РЕАЛИЗАЦИЯ МУЛЬТИКЛЕТОЧНЫХ ПРОЦЕССОРОВ

НИКОЛАЙ СТРЕЛЬЦОВ, технический директор, ОАО «Мультиклет»

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 имеет ключевое значение не только для развития микроэлектронной техники, но и для выхода на лидирующие позиции в микропроцессорной технике. В статье рассматривается принципиально новая и высокоэффективная пост-неймановская архитектура.

ВВЕДЕНИЕ

Анализ тенденций в развитии микропроцессоров показывает, что в компьютерной индустрии создалась и 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.

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, с помощью потоковых и редукционных машин. В них информационные связи между командами задавались явно (непосредственно), а результат немедленно передавался потребителю. Все эти попытки были неудачны, т.к. требовали отказа от императивных языков программирования. Но непосредственные связи можно реализовать и другим способом, 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]. Фактически, система команд на базе триад является аппаратной реализацией входного языка программирования, сохраняющей не форму, а сущность выражений языка. Именно отказ от сохранения формы позволил впервые, за многолетнюю историю попыток аппаратной реализации языков высокого уровня, получить эффективное решение — мультиклеточную архитектуру.

Для подтверждения эффективности архитектуры были разработаны две модели процессорного ядра, предназначенные для решения задач управления и цифровой обработки сигналов в приложениях, требующих высокой производительности и минимального энергопотребления, соответственно.

На данный момент завершены НИОКР по созданию этих моделей. В результате:

- разработаны и отработаны RTL-описания процессорных ядер на 4, 8 и 16 клеток;
- проведена отработка на FPGA Virtex-2 4-клеточного высокопроизводительного процессорного ядра, и выполнен его синтез для техпроцесса 0,18 мкм, V = 1,8 В;
- проведена отработка на FPGA Spartan-6 4-клеточного процессорного ядра с низким энергопотреблением, и выполнен его синтез для техпроцесса 0,18 мкм, V = 1,8 В;
- по результатам синтеза получены оценочные характеристики по производительности и энергопотреблению;
- завершена отработка на FPGA модели мультиклеточного процессора с низким энергопотреблением.

Процессор включает 4-клеточное ядро MCc0402100000 со сверхнизким энергопотреблением. Тип данных — с фиксированной запятой; разрядность — 16/32 бита; объем ОЗУ — 32 Кбайт × 16; объем ПЗУ — 4 × 4 Кбайт × 32. Частота — 75 МГц, производительность — (16/32) — 600/300 MIPS на FPGA Spartan-6. В состав периферийных устройств входят: UART (асинхронный интерфейс приема-передачи); I^2C (последовательная шина данных); SPI (последовательный периферийный интерфейс); PWM (широтно-импульсный генератор); RTC (часы реального времени). Системная шина AMBA (Specification 2.0) состоит из высокопроизводительной шины AHB (Advanced High-Performance) и шины периферии APB (Advanced Peripheral Bus).

В состав комплекса программно-инструментальных средств входит:

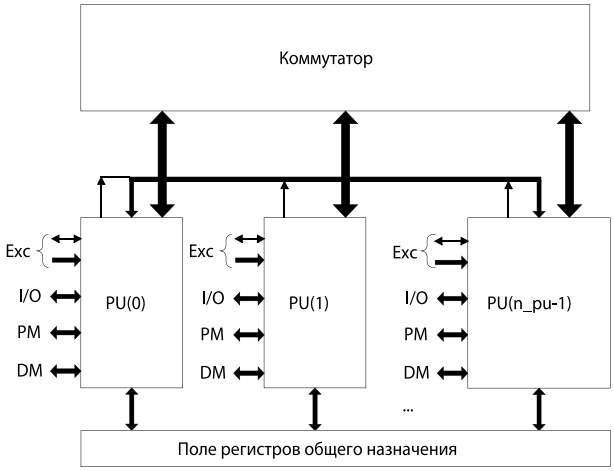
- Си-компилятор;
- ассемблер;
- загрузчик;
- отладчик на базе программной модели процессора.

Аппаратные отладочные средства построены на базе стандартных отладочных плат Xilinx. Загрузка программ и данных производится через последовательный порт UART.

РЕАЛИЗАЦИЯ МУЛЬТИКЛЕТОЧНОГО ЯДРА

Структура

Структурная схема ядра процессора представлена на рисунке 1.



