УДК 681.324-192

РЕКУРРЕНТНОЕ ОПЕРАЦИОННОЕ УСТРОЙСТВО ДЛЯ ПРОЦЕССОРОВ ОБРАБОТКИ СИГНАЛОВ

Степченков Ю.А., Петрухин В.С., Филин А.В.

1. Введение

Рекуррентное операционное устройство (РОУ), рассматриваемое в данной работе, является частью рекуррентного векторного процессора (РВП) обработки сигналов, основные принципы построения которого описаны в [1, 2]. Необходимость в экспериментальном варианте РВП возникла во время «примерки» рекуррентно-динамического подхода к выбору архитектуры цифрового вычислительного устройства обработки сигналов. По причине новизны идеи и отсутствия должных ресурсов охватить все уровни архитектуры РВП сразу не представлялось возможным. Пришлось решать задачу по этапам, разбивая её на части. Первый этап состоял в проверке идеи на практике. Поэтому он начался с разработки вычислительного процесса (ВП) нижнего (операционного) уровня, как самого плодотворного с позиции оценки эффективности идеи. На операционном уровне, как известно, можно достичь предельного распараллеливания ВП (уровень «мелкозернистого» параллелизма) и, следовательно, получить максимально-возможную производительность РВП в целом. Поскольку от результатов исследований вычислительного процесса на этом уровне зависела целесообразность проведения дальнейших этапов работы, операционный уровень стал основным центром приложения исследовательских усилий. Полученные знания о характере вычислительного процесса на операционном уровне были воплощены в модели РОУ, описываемой ниже. Учитывая исследовательский статус проекта, структурные схемы РОУ, приводимые в настоящем тексте, следует рассматривать не как структуры физических устройств, а как

модельные структуры, посредством которых объясняются принципы организации и функционирования вариантов РОУ.

РОУ реализует операционный уровень архитектуры РВП, в основу которой положена принципиально новая организация процесса вычислений, базирующаяся на *рекуррентно-динамической парадигме*, отличной от общеизвестных вычислительных парадигм (фон-неймановской *Control flow*, потоковой *Data flow*, нейровычислений, ассоциативных вычислений и других, находящихся на стадии исследования). Суть этой парадигмы – в рекуррентном способе представления управляющей (в основном) и обрабатываемой (частично) информации. В силу новизны заложенных в парадигме (и соответственно, архитектуре РВП) идей их отработку можно считать условно завершённой пока только на *операционном уровне*. В дальнейшем, в случае успеха, предполагается распространение их на остальные уровни (процедурный, диспетчерский, ввода-вывода) [2]. Поскольку РВП существенно отличается от современных процессоров ЦОС (DSP), его РОУ обладает рядом свойств и характеристик, отличающих его от других операционных устройств. Принятая парадигма - новая математическая платформа вычислений - распространяется в полном объёме пока только на нижний уровень архитектуры РВП, который характеризуется**:**

- *интеллектуальным вычислительным процессом рекуррентно-динамического типа;*

- *единым потоком активных* (самодостаточных) *данных на входе и потоком наборов данных* (в том числе активных) *на выходе*;

- *одноуровневой многосекционной организацией операционного устройства*;

- *интеллектуальной программной поддержкой* (накачкой самодостаточными данными) *со стороны процедурного уровня.*

Архитектура РОУ изначально разрабатывается как специализированная, предназначаемая для реализации *параллельных вычислительных процессов обработки сигналов в реальном времени*. Её

можно рассматривать как развитие известного *Data flow*–подхода, но построенной на серии других отправных принципов. К этим взаимосвязанным принципам РОУ (подробно рассмотренным в [1] применительно к РВП в целом; поэтому ограничимся их выборочным перечислением) относятся**:**

а) *рекуррентная реализация всего, что имеет отношение к ВП*;

б) *повышение статуса операндов и управляющих структур до уровня самодостаточных единиц*;

в) *использование событийной схемы взаимодействия объектов и процессов*;

г) *использование векторизации данных для организации естественно-масштабируемого параллелизма*.

Архитектура РОУ требует, чтобы операнды, поступающие на его вход, представляли собой активные элементы данных (в дальнейшем – *элементы самодостаточных данных* (ЭСД) [1, 2]), каждый из которых способен нести в себе рекуррентно свёрнутый алгоритм решения конкретной задачи. Активность обеспечивается наличием в формате их слова набора «функциональных управляющих структур» (рекуррентно закодированных), содержащих всю необходимую и достаточную информацию для самоуправления процессом развёртки и реализации алгоритма задачи.

2. Основные термины и определения

Некоторые из терминов, использующихся в данной работе, определены в двух сопутствующих статьях настоящего сборника [1, 2]. Остальные определяются ниже.

Ареал – подмножество однотипных (в функциональном плане) аппаратных средств, объединённых в единое целое (ареал) общностью решаемой задачи – обработкой ЭСД. Спецификация **а**рхитектуры операционного уровня определяет обрабатывающий ареал. В ареалах операционного уровня допускаются подареалы. По вертикали ареал собирается из вертикально компонуемых аппаратных секций (слайсов).

Горсть - группа из расположенных линейно (в строку) ЭСД, совместно перемещающихся в пределах вертикальных секций ареала операционного уровня**;** при этом каждый из операндов обрабатывается отведённой ему секцией. В горсть может объединяться некоторое количество смежных компонентов векторов или произвольных независимых ЭСД. Количество ЭСД в горсти может быть в диапазоне от 0 (пустая горсть) до числа, определяющего максимально возможную степень параллелизма аппаратных средств ареала (полная горсть).

Операнд – компонент слова ЭСД; это то, что поступает на вход арифметико-логического устройства РОУ.

Преобразователь тегов (ПТ) - комбинационное устройство, выполняющее требуемую модификацию (рекуррентную развертку) функциональных полей (тегов) промежуточных и результирующих операндов (на выходе из секции). В зависимости от назначения ПТ может иметь средства настройки параметров конфигурации перед разворачиванием необходимого граф-алгоритма.

Репреобразователь тегов (РПТ) - функционально не отличается от преобразователя тегов, но выполняет (при необходимости) преобразование тегов входных и промежуточных операндов на входе в секцию.

Секция (слайс) - один из параллельных трактов данных, обрабатывающий пару ЭСД (при бинарной операции) или один ЭСД (при унарной операции).

Селектирующая среда – понятие, обобщающее аппаратные средства, которые осуществляют выбор пар операндов для выполнения текущего шага обработки.

Степень параллелизма (СП) - количество секций РОУ, способных функционировать в параллель. Предполагается, что значение СП - не менее 2 и не боле 256.

Тип данных – идентификатор (статическая часть ЭСД), определяющий один из четырех базовых форматов ЭСД. Не подвергается рекуррентной развертке в устройствах ПТ.

Цикл - часть процесса вычислений, интервал, необходимый операнду для прохождения через пути данных секции.

Шаг - одна часть цикла, интервал, необходимый операнду для прохождения через одну ступень.

Другие локальные термины будут объяснены при рассмотрении структур РОУ.

3. Обзор работ по рекуррентному подходу

Нам известны лишь 5 статей, в которых анализируется возможность организации рекуррентного процесса вычислений [3-6]. В них разработано определенное представление о возможном облике вычислительных средств на базе рекуррентного подхода. Глубина разработки идеи в этих статьях не выходит за пределы концептуального уровня. Содержание статей во многом повторяется. В качестве обоснования целесообразности перехода от общепринятых методов организации вычислительных процессов к альтернативному рекуррентно-динамическому (РД) методу приводится, в основном, интуитивная аргументация. В частности, в [5] констатируется, что «*рассматриваемый концептуальный базис не подкрепляется никакими обоснованиями и доказательствами его преимуществ. Данная концептуальная основа имеет статус постулатов и может быть принята только интуитивно, на веру* (*либо вообще может не приниматься без всяких обоснований*)». Со своей стороны заметим, что сказанное *не является свидетельством слабости* предлагаемого подхода, а лишь констатирует уровень его понимания авторами этих работ.

Очевидно, что концептуальный уровень знаний недостаточен для оценки эффективности любой новой идеи и требует дальнейших научных и инженерных исследований. Его эффективность должна быть подтверждена путём сравнения с известными способами организации ВП. Частично это делается в настоящей работе, а также в статьях [1, 2].

Кроме того, в работах [3-5] ряд утверждений нельзя считать корректными. Например, утверждается, что рекуррентный подход, в отличие от классического, базируется на отказе от необходимого

принципа упреждающей подготовки трассы ВП в виде множества инструкций. В частности, в [3] декларируется, что «*при этом трасса … может наблюдаться, но нет необходимости в её фиксации, а тем более в опережающей подготовке*». К сожалению, обе составляющие этого утверждения неправильны и способны, в известной степени, дискредитировать саму идею. Дело в том, что и принцип опережающей подготовки трассы (собственно программа), и необходимость ее фиксации остаются в силе. Изменяются лишь способ и ресурсы, необходимые для ее подготовки и фиксации. Однако именно эти видоизмененные способы программирования потенциально и на практике являются источником явных преимуществ данного вида организации ВП. Есть в упомянутых статьях и другие неточности.

Следует особо оговорить ещё один некорректный момент в работе [4], связанный с неправильным толкованием понятия «*элемент активных данных*». В указанной работе активные данные обозначаются термином «*самоопределяемые данные*». Здесь сразу две неточности. Во-первых, «самоопределяемые данные» на самом деле должны называться «самоопределяющимися данными» - на это указывает приставка «само…». Во-вторых, смысл у термина «самоопределяющийся» совсем не тот, что подразумевается авторами. Напомним, что «*самоопределяемость - свойство активного объекта, обеспечивающее ему возможность по собственному выбору контролировать окружающую среду и понимать, что последняя, в свою очередь, может контролировать его*». Понятийный смысл термина «самоопределяемость» не из рассматриваемой проблемы, поэтому не совпадает с контекстом статей. Сути вопроса точно соответствует термин «*самодостаточность*». В данной работе, а также в [1, 2] элементы активных данных являются *самодостаточными* (ЭСД). Понятийный смысл термина «ЭСД» дан в работе [2], печатающейся в настоящем сборнике.

Специфику предлагаемой архитектуры позволит понять материал *Приложения 1*.

4. Цели работы

Настоящая работа обобщает результаты исследований вариантов организации РОУ для процессоров обработки сигналов, полученных с помощью программной модели «ОПЕРА», специально разработанной для этой цели. Эта модель была оптимизирована на изучение сразу четырёх альтернативных вариантов структур РОУ. Все структуры должны были строиться по секционному принципу, то есть обладать свойством масштабируемости (возможностью наращивания числа параллельно работающих секций РОУ в широких пределах – от 2 до 256). Структура всех моделировавшихся вариантов РОУ - одноуровневая (однослойная), с одним операционным устройством в каждой секции. Первоначальная мнемоника этой архитектуры – РАМС (рекуррентная архитектура с множеством секций). Информация о других уровнях архитектуры РВП содержится в [2].

Изначально была поставлена задача создания модели, с помощью которой можно было бы сделать обоснованный выбор архитектуры РОУ, сравнивая показатели качества моделируемых вариантов реализации и, таким образом, оценивая действительную эффективность рекуррентно-динамической парадигмы вычислений. В качестве основного критерия выбора рабочего варианта архитектуры РОУ (из числа альтернативных) было использовано отношение показателей "*стоимость/производительность*".

Нужно было также определить (в рамках РОУ, естественно):

- *оптимальный способ связи секционных обрабатывающих устройств в единое обрабатывающее поле*;

- *размеры тегов* (функциональных полей) *и их соотношение с размером полей операндов в словах ЭСД*;

- *объем памяти, требующейся для каждой секции*;

- *требования ко всем средствам связи и устройствам*.

Кроме того, к модели операционного уровня архитектуры РОУ был предъявлен ряд специфических требований. Она должна была

обладать определённой степенью гибкости: допускать варьирование степенью параллелизма (числом секций) в широких пределах, наращивать число функций эвольверного и арифметико-логических устройств, собирать данные для производства количественных оценок основных показателей качества структур РОУ: производительности, пропускной способности и других.

В рамках данной статьи будет рассмотрена не столько новая парадигма вычислений, сколько инженерно-технические аспекты одного из множества возможных вариантов реализации рекуррентно-динамической парадигмы вычислений.

5. Структурная организация РОУ

Были предложены и рассмотрены четыре альтернативных варианта реализации структуры операционного ареала РВП:

1) *слоевая* (с горизонтальной обработкой данных);
2) *распределенная* (с вертикальной обработкой данных по секциям);
3) *смешанная* (с горизонтально-вертикальной обработкой данных);
4) *централизованная* (с вертикальной обработкой данных и увеличенным числом модулей, общих для всех секций); финальный вариант структуры; в тексте обозначенный как *рабочий*.

