первого исполняемого параграфа равен адресу, начиная с которого клетки выбирают команды. В результате, в РМ i–ой клетки, начиная с адреса параграфа, последовательно размещаются его команды с номерами i, N+i, 2*N+i,...,.

Каждая клетк обеспечивает, начиная с указанного ему адреса, последовательную выборку команд и размещение их на регистре команд для последующего декодирования. Команды клетками выбираются синхронно.

Необходимыми условиями декодирования очередной команды являются завершение декодирования предыдущей команды всеми клетками, считывание и размещение на регистре команд всеми клетками очередной команды и готовность буферных устройств всех клеток к размещению микрокоманд (наличие свободных строк для записи микрокоманд). Если эти условия выполняются, команду декодируют и размещают в буферных устройствах.

При декодировании каждой очередной выбранной группе из N команд присваивают очередное значение тега (t), а команде из группы и, соответственно, ее результату – номер. Значение очередного тега формируется клетками синхронно и изменяется циклически от 0 до T^{max} –1, где T^{max}=2**p. Минимальное значение T^{max}, при котором принципиально невозможна взаимоблокировка команд, определяется размером окна видимости результатов и для N=2**k равно W/N, где W=2**q – размер окна видимости, q – размер поля ссылки в команде в битах.

Занятость тега фиксируют в регистре занятых тегов устройства управления. Если j–тое значение тега присвоено группе команд (тег занят), то в j–тый бит данного регистра записывают единицу. Значение тега освобождают, когда результат команды с этим тегом поступает в коммутатор. Тогда в соответствующий бит записывают ноль, и это значение может быть присвоено другой группе команд.

Номер команды равен t&i, где i – номер клетки, выбравшей данную команду. По значению номера команды и значениям ее ссылок формируют номера запрашиваемых результатов, которые сообщают коммутационному устройству. Номер запрашиваемого результата определяется следующим образом:

$$N_r = ((t\&i) - P_{op}) \bmod (T^{max})$$

где: N_r - номер запрашиваемого результата; P_{op} – значение ссылки. Следует отметить, что младшие k бит N_r содержат номер клетки, от которой должен прийти запрашиваемый результат, а старшие p – k бит содержат значение тега команды, которая формирует данный результат.

Выборка и декодирование команд продолжается до тех пор, пока не будет выбрана команда, отмеченная управляющим признаком «конец параграфа».

Адрес нового параграфа может поступить как в любой момент выборки команд текущего параграфа, так и после завершения выборки команд. Он поступает всем клеткам одновременно. Если к моменту выборки последней команды параграфа адрес следующего параграфа не вычислен, то выборка приостанавливается до получения адреса. Если адрес получен, то выборка продолжается с этого адреса. Выборка также приостанавливается, если нет места в каком-либо буферном устройстве, если занято очередное значение тега, а также, если по окончанию параграфа не закончилось выполнение команд, формирующих новые значения регистров и не выполнены все записи в память данных.

Буферные устройства формируют команды и передают их соответствующему исполнительному устройству. и состоят из буфера хранения операционной части команды и буфера хранения операндов.

Операционная часть, кроме кода команды, включает в себя всю необходимую служебную информацию для рассылки и приема результатов, а именно, номер команды и признаки готовности первого (второго) операнда для выполнения команды.

Буфер хранения операндов имеет ассоциативную адресацию. Ассоциативным адресом является номер запрашиваемого результата.

В качестве операндов при выполнении операций могут использоваться:

запрошенные результаты команд–источников, поступающие из коммутатора;

значения, вычисленные при декодировании командного слова, а также непосредственно присутствующие в командном слове или взятые из регистров общего назначения.

В первом случае признак готовности данного операнда при записи команды устанавливают в состояние «не готов», а во втором – в состояние «готов». После получения запрошенного результата от коммутационного устройства, признак, в первом случае, также устанавливают в состояние «готов».

