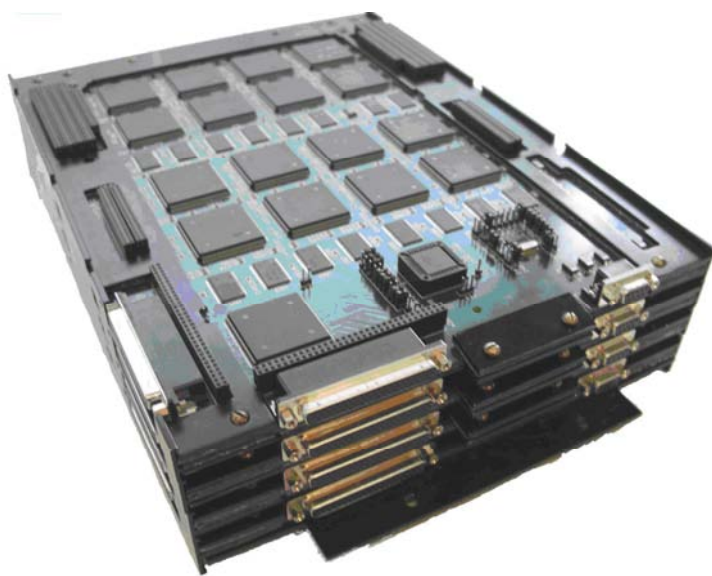


**РОССИЙСКАЯ АКАДЕМИЯ НАУК
ЮЖНЫЙ НАУЧНЫЙ ЦЕНТР**

**И.А. Каляев, И.И. Левин,
Е.А. Семерников, В.И. Шмойлов**

РЕКОНФИГУРИРУЕМЫЕ МУЛЬТИКОНВЕЙЕРНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СТРУКТУРЫ



**Ростов-на-Дону
Издательство ЮНЦ РАН
2008**

Оглавление

Предисловие.....	3
Введение.....	7
Глава 1. Принципы организации реконфигурируемых мультиконвейерных вычислительных структур.....	17
1.1. Проблема производительности многопроцессорных систем..	17
1.2. Поточковые задачи и способы их решения.....	26
1.3. Принципы мультиконвейерной обработки потока данных..	32
1.4. Мультимикро- и мультимакроконвейерные вычислительные структуры.....	40
1.5. Структурно-процедурный способ организации мультиконвейерных вычислений.....	47
Глава 2. Мультимикроконвейерные вычислительные структуры на однородных средах.....	58
2.1. Архитектура однородной вычислительной среды.....	58
2.2. Организация вычислительного процесса в ОВС.....	64
2.3. Математическая модель ОВС.....	67
2.4. Синтез спецпроцессоров в среде.....	73
2.5. Структура микропрограммного обеспечения.....	77
2.6. Процесс настройки поля ОВС.....	79
2.7. Программирование однородных сред.....	82
2.7.1. Образование и использование констант.....	82
2.7.2. Внешняя подача констант в спецпроцессор.....	85
2.7.3. Форматы операндов.....	86
2.7.4. Спецпроцессоры арифметических и логических операций...	88
2.8. Решение прикладных задач на ОВС.....	102
2.8.1. Задача обработки изображений.....	102
2.8.2. Реализация цифровых фильтров в поле ОВС.....	104

2.9. Аппаратная реализация мультимикроконвейерных структур на ОВС.....	115
2.10. О перспективах развития однородных вычислительных сред.....	123
Глава 3. Структурно-процедурная организация вычислений	
в мультиконвейерных структурах.....	132
3.1. Принципы функционирования мультимакроконвейерных вычислительных структур.....	132
3.2. Преобразование информационных графов в структурно-процедурные программы.....	142
3.3. Преобразование задачи в структурно-процедурную форму....	152
3.4. Преобразование функционально-регулярных информационных графов.....	161
3.5. Преобразование в кадровую форму решения задачи Пуассона.....	164
3.6. Преобразование в структурно-процедурную форму информационных графов нерегулярной структуры.....	181
3.7. Структурно-процедурная реализация задачи трассировки.....	186
Глава 4. Реконфигурируемые мультиконвейерные вычислительные структуры на основе ПЛИС	
структуры на основе ПЛИС	197
4.1. Модульно-наращиваемая реализация реконфигурируемых мультиконвейерных вычислительных структур.....	197
4.2. Элементная база реконфигурируемых мультиконвейерных вычислительных структур.....	207
4.2.1. Заказные СБИС.....	207
4.2.2. Базовые матричные кристаллы.....	208
4.2.3. Системы-на-кристалле.....	209
4.2.4. Программируемые логические интегральные схемы.....	211

4.3. Принципы построения базовых модулей реконфигурируемых мультиконвейерных вычислительных структур на основе ПЛИС.....	218
4.4. Реализация базовых модулей реконфигурируемых мультиконвейерных вычислительных структур на основе ПЛИС	223
4.4.1. Структуры базовых модулей первого поколения.....	223
4.4.2. Структура базовых модулей второго поколения.....	226
4.4.3. Базовые модули третьего поколения.....	231
4.4.3.1. Базовый модуль третьего поколения 16V4-50.....	232
4.4.3.2. Базовый модуль для решения задач цифровой обработки сигналов.....	234
4.4.3.3. Базовый модуль 4V4-25 малогабаритного ускорителя персонального компьютера.....	238
4.4.4. Базовые модули перспективных разработок.....	241
4.4.4.1. Структура базового модуля высокопроизводительных систем.....	243
Глава 5. Реконфигурируемые мультиконвейерные вычислительные структуры с макрообъектной архитектурой.....	248
5.1. Принципы построения реконфигурируемых вычислительных структур на основе макрообъектной архитектуры.....	248
5.2. Обобщенная структура макрообъекта.....	256
5.3. Вычислительные блоки макрообъектов.....	259
5.3.1. Представление данных в макропроцессоре.....	259
5.3.2. Обобщенная структурная схема функциональных узлов с плавающей запятой.....	260
5.3.3. Функциональный узел сложения чисел с плавающей запятой.....	261

5.3.4. Функциональный узел умножения чисел с плавающей запятой.....	262
5.4. Макрообъект для решения задач математической физики сеточными методами.....	263
5.5. Пример решения задачи математической физики с помощью макрообъекта.....	268
5.6. Макрообъект для решения задач цифровой обработки сигналов.....	271
5.7. Макрообъект для решения задач линейной алгебры.....	287
5.7.1. Структура макрообъекта для решения задач линейной алгебры.....	287
5.7.2. Реализация алгоритма решения СЛАУ методом Гаусса.....	288
5.8. Универсальный макрообъект.....	292
5.8.1. Структура макропроцессора.....	293
5.8.2. Макропроцессор с каскадной коммутационной структурой.	295
5.8.3. Каскадный коммутатор макропроцессора	298
5.8.4. Элементарный процессор.....	299
5.8.5. Блок команд макропроцессора	300
5.8.6. Блок операндов.....	302

Глава 6. Системное математическое обеспечение

реконфигурируемых мультиконвейерных вычислительных структур.....	304
6.1. Структура системного математического обеспечения.....	304
6.2. Язык ассемблер.....	314
6.2.1. Общие сведения.....	314
6.2.2. Элементы языка.....	317
6.2.3. Программирование распределенной памяти.....	323
6.2.4. Операторы управления.....	332
6.2.5. Внешние операторы контроллера распределенной памяти..	334

6.3. Язык параллельного программирования высокого уровня	
COLAMO.....	336
6.4. Среда проектирования параллельно-конвейерных программ..	364
Послесловие.....	377
Литература.....	380
Предметный указатель.....	390

Предисловие

Создание высокопроизводительных вычислительных систем входит в первую десятку жизненно важных программ ведущих государств мира. Без суперкомпьютеров невозможно обеспечить конкурентоспособность страны на мировом рынке, без них нельзя поддерживать обороноспособность государства на должном уровне. Очевидно, что без суперкомпьютеров затруднительно проводить современные научные изыскания в самых различных областях, будь то материаловедение, медицина, экономика, экология, кибернетика и т.д. Амбициозные программы по нанотехнологиям и термоядерной энергетике так и останутся впечатляющими программами, если российские ученые не будут иметь в своем распоряжении постоянно обновляющийся парк суперЭВМ.

К сожалению, в России в настоящее время имеется серьезное отставание в области суперкомпьютеров по сравнению с ведущими странами мира. В последнем списке TOP 500 (ноябрь 2007 г.), содержащем сведения о наиболее высокопроизводительных суперкомпьютерах, упомянуты семь российских суперкомпьютеров, находящихся далеко не на первых позициях, причем из них лишь два собственной разработки (self-made), остальные – покупные, от зарубежных фирм. Для сравнения, в той же 30-й редакции TOP 500 США представлены 284-мя вычислительными системами.

Что же делать? Идти по пути копирования западных образцов, как это практиковалось долгие десятилетия - малопривлекательное занятие, поскольку при таком подходе отставание будет только нарастать. Выход из сложившейся ситуации видится в поддержке и форсированном развитии оригинальных отечественных разработок в области высокопроизводительных вычислительных систем. Весьма перспективной нам представляется концепция реконфигурируемых многопроцессорных вычислителей. Суть этой концепции заключается в том, что архитектура вычислительной системы должна иметь возможности адаптироваться под структуру решаемой задачи. Фактически это означает, что пользователю должна быть предоставлена возможность

программировать проблемно-ориентированные многопроцессорные вычислительные системы, структура которых адекватна решаемой ими задаче. При этом в отличие от известных архитектур, достигается высокая реальная производительность вычислительной системы на широком классе задач, а также почти линейный рост производительности при увеличении числа процессоров.

Концепция многопроцессорных вычислителей с реконфигурируемой архитектурой развивается в нашей стране давно по нескольким направлениям. Здесь следует упомянуть работы А.В. Каляева [46], Э.В. Евреинова [40], И.В. Прангшвили [95], В.Г. Хорошевского [39], а также работы других авторов [12, 59, 126]. Несмотря на все различия, сущность этих работ состоит в организации вычислительного устройства с мультиконвейерным способом вычислений, обладающего возможностями реконфигурирования связей и, соответственно, адаптацией вычислительной системы под структуру решаемой задачи. Отличие же подходов заключается лишь в той или иной степени детализации конвейерной ступени, а также в том или ином способе реконфигурирования связей между ними. Поэтому в дальнейшем многопроцессорные вычислительные устройства такого типа будем называть реконфигурируемыми многопроцессорными вычислительными структурами с мультиконвейерным способом организации вычислений или, короче, *реконфигурируемыми мультиконвейерными вычислительными структурами*.

Однако, несмотря на достаточно большое число исследований в этом очень перспективном направлении, создание целого ряда экспериментальных и макетных образцов таких машин, дело до широкого внедрения реконфигурируемых многопроцессорных систем в нашей стране, к сожалению, не дошло. Возможно, в некоторой мере это было вызвано экономическими и политическими проблемами, имевшими место в 80-90-х годах прошлого века, но основная причина, с нашей точки зрения, была связана с отсутствием элементной базы, которая бы позволяла строить эффективные реконфигурируемые мультиконвейерные вычислительные структуры.

Созданные на основе отечественной элементной базы экспериментальные образцы реконфигурируемых мультиконвейерных вычислительных структур оказывались громоздкими, малонадежными, слаборемонтопригодными, что делало их неконкурентоспособными на отечественном и, тем более, на мировом рынках.

В конце 90-х годов XX века российским разработчикам вычислительных систем стала доступна элементная база западных фирм, которая позволяла разрабатывать эффективные реконфигурируемые мультиконвейерные вычислительные структуры. Это - так называемые программируемые логические интегральные схемы (ПЛИС) нового поколения, в английской аббревиатуре - FPGA (Field Programmable Gates Array).

ПЛИС представляют собой некоторую матрицу логических ячеек, за счет программирования и коммутации которых можно создавать аппаратные реализации различных вычислительных структур, причем перепрограммирование логических ячеек и связей может осуществляться многократно. Иными словами, ПЛИС – это некоторый «полуфабрикат» микросхемы, внутреннюю структуру которой может определять сам пользователь.

Появление ПЛИС сверхвысокой степени интеграции вдохнуло новую жизнь в достаточно старую идею построения реконфигурируемых вычислительных систем. Конечно, этот новый взгляд потребовал определенной переработки, модернизации и адаптации этой идеологии с учетом особенностей и возможностей современных ПЛИС.

В монографии излагаются результаты авторов, полученные в теории реконфигурируемых мультиконвейерных вычислительных структур, рассматривается практическая реализация этих структур на основе ПЛИС. Представленные результаты были получены в рамках фундаментальных и прикладных работ, проведенных в Научно-исследовательском институте многопроцессорных вычислительных систем Таганрогского государственного радиотехнического университета (ныне НИИ МВС Южного федерального

университета) и в Отделе информационных технологий и процессов управления Южного научного центра РАН. Авторы выражают искреннюю благодарность коллегам за неоценимую помощь при воплощении хрупких теоретических построений в «железо», а также при подготовке рукописи монографии к печати.

И.А. Каляев

23 января 2008 г.,

г. Таганрог

ВВЕДЕНИЕ

В последние десятилетия произошли заметные сдвиги в организации научного процесса: вследствие широкого внедрения вычислительной техники заметно усилилось направление компьютерного моделирования и эксперимента. Прогресс в суперкомпьютерных технологиях и распространение этих технологий во все новые сферы человеческой деятельности свидетельствуют о том, что суперкомпьютеры – это мощный инструмент, который позволяет форсировать продвижение научно-технологической мысли во многих отраслях.

Среди приоритетов в развитии наукоёмких технологий, кроме нанотехнологий и технологий космической промышленности, непременно отмечается задача создания отечественных суперЭВМ. Общеизвестно, что суперЭВМ определяют национальную безопасность и экономическую независимость государства. Наличие у страны суперЭВМ, особенно в военной промышленности, - это вопрос обороноспособности страны и возможности создания современных высокотехнологичных материалов.

Опыт развития вычислительной техники показывает, что вычислительная мощность настольных персональных компьютеров отстает от уровня производительности суперкомпьютеров приблизительно на 13 лет. Иными словами, по уровню производительности сегодняшние профессиональные персональные компьютеры соответствуют суперкомпьютерам 13-летней давности. Может показаться, что с ростом производительности настольных компьютеров сама потребность в суперЭВМ будет снижаться. Однако время показало, что устойчивой тенденцией научно-технического прогресса является постоянное появление новых приложений, для решения которых за приемлемое время необходима вычислительная техника высокой и сверхвысокой производительности.

Традиционной сферой применения суперкомпьютеров всегда были научные исследования в таких областях как физика плазмы, физика

конденсированных сред, атомная физика, теория элементарных частиц, астрофизика и т. д. Среди технических проблем, для решения которых используются суперкомпьютеры, укажем на задачи аэрокосмической и автомобильной промышленности, ядерной энергетики. Суперкомпьютеры нужны для прогнозирования в финансовой и экономической областях, разведки нефти и газа, предсказания погоды, глобальных климатических изменений, оптимизации транспортных потоков в мегаполисах, управления на крупных предприятиях. Без суперЭВМ невозможны передовые исследования и во многих других областях, например, в фармакологии, биологии, генетике и т.д. Идет процесс проникновения суперЭВМ в коммерческую среду. Здесь в качестве примеров можно привести графические приложения для кино и телевидения, а также обработки сверхбольших баз данных, визуализацию результатов расчетов и т.п. Широко используются суперкомпьютеры в химии, включая расчеты электронной структуры для целей конструирования новых материалов, например катализаторов и сверхпроводников. Суперкомпьютеры традиционно применяются для военных целей. Кроме разработки оружия массового поражения, конструирования самолетов и ракет, можно упомянуть проектирование бесшумных подводных лодок, систем противоракетной обороны и целый ряд других проектов.

Известны оценки производительности вычислительных систем для решения различных задач. Считается, что для задач аэродинамики достаточно производительности в несколько петафлопс, для задач молекулярной динамики - 20 петафлопс, а для вычислительной космологии нужна производительность на уровне 10 эксафлопс (10 квинтиллионов флопс или $10 \cdot 10^{18}$ флопс). По оценкам специалистов появление компьютеров с производительностью в секстиллион операций в секунду (10^{21} флопс) ожидается к 2029 году.

Прогресс в области суперкомпьютеров можно наблюдать из публикующегося дважды в год рейтинга самых высокопроизводительных компьютеров, так называемого TOP 500. Производительность суперкомпьютеров стремительно растет. Если в 2000 году самый мощный

компьютер имел производительность 4 терафлопс, то в 30-й версии TOP 500, опубликованной в ноябре 2007 г., суперЭВМ с производительностью 4 Тфлопс вообще не попали в этот список. Сравнительно недавно суперкомпьютеры преодолели планку производительности в один терафлопс (триллион операций с плавающей запятой в секунду), а уже грядет эра петафлопс (квадриллион или 10^{15} операций с плавающей запятой в секунду). Первые петафлопсные суперкомпьютеры, по прогнозам экспертов, появятся уже в 2008 году.

Первое место в 30-й редакции списка TOP 500 (ноябрь 2007 г.), как и в предыдущей версии, занял суперкомпьютер IBM BlueGene/L. Его производительность на тесте Linpack выросла за полгода с 280,6 Тфлопс до 478,2 Тфлопс.

По спискам TOP 500, составляемым дважды в год ведущими экспертами, можно судить не только о текущем состоянии мировой вычислительной техники, но и о тенденциях в ее развитии. Количество кластерных систем в 30-й редакции TOP 500 продолжало расти и составило 406 против 373 в 29-й редакции рейтинга. Из коммуникационных технологий наиболее популярными остаются Gigabit Ethernet – 270 систем и InfiniBand – 121 система. Количество систем, входящих в TOP 500, построенных на процессорах Intel, увеличилось с 289 (июнь 2007 г.) до 354 (ноябрь 2007 г.).

Высокопроизводительные вычислительные системы – суперЭВМ не зря называют форпостом компьютерной техники. Однако в нашей стране прочность этой цитадели вызывает опасения. Еще сравнительно недавно, в 2000 году, признанный авторитет в области суперЭВМ академик В.С. Бурцев, главный конструктор знаменитого семейства МВК «Эльбрус», с горечью констатировал [16]: «Должной поддержки наша работа не находит. Еще раз подчеркну, сегодня в России практически свернуты все перспективные работы по суперЭВМ».

Однако в последнее время положение с суперкомпьютерными технологиями в России стало меняться к лучшему. Был создан Межведомственный суперкомпьютерный центр (МСЦ) РАН. В 2001-м году

были закончены технические испытания отечественного суперкомпьютера МВС 1000М, вышедшего на символический рубеж в триллион операций с плавающей запятой в секунду.

Успешно функционирует российско-белорусская программа по созданию и применению суперкомпьютеров. В ходе ее реализации были созданы суперкомпьютеры СКИФ К-500, СКИФ К-1000 и СКИФ Syberia. Изготовлен еще более мощный суперкомпьютер (60 Тфлопс) для Московского государственного университета. Надо отметить, что отечественные разработки относятся к классу суперЭВМ с кластерной архитектурой, представляющих собой объединение множества традиционных коммерчески доступных процессорных узлов с помощью стандартных сетевых решений. Известно, что суперЭВМ данного класса имеют существенные недостатки, связанные с относительно низкой скоростью процедур межпроцессорного обмена, ограниченной пропускной способностью сети передачи данных, необходимостью синхронизации множества взаимосвязанных последовательных процессов, каждый из которых выполняется на отдельном процессоре, и т.д. Все это приводит к тому, что высокую реальную производительность кластерные суперЭВМ демонстрируют, в основном, только при решении «слабосвязанных» задач, не требующих интенсивного обмена данными между процессорными узлами, в то время как при решении «сильносвязных» задач их реальная производительность не превышает 5-10% от декларируемой пиковой производительности системы [1]. При этом увеличение числа процессорных узлов в системе зачастую не только не повышает ее реальную производительность, а наоборот, ведет к снижению последней, поскольку организация параллельного вычислительного процесса требует больше времени, чем его исполнение.

Этих недостатков лишены реконфигурируемые многопроцессорные вычислительные системы, идея и приоритет создания которых принадлежат отечественным ученым [38,39,46]. В отличие от многопроцессорных вычислительных систем с «жесткой» архитектурой, в частности, кластерных

суперЭВМ, архитектура реконфигурируемых систем может изменяться в процессе ее функционирования. В результате у пользователя появляется возможность адаптации архитектуры вычислительной системы под структуру решаемой им задачи или, иными словами, создания в рамках универсальной среды проблемно-ориентированных многопроцессорных вычислительных структур. Проведенные исследования показали [50], что реализация данной концепции обеспечивает высокую реальную производительность многопроцессорной вычислительной системы, близкую к пиковой, на широком классе задач (в том числе при решении «сильносвязанных» задач), а также практически линейный рост производительности при увеличении числа процессоров в системе.

Концепция реконфигурируемых многопроцессорных вычислительных систем в нашей стране развивалась по двум основным направлениям - это однородные вычислительные среды и структуры [37,39,40,126] и многопроцессорные вычислительные системы с программируемой архитектурой [45,46,49]. Однородная вычислительная среда (ОВС) состоит из одинаково соединенных друг с другом универсальных элементов, каждый из которых программно настраивается на выполнение логической функции и функции соединения со своими соседями. При этом под универсальным вычислительным элементом, как правило, понимался простейший одноканальный процессор, выполняющий последовательную обработку разрядов поступающих данных и соединенный с четырьмя соседними такими же простейшими процессорами.

В отличие от однородных вычислительных сред, авторы концепции многопроцессорных вычислительных систем с программируемой архитектурой (МВС ПА) предложили строить процессорный узел реконфигурируемой системы в виде некоторого множества элементарных многоразрядных процессоров, объединенных полнодоступной коммутационной структурой [47]. Иными словами, процессорный узел такой вычислительной системы, который авторы называли макропроцессором, представлял собой некоторую

реконфигурируемую структуру нижнего уровня. Макропроцессоры соединялись между собой в реконфигурируемую вычислительную структуру верхнего уровня с помощью системного коммутатора. При этом, как и в случае ОВС, так и в случае с МВС с программируемой архитектурой, процесс решения задачи сводился к организации мультиконвейерных цепочек, отображающих информационный граф решаемой задачи [50,55]. Поэтому вычислительные системы данного типа можно отнести к классу реконфигурируемых мультиконвейерных вычислительных структур.

Реализация и широкое внедрение в практику идеи реконфигурируемых многопроцессорных вычислительных систем сдерживались отсутствием в нашей стране соответствующей этой концепции элементной базы. Появление же в конце 90-х годов XX века программируемых интегральных схем (ПЛИС) со сверхвысокой степенью интеграции, на базе которых стало возможным создание больших вычислительных полей, позволило в полной мере реализовать преимущества реконфигурируемых мультиконвейерных вычислительных структур. Проведенные исследования и разработки показали [51,82], что на основе ПЛИС возможно без привлечения больших финансовых средств создавать реконфигурируемые вычислительные структуры, существенно опережающие многопроцессорные системы с традиционной «жесткой» архитектурой по таким характеристикам как соотношения «реальная производительность/пиковая производительность» (эффективность) и «реальная производительность/объем» (компактность).

Следует отметить, что по этому пути уже идут и многие ведущие мировые производители [88,140]. Компания Сгау предлагает снабдить процессоры Opteron на двухпроцессорной платформе дополнительным модулем на базе ПЛИС. На этот модуль возлагается решение трудоемких для универсального процессора задач. Реконфигурируемые устройства в виде плат сопроцессоров для задач цифровой обработки сигналов и моделирования специализированных устройств разработаны фирмами Alpha Data Systems, Annapolis Micro System, Aptix, Nallatech и Virtual Computer Corporation.

Компанией Stretch предлагается реконфигурируемый процессор S5000, представляющий собой сочетание известной микропроцессорной архитектуры RISC с кристаллами ПЛИС. Разработанный компилятор автоматически выделяет в программе места, требующие интенсивных вычислений, и реализует их с помощью ПЛИС. Компания Star Bridge Systems одна из первых представила суперкомпьютер, который основан на реконфигурируемой технологии и построен на кристаллах ПЛИС фирмы Xilinx. В исследовательском центре Berkeley Wireless Research Center также разработана суперкомпьютерная система High-End Reconfigurable Computing System на кристаллах ПЛИС. По мнению разработчиков, использование кристаллов ПЛИС обеспечивает более чем на два порядка большую производительность систем в сравнении с производительностями систем с аналогичной потребляемой мощностью и стоимостью, реализованными на стандартных микропроцессорах [80].

Однако западные производители используют ПЛИС, как правило, только в качестве сопроцессоров к стандартным вычислительным узлам, реализованным на универсальных микропроцессорах. Концепция же построения реконфигурируемых мультиконвейерных вычислительных структур открывает более широкие перспективы использования ПЛИС в качестве элементной базы для создания больших вычислительных полей, в рамках которых могут создаваться проблемно-ориентированные вычислительные системы, адаптируемые под структуру решаемой задачи. Именно при таком подходе возможно обеспечить максимум реальной производительности системы при решении конкретной вычислительной задачи.

Настоящая монография посвящена теоретическим аспектам создания реконфигурируемых мультиконвейерных вычислительных структур различных типов и их практической реализации на базе ПЛИС сверхвысокой степени интеграции.

Монография состоит из введения и шести глав. Первая глава посвящена изложению общих принципов организации реконфигурируемых

мультиконвейерных вычислительных структур. Показывается преимущество реконфигурируемых вычислительных систем по сравнению с МВС с жесткой архитектурой и делается вывод о наибольшей эффективности применения систем данного класса при решении потоковых задач, то есть задач обработки больших массивов (потоков) данных по одному и тому же алгоритму. Анализируются различные способы решения потоковых задач, и показывается, что при мультиконвейерном способе вычислений обеспечивается минимальное время обработки массива (потока) данных. Рассматриваются различные способы организации реконфигурируемых вычислительных структур, реализующих мультиконвейерный вычислительный процесс. Показывается, что в зависимости от сложности операции, реализуемой на ступени мультиконвейера, вычислительные структуры могут быть классифицированы на два типа - мультимикроконвейерные вычислительные структуры и мультимакроконвейерные вычислительные структуры. В главе также рассматриваются идеология и способы организации структурно-процедурных мультиконвейерных вычислений в случае, когда аппаратных средств мультиконвейерной структуры не хватает для отображения всего информационного графа решаемой задачи.

Вторая глава монографии посвящена организации мультимикроконвейерных вычислительных структур на однородных средах. Рассматривается архитектура однородной вычислительной среды и приводится схема ее ячейки. Описываются организация вычислительного процесса в однородной вычислительной среде и процесс ее программирования на основе микропрограммных модулей, реализующих спецпроцессоры различных операций. Описываются примеры решения прикладных задач разных предметных областей на однородных средах. Рассматриваются реализация однородных вычислительных сред на СБИС, а также перспективы и технические особенности их реализации на кремниевой пластине.

Третья глава посвящена структурно-процедурным методам организации вычислений в мультимакроконвейерных вычислительных структурах.

При реализации структурно-процедурного метода информационный граф задачи разбивается на кортеж изоморфных информационно независимых или непосредственно зависимых подграфов. Подграф выбирается таким образом, чтобы ресурсов системы было достаточно для его структурной реализации. Представлена методика преобразования кортежа изоморфных подграфов в кадровую форму, которая естественным образом может быть отображена на архитектуру макроконвейерных вычислительных структур. Кадру соответствует подграф, через который следует поток операндов. Подобная организация параллельных вычислений обеспечивает детерминизм, то есть неизменность при различных запусках параллельной программы как порядка вызова кадров, так и порядка обработки операндов в кадре. Показаны методы преобразования различных типов информационных графов в кадровую форму. Приведены примеры преобразования прикладных задач и даны оценки эффективности структурно-процедурной организации вычислений.

В четвертой главе рассматриваются принципы построения модульно-наращиваемых мультиконвейерных вычислительных структур на универсальных базовых модулях. Из анализа современной элементной базы делается вывод о том, что мультиконвейерные вычислительные структуры целесообразно строить на основе ПЛИС. Предлагается структура базового модуля на основе ПЛИС, а также приводятся примеры реализации базовых модулей ряда поколений и систем на основе ПЛИС большой степени интеграции, в которых удалось выполнить основные положения концепции реконфигурируемых мультиконвейерных вычислительных структур.

Реализация базовых модулей на основе ПЛИС позволила перейти от макропроцессорной к макрообъектной архитектуре реконфигурируемых мультиконвейерных вычислительных структур, которая дает возможность существенно повысить эффективность использования вычислительного ресурса. Вопросы создания реконфигурируемых мультиконвейерных вычислительных структур на основе макрообъектной архитектуры рассматриваются в пятой главе монографии. Вводится понятие макрообъекта с

точки зрения структурной и программной реализаций. Рассматривается обобщенная структура макрообъекта. Приводятся примеры реализации макрообъектов в таких предметных областях как математическая физика, цифровая обработка сигналов, линейная алгебра. Описана структура универсального макрообъекта.

Схемотехническое программирование реконфигурируемых мультиконвейерных структур, построенных на основе ПЛИС, представляет собой достаточно трудоемкий процесс, требующий от пользователя специальных знаний и навыков. Поэтому основной целью создания комплекса программного обеспечения реконфигурируемых мультиконвейерных структур, описанного в шестой главе, было обеспечение удобства программирования прикладных задач на мультиконвейерной структуре и приближение этого процесса к традиционному программированию многопроцессорных систем кластерного типа. Описываются общая структура системного математического обеспечения реконфигурируемых мультиконвейерных структур и язык ассемблера для программирования структурно-процедурных вычислений. Особое внимание уделяется языку программирования высокого уровня COLAMO с неявным описанием параллелизма. Распараллеливание прикладных программ, написанных на языке COLAMO, достигается за счет объявления типа доступа к информационным массивам и индексацией их элементов. Представлены принципы трансляций с языка COLAMO, обеспечивающие синтез в едином языковом пространстве структурных, потоковых, процедурных и управляющих компонентов прикладных программ, реализованных в реконфигурируемых мультиконвейерных вычислительных структурах. Приводится описание среды разработки прикладных программ для реконфигурируемых мультиконвейерных вычислительных структур.

Глава 1. ПРИНЦИПЫ ОРГАНИЗАЦИИ РЕКОНФИГУРИРУЕМЫХ МУЛЬТИКОНВЕЙЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СТРУКТУР

1.1. Проблема производительности многопроцессорных систем

Многопроцессорные вычислительные системы (МВС) предназначены для решения сложных задач, требующих обработки огромных массивов информации, для чего от МВС требуется очень высокая производительность, без которой подобные задачи не могут быть решены вообще или для этого потребуется столь большое время, что полученное решение просто потеряет практический смысл. Существует несколько путей достижения высокой производительности многопроцессорных вычислительных систем.

В настоящее время успехи развития многопроцессорных вычислительных систем обеспечиваются в основном за счет роста технологических возможностей, в частности за счет уменьшения топологических размеров при изготовлении кремниевых микросхем, вследствие чего повышается плотность компоновки вентилях на одном кристалле и возрастает скорость работы процессоров. Интенсивно развиваются новые технологии для создания компьютеров двадцать первого века, в том числе оптические и квантовые (кубитные) технологии. Однако в этих направлениях пока ведутся лишь теоретические и экспериментальные исследования, которые еще далеки от промышленного воплощения [15].

Помимо технологических путей повышения производительности вычислительных систем существуют алгоритмические, программные и архитектурные методы. Алгоритмические методы сводятся к построению более эффективных математических методов решения задач. Программные методы состоят в разработке программ, обеспечивающих эффективное использование вычислительных систем, включая память, каналы связи и т.п.

Наконец, важнейшим направлением повышения производительности вычислительных систем являются архитектурные методы. Специалистами ведущих фирм мира за последние полвека были реализованы конвейерные,

векторные, векторно-конвейерные, матричные, тороидальные, гиперкубовые, иерархические, кластерные архитектуры и множество иных архитектур вычислительных систем [1,17,23,32,59,101,144]. Теоретики неоднократно предпринимали попытки классифицировать огромное многообразие вычислительных систем, разложить их «по полочкам». Пожалуй, наибольшую известность получила классификация, предложенная в середине 60-х годов прошлого века американским специалистом Дж. Флинном. По Флинну, все вычислительные системы делились на четыре отряда: системы ОКОД (одиночный поток команд и одиночный поток данных), системы ОКМД (одиночный поток команд и множественный поток данных), системы МКОД (множественный поток команд и одиночный поток данных), системы МКМД (множественный поток команд и множественный поток данных). Несмотря на всю привлекательность простоты, классификация Флинна была мало пригодна на практике: реальные вычислительные системы зачастую не вписывались в «флинновы ложа».

Большинство из существующих архитектур многопроцессорных вычислительных систем имеет органический недостаток: если на некоторых классах задач при данной архитектуре достигается производительность, близкая к пиковой, то при решении других классов задач производительность той же вычислительной системы может резко падать, уменьшаясь на порядок или даже на несколько порядков (рис.1.1).

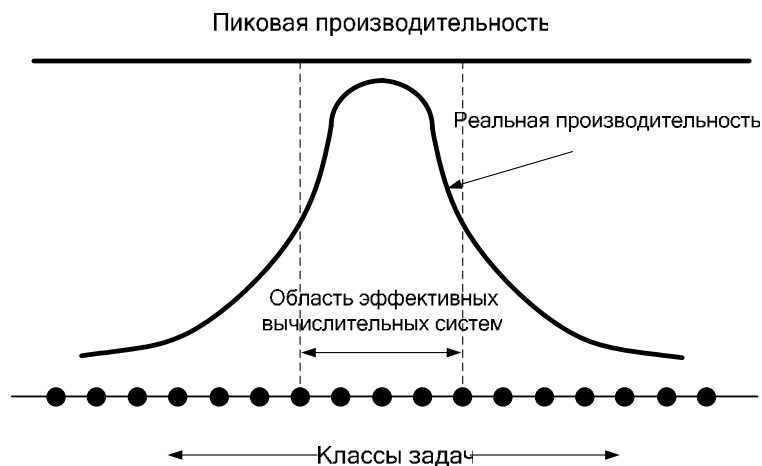


Рис. 1.1. Зависимость производительности системы от класса задач

Это является следствием неадекватности конкретной архитектуры многопроцессорной системы внутренней структуре решаемой задачи. Вообще, любой процесс, происходящий в реальной системе S , и любую задачу можно описать с помощью некоторого множества математических и логических зависимостей или в форме некоторого графа $G(Q, X)$ (рис. 1.2).