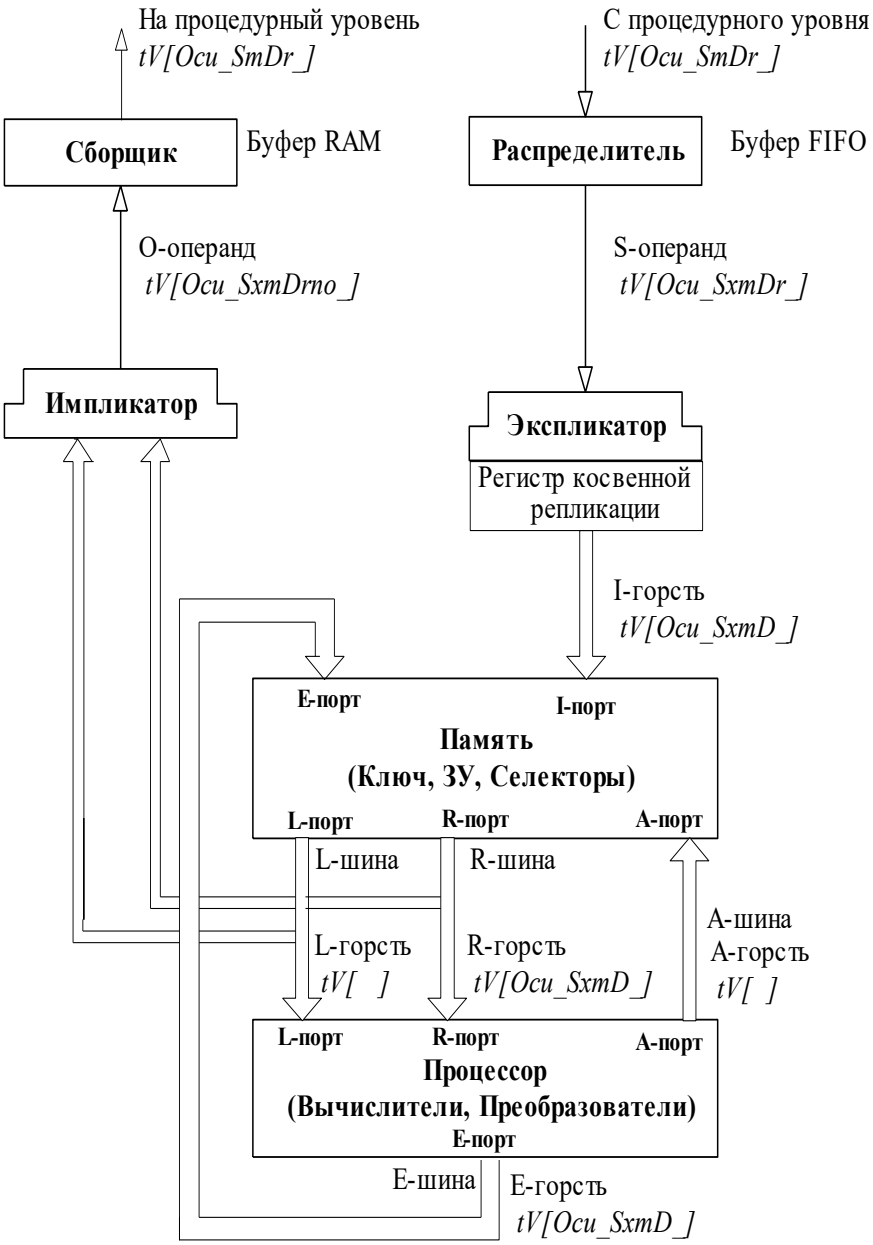
Все эти варианты объединяет *общий* принцип организации ВП на базе *самодостаточных данных* (обладающих способностью к саморазвёртке) - ключевой в общей философии построения РВП. Внешне главное различие между вариантами определяется тем, что именно контролируется компилятором при трансляции с внешнего языка программирования на внутренний (машинный) язык операционного уровня РВП.

5.1. Обобщенная модель РОУ

В результате изучения возможностей, достоинств и недостатков структур, перечисленных выше, а также решения задачи по их унификации, удалось создать *обобщённую модель* (ОМ) ареала РВП, пригодную для их исследования. Эта базовая модель получила название *обобщенной модели,* поскольку представлена в структурных терминах. Блок-схема обобщенной модели операционного уровня архитектуры приведена на рис. 1.

Общей средой ОМ, обеспечивающей возможность моделирования всех четырёх типов структур РОУ, является определённый набор программных модулей функциональных устройств и каналов связей: *распределитель* операндов (входной FIFO-буфер), *экспликатор* горстей операндов, *селектирующая среда* на базе памяти, *процессор* (обрабатывающее устройство), *импликатор* (формирователь выходных горстей операндов), *сборщик* наборов данных (выходной RAM-буфер), а также *шины* и *каналы связи*, объединяющие структурные компоненты в *операционный ареал* (ОА) РВП. Здесь курсивом выделены основные слова, использованные в структурных схемах. Одна обобщённая модель позволяет моделировать один вариант структуры РОУ в зависимости от того, какая структура исследуется.

Согласно [1], капсула может поставлять, а РОУ – воспринимать четыре типа операндов (табл. 1). Тип операнда указывается в обязательном для всех операндов поле [***t***].



Обозначения операндов: O– выходной;
S– исходный;
I– входящий;
L– левый;
R– правый;
A– аккумуляторный;
E- вычисленный (исполненный)

Рис. 1 . Обобщенная модель операционного уровня

Таблица 1. Типы операндов РОУ

t~поле		**Тип операнда**
Символ	**Код**	
E:	0	Пустой
A:	1	Вспомогательный
C:	2	Управляющий
D:	3	Содержательный

Только *содержательные* операнды содержат данные (отсюда ***D***-тип), подлежащие обработке в процессе вычислений. Этот тип операндов обрабатывается и воспринимается всеми модулями РОУ. Своеобразие вариантов структур РОУ наиболее наглядно проявляется именно при обработке содержательных операндов. Поэтому в дальнейшем (в рамках данной статьи) анализ вариантов РОУ будет ограничен этим типом операндов.

Пустой операнд (***E***-тип) не имеет структуры, не несет никакой информации и не обрабатывается ни одним элементом РОУ.

Вспомогательные операнды (***A***-тип) содержат служебную информацию, не оказывающую влияния на организацию и ход ВП. Минимальный набор вспомогательных операндов: *терминатор капсулы*, *терминатор данных*, *глобальный конфигуратор* (конфигуратор РОУ в целом), *инициализатор* и *шаблон.* Каждый из них имеет свой специфический формат и предназначен для настройки одного из блоков РОУ. Назначение вспомогательных операндов будет рассмотрено ниже (при описании обобщенной структуры РОУ).

Управляющие операнды (***C***-тип) управляют ходом ВП, координируют работу и настраивают некоторые модули структуры. Примеры операндов ***C***-типа: *конфигуратор отдельной секции* выполняет в необходимых случаях также функцию настройки преобразователей (ПТ) и репреобразователей (РПТ) тегов; *тормоз* выполняет функцию программного останова; *погонщик* реинициализирует ВП; есть еще операнды безусловного и условных переходов.

Функциональная часть всех типов операндов включает в себя три основных функциональных поля:

— поле *операции* [*O*];

— поле *селекции пары операндов* [*S*];

— поле *пересылки операнда* [*D*].

Эти функциональные поля подвергаются рекуррентной развертке в модулях ПТ и РПТ и содержат следующие подполя**:**

— подполе *операции Вычислителя* [*Oc*];

— подполе *режима использования операнда* [*Ou*];

— подполе *кода контекста* [*Sx*] (его источник - модуль «Распределитель»);

— подполе *кода совпадения* [*Sm*];

— подполе *маски репликации* [*Dr*].

Содержательные операнды имеют нотацию *tV*[*Ocu_Sm_Dr_*], где *V*- поле значений данных.

В приведённом примере нотации в квадратных скобках представлен минимальный набор тегов, встречающийся во всех четырёх структурах РОУ, а символ подчёркивания «_» в названии тегов указывает, что их состав может быть расширен.

5.2. Основные функции модулей обобщенной модели

Распределитель - представляет собой FIFO-буфер, получающий последовательности операндов, которые содержатся в капсулах, поступающих с процедурного уровня РВП. Распределитель является источником исходных S-операндов, которые появляются на его выходе в порядке их расположения в капсуле. Задача Распределителя - не только подача операндов на вход модуля «Экспликатор», но еще и анализ типа операнда на своём выходе (в верхней ячейке – вершине FIFO) и распределение некоторых типов вспомогательных и управляющих операндов (в отдельных вариантах РОУ) по их месту

назначения (на рис. 1 эти связи не показаны). Глубина буфера Распределителя в модели может варьироваться.

В РОУ каждая капсула, будучи самодостаточным объектом, обрабатывается с помощью своего собственного контекста. Этим приёмом достигается независимость капсул по данным. Контексты поддерживают конвейерную обработку капсул и различаются специальным атрибутом - *кодом контекста*. Этот атрибут назначается исходным операндам в Распределителе в соответствии с дисциплиной различения контекстов, суть которой состоит в следующем.

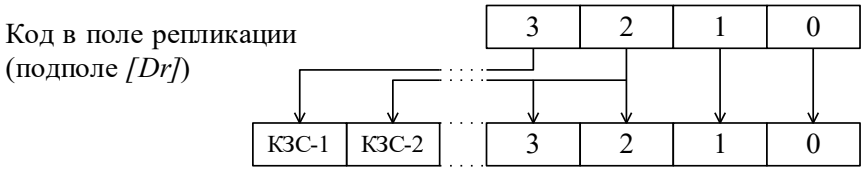
Распределитель снабжает каждый *S*-операнд текущим кодом контекста, помещая его в функциональное подполе [*Sx*]. При этом все исходные операнды капсулы получают один и тот же код. Затем каждый терминатор капсулы увеличивает текущий код контекста, изначально равный нулю, на величину пространства контекста, указанного в *Глобальном конфигураторе* по некоторому модулю, который должен гарантировать различимость, по меньшей мере, любых двух последовательных контекстов.

Таким образом, на выходе Распределителя содержательные операнды снабжаются нотацией *tV*[*Ocu_Sxm_Dr_*].

На операционном уровне некоторые компоненты архитектуры имеют дело не с операндами, а с горстями. Поэтому перед обработкой операнды из капсулы должны быть преобразованы в форму горстей. Это преобразование называется *экспликацией горстей* и выполняется *Экспликатором*. Экспликатор фактически является дополнительной ступенью Распределителя, подготавливающей горсти к выдаче в пути данных секций.

Экспликация является составной операцией, поскольку включает в себя *репликацию* (размножение или «векторизацию») *операнда* и собственно *экспликацию* (формирование горсти). Для репликации в функциональном подполе [*Dr*] всех исходных операндов предусмотрена явная *маска репликации*. Каждый разряд маски соответствует определенной секции. Занесение единицы в каждый из разрядов прямо

адресует секцию, в которую должен быть направлен операнд. *Маска прямой репликации* (МПР) инициируется ненулевым кодом репликации и прямо идентифицирует те секции, куда поступят операнды на обработку. Операнды с одной и той же маской репликации всегда направляются в одни и те же секции. Способ, при котором четырёхразрядный код репликации может быть использован для репликации операндов в любое число задействованных секций, иллюстрирует рис. 2.



КЗС- количество задействованных секций

Рис. 2. Способ задания маски репликации

Предусмотрены два режима образования МПР – стандартный и расширенный. В обобщенной модели используется только стандартный режим, а расширенный реализован только в централизованной структуре РОУ. Стандартный режим образования МПР не обеспечивает возможности индивидуальной репликации операндов в рамках секций, начиная со второй по КЗС-2 (связь, исходящая из разряда 2 на рис. 2). Операнд может либо реплицироваться во все эти секции (разряд 2 кода репликации содержит "1"), либо не реплицироваться ни в одну из них (разряд 2 кода репликации содержит "0").

Частично ограничения стандартного режима снимаются путём введения механизма *косвенной репликации*. Он включается, если у *S*-операнда нет ни одного указателя (нулевое значение подполя [*Dr*]). В Глобальном конфигураторе предусмотрены четыре вида косвенной репликации: «*гирлянда*» – для равномерного распределения элементов векторов по секциям; «*лавина*» – для прогрессирующей репликации (постепенного охвата всех задействованных секций); *горстевые сдвиги*

двух видов.

