УДК 681.3

АРХИТЕКТУРА И РЕАЛИЗАЦИЯ МУЛЬТИКЛЕТОЧНЫХ ПРОЦЕССОРОВ

Н.В.Стрельцов
ОАО «МУЛЬТИКЛЕТ»
Россия, 620014, г. Екатеринбург, ул. Челюскинцев, 2, оф.68
E-mail: synputer@mail.ur.ru

Ключевые слова: параллельные процессоры, не-фон-неймановская архитектура, параллелизм на уровне системы команд
Key words: parallel processors, non von Newman architecture, parallelism on the instruction level

Описываются архитектура и конкретная реализация процессорного ядра MCP-1.1.xx. MCP-1.1.xx – первое ядро с принципиально новой (пост-неймановской) мультиклеточной архитектурой. Предназначено для решения задач управления и цифровой обработки сигналов в приложениях, требующих минимальное энергопотребление и высокую производительность, например, при обработке аудиоинформации. Мультиклеточный процессор может состоять из 4, 8 или 16 клеток, объединенных интеллектуальной коммутационной средой. Клетки процессора имеют систему команд, построенную на базе языка триад. Типы данных – целые и дробные (как знаковые, так и беззнаковые числа) одинарной - 16(24) бит или двойной точности – 32(48) бит, а также дробные знаковые и беззнаковые упакованные (комплексные) числа одинарной точности – 32(48) бит. На данный момент разработано и отработано на модели RTL описание процессора на 4, 8 и 16 клеток. Проведена отработка на FPGA-модели (XC2V4000) 4-х клеточного процессора и выполнен его синтез для техпроцесса 0,18μm, V=1,8V (варианты: 10Mhz/40MIPS; 50Mhz/200MIPS). Получены оценочные характеристики по производительности и энергопотреблению.

ARCHITECTURE AND REALIZATION OF MULTICELLULAR PROCESSORS / N.V. Streltsov (MULTICLET corp., of. 68, Cheluskintsev str. 2, Ekaterinburg, 620014, Russia, E-mail:synputer@mail.ur.ru). There are described the architecture and the specific realization of processor core MCP-1.1.xx. The MCP-1.1.xx is the first core with a principally new (post-Neumann) multicellular architecture. It is designed to solve the tasks of controlling and digital signal processing in the applications which require minimal power consumption as well as superior performance, for instance, processing of audio information. The MCP-processor can consist of 4, 8 or 16 cells integrated by the intellectual commutation environment. The processor cells have the instruction set that is made on the triad language basis. The data types – integer and fractional (both signed and unsigned numbers) of single precision – 16(24) bits or of double precision – 32(48) bits, as well as fractional signed and unsigned packed (complex) numbers of single precision - 32(48) bits. For the present moment the description of the processor for 4, 8 and 16 cells is developed and tested at the RTL model. The 4-cell processor is tested at the FPGA-model (XC2V4000) and its synthesis for the 0,18μm, V=1,8V technological process is carried out (versions: 10Mhz/40MIPS; 50Mhz/200MIPS). We received assessment features with regard to performance and power consumption..

1. Введение

Анализ тенденций в развитии микропроцессоров показывают, что в компьютерной индустрии создалась и постоянно усугубляется ситуация, когда применяемая архитектурная модель процессора не обеспечивает эффективное использование возможностей предоставляемых технологией. Активно внедряемая многоядерность, признанная на данный момент основным направлением совершенствования микропроцессоров, также не решает этой проблемы. Требуется новая архитектурная модель, не только решающая текущие проблемы

создания микропроцессоров, например проблему сложности, но и создающая предпосылки для дальнейшего развития микропроцессорной техники.

Влияние процессорной архитектуры на развитие компьютерной индустрии трудно переоценить. Она прямо, или косвенно, влияет буквально на все компоненты компьютерных систем – как на аппаратные, так и на программные. Поэтому исследование и внедрение новых архитектурных решений, доминирование в этой сфере, имеет ключевое значение для решения задачи выхода на лидирующие позиции в микропроцессорной технике.

2. Постановка задачи

Известно, что основной тенденцией развития фон-неймановской архитектуры было повышение уровня параллелизма и, соответственно, стремление обойти требование последовательной выборки и исполнения команд. Это требование – следствие опосредованной (через память) формы реализации информационных связей между командами.

Попытки уйти от опосредованной формы и предложить рынку другую модель предпринимались неоднократно, например, потоковые и редукционные машины. В них информационные связи между командами задавались явно (непосредственно). После получения результат немедленно передавался потребителю. Все эти попытки были неудачны, так как требовали отказа от императивных языков программирования.

Но, непосредственные связи можно реализовать используя в качестве языка процессора промежуточное, машинно-независимое представление программы в виде триад, получаемое после первой фазы компиляции императивных языков.

Это решение принципиально отличается и от фон-неймановской модели и от потоковой («data flow») модели. Обе они, несмотря на все их различия, относятся к архитектурам с хранимой программой (контекстно-свободной программой). Триады контекстно-зависимы и, следовательно, использующая их модель относится к классу архитектур с хранимым алгоритмом (контекстно-зависимой программой) [1].

Система команд на базе триад, фактически, является аппаратной реализацией входного языка программирования, сохраняющей не форму, а сущность выражений языка. Именно отказ от сохранения формы позволил впервые, за многолетнию историю попыток аппаратной реализации языков высокого уровня, получить эффективное решение – мультиклеточную архитектуру.

Для подтверждения эффективности архитектуры было разработано процессорное ядро, предназначенное для решения задач управления и цифровой обработки сигналов в приложениях, требующих высокую производительность при минимальном энергопотребление. Выбор области использования обусловлен экономическими критериями. А именно, стоимостью и возможностью выхода на рынок.

На данный момент разработано и отработано на программной модели RTL описание данного ядра на 4, 8 и 16 клеток. Проведена отработка на FPGA-модели (XC2V4000) 4-х клеточного ядра и выполнен его синтез для техпроцесса 0,18μm, V=1,8V (варианты: 10Mhz/40MIPS; 50Mhz/200MIPS). Получены оценочные характеристики по производительности и энергопотреблению.

3. Особенности мультиклеточной прхитектуры

Качественные отличия предлагаемой мультиклеточной архитектуры [1] следующие:

1. От фон-неймановской модели мультиклеточная архитектура отличается непосредственным указанием информационных связей между операциями и, соответственно, снятием требования упорядоченного размещения описаний операций в программе.

Эта неупорядоченность делает ненужными все те методы (суперскалярность, широкое командное слово, суперконвейер, предсказание переходов и т.п.), которые резко усложняли процессы проектирования процессора и инструментальных программных средств.

2. От известных не-фон-неймановских архитектур она отличается использованием традиционных императивных языков программирования, последовательным способом выборки команд, использованием для указания информационных связей не адресов команд, а значений динамически формируемых тегов, а также механизмом исполнения команд - не только по «готовности данных», но и по «готовности потребителей ее результата».