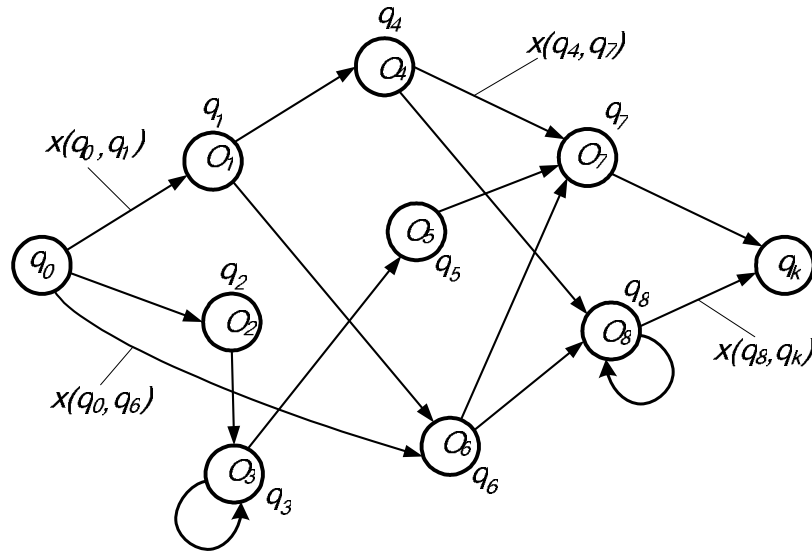


Рис. 1.2. Информационный граф задачи

Граф $G(Q, X)$ содержит множество вершин $q_i \in Q$, каждой из которых приписана некоторая операция O_i , принадлежащая множеству допустимых операций O . Дуги $x(q_i, q_{i+1}) \in X$ определяют последовательность выполнения операций, приписанных вершинам графа $G(Q, X)$, причем если две вершины q_i и q_{i+1} соединены дугой $x(q_i, q_{i+1})$, то это означает, что результат операции O_i является входным данным для операции O_{i+1} . Граф $G(Q, X)$ имеет также множество входных дуг $x(q_{0i}, q_i)$, определяющих источник входных данных, а также выходных дуг $x(q_i, q_k)$, определяющих приемник результатов ее решения. В дальнейшем граф $G(Q, X)$ будем называть *информационным графом* или *графом алгоритма решения задачи*.

При моделировании процессов, происходящих в системе S с помощью параллельной многопроцессорной вычислительной системы, в последней организуется соответствующий вычислительный процесс, который

описывается в большинстве случаев совершенно другим графом $G^*(Q^*, X^*)$ (рис. 1.3).

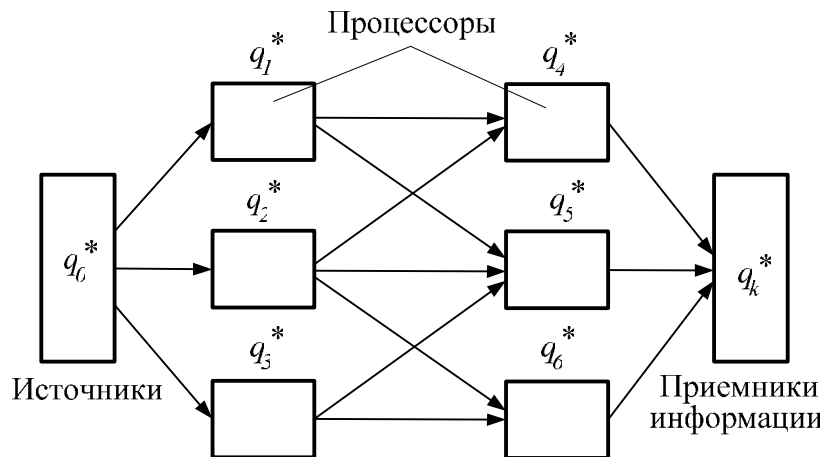


Рис. 1.3. Граф вычислительного процесса в системе

Здесь множество вершин Q^* определяется множеством процессоров вычислительной системы. Множество дуг X^* представляет собой множество каналов коммуникаций между процессорами. При этом входные и выходные дуги графа $G^*(Q^*, X^*)$ определяются каналами связи с источником входных и приемником выходных данных, в частности с памятью системы. Операции O_i из множества O реализуются в процессорах вычислительной системы в форме последовательных процедур обработки данных.

Как правило, если многопроцессорная вычислительная система имеет «жесткую» архитектуру, то графы G и G^* существенно отличаются один от другого. Различие между графами G и G^* объясняется тем фактом, что жесткая архитектура параллельной многопроцессорной вычислительной системы в большинстве случаев значительно отличается от структуры моделируемой системы S . В результате для моделирования системы S с помощью многопроцессорного компьютера с «жесткой» архитектурой в последнем приходится организовывать некоторый мультипроцедурный вычислительный процесс, который моделирует систему S , но структурно с ней не совпадает и описывается графом G^* , существенно отличным от графа G реальной моделируемой системы S .

В большинстве случаев граф G^* многопроцессорной системы с «жесткой» архитектурой значительно более сложен, чем исходный граф G моделируемой системы или решаемой задачи, и имеет более сложную систему внутренних коммуникационных каналов X^* , чем система информационных дуг X в графе G . Все это приводит к существенным потерям времени за счет операций распределения заданий и процедур между процессорами параллельной системы [99]. Снижение эффективности работы вычислительной системы происходит из-за потерь при передаче данных через промежуточные процессоры, из-за очередей и конфликтов при обращении процессоров к данным, хранящимся в памяти, и по многим другим причинам. В результате реальная производительность таких вычислительных систем оказывается существенно ниже пиковой, и, как отмечают сами создатели наиболее распространенных кластерных суперкомпьютеров, реальная производительность зачастую не превышает 10% от пиковой производительности системы [1]. Более того, зачастую при увеличении числа процессоров в таких вычислительных системах их реальная производительность не только не возрастает, а, наоборот, начинает снижаться (рис. 1.4).

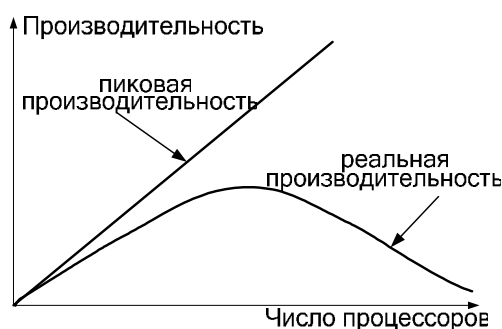


Рис. 1.4. Зависимость производительности систем с «жесткой» архитектурой от числа процессоров

Указанного недостатка вычислительных систем с «жесткой» архитектурой можно избежать, если обеспечить возможность реконфигурации архитектуры МВС таким образом, чтобы граф G^* вычислительного процесса

как можно ближе совпадал с графом G моделируемого процесса (или решаемой задачи). В этом случае пик производительности МВС можно будет «сдвигать», подстраивая его под решаемую в текущий момент времени задачу (рис. 1.5), а также обеспечить практически линейный рост производительности МВС при увеличении числа процессоров в системе (рис. 1.6).

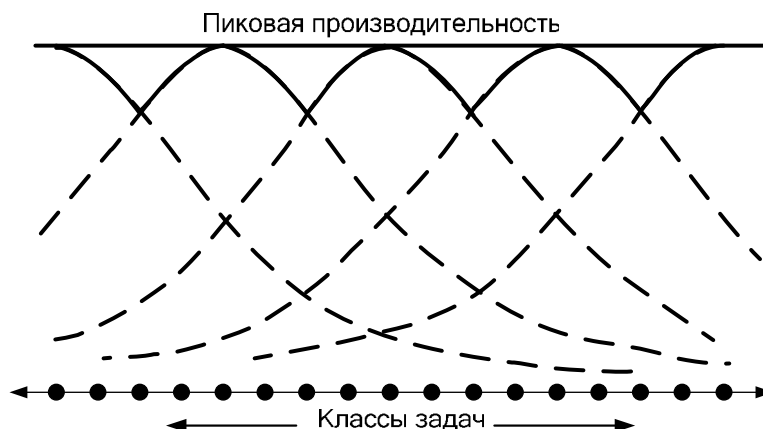


Рис. 1.5. Приближение реальной производительности системы к пиковой

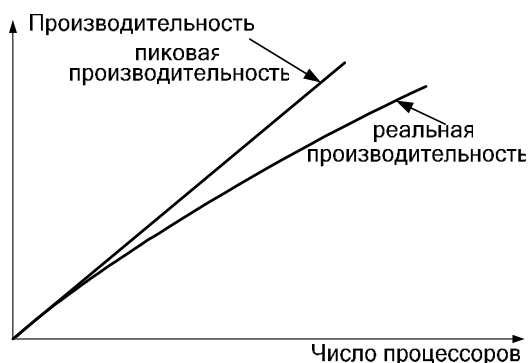


Рис. 1.6. Зависимость производительности вычислительной системы с реконфигурируемой архитектурой от числа процессоров

Данная основополагающая идея лежит в основе всех реконфигурируемых МВС различных типов, несмотря на их кажущиеся отличия. Тем не менее, рассмотрим более подробно различные подходы к организации реконфигурируемых МВС.

Первыми реальными реконфигурируемыми вычислителями можно считать аналоговые вычислительные машины и их цифровые аналоги – так называемые цифровые дифференциальные анализаторы [44]. В этих устройствах изменение (реконфигурация) архитектуры связей между

обрабатывающими узлами для обеспечения адекватности вычислительного процесса решаемой задачи осуществлялось вручную с помощью коммутационной панели.

В 1966 году в Новосибирске вышла в свет книга Э.В. Евреинова и Ю.Г. Косарева «Однородные универсальные системы высокой производительности» [37], в которой авторы предлагали строить вычислительную систему в виде «однородной универсальной вычислительной системы, состоящей из одинаковых и одинаково соединенных друг с другом универсальных вычислительных машин и обладающей возможностью программного изменения структуры». Рассматривались также *однородные вычислительные среды*, состоящие из одинаковых и одинаково соединенных друг с другом простейших универсальных элементов, программно настраиваемых на выполнение любой функции из полного набора логических функций, памяти и любого соединения со своими соседями.

Концепция создания однородных вычислительных систем и сред глубоко прорабатывалась и в московском Институте проблем управления РАН под руководством И.В. Прангишвили. В 1967 г. была опубликована интересная монография И.В. Прангишвили и группы его сотрудников с примечательным названием «Микроэлектроника и однородные структуры для построения логических и вычислительных устройств» [95], в которой практическая реализация однородных вычислительных структур напрямую увязывалась с только что зарождавшейся микроэлектроникой. В 1972 г. вышла книга Э.В. Евреинова и И.В. Прангишвили «Цифровые автоматы с настраиваемой архитектурой» [38]. Обобщая результаты исследований, авторы формулируют основные принципы создания однородных вычислительных структур:

- параллельность выполнения большого числа операций;
- конвейерность обработки информации;
- переменность логической структуры;
- однородность элементов и связей между ними.

Концепция однородных вычислительных сред и структур нашла практическое применение в ряде разработок конца 70-х годов. На Тбилисском предприятии «Элва» некоторое время выпускалась ЭВМ ПС-300, спроектированная в Институте проблем управления РАН под руководством И.В. Прангишвили. Процессорный узел ПС-300 был выполнен на однородной вычислительной среде, позволяющей посредством настройки реализовывать широкий спектр операций [97]. Здесь только следует отметить, что ячейки однородной среды собирались из микросхем серии К-155 – «рассыпухи», как они именовались специалистами, так что говорить об эффективности подобных систем не приходилось. Бортовые вычислительные устройства Научно-производственным объединением «Астрофизика» реализовывались уже на специальных микросхемах однородной вычислительной среды (ОВС), которые были освоены в серийном производстве рижским заводом «Альфа». Главным конструктором этих вычислительных устройств на ОВС был М.П. Богачев.

В дальнейшем, с середины 80-х годов, центр исследований в области однородных вычислительных сред переместился в г. Львов, где были проведены крупные работы в области программного обеспечения и разработки перспективных архитектур однородных вычислительных сред [101,104,108,125,131]. В НИИ ТТ (г. Зеленоград) было налажено производство отказоустойчивого кристалла ОВС, содержащего 130 элементарных ячеек.

В 80-х годах в американских изданиях появились статьи о так называемых систолических структурах [68]. Систолическая обработка данных подавалась как доселе неведомый прием в организации вычислений. Однако систолическая обработка задолго до публикации [145] естественным образом реализовывалась на однородных средах [10]. Описанные систолические структуры в значительной степени уступали средам в гибкости и эффективности и могли рассматриваться как весьма частный случай однородных сред.

Другим направлением реконфигурируемых вычислительных систем, развиваемым в нашей стране, стала концепция многопроцессорных вычислительных систем с программируемой архитектурой, впервые

сформулированная А.В. Каляевым в 70-х годах прошлого века [44,46]. В отличие от разработчиков однородных вычислительных сред, которые реализовывали вычислительный узел среды в виде простейшего одноканального процессора, соединенного с соседними процессорами по принципу близкодействия, А.В. Каляев предложил строить процессорный узел вычислительной системы в виде некоторой элементарной структуры с реконфигурируемой архитектурой. Процессорные узлы такого типа были названы *макропроцессорами*. Каждый макропроцессор содержал набор элементарных процессоров, объединенных некоторой полнодоступной коммутационной системой [48]. Отметим, что современная концепция мультиядерных процессоров, появившаяся в начале нового века и представляемая как нечто новое, по сути, не отличается от макропроцессоров А.В. Каляева, предложенных 30 лет назад. С помощью коммутационной системы верхнего уровня макропроцессоры конфигурируются в мультиконвейерную структуру, адекватную графу алгоритма решаемой задачи. Системы такого типа имеют два уровня конвейеризации – на уровне макропроцессоров и на уровне элементарных процессоров внутри каждого макропроцессора. Это обеспечивает гибкость системы и упрощает процесс ее программирования. Среди систем, построенных с программируемой архитектурой, следует упомянуть такие системы как ЕС-2703 (1985 г.), «Трасса» (1991 г.) и «Модуль-8» (1998 г.).

Однако, несмотря на большое число исследований в области реконфигурируемых МВС, до сих пор они не нашли широкого применения. В первую очередь это объясняется специфичностью и достаточной сложностью программирования таких систем. Действительно, процедура программирования таких систем, требующая программирования не только каждого отдельного процессорного элемента, но и структуры связей между ними в соответствии с графом алгоритма решаемой задачи, требует наличия нестандартных средств программирования, а также нестандартного мышления программиста. Все это достаточно сложно и, как правило, требует больших временных затрат.

Поэтому использовать реконфигурируемые МВС для решения часто меняющихся задач с одиночным вектором входных данных оказывается малоэффективно. Но есть класс задач, при решении которых преимущества реконфигурируемых мультиконвейерных МВС проявляются в полной мере. Это так называемые потоковые задачи, т.е. задачи обработки больших массивов (потоков) данных по одному и тому же алгоритму. Примерами таких задач могут служить задачи математической физики, цифровой обработки сигналов, криптографии и т.д. Именно решению такого рода задач с помощью реконфигурируемых мультиконвейерных вычислительных структур, уделяется основное внимание в настоящей монографии.

1.2. Потоковые задачи и способы их решения

В общем случае потоковую задачу можно сформулировать в следующем виде. Предположим, что существует некоторое упорядоченное множество векторов данных $D_i < d_1^i, d_2^i, \dots, d_k^i, >$, ($i = 1, 2, \dots, N$), каждый элемент которого должен быть обработан по фиксированному алгоритму, представляемому в виде графа $G(Q, X)$ (рис. 1.7).

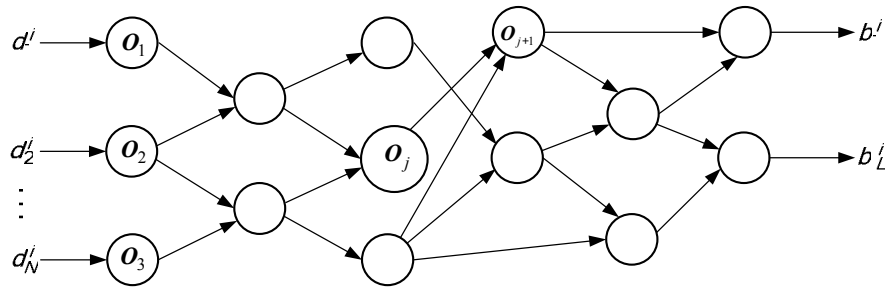


Рис. 1.7. Граф алгоритма

При этом каждой вершине $q_j \in Q$ графа $G(Q, X)$ поставлена в соответствие некоторая операция O_j , принадлежащая множеству допустимых операций O . Любая дуга $x(q_j, q_{j+1}) \in X$ определяет, что результат операции O_j является входным данным для операции O_{j+1} , поставленной в соответствие вершине q_{j+1} .

Задача потоковой обработки заключается в преобразовании кортежа (потока) векторов входных данных D_i ($i = 1, 2, \dots, N$) в кортеж (поток) векторов выходных данных $B_i < b_1^i, b_2^i, \dots, b_L^i >$ ($i = 1, 2, \dots, N$) в соответствии с графом алгоритма $G(Q, X)$.

В простейшем случае для решения сформулированной выше задачи можно использовать последовательную ЭВМ с фон-неймановской архитектурой (рис. 1.8), основным элементом которой является процессор P .

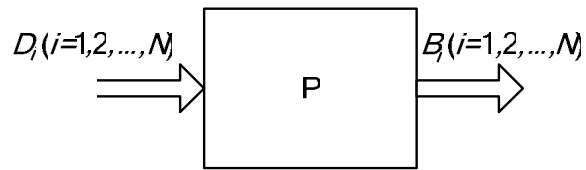


Рис. 1.8. ЭВМ с фон-неймановской архитектурой

Для этого процессор P предварительно необходимо запрограммировать на последовательное выполнение всех операций O_j ($j = 1, 2, \dots, M$), соответствующих вершинам графа алгоритма $G(Q, X)$, и затем подать на вход ЭВМ последовательность векторов D_i ($i = 1, 2, \dots, N$).

Время обработки N векторов входных данных при этом составит

$$T_{\text{реш}} = N \sum_{i=1}^M f(O_i) \tau, \quad (1.1)$$

где $M = |Q|$ – число вершин на графе $G(Q, X)$;

$f(O_i)$ – число тактов работы ЭВМ при выполнении операции O_i , соответствующей вершине графа $G(Q, X)$;

τ – продолжительность такта.

Критерием оценки эффективности обработки является условие $T_{\text{реш}} < T_{\text{PB}}$.

Здесь T_{PB} – требуемое время обработки всех векторов входных данных. Если неравенство истинно, то используемый подход можно признать эффективным и следует использовать как наиболее простой с точки зрения реализации. Однако при достаточно больших значениях основных параметров

поточковой задачи (числе входных векторов N , размере каждого вектора, числе вершин на графе вычислений) указанное условие, как правило, выполняться не будет.

Очевидным путем сокращения времени $T_{реш}$ обработки кортежа входных векторов является распараллеливание процесса обработки [20-23]. В простейшем случае параллельный способ подразумевает наличие C процессоров P_i ($i = 1, 2, \dots, C$), каждый из которых может работать независимо от других процессоров. Очевидно, если каждый из процессоров P_i ($i = 1, 2, \dots, C$) запрограммировать на реализацию графа алгоритма $G(Q, X)$ и разбить множество входных векторов D_i ($i = 1, 2, \dots, N$) на $\lceil N/C \rceil$ непересекающихся подмножеств, то каждое из этих подмножеств входных векторов может быть обработано на процессоре P_i независимо, то есть параллельно с другими подмножествами (рис. 1.9).

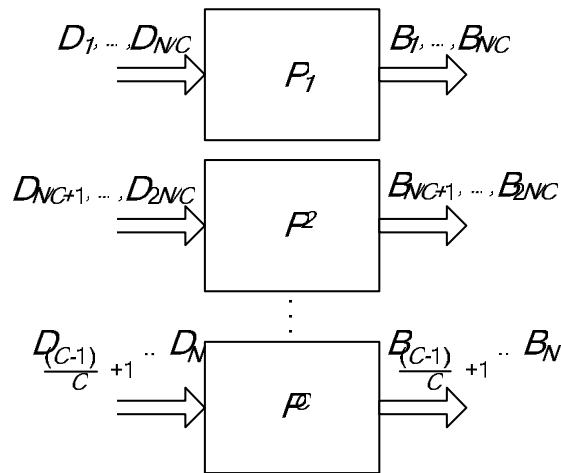


Рис. 1.9. Параллельный способ обработки векторов входных данных

В этом случае время решения, то есть время обработки всего множества векторов входных данных, будет составлять

$$T_{реш} = \left\lceil \frac{N}{C} \right\rceil \left[\sum_{i=1}^M f(O_i) \tau \right]. \quad (1.2)$$

Очевидно, что время обработки потока сокращается в C раз по сравнению с последовательной обработкой всех входных векторов на одном процессоре.

Сократить время решения можно и другим способом, который принято называть *конвейерной обработкой*. Поясним суть данного метода на примере. Граф алгоритма $G(Q, X)$ можно разделить на непересекающиеся подграфы $G_1(Q_1, X_1), G_2(Q_2, X_2), \dots, G_H(Q_H, X_H)$, как показано на рис. 1.10.

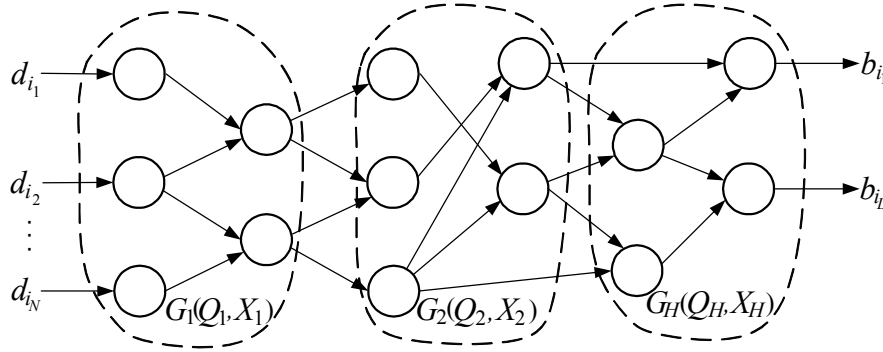


Рис. 1.10. Разбиение графа алгоритма $G(Q, X)$ на непересекающиеся подграфы

Предположим, что в системе имеется H процессоров P_i , ($i = 1, 2, \dots, H$), соединенных в цепочку, как показано на рис. 1.11, причем каждый из процессоров P_i запрограммируем на реализацию соответствующего подграфа $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, H$) общего графа алгоритма $G(Q, X)$.

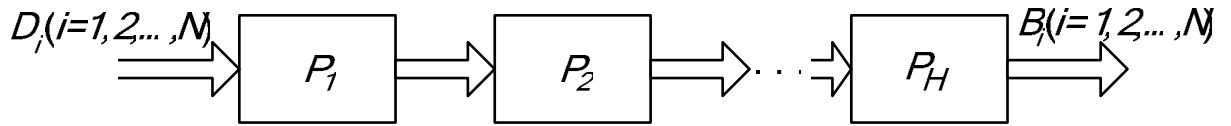


Рис. 1.11. Соединение процессоров в конвейер

Если на вход такой цепочки процессоров подать одиночный входной вектор D_i , то время его обработки, то есть время получения выходного вектора B_i будет составлять

$$T_{\text{реш}} = \sum_{i=1}^H \sum_{j=1}^{M_i} f(O_j^i) \tau, \quad (1.3)$$

где M_i – число вершин в подграфе $G_i(Q_i, X_i)$;

O_j^i – операция, приписанная j -й вершине подграфа $G_i(Q_i, X_i)$.

Учитывая, что $H \times M_i = M$, получим выражение (1.2) в следующем виде:

$$T_{\text{печ}} = \sum_{i=1}^M f(O_i) \tau. \quad (1.4)$$

Таким образом, время обработки одиночного вектора данных на цепочке процессоров равно времени обработки одиночного вектора данных на одном процессоре. Однако если на вход той же цепочки процессоров подать последовательность из N векторов входных данных D_i ($i = 1, 2, \dots, N$), то время их обработки будет значительно меньше времени обработки той же последовательности на одном процессоре. Действительно, как только первый вектор D_1 будет обработан на первом процессоре P_1 цепочки согласно графу алгоритма $G_1(Q_1, X_1)$ и результаты обработки будут переданы на вход процессора P_2 , то процессор P_1 может приступить к обработке второго вектора D_2 множества входных векторов. После обработки второго вектора процессор P_1 может приступить к обработке вектора D_3 , согласно графу алгоритма $G_1(Q_1, X_1)$, в то время как процессор P_2 по завершении операций, приписанных ему графом $G_2(Q_2, X_2)$ над результатом обработки вектора D_1 в процессоре P_1 , может переходить к выполнению тех же операций над вектором D_2 . Если принять, что время обработки входного вектора в процессоре P_i , согласно графу алгоритма $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, H$), равно $T_{\text{печ}}^i = \sum_{j=1}^{M_i} f(O_j^i) \tau$, то через время $T_1 = \sum_{i=1}^H T_{\text{печ}}^i$ на выходе цепочки появится первый вектор B_1 выходных данных и далее через промежутки времени $\Delta T = \max_{i=1, \dots, F} (T_{\text{печ}}^i)$ все остальные искомые выходные вектора $B_2, \dots, B_N$.

Таким образом, общее время обработки множества из N векторов данных в такой цепочке процессоров будет составлять

$$\begin{aligned} T_{\text{печ}} &= \sum_{i=1}^H T_{\text{печ}}^i + (N-1) \cdot \max_{i=1, \dots, H} (T_{\text{печ}}^i) \approx \\ &\approx H \cdot \max_{i=1, \dots, H} (T_{\text{печ}}^i) + (N-1) \cdot \max_{i=1, \dots, H} (T_{\text{печ}}^i) = (N+H-1) \cdot \Delta T, \end{aligned} \quad (1.5)$$

где $\Delta T = \max_{i=1, \dots, H} (T_{\text{печ}}^i) = \max_{i=1, \dots, H} \left(\sum_{j=1}^{M_i} f(O_j^i) \cdot \tau \right)$.

При достаточно большом значении числа N величиной $(H-1)$ можно пренебречь, вследствие чего значение $T_{\text{реш}}$ можно приблизительно определить как

$$T_{\text{реш}} \approx N \cdot \Delta T \approx N \cdot \frac{\sum_{j=1}^M f(O_j) \cdot \tau}{H}, \quad (1.6)$$

то есть по сравнению с обработкой множества из N векторов данных на одном процессоре (см. (1.1.)) в данном случае время решения сокращается приблизительно в H раз, где H – число процессоров в цепочке.

Обычно принято такую цепочку процессоров называть *конвейером*, каждый процессор цепочки – *ступенью конвейера*, а общее число H процессоров в цепочке – *длиной конвейера*.

Как видно из выражения (1.5), время обработки массива из N данных на конвейере процессоров определяется величиной $\Delta T = \max_{i=1, \dots, H} (T_{\text{реш}}^i)$, т.е. временем обработки данных на самой «медленной» ступени конвейера. В дальнейшем величину, обратную величине ΔT , будем называть темпом обработки данных в конвейере. Чем больше темп работы конвейера, тем меньшее время потребуется для обработки массива векторов входных данных. Понятно, что для достижения максимального темпа работы конвейера, разбиение графа алгоритма $G(Q, X)$ на подграфы $G_i(Q_i, X_i)$ желательно осуществлять таким образом, чтобы $T_{\text{реш}}^i \approx T_{\text{реш}}^j$ ($i = 1, 2, \dots, H; j = 1, 2, \dots, H$).

Существенным преимуществом конвейерного способа обработки является сокращение числа входных и выходных информационных каналов по сравнению с параллельным, поэтому в ряде систем конвейерные способы обработки информации получили широкое распространение. Однако если предел степени параллелизма для параллельного способа обработки информации ограничивается числом N – размерностью кортежа векторов, то конвейерный способ ограничивается числом M – числом операций на графе алгоритма задачи, которое, как правило, значительно меньше N .

Для дальнейшего сокращения времени обработки входного потока векторов данных целесообразно использовать совмещение параллельного и конвейерного способов обработки. Для этого необходимо организовать C параллельно работающих конвейеров длиной H , каждый из которых обрабатывает некоторое подмножество векторов входных данных D_i (рис. 1.12).

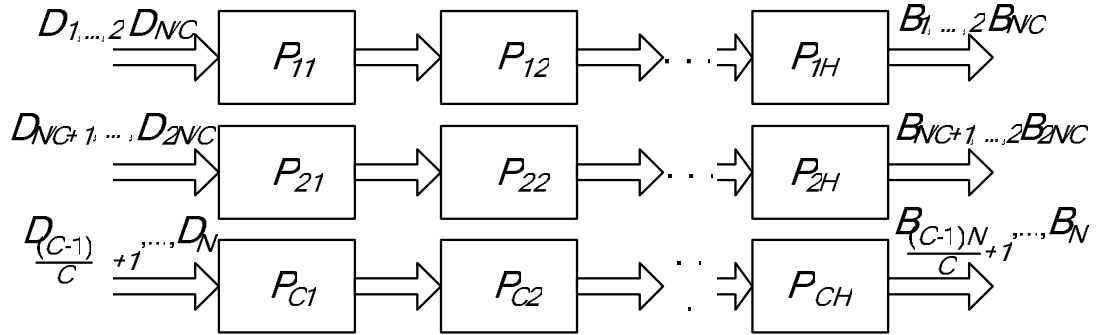


Рис. 1.12. Совмещение параллельного и конвейерного способов обработки

При таком способе организации вычислений время обработки потока из N векторов данных будет составлять:

$$T_{\text{реш}} = \left(\left\lceil \frac{N}{C} \right\rceil + H - 1 \right) \Delta T. \quad (1.7)$$

При большом значении N величиной $(H-1)$ можно пренебречь, поэтому

$$T_{\text{реш}} \approx \frac{N}{C} \cdot \Delta T \approx \frac{N}{C} \cdot \max_{i=1, \dots, H} \left(\sum_{j=1}^{M_i} f(O_j) \cdot \tau \right) \approx \frac{N}{C \cdot H} \cdot \sum_{j=1}^M f(O_j) \cdot \tau. \quad (1.8)$$

1.3. Принципы мультиконвейерной обработки потока данных

Выше при описании принципов конвейерной обработки подразумевалось, что процессор P_i , расположенный на i -й ступени конвейера, последовательно выполняет все операции O_j^i ($j=1, 2, \dots, M_i$), приписанные вершинам подграфа $G_i(Q_i, X_i)$. При этом, поскольку темп обработки данных в конвейере «привязан» к наиболее «медленной» ступени, последовательная обработка данных на ступенях конвейера приводит к снижению этого темпа и соответственно к

увеличению общего времени $T_{\text{реи}}$ обработки всего множества векторов входных данных.

Очевидный путь увеличения темпа обработки информации в конвейере – это распараллеливание обработки на каждой его ступени. В предельном случае можно представить i -ю ступень конвейера в виде M_i параллельно работающих процессоров P_{ij} ($j = 1, 2, \dots, M_i$), каждый из которых реализует операцию, соответствующую операционной вершине $q_j \in Q_i$ подграфа $G_i(Q_i, X_i)$ (рис. 1.13).

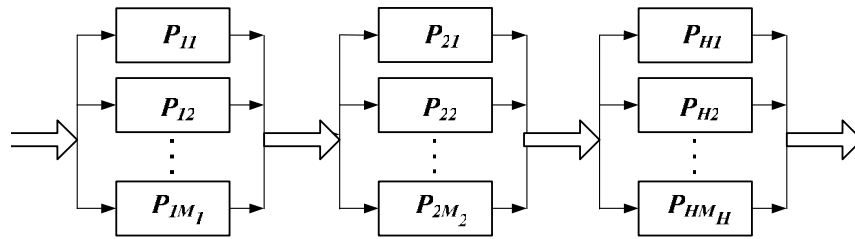


Рис. 1.13. Представление i -й ступени конвейера в виде M_i параллельно работающих процессоров P_{ij} ($j = 1, 2, \dots, M_i$)

Время обработки потока из N векторов в этом конвейере будет составлять

$$T_{\text{реи}} = (N + H - 1) \cdot \Delta T, \quad (1.9)$$

причем в данном случае

$$\Delta T = \max_{i=1, \dots, H} (T_{\text{реи}}^i) = \max_{i=1, \dots, H} \left(\max_{j=1, \dots, M_i} f(O_j^i) \cdot \tau \right) = \max_{i=1, \dots, M} f(O_i) \cdot \tau,$$

где $\max_{i=1, \dots, M} f(O_i)$ – наиболее длительная операция, приписанная вершинам графа $G(Q, X)$.

При большом значении N , а также принимая, что $M_i \approx \frac{M}{H}$, выражение (1.9) можно записать в следующем приближенном виде:

$$T_{\text{реи}} = N \frac{\sum_{j=1}^M f(O_j) \cdot \tau}{H \cdot M / H} = \frac{N}{M} \sum_{j=1}^M f(O_j) \cdot \tau. \quad (1.10)$$

Таким образом, время $T_{\text{реи}}$ в данном случае по сравнению с обычным конвейером (1.6) сокращается приблизительно в M/H раз, где M – число вершин в графе $G(Q, X)$, H – число ступеней конвейера.

Но схема на рис. 1.13 будет эффективна только в том случае, если все вершины подграфа $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, H$) информационно независимы и могут быть реализованы полностью параллельно (одновременно). Разумеется, можно разбить подграфы $G_i(Q_i, X_i)$ на информационно-независимые подграфы, однако степени распараллеливания каждого вновь образованного подграфа будут различны, в результате чего понизится темп обработки в ступенях конвейера и, как следствие, всего конвейера.

В общем же случае операционные вершины подграфов $G_i(Q_i, X_i)$ являются информационно зависимыми, т.е. некоторая вершина $q_j \in Q_i$ не может быть реализована до тех пор, пока не получен результат реализации вершины $q_{j-1} \in Q_i$ (рис. 1.10). Поэтому максимальный темп конвейерной обработки и, как следствие, минимальное общее время $T_{\text{реи}}$ обработки всего множества входных данных можно обеспечить в случае, если структура связей между ступенями конвейера будет полностью адекватна топологии связей вершин в графе алгоритма $G(Q, X)$. Такой способ организации конвейерных вычислений называется *структурным* [43].

Действительно, максимальный темп обработки можно обеспечить, если каждой операционной вершине q_i ($i = 1, 2, \dots, M$) графа-алгоритма $G(Q, X)$ поставить в соответствие свой процессорный элемент P_i ($i = 1, 2, \dots, M$) и обеспечить связи между P_i согласно топологии информационных дуг графа $G(Q, X)$ (рис. 1.14). Если на вход такой вычислительной структуры последовательно подавать векторы D_i ($i = 1, 2, \dots, N$) входного множества, то время их обработки будет составлять

$$T_{\text{реи}} = (N + H_{\text{max}} - 1) \cdot \Delta T,$$

где H_{max} — число вершин на критическом пути в графе $G(Q, X)$, т.е. пути

между входной и выходной вершиной, для которого значение $\sum_{j=1}^K f(O_j) \cdot \tau$

максимально;

$O_j (j = 1, 2, \dots, K)$ – операции, приписанные вершинам критического пути;
 $\Delta T = \max_{i=1, \dots, H_{\max}} (f(O_j)) \cdot \tau$ – время выполнения наиболее длительной операции, приписанной вершинам критического пути.