Exc - внешние исключения
I/O - система ввода/вывода

Рис. 1. **Структурная схема процессора**

Ядро включает: *n_pu* идентичных процессорных блоков (клеток), имеющих нумерацию *0…* (*n_pu-1*); поле регистров общего назначения; коммутатор.

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

Тип данных задается в операциях чтения данных 4-битным кодом [1, 2]. Этот код сопровождает выбранные данные на всех последующих этапах их обработки с момента чтения из памяти и служит для контроля типов при выполнении операций.

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 допускается другое использование поля, например, в качестве 4-разрядного кода.

Формат AA определяет группу команд, использующих в качестве первого операнда адрес коммутатора, который формируется по значению ссылки на первый операнд, а в качестве второго операнда — либо адрес коммутатора, либо номер регистра общего назначения. Структура командного слова формата AA приведена ниже.

15 8	7 4	3 0
OC	F1	F2

где OC — код операции; F1 — ссылка на команду-источник первого операнда; F2 — поле второго операнда (ссылка на команду-источник второго операнда или номер регистра общего назначения).

Формат AV задает группу команд, использующих в качестве первого операнда также 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:

31 24	23 20	19 16	15 0
OC	F1	F2	V

где OC — код операции; F1 — ссылка на команду-источник первого операнда; F2 — поле, являющееся номером регистра общего назначения, используемого для определения второго операнда; V — 16-х разрядный код, используемый для определения второго операнда.

Состав команд приведен в [1, 2].

Процессорный блок

Процессорный блок включает:

- планировщик памяти программ (PMS);
- блок управления (CU);
- буферы хранения микрокоманд (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 рисунке 2.

Планировщик памяти программ

Планировщик памяти программ (PMS) 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 (PM(0)…PM (2^{32} – 1)). Нулевой элемент массива может содержать любую команду. Состоящие из двух полуслов команды могут размещаться с любого адреса.

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 команда формата AV полностью размещается в одном сегменте в ячейках с логическими адресами A, A+ n_pu и A+ 2*n_pu. Например, следующая последователь-

Таблица 1. **Размещение линейных участков в PM**

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

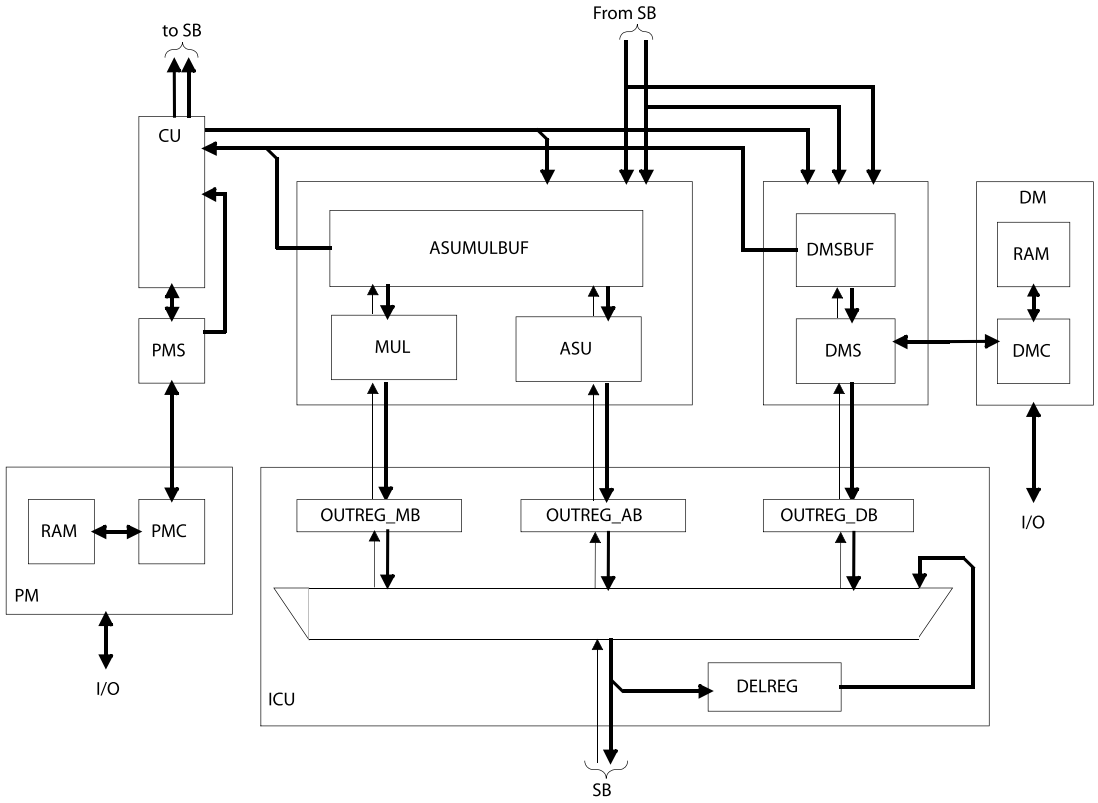


Рис. 2. **Структурная схема процессорного блока**

ность 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 увеличением насыщенности цветового тона (см. табл. 1).