3. Система команд клетки, основана на промежуточном представлении компилируемой программы после синтаксического анализа (триадах) и, фактически, является аппаратной реализацией входного языка программирования.

4. Неупорядоченность триад обеспечивает, при необходимости, получение после каждой компиляции индивидуального объектного кода для каждого процессора. Это, а также замкнутость подмножеств триад, резко ограничивают возможности незаметного и несанкционированного вмешательства извне в работу системного программного обеспечения.

5. Триады обеспечивают возможность одновременного чтения и исполнения нескольких команд без анализа их очередности выполнения и информационной связности т.е. обеспечивают «естественную» реализацию параллелизма.

6. Полносвязная интеллектуальная коммутационная среда, работающая в режиме «широковещательной» рассылки, не вносит каких-либо топологических ограничений на межклеточный обмен данными и, следовательно, обеспечивает эффективную реализацию любого класса задач, а также эффективное масштабирование процессора.

7. Откомпилированная программа может быть выполнена на любом количестве клеток. При этом возможно динамическое изменение их количества, что обеспечивает реализацию методологии постепенной деградации процессора при отказах его клеток. Подобная независимость кода от используемых ресурсов создает также основу решения проблем непрерывной самоадаптации процессора к потоку задач и его самовосстановления после сбоев или после подключения новых ресурсов.

8. 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 и сложности объектов проектирования;
- уменьшение площади кристалла, так как объем оборудования при децентрализованном управлении меньше, чем при централизованном;
- увеличение производительности и сокращение энергопотребления в несколько раз (см. характеристики), так как позволяет реализовать эффективный вычислительный процесс;
- при реализации, в перспективе, на одном кристалле десятков и сотен клеток - использование индивидуальной системы синхронизации для каждой клетки.

4. Реализация процессорного ядра

4.1. Структура

Структурная схема ядра процессорного ядра представлена на рис.1.

Ядро включает:

n_pu идентичных процессорных блоков (клеток), имеющих нумерацию от *0* до (*n_pu-1*);

поле регистров общего назначения;

коммутатор.

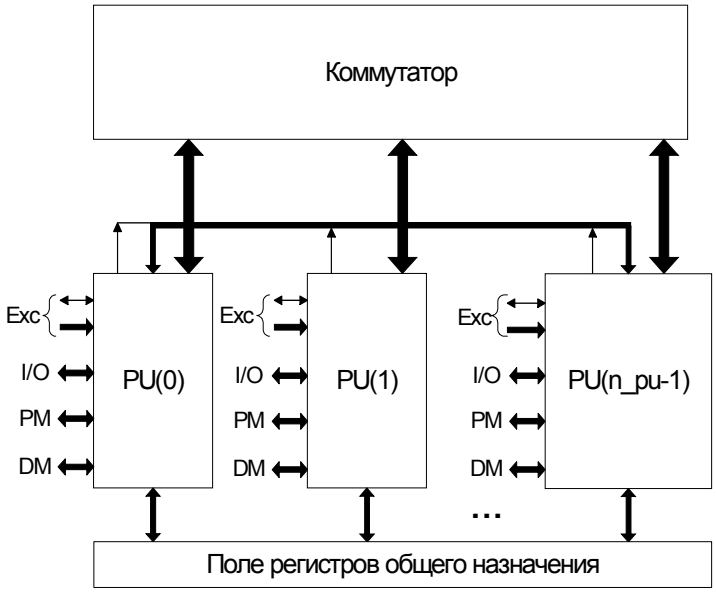


Рис.1. Структурная схема процессоргого ядра

4.2. Форматы данных

Операции в процессорных элементах выполняются над знаковыми:

- дробными числами длиной dw бит (полусловами), где: dw=16 или dw=24;
- дробными числами двойной точности(словами) длиной (dw*2) бит;
- дробными упакованными числами(словами) длиною (dw*2) бит;
- комплексными дробными числами(словами) длиною (dw*2) бит;
- короткими целыми числами длиной dw бит (полусловами), где: dw=16 или dw=24;
- целыми числами длиной (dw*2) бит .

А также над беззнаковыми:

- дробными числами длиной dw бит (полусловами);
- дробными числами двойной точности(словами) длиной (dw*2) бит;
- дробными упакованными числами(словами) длиною (dw*2) бит;
- короткими целыми числами длиной dw бит (полусловами), где: dw=16 или dw=24;
- целыми числами длиной (dw*2) бит.

Тип данных задается в операциях чтения данных четырехбитным кодом. Этот код (см. таблицу) сопровождает выбранные данные на всех последующих этапах их обработки с момента чтения их из памяти.

Таблица - Типы и кодировка типов данных

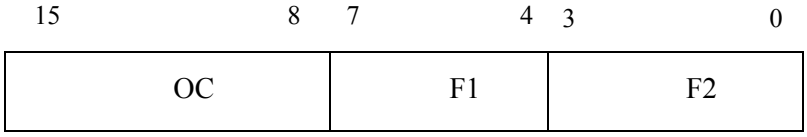
	Типы данных	Кодировка типов
1	Неопределенный	0000
2		0001
3		0010
4	Знаковый короткий целый (short)	0011

5	Знаковый длинный целый (long)	0100
6	Знаковый дробный (fixed)	0101
7	Знаковый дробный упакованный (pack_fixed)	0110
8	Знаковый двойной дробный (double_fixed)	0111
9	Знаковый комплексный дробный (complex_fixed)	1000
10		1001
11		1010
12	Беззнаковый короткий целый (uns_short)	1011
13	Беззнаковый длинный целый (uns_long)	1100
14	Беззнаковый дробный (uns_fixed)	1101
15	Беззнаковый дробный упакованный (uns_pack_fixed)	1110
16	Беззнаковый двойной дробный (uns_double_fixed)	1111

4.3. Система команд

Командные слова имеют два формата (AA,AV), наименование которых, а также наименование их полей даны исходя из типовых вариантов их использования. В отдельных командах возможно другое использование поля, например, как 4-х разрядного кода.

Формат AA определяет группу команд, использующих в качестве первого операнда адрес коммутатора, а в качестве второго операнда либо адрес коммутатора, либо номер регистра общего назначения. Структура командного слова формата AA приведена ниже.

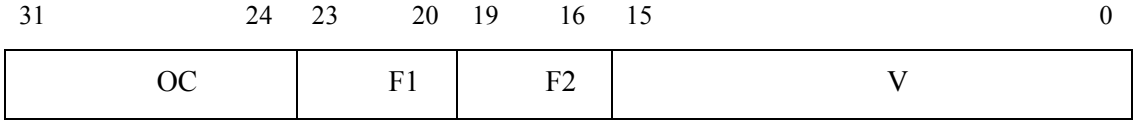


OC – код операции;
F1 – поле адреса первого операнда;
F2 – поле адреса второго операнда.

Формат AV задает группу команд, использующих в качестве первого операнда адрес коммутатора, а в качестве второго операнда значение, определяемое полями F2 и V содержащимися в команде. Это значение может быть использовано в качестве:

- числа (кода);
- исполнительного адреса для доступа к памяти данных.