Команда, получившая все операнды, проходит приоритетный отбор среди других готовых команд в буферном устройстве, после чего выдается на исполнение при условии незанятости исполнительного устройства.

При записи команды в буфере хранения операционной части устанавливают признак занятости соответствующей строки, который снимают когда команду выдают на исполнение, и, если есть хотя бы одна свободная строка, формируют признак готовности буферного устройства к приему следующей команды.

Доступ к коммутационному устройству осуществляется устройством внутренней рассылки, который обеспечивает выбор наиболее приоритетного результата и выдачу его в коммутатор. Причем, до тех пор, пока разница между номером результата и номером декодируемой в этот момент команды не превысит размер окна видимости, результат задерживают в этом устройстве.

Каждый полученный результат вместе с его тегом посылают во все коммутационные устройства, которые отбирают по номерам, запрошенные данной клеткой результаты, и передают их в ее буферные устройства в качестве операндов.

Каждое коммутационное устройство состоит из N однотипных устройств выбора, порядковый номер которых указывает на клетку, результаты которой отбираются данным устройством выбора и двух блоков передачи операндов. Теги запрошенных результатов хранятся в позиционном виде в памяти запросов устройства выбора. Память запросов представляет собой два регистра размером T^{max} бит, соответственно, для первого и второго операндов. Так при запросе результата с тегом j для использования в качестве первого операнда, в j–том бите первого регистра записывают единицу («запрос установлен»), а после получения результата с этим тегом в j–тый бит записывают ноль («запрос снят»).

В общем случае, каждое коммутационное устройство может одновременно получить до N результатов и все они могут быть отобраны. Для парирования неравномерности их поступления внутри каждого блока передачи операндов используется буферная память.

Так, при поступлении одновременно нескольких отобранных результатов, один или два результата сразу передают в буферные устройства, а остальные записывают в буферную память блока передачи операндов и передают позже. Время задержки результата в буферной памяти определяется интенсивностью и шириной

потока, а также принятой дисциплиной обслуживания результатов, находящихся в буферной памяти.

Для сокращения объёма буферной памяти используют блокировку выдачи результатов в коммутационное устройство тем клеткам, в блоках передачи операндов которых переполнена буферная память. Обеспечивается это сигналами занятости, которые формируют все коммутационные устройства.

5. Особенности мультиклеточной архитектуры

Принципиальным отличием мультиклеточной архитектуры от известных фон-неймановской и потоковой архитектур является использование непосредственного указания информационных связей между командами, и, как следствие, полная независимость объектного программного кода от количества клеток, а также снятие требования упорядоченного размещения команд в программе.

Непосредственное указание информационных связей между операциями обеспечивает возможность одновременного чтения и исполнения нескольких команд без анализа их очередности выполнения, т.е. обеспечивают «естественную» реализацию параллелизма, которая изначально обусловлена видом и механизмами исполнения команд. В мультиклеточном процессоре нет аппаратных средств, обеспечивающих выявление информационных связей между выбранными операциями (командами) и их распределение по функциональным устройствам, т.е. динамическое распараллеливание отсутствует. Нет и статического распараллеливания, т.к. программа в виде триад хотя и описывает информационные связи, но имеет линейную форму и не содержит каких-либо указаний, что и как можно выполнять параллельно. В этом состоит принципиальное отличие от фон-неймановской модели.

Благодаря этой же особенности, потенциально обеспечивается живучесть мультиклеточного процессора, т.е. возможность непрерывного исполнения программы без перекомпиляции или перезагрузки при отказах его отдельных клеток

(деградации процессора). Деградация связана с потерей производительности и, соответственно, увеличением времени решения задач. Но, для целого ряда встроенных применений, живучесть мультиклеточного процессора позволяет управляемому объекту выполнять основные функции, либо за счет снижения их качества, либо за счет отказа от решения второстепенных задач [4]. Подобная независимость кода от используемых ресурсов создает основу для решения задачи непрерывной самоадаптации процессора к потоку задач, а также его самовосстановления после сбоев или подключения новых ресурсов.