Подготовленная горсть (*I*-горсть) поступает в модуль «Память», имеющийся в каждой секции РОУ. Память представляет собой особую сравнивающую среду, содержащую запоминающее устройство (ЗУ), селекторы и переключатели. Всё это обеспечивает хранение в ЗУ исходных и промежуточных операндов, а также формирование и экспозицию (выдачу) пар операндов. Степень параллелизма РОУ определяет тот набор ЗУ, который способен запомнить сразу всю горсть операндов. В каждой секции РОУ имеется своё ЗУ.

Адресация и методы доступа к Памяти операционного уровня зависят от структуры операционного ареала РВП. Модуль «Память» способен выполнять следующие функции:

— *выбор и коммутацию запоминаемых горстей*;

— *хранение операндов*;

— *селекцию операндов, у которых совпали коды подполей* [*Sxm*], *и выдачу их в качестве L* (левого) *и R* (правого) *операндов на обработку в обрабатывающее устройство* (модуль «Процессор»);

— *формирование у R-операнда полного комплекта полей tV*[*Ocu_SxmD_*], *а у L-операнда - сокращенного комплекта tV*[ ].

Процессор предназначен, в основном, для выполнения операций над парами экспонированных горстей. Степень параллелизма структуры РОУ определяет в Процессоре число *процессоров-исполнителей*, способных работать в параллель и обрабатывать пару полных горстей сразу. Каждый Исполнитель является частью определенной секции. Любой модуль «Процессор» способен выполнять:

— *арифметико-логические функции над содержательной частью операндов, используя информацию, поступающую с L и R шин*: L**:** *tV*[ ] и R**:** *tV*[*Oc*];

— *рекуррентную развертку функциональных полей операндов, используя информацию, поступающую с R-шины:* [*Ocu_SxmD_*].

В каждом Исполнителе работают в паре модули «Вычислитель»

и «ПТ». Вычислитель содержит АЛУ, Сдвигатель и Аккумулятор. Преобразователь тегов служит для преобразования функциональных полей (*tV*[*Ocu_SxmD_*]) результирующих Е-операндов, появляющихся на выходе Процессора. Вспомогательная А-шина служит для быстрого привлечения содержимого Аккумулятора (А-операнда) к следующей операции (более короткий путь в селектирующей среде). Способы записи результата Вычислителя в Аккумулятор и использование содержимого последнего ареалозависимы (различны для разных вариантов РОУ).

Е-операнды возвращаются в Память (по шине А или Е) для дальнейшего участия в вычислениях (промежуточные результаты) или для вывода на процедурный уровень. Подготовка операндов для вывода включает в себя импликацию горстей и сборку набора данных (НД) – см. [1].

В рамках обобщенной модели РОУ предусмотрен единственный механизм вывода операндов – селекция со специальным вспомогательным операндом-шаблоном типа [*At*]. Отселектированная пара операндов с одинаковыми полями селекции [*Sxm*] - Шаблон (в качестве R-операнда) и операнд-результат (в качестве L-операнда) - появляются на выходе модуля «Память». Тип операнда (*t*-поле) одновременно анализируется модулями «Процессор» и «Импликатор». Отселектированная пара операндов поступает в Импликатор, который инициализируется Шаблоном, а Процессор на этом шаге ВП простаивает.

Импликатор отвечает за формирование горстей выходных операндов. Он получает операнды и выдает последовательность выходных операндов, используя Шаблоны для импликации. Импликатор выполняет соответствующую процедуру преобразования операндов-результатов в формат операндов в НД, аккумулируемых в модуле «Сборщик».

Нотация операнда-шаблона *At* на выходе Распределителя имеет структуру *t%Tcuxmr_no*[*Ou_SxmDr_*]. Структура вспомогательных

полей (после символа «%» - идентификатора вспомогательных полей) повторяет структуру функциональных полей содержательных операндов (*%Tcxmr)* с двумя новыми вспомогательными полями *%Tno*:

— [*%Tn*] - *подполе шаблонного номера выходного набора данных*;

— [*%To*] - *подполе шаблонного смещения в буфере выходного набора данных*.

Подполя [*%T*] ареалозависимы и отражают структуру всего набора функциональных полей содержательных операндов. Таким образом, шаблонные подполя повторяют (но не заменяют) аналогичные функциональные подполя.

Импликатор, обнаружив операнд для вывода на процедурный уровень архитектуры РВП, выполняет следующую процедуру:

- *формирует функциональные поля О-операнда, используя соответствующие вспомогательные* [*%T_*]-*подполя Ш*аблона;

- *присоединяет вспомогательные* [*%T_*]-*подполя Шаблона к функциональной части О-операнда в качестве* [*Dno*] - *подполей;*

- *последовательно выдает полученные О-операнды в модуль «Сборщик» с отбрасыванием Шаблонов*.

Модуль «*Сборщик наборов данных»* представляет собой активный буфер, формирующий НД. Сборщик получает последовательность выходных операндов, компонует выходные НД и отправляет их на процедурный уровень.

Каждый *О*-операнд из Импликатора записывается в буфер, указанный в функциональном поле [*Dn*] (в Сборщике предусмотрено два буфера), а внутри буфера – в ячейку, определяемую смещением в его функциональном подполе [*Do*]. Содержимое указателя-счетчика увеличивается на единицу, и следующий выходной результат помещается в следующую по номеру ячейку.

После окончания сборки НД отсылается наверх (на процедурный уровень). С момента, когда буфер заполнен до отказа или когда

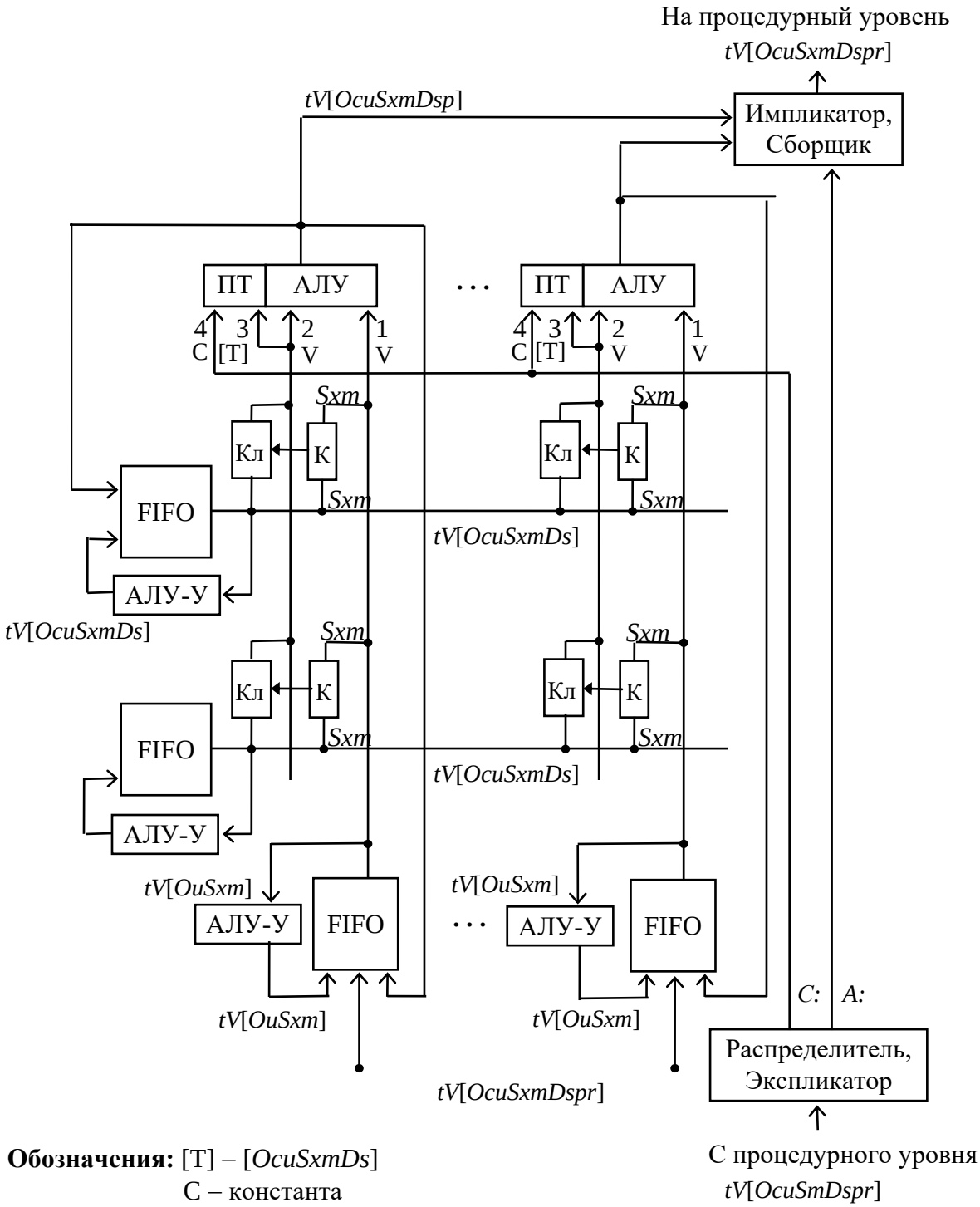
достигнут указанный при инициализации размер формируемых выходных данных, считается, что сборка окончена.

5.3. Слоевая структура РОУ

Слоевой подход характеризуется организацией ВП (в том числе и обмена данными) по горизонтали (то есть сопровождается *расслоением*) и предполагает, что компилятор должен следить за слоями этого процесса, отражаемыми в расположении операндов в блоках памяти с FIFLO-дисциплиной доступа (*First-In-First-or-Last-Out*). *Слой памяти* – множество операндов, необходимых для вычисления слоя процесса или его части.

Слоевой подход к организации ВП предполагает, естественно, ориентацию на слоевую структуру, для реализации которой в структуре слоевого операционного ареала должны присутствовать следующие составляющие, специфичные для данного подхода (рис. 3):

- *ортогональный селектор*, состоящий из:
  - ортогональной матрицы шин селекции данных – подполе [*Sxm*],
  - ортогональной матрицы шин данных – поле *V*,
  - матрицы компараторов для выявления пар операндов (К),
  - матрицы ключей (Кл);
- два набора *FIFLO-памятей – правый и левый*;
- два набора *канальных мультиплексоров* (на входах FIFO-памятей);
- два набора *унарных АЛУ* (АЛУ-У);
- специальная *шина* (*A*) - для передачи Шаблонов в Импликатор и настроечных параметров в Имликатор и Сборщик;
- специальная *шина* (*C*) - для передачи константы (С) - конфигурационного параметра - в преобразователь тегов.



Обозначения: [T] – [*OcuSxmDs*]
С – константа
К – компаратор
Кл – ключ
АЛУ-У – АЛУ для унарных операций

Рис. 3. Слоевой вариант операционного уровня

Функциональное назначение основных элементов структуры слоевого подхода соответствует обобщенной модели. Основное различие заключается в организации Памяти (Ключей, ЗУ и селектирующей среды).

Память - стековое (неадресуемое) ЗУ, использующее метод доступа FIFO+LIFO (комбинированный). Она делится на два блока, называемых *плечами*: левое (ЛПП) и правое плечо памяти (ППП).

При этом должны быть определены следующие процедуры:

- *распределение S-операндов между стеками ЛПП и ППП*;

- *селекция пар операндов для одной обрабатывающей секции*.

Нотации содержательных операндов в капсуле имеют вид *tV*[*OcuSmDspr*]. Как видим, добавлены два новых подполя [*D*] и *конкретизировано* подполе [*Ou*]:

- [*Dp*] - *подполе указателя плеча Памяти*;

- [*Ds*] - *подполе способа записи в Память*;

- [*Ou*] - *подполе режима использования операнда в Памяти*.

Двухразрядный указатель плеча памяти [*Dp*] определяет место назначения операнда – правое, левое или оба плеча. [*Ds*] кодирует способы записи операнда**:**

- *в «вершину» стека с замещением её содержимого* (с потерей текущего операнда);

- *в «вершину» стека с продвижением внутрь её содержимого* (текущий операнд сохраняется);

- *в «дно» стека*.

Подполе [*Ou*] определяет режим использования операнда, находящегося в вершине стека, после селекции (нахождения) второго операнда пары - *одно-* или *многократный*. При однократном режиме операнд удаляется из стека (указатель вершины будет идентифицировать следующий операнд), а при многократном – остается в вершине (верхней ячейке) стека.

Число секций каждого плеча Памяти определяет степень их параллелизма. Все секции Памяти идентичны. В исходном состоянии на

правые входы секционных АЛУ поступают операнды с вершин ППП. Селектор - ортогональная матрица компараторов и ключей, в которой входы каждого из компараторов Cij соединены с выходами ЛПП i-ой секции и ППП j-ой секции. Факт нахождения пары имеет место, если коды селекции (подполя [*Sxm*]) на двух входах одного компаратора совпадают. При этом открывается соответствующий ключ, и на левый вход АЛУ поступает второй операнд пары.