Структура командного слова формата AV следующая:



ОС - код операции;

F1 – поле адреса первого операнда;

F2 – поле, являющееся адресом коммутатора либо номером регистра общего назначения, используемого для определения второго операнда;

V – 32-х разрядный код, используемый для определения второго операнда.

Состав команд показан в нижеследующей таблице.

Таблица - Состав команд

Наименование операции	Мнемокод	Код операции
Упаковка числа	pack	000000xx
Сдвиг кода	shift	000001xx
Выбор наибольшего	max	000010xx
Выбор наименьшего	min	000011xx
Абсолютное значение	abs	000100xx
Сложение	add	000110xx
Вычитание	sub	001000xx
Обратное вычитание	insub	001001xx
Логическое сложение	or	001010xx
Логическое умножение	and	001100xx
Отрицание	not	001101xx
Сложение по mod2	xor	001110xx
Нормализация (поиск первой единицы)	norm	001111xx
Округление	rnd	010000xx
Умножение без округления	mpyh	010001xx
Умножение	mul	010010xx
Проверка на EQ	eq	011100xx
Проверка на LT	lt	011101xx
Проверка на GT	gt	011110xx
Чтение	rd	100000xx
Извлечение	get	100100xx
Запись	wr	101000xx
Запись с формированием признака конца линейного участка	wrle	101001xx

Установка PON	set	10110001 10110010 10110011
Установка первого теневого PON	sh1set	10110101 10110110 10110111
Установка второго теневого PON	sh2set	10111001 10111010 10111011
Передача управления по 'true'	then	110000xx
Передача управления по 'false'	else	110001xx
Цикл	loop	11001001 11001010
Установка PSW	begin	11110001 11110010 11110011
Ожидание	wait	11110100
Сброс	reset	11111000
Конец линейного участка	endline	11111100

4.4. Процессорный блок

Процессорный блок включает:

- планировщик памяти программ (PMS);
- блок управления (CU);
- буфера хранения команд (ASUMUL_BUF, DMS_BUF);
- исполнительные устройства для выполнения арифметических, логических операций (ASU), операций умножения (MUL), доступа к памяти данных (DMS);
- блок внутренней рассылки (ICU).

Структурная схема процессорного блока показана на рис.2.

Планировщик памяти программ обеспечивает, начиная с указанного ему адреса и до выборки команды «конец линейного участка», последовательную выборку команд и размещение их на регистре команд.

Команды выбираются из памяти программ (PM), которая рассматривается как одномерный массив размерностью 2^{32} полуслов (от PM(0) до PM (2^{32}-1)). Нулевой элемент массива может содержать любую команду. Команды, состоящие из трех полуслов могут размещаться с любого адреса.

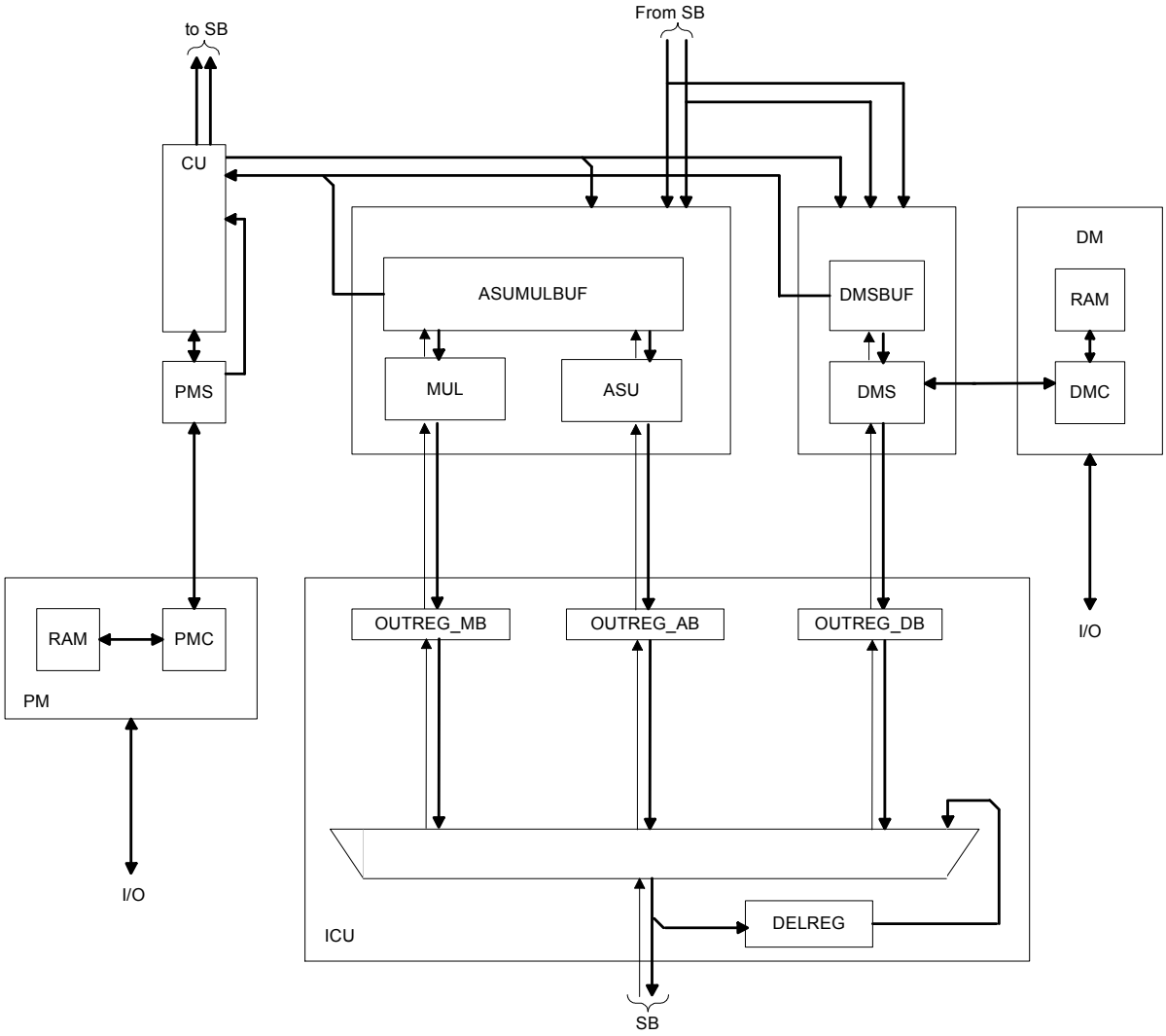


Рис.2. Структурная схема процессорного блока

Программа процессора рассматривается как набор последовательно размещенных линейных участков. Каждый линейный участок размещается начиная с PU0. Команды линейного участка размещаются последовательно. Каждая очередная команда размещается в сегменте PM принадлежащем очередному PU. При этом, формата AV полностью размещается в одном сегменте в ячейках с логическими адресами A, A+ n_pu и A+ 2*n_pu. Так, например, следующая последовательность команд, образованная двумя линейными участками:

«av0,av1,aa2,aa3,aa4,av5» «aa6,aa7,av8,av9,aa10, av11,av12»

может быть размещена так как показано на рисунке. Порядок считывания инструкций показан в нижеследующей таблице увеличением насыщенности цветового тона.