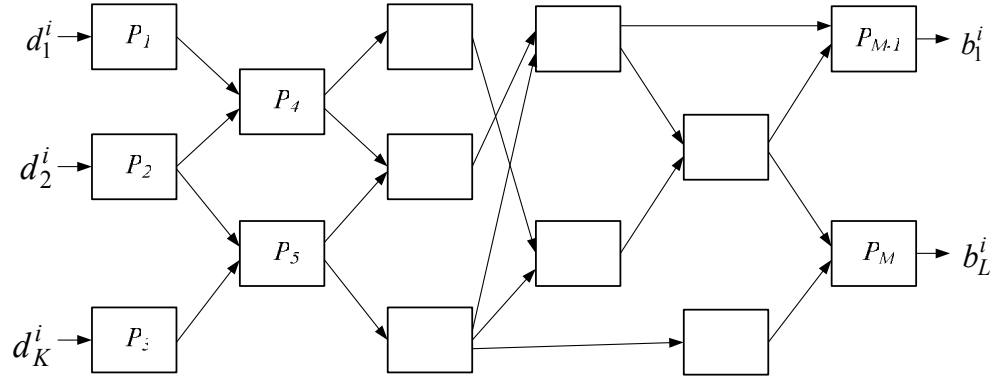


Рис. 1.14. Обеспечение максимального темпа конвейерной обработки данных

При большом N , а также учитывая, что

$$\Delta T = \max_{i=1, 2, \dots, H_m} f(O_j) \approx \frac{\sum_{j=1}^M f(O_j)}{M},$$

получаем

$$T_{\text{пер}} = \frac{N}{M} \sum_{j=1}^M f(O_j) \cdot \tau. \quad (1.11)$$

Таким образом, при структурном способе организации вычислений обеспечивается минимальное время обработки массива векторов данных по сравнению со всеми рассмотренными выше способами организации вычислительной системы (см. (1.1), (1.2), (1.6), (1.8), (1.10)).

Можно заметить, что вычислительная структура, представленная на рис.1.14, сочетает в себе как параллельный, так и конвейерный способ обработки информации, поскольку входные данные обрабатываются одновременно по различным конвейерным цепочкам процессоров. Конвейерную структуру такого типа в дальнейшем будем называть *мультиконвейерной вычислительной структурой (МКВС)*.

Если граф алгоритма решения задачи $G(Q, X)$ остается неизменным на всем периоде функционирования вычислителя, то в этом случае для решения

задачи целесообразно создать специализированную мультиконвейерную вычислительную систему.

Очевидно, что для создания вычислителя, с помощью которого можно решать различные потоковые задачи, отличающиеся графом алгоритма $G(Q, X)$, такой подход неприемлем. В данном случае необходимо обеспечить возможность изменения (реконфигурации) архитектуры информационных связей между процессорами для отображения графов алгоритмов с различной топологией. Мультиконвейерный вычислитель, обладающий такими возможностями, в дальнейшем будем называть мультиконвейерной структурой с реконфигурируемой архитектурой или *реконфигурируемой мультиконвейерной вычислительной структурой*.

Понятно, что для обеспечения возможности реконфигурации архитектуры мультиконвейерного вычислителя в его состав необходимо ввести некую коммутационную систему (КС), с помощью которой могут быть организованы различные информационные связи между процессорами P_i ($i = 1, 2, \dots, M$) (рис. 1.15). Рассмотрим несколько подходов к построению такой коммутационной системы.

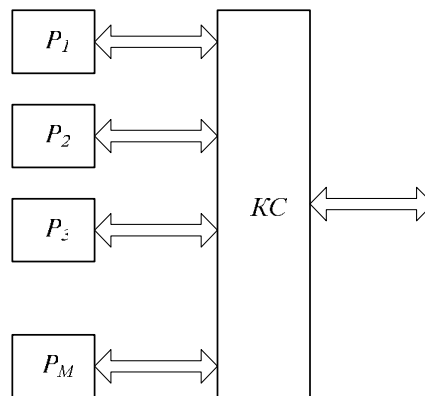


Рис. 1.15. Схема реконфигурируемого мультиконвейерного вычислителя

Простейший подход – это использование некоторого коммутатора, обеспечивающего полnodоступную информационную связь между всеми процессорами структуры [59]. Преимущество такого подхода заключается в возможности организации произвольных мультиконвейерных структур на его основе. Однако, поскольку аппаратные затраты на создание такого коммутатора

будут пропорциональны величине M^2 , где M – число процессоров в структуре, то при большом значении M такой подход будет трудно реализуем.

Можно предложить иной подход, в отличие от предыдущего не требующий больших аппаратных затрат. Суть этого подхода заключается в создании мультиконвейера с жесткой архитектурой и обеспечении коммутационных возможностей за счет придания каждому процессору дополнительных функций транзитной передачи информации. Например, на рис. 1.16 показана организация мультиконвейерного вычислителя с «жесткой» матричной архитектурой.

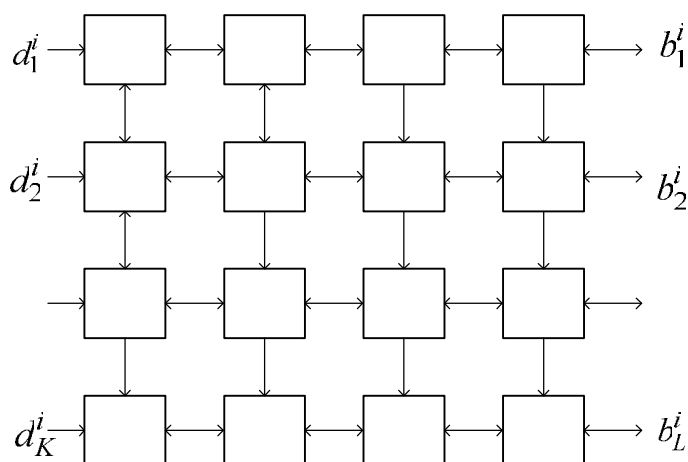


Рис. 1.16. Организация мультиконвейерного вычислителя с матричной архитектурой

При этом каждый процессор P_i ($i = 1, 2, \dots, M$) в такой структуре, кроме функций по реализации операций из множества O , должен обладать дополнительной функцией транзита данных, поступающих на его вход и на любой из выходов без какой-либо обработки.

В принципе, в такую матричную мультиконвейерную структуру можно отобразить любой граф алгоритма $G(Q, X)$, как, например, показано на рис. 1.17, но при этом значительная часть процессоров будет заниматься не эффективной работой, а транзитом информации между другими процессорами структуры.

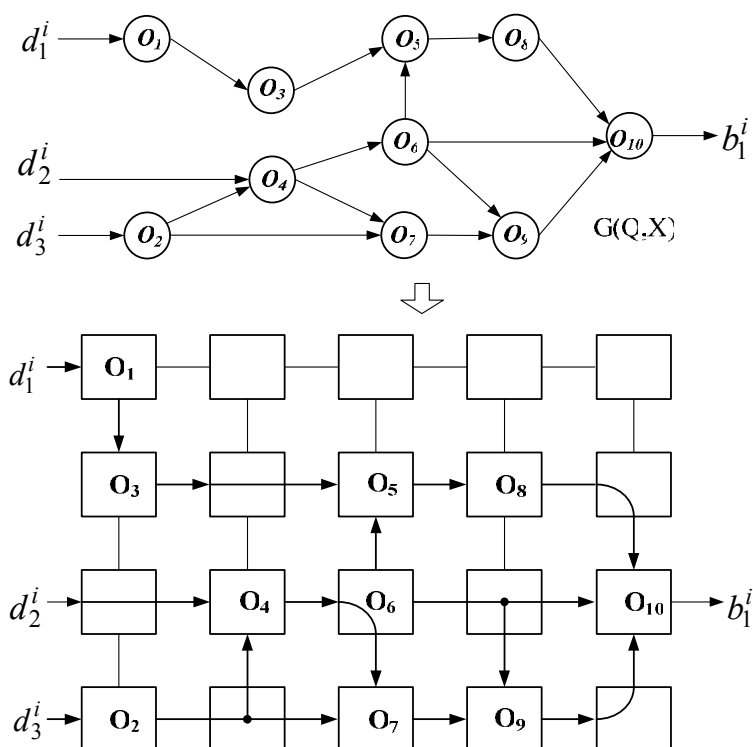


Рис. 1.17. Отображение графа $G(Q, X)$ в матричную мультиконвейерную структуру

Понятно, что такой процесс отображения графа $G(Q, X)$ в мультиконвейерную структуру с жесткой архитектурой достаточно сложен. При этом необходимо таким образом разместить операционные вершины графа $G(Q, X)$ по процессорам, чтобы обеспечить их максимально эффективное использование, т.е. минимизировать число процессоров, реализующих лишь только транзитные функции.

Преимуществом рассмотренного подхода является отсутствие сложного коммутатора, но, в то же время, подход обладает двумя существенными недостатками. Во-первых, хотя в структуре коммутатор как таковой отсутствует, его функции распределяются между процессорами, в результате чего часть процессоров используется в качестве коммутирующих узлов, что в целом снижает эффективность использования аппаратных ресурсов структуры. Во-вторых, транзит информации через процессоры вносит дополнительные временные задержки при прохождении данных через мультиконвейер и, соответственно, снижает темп конвейерной обработки, тем самым, увеличивая общее время обработки всего потока.

Можно предложить комбинированный подход, сочетающий оба рассмотренных ранее варианта. Идея этого подхода заключается в каскадном построении мультиконвейера (рис. 1.18). При этом каждый каскад представляет собой некоторое подмножество процессоров P и коммутатора K_i , с помощью которого обеспечивается полнодоступная коммутация между процессорами i -го и $(i+1)$ -го ($i = 1, 2, \dots, H$) каскадов, а также возможность прямой передачи информации на входы коммутатора K_{i+1} следующего каскада.

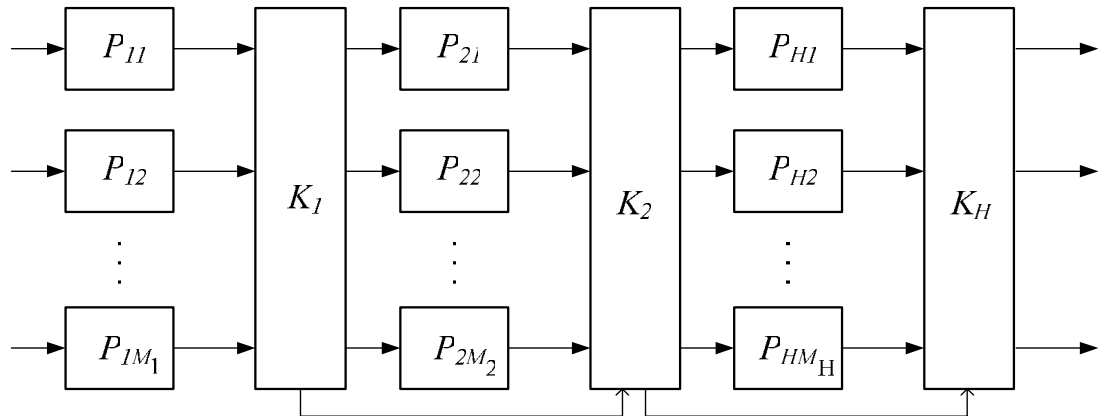


Рис. 1.18. Схема каскадного построения мультиконвейера

При использовании такого подхода по сравнению с первым подходом существенно снижаются аппаратные затраты на реализацию коммутаторов. Действительно, аппаратные затраты для создания H каскадов коммутаторов в данном случае будут пропорциональны величине

$$H \cdot M_i^2 \approx H \cdot \left(\frac{M}{H} \right)^2 = \frac{M^2}{H},$$

где M_i – число процессоров P на i -м каскаде мультиконвейера, т.е. будут в H раз меньше, чем для коммутатора, обеспечивающего полнодоступную связь между всеми процессорами структуры. В то же время, по сравнению со вторым подходом, при использовании каскадной организации мультиконвейера существенно увеличивается полезная нагрузка процессоров, поскольку коммутационные функции с процессоров перекладываются на каскады коммутаторов.

1.4. Мультимикро- и мультимакроконвейерные вычислительные структуры

Ранее были рассмотрены базовые принципы создания мультиконвейерных вычислительных структур. При этом не оговаривалось, что следует понимать под операцией $O_j \in \mathbf{O}$, реализуемой на ступени мультиконвейера. От сложности этой операции зависят не только требования к вычислительной мощности ступени мультиконвейера, но и темп работы мультиконвейера, а также эффективность использования того или иного способа коммутации процессорных ступеней друг с другом. Рассмотрим эту проблему более подробно.

Предположим, что задан некоторый граф алгоритма $G(Q, X)$, каждая вершина q_j которого определяет некоторую операцию O_j , принадлежащую некоторому множеству допустимых операций $\mathbf{O}$ (рис.1.19).

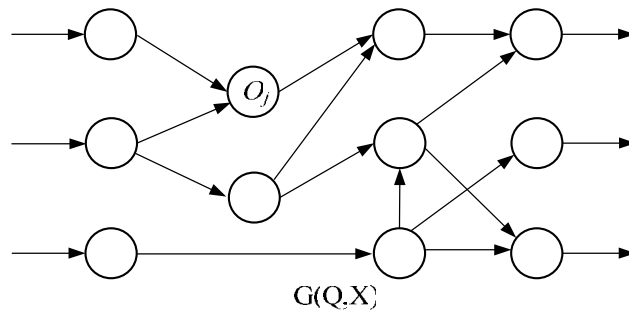


Рис. 1.19. Исходный граф алгоритма $G(Q, X)$

Допустим, что операция $O_j \in \mathbf{O}$ может быть разбита на ряд более простых операций. В общем случае операция O_j может быть представлена в виде некоторого графа алгоритма $G_j^{-1}(Q_j^{-1}, X_j^{-1})$, вершинами которого являются более простые операции, чем операция O_j (рис.1.20). Аналогичным образом могут быть детализированы вершины графа G_j^{-1} , каждая из которых может быть представлена в виде графа алгоритма $G_l^{-2}(Q_l^{-2}, X_l^{-2})$ более низкого уровня.

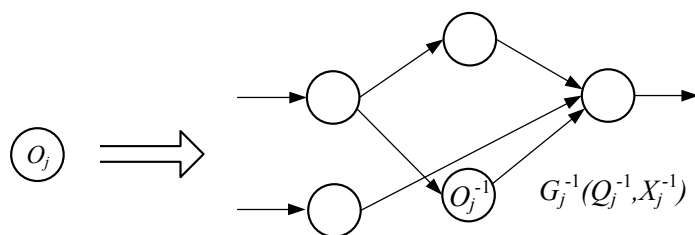


Рис. 1.20. Представление операции O_j в виде графа алгоритма $G_j^{-1}(Q_j^{-1}, X_j^{-1})$

Процесс разбиения (детализации) можно продолжать и далее вплоть до достижения некоторого уровня неделимых элементарных операций, которые в дальнейшем будем называть *микрооперациями*. В результате такого разбиения исходный граф $G(Q, X)$ может быть заменен графом микроопераций $G^{-n}(Q^{-n}, X^{-n})$ (где n – число уровней разбиения), который будет значительно более сложным, чем исходный граф $G(Q, X)$.

Полученный таким образом граф микроопераций может быть реализован с помощью мультиконвейерного вычислителя, состоящего из процессоров, реализующих микрооперации, приписанные вершинам графа $G^{-n}(Q^{-n}, X^{-n})$. Очевидно, что для реализации таких микроопераций не имеет смысла использовать сложные процессоры, а достаточно использовать простейшие однокбитные процессорные элементы, которые в дальнейшем будем называть *ячейками однородной вычислительной среды*. Мультиконвейер, состоящий из вычислительных ячеек, реализующих некоторый набор простейших микроопераций, будем в дальнейшем называть *мультимикроконвейерной вычислительной структурой*.

Преимущество мультимикроконвейерной структуры заключается в максимально возможной конвейеризации при обработке потока данных и, как следствие, обеспечении максимально возможного темпа работы конвейера и минимально возможного времени обработки большого массива векторов данных конвейера. Действительно, как показано выше, темп работы мультиконвейера обратно пропорционален времени ΔT выполнения самой длительной операции на его ступени, находящейся на критическом пути. Поэтому понятно, что чем проще будет эта операция, то тем меньше будет

время ΔT . Общее время обработки массива из N векторов входных данных при этом будет равно

$$T_{\text{реи}} = (N + H_{\text{max}} - 1) \cdot \Delta T, \quad (1.12)$$

где H_{max} – число ступеней мультимиконвейера, расположенных на критическом пути.

Естественно, чем проще будут микрооперации, приписанные вершинам графа $G^n(Q^n, X^n)$, тем сложнее будет сам граф и, как следствие, тем больше будет величина H_{max} . Однако при большом значении N величиной H_{max} можно пренебречь. В результате получаем

$$T_{\text{реи}} \approx N \cdot \Delta T,$$

то есть чем меньше будет величина ΔT , тем меньше будет и общее время обработки массива векторов данных.

Однако практическая реализация мультимикроконвейерного способа организации вычислений наталкивается на целый ряд технических сложностей. Во-первых, сама по себе микродетализация алгоритма – очень сложный, трудоемкий процесс, приводящий к графу очень большого размера. В свою очередь, реализация такого графа микроопераций в мультимикроконвейере потребует очень большого числа вычислительных ячеек. И хотя сама по себе каждая ячейка среды может быть достаточно проста, при их очень большом количестве суммарные аппаратные затраты на мультимикроконвейер могут выходить за допустимые пределы.

Проблема резкого роста аппаратных затрат на реализацию мультимикроконвейера еще более усугубляется в случае, если необходимо создать универсальный мультимикроконвейер, способный решать различные потоковые задачи, отличающиеся графами алгоритмов обработки информации. В этом случае, как было показано в параграфе 1.3, в состав мультимиконвейера должен входить коммутатор, обеспечивающий возможности полнодоступной коммутации между всеми процессорами мультимиконвейера. При этом, поскольку число вычислительных ячеек в мультимикроконвейере должно быть

достаточно велико, то аппаратные затраты на полнодоступный коммутатор (которые пропорциональны квадрату числа процессоров) будут опять-таки недопустимо велики. Поэтому универсальный мультимикроконвейер целесообразно строить только с использованием описанного в параграфе 1.3 принципа коммутации на основе придания процессорам дополнительных функций транзита данных. При этом универсальный мультимикроконвейер может быть представлен в виде некоторой матрицы элементарных процессоров (рис. 1.21), соединенных друг с другом по принципу близкодействия. Каждая из вычислительных ячеек такого мультимикроконвейерного вычислителя кроме возможностей по реализации микроопераций должна также обладать возможностями транзита данных, поступающих на ее входы, на любые свои выходы (и далее на входы соседних ячеек), в том числе с ветвлением передаваемых транзитом данных одновременно на несколько выходов.

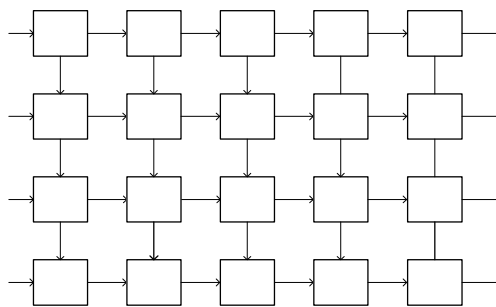


Рис. 1.21. Представление универсального мультимикроконвейера в виде матрицы вычислительных ячеек

Отображение алгоритма микроопераций в такую матричную структуру является нетривиальной задачей, но ее решение проще, чем обеспечение полнодоступной связи между вычислительными ячейками мультимикроконвейера на аппаратном уровне. Более подробно принципы организации, функционирования и программирования мультимикроконвейерных вычислительных структур будут рассмотрены во второй главе настоящей монографии.

Выше был рассмотрен вариант дробления исходного графа алгоритма до уровня микроопераций. Альтернативой этому подходу является укрупнение вершин графа алгоритма. Пусть у нас имеется граф алгоритма $G(Q, X)$

некоторой потоковой задачи. Разобьем этот граф на ряд подграфов $G_j(Q, X_j)$ ($j = 1, 2, \dots, M$) и каждому такому подграфу поставим в соответствие вершину q_j^1 графа $G^1(Q^1, X^1)$, дуги которого соединяют вершины q_j^1 и q_l^1 , соответствующие инцидентным подграфам $G_j(Q, X_j)$ и $G_l(Q, X_l)$ графа $G(Q, X)$ (рис. 1.22).

Каждой вершине q_j^1 графа $G^1(Q^1, X^1)$ припишем некоторую «крупную» операцию O_j^1 , алгоритм реализации которой описывается подграфом $G_j(Q, X_j)$. В результате получим «укрупненный» граф алгоритма решения задачи $G^1(Q^1, X^1)$.

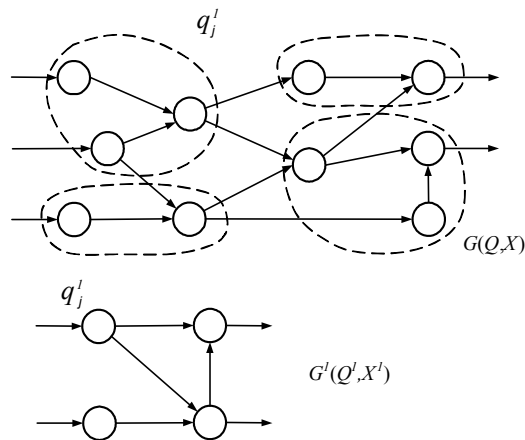


Рис. 1.22. Разбиение графа алгоритма $G(Q, X)$ на подграфы

Процесс укрупнения графа алгоритма можно продолжать и далее, разбивая граф $G^1(Q^1, X^1)$ на подграфы и формируя в результате еще более укрупненный граф алгоритма $G^2(Q^2, X^2)$ и т.д., вплоть до того момента, когда граф алгоритма «сожмется» в граф $G^n(Q^n, X^n)$, содержащий одну единственную вершину q^n .

Граф $G^i(Q^i, X^i)$ ($i = 1, 2, \dots, n$), полученный в результате укрупнения исходного графа алгоритма $G(Q, X)$, будем в дальнейшем называть *макрографом алгоритма*, а операции, приписанные вершинам этого графа, — *макрооперациями*.

Для того чтобы создать мультиконвейерную структуру, реализующую макрограф алгоритма $G^i(Q^i, X^i)$, необходимо каждой его вершине поставить в соответствие процессор и соединить эти процессоры между собой согласно

топологии макрографа. При этом, поскольку каждой вершине макрографа приписана «крупная» макрооперация, то процессор мультиконвейера должен быть способен ее реализовать, т.е. процессор должен обладать достаточными вычислительными возможностями. В дальнейшем мультиконвейер, состоящий из процессоров, способных выполнять крупные макрооперации, будем называть *мультимакроконвейерной вычислительной структурой*.

Основное преимущество мультимакроконвейерной архитектуры по сравнению с мультимикроконвейерной заключается в резком уменьшении числа процессоров за счет их «укрупнения». Данное обстоятельство открывает возможности организации мультимакроконвейерного вычислителя с использованием полнодоступного коммутатора. Это, в свою очередь, обеспечивает большие возможности структуры с точки зрения адаптации под топологию различных графов алгоритмов. Однако это преимущество достигается за счет значительного усложнения отдельных процессоров, а, главное, за счет снижения темпа работы мультиконвейера и, как следствие, увеличения общего времени обработки всего массива входных данных. Действительно, чем сложнее будут макрооперации, тем больше времени потребуется для их реализации на ступенях мультимикроконвейера и, соответственно, тем больше будет величина ΔT , определяющая темп выдачи данных на выходах мультиконвейерной структуры. Например, если ступени мультиконвейера реализовать с помощью обычного последовательного процессора с фон-неймановской архитектурой, то значение времени ΔT в данном случае будет приблизительно в K раз больше, чем в случае мультиконвейерной организации, где K – число вершин микрографа алгоритма, соответствующих одной вершине макрографа алгоритма.

Для устранения этого недостатка был предложен иной подход к организации мультимакроконвейера [46], суть которого заключается в следующем. Поставим в соответствие каждой вершине q_j^i макрографа $G^i(Q^i, X^i)$ специальный процессор, представляющий собой набор элементарных процессоров (ЭП), и коммутатор, обеспечивающий возможности их

полнодоступных связей (рис. 1.23). Каждый ЭП должен обеспечивать возможности реализации набора операций, приписанных вершинам исходного графа алгоритма $G(Q, X)$, а их число должно быть достаточно для отображения всех вершин подграфа $G_j(Q_j, X_j) \in G(Q, X)$, соответствующего вершине q_j^i макрографа $G^i(Q^i, X^i)$.

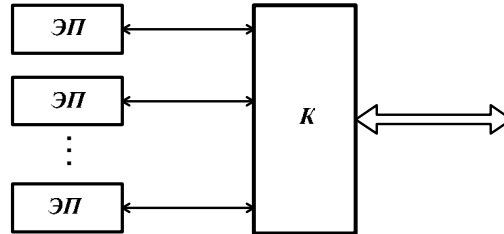


Рис. 1.23. Организация макропроцессора

Процессор такого типа в дальнейшем будем называть *макропроцессором*. С помощью такого макропроцессора можно организовать локальный мультиконвейер, реализующий подграф $G_j(Q_j, X_j)$, соответствующий вершине q_j^i макрографа $G^i(Q^i, X^i)$. Тогда организацию мультимакроконвейера можно представить в виде, как показано на рис. 1.24.

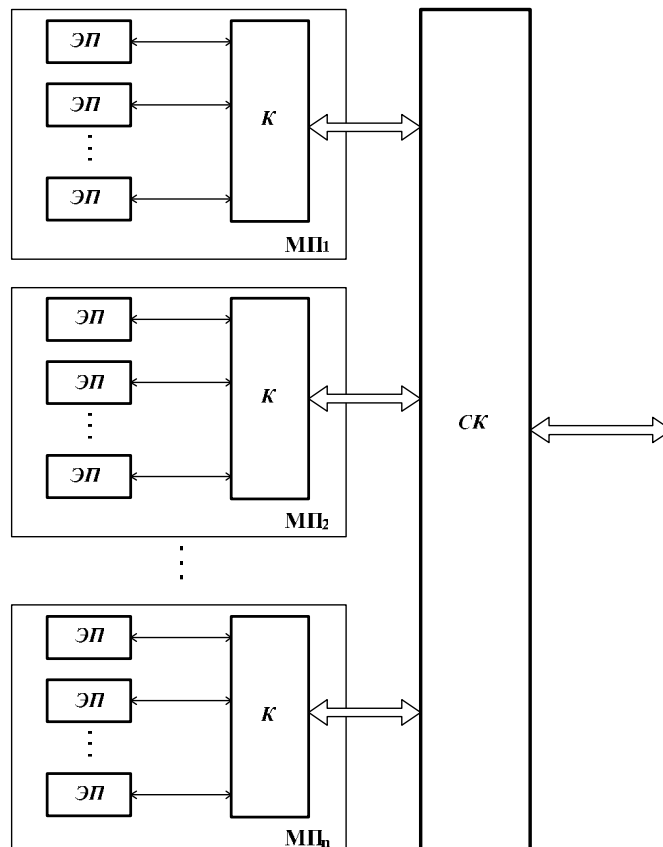


Рис. 1.24. Организация мультимакроконвейера

С помощью каждого макропроцессора, входящего в состав такой структуры, организуется локальный мультиконвейер, отображающий соответствующий подграф $G_j(Q_j, X_j)$ графа алгоритма $G(Q, X)$, а с помощью системного коммутатора верхнего уровня (СК) организуются информационные связи между макропроцессорами согласно топологии макрографа $G^i(Q^i, X^i)$. При этом темп обработки данных в такой мультиконвейерной вычислительной структуре будет «привязан» к времени выполнения самой длительной операции из множества операций O_j , реализуемых в элементарном процессоре.

По сравнению с мультимикроконвейерным способом организации вычислительной структуры последний подход обладает рядом преимуществ. Во-первых, при этом обеспечиваются возможности более гибкой коммутации между процессорными элементами, причем поскольку коммутатор разбивается на два уровня – локальный коммутатор макропроцессора и системный коммутатор верхнего уровня, то аппаратные затраты на их реализацию будут не столь велики. Кроме того, значительно упрощается процесс программирования таких структур, так как программирование происходит не в терминах простейших микроопераций, а в терминах крупных макроопераций. Однако, с другой стороны, темп обработки данных в мультимакроконвейере будет ниже, чем в мультимикроконвейере.

Более подробно принципы организации, функционирования и программирования мультимакроконвейерных структур будут рассмотрены далее.

1.5. Структурно-процедурный способ организации мультиконвейерных вычислений

Выше были рассмотрены способы организации мультимикро- и мультимакроконвейерных вычислений. Результатом использования обоих способов является создание некоторой мультиконвейерной структуры с той или иной степенью детализации отображаемого в ней графа алгоритма решения

поточковой задачи. При этом предполагалось, что аппаратных средств мультиконвейерной структуры (процессоров и коммутационных возможностей) должно хватить для отображения всего графа алгоритма. Однако в реальности это условие может и не выполняться. Как правило, число операционных вершин графа алгоритма решаемой задачи оказывается значительно больше, чем число процессорных элементов в мультиконвейере. Поэтому возникает проблема отображения графа алгоритма большого размера на ограниченную мультиконвейерную структуру [50,80]. Очевидный способ решения этой проблемы предложен в работе [50]. Суть этого способа заключается в том, чтобы предварительно разрезать исходный граф алгоритма $G(Q, X)$ на подграфы $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) таким образом, чтобы каждый подграф $G_i(Q_i, X_i)$ мог быть реализован с помощью имеющейся в наличии мультиконвейерной структуры. При этом разбиение исходного графа алгоритма на подграфы должно осуществляться, исходя из следующих соображений.

1. Число вершин в подграфе $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) должно быть по возможности равным (но не большим) числу процессоров в мультиконвейере.
2. Число внутренних дуг подграфа $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) должно быть не больше, чем число связей, обеспечиваемых коммутатором мультиконвейерной структуры.
3. Число входных и выходных дуг подграфа должно быть не больше числа каналов связи мультиконвейера с внешней памятью.

На рис. 1.25 показан пример графа $G(Q, X)$ базовой операции ЦОС и его разрезания на последовательность более простых подграфов $G_i(Q_i, X_i)$ ($i = 1, 2, 3, 4$).

Такое многокритериальное разрезание графа $G(Q, X)$ на подграфы - достаточно непростой процесс, который более подробно рассмотрен в [50]. А здесь, чтобы не затруднять общее понимание идеи, будем считать, что граф алгоритма $G(Q, X)$ был каким-то образом разрезан на подграфы $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$), отвечающие сформированным выше требованиям. Далее, для того

чтобы осуществить процесс решения потоковой задачи, описываемый графом алгоритма $G(Q, X)$, необходимо осуществить процедуру последовательного отображения подграфов $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) в мультиконвейерную структуру. При этом необходимо учесть следующие соображения.

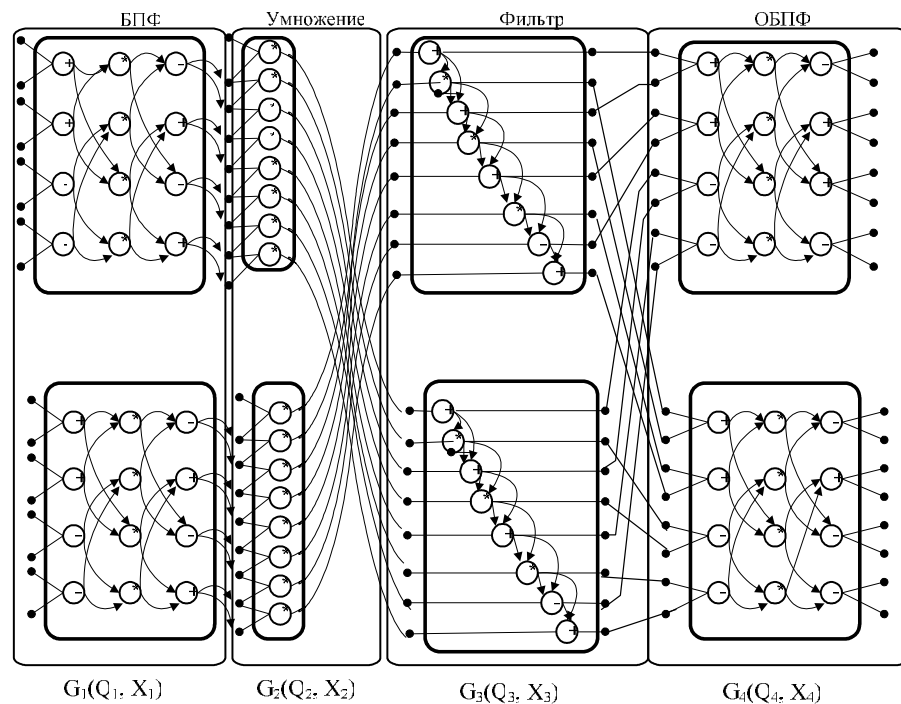


Рис. 1.25. Информационный граф базовой операции ЦОС

Поскольку процесс отображения подграфа $G_i(Q_i, X_i)$ в мультиконвейер достаточно сложный, то для сокращения времени решения потоковой задачи необходимо, чтобы каждый подграф $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) отображался в мультиконвейер один раз. При этом через мультиконвейер, отображающий подграф $G_i(Q_i, X_i)$, должен быть пропущен весь поток векторов входных данных для данного подграфа.

Для этого необходимо, чтобы к моменту отображения подграфа $G_i(Q_i, X_i)$ в мультиконвейер все вектора входных данных, пропускаемых через этот подграф, были готовы. Иными словами, прежде чем приступить к реализации подграфа $G_i(Q_i, X_i)$ в мультиконвейере, необходимо реализовать все подграфы, соединенные с подграфом $G_i(Q_i, X_i)$ исходящими дугами. Результаты обработки потока данных в мультиконвейере, реализующем подграф $G_i(Q_i, X_i)$, должны сохраняться в некоторой промежуточной памяти, причем эти вектора данных

должны быть доступны для последующей обработки при отображении в мультиконвейере подграфов $G_{i+1}(Q_{i+1}, X_{i+1}), \dots, G_F(Q_F, X_F)$, инцидентных подграфу $G_i(Q_i, X_i)$. При этом желательно, чтобы частота записи/считывания векторов данных в память и из памяти была не меньше темпа обработки данных на ступенях мультиконвейера. В противном случае темп обработки информации в мультиконвейере должен будет привязан к частоте записи/считывания данных из памяти.

Учитывая приведенные выше соображения, укрупненную схему организации мультиконвейерного вычислителя, реализующего графы-алгоритмы большой размерности, можно представить в следующем виде (рис. 1.26).

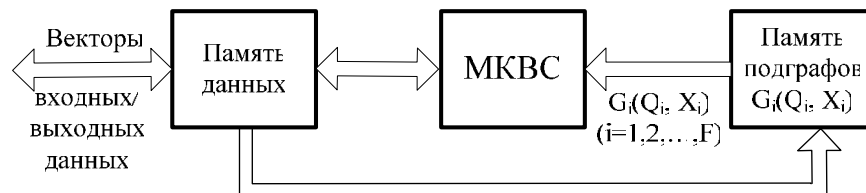


Рис. 1.26. Структура мультиконвейерного вычислителя (МКВС), реализующего информационные графы большой размерности

В целом, такой вычислитель должен работать следующим образом (рис. 1.27).