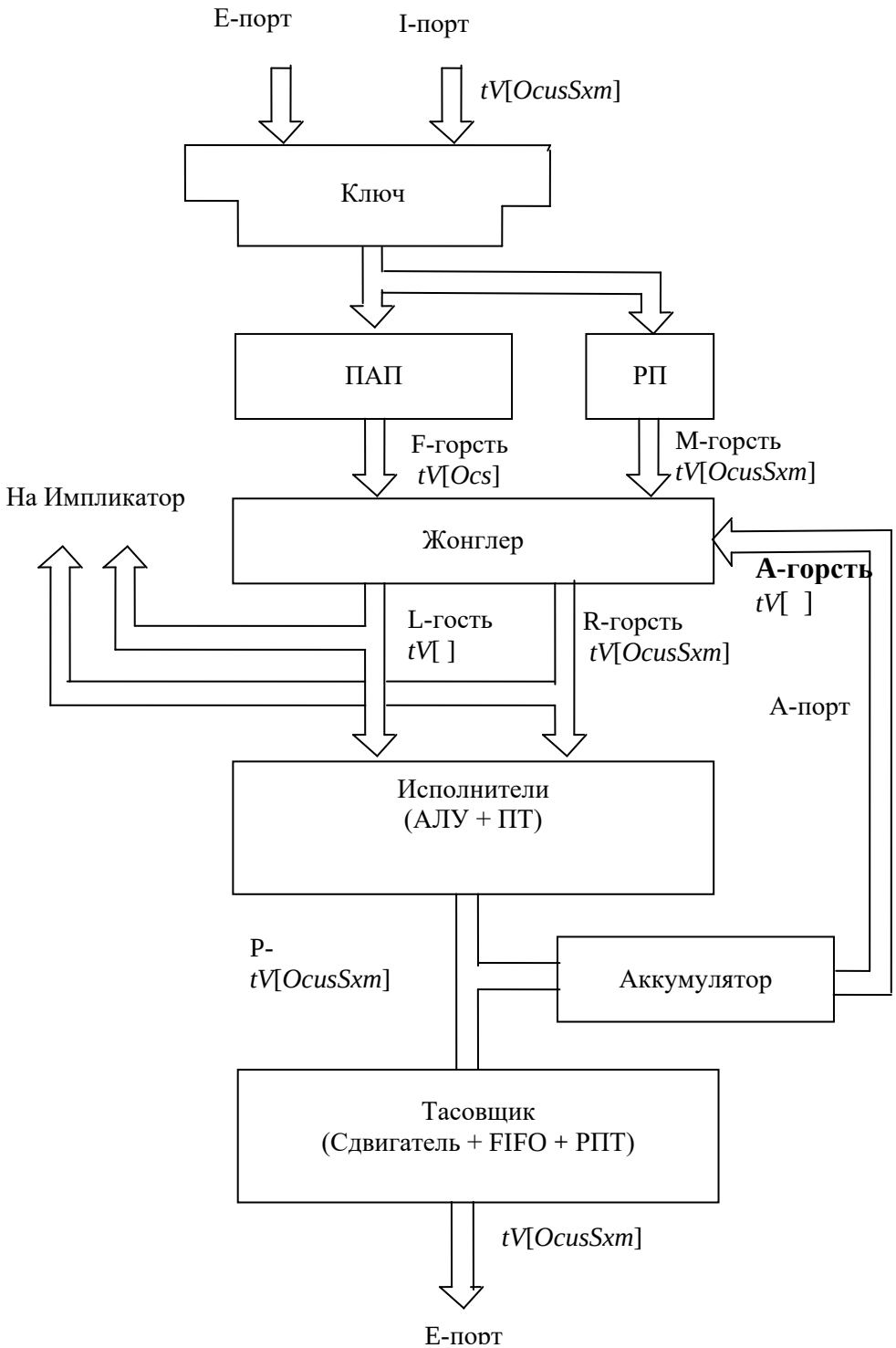
Принцип связи «каждый с каждым» наиболее существенен в слоевом подареале. Принципиально он обеспечивает возможность рассылки одного операнда из левого плеча (как исходного, так и промежуточного) в произвольные секции – в какую-либо одну, во все или в любое разумное их сочетание. При этом должна быть решена проблема единственности срабатывания компаратора в рамках одной вертикальной секции. Если на вершинах ЛПП будут операнды с одинаковыми [*Sxm*]-подполями, то возможно возникновение коллизий, которые придётся как-то разрешать.

Несмотря на такие мощные ресурсы селектирующей среды слоевой структуры РОУ, легкость и эффективность программирования уступают другим видам реализаций РОУ.

5.4. Распределенная структура

Предполагается, что компилятор при распределенном подходе должен отслеживать распределение операндов между вертикальными секциями. Данный вариант характеризует большая степень распределенной (в рамках отдельных секций) обработки по сравнению с централизованным вариантом РОУ. Структуру распределенного РОУ составляют следующие программные модули (рис. 4):

- *память адресной проверки* (ПАП) – секционная память совпадений;



Обозначения операндов: **F** – совпавший;
M – проверяющий;
P – процесорный

Рис. 4. Распределенный вариант операционного уровня

- *ключ или блок мультиплексоров* (на входах ПАП);

- *блок жонглирования операндами* (Жонглёр)*;*

- *блок тасования операндов.*

Ключ – набор мультиплексоров-демультиплексоров, выбирающих очередную горсть для направления операндов в Память, где производится сравнение кодов совпадения.

ПАП – ЗУ с произвольной выборкой и "расщепленным циклом", имеющее число секций (с одним входом и двумя выходами), равное степени параллелизма. Память имеет специальный разряд наличия данных: "*есть*/*нет*". В памяти нет явной операции чтения, но при любой записи сначала по адресу селекции ([*Sxm*]-поле) читается содержимое ячейки. Если она пуста (признак «нет»), то в нее записывается операнд, проверяющий пару (М-операнд), с установлением признака «есть». Если ячейка не пуста (считывается признак «есть»), то фиксируется наличие компонента пары в ПАП: М-операнд записывается в регистр памяти (РП), а совпавший (ждущий) операнд (F-операнд) считывается из ПАП, и пара операндов поступает на вход Жонглера.

Жонглер операндов – набор коммутаторов (коммутационная сеть) для пар операндов с совпавшими полями. *Жонглер* направляет операнды с совпавшими полями в *Процессор*.

Назначение Жонглера – обеспечить правильный приход операндов на входы L и R Исполнителя (например, вычитаемое и уменьшаемое, делитель и делимое). Во всех структурах РОУ имеется возможность привлечь в текущем цикле содержимое Аккумулятора. Если на вход Жонглера поступает управляющий операнд (Cac) – так называемый заменяемый операнд, то Жонглер замещает его операндом из Аккумулятора. При этом *A*-операнд (содержимое Аккумулятора*)* должен быть коммутирован на нужный выходной порт Жонглера.

Содержательные операнды в капсуле содержат нотации вида *tV*[*OcusSm*], то есть добавляется новое подполе [*Os*] и уточняется подполе [*Su*]:

– [*Os*] - *подполе операций* Тасовщика;

– [*Su*] - *подполе режима использования операнда в ПАП*.

Подполе [*Os*] определяет возможную задержку передачи *P*-операнда (операнда на выходе Процессора) через Тасовщик и направление передачи: в собственную секцию, в соседнюю или в обе одновременно. В этом (с позиции рассылки *E*-операндов) и заключается принципиальное отличие распределенного подхода от слоевого.

Подполе [*Su*] определяет условие селекции пары операндов и состояние ПАП после нее. Входящий в ПАП операнд может затереть находящийся там операнд (без образования пары) или образовать пару - без изменения состояния ячейки ПАП либо с ее замещением на входящий операнд.

Программируемость этого варианта исполнения РОУ существенно проще, чем слоевого. Однако относительное быстродействие реализации целого класса алгоритмов хуже, если необходим интенсивный вывод *O*-операндов на процедурный уровень либо произвольная рассылка *I*- или *E*-операндов между секциями.

5.5. Смешанная структура

При исследовании этого варианта структуры РОУ преследовались цели:

- *добиться максимума быстродействия в выполнении вычислительных процедур по сравнению с распределенным методом*;

- *существенно повысить гибкость формирования требуемой последовательности слоев по сравнению со слоевым подходом*;

- *уменьшить аппаратные затраты на реализацию*.

Смешанным этот метод назван потому, что компилятор должен следить не только за распределением операндов по слоям, но и отслеживать распределение операндов между вертикальными секциями.

Смешанный вариант структуры РОУ отличается от других вариантов, прежде всего, частичным отходом от принципа ЭСД в пользу автоматного принципа синхронизации процессов (перехода к рекуррентной развертке по жесткой программе). По существу все остальные отличия рассматриваемого подареала являются следствием использования автоматного принципа.

Коренное же отличие заключается в функционировании ПТ. В этом подходе рекуррентно разворачиваются не функциональные поля операндов, а состояния самого ПТ (выход ПТ непосредственно заходит на его вход). Будучи однажды инициирован из капсулы на реализацию процедуры, ПТ рекуррентно формирует последовательность своих состояний автономно и независимо от операндов на входе Процессора. Таким образом, возникает возможность отрыва большей части функциональных полей операнда от самого операнда (содержательной части слова данных). При этом по путям данных РОУ, кроме содержательной части операнда, передаются только [*t*]- и [*Dp*]-поля.

Функциональные поля локализованы в ПТ, хранятся там и рекуррентно разворачиваются. Совокупность функциональных полей операнда, поступившего на обработку в модуль «Вычислитель», в других вариантах структур РОУ, как и функциональных полей ПТ, эквивалентна.

5.6. Централизованная структура

Централизованный подход к выбору структуры РОУ это попытка объединить все лучшее, что есть в каждом из трёх рассмотренных подходов, и, тем самым, насколько возможно улучшить показатели качества функционирования рабочего варианта структуры РОУ. Этот подход характеризуется также введением ряда дополнительных модулей, общих для всех секций РОУ, и уменьшением доли распределенной обработки (обработки в рамках отдельных секций). Отсюда название структуры – централизованная.

Основные отличия данного подхода (см. рис. 5):

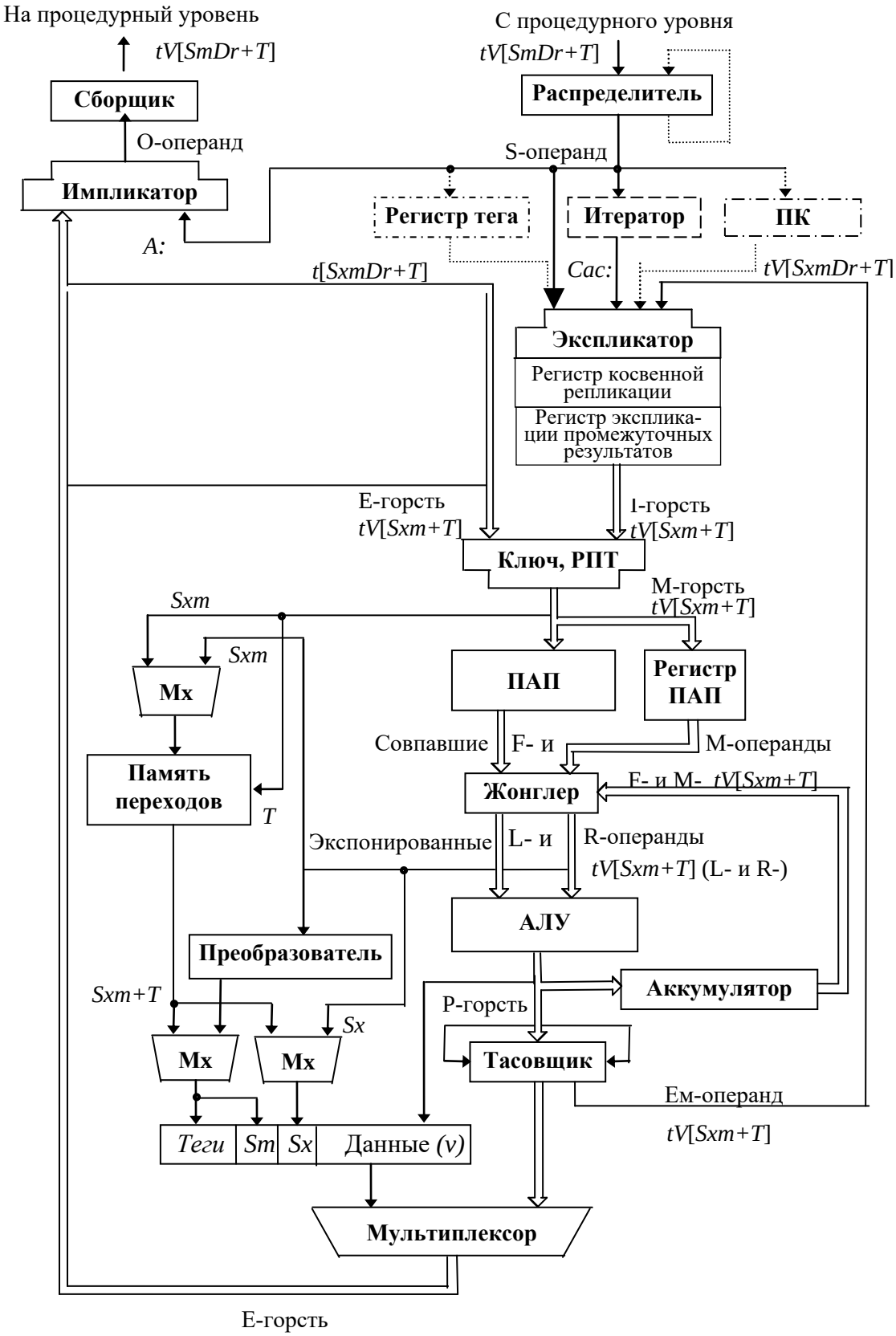
1) Выход Распределителя соединен одинарной шиной с одним из входов Импликатора (тиражируемого по числу секций элемента РОУ). В состав Импликатора вводятся регистры (их количество равно числу секций) для хранения Шаблонов. Поток Шаблонов и настроечных параметров для Импликатора и Сборщика в этом случае не мешает реализации основного ВП.