Таблица - Размещение линейных участков в PM

PU0	PU1	PU2	PU3
av0	av1	aa2	aa3
av0	av1		
aa4	av5		
	av5		
aa6	aa7	av8	av9
aa10	av11	av8	av9
	av11	av12	
		av12	

Блок управления обеспечивает:

- управление процессом выборки и загрузки команд в регистр команд;
- декодирования команд, запись их микрокоманд в буфера исполнительных устройств и выставление запросов на аргументы в коммутатор.

Необходимыми условиями декодирования очередной команды является завершение декодирования предыдущей команды всеми PU, считывание и размещение на регистре команд всеми PU очередной команды и готовность буферов всех PU к размещению микрокоманд (наличие свободных строк для записи микрокоманд). Если эти условия выполняются, команда декодируется и ее микрокоманды размещаются в буферах исполнительных устройств (арифметических блоков и блока обращения к памяти данных). Команды, не использующие косвенную адресацию, декодируются в одну микрокоманду, команды, использующие косвенную адресацию, декодируются в две микрокоманды, одна из которых записывается в буфер блока обращения к памяти данных, а вторая в буфер арифметических блоков (сумматор, умножитель).

Выборка и декодирование команд продолжается до тех пор пока не будет выбрана команда, формирующая признак «конец линейного участка». Адрес нового линейного участка может поступить как в любой момент выборки команд текущего линейного участка, так и после завершения выборки команд. Он поступает всем PU одновременно. Если к моменту выборки команды «конец линейного участка» адрес следующего линейного участка не вычислен, то выборка приостанавливается до получения адреса. Если адрес получен, то выборка продолжается с этого адреса.

Функционально буфера решают задачу формирования команд и передачу их соответствующему исполнительному устройству.

Буфер, как показано на рис. 3 состоит из:

- буфера хранения операционной части микрокоманды OP_BL;
- буфера хранения аргументов ARG1(2)_BL.

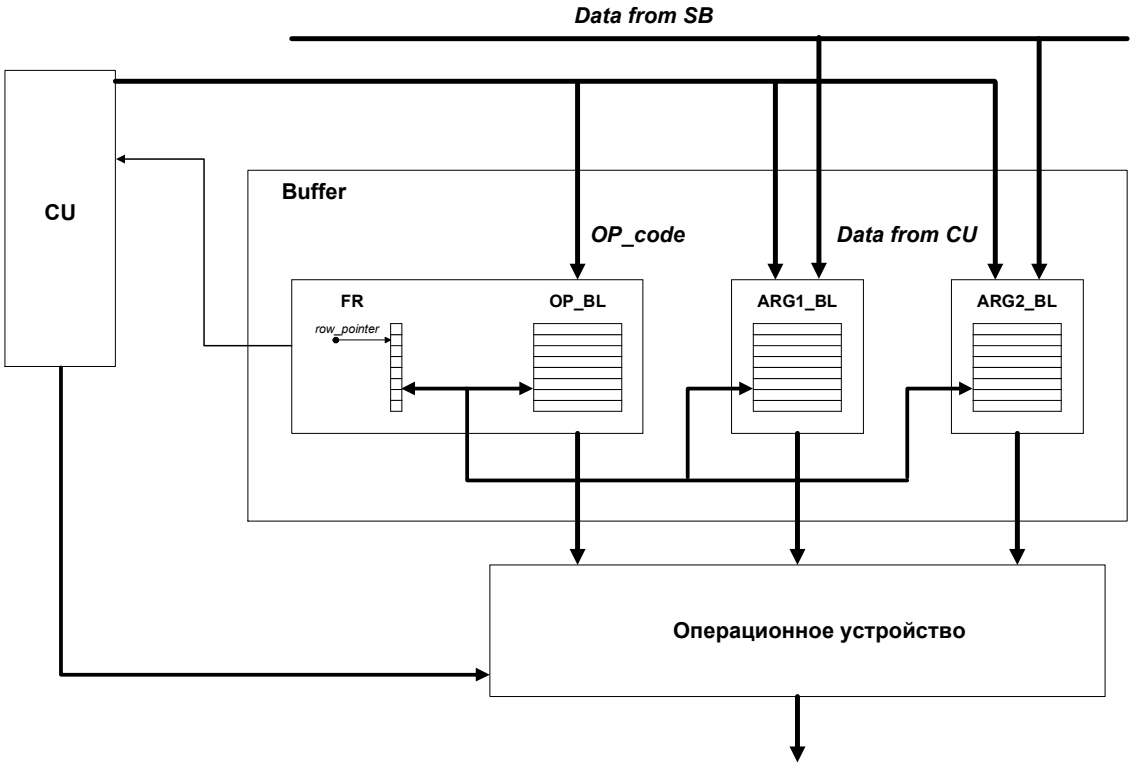


Рис.3. Структура буфера

Формирование команды включает три шага:

- запись операционной части команды в буфер;
- приём аргументов;
- выдачу команды на исполнение.

Операционная часть, кроме кода команды, включает в себя всю необходимую служебную информацию для приема аргументов и рассылки результатов, а именно тег команды и признаки необходимости первого(второго) аргумента для выполнения команды.

Буфер имеет ассоциативную адресацию. Ассоциативным адресом является тег команды-источника.

В качестве аргументов при выполнении операций могут использоваться:

- данные, поступающие из коммутатора;
- данные, находящиеся в регистрах общего назначения;
- значения, вычисленные при декодировании командного слова или непосредственно присутствующие в командном слове.

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

Интерфейсные сигналы буфера приведены на рис. 4.

Информация в буфере хранения операционной части показывает, какую операцию надо выполнить.

Сигналом для записи операционной части команды является значение *op_ready = ‘1’*. При появлении этого сигнала в строку буфера хранения операционной части записывается код команды и служебная информация. Запись производится в строку, на которую указывает вектор *sel_row* (фактически, это позиционный код номера строки *row_num*).

Сигнал готовности команды для выполнения *both_ready* вырабатывается, когда для нее готовы все необходимые аргументы. Этот сигнал используется в приоритетном отборе для выбора строки на исполнение. Вектор *open_row* открывает выбранную строку в буфере хранения операционной части и в буферах хранения аргументов.