Неупорядоченность команд в параграфе позволяет, при необходимости, получать после каждой компиляции индивидуальный объектный код для каждого процессора. Это, а также замкнутость параграфа, резко ограничивают возможности незаметного и несанкционированного вмешательства извне в работу системного программного обеспечения. Индивидуальность системного кода позволяет создать эффективную защиту от вредоносных программ.

Полносвязная интеллектуальная коммутационная среда, работающая в режиме «широковещательной» рассылки, не накладывает каких-либо топологических ограничений на межклеточный обмен данными и обеспечивает эффективную реализацию любого класса задач, что придает универсальность архитектуре и допускает эффективное масштабирование процессора. При увеличении количества клеток и наличии потенциального параллелизма алгоритма, рост производительности процессора практически пропорционален количеству клеток.

Использование в качестве прототипа языка триад дает возможность использовать общепринятые языки высокого уровня, такие, как C и C++, что, наряду с ассемблером, минимизирует риск, связанный с адаптацией наработанного программного обеспечения к новой аппаратной платформе.

Асинхронная и децентрализованная организация мультиклеточного процессора, как на системном уровне – между клетками (при реализации параллелизма), так и на внутриклеточном уровне – между блоками клетки (при реализации команд), дополнительно обеспечивает:

• минимизацию номенклатуры объектов проектирования и уменьшение их сложности;

• уменьшение площади кристалла процессора (объем оборудования при децентрализованном управлении меньше, чем при централизованном);

• увеличение производительности и сокращение энергопотребления за счет реализации более эффективного вычислительного процесса;

• использование индивидуальной системы синхронизации для каждой клетки при реализации на одном кристалле десятков и сотен клеток.

В результате, получается хорошо структурированная и модульная система, позволяющая резко уменьшить сложность процессора и, соответственно, снизить затраты и повысить качество проектирования. При этом по сравнению с фон-неймановской моделью, улучшаются и количественные характеристики процессора.

6. Программное обеспечение

Программное обеспечение для мультиклеточных процессоров представляет собой набор бинарных утилит, разрабатываемых ОАО «Мультиклет» с привлечением сторонних организаций:

- ассемблер;
- редактор связей (компоновщик);
- препроцессор C;
- компилятор C99;
- драйвер сборки;
- функциональную модель процессора;
- программный загрузчик;
- отладчик.

Ассемблер и редактор связей являются разработками компании «Мультиклет», хотя общие условия их использования схожи с аналогичными бинарными утилитами GNU, информация о которых может быть найдена по адресу

http://sourceware.org/binutils/. Препроцессор C является полностью сторонней разработкой, см. http://mcpp.sourceforgr.net/.

Разрабатываемый компанией «Мультиклет» компилятор C99, в основе которого лежит новая технология LiME, имеет следующие особенности:

• унифицированную обработку исходных текстов на языках программирования с разными грамматиками. Подход LiME, основанный на выводе графа программы по специальным формам и описанию синтаксической структуры текста на универсальном язык, должен существенно упростить разработку интерфейсов других языков программирования;

• возможность расширения языка пользователем путем создания библиотеки собственных языковых конструкций;

• промежуточное представление программ в LiME более абстрактно, чем в традиционных процессорах с фон-неймановской архитектурой. Это позволяет реализовать более эффективные компиляторы для процессоров с принципиально новыми архитектурами.

Представленные выше утилиты разрабатывались как кросс-платформенные, и на сегодняшний день могут быть использованы под управлением операционных систем Windows и Linux.

В ближайших планах компании стоят такие задачи, как сопряжение инструментария разработки ПО для процессоров MULTICLET с существующей свободно распространяемой средой разработки, адаптация ОСРВ (в частности, eCos; портирование FreeRTOS завершено), а также окончание работ над альфа-версией компилятора Си-99 на базе каркаса для построения событийно-управляемых трансляторов LiME [5]. В 2014-2015 годах ОАО «Мультиклет» планирует завершение двух потребительских проектов по выпуску мультиклетов – устройств с широким спектром применения.