В каждом такте ВП Шаблон отслеживает состояние подполей [*Sxm*] каждого *E*-операнда. При совпадении их со значением аналогичных подполей Шаблона в Импликаторе последний направляет этот ***E***-операнд в Сборщик по процедуре, описанной выше. Таким образом, принцип передачи результатов по их готовности, используемый в DF/S-машинах, остается в силе. Общее число шин на кристалле при этом уменьшается, а, быстродействие увеличивается; суммарный же объем регистровой памяти остается тем же (Шаблон, ранее хранившийся в ПАП, теперь хранится в Импликаторе).

2) В состав модели централизованного варианта РОУ добавлен модуль «*Итератор*» – активный буфер заменяемых операндов (имитирует кольцевой буфер-счетчик FIFO). Это необязательный элемент архитектуры, предназначенный для приема и обработки *заменяемых* операндов типа *Cac* и позволяющий эффективно организовывать циклические процедуры, вовлекать константы в ВП и регулировать разгрузку капсулы. Итератор, получив один операнд из капсулы, может быть источником циклически повторяющихся операндов.

3) В состав Экспликатора введен регистр репликации входных *S*- и промежуточных *Em*-операндов, что позволяет реализовать расширенный режим МПР (см. раздел 5). Теперь операнды могут быть распределены между секциями в необходимой последовательности.

4) Дополнительную гибкость централизованной структуре придает введение РПТ, который подсоединяется параллельно ключу. Теперь, при необходимости, перед поступлением в ПАП теговые поля операндов могут быть подвергнуты рекуррентной развертке.



Необязательный элемент операционного уровня

Возможное развитие операционного уровня

Рис. 5. Централизованный вариант операционного уровня

5) В структуру вводится Память переходов, в которую перед инициацией ВП заносится набор теговых полей в соответствии с возможными ветвлениями в ходе ВП. Функции Жонглера существенно расширены, что повышает гибкость программирования в РОУ.

6) Регистры Тасовщика объединены в кольцо, что позволяет секциям эффективно обмениваться *E*-операндами, не занимая традиционных каналов обмена.

Нотации содержательных операндов в капсулах имеют вид *tV*[*OcuSmDrsea*], то есть появляются три новых [*D*]-подполя (функции остальных – те же, что в распределенном варианте):

— [*Ds*] - подполе передачи операнда между соседними секциями (во многом совпадает с [*Os*]-подполем в распределенном варианте);

— [*De*] - подполе передачи операнда на *Em*-шину;

— [*Da*] - подполе передачи операнда в Кольцо.

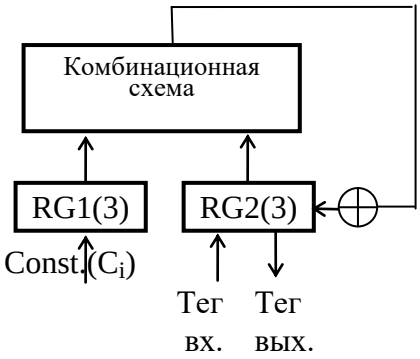
На рис. 5 изображено (пунктирно) предполагаемое развитие вертикального варианта РОУ, о чем будет упомянуто позже.

6. Преобразователь тегов

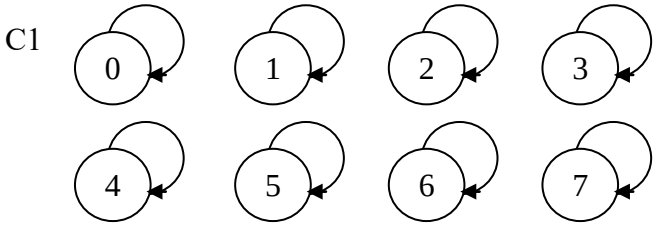
Как описывалось выше, ПТ представляет собой регистрово-комбинационную схему. Например, на рис. 6 представлена функциональная схема 3-разрядного преобразователя, которая на каждом шаге рекуррентной развертки выполняет следующую функцию:

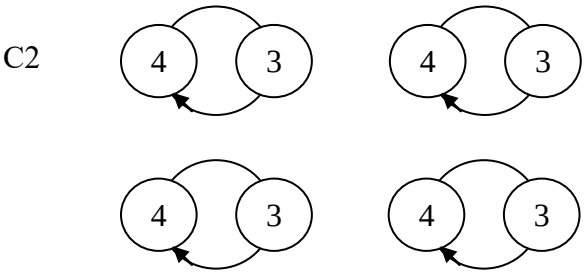
$$(RG_2)_{new} = \{[(RG_1) * (RG_2)_{old}] \rightarrow \} \oplus (RG_2)_{old,}$$

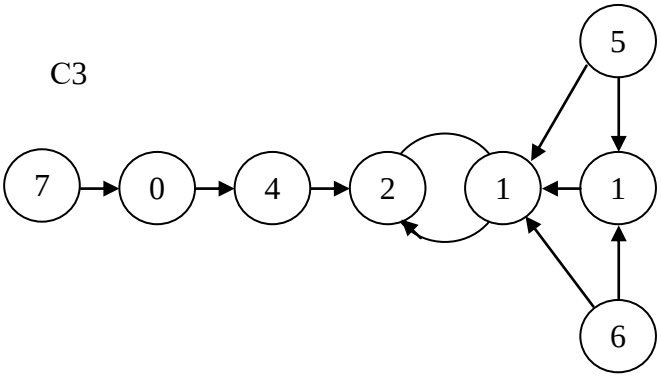
где регистр RG_1 содержит константу (настроечный параметр); на RG_2 поступает входной тег; символ “$\rightarrow$” обозначает циклический сдвиг на один разряд в сторону младших разрядов, символ $\oplus$ обозначает операцию сложения по модулю 2.

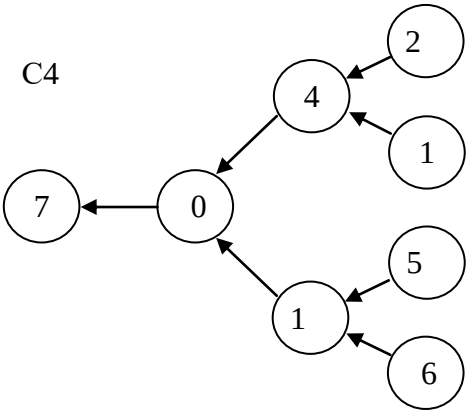


Функциональная схема ПТ









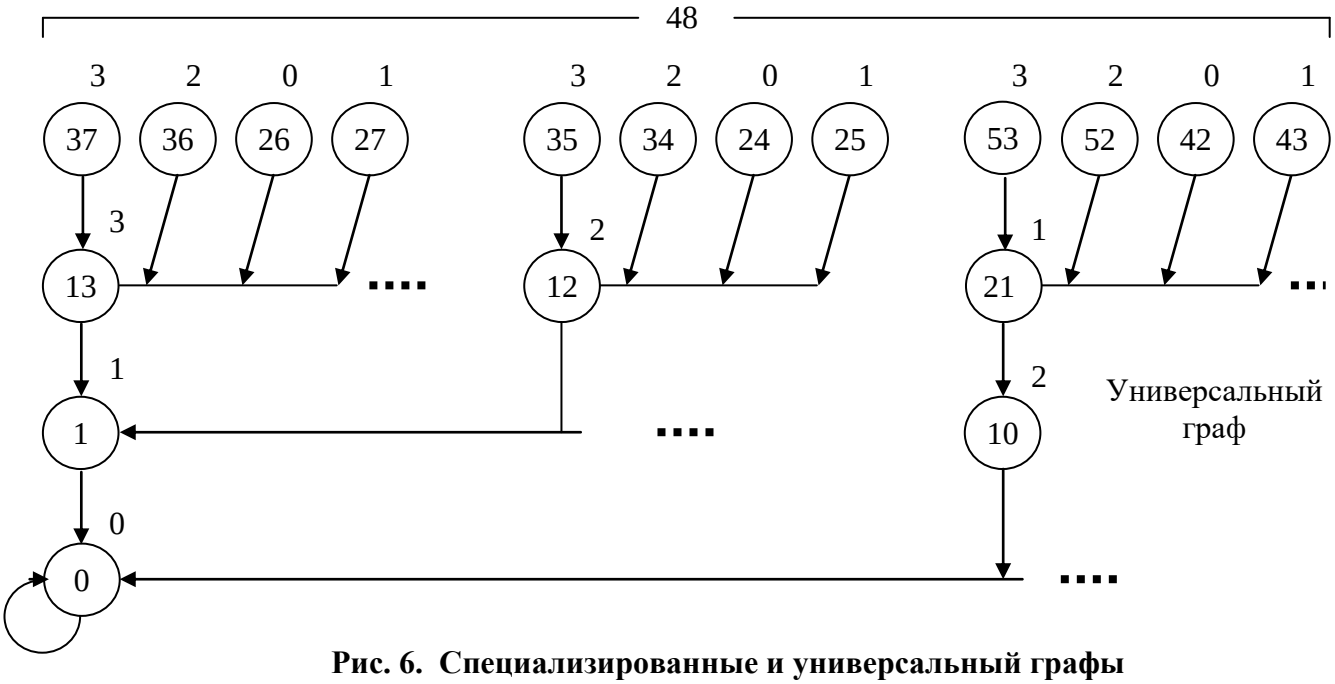


Рис. 6. Специализированные и универсальный графы

Если результат преобразования – выходной тег – снова завести на вход RG_2, то на выходе ПТ получится некоторая последовательность чисел. На рис. 6 приведены структуры графов, которые порождаются при разных значениях констант (C=1, 2, 3, 4):

– *самовозвратные полюса* (C = 1),

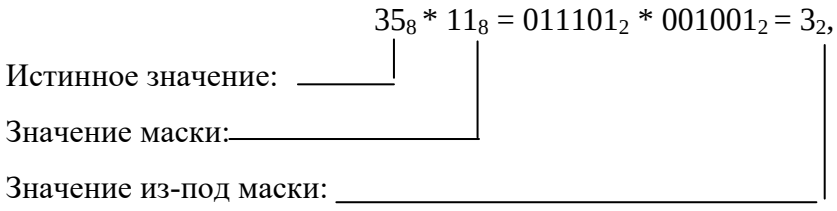
– *кольца разной длины* (C = 2),

– *деревья* (C = 4),

– *деревья, сводящиеся к кольцу* (C = 3).

Для порождения более сложных структур используются различные приемы: конкатенация отдельных ПТ, маскирование отдельных разрядов, последовательное и параллельное соединение ПТ и т.д. В структуре порождаемого графа все вершины уникальны, имеют неповторяющиеся (натуральные) номера. При реализации реальных алгоритмов вершины, например, интерпретируемые как коды операций АЛУ, могут повторяться.

На рис. 6 приведена структура универсального ПТ (УПТ), который обеспечивает повторяемость (нумерацию) вершин, если их брать из-под маски на протяжении трех шагов (при условии, что выход ПТ замкнут на его вход). Натуральные значения показаны внутри вершин, а значения из-под маски – рядом с вершиной.



Программирование состоит в выборе одного из 48 начальных кодов на графе, так называемого НУР (начального условия развертки).