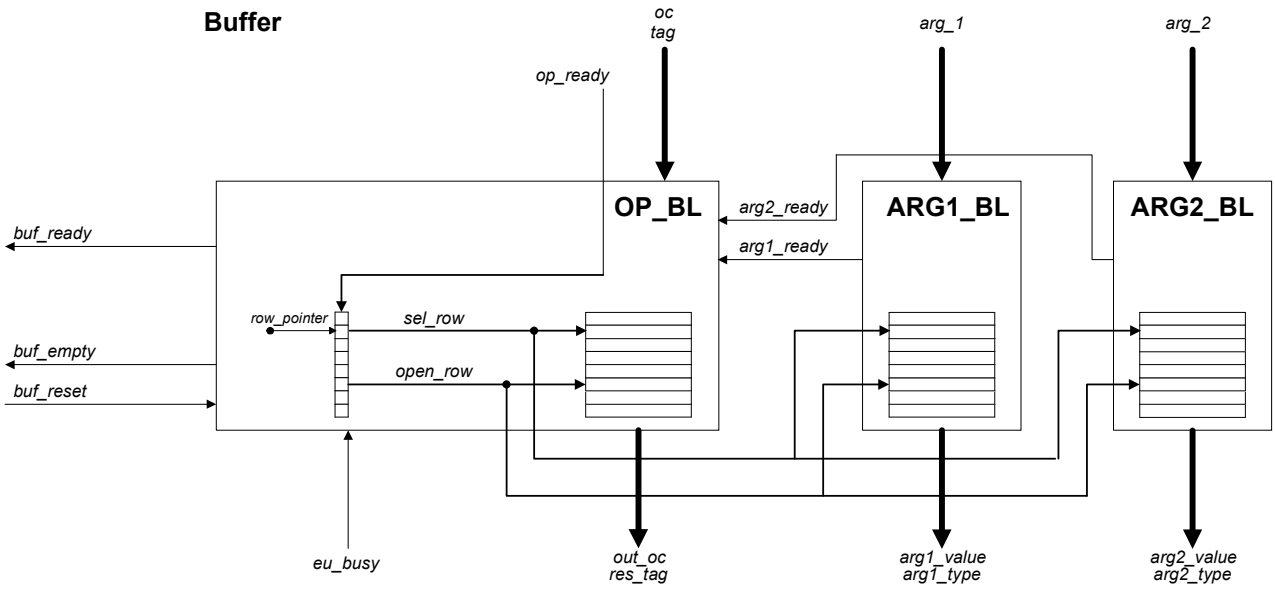


Рис.4. Основные интерфейсные сигналы в буфере

Аргументы могут быть получены из коммутатора (SB) или из устройства управления (CU). По приходу аргументов производится их запись в буфера хранения аргументов ARG1_BL или ARG2_BL. Запись аргументов сопровождается формированием сигналов *arg1_ready[i]='1'* или *arg2_ready[i] = '1'* и записью их в i-ой строке буфера операционной части.

После получения двух аргументов микрокоманда готова к исполнению.

При записи операционной части микрокоманды блок OP_BL устанавливает занятость строки *row_infill[row_num] = '1'*, а также формирует новое значение *row_num* из числа свободных строк и устанавливает *ready = '1'*. Если все строки заняты, то сигнал *ready = '0'*.

При получении всех необходимых аргументов для i-й строки буфер операционной части устанавливает сигнал *both_ready[i]='1'*. В общем случае возможно появление сигналов *both_ready* от нескольких строк сразу. Блок OP_BL, получив эти сигналы, выбирает одну команду и отдает её на исполнение в исполнительный модуль, сформировав сигнал *open_row[j],* где j- номер выбранной строки.

Сигнал сброса буфера *buf_reset='1'* устанавливает все значения сигналов *row_infill[i]* равными нулю.

Сигнал заполненности буфера *buf_empty='0'* означает, что в буфере нет команд. Если *empty='1'*, то в буфере есть хотя бы одна не выполненная команда

Блок арифметических и логический операций обеспечивает исполнение микрокоманд, выполняющих все операции сложения, вычитания, поиска экстремума, логические операции (OR, AND, NOT, XOR), операции сдвига и нормализации. Блок является комбинационной схемой, выполняющей данные операции. Данный блок не конвейеризирован и выполняет все операции за один такт.

Блок MUL выполняет операции умножения над дробными, дробными упакованными, знаковыми и беззнаковыми аргументами, а так же операцию умножения с накоплением суммы (MAC). При операциях с дробными аргументами производится округление полученного результата методом конвергентного округления.

Блок MUL синтезируется в зависимости от параметра *dw*, который может принимать два значения 16 и 24 – в зависимости от разрядности данных. Если *dw = 16* то в качестве умножителя используется блок MUL _16, когда *dw = 24* синтезируется MUL _24. В обоих блоках умножение построено по алгоритму Бута.

Конвейер умножителя состоит из двух ступеней, т.е. результат выдается на втором такте. Конвейер может быть отключен установкой параметра *mul_pipeline,* равным нулю.

Все полученные результаты, кроме результатов чтения, полученных при реализации косвенной адресации, рассылаются через коммутатор. Результаты чтения полученные при реализации косвенной адресации поступают непосредственно в буфер.

Доступ к коммутатору осуществляется блоком внутренней рассылки (ICU), который состоит из набора выходных регистров блоков исполнения и блока внешней рассылки (MUX_SB).

MUX_SB обеспечивает выбор наиболее приоритетного результата среди одновременно полученных и выдачу его в SB.

Результат выполнения микрокоманды каждого операционного устройства записывается в выходной регистр (OUTREG_xxx), где: xxx– это ASU, MUL, DMS. Выходной регистр сохраняет очередной полученный результат до выполнения рассылки, после чего регистр становится доступным для записи следующего результата. Время нахождения результата в выходном регистре определяется интенсивностью потока результатов, имеющих однотипную рассылку. Например, два одновременно пришедших результата, имеющих внешнюю рассылку, будут выданы в SB поочередно и, следовательно, один из них будет задержан.

Устройство доступа к памяти данных обеспечивает доступ к памяти данных, которая рассматривается как одномерный массив размерностью 2^{32} полуслов (от DM(0) до DM (2^{32}-1)). Слова, состоящие из двух полуслов, могут размещаться только с чётного адреса, следовательно, слово должно размещаться в памяти одного PU. Адресация DM для 4-х PU показана ниже:

PU0		PU1		PU2		PU3	
0...00	0...01	0...02	0...03	0...04	0...05	0...06	0...07
0...08	0...09	0...0A	0...0B	0...0C	0...0D	0...0E	0...0F
F...F8	F...F9	F...FA	F...FB	F...FC	F...FD	F...FE	F...FF

GPR – поле регистров общего назначения (РОН). Состоит из 16-ти регистров и рассматривается как одномерный массив от GPR(0) до GPR(15). Регистры имеют размер (dw*2)+4 бита. Из них dw*2 бита информационные, и 4 бита, определяющие тип данных содержащихся в информационных разрядах. У каждого регистра есть два теневых регистра той же размерности, которые используются для хранения описаний массивов и расчета индексов при доступе к элементам массива. В зависимости от команды регистр GPR может использоваться как промежуточная память, либо как индесный регистр.