7. Сравнение мультиклеточного процессора MULTICLET P1 с аналогами

Большинство существующих на сегодня процессоров имеют фон-неймановскую архитектуру (CISC, RISC, VLIW и пр.) и, при использовании идентичных технологических процессов и методологий проектирования, уступают мультиклеточным процессорам как по быстродействию и энергопотреблению (в удельных показателях), см. табл.1.

№ п/п	Параметры/Процессор	Мультиклет	Отечественные аналоги		Зарубежные аналоги				
		Мультиклет МСр0411100101	**Элвис 1892ВМ3Т(МС-12)**	**Элвис 1892ВМ5Я(МС-0226)**	**Texas Instruments OMAP L-138 (TI)**	**Analog Devices ADSP-21469**	**Texas Instruments TMS320C6701**	**Analog Devices ADSP - TS201S**	**Intel Pentium 4**
Технологические характеристики									
1	**Ядро**	Мультиклет	Мультикор	Мультикор	C674+ARM9	SHARC	TMS320C6000	TigerSHARC	Willamette
2	**Кол-во клеток / кол-во ядер / кол-во исполнительных устройств**	4 / - / -	- / 2 / -	- / 3 / -	- / 1 / 8	- / - / 6	- / - / 8	- / - / 6	- / 1 / -
3	**Архитектура**	Мультиклеточная	MIPS32 + RISC	MIPS32 + RISC	Гарвардская RISC	Гарвардская	Гарвардская	Гарвардская	Гарвардская IA-32
4	**Тип корпуса**	QFP-208	PQFP-240	BGA-416	PBGA-361	PBGA-324	BGA-352	Ball BGA-576	FC-PGA2-423
5	**Технологический процесс (мкм)**	0,18	0,25	0,25	0,065	0,065	0,18	0,13	0.18
6	**Температурный диапазон**	от - 60 до +125° С[2]	от -65 до 85° С	-от 65 до 85° С	от -40 до 90° С	от -40 до +85° С	от -40 до 105° С	от -40 до +85° С	от -40 до +85° С
7	**Площадь кристалла (кв.мм)**	100	НД[1]	НД[1]	НД[1]	НД[1]	НД[1]	НД[1]	217

8	**Разрядность (бит)**		32/64	32	32	32	32	32	32/64	32
9	**Тактовая частота (МГц)**		100	80	100	456	400	167	600	2000
10	**Производительность (Гфлопс)**		2,4	0,24	1,2	2,7	2,4	1	3,6	2
11	**Память (Кб)**	**ПЗУ**[4]	–	134	128	64	500	64	НД[1]	НД[1]
		ОЗУ	ПД - 128 (4*4К*64) ПП - 128 (4*4К*64)	16	16	8	625	64	3000	НД[1]
	Напряжение (В)	**Ядра**	1,8	2,5	2,5	1,8	1,8	1,8	1,2	1,8
12		**Периферии**	3,3	3,3	3,3	3,3	3,3	3,3	3,3	НД[1]
13	**Максимальная потребляемая мощность (Вт)**		1,08	1,2	1,6	1	1	1,8	3,4	100
14	**Арифметика с плавающей запятой**		+	+	+	+	+	+	+	+
Характеристики периферии										
15	**SPI**		2	–	–	2	2	McBSPs[3]	–	–
16	**I2S**		1	–	–	1	1	–	–	–
17	**I2C**		2	–	–	2	1	–	–	–
18	**USART**		–	–	–	–	–	–	–	–
19	**UART**		4	1	1	3	1	–	–	–
20	**USB**		1.1 FS (device)	–	–	2	1.1 FS (host) + 2.0	–	–	-
21	**Ethernet**		Ethernet контроллер 10/100Мб/с	–	–	Ethernet контроллер 10/100Мб/с	–	–	–	-
22	**PWM**		1	–	–	2	4	–	–	–