В табл. 2 сведены результаты выполнения на модели операционного уровня набора алгоритмов для четырех рассмотренных вариантов исполнения РОУ (с использованием универсального ПТ). Анализ результатов показывает безусловное преимущество централизованной структуры по всем показателям.

Таблица 2. Некоторые итоги выполнения алгоритмов обработки сигналов
(без настройки Преобразователей и их перезагрузки)

Ареал	**Распределенный**					**Централизованный**					**Слоевой**					**Смешанный**				
Пример	**Ц**	**Ш**	**К**	**С**	**П**	**Ц**	**Ш**	**К**	**С**	**П**	**Ц**	**Ш**	**К**	**С**	**П**	**Ц**	**Ш**	**К**	**С**	**П**
Автокорреляция	5	5	12+i	10	3	2	0	4+i	9	2	2	5	7+i	10	2/1	2	11	15+i	9	0/1
«Бабочка» БПФ	12	-	16	4	3	5	0	14	4	2	5	0	20	4	0/1	4	-	20	4	0/1
Вектор 1	4	6	10+2j	2	3	2	2	11+2j	2	2	2	2	9+2j	3	0/1	2	6	12+2j	2	0/1
Вектор 2	5	3	7+2k	1	4	1	1	7+2k	1	2	2	2	6 +2k	2	0/0					
Прекомпенсация (1)						5	0	14+(i-1)	1	4										
Прекомпенсация (2)	5	9	10+i	2	3	3	0	14+(i-1)	2	2	8	10	15+(7m)	3	2/3					
Прекомпенсация (3)						2	1	10+(i-1)	3	2										
Факториал (1)	3	7	8 +	1	4	3	1	+	1	3	3	0	5	2	2/1					
Факториал (2)						2	2	+	2	1										

Ц – число шагов для вхождения в цикл
Ш - число шагов алгоритма в цикле;
К - длина капсулы в операндах;
С - число используемых секций;
П - число необходимых ячеек Памяти;

i- число операндов S-типа;
j – число пар операндов начиная с С2,С3;
K – число пар операндов А и В типа, начиная с А2 и В2;
l – число входных операндов S-типа;
m – число входных операндов S-типа при S>= 3.

Автокорреляция

$$ACF(k) = \sum_{i=k}^{N-1} S(i) * S(i-k);$$

k = 0, 1,..., 8; N = 1, 2, ... 160.

Прекомпенсация

$\mathbf{S_{of}}(k) = \mathbf{S_o}(k) - \mathbf{S_o}(k\text{-}1) + \mathbf{alpha} * \mathbf{S_{of}}(k\text{-}1)$.
$\mathbf{S}(k) = \mathbf{S_{of}}(k) - \mathbf{beta} * \mathbf{S_{of}}(k\text{-}1)$.
k = 0, 1, …..159, $\mathbf{S_{of}}(0) = 0$

Вектор 2

X (i) = A (i) * X (i-1) + B (i)
Дано: X (0), A (i), B (i);
i = 1,..., 16.

Факториал

FAC = N!

Вектор 1

A (i) = B (i) * C (i)
B (i+2) = A (i) + T
Дано: T, B(0), B(1),C(0),..., C(n).
T – константа, A, B и C – однокомпонентные вектора

Комплексное БПФ "Бабочка"

Все значения - комплексные.
A' = A + BW
B'= A - BW

В *Приложении 2* представлена технология программирования вертикального подареала на примере реализации одного из алгоритмов обработки радиосигналов - алгоритма *прекомпенсации*. Входным языком РВП, как и в потоковых машинах, является граф. Поэтому сначала разрабатывается граф-программа реализации алгоритма прекомпенсации с использованием общепринятой семантики [7]. Из нее видно, что максимально возможная степень параллельности реализации этого алгоритма равна 3. В *Приложении 3* представлена процедура преобразования уравнений; в соответствии с ней составлена программа-капсула, в глобальном конфигураторе ***Ac:*** которой указано, что для ее обработки в РОУ привлекается три секции (вспомогательное подполе глобального конфигуратора [*Cp*] = 3). В *Приложении 4* в наглядной форме представлена процедура реализации алгоритма на трех секциях с использованием моделирующей программы «ОПЕРА».

Из капсулы видно (это характерно для всех циклических процедур), что после вхождения в цикл значения функциональных полей содержательных операндов повторяются. Например, начиная с операнда s2 и до S159 весь комплект функциональных полей [*OcuSmDsea*] остается неизменным. Поэтому в структуру ценрализованного варианта РОУ предполагается ввести Регистр тега, в который будут заноситься повторяющиеся значения функциональных полей. Это позволит в рамках одного элемента капсулы разместить не один содержательный операнд, а два, но уже без функциональных полей. Регистр тегов будет служить источником этих полей для капсульных безтеговых операндов. На Экспликатор операнды будут приходить уже с полным комплектом полей, а это позволит почти в два раза уменьшить объем капсулы и в два раза снизить трафик между операционным и процедурным уровнями РДА.

Реализация большинства алгоритмов сигнальной обработки характеризуется использованием различного рода констант (например, значений поворотных коэффициентов при БПФ, постоянных коэффициентов фильтров и т.д.). В ряде случаев, целесообразно хранить

эти константы в сжатом рекуррентно-свернутом виде в специальном модуле «Преобразователь константы» (ПК). Функционально ПК идентичен ПТ, выход которого соединен с его входом. Однако на входе и выходе ПК имеют место не значения функциональных полей, а собственно значения константы. При каждой новой инициализации ПК на его выходе будет новое значение константы.

Введение модуля ПК исключит необходимость пересылки констант с процедурного уровня и позволит существенно сократить объем запоминающей среды для хранения констант в РОУ.

Размер теговых полей в РОУ с настраиваемыми ПТ приблизительно равен размеру содержательной части операндов в 32-разрядных словах (значительно меньше, чем в других архитектурах). С увеличением разрядности слов теговые поля не увеличиваются.

7. СРАВНЕНИЕ АППАРАТНОЙ СЛОЖНОСТИ СТРУКТУР РОУ

Для обоснованного выбора рабочей структуры была проведена сравнительная оценка сложности реализации слоевого, слайсового и смешанного вариантов структур. Вертикальный вариант менее сложен в реализации, чем слайсовый (см. 5.4), но отдельно в оценках не фигурирует. Оценка включала разработку схем в базисе КМОП-технологии со спуском на транзисторный уровень схем (с целью получения достаточно точной оценки).

Были приняты следующие условия:

а) *используемая технология* - КМОП с двухслойной металлизацией и минимальным топологическим размером 0,8 мкм;

б) *плотность упаковки* (*транзисторов/мм*^{2}): 6000 - для арифметических устройств и регистров на основе триггеров, 25000 - для устройств памяти на основе ОЗУ, 3000 - для прочих устройств;

в) *стековая, буферная и секционная памяти* - ОЗУ динамического типа на базе одно-транзисторных ячеек.

Результаты оценки для трех значений размеров внутренней памяти в расчете на один канал приведены в табл. 3.

Таблица 3. Сравнение вариантов реализации структур

NN пп	**Характеристика**	**Объем внутренней памяти**	**Варианты**		
			Слоевой	**Распределенный**	**Смешанный**
1	Разрядность, *бит*	-	41	41	21*)
2	Количество транзисторов $\times 10^3$	1024 128 64	250 120 110	250 120 110	360 110 89
3	Активная площадь, мм^2	1024 128 64	20 14 14	21 15 15	24 12 11
4	Общая площадь, мм^2	1024 128 64	41 30 29	30 23 32	41 21 19
5	Коэффициент заполнения кристалла	-	0,5	0,7	0,6

*) При основной 16-разрядной сетке данных

Анализ результатов показал:

1) При использовании динамических ОЗУ для реализации памяти они (во всех вариантах и в подавляющем большинстве случаев) занимают меньшую часть кристалла по сравнению с арифметическими и логическими устройствами.

2) Наличие ортогонального селектора в слоевом и смешанном вариантах и ортогонального компаратора в слоевом варианте при двухслойной металлизации ведет к неэффективному использованию площади кристалла и появлению длинных линий связи, что снижает общее быстродействие.

3) Наилучшим по числу транзисторов и площади, занимаемой ими на кристалле (при внутренней памяти объёмом 64 слова на канал), является смешанный вариант. Слоевая и распределенная структуры практически одинаковы. Разница в площадях может быть нивелирована

при использовании трехслойной металлизации и введении дополнительных слоев для разводки.

4) Вертикальный вариант более технологичен с точки зрения топологического проектирования; в сочетании с относительно простой реализацией компилятора для него это может служить основанием для выбора его в качестве базового.

Приведенные оценки сложности реализации вариантов РОУ имеют точность 5÷10 % - по числу транзисторов и 20÷30 % - по площади.

8. Выводы

Рекуррентное операционное устройство отличается от подобных ему устройств современных DSP-процессоров рядом особенностей, которые есть смысл аккумулировать вместе в качестве выводов статьи. Архитектура РОУ**:**

1) ориентируется на производство параллельных вычислений рекуррентно представленных алгоритмов;

2) опирается на рекуррентно-динамическую парадигму (модель) вычислений, математической основой которой является теория рекуррентно-динамических графов;

Собственно интегральный вариант РОУ отличается от своих конкурентов рядом особенностей. В частности, РОУ оригинальна тем, что:

а) имеет предельно-минимальное количество фаз обработки слов ЭСД – их три**:** «*сравнение функциональных полей – выполнение операции – запись результата*»**;** а с учетом статистической информации - их две:

б) обеспечивает аппаратную обработку и векторных, и скалярных величин, и снимает все ограничения на программирование векторных операций, свойственные традиционным векторным процессорам**;**

в) является *самодостаточным* аппаратным устройством, управление работой которого осуществляется только от поступающих на его вход ЭСД, содержащих в рекуррентно-сжатом виде программу выполнения алгоритма и сведения об операндах;

г) использует нетрадиционные для DSP-процессоров операционные исполнительные устройства: ассоциативную память и преобразователь тегов (ПТ). Функция ПТ – реализация программы пошаговой обработки аргументов алгоритма по мере его рекуррентного саморазвёртывания и исполнения. Память обеспечивает хранение исходных и промежуточных операндов, а также их паросочетание на каждом шаге исполнения алгоритма.

д) существенно сокращает потери времени на выполнение некоторых вспомогательных процедур, исключая потери традиционных Data flow архитектур, связанные с занесением в операционное устройство программы (набора инструкций);

е) существенно сокращает суммарный объем запоминающей среды: за счет рекуррентного представления алгоритмов;

ж) соотношение размеров тегов (по числу разрядов) и содержательной части ЭСД (собственно данные) лучше, чем в известных теговых архитектурах.

Выбранный вариант организации РОУ представляется перспективным для использования в DSP-процессорах.

Приложение 1.

СОПОСТАВЛЕНИЕ АРХИТЕКТУРЫ РОУ С СУЩЕСТВУЮЩИМИ АРХИТЕКТУРАМИ ОПЕРАЦИОННЫХ УСТРОЙСТВ

Предлагаемая РДА операционного уровня нетрадиционна и радикально отличается по основным моментам не только от классической архитектуры фон Неймана, но и от других нетрадиционных параллельных архитектур.

Для существующих традиционных и нетрадиционных компьютерных архитектур характерно наличие двух потоков: активного потока инструкций и пассивного потока данных.

В РДА оба потока сливаются в один общий поток самодостаточных данных, в котором, помимо собственно обрабатываемых данных, содержится и необходимая для их обработки управляющая и служебная информация (в существующих архитектурах кодируемая в форме инструкций).

Наглядное представление о сравнительных качествах рассматриваемых архитектур дает их сопоставление:

- по организации памяти (рис. П1);

- по количеству шагов, необходимых для выполнения инструкции (рис. П2).

Рисунки условны и приводятся только для наглядности.

Для выполнения программы в традиционной классической фон-неймановской архитектуре требуется некоторый объем запоминающей среды (памяти), которая функционально (а для гарвардской архитектуры и физически) разделена на две области - программ и данных. Инициатором выполнения вычислительной процедуры является поток инструкций, извлекаемый из памяти программ ЦПУ (первичный поток инструкций). Программа-инициатор процесса хранится в памяти инструкций в полном объеме и в статическом виде (отсюда название CF/S - Control Flow/Static). При этом существует проблема определения момента готовности данных для их обработки.