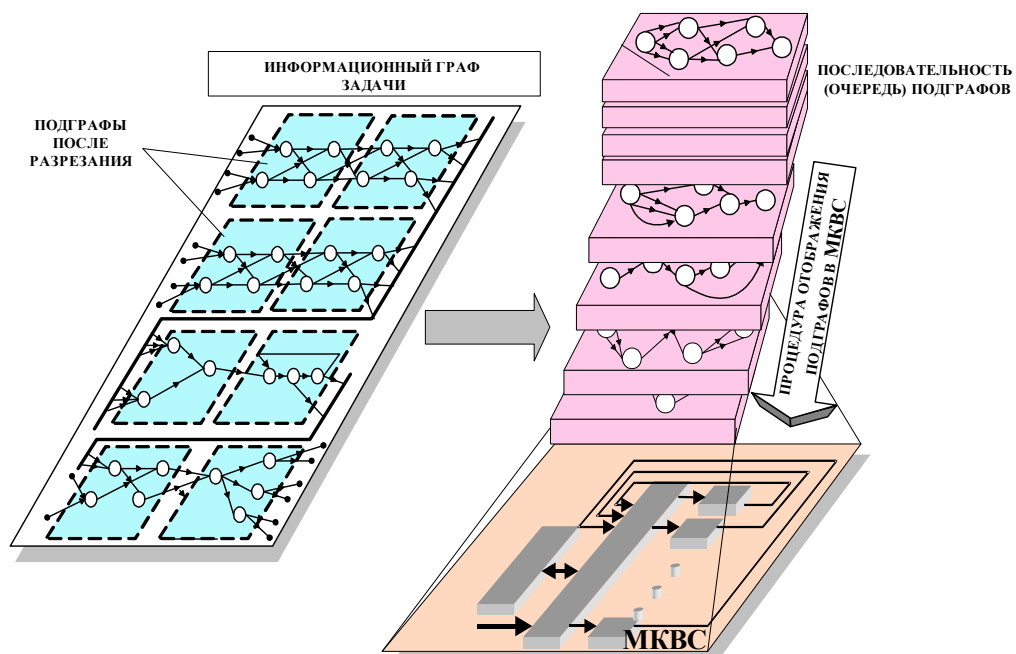


Рис. 1.27. Структурно-процедурная организация вычислений

Исходный граф алгоритма $G(Q, X)$ решения потоковой задачи разрезается на подграфы $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$), согласно описанным выше требованиям. Затем формируется очередь подграфов, причем номер i подграфа $G_i(Q_i, X_i)$ в очереди должен быть выбран таким образом, чтобы подграфы, являющиеся предшественниками для подграфа $G_i(Q_i, X_i)$, имели меньший номер.

Подграфы $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) последовательно, согласно установленной очереди, отображаются в мультиконвейерную структуру. При этом на входы мультиконвейера из памяти выдается поток векторов данных, входных для подграфа $G_i(Q_i, X_i)$, а вектора выходных данных записываются обратно в память. Процесс продолжается до тех пор, пока в мультиконвейере не будут реализованы все подграфы $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$), и через каждый из них не будет пропущен соответствующий поток векторов входных данных. После завершения процедуры прогона потока данных через последний подграф $G_F(Q_F, X_F)$ в памяти будет накоплено множество векторов результирующих входных данных, которые далее выдаются на выход системы.

Описанную выше процедуру в дальнейшем будем называть *структурно-процедурным способом организации мультиконвейерных вычислений*, потому что каждый из подграфов очереди реализуется в мультиконвейере структурно, в то время как отображение подграфов в мультиконвейер осуществляется путем организации последовательной процедуры [143].

Время обработки массива из N данных на мультиконвейерной структуре, реализующей структурно-процедурный способ организации вычислений, будет определяться выражением

$$T_{\text{рек}} = \sum_{i=1}^F \left((N + H_{\text{max}}^i - 1) \cdot \Delta T_i + T_{\text{рек}}^i \right),$$

где H_{max}^i — длина критического пути в подграфе $G_i(Q_i, X_i)$;

ΔT_i — время выполнения наиболее длительной операции, приписанной вершинам подграфа $G_i(Q_i, X_i)$;

$T_{\text{рек}}^i$ — время настройки (реконфигурации) мультиконвейера для

отображения подграфа $G_i(Q_i, X_i)$;

F – число подграфов, на которые разрезан исходных граф алгоритма.

Если принять, что $\Delta T_i = \Delta T (i = 1, 2, \dots, F)$, $T_{рек}^i = T_{рек} (i = 1, 2, \dots, F)$ и $H_{\max}^i = H (i = 1, 2, \dots, F)$, то последнее выражение можно переписать в виде

$$T_{реш} = F((N + H_{\max} - 1)\Delta T + T_{рек}), \quad (1.13)$$

или при большом значении N

$$T_{реш} = F(N\Delta T + T_{рек}). \quad (1.14)$$

Как видно из последнего выражения, существенную часть времени обработки массива N данных в мультиконвейерной вычислительной системе, реализующей структурно-процедурный способ организации вычислений, занимает время реконфигурации, когда полезные вычисления не выполняются. Для того чтобы сократить общее время решения $T_{реш}$, необходимо каким-то образом уменьшить временные затраты на реконфигурацию системы. Простейший путь решения этой проблемы состоит в использовании мультиконвейерной вычислительной структуры, содержащей несколько реконфигурируемых подсистем. Рассмотрим этот подход более подробно.

Предположим, что у нас имеется вычислительная система, состоящая из двух МКВС, каждая из которых может структурно реализовывать свой подграф $G_i(Q_i, X_i)$ графа алгоритма $G(Q, X)$ решаемой задачи (рис. 1.28).

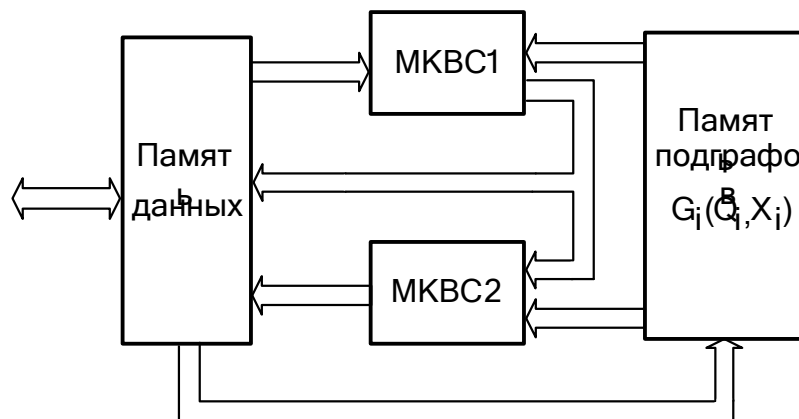


Рис. 1.28. Двухкаскадная МКВС с параллельной реконфигурацией

Тогда, если в эти МКВС параллельно загрузить два последовательных подграфа $G_i(Q_i, X_i)$ и $G_{i+1}(Q_{i+1}, X_{i+1})$, ($i = 1, 2, \dots, F/2$) (рис. 1.29) графа $G(Q, X)$, и на

вход МКВС1 подать массив векторов входных данных подграфа $G_i(Q_i, X_i)$, то через время $T_{\epsilon}^{i,1} = H_{\max}^i \cdot \Delta T_i$ первый из обработанных согласно подграфу алгоритма $G_i(Q_i, X_i)$ векторов данных появится на выходе МКВС1.

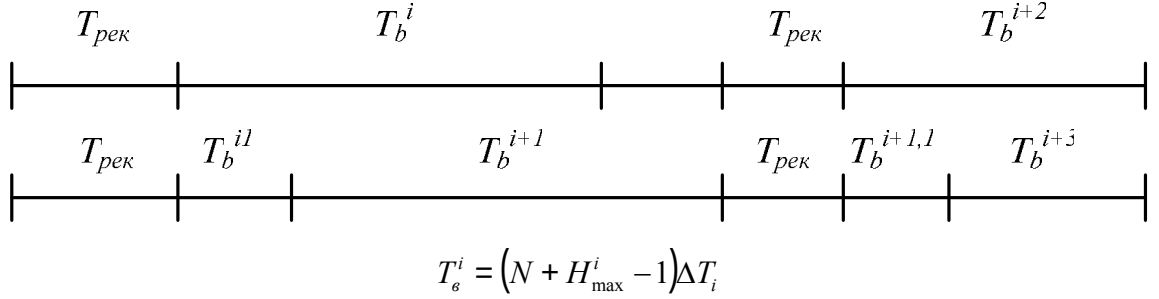


Рис. 1.29. Временная диаграмма вычислительного процесса в двухкаскадной МКВС с параллельной реконфигурацией

Этот вектор данных сразу же поступает на входы МКВС2, где начинается его обработка согласно подграфу алгоритма $G_{i+1}(Q_{i+1}, X_{i+1})$. В результате через время $(T_{\epsilon}^{i,1} + T_{\epsilon}^{i+1})$ весь массив данных будет «пропущен» через подграфы $G_i(Q_i, X_i)$ и $G_{i+1}(Q_{i+1}, X_{i+1})$, где T_{ϵ}^i – время обработки всего массива векторов данных в МКВС согласно подграфу алгоритма $G_i(Q_i, X_i)$ (см. рис. 1.29). После этого в МКВС1 и МКВС2 загружаются следующие два подграфа $G_{i+3}(Q_{i+3}, X_{i+3})$ и $G_{i+4}(Q_{i+4}, X_{i+4})$ графа алгоритма $G(Q, X)$, и процесс конвейерной прогонки массива данных через них повторяется заново. В результате общее время решения при такой организации вычислительного процесса будет составлять

$$T_{\text{реш}} = \sum_{i=1}^{F/2} (T_{\epsilon}^{i,1} + T_{\epsilon}^{i+1} + T_{\text{рек}}) = \sum_{i=1}^{F/2} ((H_{\max}^i + H_{\max}^{i+1} + N - 1) \Delta T + T_{\text{рек}}),$$

где $\Delta T = \max(\Delta T_i, \Delta T_{i+1})$;

$$T_{\text{рек}} = \max(\Delta T_{\text{рек}}^i, \Delta T_{\text{рек}}^{i+1});$$

F – число подграфов, на которые разрезан исходный подграф $G(Q, X)$.

Или считая, что $H_{\max}^i \approx H_{\max}^{i+1}$, получаем

$$T_{\text{реш}} \approx F/2 \left((2H_{\max} + N - 1) \cdot \Delta T + T_{\text{рек}} \right), \quad (1.15)$$

то есть время решения сокращается почти в два раза по сравнению с вариантом использования одной МКВС (см. 1.13).

Увеличивая число каскадов в МКВС, общее время структурно-процедурного решения задачи можно еще более сократить. Однако понятно, что этот выигрыш во времени решения будет достигаться за счет пропорционального увеличения аппаратных затрат.

При организации структурно-процедурных вычислений необходимо обратить внимание на еще одну потенциальную проблему. Допустим, что на мультиконвейерном вычислителе необходимо обрабатывать бесконечный (или очень большой) поток векторов входных данных, причем данные потоки поступают с некоторого источника информации через постоянные, равные промежутки времени $\Delta T_{\text{вх}}$. Если такой поток обрабатывать на обычном мультиконвейерном вычислителе, то темп обработки информации в нем необходимо просто привязать к величине $\Delta T_{\text{вх}}$ и через тот же самый промежуток $\Delta T_{\text{вх}}$ получать на выходе результирующие выходные вектора данных. При использовании же структурно-процедурного способа организации мультиконвейерных вычислений такой подход неприемлем. Действительно, поскольку, как правило, время реконфигурации мультиконвейерного вычислителя достаточно велико, то может оказаться (см. (1.13)), что

$$\Delta T_{\text{вх}} < T_{\text{реш}} = F((1 + H - 1) \cdot \Delta T + T_{\text{рек}}) = F(H \cdot \Delta T + T_{\text{рек}}),$$

т.е. к моменту поступления очередного вектора входных данных мультиконвейер не будет успевать обрабатывать предыдущий вектор данных. Очевидный выход из данной ситуации заключается в накоплении некоторого пакета (подмножества) векторов входных данных, который затем уже поступает на вход мультиконвейера со структурно-процедурной организацией вычислений. Если число векторов данных в пакете будет большим, то мультиконвейер может не успеть их обработать до момента накопления очередного пакета. Если же, наоборот, число данных в пакете будет малым, то мультиконвейер обработает их раньше, чем будет накоплен очередной пакет данных, и будет бесполезно простаивать. Поэтому оптимальный размер пакета векторов данных должен удовлетворять условию: время обработки пакета на мультиконвейере должно равняться времени накопления пакета, т.е.

$$N \cdot \Delta T_{\text{вх}} = T_{\text{реш}},$$

или, учитывая (1.13),

$$N \cdot \Delta T_{\text{вх}} = F((N + H - 1)\Delta T + T_{\text{рек}}), \quad (1.16)$$

где N – число векторов входных данных в пакете;

$\Delta T_{\text{вх}}$ – промежуток времени (темп) между поступлением входных данных;

$T_{\text{реш}}$ – время обработки пакета в мультиконвейере со структурно-процедурной организацией вычислений.

Из выражения (1.16) можно определить оптимальное число N векторов данных в пакете

$$N = \frac{F((H - 1)\Delta T + T_{\text{рек}})}{T_{\text{вх}} - F\Delta T}. \quad (1.17)$$

Таким образом, для того чтобы организовать обработку большого множества (потока) входных данных, поступающих с темпом $1/\Delta T_{\text{вх}}$, согласно сложному графу алгоритму $G(Q, X)$, необходимо накапливать пакеты векторов данных размером (1.17) и затем каждый пакет обрабатывать на мультиконвейере со структурно-процедурной организацией вычислений.

Как показано выше, одним из основополагающих принципов организации структурно-процедурных вычислений является установление очереди подграфов $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) исходного графа алгоритма $G(Q, X)$ таким образом, чтобы к моменту отображения в мультиконвейер подграфа $G_i(Q_i, X_i)$ был полностью готов пакет векторов входных данных, необходимых для его реализации. Это требование достаточно просто можно удовлетворить в случае, если данные d_l^i ($l = 1, 2, \dots, k$), входящие в вектор входных данных $D_i = \langle d_1^i, d_2^i, \dots, d_k^i \rangle$, поступают одновременно или в заранее определенной последовательности. Однако если данные d_l^i ($l = 1, 2, \dots, k$) поступают от различных несинхронизированных источников информации, то эти данные могут поступать в случайные, заранее неопределенные моменты времени. В этом случае установить четкую последовательность реализации подграфов $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) графа алгоритма $G(Q, X)$ на мультиконвейере не представляется

возможным. В данных условиях имеет смысл отображать в мультиконвейер тот подграф $G_i(Q_i, X_i)$, для которого в текущий момент времени готов пакет векторов входных данных.

Структуру вычислительной системы, реализующей такой подход, можно представить в следующем виде (рис. 1.30).

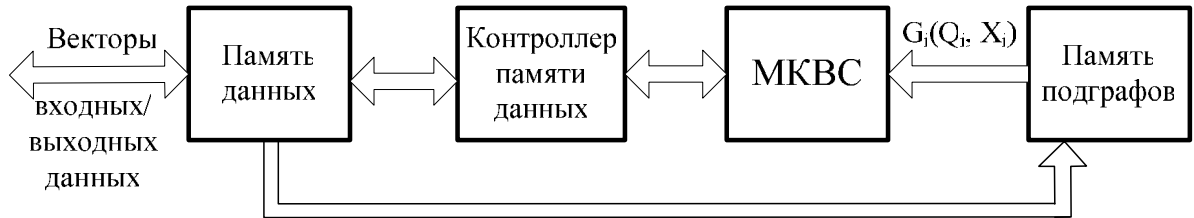


Рис. 1.30. Структура мультиконвейерного вычислителя с заранее неизвестной последовательностью реализуемых подграфов $G_i(Q_i, X_i)$

В целом система должна работать следующим образом. В памяти данных для каждого подграфа $G_i(Q_i, X_i)$ должен быть отведен свой банк, размер которого достаточен для размещения пакета из N векторов данных. Входные данные, поступающие с источников информации, записываются в соответствующие ячейки памяти данных. Контроллер памяти данных (КПД) постоянно сканирует банки памяти данных, относящихся к подграфам $G_j(Q_j, X_j)$ ($j = 1, 2, \dots, F$). Как только банк подграфа $G_i(Q_i, X_i)$ заполняется пакетом векторов входных данных, то КПД выдает соответствующую команду в память подграфов, по которой подграф $G_i(Q_i, X_i)$ отображается в мультиконвейерную вычислительную структуру. Если сразу несколько подграфов готовы к исполнению, то КПД устанавливает их очередь. После того как МКВС запрограммирована на реализацию подграфа $G_i(Q_i, X_i)$, КПД выделяет в памяти данных банк с пакетом входных данных для данного подграфа и с темпом $1/\Delta T$ выдает их на входы МКВС. Выходные данные, полученные в результате обработки на МКВС, передаются в КПД, который записывает их в соответствующие банки памяти данных.

После этого КПД определяет следующий готовый к исполнению подграф $G_i(Q_i, X_i)$ (т.е. подграф, для которого сформирован пакет входных данных) и

выдает команду в память подграфов для его отображения в МКВС. Процесс продолжается до момента обработки на МКВС всех подграфов $G_i(Q_i, X_i)$ ($i = 1, 2, \dots, F$) графа алгоритма $G(Q, X)$. Полученные в результате вектора выходных данных выдаются на выходы системы.

Более подробно принципы организации структурно-процедурных вычислений в мультиконвейерных структурах будут рассмотрены в третьей главе монографии.

Глава 2. МУЛЬТИМИКРОКОНВЕЙЕРНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СТРУКТУРЫ НА ОДНОРОДНЫХ СРЕДАХ

2.1. Архитектура однородной вычислительной среды

Как показано в предыдущей главе, одним из вариантов построения мультимикронвейерных вычислительных структур являются структуры, которые строятся на основе процессорных элементов, реализующих некоторый набор простейших однобитных операций и связанных друг с другом по принципу близкодействия. В литературе такие мультимикронвейерные структуры принято называть однородными вычислительными средами (ОВС), а их процессорные элементы – ячейками ОВС [12]. Как показано выше, преимущества организации мультимикронвейерной вычислительной структуры на ОВС заключаются в простоте ее технической реализации и обеспечении максимального темпа конвейерной обработки. К недостаткам подхода следует отнести сложности, связанные с микродетализацией графа алгоритма и его эффективным отображением в ячейки среды.

Впервые термин «однородная вычислительная среда» был предложен Э.В.Евреиновым в 60-х годах прошлого века в работе [34]. Теория однородных вычислительных сред и особенности их применения были изложены в работах [35-40, 95, 96]. Появление этих работ, а также проведение конференций по однородным вычислительным системам, структурам и средам вызвали известный интерес к этому направлению среди специалистов как в нашей стране, так и за рубежом. В частности, близкими по существу к однородным вычислительным структурам оказались транспьютеры, а к вычислительным средам - систолические массивы [69, 119], описанные в известной статье Х. Кунга и С. Лейзерсона «Systolic Arrays» [145], опубликованной в 1978 году.

Как определено в [40], однородная вычислительная среда представляет собой n -мерную решетчатую структуру. Каждая ячейка соединена с соседними ячейками каналами передачи рабочей и настроечной информации. Ячейка является автоматом с программируемой структурой.

Имеется возможность путем настройки элементов реализовать в вычислительной среде любую вычислительную машину при наличии достаточного числа ячеек.

Организация мультимикроконвейерной вычислительной структуры на однородной среде показана на рис. 2.1. Структура состоит из двух основных частей: матрицы вычислительных ячеек однородной вычислительной среды (ОВС) и матрицы запоминающих ячеек однородной запоминающей среды (ОЗС), связанных между собой через свои периферийные элементы [12].

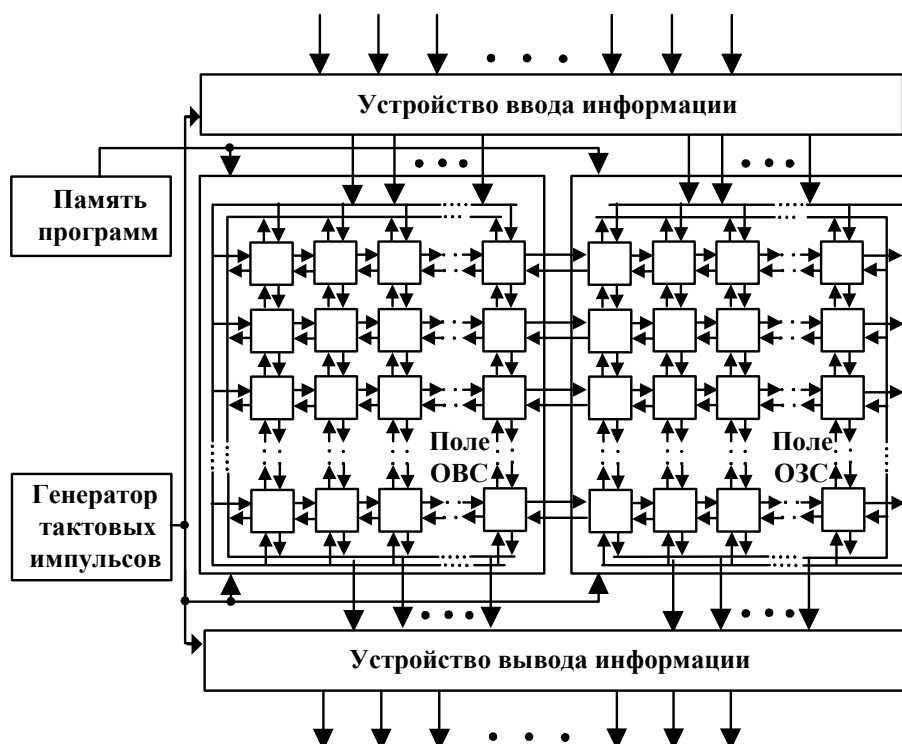


Рис. 2.1. Схема вычислительной структуры на ОВС

Входами и выходами вычислительной структуры служат входы и выходы вычислительных и запоминающих ячеек, расположенных по периметру матриц. Подключение источников и приёмников информации осуществляется через устройства сопряжения с использованием в необходимых случаях регистровых запоминающих устройств.

Процесс настройки ОВС и ее работа разделены во времени: вначале формируется поток команд для настройки, производится его загрузка в регистры команд каждой вычислительной и запоминающей ячейки, а затем

запускается поток данных. Память в виде однородной запоминающей среды реализуется таким образом, чтобы исключался процесс адресации. Загружаемая программа учитывает все варианты ветвления вычислений.

Хранение программы в процессе решения производится в регистрах команд ячеек среды. Работа элементов однородной вычислительной среды и однородной запоминающей среды синхронизируется от одного внешнего генератора тактовых импульсов. Обработывается информация конвейерным способом, при котором поток информации, подаваемый из устройства приема на входы матриц вычислительных и запоминающих ячеек, обрабатывается в соответствии с программой, продвигаясь синхронно с тактовыми импульсами [2]. Таким образом, в мультимикроконвейерной структуре проводится аппаратная реализация алгоритма.

Основу однородной вычислительной среды составляет вычислительная ячейка, реализующая некоторый набор простейших однобитных операций. Схема такой вычислительной ячейки показана на рис. 2.2.

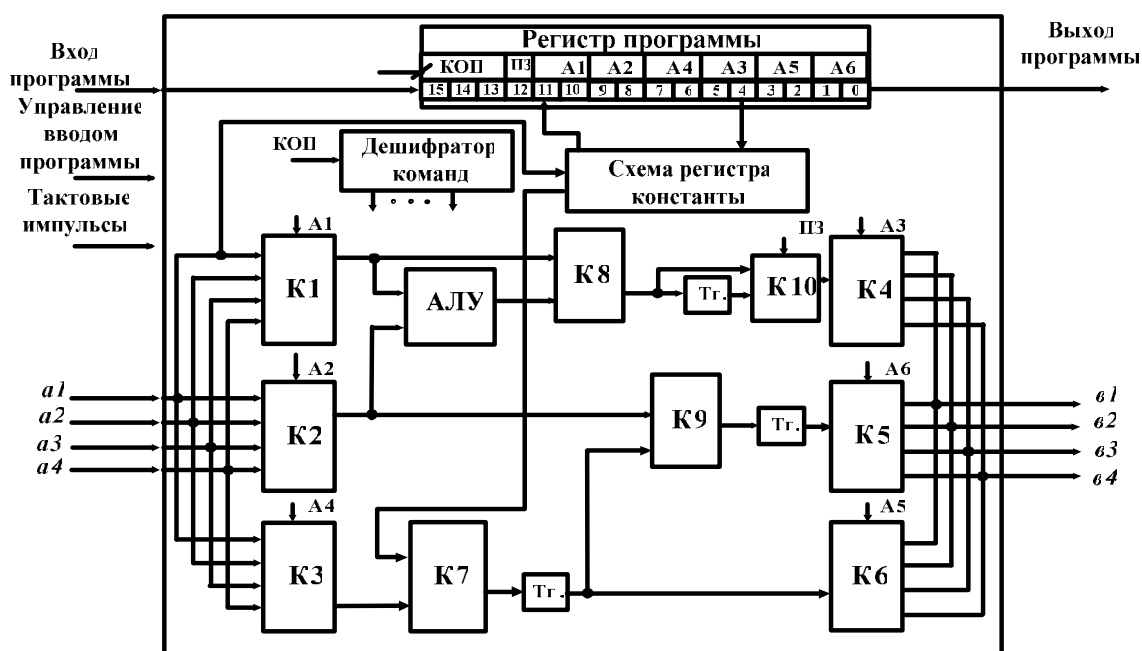


Рис.2.2. Структурная схема вычислительной ячейки среды

Ячейка представляет собой одноразрядный тактируемый модуль, содержащий собственный регистр программы. Вычислительная ячейка имеет четыре информационных входа ($a1, \dots, a4$) и выхода ($v1, \dots, v4$), вход управления

вводом программ, вход и выход канала программирования, входы тактовых импульсов и “Сброса”. Ячейка содержит в себе такие функциональные элементы как регистр программы, дешифратор команд, арифметико-логическое устройство (АЛУ), три входных и три выходных коммутатора (K1, K2, K3 и K4, K5, K6), коммутаторы канала АЛУ (K8 и K10), константы (K7), транзита (K9) и схему регистра константы.

Регистр программы имеет 16 разрядов, которые объединены в следующие поля:

- разряды 0,1 - адрес выхода второго канала транзита;
- разряды 2,3 - адрес выхода первого канала транзита;
- разряды 4,5 - адрес выхода канала АЛУ;
- разряды 6,7 - адрес входа первого канала транзита;
- разряды 8,9 - адрес входа второго операнда;
- разряды 10,11 - адрес входа первого операнда;
- разряд 12 - признак задержки;
- разряды 13,14,15 - код операции АЛУ.

Вычислительная ячейка может находиться в двух режимах: в режиме настройки и режиме работы. По сигналу управления вводом программы, равному логической “1”, вычислительная ячейка переходит в режим “настройка”, где с каждым тактовым импульсом осуществляется запись информации в регистр программ. При сигнале, равном логическому “0”, вычислительная ячейка переходит в режим “работа”, когда она по заданному коду регистра программы осуществляет в канале АЛУ арифметико-логическую операцию над одноразрядными операндами, поступающими с входных коммутаторов K1, K2, и передачу в следующем такте работы результата обработки через коммутаторы K8, K10 на выходной коммутатор K4. При этом ячейка параллельно может осуществлять коммутацию входных информационных потоков с её входов на выходы по двум каналам транзита. Причем каналы транзита через коммутаторы K2, K9, K5 и K3, K7, K6 передают на выход ячейки информацию с задержкой, соответственно, на один и два

такта. При значении ПЗ=1 ячейка осуществляет в канале АЛУ через коммутатор К10 дополнительную задержку информации на один такт.

В зависимости от кода операции (КОП) вычислительная ячейка выполняет следующие арифметико-логические операции:

- суммирование с запоминанием переноса (КОП=001, "++");
- логическое умножение с инверсией (КОП=010);
- суммирование "по модулю 2" (КОП=101, " $\oplus$ ");
- запоминание "1" (КОП=110, "ЗП1");
- генерирование константы (КОП=111, "ГК");
- расширенный транзит (КОП=011, "РТ");
- нет операции (КОП=000, "НОП").

Таким образом, система команд вычислительной ячейки включает арифметические и логические операции, а также операции транзита.

Операция "Запоминание 1" выполняется следующим образом:

- если второй операнд равен логическому "0", ячейка осуществляет передачу на выход значения первого операнда;
- если второй операнд равен логической "1", ячейка сохраняет на выходе своё предыдущее значение.

При выполнении этой операции результат на выходе ячейки появляется через два такта.

Выполняя операцию "Генерация константы", ячейка осуществляет:

- при ПЗ="0" - циклическую передачу на выход ячейки восьмиразрядной константы, предварительно записанной в разрядах 4, ..., 11 регистра программы;
- при ПЗ="1" - передачу информации, поступающей со входа "a1" ячейки через разряды 4, ..., 11 регистра программы, на её выход, производя тем самым задержку информации на 8 тактов.

При реализации операций "Расширенный транзит" и "Нет операции" ячейка производит коммутацию информационных потоков с входов на выходы ячейки по каналу АЛУ и двум каналам транзита.

На рис.2.3 показана структурная схема однородной запоминающей среды.

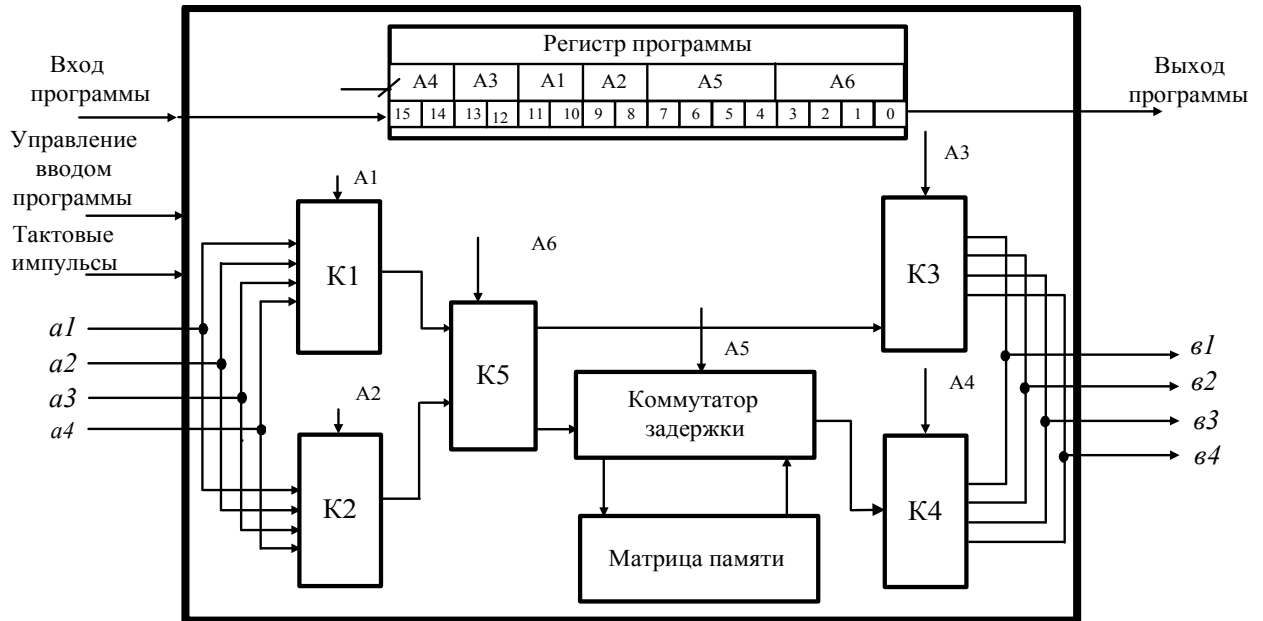


Рис.2.3. Структурная схема запоминающей ячейки среды

Запоминающая ячейка представляет собой устройство, выполняющее функции запоминания информации, выдачи её на выход ячейки с задержкой на число тактовых импульсов, указанное в программном регистре, а также циклического переключения информационных каналов. Ячейка имеет четыре информационных входа ($a_1, \dots, a_4$) и выходы ($v_1, \dots, v_4$), вход и выход программы, входы управления вводом программы и тактовых импульсов. Ячейка содержит шестнадцатитрибитовый регистр программы, два коммутатора входа (K1, K2) и выхода (K3, K4), коммутатор переключения каналов (K5), коммутатор матрицы памяти с дешифратором кода величины задержки и матрицу памяти.

Формат регистра программы следующий:

- разряды 0, ..., 3 (поле A6) - управление коммутатором каналов,
- разряды 4, ..., 7 (поле A5) - управление величиной задержки,
- разряды 8, 9 (поле A2) - управление вторым входным коммутатором,
- разряды 10, 11 (поле A1) - управление первым входным коммутатором,
- разряды 12, 13 (поле A3) - управление вторым выходным коммутатором,
- разряды 14, 15 (поле A4) - управление первым выходным коммутатором.

Для запоминающей ячейки также предполагаются режимы настройки и работы. По сигналу управления вводом программы, равному логической "1",

ячейка переходит в режим “настройка”. При сигнале, равном логическому “0”, ячейка переходит в режим ”работа”, при котором реализуется операция текущей микрокоманды. Запоминающая ячейка одновременно выполняет операции запоминания информации и выдачи ее на выход ячейки с установленной задержкой и, кроме того, реализует циклическое переключение каналов.

2.2. Организация вычислительного процесса в ОВС

Определим понятия микрооперации, микрокоманды и микропрограммного модуля для ОВС.

Микрооперация - элементарная операция, выполняемая в операционном или транзитном каналах ячейки, согласно коду, размещенному в поле микрокоманды.

Микрокоманда - это код операций, выполняемых одновременно в вычислительной ячейке за один такт.

Микрокоманда записывается в 16-разрядный регистр настройки ячейки. Микрокоманда вычислительной ячейки содержит 8 полей. Совокупность микрокоманд, то есть команд настройки некоторой, как правило, прямоугольной области однородной вычислительной среды, в результате которой в среде синтезируется спецпроцессор для выполнения определенной арифметической, логической или "крупной" операции, назовём *микропрограммным модулем* (МПМ).

На рис. 2.4 в качестве примера приведена структурная схема спецпроцессора, выполняющего операцию вычитания n -разрядных чисел в модифицированном дополнительном коде.

При графическом изображении спецпроцессоров будем придерживаться определенных правил: трассы операндов, поступающие в АЛУ ячейки, изображаются сплошными линиями, а транзитные трассы – пунктирными. Графически показаны операции, выполняемые в ячейке; обозначены также такты, на которых поступают операнды.

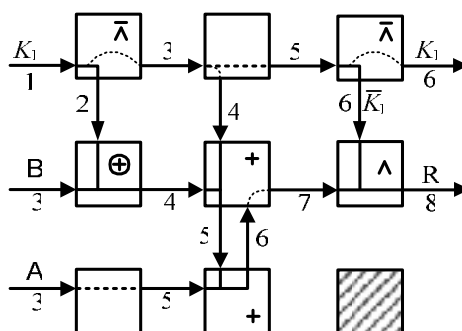


Рис.2.4. Структурная схема спецпроцессора вычитания

Схеме спецпроцессора на рис.2.4 соответствует микропрограммный модуль операции вычитания, представленный в виде табл. 2.1 и определяющий коды настройки каждой из ячеек, входящих в модуль вычитания.

Таблица 2.1

Номер ячейки	КОП	ПЗ	A1	A2	A4	A3	A6	A5
1.1	100	0	00	00	00	11	10	01
1.2	001	0	00	00	00	01	10	11
1.3	100	0	00	00	00	11	01	10
2.1	101	0	01	00	00	10	01	11
2.2	001	0	01	00	11	11	00	10
2.3	010	0	00	01	00	10	01	11
3.1	000	0	00	00	00	01	10	11
3.2	001	0	01	00	00	01	10	11
3.3	000	0	00	00	00	00	00	00

Таким образом, некоторая область среды, настроенная микрокомандами, составляющими микропрограммный модуль, представляет собой спецпроцессор, настроенный на выполнение определенной операции с операндами произвольной, но фиксированной разрядности.