23	**GPIO**	104	НД[1]	НД[1]	НД[1]	НД[1]	НД[1]	НД[1]	НД[1]
24	**Системный таймер, разрядность**	1/32	–	–	–	1/32	–	–	НД[1]
25	**Таймер общего применения, разрядность**	7/32	1/32	1/32	3/64	2/32	2/32	2/64	НД[1]
26	**Сторожевой таймер**	1	1	1	1	1	1	1	1
Программное обеспечение									
28	**Ассемблер**	+	+	+	+	+	+	+	+
29	**Си компилятор**	+	+	+	+	+	+	+	+
30	**Отладчик, JTAG**	+	+	+	+	+	+	+	+
31	**Операционная система**	FreeRTOS	Linux, QNX	Linux, QNX	Linux, BIOS, Win CE	НД[1]	НД[1]	НД[1]	Windows, Linux
32	**Среда разработки**	+	+	+	+	+	+	+	+
Прочие параметры									
33	**Область применения**	обще-промышленная	обще-промышленная	общепромышленная	обще-промышленная	обработка аудио сигналов	обще-промышленная	обще-промышленная	обще-промышленная
34	**Стоимость (руб)**	от 565	по запросу	по запросу	600	1200	по запросу	9 000	по запросу

Данные взяты из открытых интернет-источников, компания "Мультиклет" не несет ответственности за их корректность

[1] НД - нет данных

[2] В пластиковом корпусе

[3] McBSPs - многоканальный последовательный порт

[4] Список рекомендуемых флэш ПЗУ для использования с процессором МСр0411100101

№ п/п	Наименование	Производитель	Корпус	Питание, В	Температ. диапазон	Поставщики в РФ	Рекомендовано МО РФ*
1	XCF04S	Xilinx	VO20/VOG20	1.8 – 3.3	–40°C +85°C	Макро Групп macrogroup.ru	нет

2	XCF08P	Xilinx	VO48/VOG48 FS48/FSG48	1.8 – 3.3	–40°C +85°C	Макро Групп macrogroup.ru	нет
3	XCF16P	Xilinx	VO48/VOG48 FS48/FSG48	1.8 – 3.3	–40°C +85°C	Макро Групп macrogroup.ru	да
4	XCF32P	Xilinx	VO48/VOG48 FS48/FSG48	1.8 – 3.3	–40°C +85°C	Макро Групп macrogroup.ru	да

* Рекомендовано МО РФ в Справочных материалах "Рационально-унифицированная и оптимизированная номенклатура ЭКБ иностранного производства для применения в РЭА" "Номенклатура 2012" Книга 2

8. Конкурентные преимущества мультиклетов

• увеличение производительности в 4-5 раз при одновременном снижении энергопотребления (в 2-4 раза по сравнению с аудиопроцессорами; в 10-15 раз по сравнению с процессорными ядрами со сверхнизким энергопотреблением компаний TI, ARM);

• «естественная» реализация параллелизма (без решения задачи распараллеливания);

• уменьшение площади кристалла;

• эффективная реализация любого класса задач (коммутационная среда не накладывает ограничений на межклеточный обмен данными);

• выполнение программы без перекомпиляции на любом количестве клеток;

• непрерывное выполнение программы при деградации аппаратной среды (отказах клеток);

• естественный иммунитет к вредоносному коду.

9. Сфера применения мультиклетов

Мультиклеточная архитектура является универсальной архитектурой, что обеспечивает широкую сферу применения – от процессоров для персональных компьютеров и суперкомпьютеров, до специализированных промышленных систем. Структуру мирового рынка специализированных процессоров иллюстрирует рис.2.