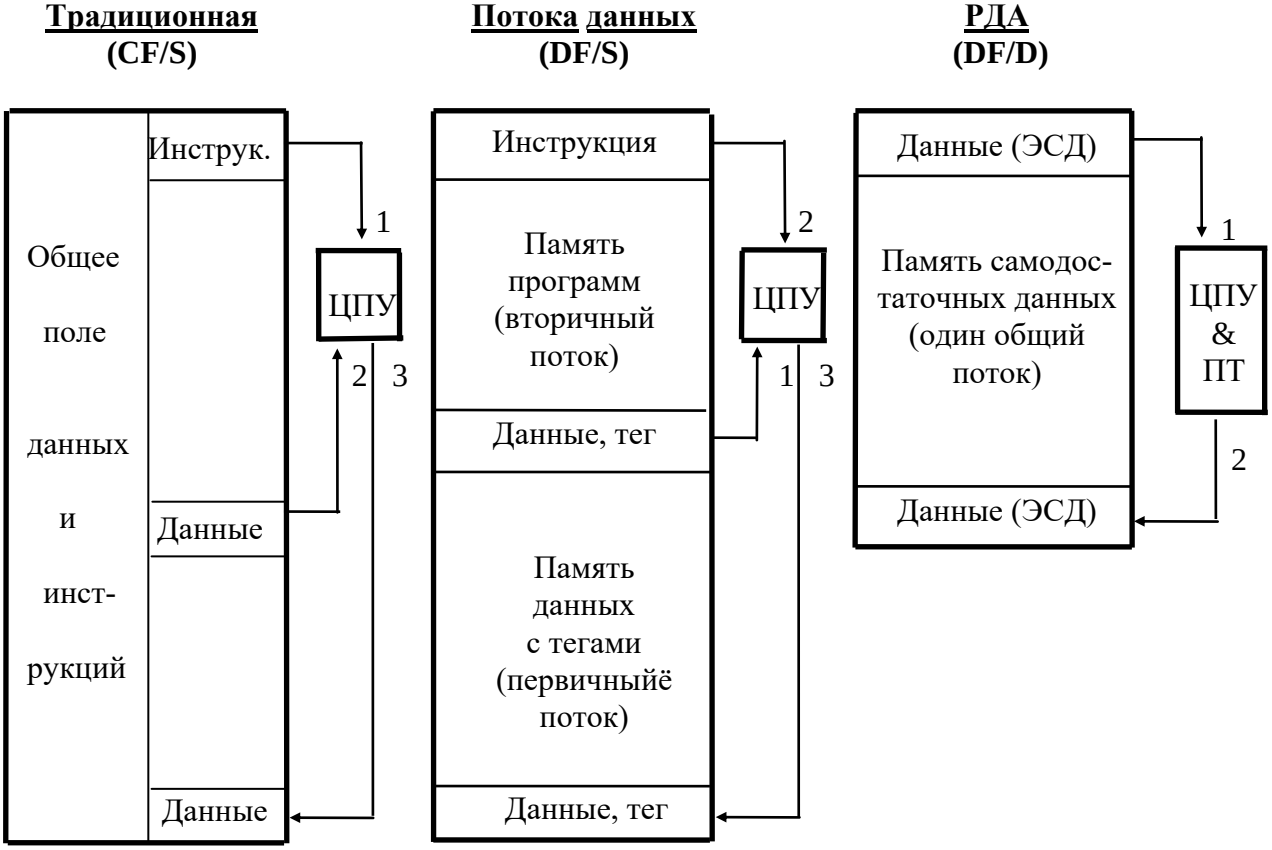


Рис. П1. Принципиальные отличия сравниваемых архитектур

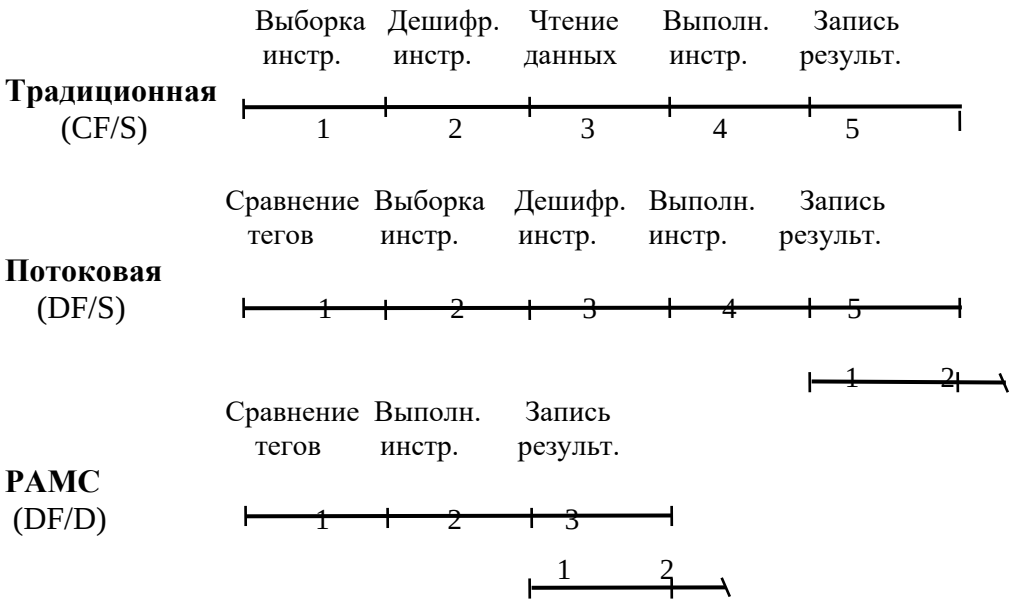


Рис. П2. Выполнение инструкций в сравниваемых архитектурах

Для потоковых архитектур (DF - Data Flow) сохраняется разделение ресурсов памяти на две области. Однако статус памяти данных меняется - из пассивной (вторичной) она превращается в активную (первичную), в ячейках которой хранятся данные (операнды) с дополнительными функциональными полями (полями тегов). Суммарный объем требующихся ресурсов памяти ориентировочно не изменяется. Как правило, функциональные поля содержат в себе информацию об исполнительном адресе инструкции (микрокоманды), которая должна быть извлечена из памяти программ для выполнения требуемых действий. Полный объем привлекаемых инструкций также хранится в статическом виде.

РДА относится к классу потоковых архитектур, причем в отличие от них тегируемые данные являются самодостаточными (рекуррентно разворачивающимися). Теги содержат некоторую начальную сжатую информацию, обеспечивающую выполнение требуемой процедуры. Каждый следующий шаг процедуры рекуррентно самоопределяется, в том числе с учетом результата предыдущего шага.

В разрабатываемой архитектуре в состав ЦПУ включено автономное устройство преобразования тегов (ПТ), которое обладает возможностью саморазвертки рекуррентного вычислительного процесса. Устройство ПТ инициируется операндами, пришедшими на обработку в ЦПУ, работает параллельно с ЦПУ и определяет действие на следующем шаге вычислительного процесса (модифицирует тег). Устройство ПТ представляет собой относительно простую (по аппаратным затратам) комбинационную схему, содержащую средства настройки (в необходимых случаях) на предметную область.

В памяти нет исполняемой программы в традиционном смысле. Есть только начальные значения функциональных полей операндов, которые динамически подвергаются рекуррентной саморазвертке устройствами ПТ и РПТ (отсюда название архитектуры DF/D - Data Flow/Dynamic). Для выполнения алгоритма необходимо задать начальные значения функциональных полей. Для получения искомого результата в CF/S выполняется последовательность действий 1-5 (см. рис. П2). Число шагов в DF/S не меняется. В архитектуре DF/D оно уменьшается до логически

необходимого минимума - 3; уменьшается также суммарный объем требуемых ресурсов памяти и число пересылок между функциональными устройствами ЭВМ. Если результат операции в DF/D – промежуточный, и данные (его пара) уже готовы (пришли на обработку), то шаг 3 (запись результата) совпадает по времени (совмещается) с шагом 1 выполнения следующей инструкции. В результате за 5 шагов может быть выполнена не одна, а две инструкции. Аналогично и в DF/S: вместо десяти шагов - девять.

На последнем обстоятельстве стоит остановиться более подробно, поскольку именно здесь обеспечивается серьезное преимущество предлагаемой архитектуры по сравнению с другими архитектурами высокопараллельной обработки. Предлагаемая архитектура одинаково эффективно реализует как алгоритмы, параллельное выполнение которых не зависит от результата обработки на текущем шаге алгоритма (например, алгоритмы векторной обработки), так и сугубо рекурсивные алгоритмы. Результат операции на текущем шаге обработки может быть использован следующим образом:

1) Результат текущей операции (шаг 3 на рис. 2) используется другим операндом, прошедшим шаг 1 (механизм связывания будет описан позже). Отметим сразу, что любой результат в любой секции РДА безусловно заносится в аккумулятор этой секции. Инициатором операции в следующем шаге является операнд из капсулы, только что поступивший на обработку.

2) Результирующий операнд (однократного или многократного использования) текущей операции поступает в селектирующую среду, где его ждет партнер для следующей операции. Инициатором операции на следующем шаге является операнд-результат предыдущего шага.

3) Результирующий операнд (однократного или многократного использования) текущей операции поступает в селектирующую среду, куда одновременно поступает его партнер (однократного использования) по операции. Инициатором операции на следующем шаге является пара операндов - результирующий операнд предыдущего шага и текущий операнд из капсулы.

4) Результирующий операнд (однократного использования) текущей операции поступает в селектирующую среду, куда одновременно поступает

его партнер (многократного использования) по операции. Инициатором операции на следующем шаге является пара операндов - результирующий операнд предыдущего шага и текущий операнд из капсулы.

5) Результат текущей операции поступает в качестве результирующего операнда на верхний (процедурный) уровень РДА и одновременно используется в качестве промежуточного результата (см. п.п. 1 - 5) - с соответствующим инициатором операции.

6) Результат текущей операции поступает в качестве результирующего операнда на верхний (процедурный) уровень РДА. Его дальнейшее использование не предполагается. Во время исполнения текущего шага 2 подготовлена пара операндов для следующего шага. Инициатором операции в следующем шаге является один или пара операндов из капсулы.

7) Результирующий операнд (однократного или многократного использования) текущей операции поступает в селектирующую среду и ждет поступления своего партнера по операции из капсулы. Инициатором операции в следующем шаге является операнд из капсулы. Как правило, этот случай – пример либо неэффективного программирования, либо отсутствия сбалансированного параллелизма в исходном алгоритме (достаточно редкого события).

Перечисленные случаи (1-6) позволяют совместить шаг 3 текущей и шаг 1 следующей операции. В идеальном случае (если случаи 1-6 достаточно часты) и при рекурсивном виде реализуемого алгоритма число логически требуемых шагов для реализации операции приближается к 2, что дает производительность в 2,5 раза большую, чем в традиционной и DF/S архитектурах. Этот вывод справедлив и без использования такого эффективного механизма повышения быстродействия, как конвейеризация исполнения отдельных фаз операции, применимая, естественно, и в РДА.

Использование конвейеризации в рамках архитектур CF/S и DF/S позволяет сократить число необходимых шагов для реализации инструкций на операционном уровне. Однако существует достаточно много случаев, когда работа конвейера инструкций в CF/S-архитектуре (или тегируемых пар операндов в DF/S-архитектуре) может быть нарушена [8]:

1) при необходимости использования результата текущей инструкции в следующей инструкции;

2) если результат текущей инструкции определяет адрес операнда следующей инструкции;

3) при изменении последовательности выполнения инструкций в ходе выполнения инструкции безусловного перехода;

4) при изменении последовательности выполнения инструкций в ходе выполнения инструкции условного перехода;

5) при необходимости реализации рекурсивных операций (результат текущей инструкции не обязательно используется в следующей инструкции, но все равно разрушает конвейер инструкций);

6) при необходимости обеспечения когерентности основной памяти и кэш-памяти (при отсутствии в последней инструкции или адреса операнда);

7) при возникновении конфликтов при доступе к памяти (к инструкции, операнду - для чтения или записи);