Все команды всех процессорных устройств при декодировании имеют одновременный доступ на чтение ко всем регистрам GPR. Запись в регистры осуществляется также одновременно, после формирования признака «конец линейного участка».

4.5 Коммутатор

Устройство управления каждого процессорного блока, при формировании “заготовки” команды записываемой в буфер, формирует запрос к коммутатору на получение требуемых значений (аргументов операции). Если коммутатор не используется, то блок формирует нулевой запрос. Таким образом, в коммутаторе одновременно происходит два процесса:

- заполнение памяти запросов на аргументы (RFM) при декодировании очередной команды;
- отбор из входного массива результатов запрошенных результатов в качестве аргументов для выполнения уже декодированных команд.

Запрос включает:

- признаки необходимости получения результатов (*s1, s2*);
- номера процессорных блоков *(A1, A2)*, результаты которых запрошены;
- значение тегов команды, результаты которых запрашиваются *(log_tag1, log_tag2)*.

Значения признаков и адресов следующие:

- *s1='0'*, первый аргумент не используется (одноместная операция), либо поступает не из коммутатора;
- *s1='1'*, в качестве первого аргумента берётся результат операции, выполненной процессорным блоком с номером *A1*;
- *s2='0'*, второй аргумент не используется, либо поступает не из коммутатора;
- *s2='1'*, в качестве второго аргумента берётся результат операции, выполненной процессорным блоком с номером *A2*.

Если *s1=s2=0*, то запрос считается нулевым, иначе – ненулевой. Все запросы, поступившие в коммутатор, записываются в память признаков запросов.

Память признаков запросов (*RFM*) каждого функционального блока, как показано на рис.5, представляет собой матрицу двухбитных слов размером *n x l*, где: *n* – количество функциональных блоков; *l* – ширина окна выбранных команд (размер буфера).

Значения *RFM(i,j)* k-го функционального блока, где *k=0,1,..,n-1*, следующие:

RFM(i,j)="00" – результат *i*-того процессорного блока командой, у которой *log_tag1=j,* не используется;

RFM(i,j)="10" – результат i-того процессорного блока командой, у которой *log_tag1=j,* используется в качестве первого аргумента;

RFM(i,j)="01" – результат *i*-того процессорного блока командой, у которой *log_tag2=j*, используется в качестве второго аргумента;

RFM(i,j)="11" – результат *i*-того процессорного блока командой, у которой *log_tag2=j,* используется в качестве первого и второго аргумента.

Функционально коммутатор состоит из n идентичных коммутационных устройств, каждое из которых обслуживает один блок. Коммутационное устройство осуществляет отбор результатов для «своего» блока и имеет входы для приема *s1, s2, A1, A2, log_tag1, log_tag2, n* входов для приема результатов и два выхода для выдачи отобранных результатов в качестве первого и второго аргументов соответственно.

Каждый процессорный блок, выполнив команду, посылает результат её выполнения в коммутатор. Результат сопровождается тегом команды.

Каждое коммутационное устройство, получив результат от *i*-го процессорного блока, выбирает из своего RFM значение RFM (*i, res_tag*).

Если RFM(*i, res_tag*)≠"00", то полученный результат был запрошен в качестве аргумента(тов) для выполнения операции с меткой *log_tag1(2)=res_tag*. Если RFM(*i,res_tag*)="00", то этот результат в данном процессорном блоке не используется.

Таким образом, все запрошенные результаты отбираются из входного потока результатов и записываются в строку буфера (в поле аргументов), занимаемую командой с меткой *ins_tag=res_tag.*

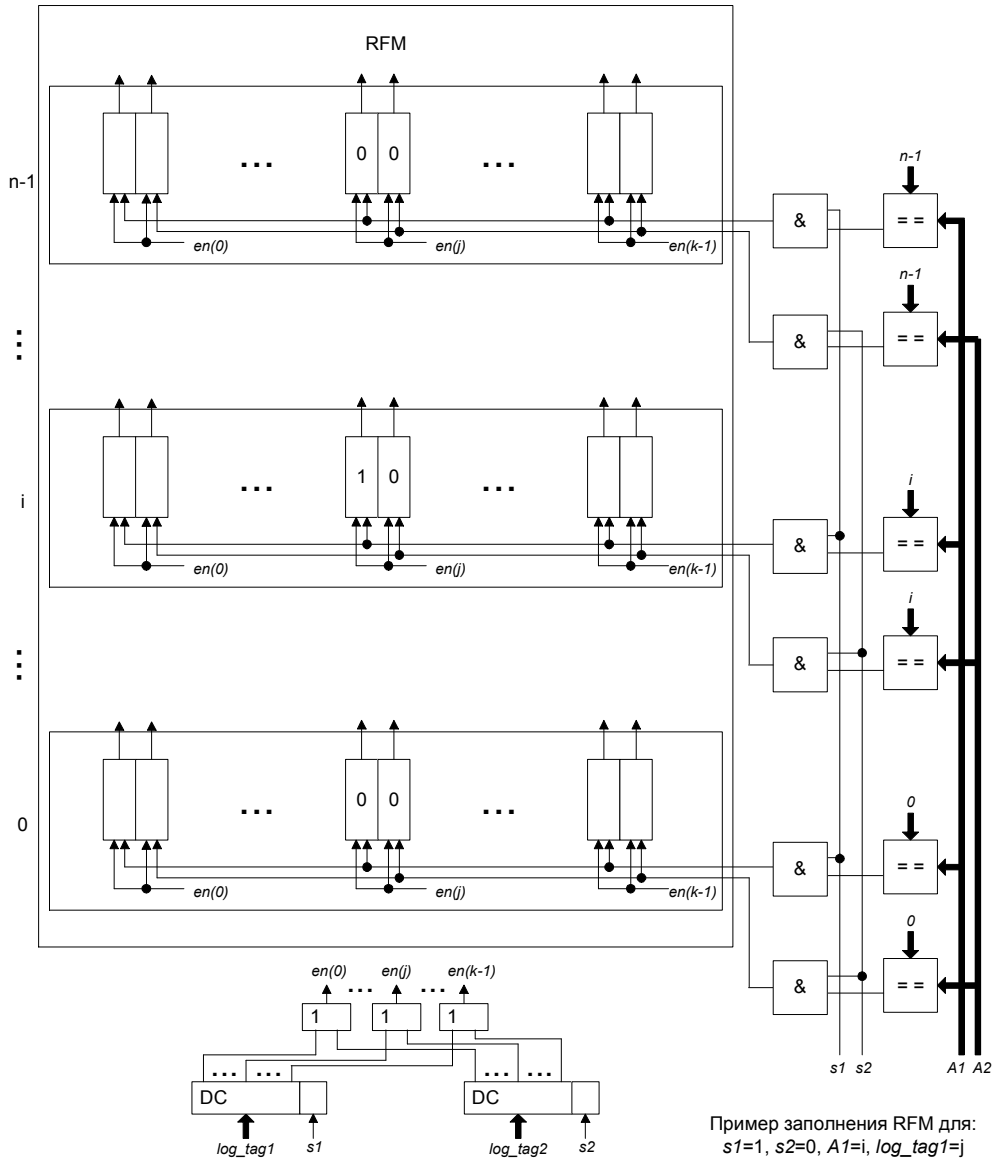


Рис.5. Структура RFM

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 канала используется буферная память (см. рис.6).