В большинстве случаев операции, которые выполняются синтезированными в поле среды спецпроцессорами, - двухместные, а, следовательно, на входы непрерывно поступают два вектора A и B операндов.

С задержкой в S_k тактов, имеющей определенную величину, с выхода спецпроцессора непрерывно начинает поступать последовательным кодом младшими разрядами вперед результат, как правило, той же разрядности, что и входные данные. Таким образом, спецпроцессор по принципу поступления информации представляет собой устройство последовательной обработки

информации. Однако есть существенное отличие спецпроцессора на ОВС от арифметического устройства последовательного действия. В спецпроцессоре на ОВС любая операция выполняется не программно, как в обычных ЭВМ, а аппаратно, то есть весь алгоритм операции во времени выполняется параллельно, и если входные данные поступают непрерывно, то происходит непрерывная обработка различных временных срезов, алгоритм в любой момент времени реализуется в каждой своей фазе.

В каждый момент времени в ОВС работают N спецпроцессоров, в которых также параллельно, то есть одновременно, работают m вычислительных ячеек, аппаратно реализующих операции, а совокупность одновременно работающих спецпроцессоров аппаратно реализует алгоритм.

Для организации информационных обменов между спецпроцессорами (СП) в ОВС используется дискретный регистровый принцип коммутации. Регистровые каналы, по которым перемещается информация в среде как от периферийных ячеек среды, служащих входами поля, так и между спецпроцессорами, представляют собой последовательные сдвигающие регистры, которые имеют возможность не только сдвигать, но и разветвлять информацию (рис.2.5).

Двоичные коды перемещаются по коммутационным регистрам, настроенным в среде, с тактовой частотой. С каждым тактом на входы спецпроцессора поступают последовательно разряд за разрядом, начиная с младших разрядов, двоичные числа. Через каждые n тактов, где n связано с разрядностью обрабатываемых данных, на входы поступает новое машинное слово.

Следует отметить, что дискретный регистровый принцип коммутации исключает затухание информации по мере ее продвижения через систему коммутации и позволяет строго калибровать время прохождения информации. При этом устраняются помехи в разомкнутых регистровых каналах связи, исключается взаимное влияние коммутационных ячеек друг на друга и повышается надежность работы коммутационной структуры в целом.

Таким образом, настройка ОВС на реализацию алгоритма решения задачи осуществляется на двух уровнях: первый уровень – это настройка аппаратной реализации операций в спецпроцессорах, а второй уровень – это структурное отображение алгоритма в ОВС путем организации регистровых каналов связи между спецпроцессорами $СП_i$ (рис.2.5).

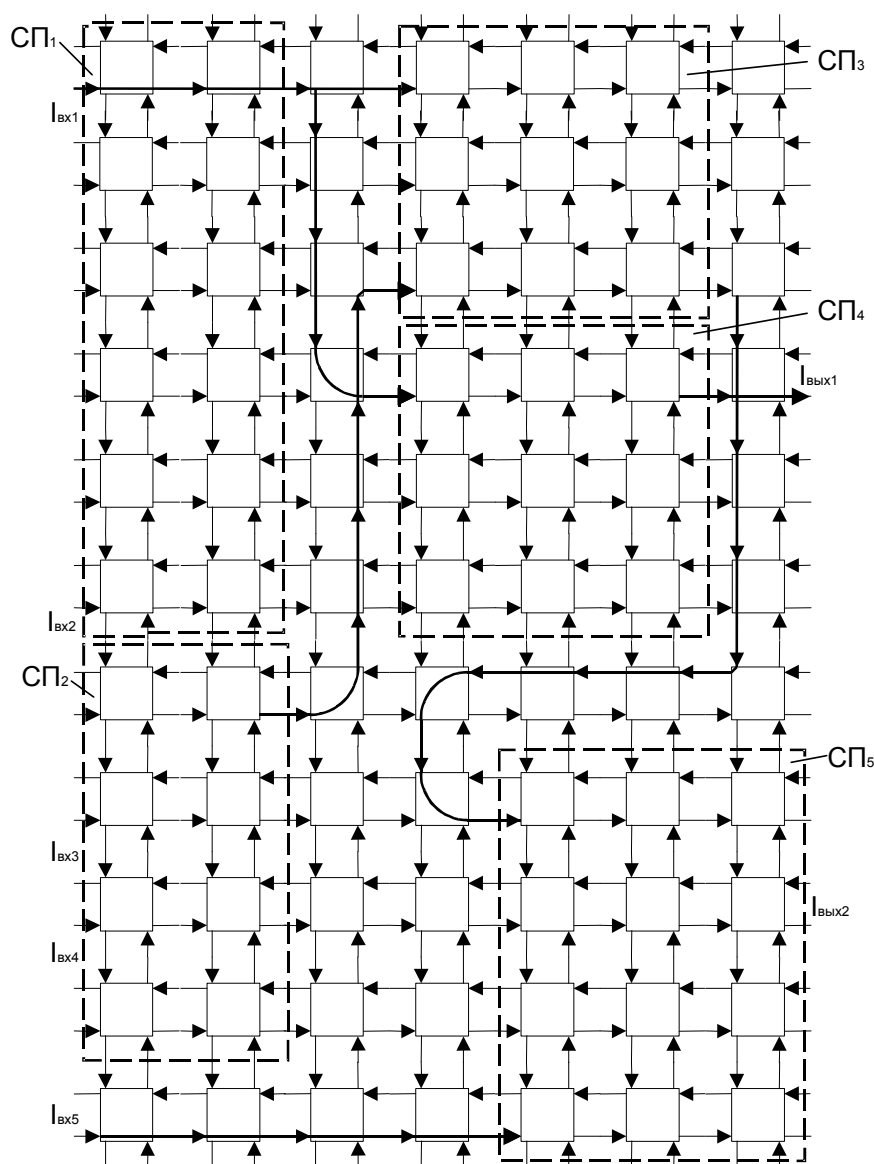


Рис.2.5. Реализация задачи в поле ОВС

2.3. Математическая модель ОВС

Как показано выше, в ОВС различаются два уровня:

- уровень вычислительной ячейки;
- уровень спецпроцессора.

Функциональная схема вычислительной ячейки ОВС была описана выше. Ячейка в среде соединена с четырьмя соседними ячейками. Обозначив входы и выходы ячейки, соответственно, через a_{ij} и b_{ij} , запишем:

$$a_1(i, j) \leftarrow b_3(i, j-1);$$

$$a_2(i, j) \leftarrow b_4(i-1, j);$$

$$a_3(i, j) \leftarrow b_1(i, j+1);$$

$$a_4(i, j) \leftarrow b_2(i+1, j).$$

Вычислительная ячейка является двухоперандной, т.е. осуществляется обработка двух операндов в АЛУ. Одновременно с операциями в АЛУ осуществляется транзит данных, то есть передача их на выходы ячейки без обработки. Результаты обработки, как и транзит данных, могут передаваться на один из своих четырёх выходов ячейки. Настройка ячейки на выполнение той или иной операции и функции транзита осуществляются 16-разрядным кодом, который размещается в регистре программы (см. рис. 2.1).

Разряды 0-2 определяют код операции, разряд 3 устанавливает задержку сигнала в цепи АЛУ, разряды 4 -15 фиксируют адресацию входных и выходных потоков. Содержимое регистра программы ячейки определяется в процессе настройки и остаётся неизменным в течение всей работы ОВС, то есть ячейка настраивается кодом $\langle f, A_1, A_2, A_3, B_1, B_2, B_3 \rangle$, где f - код операции ячейки, A_i - коды настройки входов, B_i - коды настройки выходов.

В вычислительной ячейке следует различать внутренние и внешние входы. Каждая ячейка соединена с четырьмя соседними, то есть имеет четыре внешних или коммутационных входа (выхода). Внутренних входов или выходов собственно в ячейке три: два входа использованы под операнды для АЛУ (A_1 и A_2) и один вход задействован под транзитный канал (A_3). Столько же выходов из ячейки: один выход из АЛУ (B_1) и два выхода (B_2 и B_3). По этим выходам может поступать информация, входящая в транзитный канал.

Выбор функций коммутации и транзита данных задаётся множеством функции $K_i = \{K_1, \dots, K_n\}$, которая осуществляет задание кода адреса внешнего

входа (выхода) ячейки, путём занесения 12-разрядного кода в регистр команд каждой ячейки (с 4 разряда по 15-й разряд).

$$\varphi(k) = \langle A_1, A_2, A_3, B_1, B_2, B_3 \rangle.$$

Код настройки состоит из кода настройки функциональных автоматов и кода настройки коммутационных автоматов.

Вычислительная ячейка относится к М-автомату, который можно классифицировать следующим образом:

- по способу настройки – автомат со структурной настройкой;
- по наличию памяти – автомат с памятью;
- по способу настройки – автомат с отдельными информационным и управляющим входами и с периодической настройкой.

При помощи входных и выходных коммутаторов информация с внешних входов ячейки поступает на входы в операционный или транзитный каналы.

Создание микропрограммного модуля спецпроцессора начинается с разработки алгоритма выполнения данной операции. Этот алгоритм можно представить в виде логической сети или графа, вершины которого представляют собой операции, входящие в систему команд, выполняемой ячейкой ОВС. Затем этот взвешенный граф размещается в поле ОВС. Число ячеек, требующееся для размещения графа, минимизируется и, таким образом, получаем матрицу ячеек ОВС, настройка которой и обеспечивает реализацию определенной арифметической или логической операции. Следовательно, спецпроцессор, настроенный в ОВС, можно описать прямоугольной матрицей, каждый элемент P_{ij} которой есть 16-разрядный двоичный код, определяющий как операцию, выполняемую ячейкой, так и настройку входных и выходных коммутаторов этой ячейки:

$$P = \begin{pmatrix} P_{11} & \dots & P_{1m} \\ \dots & \dots & \dots \\ P_{nl} & \dots & P_{nm} \end{pmatrix},$$

$$P_{ij} = \langle f, A_1, A_2, A_3, B_1, B_2, B_3 \rangle,$$

где f – код операции;

A_1, A_2, A_3 – коды, определяющие подключение входов операционного и транзитного каналов к выходам соседних ячеек;

B_1, B_2, B_3 – коды, определяющие подключение выходов операционного и транзитного каналов к соответствующим входам соседних ячеек среды.

Структурная схема спецпроцессора может быть описана матрицами F , $Q_{вх}$, $Q_{вых}$:

$$F = \begin{pmatrix} f_{11} & \dots & f_{1m} \\ \dots & \dots & \dots \\ f_{n1} & \dots & f_{nm} \end{pmatrix},$$

где $f_{i,j}$ – код настройки ячейки с координатами i, j ,

$$Q_{вх} = \begin{pmatrix} q_{11}(A_1, A_2, A_3) & \dots & q_{1m}(A_1, A_2, A_3) \\ \dots & \dots & \dots \\ q_{n1}(A_1, A_2, A_3) & \dots & q_{nm}(A_1, A_2, A_3) \end{pmatrix},$$

$$Q_{вых} = \begin{pmatrix} q_{11}(B_1, B_2, B_3) & \dots & q_{1m}(B_1, B_2, B_3) \\ \dots & \dots & \dots \\ q_{n1}(B_1, B_2, B_3) & \dots & q_{nm}(B_1, B_2, B_3) \end{pmatrix},$$

где $q_{i,j}(A_1, A_2, A_3)$ и $q_{i,j}(B_1, B_2, B_3)$ – коды, определяющие подключение “внутренних” входов и выходов ячейки.

В определенном спецпроцессоре каждый из четырех “внешних” входов ячейки связан с соответствующим выходом соседней ячейки.

Спецпроцессор, настроенный в ОВС, может правильно функционировать, если операнды будут поступать на его входы в строго определенные моменты времени. Координаты ячеек, служащих входами в спецпроцессор, равно как и такты прихода операндов в модуль, зафиксированы в так называемом *паспорте спецпроцессора*, который будет подробно рассмотрен ниже. В этом же паспорте указаны координаты ячеек, через которые информация выходит из спецпроцессора, а также номера тактов выхода.

ОВС состоит из некоторого числа параллельно, т.е. одновременно функционирующих спецпроцессоров. В ОВС схемно реализуется информационный граф задачи (рис. 2.6). Вершины графа соответствуют

спецпроцессорам, а ребра - связям между ними. Связи реализуются регистровыми каналами, настраиваемыми в поле ОВС (рис. 2.7).

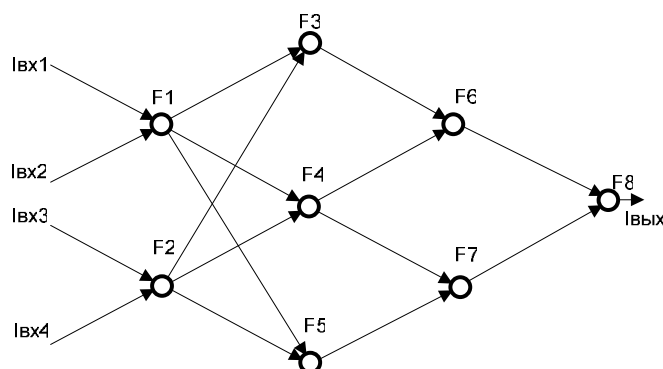


Рис. 2.6. Граф задачи

На рис.2.7 спецпроцессоры F_i , настроенные в однородной среде, показаны одинаковыми прямоугольниками. В действительности, спецпроцессоры, выполняющие различные операции, включают неодинаковое число ячеек среды, то есть занимают различные площади.

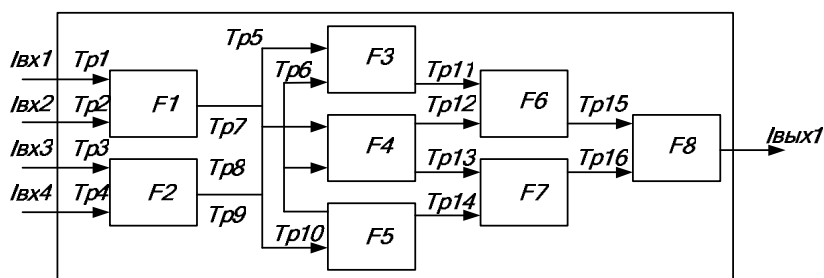


Рис.2.7. Размещение графа задачи в поле ОВС

При этом программирование ОВС на реализацию графа задачи осуществляется с помощью двух групп микропрограмм. К первой группе относятся микропрограммы, состоящие из команд настройки совокупности ячеек, занимающих, как правило, прямоугольную область однородной среды и реализующих некоторую арифметическую, логическую или "крупную" операцию. В среде микропрограммами этой группы настраивается спецпроцессор, выполняющий определенную операцию. В эту группу входят также микропрограммы, настраивающие совокупность ячеек на функцию задержки информации. Так как микропрограммы первой группы состоят из

микрокоманд, настраивающих участки среды, как правило, прямоугольной формы, то эти микропрограммы получили название *микропрограммных модулей*.

К микропрограммам второй группы относятся микропрограммы, состоящие из микрокоманд настройки коммутационных трасс. В среде используется регистровая коммутация, то есть информация от одного спецпроцессора к другому подается не по шинам, а по регистровым структурам [45]. Регистровые сдвигающие структуры образованы из тех же ячеек среды, из каких состоят и спецпроцессоры. По этим регистровым каналам последовательным кодом с тактовой частотой перемещается вся информация, перерабатываемая в ОВС. Помимо "рабочей" информации, по аналогичным регистровым каналам, настроенным в среде, продвигается и вспомогательная информация, в частности, служебные константы, сигналы переполнения. Так как в ОВС имеет место пространственная реализация алгоритма, то любой канал связи между спецпроцессорами, по которому проходит передача информации, любой канал служебной константы или канал контрольной информации представляют собой регистровую структуру. В отличие от микропрограмм первой группы, настраивающих некоторые компактные области среды, микропрограммы второй группы содержат микрокоманды настройки ячеек сдвигающих регистров - коммутационных каналов, имеющих изломы и разветвления. Микропрограммы второй группы носят название *микропрограммы трасс*. Если микропрограммы первой группы, то есть микропрограммные модули, могут храниться в библиотеках, то микропрограммы второй группы всякий раз генерируются заново специальными программами.

Так как в мультиконвейерной структуре настройка среды идет по строкам, то есть регистры команд ячеек, расположенных в одном ряду, соединены последовательно друг с другом, образуя «длинный» регистр, то необходимо производить соответствующую запись в регистры ячеек, не занятых в структурной реализации данного алгоритма. Подготовленная

микропрограмма настройки «свободных» ячеек сортируется, как и все остальные микропрограммы, используемые при настройке алгоритма в ОВС, специальной программой, которая формирует файл настроечной информации в соответствии с принятым в ОВС способом настройки ячеек среды [9].

На рис.2.8 приведена структура микропрограммного обеспечения ОВС.



Рис. 2.8. Структура микропрограммного обеспечения

2.4. Синтез спецпроцессоров в среде

Процесс синтеза спецпроцессора в среде состоит из двух этапов. Вначале происходит разработка микропрограммного модуля спецпроцессора, то есть матрицы микрокоманд. Будучи размещенными в регистрах вычислительных ячеек среды, микрокоманды, входящие в микропрограммный модуль, настраивают некоторую связную область, геометрия которой однозначно определяется микропрограммным модулем, на выполнение той или иной арифметической или логической операции.

Рассмотрим структуру технологического процесса разработки микропрограммных модулей спецпроцессоров (рис. 2.9).

Структурная схема спецпроцессора разделяется на блоки, для каждого из которых разрабатываются графы логической сети с учетом коммутационных возможностей ячеек. Графы логической сети блоков размещаются в однородной среде. Отлаженные блоки спецпроцессора компонуются в итоговую программу, то есть микропрограммный модуль, настраивающий некоторую прямоугольную область однородной среды на реализацию определенной операции. Микропрограммный модуль спецпроцессора включается в библиотеку микропрограммных модулей. Библиотечные микропрограммные модули используются при сборке итоговой настройки ОВС при реализации информационного графа конкретной задачи.

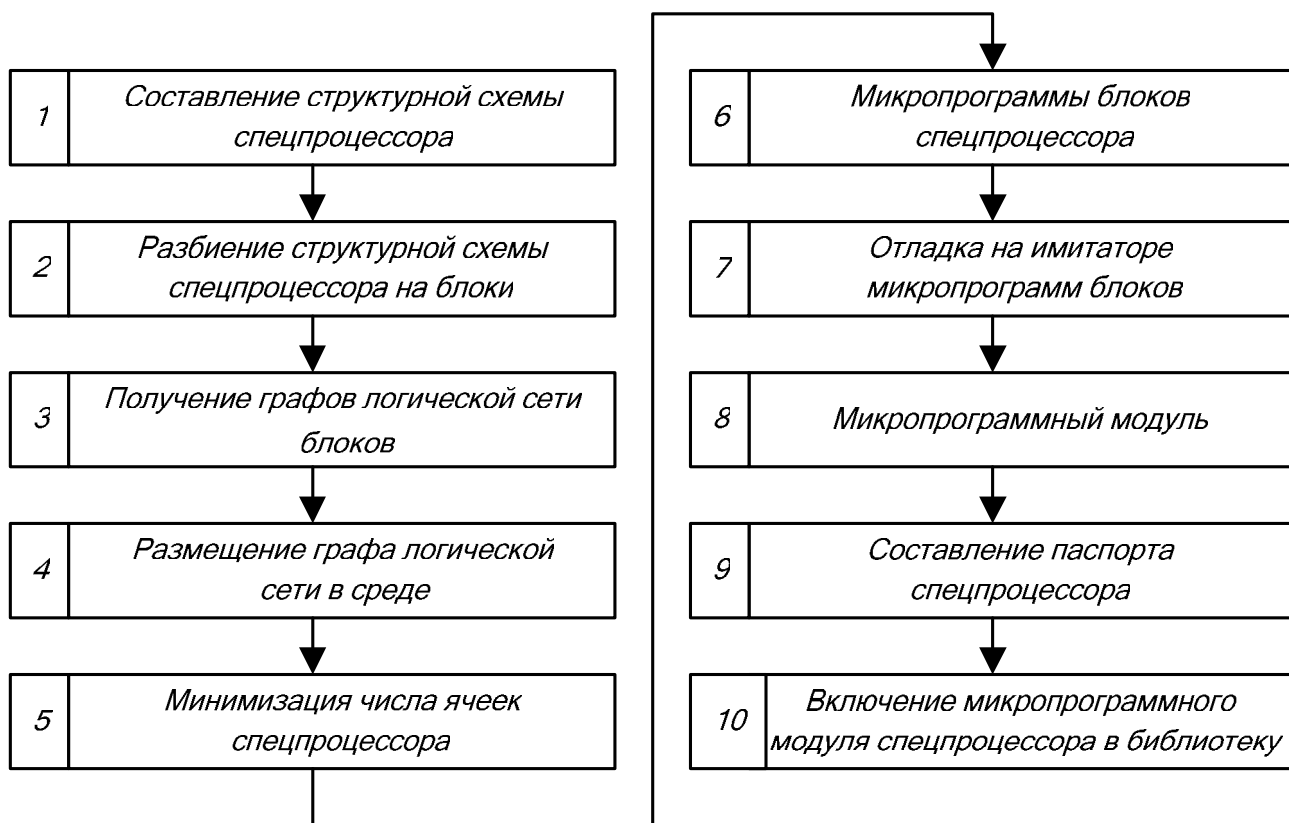


Рис. 2.9. Технологический процесс разработки микропрограммного модуля спецпроцессора

Одним из подходов к построению экономических схем спецпроцессоров в однородных вычислительных средах является метод уравнивания главных осей [36]. В методе уравнивания главных осей информационный граф операции накладывается на координатную сетку среды. Ребра графа ориентируются по осям координат среды, а вершины совмещаются с

определенными ячейками среды. Вводятся понятия оси, весов оси, уравновешенной оси. Понятия оси и уравновешенной оси вводятся на основании количества ребер графа, которые подходят к выбранной оси с правой и левой стороны. Для оптимизации реализованные участки графа перемещаются таким образом, чтобы главные оси были уравновешены. Это ведет к минимизации суммарной длины каналов связи в среде и, тем самым, позволяет уменьшить число ячеек среды, идущих на реализацию графа операции.

Синтезированный спецпроцессор описывается с помощью паспорта спецпроцессора, в общем случае содержащего четыре таблицы: таблицу характеристик спецпроцессора, таблицу входов, таблицу выходов, таблицу транзитов и сегментирования.

Рассмотрим содержание этих таблиц более подробно. Таблица характеристик спецпроцессора содержит следующие сведения.

1. Мнемоника. Она определяет вид операции, выполняемой спецпроцессором, который синтезируется в однородной среде.

2. Тип данных. Указание о типе определяет код и формат данных, с которыми может работать синтезированный в среде спецпроцессор.

3. Разрядность данных. Одно из важных достоинств ОВС - это возможность работать с данными именно той разрядности, которая необходима для получения результата с требуемой точностью. При установлении в программе системного оператора, определяющего разрядность, из библиотеки микропрограммных модулей будет выбран соответствующий модуль согласно реквизиту "Разрядность данных" в паспортной таблице или синтезирован специальной программой.

4. Признак коммутативности входов. Спецпроцессор двухместных операций имеет два входа, на которые поступают операнды А и В. В спецпроцессорах вычитания, деления, сравнения и некоторых других за операндами А и В входы фиксируются.

5. *«Высота» спецпроцессора.* Одной из предпосылок структурной реализации алгоритма в среде является возможность выбрать для конкретной задачи с определенным составом операций и графом информационных связей спецпроцессоры той или иной геометрической конфигурации, которые позволяют наиболее эффективно использовать вычислительные ресурсы однородной среды с учетом конструктивных особенностей и ограничений ОВС.

6. *«Длина» спецпроцессора.* Спецпроцессоры, реализующие ту или иную арифметическую или логическую операцию, включают в себя различное количество ячеек среды. Число ячеек в некоторых спецпроцессорах, например выполняющих операции умножения или деления, пропорционально разрядности обрабатываемых данных.

7. *Период поступления информации.* Однородная вычислительная структура имеет максимальную производительность при непрерывном, разряд за разрядом, с тактовой частотой поступлении операндов на входы спецпроцессоров. Однако в некоторых случаях, когда вычислительные ресурсы недостаточны для структурной реализации алгоритма при непрерывном поступлении информации, может использоваться режим подачи информации с разрежением.

Таблица входов содержит два реквизита: координаты ячеек, в которые поступают операнды или служебная константа, и такты поступления операндов в спецпроцессор относительно единицы константы KI. В таблице выходов определены координаты ячеек, через которые выходит информационный поток, состоящий из слов результата операции, выполняемой в спецпроцессоре, а также указывается число тактов, необходимое для выполнения операции.

В связи с тем, что спецпроцессоры, реализующие крупные операции или работающие с числами в формате с плавающей запятой, а также спецпроцессоры умножения и деления с фиксированной запятой, обрабатывающие данные разрядности 32 или 64, имеют весьма значительную длину, что усложняет установление в ОВС регистровых коммутационных каналов, соответствующих информационному графу алгоритма, то следует

предусмотреть возможность пересечения областей спецпроцессоров, размещенных в среде транзитными каналами. Такая процедура обеспечивается вставкой в матрицу ячеек, реализующих спецпроцессор, столбца вычислительных ячеек, настроенных как на передачу информации, проходящей в спецпроцессоре, так и пересылку транзитной информации. Очевидно, что в произвольном сечении спецпроцессора, синтезированного в среде, нельзя сделать транзитные вставки, не нарушая синхронизации информационных потоков. Такие каналы возможно организовать в сечении спецпроцессора, в котором нет обратного хода информации, то есть перемещения информации справа налево. При разработке микропрограммного модуля спецпроцессора устанавливаются координаты сегментирования, то есть области спецпроцессора, между которыми могут быть вставлены столбцы ячеек, служащих для транзита посторонней информации.

Данные таблиц полностью определяют функционирование спецпроцессоров и служат основой при структурной реализации алгоритма в однородной среде. В табл. 2.2 в качестве примера приведен паспорт рассмотренного в параграфе 2.2 спецпроцессора операции вычитания чисел.

Таблица 2.2

Наименование характеристики		Значение характеристики	
Формат		F2	
Разрядность данных		32	
Коммутативность входов		0	
«Высота» спецпроцессора		3	
«Ширина» спецпроцессора		3	
Период поступления информации		32	
Характеристики входов			
Каналы	Координаты ячеек	Адреса входов	Такты входов
A	3 1	00	3
B	2 1	00	3
K1	1 1	00	1
Характеристики выходов			
Каналы	Координаты ячеек	Адреса выходов	Такты выходов
R	2 3	10	8
K1	1 3	10	6

2.5. Структура микропрограммного обеспечения

Микропрограммное обеспечение содержит микропрограммные модули (МПМ) трех уровней:

- МПМ спецпроцессоров арифметических и логических операций;
- МПМ спецпроцессоров ”крупных“ операций;
- МПМ типовых задач.

Микропрограммные модули первого уровня детально будут рассмотрены в параграфе 2.7.4.

Набор крупных операций $O=\{O_1, O_2, \dots, O_m\}$ должен быть, с одной стороны, минимальным, чтобы обеспечить простоту настройки спецпроцессора на выполнение конкретного алгоритма, а с другой стороны, достаточно универсальным, чтобы обеспечить решение различных задач. Желательно также, чтобы операции совпадали, по возможности, с операциями обычного математического языка. Наконец, алгоритмы выполнения крупных операций должны быть как можно более простыми и обеспечивать в то же время необходимую точность вычислений.

Как отмечено в работе [125], выбрать достаточно удобный универсальный набор укрупненных операций не удастся. Если набор арифметических и логических операций исчисляется единицами, то число «крупных» операций гипотетического универсального набора - сотни и тысячи. Поэтому каждый такой набор, как правило, специализируется на то или иное применение: уравнения математической физики, задачи линейной алгебры, цифровую обработку сигналов и т. д. В работе [46] во множество крупных операций предлагается включить следующие базовые операторы:

- функциональные операторы, реализующие функциональные зависимости;
- дифференциальные операторы, осуществляющие операции дифференцирования по независимой и зависимой переменным, определения частных производных;

- интегральные операторы, выполняющие интегрирование по Риману и Стильтесу;
- матричные операторы, реализующие операции сложения, умножения, транспонирования и др.;
- векторные и тензорные операции.

Структурный синтез спецпроцессоров “крупных” операций можно осуществить из набора микропрограммных модулей спецпроцессоров арифметических и логических операций.

Подобно тому, как формируется библиотека базовых микропрограммных модулей спецпроцессоров “крупных” операций, должна комплектоваться и библиотека типовых задач, эффективно решаемых в однородной вычислительной среде. Наличие типовой задачи в библиотеке микропрограммных модулей гарантирует присутствие в последней базовых микропрограммных модулей спецпроцессоров крупных операций, а также информационного графа задачи и его размещение в однородной среде. Постановка задачи из класса задач, для которого имеется реализация типового варианта, многократно ускоряет процесс подготовки задачи к структурной реализации в среде.

В однопроцессорных ЭВМ микропрограмма - это последовательность микрокоманд. В ОВС - это совокупность микрокоманд, связанных между собой не только временными, но и пространственными соотношениями, так как каждая микрокоманда, входящая в микропрограмму, определяет настройку одной конкретной ячейки, имеющей свои координаты в модуле. При размещении микропрограммного модуля микрокоманда обретает абсолютный адрес. После настройки среды любой ячейке однозначно соответствует микрокоманда, которая определяет автоматную и коммутационную функции, неизменно выполняемые ячейкой в течение всего процесса структурного моделирования алгоритма до перестройки среды на реализацию другого алгоритма.

2.6. Процесс настройки поля ОВС

Блок-схема процесса настройки ОВС на схемную реализацию алгоритма показана на рис. 2.10.

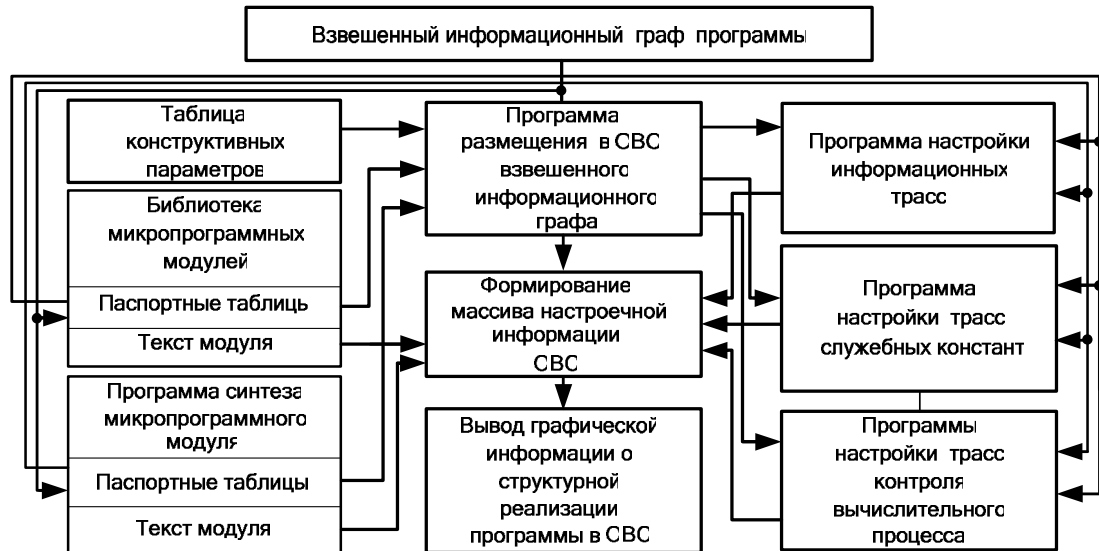


Рис. 2.10. Блок-схема процесса настройки поля ОВС

Программа, записанная на языке программирования среды, транслируется специальным комплексом программ. Одним из первых и важнейших этапов трансляции является построение взвешенного информационного графа исходной задачи, который представляет собой исчерпывающую информацию, позволяющую произвести структурную реализацию алгоритма в однородной среде. Вершинами информационного графа являются операторы, а ребрами - связи, существующие между ними.

Следующий этап - размещение в ОВС взвешенного информационного графа. Программа размещения графа должна учитывать конструктивные параметры вычислительной структуры: ширину поля среды, число входов, имеющиеся неоднородности на стыках плат, каналы быстрого транзита и другие конкретные архитектурные особенности.

После того как определены вершины информационного графа и каждой вершине поставлена в соответствие некоторая арифметическая, логическая или крупная операция, программа размещения обращается к паспортным таблицам спецпроцессоров, в которых отражены все параметры, характеризующие

спецпроцессор, получаемый в результате настройки области среды. Среди этих параметров - геометрические размеры спецпроцессора, координаты входов операндов, служебной константы, координаты ячеек выходов и т. д.

Если используются модули, которые синтезируются специальными программами, то паспортные таблицы сгенерированных спецпроцессоров также поступают в блок размещения информационного графа. С учетом размеров вызываемых спецпроцессоров, настраиваемых в среде, а также числа и характера распределения ребер информационного графа, которые определяют потоки информации между спецпроцессорами, проводится размещение в первом приближении графа в среде.

После того как спецпроцессоры размещены в среде, начинают работать программы настройки и синхронизации регистровых коммутационных каналов. Эти программы также используют паспортные таблицы спецпроцессоров, в которых определены значения тактов прихода в спецпроцессор операндов относительно такта прихода единицы служебной константы.

Программы синхронизации информации могут использовать блоки задержки, применение которых, возможно, повлечет переразмещение спецпроцессоров в среде [3]. После этапа оптимизации, позволяющего наиболее экономно использовать вычислительные ресурсы среды, формируется массив настроенной информации ОВС, в поле которой структурно реализован граф задачи.

Программа формирования итогового массива использует тексты микропрограммных модулей спецпроцессоров, которые определены информационным графом. Используются также микропрограммы, обеспечивающие настройку регистровых коммутационных трасс служебной константы и цепей, контролирующих вычислительный процесс.

Сформированный массив настроенной информации всех ячеек поля среды должен составляться с учетом способа записи настройки в шестнадцатиразрядные регистры ячеек. Для визуального контроля структурной реализации алгоритма в среде и степени эффективности использования

вычислительных ресурсов используется вывод графической информации с обозначением областей, настроенных на реализацию тех или иных операций.

2.7. Программирование однородных сред

2.7.1. Образование и использование констант

При разработке спецпроцессоров, реализующих арифметические и логические операции, широко используются те или иные константы, которые записываются в регистре ячейки или в регистрах группы ячеек при кодах настройки ГК0 или ГК1. Константы необходимы для выделения знака и абсолютной величины числа, выделения, гашения или «растяжения» разряда и для других целей.

Как показала практика микропрограммирования, в спецпроцессорах используется довольно ограниченный набор констант: константы, содержащие одну единицу, и константы, имеющие количество единиц, связанное с разрядностью либо с форматом чисел [27, 85, 124]. Типичные константы встречаются во многих спецпроцессорах, реализующих арифметические или логические операции.

Рассмотрим некоторые группы 16-разрядных констант.