Блок управления

Блок управления обеспечивает: управление процессом выборки и загрузки команд в регистр команд; декодирование команд, запись их микрокоманд в буферы 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 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.

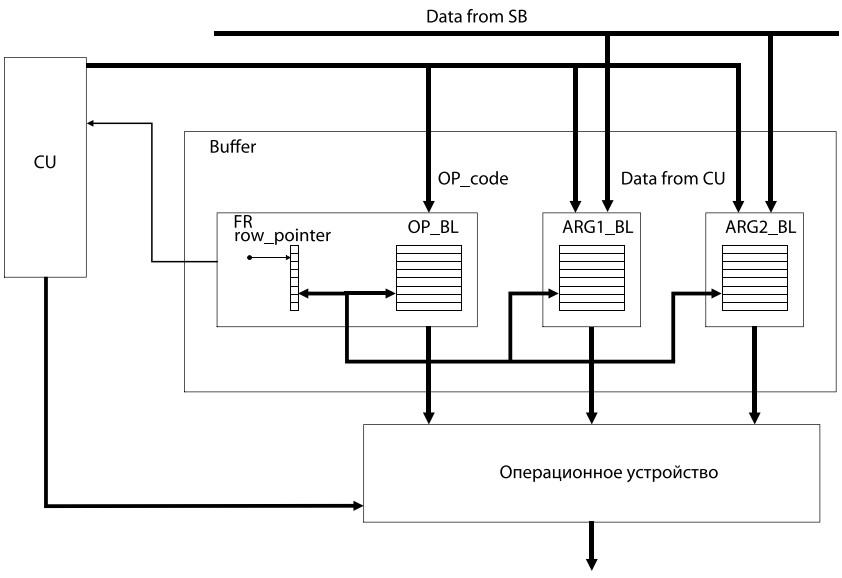


Рис. 3. **Структура буфера**

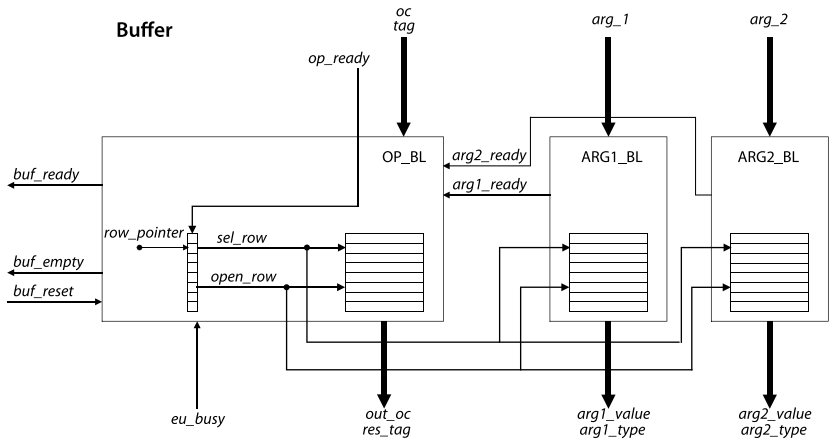


Рис. 4. **Основные интерфейсные сигналы в буфере**

Общие принципы построения буферов

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

Как показано на рисунке 3, буфер состоит из: буфера хранения операционной части микрокоманды OP_BL; буфера хранения аргументов ARG1(2)_BL.

Формирование команды включает три шага: запись операционной части команды в буфер; прием аргументов; выдачу команды на исполнение.

Операционная часть, кроме кода команды, заключает 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 рисунке 4.

Информация в буфере хранения операционной части показывает, какую операцию следует 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*). При этом формируется вектор *en_op*, информирующий блок FR о том, что в соответствующую строку произошла запись.

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.