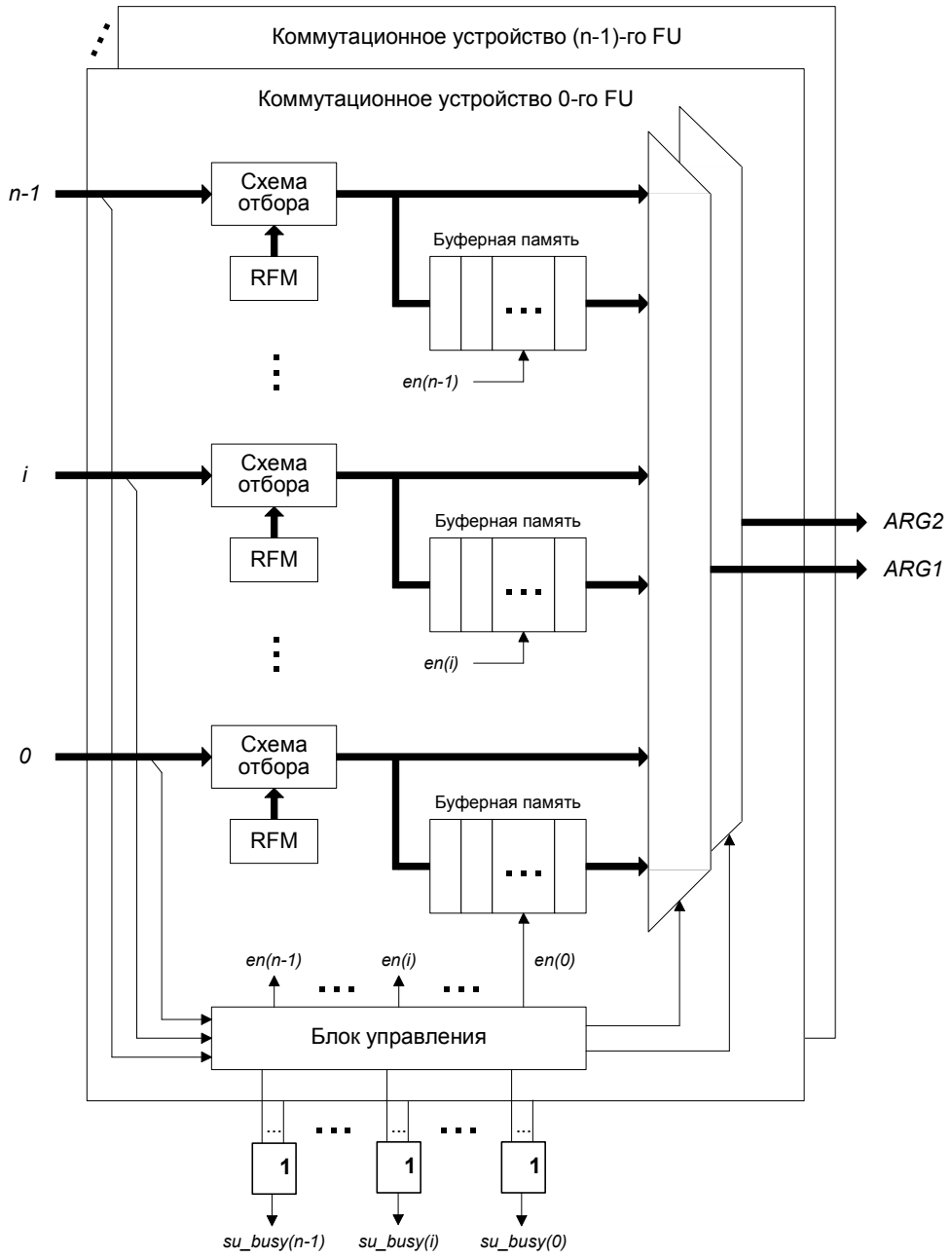


Рис.6. Буферизация потока отобранных результатов

В этом случае, при поступлении одновременно нескольких отобранных результатов, один или два результата сразу передаются в буфер команд, а остальные записываются в буферную память коммутационного устройства и передаются позже. Время задержки результата в буферной памяти определяется интенсивностью и шириной потока, а также принятой дисциплиной обслуживания результатов, находящихся в буферной памяти. При этом, вне зависимости от этих характеристик буферная память объемом (*l-1*) слов (для запоминания *l-1* результата) обеспечивает гарантированное сохранение отобранных результатов.

Если предположить, что результаты запрашиваются не каждый такт, а запрошенные более или менее равномерно распределяются в пространстве (по процессорным блокам) и во времени, то объём буферной памяти можно существенно сократить, обеспечив блокировку выдачи результатов в коммутатор тем блокам, в каналах которых переполнена буферная память. Обеспечивается это сигналами занятости каналов (*busy*), которые формируют все коммутационные устройства.

Так как коммутационных устройств *n* и каждое из них имеет *n* входных каналов, то сигналы *busy* образуют матрицу *n x n.* Если i-тый канал в j-ом коммутационном устройстве переполнен, то *busy(i,j)='1'*, иначе *busy(i,j)='0'*.

Сигналом, запрещающим *i*-тому процессорному блоку выдачу результатов в коммутатор, является логическая сумма всех *i*-тых сигналов занятости равная 1.

При параллельной обработке ключевыми факторами, определяющим ее эффективность, являются скорость и топология коммутационной среды. Топологические ограничения сужают круг эффективно решаемых задач. В коммутационной среде мультиклеточного процессора нет подобных ограничений. Она полносвязная, поэтому процессор эффективен при решении задач любой предметной области.

Объем аппаратных затрат на реализацию коммутатора для четырехклеточного процессора по результатам синтеза составляет 8,4% от общего объема процессорного ядра. Для 8-ми клеток 16,8%, для 16-ти 36,8%. Для 32-х и более клеток реализация полносвязного коммутатора неэффективна. В этом случае будет использоваться «несимметричный», частичносвязанный коммутатор, аппаратные затраты на реализацию которого имеют линейную зависимость. При этом, для сохранения производительности меняется логика выборки команд и логика выдачи результатов в коммутатор. Исследования в этом направлении проводятся в настоящее время.

В 8-ми клеточном варианте коммутатор организован как двухстадийная конвейерная система, обеспечивающая каждый такт прием одного результата от одной клетки, так и выдачу двух результатов. В 16-ти клеточном варианте коммутатор имеет трехстадийную конвейерную организацию. При этом, какого-либо заметного снижение скорости не происходит.