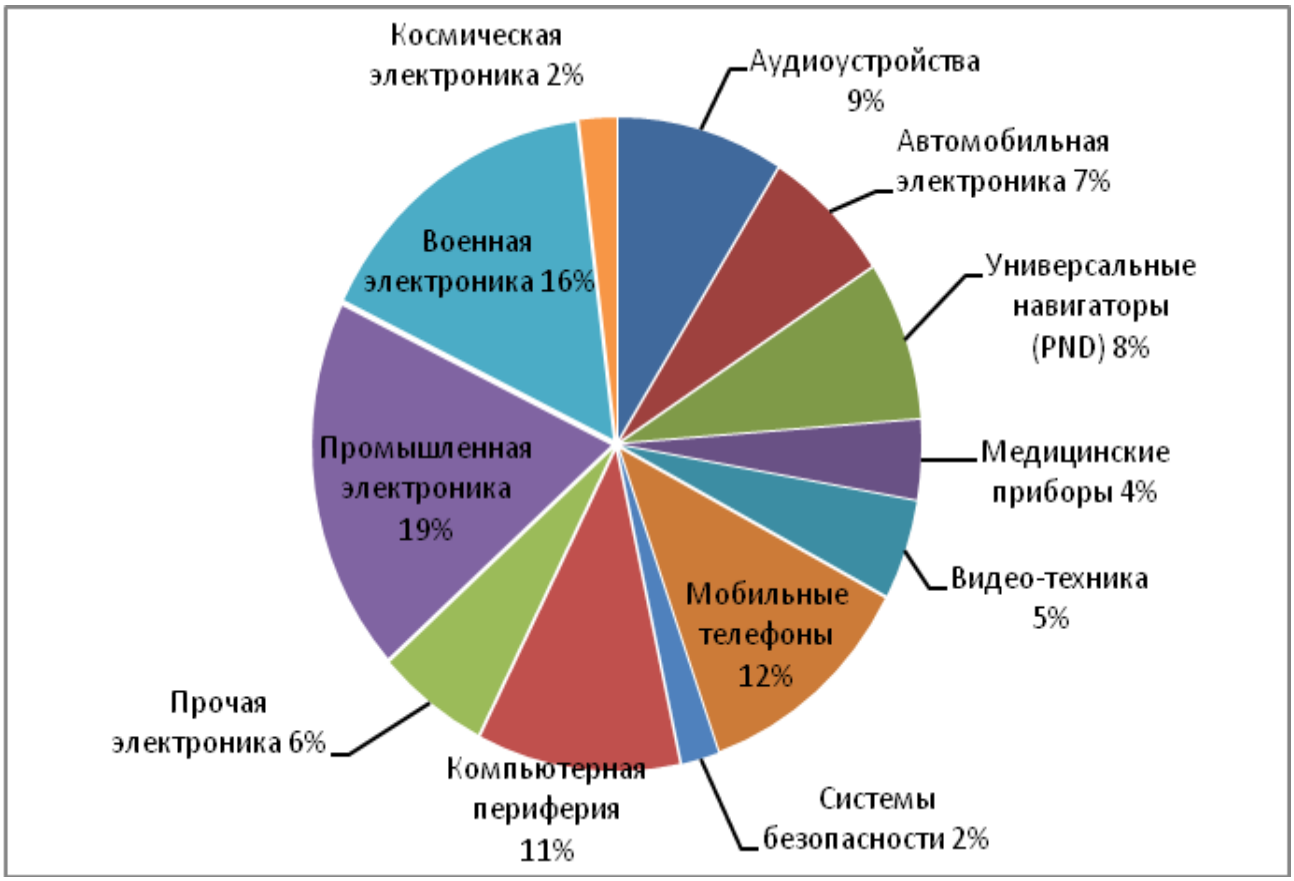

Рис.2. Диаграмма распределения специализированных процессоров

Мультиклеты – самостоятельно или в составе сборки – могут успешно использоваться в различных областях техники:

• космическом и авиационном оборудовании, как бортовом, так и наземного базирования;

• промышленной аппаратуре;

• специальных приложениях на FPGA;

• автомобильной электронике («интеллектуальные» бортовые системы);

• настольных суперкомпьютерах терафлопного класса;

• траст-процессорах «Антихакер» для банковских приложений;

• приемниках ГЛОНАСС/GPS/Galileo;

• звуковых процессорах;

• 3D телевидении;

• мобильной и видеосвязи.