Аргументы могут быть получены из коммутатора (SB) или из устройства управления (CU). По приходу аргументов производится их запись в буферы 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-й 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. Получив эти сигналы, блок OP_BL выбирает одну команду и отдает ее 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 заполнения буфера *buf_empty='0'* означает, что в буфере нет команд. Если *empty='1'*, то в буфере есть хотя бы одна невыполненная команда.

Блок арифметических и логических операций

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.

Умножитель

Блок MUL выполняет операции умножения над дробными, дробными упакованными, знаковыми и беззнаковыми числами. 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.

Блок внутренней рассылки

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, выдаются в SB поочередно и, следовательно, один из них задерживается.

Устройство доступа к памяти данных

Это устройство обеспечивает доступ к памяти данных, которая рассматривается как одномерный массив размерностью 2^{32} полуслов (DM(0)…DM (2^{32} – 1)). Слова, состоящие из двух полуслов, могут размещаться только с четного адреса, следовательно, слово должно размещаться в памяти одного PU.

Поле регистров общего назначения

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

Коммутатор

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

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

Формирование запросов на результаты

Запрос включает: признаки необходимости получения результатов *(s1, s2)*; номера функциональных блоков *(A1, A2)*, результаты которых запрошены; значение тегов команды, результаты которых запрашиваются *(loc_tag1, loc_tag2)*.

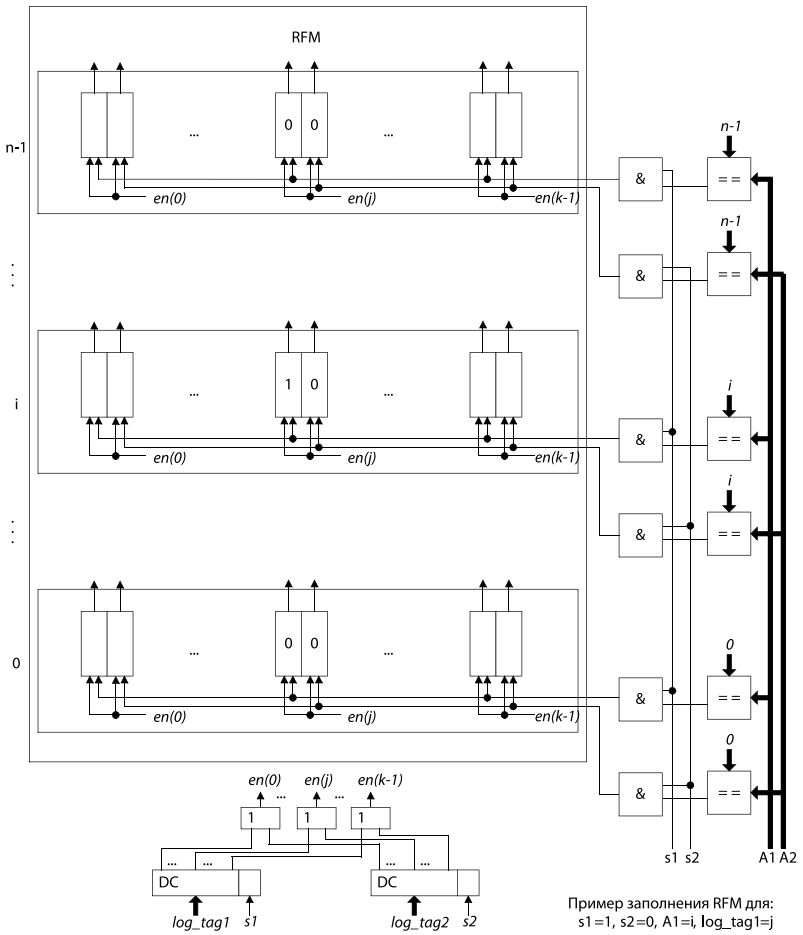


Рис. 5. **Структура RFM**

Значения признаков и адресов следующие:

- *s1='0'*, первый аргумент не используется (одноместная операция) либо поступает не из коммутатора;
- *s1='1'*, в качестве первого аргумента берется результат операции, выполненной функциональным блоком с номером A1;
- *s2='0'*, второй аргумент не используется либо поступает не из коммутатора;
- *s2='1'*, в качестве второго аргумента берется результат операции, выполненной функциональным 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 рисунке 5, представляет собой матрицу двухбитных слов размером $n \times l$, где n — количество функциональных блоков; l — ширина окна выбранных команд.