5. Производительность и энергопотребление

Оценка быстродействия и энергопотребления (см. таблицы) проводилась на задаче вычисления комплексного БПФ, как на одной из наиболее ресурсоемких задач цифровой обработки сигналов. При этом необходимо иметь в виду следующее:

1.Система команд мультиклеточного процессора включает команды комплексной арифметики, выполняемые за один такт.

2.Длительность решения задачи существенно зависит от того, как организован цикл, и сколько «бабочек» выполняется в теле цикла. Например, если в теле цикла выполняется одна бабочка, то для ее выполнения, в общем случае, необходимо два чтения из памяти (два отсчета), одно умножение на коэффициент (которое подразумевает и чтение данного коэффициента из памяти), одно сложение, одно вычитание и две записи в память. Итого 7 операций. Если не считать расходов на организацию цикла и, предполагая, что каждая операция задается одной командой, выбирается за один такт и исполняется также за один такт, общее время выполнения составит 128*8*7=7168 тактов. Если же в теле цикла выбирается четыре отсчета и выполняется четыре «бабочки» с сохранением промежуточных результатов в буферах, то время выполнения уже составит 64*4*20=5120 тактов. Для 8 точек (реальная тестовая программа) оно составит, соответственно, 32*40+32*2*52=4608 такта. Из этого количества можно исключить умножение на 0 в вырожденных «бабочках» (192 операции). Итого 4416 тактов. При сделанных допущениях и при возможности одновременной выборки и выполнения 4-х команд, время выполнения должно составить 1104 такта. Разница между расчетной оценкой и реально полученным значением – это накладные расходы на организацию цикла.

БПФ, несмотря на ее кажущуюся параллельность, распараллеливается не просто. TI C647, который имеет три таких ядра как у процессора TI C64xx-C. БПФ решает всего в 1,5 раза быстрее.

Рост производительности мультиклеточной архитектуры, по сравнению с традиционными решениями, объясняется двумя факторами.

Во-первых, внеочередным исполнением команд. Как известно, внеочередное исполнение команд дает рост быстродействия в 1,5-1,8 раз. В VLIW архитектурах этот метод практически не используется, так как планирование вычислений осуществляется статически компилятором

Во-вторых асинхронностью и коммутационной средой. В мультиклеточном процессоре обмен данными не требует приостановки процессов, протекающих в других блоках, ни для передачи ни для приема данных. Он осуществляется на фоне работы других блоков. Исполнительное устройство, выдающее результат в коммутатор, не знает кто потребитель данного результата. Выдав результат оно приступает к исполнению очередной операции. Коммутатор, получив результат, в режиме широковещательной рассылки выдает его всем потребителям, ранее выставившим заявки на этот результат. Запрошенный результат принимает буферная часть, хранящая аргументы. В этот момент, управляющая часть буфера может выдавать на исполнение ранее сформированные команды. Все эти процессы протекают параллельно, одновременно и асинхронно. В результате, время исполнения программы определяется временем работы самого загруженного блока. Как правило, это блок загрузки команд.

Параллельность и асинхронность работы блоков процессорного элемента (клетки) и коммутатора, обеспечивают возможность практически пропорционального роста производительности мультиклеточного процессора при увеличении количества клеток. Так, выполнение CFFT- 256 8-ми клеточным процессором составило 639 тактов (рост в 1,87 раза по сравнению с 4-х клеточным процессором), а 16-ти клеточным - 338 тактов (рост в 3,5 раза). Отличие в быстродействии (от теоретических значений) объясняется наличием постоянной составляющей на организацию циклов.

Таблица - Производительность

	MCP-1.xx-4	MCP-1.xx-8	MCP-1.xx-16	TI C64xx-C*	TMS320C647x**
Количество операций выбираемых и исполняемых за один такт	**4**	**8**	**16**	**8**	**24(3*8)**
CFFT-256(такты)	**1192** (radix-2)	**639** (radix-2)	**338** (radix-2)	**1246** (radix-4)	**806** (radix-4)

* Одноядерный процессор. Ядро типа C64xx имеет VLIW архитектуру. Командное слово содержит 8 полей для задания 8 операций, которые могут исполняться параллельно [2].

**Процессор содержит три ядра типа C64xx [3].

Таблица - Энергопотребление

Характеристики	Размерность	MCP-1.xx-4			TMS320VC5504***
		Результат синтеза	Расчетное значение*	Прогноз**	
Топологическая норма	μm	**0,18**	**0,13**	**0,13**	**0,09**
Напряжение	V	**1,8**	**1,2**	**1,2**	**1,05**
Энергопотребление на задаче CFFT-256	μW/Mhz	**426**	**98,6**	**54,6**	**-**
	μW/MIPS	**106,5**	**24,6**	**13,64**	**-**
Энергопотребление на смеси 75%DMAC+25%ADD (Typical Sine Wave Data Switching)	μW/Mhz	**319,5**	**73,9**	**39,4**	**150**
	μW/MIPS	**79,9**	**18,5**	**9,8**	**75**

* Расчетное значение учитывает только уменьшение топологической нормы и напряжения питания.

** Прогноз учитывает уменьшение энергопотребления на 60% после оптимизации RTL-кода (полномасштабное введение следующих методов сокращения энергопотребления: «clock gating», «operand isolation for functional unit», «operand isolation for multiplexers», «latching of register addresses instructional decoder» [4].

*** Процессор TMS320VC5504*** анонсирован в августе 2009г. как процессор со сверхнизким энернгопотреблением [5].

5. Заключение

Многократное улучшение ключевых количественных характеристик, а также качественные отличия предлагаемой мультиклеточной архитектуры от известных моделей – свидетельствуют о принципиальной новизне предлагаемого решения. Создание мультиклеточной архитектуры решает целый ряд задач компьютерной индустрии, которые принципиально не могут быть решены в рамках традиционной фон-неймановской модели или других известных моделей. Это уменьшение сложности и стоимости проектирования процессора, при одновременном повышении его технических характеристик. Повышение языкового уровня процессора и, как следствие, сокращение затрат на создание как компиляторов, так и программного обеспечения в целом. Поддержка дефектоустойчивости в производстве и постепенной деградации в эксплуатации, эффективное динамическое реконфигурирование процессора.

Разноплановость и качественный состав преимуществ предлагаемой архитектуры, позволяют позиционировать ее как принципиально новое и высокоэффективное пост-неймановское направление развития микропроцессорной техники.

Литература

1. Стрельцов Н.В. Организация мультиклеточной обработки // Труды IV Международной научной конференции «Параллельные вычисления и задачи управления» - Москва, 27-29 октября 2008.
2. Buyer's Guide to DSP Processors, 2001 Edition (Berkeley Design Technology, Inc. (BDTI), стр. 645).
3. http://focus.ti.com/dsp/docs/dspplatformscontento.tsp?sectionId=2&familyId=1635&tabId=2432
4. http://www.retarget.com/resources/pdfs/goossens-ip07.pdf
5. http://focus.ti.com/lit/ds/symlink/tms320vc5504.pdf.