10. Перспективы развития

На начало 2014 года запланирован выход мультиклеточного процессора MULTICLET P2 (серии P – Performance), который ориентирован на максимальную производительность при одновременном снижении энергопотребления. В процессоре MULTICLET P2 увеличены тактовая частота и объем памяти на кристалле, расширен состав периферийных устройств (см. рис.3).

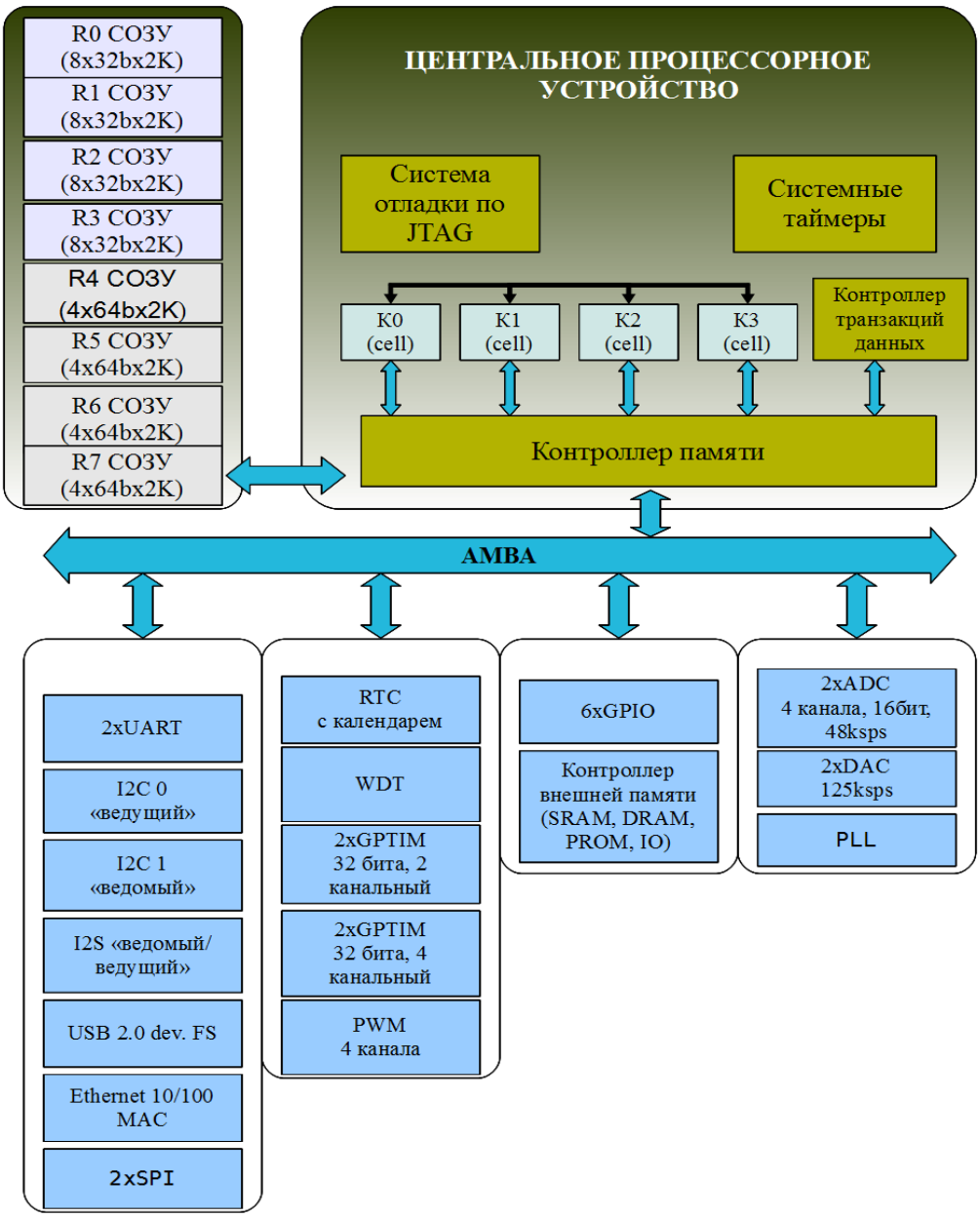

Рис.2. Структурная схема процессора MULTICLET P2

Также в ближайших планах компании – выпуск первого процессора с динамической реконфигурацией MULTICLET R1 серии «R» (Reconfiguration). Процессоры MULTICLET P2 и MULTICLET R1 создаются для общепромышленного применения, а также могут использоваться в автомобильной и специальной электронике. Выпуск «живучего» микропроцессора MULTICLET L1 серии «L» (Liveness), способного выполнять без перезагрузки и перекомпиляции свои программы даже при выходе из строя одной, двух и более клеток, намечен на конец 2014 года.

Проводятся работы по созданию мультиклеточной архитектуры на базе поименной рассылки результатов.

В планах компании расширение линейки для потребностей радиоэлектронной промышленности, что требует дополнительного проведения НИР в направлениях:

- Разработка 1024-клеточных высокопроизводительных микропроцессоров не фон-неймановской мультиклеточной архитектурой, с удельной производительностью 440 ГФлопс/Вт для построения вычислительных систем экзафлопного класса, на топологической норме 22нм.

- Разработка 128-клеточных высокопроизводительных (до 0,7 ТФлопс) микропроцессоров с не фон-неймановской мультиклеточной архитектурой, удельной производительностью 600 ГФлопс/Вт, для создания универсальных плат-ускорителей, применяемых в существующих персональных компьютерах, а также при разработке портативных суперкомпьютерных систем, на топологической норме 40нм.