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

- *RFM(i,j)*="00" — результат *i*-го процессорного блока командой, у которой *ins_tag=j*, не используется;
- *RFM(i,j)*="10" — результат *i*-го процессорного блока командой, у которой *ins_tag=j*, используется в качестве первого аргумента;
- *RFM(i,j)*="01" — результат *i*-го процессорного блока командой, у которой *ins_tag=j*, используется в качестве второго аргумента;
- *RFM(i,j)*="11" — результат *i*-го процессорного блока командой, у которой *ins_tag=j*, используется в качестве первого и второго аргумента.

Отбор результатов

Структура коммутационного устройства является масштабируемой конструкцией и зависит от количества PU в процессорном ядре.

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, loc_tag1, loc_tag2, n* входов для приема результатов и два выхода для выда-

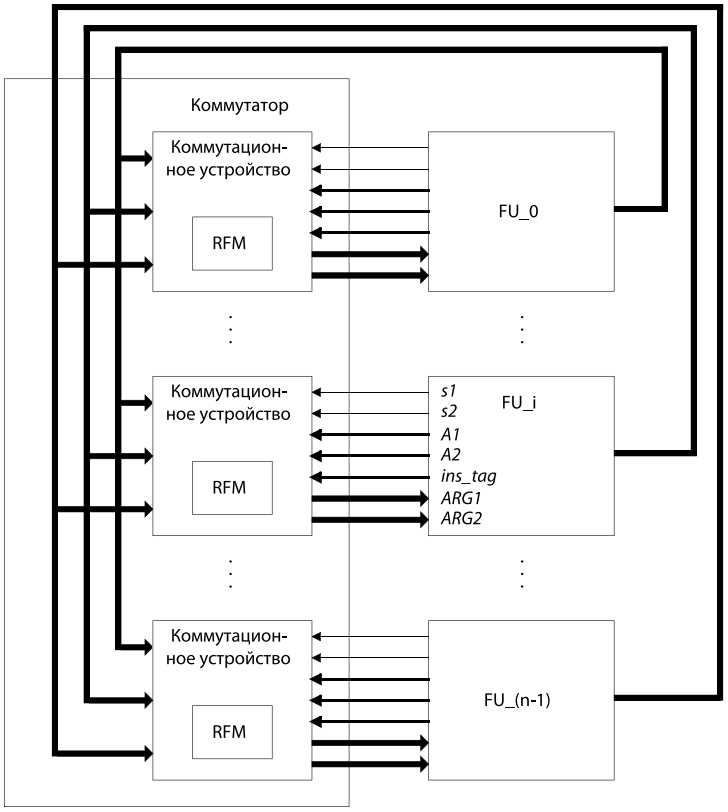


Рис. 6. **Структура коммутатора**

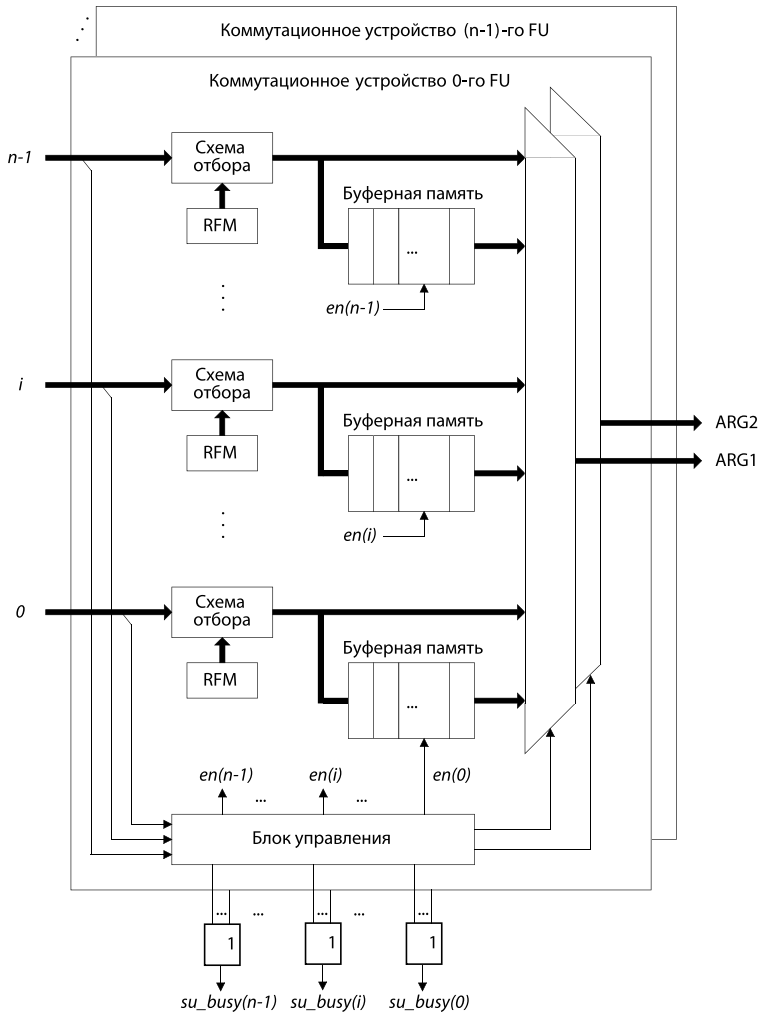


Рис. 7. **Буферизация потока отобранных результатов**

чи отобранных результатов в качестве первого и второго аргументов, соответственно (см. рис. 6).

Каждый процессорный блок, выполнив команду, посылает результат ее 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 *ins_tag=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*.

В общем случае, каждое коммутационное устройство может одновременно получить до *n* результатов, и все они могут быть отобраны. Очевидно, что реализация в буфере *n*-входовых полей аргументов для одновременной записи *n* отобранных результатов потребует неоправданно больших затрат кремниевых ресурсов. Ограничение количества входов и, соответственно, выходов из коммутационного устройства возможно лишь при использовании внутри каждого канала поступления отобранных результатов буферной памяти, парирующей неравномерность их поступления (см. рис. 7).

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 функциональным блокам) и во времени, то объем буферной памяти можно существенно сократить. В этом случае экономятся кремниевые ресурсы, уменьшается энергопотребление, но возникает принципиальная возмож-

ность переполнения буферной памяти в каком-либо канале. Следовательно, коммутационные устройства при переполнении должны обеспечивать блокировку выдачи результатов в коммутатор тем функциональным блокам, в каналах которых переполнена буферная память. Обеспечивается это сигналами занятости каналов (*busy*), которые формируют все коммутационные устройства.