Константы для выделения знака и абсолютной величины:

```
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Константы с единицей для выделения разряда:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
.....
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Вентильные константы:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
.....
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Константы с нулем для гашения разряда:

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
.....
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Константы для «растяжения» разряда, содержащие две единицы и произвольное число нулей между ними:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Специальные константы. К этой группе принадлежат все остальные 16-разрядные константы.

В ОВС, в принципе, можно реализовать вычисления как с произвольной, так и с изменяющейся разрядностью.

При синтезе спецпроцессоров арифметических и логических операций имеются две возможности размещения констант: внутри спецпроцессора и вне его. В качестве примера спецпроцессора со встроенной константой рассмотрим спецпроцессор операции сравнения $|A| < |B|$ для чисел в прямом коде.

Алгоритм операции «Сравнение» $|A| < |B|$:

- инвертируем код числа A ;
- выделяем абсолютную величину числа B ;
- выделяем абсолютную величину числа $\overline{A}$;
- складываем числа $|\overline{A}|$ и $|B|$;
- появление единицы переноса из старшего разряда соответствует выполнению неравенства $|A| < |B|$.

Разряд переноса используется в качестве разряда-признака.

На рис. 2.11 показана структурная схема спецпроцессора операции «Сравнение $|A| < |B|$ » для разрядности $n < 80$.

Разряд-признак выходит из спецпроцессора на s -м такте, где $s = 17 + n$. Начиная с такта s , разряд-признак вырабатывается спецпроцессором с

периодом, равным разрядности входных чисел. Состояние промежуточных разрядов является произвольным.

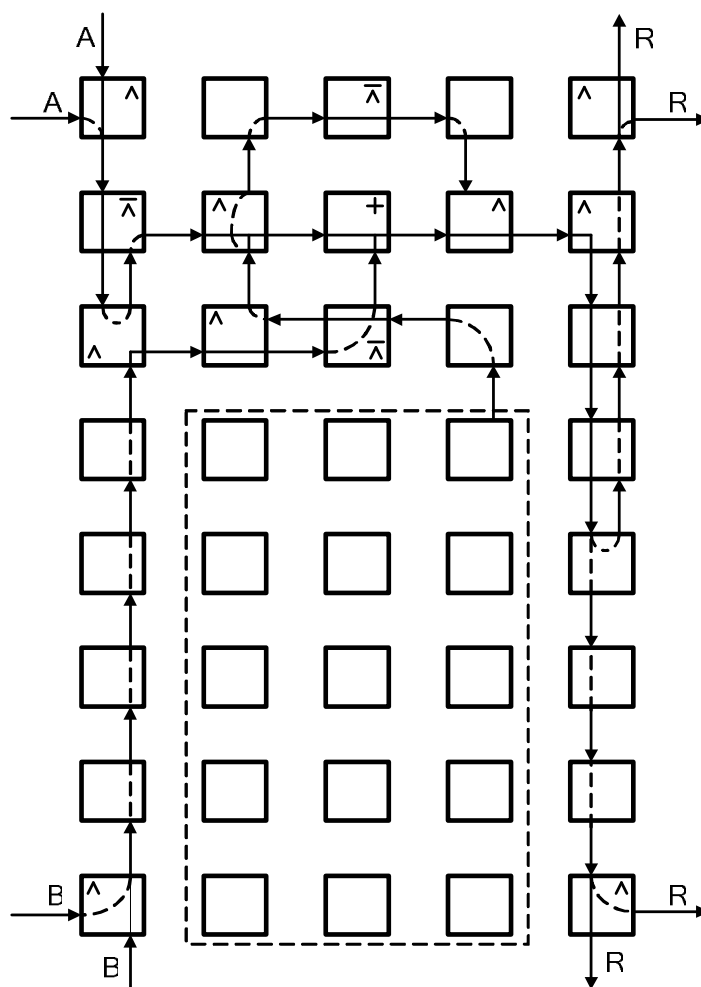


Рис. 2.11. Спецпроцессор со встроенной константой

С ростом разрядности операндов растет число вычислительных ячеек, в регистры которых записывается необходимая для работы спецпроцессора константа. С целью автоматической генерации спецпроцессора обработки данных произвольной разрядности ($n \leq 80$) в спецпроцессоре операции сравнения $|A| < |B|$ зарезервирована группа ячеек, в которых записывается константа необходимой разрядности.

На структурной схеме спецпроцессора, представленной на рис. 2.12, этот блок обозначен штриховой линией.

Включение блока констант в спецпроцессор операции имеет существенные недостатки. Прежде всего, это потеря значительного количества

ячеек на организацию констант в каждом спецпроцессоре, причем удельный вес этих потерь растет с ростом разрядности обрабатываемых данных.

2.7.2. Внешняя подача констант в спецпроцессор

Выше отмечались недостатки, свойственные построению спецпроцессоров со встроенными константами. Для преодоления трудностей был предложен подход, связанный с внешней подачей в спецпроцессор только одной базовой константы. Всё многообразие необходимых констант получается их конструированием из этой базовой константы в спецпроцессоре [8].

Базовая константа, реализуемая в спецпроцессоре константы, передается по регистровым каналам ОВС ко всем спецпроцессорам, размещенным в среде.

Использование внешней базовой константы влечет за собой изменения принципов построения спецпроцессоров. В спецпроцессоре необходим дополнительный вход для подачи базовой константы. Базовая константа выдается в неизменном виде из спецпроцессора через дополнительный выход вместе с результатами операций.

Для подачи базовой константы и констант, используемых при вычислениях, необходимо наличие самостоятельных спецпроцессоров констант, которые располагаются на плате ОВС наряду со спецпроцессорами операций. Спецпроцессор констант обладает регулярностью и позволяет генерировать любую константу.

На рис. 2.12 показана структурная схема спецпроцессора операции сравнения $|A| \leq |B|$ для n -разрядных чисел с фиксированной запятой, представленных в прямом коде.

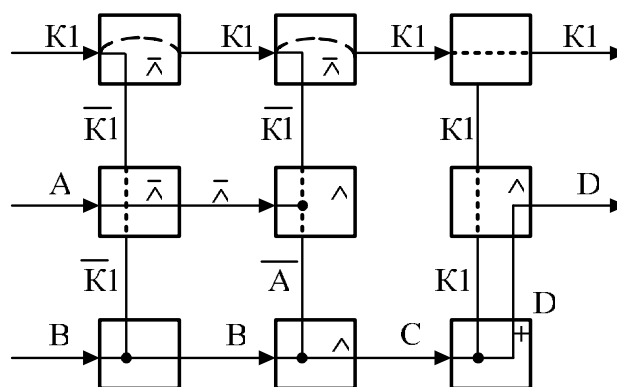


Рис. 2.12. Спецпроцессор сравнения

В отличие от спецпроцессора операции сравнения $|A| < |B|$ со встроенной константой в спецпроцессора, представленном на рис. 2.11, базовая константа $K1$ (0...01) подается на спецпроцессор извне. В результате использования внешней константы спецпроцессор операции $|A| \leq |B|$ имеет всего девять вычислительных ячеек, причем размеры спецпроцессора не зависят от размерности обрабатываемых данных. Важное свойство независимости размеров спецпроцессора в случае внешней подачи константы от разрядности обрабатываемых данных существует и при реализации других операций, например операции сложения.

2.7.3. Форматы операндов

В ОВС двоичные числа поступают последовательным кодом младшими разрядами вперед. Для двоичных чисел с фиксированной запятой, имеющих разрядность n , в прямом коде используется обычно формат, представленный на рис. 2.13а. Однако при разработке спецпроцессоров, которые предназначены для выполнения арифметических операций при непрерывном поступлении информации последовательным кодом, представление чисел в данном формате встречает существенные затруднения. При выполнении операции алгебраического сложения чисел эффективен формат числа, показанный на рис. 2.13б. Чтобы обеспечить такой формат при непрерывном поступлении информации в формате рис. 2.13а, необходимо распараллеливать машинное слово внутри спецпроцессора с последующим объединением потоков. Однако

способ распараллеливания входного формата внутри спецпроцессоров приводит к резкому, более чем вдвое, увеличению числа ячеек в спецпроцессорах [7, 11].

Можно предложить иной метод, не приводящий к увеличению числа ячеек в спецпроцессоре, заключающийся во включении в формат двух буферных разрядов (рис. 2.13в).

Таким образом, числа в прямом коде (формат Ф1) представляются как последовательность битов, в которых два самых левых разряда являются буферными, следующий за буферными разряд – знаковый; в остальных разрядах записывается дробная часть числа.

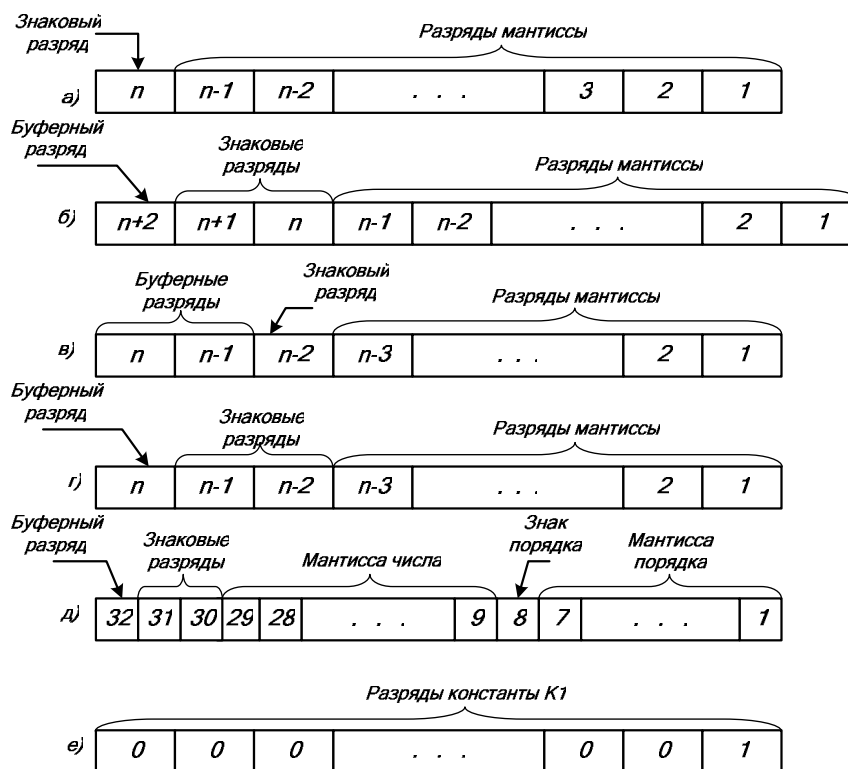


Рис. 2.13. Форматы операндов

Точка предполагается между 3-м и 4-м разрядами. Числа в модифицированном дополнительном коде (формат Ф2) имеют один буферный разряд, два последующих разряда - знаковые, в остальных записывается дробная часть числа (рис. 2.13з).

Числа в формате с плавающей запятой (формат П) представляются в виде следующей последовательности 32 битов: самый левый разряд - буферный,

последующие два разряда - знаковые, после знаковых разрядов 21 разряд занимает мантисса числа, 20-й разряд определяет знак порядка и оставшиеся 7 разрядов - мантисса порядка (рис. 2.13д). Форматы Ф1 и Ф2 могут содержать произвольное число разрядов, однако в качестве основных приняты форматы, содержащие 8, 16 и 32 разряда.

Помимо форматов Ф1, Ф2 и П, в ОВС могут обрабатываться и другие форматы, например константа К1, имеющая вид, представленный на рис. 2.13е, - двоичные коды.

2.7.4. Спецпроцессоры арифметических и логических операций

В связи с тем, что в ОВС операции реализуются аппаратно, существенное значение имеет использование таких кодов и форматов данных, которые обеспечивают наименьшие затраты вычислительных ресурсов среды при выполнении той или иной операции. Другими словами, МПМ спецпроцессоров арифметических и логических операций должны содержать минимальное число микрокоманд. Каждая микрокоманда модуля, идентифицированная координатами (i, j) , является программой настройки ячейки с этими же относительными координатами (i, j) в некоторой связной области среды, настраиваемой совокупностью микрокоманд микропрограммного модуля на аппаратную реализацию конкретного спецпроцессора арифметической или логической операции.

Абсолютный адрес настраиваемой ячейки, входящий в k -й спецпроцессор, размещенный в однородной среде, определяется координатами (p_k+i, q_k+j) , где (p_k, q_k) - координаты левой верхней ячейки спецпроцессора, имеющего k -й номер. Координаты (p_k, q_k) может устанавливать программист при размещении спецпроцессора в поле ОВС в соответствии с графом структурно реализуемого в среде алгоритма, которые, однако, в процессе автоматической прокладки регистровых коммутационных каналов,

синхронизации информационных потоков и оптимизации, возможно, претерпят изменения.

Рассмотрим схемы спецпроцессоров, реализующих арифметические и логические операции на ячейках однородной среды. Структура технологического процесса разработки микропрограммных модулей спецпроцессоров была рассмотрена выше.

Спецпроцессоры операций сложения и вычитания

Спецпроцессоры обрабатывают числа в модифицированном дополнительном коде (МДК) во всем диапазоне их допустимых значений. Числа поступают в спецпроцессор на разных тактах, что обеспечивает оптимальную реализацию операции. Нумерация тактов идет относительно такта прихода константы $K1$ вида $00...01$. Разрядность константы совпадает с разрядностью операндов. Этот такт считается первым. Константа выводится из спецпроцессора в неизменном виде.

Так как обработка переполнения разрядной сетки в ОВС связана с выводом сигнала переполнения из спецпроцессора операции на периферийные ячейки поля по регистровым каналам, образованным из вычислительных ячеек (что связано с большими затратами оборудования), то были разработаны версии спецпроцессоров операций сложения и вычитания, не содержащие сегменты индикации переполнения. Спецпроцессоры операций сложения и вычитания могут оперировать числами произвольной разрядности, которая задается длиной константы $K1$.

Операция сложения. Числа A и B в формате Ф2 (модифицированный дополнительный код $ozza_{n-3}...a_1$) поступают в спецпроцессор операции сложения на тактах, соответственно, 1 и 2 и складываются с переносом. Затем происходит очистка буферного разряда с помощью инвертированной константы $K1$. Структурная схема спецпроцессора операции сложения без индикации переполнения приведена на рис. 2.14.

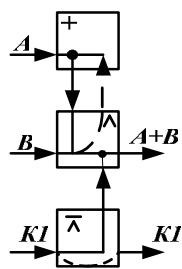


Рис. 2.14. Спецпроцессор
операции сложения

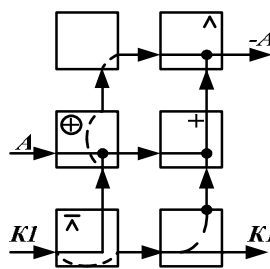


Рис. 2.15. Спецпроцессор операции
изменения знака

Операция изменения знака. Эта операция предназначена для изменения знака числа на противоположный. Для чисел в МДК эта операция сопряжена с дополнительными трудностями - необходимо предусматривать обработку нулевого кода 0110...0. Структурная схема спецпроцессора операции изменения знака дана на рис. 2.15.

Операция вычитания. Над вычитаемым (операнд В) проводятся все действия, осуществляемые при выполнении операций изменения знака числа. Затем числа А и (- В) складываются с переносом и очищается буферный разряд. Структурная схема спецпроцессора операции вычитания чисел в МДК показана в параграфе 2.2.

Операция сложения с выделением переполнения

Спецпроцессор операции сложения чисел в МДК с выделением переполнения реализован по следующему алгоритму:

- сложение побитно операндов с переносом переполнения;
- выделение сигнала переполнения;
- формирование двойного знака результата сложения;
- обнуление буферного разряда результата сложения.

Структурная схема спецпроцессора сложения показана на рис. 2.16. Операнд А подается в ячейку матрицы ячеек с координатами (3, 1) на втором такте, операнд В подается в ячейку (4, 1) на первом такте. Константа К1 поступает в ячейку (2, 1) на первом такте. Результат сложения выводится из ячейки (2.3) на десятом такте.

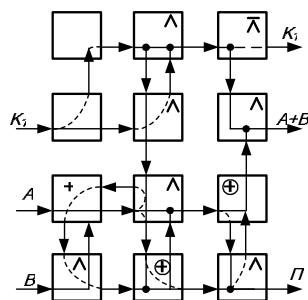


Рис. 2.16. Модифицированный спецпроцессор операции сложения

Спецпроцессор операции умножения

Числа в ОВС могут поступать как непрерывно одно за другим ($\rho = 1$), так и с некоторыми интервалами между ними ($\rho < 1$). При поступлении чисел с $\rho < 1$ спецпроцессоры для ряда операций, например для операции умножения, требуют для своей реализации значительно меньшего числа вычислительных ячеек.

Рассмотрим спецпроцессоры операции умножения чисел с фиксированной запятой в прямом коде при $\rho = 0,5$. Алгоритм, который выполняется спецпроцессором операции умножения, следующий:

- формирование знака произведения как суммы по модулю два знаков множимого и множителя. Знак произведения записывается в знаковый разряд множителя;

- умножение младшими разрядами вперед. Выделяется по одному разряду множителя, начиная с младшего. Логически умножаются разряды множимого на выделенный разряд множителя и результаты суммируются. Очистка младших разрядов накапливаемой суммы выполняется после трех суммирований. Это возможно благодаря наличию двух буферных и знакового разрядов;

- присоединение знака результата к произведению, накопленному в канале суммирования.

Структурная схема спецпроцессора операции умножения 8-разрядных чисел в прямом коде ($\rho = 0,5$) показана на рис. 2.17.

Спецпроцессор имеет размеры 4x11 ячеек. Операнды поступают на 3-м и 5-м тактах в ячейку с координатами, соответственно, (4, 1) и (2, 1). Служебная

константа $K1\ 0...01$ поступает в ячейку (3, 1) на первом такте. Результат умножения поступает на выход спецпроцессора с ячейки (4, 11) на двадцать шестом такте. Разработаны спецпроцессоры операции умножения чисел, поступающих непрерывно одно за другим ($\rho = 1$).

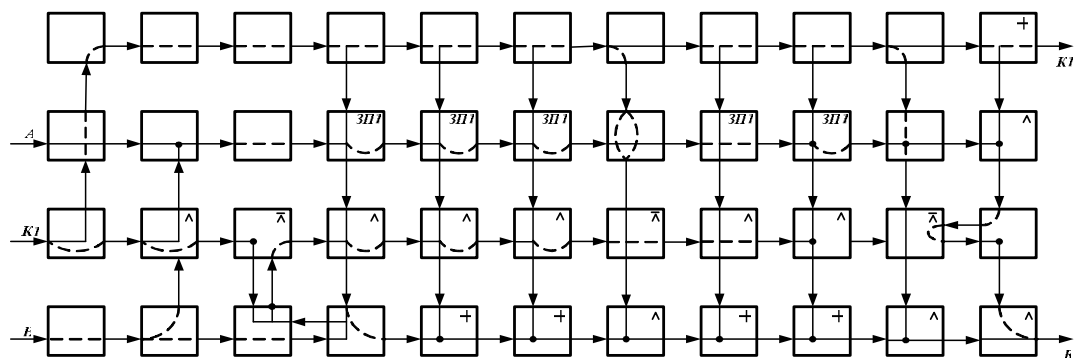


Рис. 2.17. Спецпроцессор операции умножения ($\rho = 0,5$)

Умножение выполняется, начиная с младших разрядов. Все разряды множимого умножаются на фиксированный разряд множителя. Каждый разряд частичного произведения получается логическим умножением соответствующих разрядов сомножителей. Полученные частичные произведения суммируются со сдвигом.

Умножение в двоичном коде эффективно выполняется в прямом коде. Знак произведения определяется как сумма по модулю два знаков сомножителей. Чтобы разрядность входных и выходных данных совпадала, в спецпроцессоре операции умножения результат округляется. Матрица ячеек среды, занимаемая спецпроцессором умножения чисел с фиксированной запятой в прямом коде (Ф1) при непрерывном поступлении информации, имеет размеры 8×23 .

Умножение чисел с фиксированной запятой в модифицированном дополнительном коде

При умножении в МДК поступающая информация переводится в прямой код, а выходная информация – в МДК. На рис. 2.18 показана схема спецпроцессора умножения 8-разрядных чисел в МДК. Сомножители A и B поступают в спецпроцессор через ячейки с координатами (3, 1) и (6, 1) на пятом

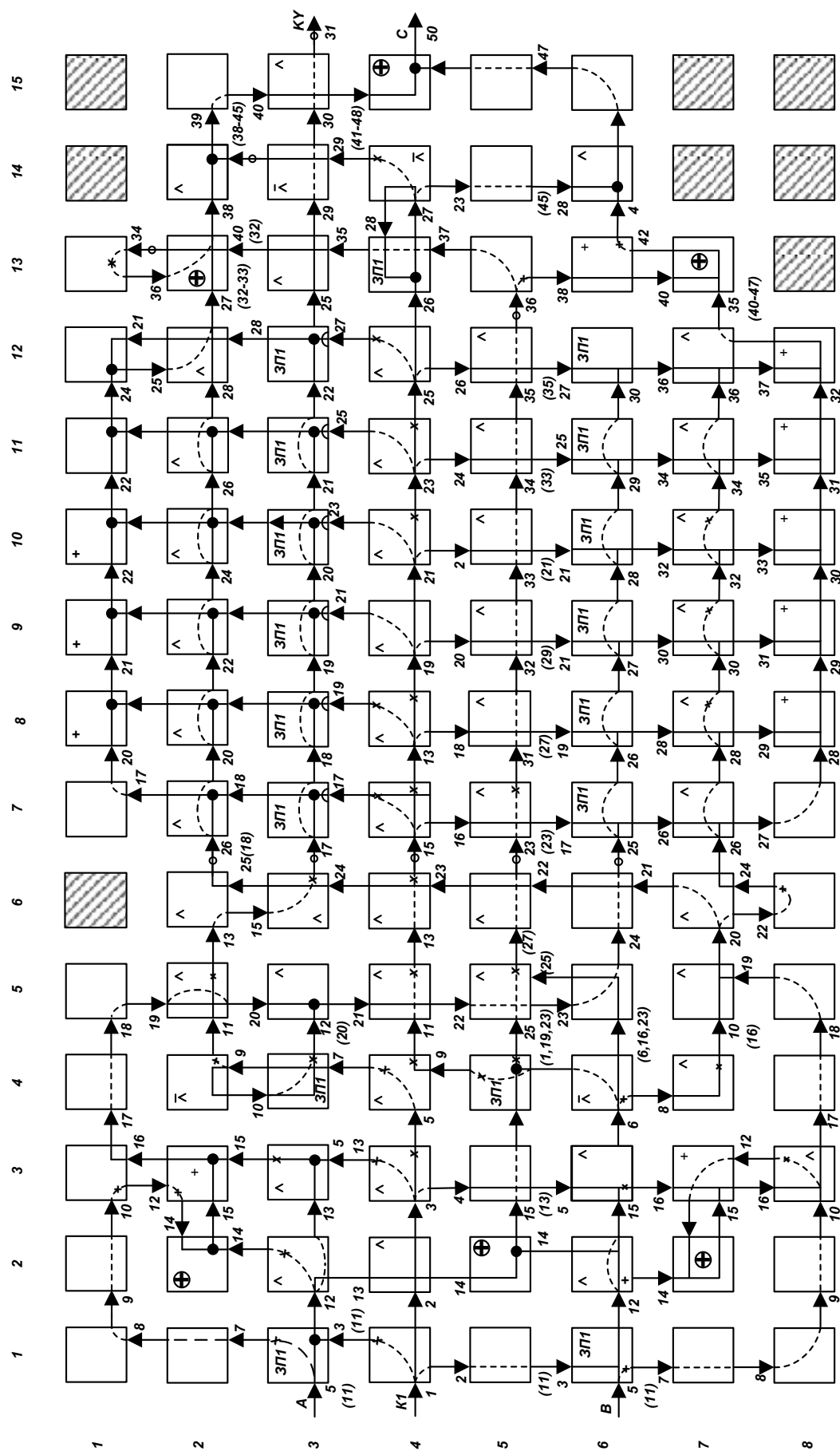


Рис. 2.18. Специализированная операция умножения

такте. На первом такте в спецпроцессор на вход ячейки (4, 1) подается единица константы K1, имеющей вид 00000001. На тридцать первом такте из спецпроцессора умножения выходит единица константы K1. Начиная с пятидесятого такта, в случае непрерывного поступления в спецпроцессор умножения разрядов сомножителей с выхода ячейки (4, 15) снимаются разряд за разрядом 8-разрядные произведения в формате Ф2.

Спецпроцессор умножения 8-разрядных чисел в МДК содержит матрицу вычислительных ячеек размером 8x15, т.е. аппаратные затраты составляют 120 ячеек. Размеры матриц вычислительных ячеек спецпроцессора операции умножения 16- и 32-разрядных чисел в МДК составляют, соответственно, 8x23 и 8x39 ячеек, т.е. спецпроцессоры умножений содержат 184 и 312 ячеек.

Из рассмотрения структурных схем спецпроцессора операции умножения чисел различной разрядности легко заметить, что размеры спецпроцессора операции умножения прямо пропорциональны разрядности входных данных. Обработка дополнительного разряда требует включения в спецпроцессор одного столбца ячеек (рис. 2.19).

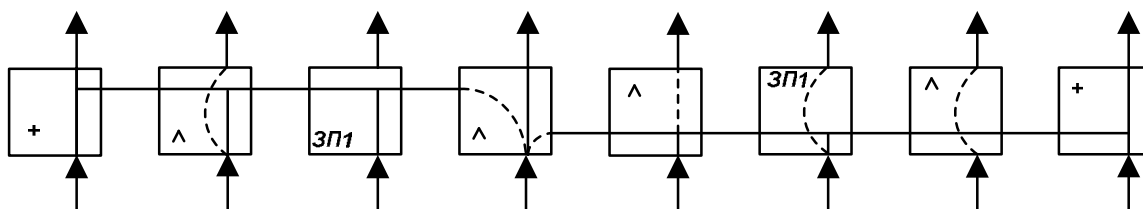


Рис. 2.19. Реализация разряда в спецпроцессоре операции умножения

Следовательно, автоматически программным путем может быть синтезирована микропрограмма спецпроцессора операции умножения для обработки данных произвольной разрядности.

Спецпроцессоры операции деления

Операция деления является самой трудоемкой арифметической операцией - на ее выполнение в ЭВМ затрачивается гораздо больше времени, чем на выполнение других арифметических операций. Естественно, что при

реализации операции деления аппаратным путем в однородной среде эта операция требует значительных затрат вычислительных ресурсов.

В современных ЭВМ операция деления обычно проводится с помощью выполнения последовательности операций вычитания и сдвига. Наибольшее распространение получили алгоритмы «Деление с восстановлением остатка» и «Деление без восстановительного остатка». Оказалось, что в рассматриваемой однородной среде более эффективно, с меньшими аппаратными и временными затратами, реализуется алгоритм «Деление с восстановлением остатка».

Деление двоичных чисел начинается с вычитания делителя из делимого. Если получившийся остаток не отрицателен, то в разряд первой цифры частного заносится единица. Затем получившийся остаток сдвигается на один разряд влево, вычитание повторяется и т.д. Если же остаток получился отрицательный, то в соответствующий разряд частного заносится нуль, к полученному остатку прибавляется делитель, после чего восстановленный остаток сдвигается на один разряд влево и т.д.

Структурную схему спецпроцессоров операции деления чисел в прямом коде можно условно разбить на две части: постоянную и постоянно-переменную. В постоянной части спецпроцессоров операции проверяется условие $|A| \geq |B|$, где A – делимое, B – делитель, и происходит преобразование прямого кода делителя в МДК. Знак частного определяется как сумма по модулю два знаков делимого и делителя.

Постоянно-переменная часть состоит из нескольких, в зависимости от точности получения частного, одинаковых блоков, результатом каждого из которых есть получение одного разряда частного.

Структурная схема спецпроцессора операции деления для 8- и 16-разрядных чисел с внешней подачей константы занимает матричное поле вычислительных ячеек, соответственно, на 8×20 и 8×44 ячеек.

Структурная схема спецпроцессоров операции деления для 8-разрядных чисел приведена на рис. 2.20.

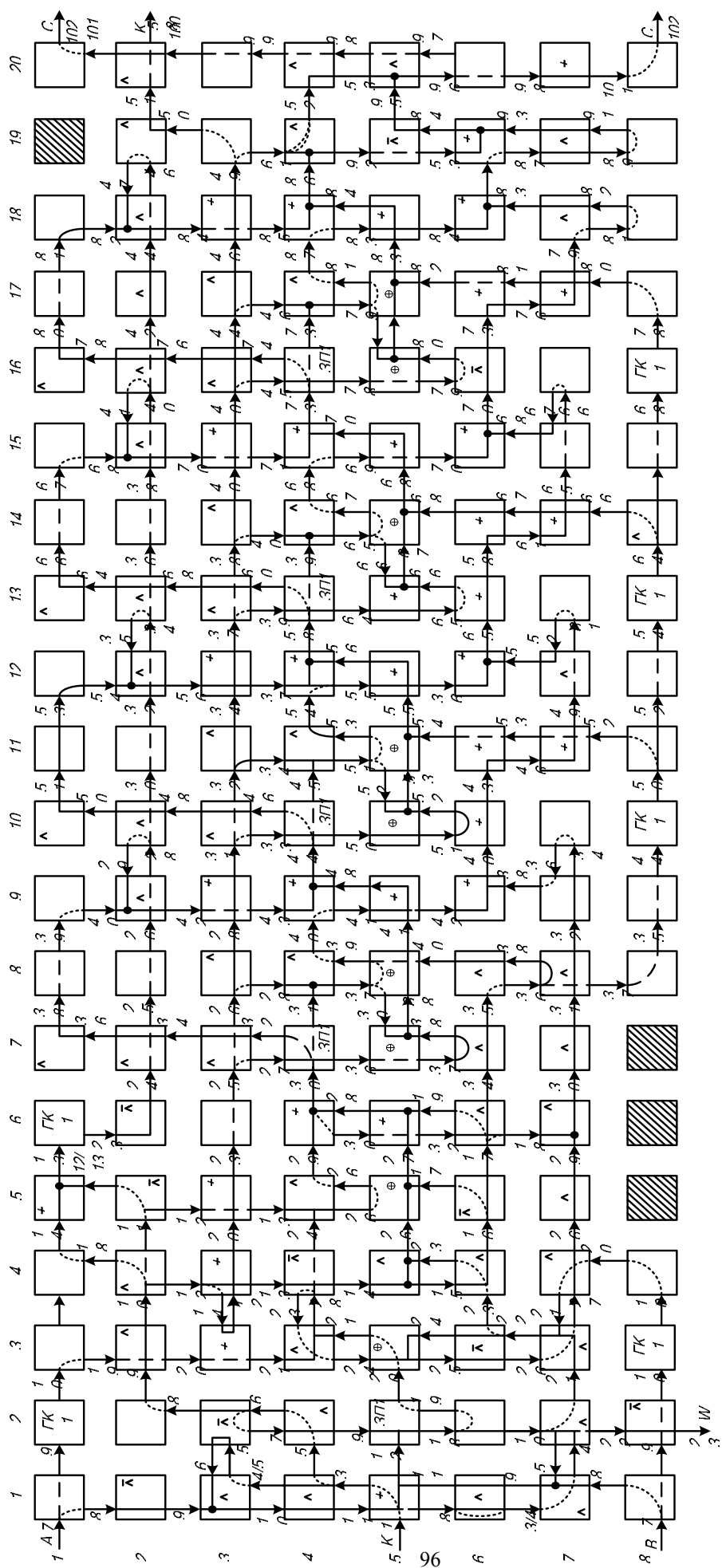


Рис. 2.20. Специализированный процессор деления

Кратко опишем структурную схему. Потoki чисел $\bar{A}$ и $\bar{B}$ поступают в прямом коде в формате Ф1: $ooza_{n-3}...a_1$. Константа вида 00000001 поступает на вычислительную ячейку (5, 1). Признак переполнения снимается с ячейки (8, 2) на 23-м такте. Частное от деления чисел C появляется на выходах ячеек с координатами (1, 20) и (8, 20) на 102-м такте. Константа K1 выходит из спецпроцессора на 52-м такте.

Размеры матрицы ячеек среды, занимаемой спецпроцессором деления 16-разрядных чисел, составляют 8x44. Первый разряд частного поступает на выходы ячеек с координатами (1, 44) и (8, 44) на 326-м такте. Для получения одного разряда частного используется блок из трех столбцов (рис. 2.21).

Включая в спецпроцессор операции деления необходимое число блоков получения разрядов частного, можно сформировать спецпроцессор для обработки данных любой заданной разрядности.

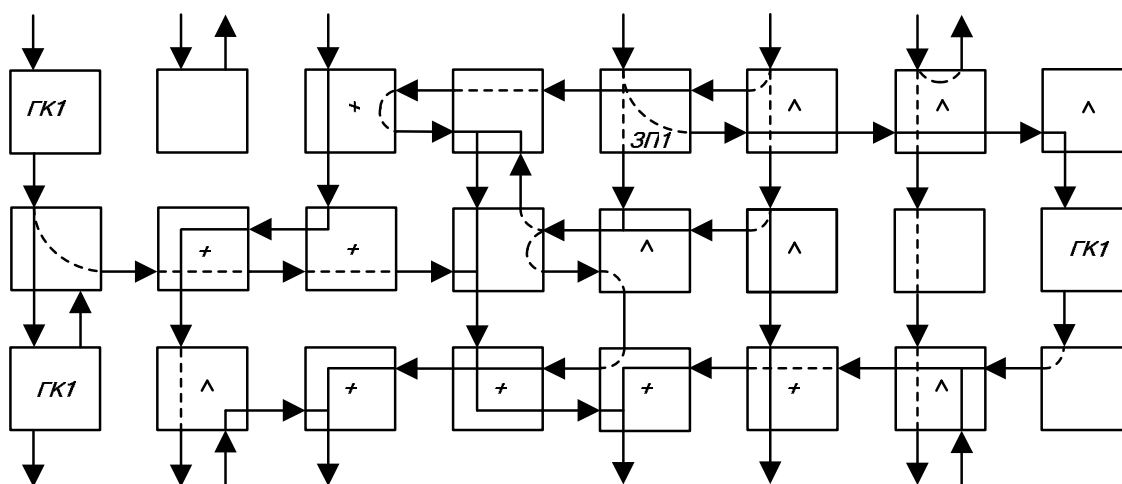


Рис. 2.21. Реализация разряда в спецпроцессоре деления

Аналогичным образом строятся спецпроцессоры операций сравнения, логических и арифметических сдвигов, модули преобразования кодов и других операций.

В табл. 2.3 приведены сравнительные характеристики спецпроцессоров арифметических и логических операций для чисел с фиксированной точкой, в частности, в третьей колонке показано число ячеек ОВС, необходимых для их

реализации. В последней колонке табл. 2.3 указано число тактов, необходимых для получения результата выполнения операции.

Таблица 2.3

№	Наименование микропрограммного модуля	Размер модуля	Число тактов
1	Сложение	3x1	5
2	Сложение с контролем переполн.	4x3	10
3	Вычитание	3x3	8
4	Вычитание с контролем переполн.	4x3	15
5	Умножение 8-и разрядных чисел	4x10	27
6	Умножение 16-и разрядных чисел	4x21	46
7	Умножение 32-х разрядных чисел	4x44	87
8	Деление 8-разрядных чисел	8x23	93
9	Деление 16-и разрядных чисел	8x46	308
10	Деление 32-х разрядных чисел	10x94	1073
11	Сравнение $A = B$	3x1	4
12	Сравнение $A \geq B$	3x2	N+13
13	Сравнение $A > B$	3x2	N+13
14	Сравнение $A \leq B$	3x2	N+13
15	Сравнение $A < B$	3x2	N+13
16	Сравнение $A = 0$	3x1	N+13
17	Сложение абсолютных величин	3x2	9
18	Вычитание абсолютных величин	4x5	24
19	Сравнение модулей $ A = B $	3x1	5
20	Сравнение модулей $ A \geq B $	3x2	10
21	Сравнение модулей $ A > B $	3x3	10
22	Сравнение модулей $ A \leq B $	3x3	10
23	Сравнение модулей $ A < B $	3x3	10
24	Сравнение модулей $ A = 0$	2x2	N + 6
25	Сдвиг логический влево 8-разр. чисел	6x7	17
26	Сдвиг логический влево 16-разр. чисел	6x9	20
27	Сдвиг логический влево 32-разр.. чисел	6x11	23
28	Сдвиг логический вправо 8-разр. чисел	6x7	27
29	Сдвиг логический вправо 16-разр. чисел	6x9	40
30	Сдвиг логический вправо 32-разр. чисел	6x12	61
31	Арифметический сдвиг влево 8-разр. чисел	6x7	18
32	Арифметический сдвиг влево 16разр.чисел	6x9	21
33	Арифметический сдвиг влево 32-разр.чисел	6x11	24
34	Арифметический сдвиг вправо 8-разр.чисел	6x7	27
35	Арифметический сдвиг вправо 16-разр.чисел	6x9	39
36	Арифметический сдвиг вправо 32-разр.чисел	6x13	61
37	Преобразование кодов	4x4	25

Следует обратить внимание, что некоторые операции, например сложение двух чисел, выполняются спецпроцессором, размеры которого не зависят от разрядности обрабатываемых данных. Размеры спецпроцессора операций умножения, деления и сдвигов прямо пропорциональны разрядности данных, поступающих на входы.