- Разработка 16 - клеточных микропроцессоров с производительностью не менее 0,1 ТФлопс, оптимизированных для использования в коммутационно-связном оборудовании на базе мультиклеточной архитектуры для особо ответственных применений и импортозамещения, на топологической норме 40нм.

- Разработка 4-клеточных радиационно-стойких высокопроизводительных (до 3 ГФлопс) микропроцессоров с не фон-неймановской мультиклеточной архитектурой, удельной производительностью 6 ГФлопс/Вт, для применений в космической и атомной отраслях, на топологической норме 180нм.

11. Литература

1. K. Sankaralingam, R. Nagarajan, P. Gratz, R. Desikan, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, W. Yoder, R. McDonald, S.W. Keckler, and D.C. Burger The Distributed Microarchitecture of the TRIPS Prototype Processor. 39th International Symposium on Microarchitecture (MICRO), December, 2006.

2. Патент РФ № 2179333 на изобретение «Синергетическая вычислительная система», дата подачи 13.06.2000 г., опубликовано 10.02.2002 г., Streltsov N., Sparso J., Bokov S., Kleberg S. The Synputer – A Novel MIMD Processor Targeting High Performance Low Power DSP Applications // International Signal Processing Conference, Dallas, 1-3 April, 2003).

3. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты – М,: Диалектика, 2003.

4. Зырянов Б.А., Стрельцов Н.В. Обеспечение живучести мультиклеточного процессора. Труды РАСО'2012, ИПУ РАН, 2012, с.126.

5. Бахтерев М.О., Косенко В.В. Разработка компилятора для языка программирования RiDE.L. Саров, РОСАТОМ, 2010, с.16-17.