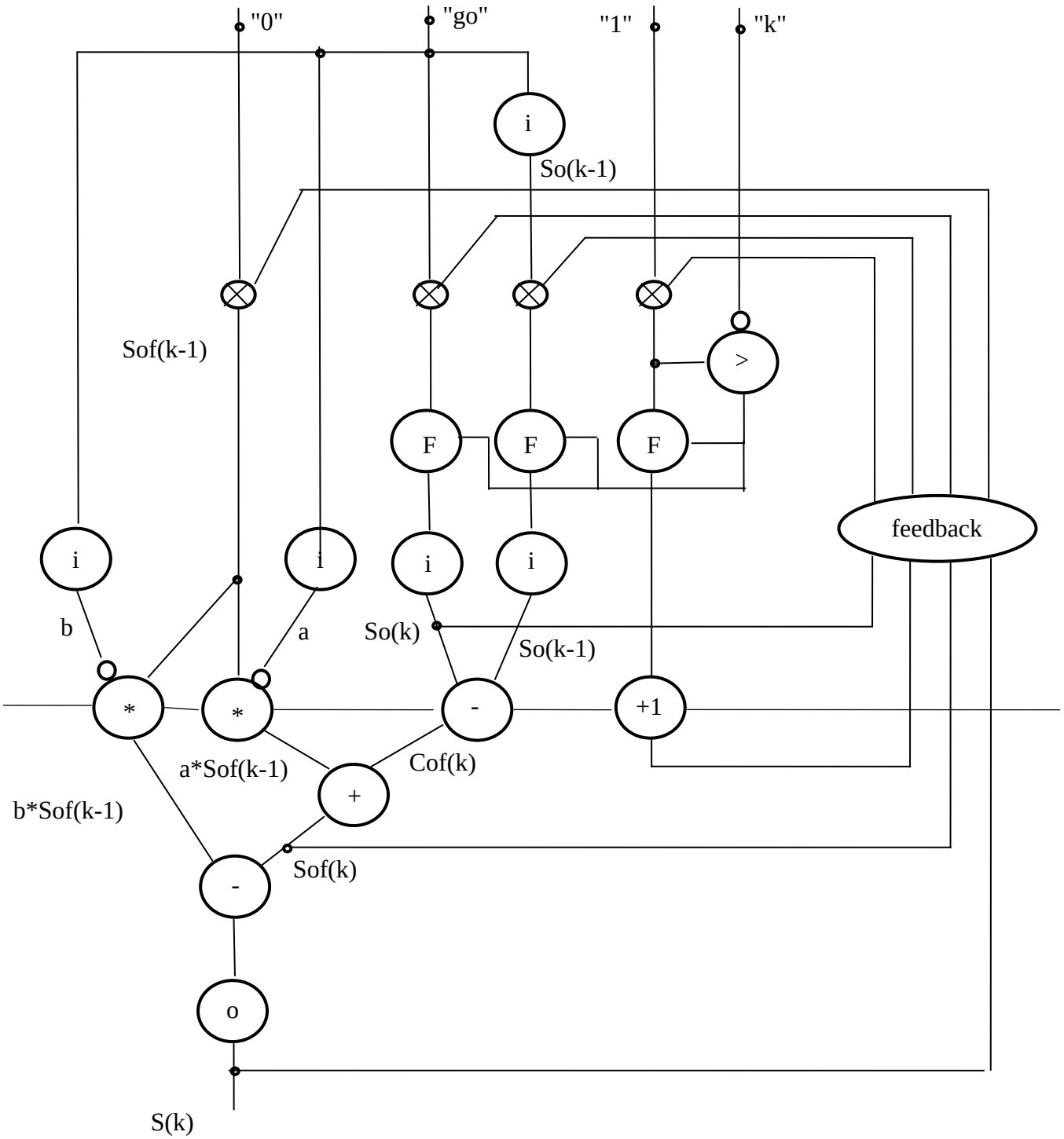
8) при возникновении прерываний, переключении процессов.

События (1-7) в случае РДА исключены по определению, а последствия маловероятных событий (8) в одинаковой степени снижают быстродействие рассматриваемых архитектур. С учетом сказанного можно утверждать, что 2,5-кратное преимущество РДА с точки зрения числа необходимых шагов остается справедливым для выполнения как одиночной инструкции, так и конвейеризованного потока инструкций.

Отметим еще одно обстоятельство. Для инициации программы на операционном (обрабатывающем) уровне должны быть предварительно выполнены некоторые вспомогательные процедуры: ввод обрабатываемых данных, занесение программы и ее инициация. Издержки, связанные с выполнением этих процедур в рассматриваемых архитектурах, существенно разнятся при одинаковой СП. По сравнению с рассматриваемыми архитектурами уровень издержек в РДА существенно меньше.

Приложение 2.

Граф-программа реализации алгоритма "Прекомпенсации"



Обозначения: "feedback" – возвратная дуга синхронизации;
"i" (input) – ввод;
"o" (output) – вывод;
"F" – коммутатор (вершина управления);
двухвходовые операционные вершины:
"*" – умножение,
"+" – сложение,
"-" – вычитание

Приложение 3.

АЛГОРИТМ ПРЕКОМПЕНСАЦИИ

Исходные уравнения:

Компенсация смещения (дельта-модуляция):

$$\mathbf{S_{of}}(k) = \mathbf{S_o}(k) - \mathbf{S_o}(k-1) + \mathbf{alpha} * \mathbf{S_{of}}(k-1).$$

Прекомпенсация (фильтр 1-го порядка):

$$\mathbf{S}(k) = \mathbf{S_{of}}(k) - \mathbf{beta} * \mathbf{S_{of}}(k-1),$$ начальные условия: f(0) = 0; k=0÷159

Алгоритм преобразования уравнения:

0) a(k) = a * f(k-1) ; произведение alpha
1) c(k) = s(k) - s(k-1) ; текущая типовая разность
0) f(k) = a(k) + c(k) ; свободное типовое значение смещения
2) b(k) = b * f(k-1) ; произведение beta
r(k) = f(k) - b(k) ; типовое значение результата
r(k) | Atm ; значение выхода с шаблоном

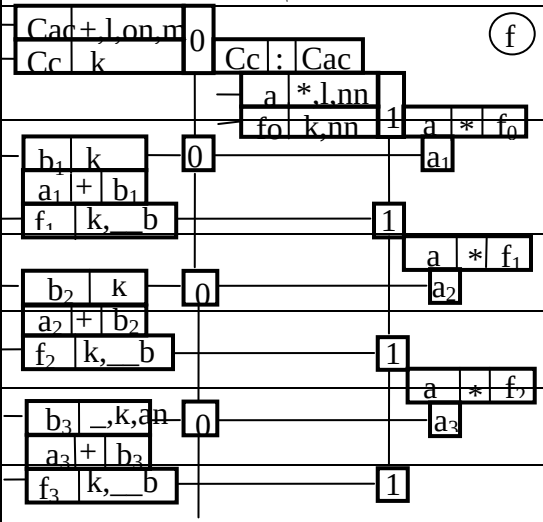
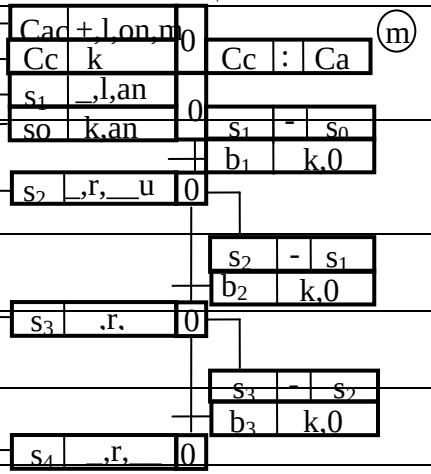
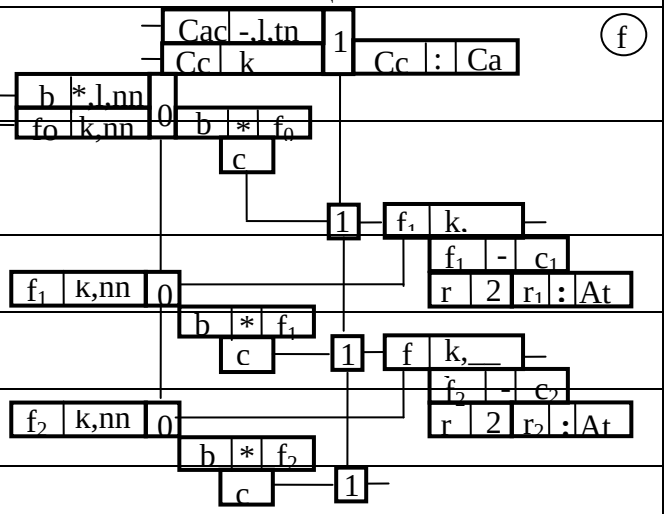
Обозначения 0), 1) и 2) соответствуют выполняемому в секциях 0, 1 и 2.

Структура капсулы

```
Ac: Cx=2 Cp=3 Cc=d Cr= Ce= Cs=ei_xi Cd= ;
Ai: I1n=Sf I1i=: I1s= ;
Am: %Mv=100 %Mt=e %Mo=0 %Mm=s ;
Atm: Tc= Tm= Tu= Tr= Tn=0 To=0 Ts= Te= Ta= [ Sm=2 Dr=0100 ];
Cacn: [ Oc=+ Ou=l Sm=0 Dr=0011 Ds=on De=m Da= ];
Cacn: [ Oc=- Ou=l Sm=1 Dr=0100 Ds=tn De= Da= ];
Cc: Cj=f Cd=1 Cl=> Ce= Cr=d [ Oc= Ou=k Sm=0 Dr=0001 Ds= De= Da= ];
Cc: Cj=f Cd=1 Cl=> Ce= Cr= [ Oc= Ou=k Sm=1 Dr=0100 Ds= De= Da= ];
Cc: Cj=m Cd=1 Cl=> Ce= Cr= [ Oc= Ou=k Sm=0 Dr=0010 Ds= De= Da= ];
Di: V=a [ Oc=* Ou=l Sm=1 Dr=0001 Ds=nn De= Da= ];
Di: V=s1 [ Oc=_ Ou=l Sm=0 Dr=0010 Ds=an De= Da= ];
Di: V=b [ Oc=* Ou=l Sm=0 Dr=0100 Ds=nn De= Da= ];
Di: V=f0 [ Oc= Ou=k Sm=1 Dr=0001 Ds=nn De= Da= ];
Di: V=f0 [ Oc= Ou=k Sm=0 Dr=0100 Ds=nn De= Da= ];
Di: V=s0 [ Oc= Ou=k Sm=0 Dr=0010 Ds=an De= Da= ];
Di: V=s2 [ Oc=_ Ou=r Sm=0 Dr=0010 Ds=_u De= Da= ];
Di: V=s3 [ Oc=_ Ou=r Sm=0 Dr=0010 Ds=_u De= Da= ];
Di: V=s4 [ Oc=_ Ou=r Sm=0 Dr=0010 Ds=_u De= Da= ];
. . . . . .
Di: V=s157 [ Oc=_ Ou=r Sm=0 Dr=0010 Ds=_u De= Da= ];
Di: V=s158 [ Oc=_ Ou=r Sm=0 Dr=0010 Ds=_u De= Da= ];
Di: V=s159 [ Oc=_ Ou=r Sm=0 Dr=0010 Ds=_u De= Da= ];
Cacd:
```

Приложение 4.

Граф-программа реализации алгоритма «Прекомпенсации» на трех секциях РОУ

RAMS		Operational vertical	GSM: OFFSET COMPENSATION and PREEMPHASIS	%Cc (discard); %Cs=ei (merging), %Ce (selecting)
Шаг	Пам.	Секция 0	Секция 1	Секция 2
1	Cc, Cac(n)			
2	a, b, f_0, s_0, s_1			
3	s_2			
4				
5	s_3			
6				
7	s_4			
8				

Двухшаговый стационарный цикл.
Число шагов для вхождения в цикл - 1.
По сравнению с вариантом prec_v02 число шагов в цикле сокращено с 3 до 2 (за счет использования шины промежуточных результатов Em).
Размер капсулы - 10+(i-1), где i - число входных операндов S-типа.
Число используемых секций - 3.
Число используемых ячеек памяти в секциях -2.
Файл Prec2.doc для капсулы Prec2.cap

ЛИТЕРАТУРА

1. *Филин А.В.* Особенности обработки сигналов на процессоре с рекуррентно-динамической парадигмой вычислений. // Наст. сб.
2. *Филин А.В.* Динамический подход к выбору архитектуры вычислительных устройств обработки сигналов. // Наст. сб.
3. *Махибoрода А.В.* Проблема создания вычислительных средств с непроцедурными внутренними языками. УСиМ, № 3, 1993. - Киев, Наукова думка. - с. 52-62.
4. *Мизин И.А., Махибoрода А.В.* Концепция самоопределяемых данных и архитектура распределенных вычислительных систем. - Информационные технологии и вычислительные системы, том 1, № 1, 1995. - Москва, Наука. - с. 12-22.
5. *Махибoрода А.В.* Проблема реализации динамического параллелизма в естественно-надежных компьютерах. - Системы и средства информатики. Москва. Наука. 1995. Вып. 7. С. 141-146.
6. Исследования в области архитектуры персональных ЭВМ и основы структурной динамики. // Яковлев Ю.С., Махибoрода А. В. - Киев, 1987. - 31 с. (Препринт АН УССР, Ин-т кибернетики им. В.М. Глушкова; 87 - 35).
7. *Амания М., Танака Ю.* Архитектура ЭВМ и искусственный интеллект: Пер. с японск. - М.: Мир, 1993. - 400 с.
8. *Von Neumann J.* Collected Works (ed. A.H.Taub), 6 vols., Pwrgamon, New York (1963); vol. 5: design of Computers. Theory of Automata and Numerical Analysis.