Спецпроцессоры арифметических операций для чисел с плавающей запятой

В ОВС двоичное число с плавающей запятой имеет следующее представление: мантисса порядка занимает 1-7-й разряды; знак порядка - 8-й разряд; мантисса числа помещается в 9-29-й разряды; знак числа находится в 30-м и 31-м разрядах; буферный разряд занимает 32-й разряд.

Спецпроцессоры операций сложения и вычитания чисел с плавающей запятой

При сложении (вычитании) чисел с плавающей запятой операции выполняются в следующей последовательности: вычитание порядков; формирование константы сдвига числа с меньшим порядком; арифметический сдвиг вправо мантиссы с меньшим порядком; выполнение операции сложения (вычитания) над мантиссами; нормализация полученной мантиссы; формирование порядка результата.

При реализации арифметических операций с плавающей запятой возникает необходимость разделять поступающую информацию на два потока: мантиссу и порядок. Этой цели служит блок распараллеливания.

На его вход поступают два операнда, на выходе имеем четыре числа: мантисса первого операнда со знаком - разряды с 9-го до 32-го, разряды с 1-го по 8-й - нулевые; порядок первого операнда со знаком - разряды с 1-го по 8-й, остальные разряды - с 9-го до 32-го - нулевые; мантисса второго операнда и порядок второго операнда. Нулевые разряды выполняют функции буферных разрядов.

Рассмотрим спецпроцессор операции умножения чисел с плавающей запятой. Пусть имеются два числа с плавающей запятой: $A=M_1 \cdot 2^{P_1}$ и $B=M_2 \cdot 2^{P_2}$, где M_1 и M_2 - мантиссы чисел; P_1 и P_2 - порядки. Произведение этих чисел равно $AB=M_1M_2 \cdot 2^{P_1+P_2}=M_3 \cdot 2^{P_3}$.

Учитывая непрерывное разряд за разрядом поступление информации, в спецпроцессоре операции умножения чисел с плавающей запятой, как и в спецпроцессоре операции умножения чисел с фиксированной запятой, осуществляется распараллеливание потока информации. Умножение мантисс чисел в спецпроцессоре реализуется аналогично умножению чисел с фиксированной запятой. Полученный порядок произведения анализируется на переполнение. Если порядок не превышает установленного предела, то он присоединяется в одном из сегментов спецпроцессора к нормализованной мантиссе.

Размеры матрицы ячеек, занимаемой спецпроцессором операции умножения чисел с плавающей запятой, составляют 8×42 , т.е. в спецпроцессоре используются 336 ячеек среды.

При выполнении деления чисел с плавающей запятой мантисса частного определяется делением мантиссы делимого на мантиссу делителя, а порядок частного - вычитанием порядка делителя из порядка делимого. При делении чисел с плавающей запятой нормализованная мантисса делимого по абсолютной величине может превышать нормализованную мантиссу делителя, хотя и не более чем в 2 раза.

При настройке спецпроцессора операции для деления чисел с плавающей запятой используется матрица ячеек размером 8×73 , т. е. 584 ячейки среды.

Спецпроцессоры крупных операций

Спецпроцессоры крупных операций могут быть скомплектованы из библиотечных МПМ спецпроцессоров элементарных операций.

В качестве примеров спецпроцессоров крупных операций можно привести реализацию спецпроцессоров перевода чисел из десятичной системы

счисления в двоичную и, наоборот, перевода углов из градусной меры в радианную.

1. Спецпроцессор перевода чисел из десятичной системы счисления в двоичную разработан для 16-, 32- и 64-разрядных чисел. Целые десятичные числа, заданные в прямом двоично-десятичном коде, переводятся в двоичные числа, представленные в МДК. Спецпроцессор двухвходовый; помимо преобразуемого числа в него поступает служебная константа К1. Спецпроцессоры преобразования 16-, 32- и 64-разрядных чисел требуют для своей реализации соответственно 35, 54 и 102 ячеек. Задержка в спецпроцессорах составляет 46, 63 и 103 такта.

2. Спецпроцессоры перевода целых чисел из двоичной системы счисления в десятичную также разработан для 16-, 32- и 64 разрядов. В спецпроцессор поступают целые числа в двоичной системе счисления в МДК и служебная константа К1. На выход спецпроцессора двоично-десятичные числа в прямом коде поступают, соответственно, на 82-м, 130-м и 215-м тактах. При реализации используются 90, 174 и 294 ячеек при обработке 16-, 32- и 64-разрядных данных.

3. Спецпроцессоры перевода углов из градусной меры в радианную. Значения углов в радианной мере в МДК для 16-, 32- и 64-разрядных чисел поступают на выходы спецпроцессоров, использующих, соответственно, 30, 98 и 132 ячеек на 28-м, 85-м и 158-м тактах.

Помимо библиотек микропрограммных модулей, настраивающих спецпроцессоры на выполнение операций с числами в форматах фиксированной и плавающей запятой, была разработана библиотека микропрограммных модулей для настройки спецпроцессоров, реализующих арифметические и логические операции с целыми числами.

Был разработан целый ряд библиотек микропрограммных модулей для настройки спецпроцессоров, реализующих в ОВС арифметические операции с комплексными числами, матричные операции, алгоритмы корреляционного анализа со знаковой дельта-модуляцией, алгоритмы нерекурсивной фильтрации

и т.д. [8,9,14,25,26,84]. В целом ряде публикаций детально рассматривались вопросы автоматизации программирования ОВС [5,11,27,85,127] и перспективные ячейки однородной вычислительной среды [7,56,58,103,106].

2.8. Решение прикладных задач на ОВС

2.8.1. Задача обработки изображений

Имеется ряд публикаций, в которых рассматривается эффективность реализации в ОВС задач обработки изображений [18,24,43,62,63,64,132]. Для псевдомедианной фильтрации требуется построение вариационного ряда, состоящего только из N элементов, что существенно сокращает аппаратные затраты. Алгоритм псевдомедианной фильтрации заключается в следующем. Вначале для каждой строки скользящего окна вычисляется значение медианы. Полученные N значений медианы используются для вычисления “новой” медианы.

Спецпроцессоры для нахождения максимального и минимальных значений их двух чисел разработаны для 8-, 16-разрядных операндов и операндов произвольной разрядности. Спецпроцессоры можно использовать как для целых чисел, так и для чисел с фиксированной запятой. В спецпроцессор подаются два операнда A и B и константа $K1$, которая имеет вид 00000001 и поступает на первом такте. На рис. 2.22 приведена структурная схема спецпроцессора операции $\max(A,B)$ для 8-разрядных операндов.

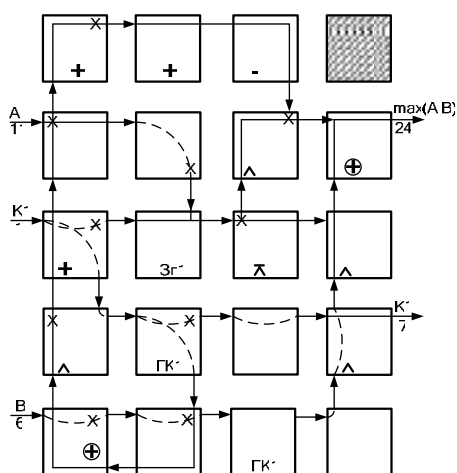


Рис. 2.22. Спецпроцессор операции $\max(A,B)$

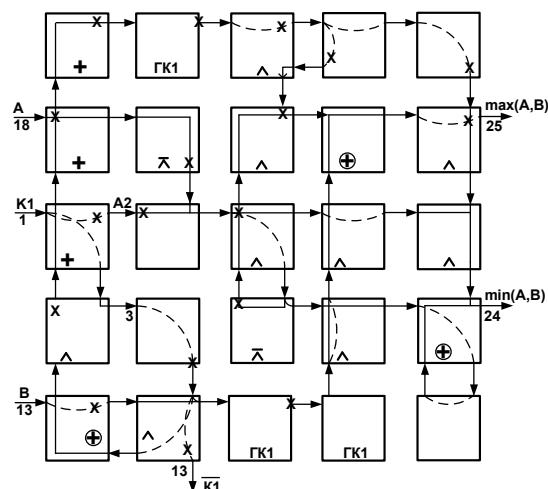


Рис. 2.23. Спецпроцессор операции $\max(A,B)$ и $\min(A,B)$

На рис. 2.23 приведен СП операции упорядочения числа для 16-разрядных операндов. Алгоритм нахождения минимального числа из двух чисел (А и В) незначительно отличается от алгоритма нахождения большего из двух чисел.

Рассмотрим вычислительные возможности поля однородной вычислительной среды, содержащего матрицу 24x30 вычислительных ячеек на примере реализации простого порогового метода подавления шума на изображении [83].

Суть метода состоит в следующем. Изображение обрабатывается скользящим квадратным окном размером $N \times N$ элементов. Окно продвигается с шагом в один элемент изображения слева направо и сверху вниз. На каждом шаге определяется средняя яркость группы элементов окна в окрестности центрального элемента.

Пусть для простоты организации вычислений входной сигнал представляет собой поток элементов изображения, поступающих в последовательном коде младшим разрядами вперед. Яркость каждого элемента задаётся 16 двоичными разрядами. Размер кадра изображения 256 x 256 элементов. Темп поступления информации равен тактовой частоте вычислительного поля.

Организация оконных методов обработки предусматривает выравнивание определенного числа строк входного кадра. Для этого необходимо перед подачей входной информации на однородную вычислительную среду пропустить её через однородную запоминающую среду. Поле однородной среды настраивается на распараллеливание потоков информации со сдвигом относительно друг друга на строку, что равно 4095 тактов.

В поле запоминающей среды размером 8 x 10 ячеек можно получить до 10 выходных потоков с временным сдвигом между соседними потоками, равным 4096 тактов. Эти потоки являются входной информацией для вычислительной среды. Однородная среда имеет размерность 24 x 30 вычислительных ячеек и позволяет разместить программу, реализующую

алгоритм обработки изображений. Размер "скользящего" окна при этом может принимать одно из четырех значений: 3x3, 5x5, 7x7 или 9x9 элементов. В поле однородной среды размещена также программа определения среднего значения яркости центрального элемента X_u в пределах заданного окна и программа выделения контура изображения.

2.8.2. Реализация цифровых фильтров в поле ОВС

Благодаря максимально возможному распараллеливанию по разрядам спецпроцессоры на однородных вычислительных средах имеют потенциально максимальное быстродействие и обеспечивают решение разнообразных практических задач цифровой обработки сигналов. В этом параграфе рассмотрены примеры реализации на ОВС устройств цифровой разностной фильтрации [84,89,91,117]. Построение фильтров осуществляется с помощью спецпроцессоров, которые выполняют базовые операции и обеспечивают обработку входных сигналов с тактовой частотой поля.

Основой этих структур являются предложенные в [90] алгоритмы. Рассмотрены также фильтры с перенастройкой, которые обеспечивают при заданной тактовой частоте поля необходимое быстродействие. Аппаратные затраты процессоров фильтров в первом приближении пропорциональны числу одновременно использованных процессоров базовых операций, длине импульсной характеристики фильтра и разрядности обрабатываемых данных.

Для фильтров аппаратные затраты и темп обработки зависят от степени распараллеливания и длины импульсной характеристики. В спецпроцессоре формируются значения приращений, которые суммируются с постоянной величиной и объединяются для получения второй разности выходного сигнала. Значения второй разности в двух выходных накопителях формируют последовательность значений первой разности и выходного сигнала фильтра. Это осуществляется при помощи накопителей последовательного кода с большей разрядностью.

В данном параграфе используются следующие форматы: ИКМ (импульсно-кодовая модуляция), ЛДМ (линейная дельта-модуляция), МДМ (многоуровневая дельта-модуляция). Например, формат ЛДМ-ИКМ обозначает, что входными данными являются два сигнала, один из которых представлен с помощью линейной дельта-модуляции, а второй - импульсно-кодовой модуляции.

Накопитель параллельного входного потока

Наиболее распространенной математической операцией в алгоритмах цифровой обработки сигналов является суммирование с накоплением. При цифровой фильтрации эти операции необходимы для вычисления свертки входного сигнала с импульсной характеристикой фильтра.

Низкая разрядность дельта-модуляции предопределяет упрощение операций умножения, приводя их в случае линейной дельта-модуляции к логическим операциям ИЛИ-НЕ, в результате чего накапливание с дельта-модуляцией становится одной из основных операций. Накопитель, который реализует указанную операцию, фактически является цифровым интегратором и имеет самостоятельное назначение.

При реализации накопителей на ОВС основная сложность при использовании низкоразрядных дельта-модулированных сигналов заключается в согласовании темпов поступления входных данных с темпами формирования многоразрядных результатов на выходе. Для согласования одnorазрядного сигнала с L – разрядной суммой классическая схема предусматривает размножение входного одnorазрядного потока на L прореженных потоков, каждый из которых поступает на свой накопитель. Выходные данные всех накопителей суммируются для получения конечного результата. Такой принцип построения требует больших затрат и допустим при низкой частоте входного сигнала. При высоком темпе поступления входного одnorазрядного сигнала накопления целесообразнее реализовать следующим образом. Поступающие данные разделяются на входе на два потока, один из которых формируется из четных отсчетов сигнала, а второй – из нечетных.

Последовательность действий в модуле формирования приращений следующая:

- формирование произведения: $P_i = \overline{B_i^{(x)}} \oplus B_m^{(h)}$;

- прореживание потоков $P_{1,i}$ и $P_{2,i}$ при помощи двухразрядной константы $K_1(2)$:

$$P_{1,i}^{(pr)} = P_{1,i} \& K_1(2), P_{2,i}^{(pr)} = P_{2,i} \& K_1(2);$$

- обнуление частичных сумм накопителей инверсной константой $\overline{K_2(4)}$:

$$S_1^{(2)} = S_1^{(2)} \& \overline{K_2(4)}, S_2^{(2)} = S_2^{(2)} \& \overline{K_2(4)};$$

- суммирование соответствующих произведений:

$$S_1^{(2)} = S_1^{(2)} + P_{1,i}^{(pr)}, S_2^{(2)} = S_2^{(2)} + P_{2,i}^{(pr)};$$

- формирование выходных сумм:

$$S_1^{(4)} = S_1^{(2)} \& K_2(4), S_2^{(4)} = S_2^{(2)} \& K_2(4).$$

На рис. 2.24 приведена структурная схема модуля формирования приращений.

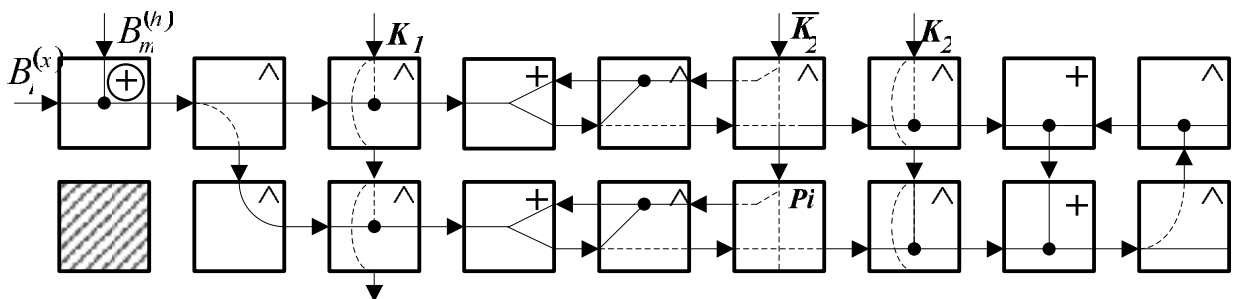


Рис. 2.24. Схема спецпроцессора операции формирования приращений

Спецпроцессор размножения потоков с удвоением разрядности

Спецпроцессор используется для построения цифровых фильтров дельта-модуляции с высокой разрешающей способностью. В основе алгоритма - распараллеливание одного 16-разрядного входного сигнала на два 32-разрядных и суммирование их с соответствующими входными сигналами. Сигналы даны в дополнительном коде.

Работа спецпроцессора заключается в раздвоении входного 16-разрядного

потока $\delta y_k^{(16,d)}$. Прореживание каждого из полученных потоков P_1 и P_2 осуществляется константой $K_{16}(32)$. Поток P_2 до прореживания задерживается на 16 разрядов. После прореживания получаем 32-разрядные потоки, которые суммируются, соответственно, с потоками B и C той же разрядности. Константа $K_1(32)$ при этом корректирует буферный разряд результата. Структурная схема спецпроцессора размножения потоков показана на рис. 2.25.

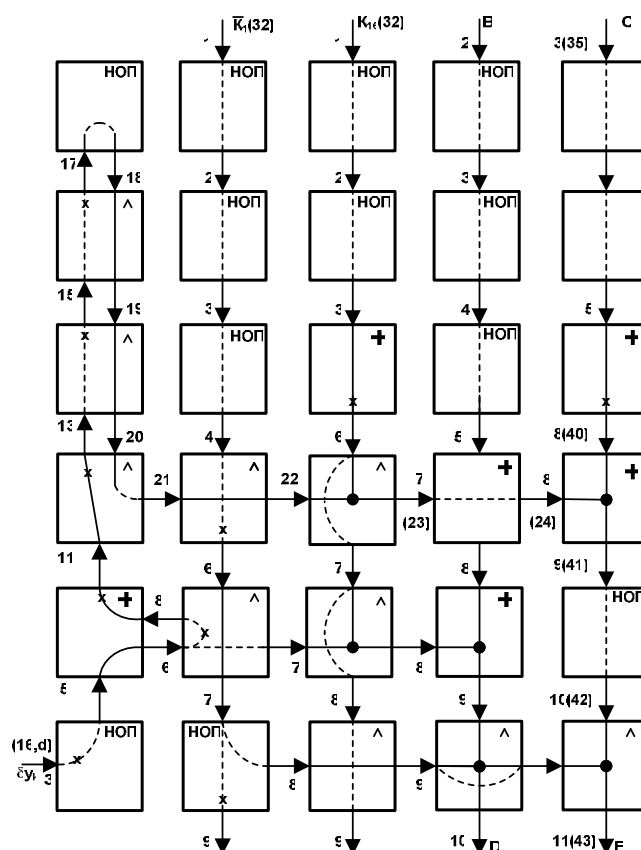


Рис. 2.25. Схема спецпроцессора операции размножения потоков

Спецпроцессор умножения сигналов в формате ЛДМ-ЛДМ

Разрядность операндов входных сигналов равна 1, разрядность выходных сигналов – r . Перемножение выполняется с тактовой частотой ОВС. Структурная схема спецпроцессора показана на рис. 2.26.

В основе алгоритма умножения ЛДМ-сигналов используется суммирование по модулю 2 сигналов $\{B_i^{(x)} \oplus B_i^{(y)}\}$, затем последовательность этой суммы ветвится на r выходов.

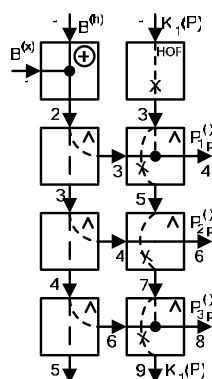


Рис.2.26. Схема спецпроцессора операции умножения сигналов в формате ЛДМ-ЛДМ

Спецпроцессор умножения сигналов в формате ЛДМ-ИКМ

Спецпроцессор этой операции разработан для сигналов, один из которых задан последовательностью битов в формате ЛДМ, а второй – в формате ИКМ или МДМ в дополнительном коде. Разрядность кодировки операндов и результатов одинаковая и равна r . Изменение разрядности результата осуществляется путем изменения разрядности ИКМ-сигнала и длины управляющей константы $K_1(r)$. Структурная схема спецпроцессора показана на рис. 2.27.

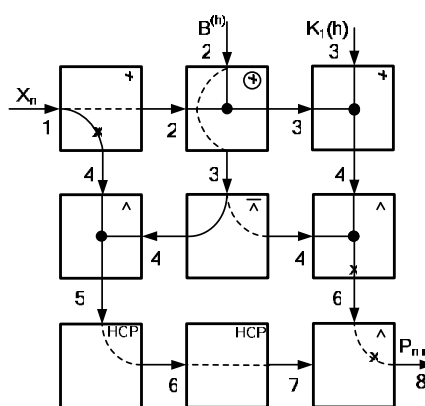


Рис.2.27. Схема спецпроцессора операции умножения сигналов в формате ЛДМ-ИКМ

В спецпроцессор направляются сигналы ИКМ $\{x_n\}$, ЛДМ $\{B_i^{(h)}\}$ и константа $K_1(r)$; $\{x_n\}$ поступает на 1-м такте, $\{B_i^{(h)}\}$ – со 2-го такта, константа

$K_1(r)$ – с 3-го такта, результат умножения $\{P_{n,r}^{(i)}\}$ формируется в дополнительном коде, начиная с 8-го такта, по формуле

$$P_{n,r}^{(i)} = \sum_{k=0}^{r-1} B_n^{(p)k} 2^{r-k-1} + B_i^{(h)},$$

где

$$B_n^{(p)k} = B_n^{(x)k} \oplus \overline{B_i^{(h)}}.$$

Последовательность действий в модуле PLI следующая:

- суммирование по модулю два разрядов ИКМ-сигнала с ЛДМ-кодом:

$$P_k^{(1)} = B_n^{(x)k} \oplus B_i^{(h)};$$

- суммирование значения $P_k^{(1)}$ с константой $K_1(r)$, то есть получение результата умножения $P_{n,r}^{(i)}$ в дополнительном коде;

- логическое умножение всех битов входного сигнала x_n с ЛДМ-кодом $P_k^{(2)} = B_n^{(x)k} \& B_i^{(h)}$, т.е. получение результата умножения в дополнительном коде;

- логическое умножение результата операции 2 с инвертированным ЛДМ-кодом $\overline{B_i^{(h)}}$;

- объединение результатов 3 и 4 по ИЛИ, т.е. получение значения $P_{n,r}^{(i)}$.

Спецпроцессор операции умножения сигналов в формате ИКМ-МДМ

Спецпроцессор этой операции разработан для сигналов, один из которых представлен в формате ИКМ дополнительным кодом с разрядностью r . Ярусное построение спецпроцессора с числом ярусов $/2$ позволило простым изменением константы $K_1(r)$ и увеличением числа ярусов достичь нужной разрешающей способности цифрового фильтра в формате ИКМ-МДМ. В алгоритме умножения сигналов в формате ИКМ-МДМ используется операция логического умножения всех битов одного сигнала на один из битов второго и

суммирование с необходимым сдвигом результатов этих произведений для всех битов другого сигнала

В спецпроцессор подаются сигналы $\{x_n\}$ и $\{s_m^{(h)}\}$ с первого и четвертого тактов и константа $K_1(r)$ – со второго такта. Результат произведения формируется с $(r+5)$ -го такта. Структурная схема спецпроцессора операции умножения сигналов в формате ИКМ-МДМ для r -разрядных данных показана на рис. 2.28.

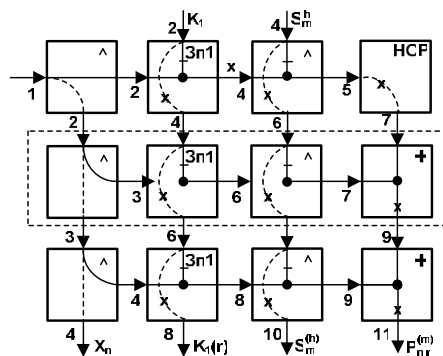


Рис.2.28. Схема спецпроцессора операции умножения сигналов в формате ИКМ-МДМ

Алгоритм операции умножения сигналов в формате ИКМ-МДМ следующий:

- за единичным значением в константе $K_1(r)$ осуществляется запоминание k -го разряда сигнала $\{x_n\}$;

- осуществляется логическое умножение сигналов $B_n^{(x)k}$ и $s_m^{(h)}$

$$P_n^{(x)k} = B_m^{(h)i} \& B_n^{(x)k}, i = \overline{1, r};$$

- операции 1 и 2 повторяются для всех r разрядов x_n ;

- суммируются значения логических произведений операции 2 со сдвигом в такт.

Спецпроцессор базовой операции свертки в формате ЛДМ-ЛДМ

Спецпроцессор этой операции разработан для сигналов, представленных последовательностями битов в ЛДМ-формате. Разрядность операндов входных

сигналов $\{B_i^{(x)}\}$ и $\{B_i^{(y)}\}$ равна единице, разрядность частичной суммы равна r . Поскольку входные сигналы поступают с тактовой частотой ОВС, спецпроцессор позволяет получить максимальное быстродействие процессоров ОВС-фильтров, которые обрабатывают сигналы в формате ЛДМ-ЛДМ.

Основой алгоритма свертки является суммирование по модулю два ЛДМ-кодов входных сигналов. Затем осуществляется распараллеливание результата умножения на r потоков и суммирование их с соответствующими r -разрядными входными потоками частичных сумм.

В спецпроцессор поступают на первом такте сигналы $\{B_i^{(x)}\}$ и $\{B_i^{(y)}\}$, на втором такте – константа $K_1(r)$. Результаты суммирования $\{S_{k,r}^{(i+1)}\}$ снимаются на $(3k+3)$ -м такте, где k – номер входного потока. Структурная схема спецпроцессора показана на рис. 2.29.

Спецпроцессор операции свертки сигналов в формате ЛДМ-ИКМ

Спецпроцессор разработан для сигналов, один из которых приходит в формате ЛДМ, а второй – в ИКМ. Разрядность всех сигналов равна r . Применение дополнительного кода для входного сигнала в формате ИКМ и частичной суммы позволило сократить размер поля ОВС при реализации модуля.

В основе алгоритма операции «свертка сигналов в формате ЛДМ-ИКМ» используется операция умножения сигналов в формате ИКМ на соответствующее ЛДМ-представление второго сигнала и суммирование результата умножения с частичной суммой. Последовательность действий в спецпроцессоре (рис. 2.30а):

- суммирование по модулю двух битов ИКМ-сигнала с ЛДМ- кодом:

$$P_k^{(1)} = B_n^{(x)k} \oplus B_m^{(h)};$$

- суммирование полученного результата с константой $K_1(r)$, то есть получение результата умножения ЛДМ-ИКМ в дополнительном коде;

- логическое умножение результата предыдущей операции с инвертированным ЛДМ-кодом $\overline{B_m^{(h)}}$;

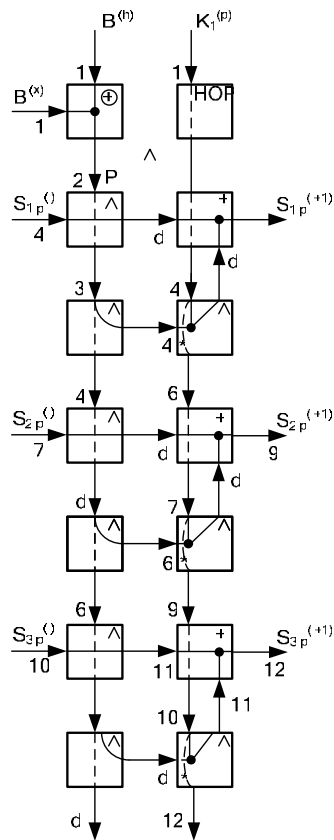


Рис. 2.29. Схема специализированного процессора операции свертки сигналов в формате ЛДМ-ЛДМ

- логическое умножение сигналов с

ЛДМ-кодом $B_m^{(h)}$;

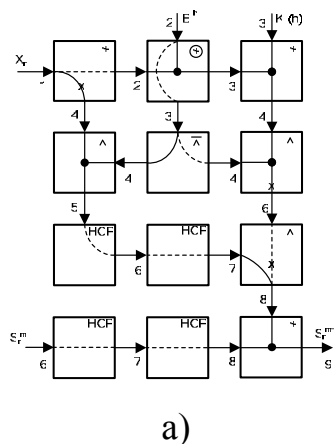
- произведение сигналов в формате ЛДМ-ИКМ в дополнительном коде;

- алгебраическое суммирование результата умножения ЛДМ-ИКМ с частичной суммой $S_{n,r}^{(m)}$. Для

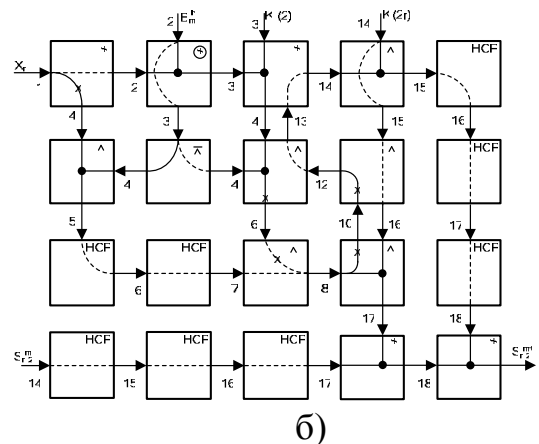
специализированного процессора операции свертки сигналов в формате ЛДМ-ЛДМ, представленного на рис.2.30б, результат алгебраического суммирования задерживается на r тактов. Затем полученный результат и текущий алгебраически складывается с частичной $2r$ -разрядной суммой $S_{n,2r}^{(m)}$. Поэтому за период 2τ в модуле осуществляются два умно-

ножения сигналов в формате ЛДМ-ИКМ и суммирование их с входным потоком.

На рис. 2.30а и 2.30б приведены структурные схемы специализированного процессора операций свертки сигналов в формате ЛДМ-ИКМ.



а)



б)

Рис.2.30. Схемы специализированных процессоров операций свертки сигналов в формате ЛДМ-ИКМ

Спецпроцессор операции свертки сигналов в формате МДМ-МДМ в прямом коде

Спецпроцессор этой операции разработан для сигналов, представленных форматом МДМ, и может применяться для формата ИКМ-ИКМ. Структурная схема спецпроцессора, который позволяет обрабатывать сигналы в прямом коде, показана на рис. 2.31.