Поскольку коммутационных устройств n и каждое из них имеет n входных каналов, то сигналы busy образуют матрицу $n \times n$. Если i-й канал в j-ом коммутационном устройстве переполнен, то *busy(i,j)='1'*, иначе *busy(i,j)='0'*.

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

ЗАКЛЮЧЕНИЕ

Создание мультиклеточных процессоров решает целый ряд проблем компьютерной индустрии, которые принципиально не могут быть решены в рамках фон-неймановской модели. В частности, проблемы сложности, «естественной» реализации параллелизма (без решения задачи распараллеливания), эффективного масштабирования процессора.

По сравнению с фон-неймановской моделью улучшаются и характеристики процессора. Оценки, полученные на DSP-задачах (FFT, FIR и др.) позволяют говорить о росте производительности в 2–4 раза и снижении энергопотребления в 10–15 раз) [2].

Система команд мультиклеточного процессора фактически является аппаратной реализацией входного языка программирования, опирающейся не на внешнюю форму, а на сущность языковых выражений.

В результате:

- сохраняется все программное обеспечение, созданное на традиционных императивных языках высокого уровня;
- процесс компиляции с языка высокого уровня ограничивается начальной машинно-независимой (front-end) фазой, что позволяет резко сократить затраты на разработку компиляторов;
- исчезает понятие «программирование на ассемблере», поскольку язык процессора не наглядный и поэтому «непрограммируемый». Программное обеспечение становится фактически машинно-независимым;
- основной формой существования программ становится исходный текст и, соответственно, все программы становятся открытыми. Исчезают возможности внедрения программных «закладок».

Архитектура мультиклеточного процессора обеспечивает принципиальную возможность использования процессорных кристаллов с производственными дефектами (что позволяет снизить их цену) и реализацию методологии постепенной деградации процессора при отказах его отдельных устройств. Процессор может быть работоспособен до тех пор, пока исправна хотя бы одна клетка. Следовательно, появляются новые возможности для построения сверхнадежных встраиваемых систем.

Разноплановость и качественный состав преимуществ предлагаемых архитектур позволяют позиционировать их как принципиально новое и высокоэффективное постнеймановское направление развития микропроцессорной техники.

ЛИТЕРАТУРА

1. Н.В. Стрельцов. Организация мультиклеточной обработки//Труды IV Международной научной конференции «Параллельные вычисления и задачи управления». Москва. 27–29 октября. 2008.

2. www.multiclet.com.