На входы спецпроцессора поступают значения неполной свертки $S_{n,16}^{(m-1)}$ в дополнительном коде, начиная с 5-го такта; сигнал $\{s_m^{(h)}\}$ – с первого такта; сигнал $\{s_n^{(x)}\}$ и константа $K_1(16)$ – со второго такта, константа $e_m^{(h)}$, которая отвечает знаковому разряду $s_m^{(h)}$, – с третьего такта. Результат накопления $S_{n,16}^{(m)}$ формируется в дополнительном коде, начиная с 13-го такта. Константа $e_m^{(h)}$ имеет вид $K_0(16) = 00...0$ для положительного значения $s_m^{(h)}$ и $K_{16}(16) = 11...1$ – для отрицательного.

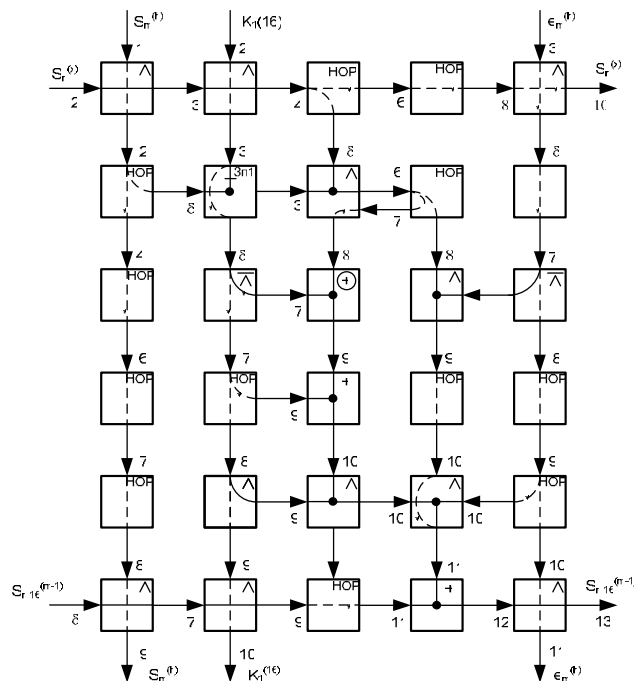


Рис. 2.31. Структурная схема спецпроцессора операции свертки сигналов в формате МДМ-МДМ в прямом коде

В спецпроцессоре операции свертки сигналов выполняется следующая последовательность операций:

- по единичному значению константы $K_1(16)$ запоминание k -го разряда сигнала;

- логическое умножение сигналов $B_m^{(h)k}$ и $s_n^{(x)}$. Результат этой операции равен значению входного сигнала $s_n^{(x)}$, если $B_m^{(h)k} = 1$, и нулю, если $P_n^{(d)} = B_m^{(h)k} \& B_{n-m}^{(x)}$;

- инвертирование константы $K_1(16)$, т.е. формирование константы вида $\overline{K_1(16)}$;

- суммирование по модулю 2 результата операции 2 и константы $\overline{K_1(16)}$:

$$D_n^{(d)} = P_n^{(d)} \oplus \overline{K_1(16)};$$

- суммирование значения $D_n^{(d)}$ со значением $K_1(16)$. В итоге формируется значение входного сигнала $s_n^{(x)}$, умноженное на значение $B_m^{(h)k}$ в случае, если $s_m^{(h)} < 0$;

- логическое умножение последовательности $\{P_n^{(d)}\}$ и константы $\overline{B_m^{(h)k}}$. Результат $s_n^{(x)}$ умножается на $B_m^{(h)k}$, если $s_m^{(h)} < 0$;

- логическое умножение значения операции 5 на константу $e_m^{(h)}$;

- объединение результатов операций 6 и 7 по ИЛИ, формирование результата

$$P_{n,r} = e_n^{(x)} e_m^{(h)} |s_n^{(x)}|;$$

- суммирование значения $P_{n,r}$ с неполной сверткой, формирование сигнала

$$S_{n,16}^{(m)} = S_{n,16}^{(m-1)} + P_{n,r}.$$

2.9. Аппаратная реализация мультимикроконвейерных структур на ОВС

Однородность и простота ячеек ОВС открывают широкие перспективы для ее реализации в микроэлектронном исполнении. Одним из первых шагов в этом направлении была организация в 80-х годах прошлого века серийного

выпуска СБИС ОВС Н1841В61 «Райта-5» на предприятии «Альфа» (г. Рига) [12,13].

Структурная схема микросхемы Н1841В61 показана на рис. 2.32.

СБИС ОВС представляли собой матрицу 5х4 вычислительных ячеек. Входы и выходы периферийных вычислительных ячеек матрицы через согласователи ТТЛ-уровней соединены с входами микросхемы. Микросхема имеет 42 вывода, среди которых:

- шина питания и корпусная шина «Общий»,
- вход управления вводом программы,
- вход тактовых импульсов,
- вход и выход программирования,
- 18 информационных входов и 18 информационных выходов.

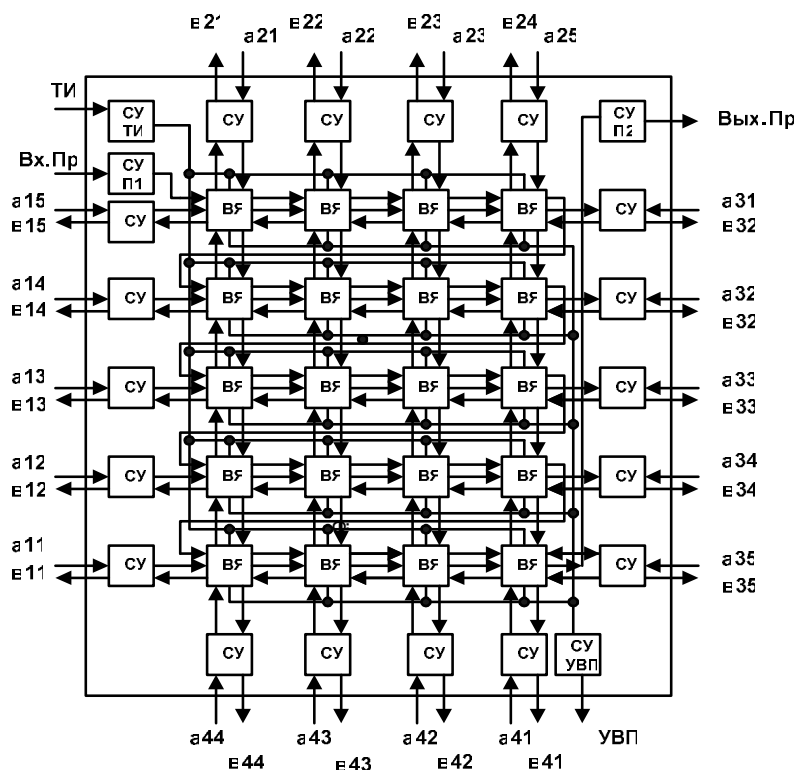


Рис. 2.32. Структурная схема СБИС Н1841 ВФ1

На основе СБИС ОВС Н1841В61 изготавливались платы вычислительной среды. На плате устанавливались 132 микросхемы, реализующие поле размерностью 30х88 вычислительных ячеек. Плата содержала по 30 информационных входов и выходов с левой и правой сторон, а также по 88 информационных входов и выходов по её верхней и нижней сторонам, что в

общей сложности составляло 118 информационных входов и 118 информационных выходов. Для осуществления соединений с другими платами с целью наращивания размеров поля ОВС на плате устанавливались по её узкой стороне 135-контактный разъем и по обеим широким сторонам два 90-контактных разъема. Выбранная конструкция платы позволяла наращивать размеры поля по двум осям координат [126].

На рис. 2.33 показаны платы ОВС, реализованные на СБИС Н1841 ВФ1.



Рис. 2.33. Платы однородной вычислительной среды

Для осуществления соединений с другими платами с целью наращивания размеров поля ОВС на плате устанавливались 135-контактный разъём на узкой стороне и по два 96-контактных разъёма на обеих широких сторонах платы. Выбранная конструкция платы позволяет создавать как ленточные структуры поля ОВС, так и наращивать размеры поля по двум координатам (табл. 2.4)

Таблица 2.4

Конфигурация поля	Число ячеек поля
1x16	30x1408
2x8	60x704
4x4	120x352
8x2	240x176
16x1	480x88

Если количество плат $n = 16$, то возможны различные конфигурации (рис. 2.34). Для оптимального отображения прикладной задачи на поле ОВС могут потребоваться и конфигурации с неравномерной шириной поля.

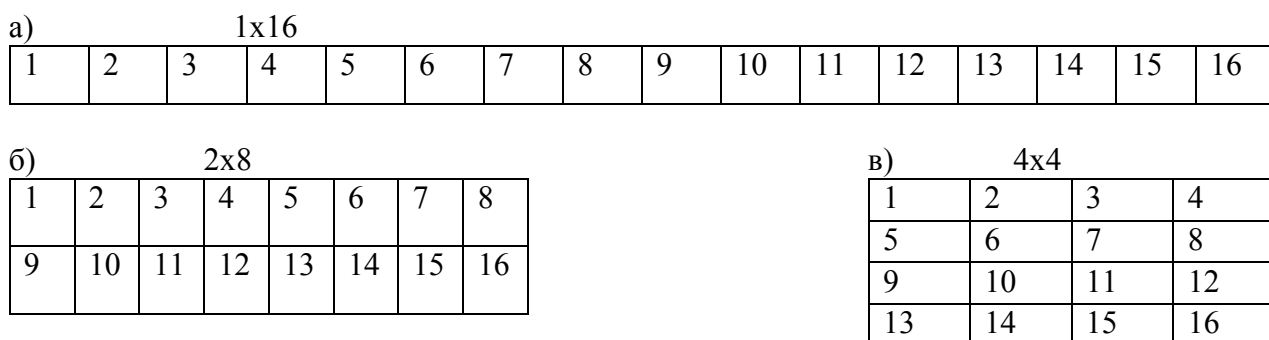


Рис. 2.34. Конфигурации поля ОВС

Для обеспечения возможных конфигураций при количестве плат $n = 16$ требуется сложный и громоздкий электронный коммутатор. Однако эта задача может быть решена и очень простыми средствами. Достаточно кроме основного 135-контактного разъема, используемого для установки плат в кассету, установить на кроссплате рядом с каждой платой ОВС два дополнительных разъема, один из которых используется для входов и выходов левой стороны поля, а другой, соответственно, для входов и выходов правой стороны поля. Соединяя с помощью кабелей соответствующие дополнительные разъемы, а также соединяя плоскими кабелями верхние и нижние разъемы, можно образовывать произвольные конфигурации. На рис. 2.35 иллюстрируется образование различных конфигураций поля ОВС.

Конфигурация изображения на рис. 2.35а представляет собой ленточную структуру поля ОВС, образованную 16-ю платами с попарно соединенными с помощью кабелей дополнительными разъемами плат ОВС. Разъем, соответствующий левой стороне крайней левой на рисунке платы, и разъем, соответствующий правой стороне крайней правой платы, подключаются к интерфейсной плате для организации обмена с ЭВМ.

Конфигурация 2x8 плат ОВС, представленная на рис. 2.35б, образуется при попарном объединении нижних и верхних разъемов соседних плат и соединений разъема правой стороны 1-й платы с разъемом левой стороны 3-й платы, соединение разъема правой стороны 2-й платы с разъемом левой

стороны 4-й платы и т.д. Выполняя необходимые соединения, можно образовать поле ОВС требуемой конфигурации (2.35в, г, д, е).

Выбранная конструкция платы и стенда позволяет организовать итерационную обработку данных в поле ОВС [126]. Структура для итерационной обработки данных представлена на рис. 2.36.

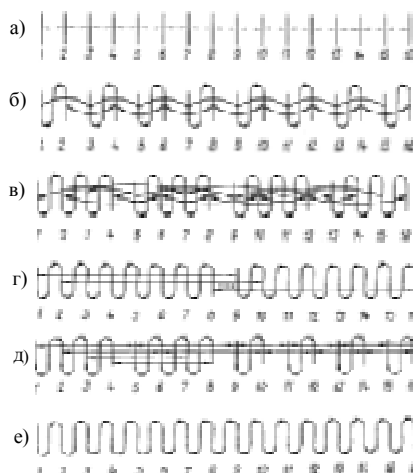


Рис. 2.35. Образование поля ОВС различной конфигурации



Рис. 2.36. Структура итерационной обработки

Организация обработки данных в трёхмерных структурах в принципе не отличается от обработки данных в двумерных структурах. В трёхмерных структурах, как и в двумерных, есть возможность организовать итерационную обработку данных [126].

При отображении решаемой задачи в реальное поле ОВС часто возникает необходимость в передаче информации на вычислительные ячейки некоторой платы, не используя для этой цели каналы транзита ячеек. Если настраивать в поле ОВС длинные регистровые каналы для передачи данных, то значительное число вычислительных ячеек используются неэффективно. Чтобы избежать этого, по периферии платы устанавливаются дополнительные мультиплексоры, которые позволяют обеспечивать передачу информации, минуя вычислительные ячейки. Кроме того, мультиплексоры выполняют функцию усилителей. Для настройки мультиплексоров необходимо ввести для каждого

мультиплексора отдельный триггер в дополнительном регистре, входящем в состав интерфейса.

СБИС ОВС второго поколения была разработана в 90-е годы в НИИ ТТ (г. Зеленоград) под руководством В.С. Седова [108,109]. Вычислительная матрица имеет резервные ячейки, что повышает дефектоустойчивость СБИС при их изготовлении и отказоустойчивость при их эксплуатации. Введение дополнительной логики позволяет провести тестирование всех ячеек, выявить дефектные ячейки и провести подключение годных ячеек, организовав матрицу заданной размерности. Вычислительная ячейка СБИС ОВС второго поколения представляет собой одноразрядный тактируемый вычислительный модуль, содержащий свой регистр команд. Система команд ячейки соответствует системе команд вычислительной ячейки, используемой в микросхемах Н1841В61. Ячейки объединяются в матрицу, как показано на рис. 2.37.

Пользователю предоставляется для работы однородная вычислительная среда, образующая матрицу вычислительных ячеек размерностью 10x10. В СБИС имеется десять линий для горизонтальной загрузки программы, входы управления вводом программы и тактовых импульсов.

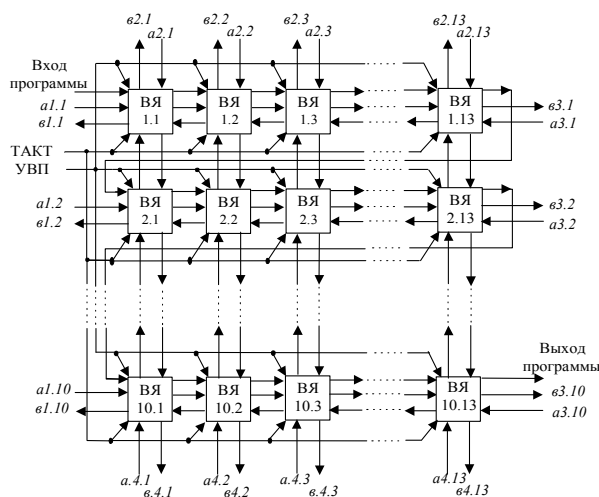


Рис. 2.37. Структурная схема СБИС ОВС второго поколения

Реализованная в СБИС матрица содержит 130 процессорных элементов с организацией 13x10 и дополнительной логикой, обеспечивающую отказо- и дефектоустойчивость. Схема СБИС ОВС, разработанная в НИИ ТТ, приведена на рис. 2.38.

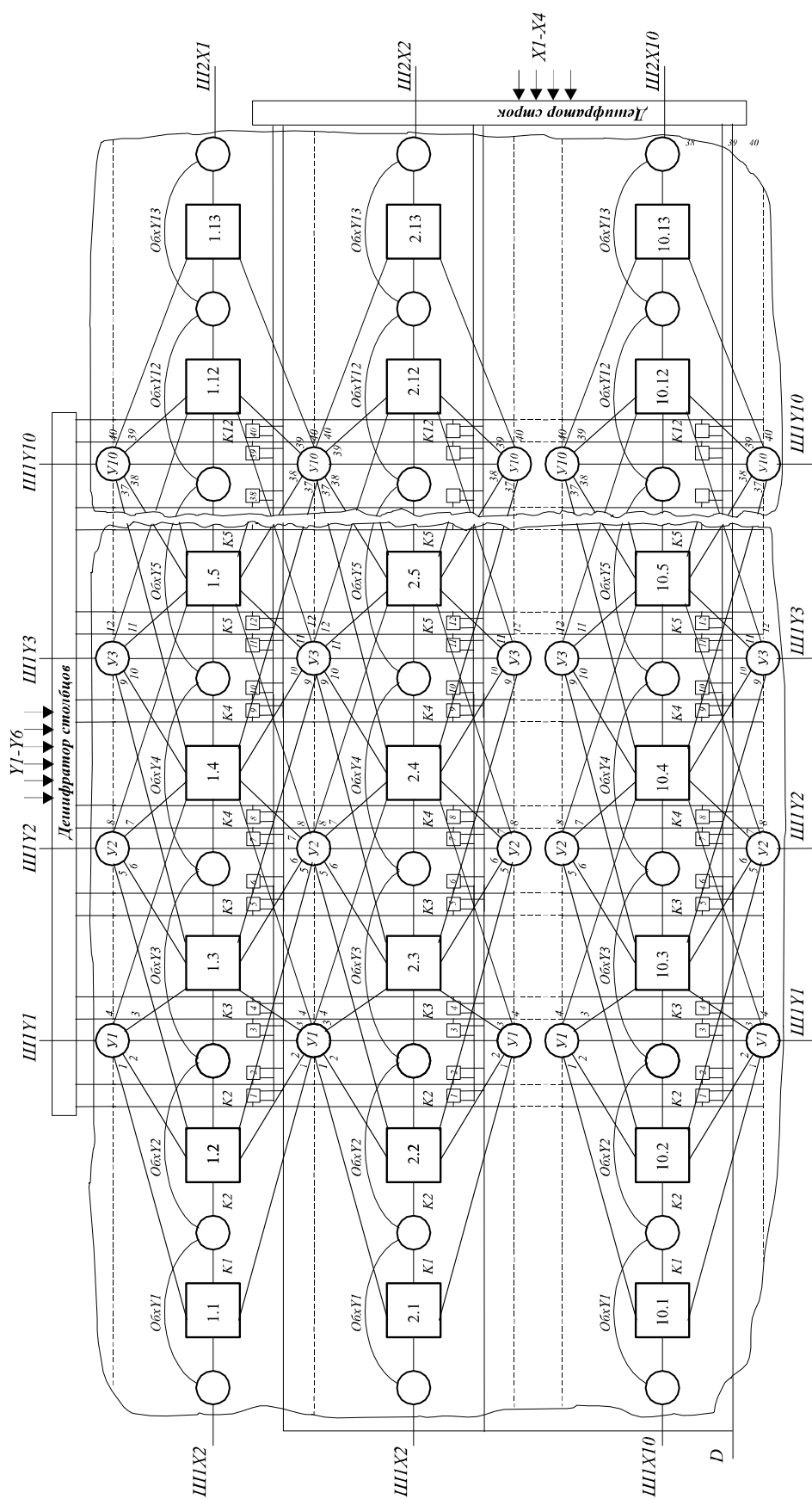


Рис. 2.38. Организация процессорных элементов в СБИС ОВС второго поколения

Горизонтальные обходы процессорных элементов, вместе с переключателями обеспечивают либо подключение i -го процессорного элемента к горизонтальной шине, либо обход ячейки. Логическими узлами У1-У10 каждой строки осуществляется подсоединение вычислительных ячеек, расположенных по вертикали.

Запоминающие элементы-триггеры обозначены на рис. 2.38 маленькими квадратами. Дешифраторы строк и столбцов служат для выбора запоминающего элемента. Шина сигнала обеспечивает запись "нуль" или "единица" в выбранный запоминающий элемент. В СБИС ОВС каждая строка имеет 52 управляющих триггера, что соответствует числу линий связи вычислительных ячеек с вертикальными каналами.

Установка и сброс любого управляющего триггера возможны при наличии соответствующих сигналов с выхода дешифратора столбцов и дешифратора строк. Перед тестированием СБИС ОВС все управляющие триггеры должны быть установлены в нулевое состояние. Установка управляющих триггеров в "0" осуществляет отключение процессорных элементов от горизонтальных и вертикальных каналов.

Рассмотрим работу дополнительной логики, обеспечивающей дефекто- и отказоустойчивость, причем достаточно рассмотреть одну строку. Если запоминающие элементы установлены в "0", то все вычислительные ячейки отключены, и сквозные связи устанавливаются по горизонтальным и вертикальным обходам.

Отключение вычислительной ячейки от логических связей осуществляется установкой соответствующего запоминающего элемента в "0". Такая структура организации матрицы позволяет осуществлять доступ извне к любой ячейке, что дает возможность протестировать все ячейки с логическими связями и иметь полное представление о работоспособности матрицы. В СБИС ОВС подключаются любые 10 годных ячеек в строке при работоспособности соответствующих логических связей, т.е. здесь нет деления на основные и резервные вычислительные ячейки.

После проверки каждой вычислительной ячейки в отдельности следует, если это возможно, сформировать в СБИС ОВС поле ячеек путем подключения годных и проверить их взаимодействие. Если формирование заданного поля невозможно, то СБИС считается негодной. Тестом проверки взаимодействия вычислительных ячеек между собой служит любая задача, где каждая ячейка рабочей матрицы обменивается данными с четырьмя соседними при условии возможности контроля такого обмена.

2.10. О перспективах развития однородных вычислительных сред

Опыт создания СБИС ОВС первых двух поколений показал, что ОВС наилучшим образом подходят для реализации не в отдельном чипе, а целиком на кремниевой пластине [56,64,66,103]. Использование пластин вместо кремниевых чипов даёт возможность не стремиться любой ценой к минимально возможным топологическим размерам. Очевидно, что чем меньше топологические размеры, тем сложнее добиться приемлемого процента выхода интегральной схемы, тем она дороже. Неслучайно около 90% выпускаемых в мире микросхем имеют топологические размеры элементов и межсоединений, укладывающиеся в нормы 1 мкм и более, хотя уже взят рубеж в 0,045 мкм.

Работы над однородными вычислительными средами, ориентированными на пластину, были начаты во второй половине восьмидесятых годов прошлого века [41,42,92]. Результаты изысканий и первые архитектурные решения нашли отражение в ряде публикаций [57,58,106,126]. Была заново пересмотрена концепция однородных сред. В результате ревизии функции ячейки ОВС было признано целесообразным расширить, включив в ячейку достаточно большую регистровую память. Увеличивались коммутационные возможности ячейки, что, прежде всего, вызывалось включением в ячейку канала памяти и служебного канала [65,129-131].

На рис. 2.39 показана матрица ОВС нового поколения [133-136], ячейки которой соединены между собой локальными шинами. Как видно из рисунка, каждая ячейка соединена с восемью соседними ячейками. Кроме того, ячейка

имеет восемь дополнительных входов и выходов. Эти входы-выходы имеют фиксированную коммутацию и показаны на рисунке тонкими линиями. Некоммутируемые связи используются в ОВС при реализации алгоритмов клеточных автоматов.

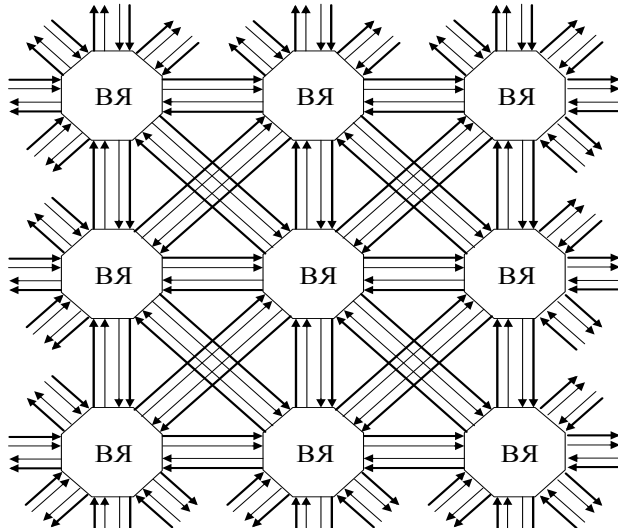


Рис. 2.39. Матрица ОВС

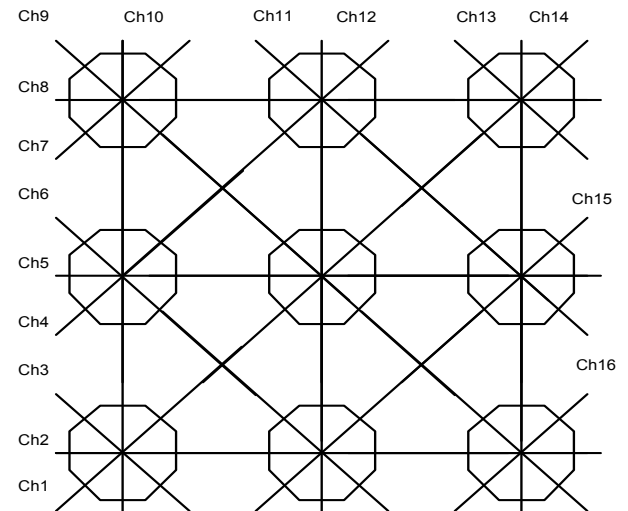


Рис. 2.40. Шины ОВС

Имеются также горизонтальные, вертикальные и диагональные шины, посредством которых осуществляется обмен информацией между удалёнными друг от друга ячейками (рис. 2.40).

На рис. 2.41 показана функциональная схема ячейки ОВС нового поколения, здесь

- 1- регистр инструкции;
- 2- регистр адреса;
- 3- регистр управления;
- 4- регистр сравнения;
- 5- блок ввода данных с входов $a1, \dots, a8$;
- 6- блок вывода данных на выходы $b1, \dots, b8$;
- 7- блок ввода данных с шин $Ch1, \dots, Ch4$;
- 8- блок вывода данных на шины $Ch1, \dots, Ch4$;
- 9- входные коммутаторы;

- 10- выходные коммутаторы;
- 11- арифметико-логическое устройство;
- 12- блок памяти;
- 13- блок перепрограммирования;
- 14- блок транзитов;
- 15- служебный регистр;
- 16- блок оптоэлектронных коммутаторов.

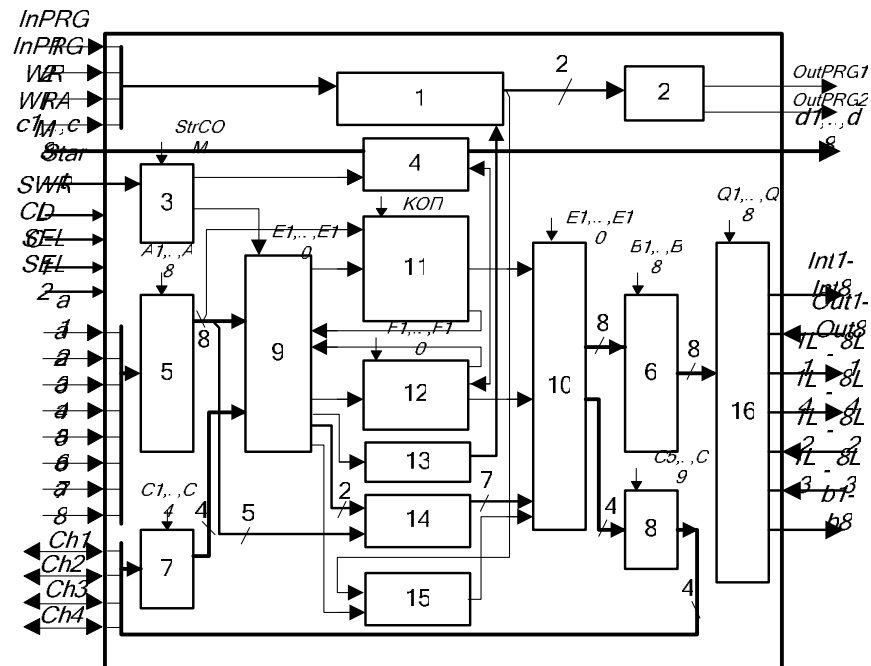


Рис. 2.41. Функциональная схема ячейки

Ячейка представляет собой однобитовый тактируемый модуль, выполняющий вычислительные функции, функции коммутации информационных потоков, памяти, диагностики и перепрограммирования. Ячейка имеет восемь информационных входов ($a1, \dots, a8$) и выходов ($b1, \dots, b8$), входы оптических преобразователей $Int1-Int8$, выходы оптических преобразователей $Out1-Out8$, входы с локальных шин $1L2-8L2$, $1L3-8L3$, выходы на локальные шины $1L1-8L1$, $1L4-8L4$, фиксированные входы и выходы ($c1, \dots, c8$ и $d1, \dots, d8$), входы записи в регистры инструкции (WRI), данных (SWRD) и памяти (WRRAM), две входные ($InPRG1$, $InPRG2$) и две выходные

(*OutPRG1*, *OutPRG2*) шины настройки, двунаправленные шины ввода-вывода данных (*Ch1*, ..., *Ch4*), входы тактовых импульсов (*CLC*) и сброса (*RESET*).

Блок регистра инструкции представляет собой два параллельных регистра длиной в 154 бит каждый. Код программы последовательно заносится в первый из регистров, после чего осуществляется перезапись данных из первого регистра во второй. В дальнейшем второй регистр блокируется для последующей записи и в соответствии с занесенной информацией осуществляет настройку всех узлов ячейки на выполнение заданной программы. Блок адреса представляет собой регистр и дешифратор адреса. В регистр адреса с выхода регистра инструкции заносится значение адреса, присвоенное данной ячейке. Блок управления обеспечивает организацию программирования и настройки ячейки, управляет работой клеточного автомата, обслуживает процесс идентификации номера ячейки и синхронизирует работу служебного канала. В регистре сравнения осуществляется сравнение значения состояния соседних клеток с заданным, которое предварительно записано в блоке памяти ячейки.

Блок ввода данных состоит из восьми входных коммутаторов, то есть имеющих восемь входов и один выход. Выходы блока ввода данных формируют в ячейке восемь независимых информационных каналов. Блок ввода и вывода данных с шин *Ch1*, ..., *Ch4* имеет четыре коммутатора входа типа 4x1 и четыре коммутатора выхода типа 1x4 и связывает ячейку с двунаправленными шинами быстрого доступа *Ch1*, ..., *Ch4*.

Арифметико-логическое устройство (АЛУ) включает однобитовое двухоперандное устройство, в котором реализуются операции над данными, поступающими в последовательном коде. Система команд была приведена в параграфе 2.1. Блок памяти реализован на секционированном регистре длиной в 128 Кбит. Логически он может быть разделён на блоки памяти для хранения констант и оперативных данных. Константы предварительно заносятся в блок памяти в режиме программирования. Данные могут поступать с других ячеек в

режиме работы. Блок программирования обеспечен регистром данных для хранения микропрограммы длиной в 8Кбит.

Служебный канал содержит регистры данных и адреса, которые служат для осуществления тестирования ячейки в режиме “работа”. Блок оптоэлектронных коммутаторов включает восемь коммутаторов, каждый из которых имеет связь с входом соседней ячейки, с входами и выходами оптических преобразователей, с двумя входами и выходами локальных шин.

Работа с ячейкой предполагает режимы: "программирование и настройка", “настройка памяти”, "работа клеточного автомата", "работа" и "перепрограммирование".

Вычислительная ячейка осуществляет обработку информационных потоков, идущих с восьми направлений. Поэтому внутри неё можно выделить восемь информационных трасс, которые, вообще говоря, имеют разное функциональное назначение. Из этого числа информационных трасс ячейки по две предназначены для каналов арифметико-логической обработки данных и памяти, одна трасса - для служебного канала, три оставшиеся трассы отведены для канала транзитов. Благодаря аппаратным решениям ячейка может либо сочетать в себе функции обработки, коммутации и памяти, либо осуществлять передачу информационных потоков данных.

Для целей диагностики и перестройки участка поля среды в ячейке был введен служебный канал, с помощью которого в ячейках, функционирующих в режиме “работа”, бронируется трасса, по которой можно выводить из ячейки информацию о её состоянии и вводить данные для перепрограммирования её регистра инструкции. Такого канала ячейки однородных вычислительных сред первых поколений не имели.

Составными частями канала регистров являются регистры инструкции, адреса и последовательный регистр памяти, которые в режиме настройки ячейки программируются. В дальнейшем регистр инструкции настраивает ячейку на выполнение вычислительных и коммутационных функций. Данные, занесенные в память, могут использоваться АЛУ ячейки для обработки

информации и быть переданы в АЛУ других ячеек. Оставшаяся часть памяти по умолчанию используется в ячейке для записи, хранения и чтения данных в качестве оперативной памяти.

Регистр инструкции состоит из регистра кода операций, регистра управления коммутаторами, регистров управления памятью, вводом - выводом данных канала памяти, вводом - выводом данных канала АЛУ, шинными каналами и регистров управления входами и выходами ячейки. Для каждого из вышеуказанных регистров выделена своя группа разрядов регистра инструкции.

Как уже было сказано выше, ОВС нового поколения ориентированы на реализацию на кремниевых пластинах. Связь между пластинами обеспечивается с помощью оптоэлектронных коммутаторов, входящих в состав ячейки ОВС (рис.2.42).

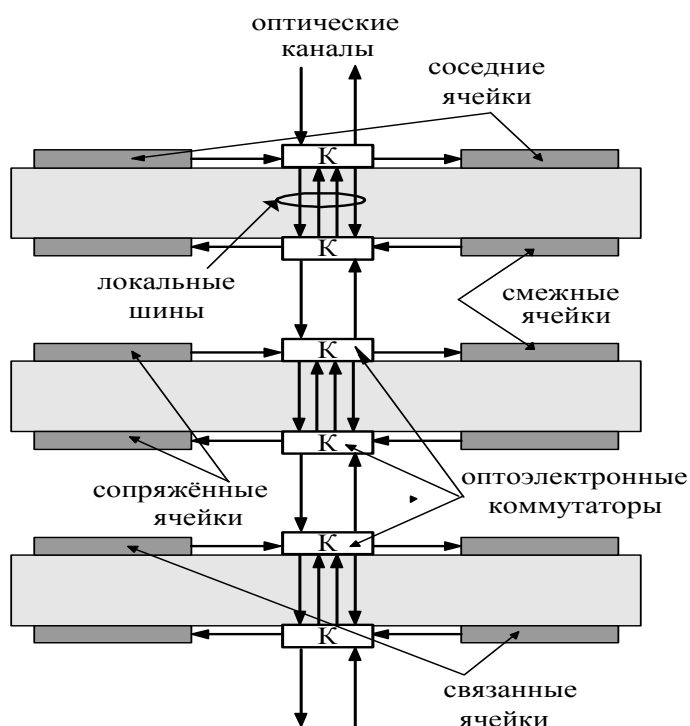


Рис. 2.42. Оптическая коммутация ячеек ОВС соседних пластин

Оптические каналы обеспечивают передачу информации между ячейками соседних пластин. Каждый из оптических каналов состоит из пары преобразователей - электронно-оптического и оптоэлектронного. Назовём

“сопряжёнными” ячейки, расположенные на различных поверхностях пластины и находящиеся одна над другой. Под “смежными” ячейками будем понимать ячейки, которые находятся на соседних пластинах и также размещены друг над другом. Смысл “соседних” и “связанных” ячеек раскрывается из рис. 2.42.

Оптоэлектронный коммутатор состоит из четырёх каналов, каждый из которых имеет свой адресуемый выход (рис.2.43). Так, выход канала a^6 поступает на вход соседней ячейки, выход канала I^6 - на вход оптического преобразователя. Выход ещё двух каналов коммутатора связан, соответственно, с локальными шинами $L1$ и $L4$.

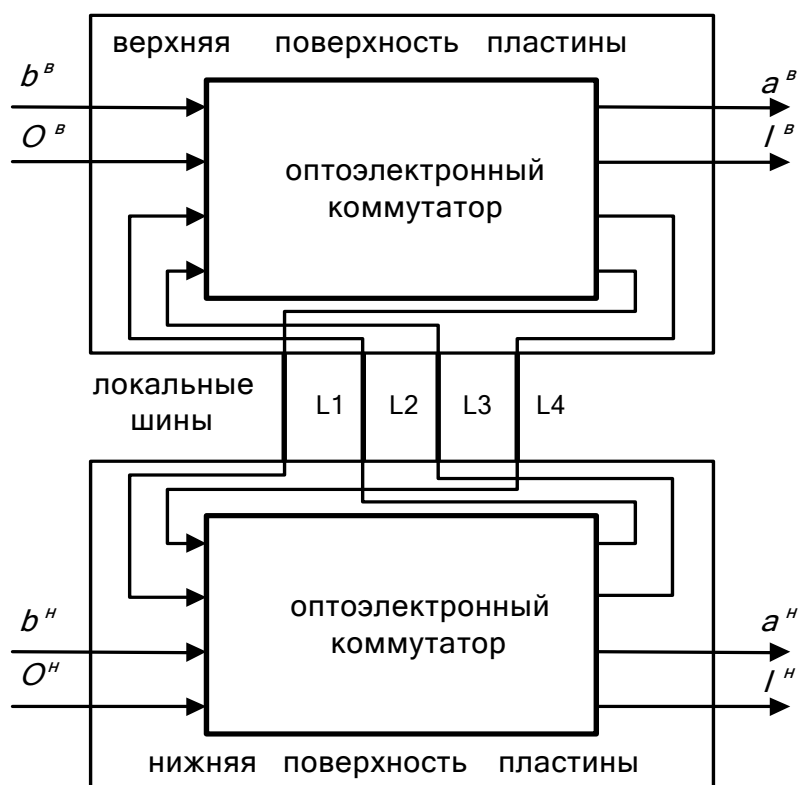


Рис. 2.43. Оптоэлектронные коммутаторы

Входами первых двух каналов коммутатора являются выходы ячейки b^6 и оптического канала O^6 , а также локальные шины $L2$, $L3$. Аналогично устроен коммутатор ячейки, расположенной на нижней поверхности пластины.

Когда локальные шины не задействованы, “сопряжённые” коммутаторы обеспечивают коммутацию данных с выхода ячейки и оптического

преобразователя и их передачу на вход соседней ячейки и вход оптического преобразователя. Каждый из «сопряжённых» коммутаторов состоит из двух пар двухвходовых и четырёхвходовых коммутаторов. Двухвходовые коммутаторы обеспечивают связь с локальными шинами и коммутацию данных с выходов ячейки и оптического преобразователя. Четырёхвходовые коммутаторы связаны с входами соседней ячейки и оптического канала и осуществляют коммутацию данных с выходов ячейки, оптического канала и локальных шин.

Поскольку вычислительная ячейка связана с восемью соседними и содержит восемь оптоэлектронных коммутаторов, то поле управления ими состоит из восьми подполей управления каждого из коммутаторов (рис. 2.44).

Q 1			Q 2			Q 3			Q 4			Q 5			Q 6			Q 7			Q 8		
1	...	6	7	...	12	13	...	18	19	...	24	25	...	30	31	...	36	37	...	42	43	...	48
Управление коммутатором ячейки												Управление коммутатором ячейки											
P 1		P 2		P 3	P 4	...						P 1		P 2		P 3	P 4						
1	2	3	4	5	6	...						1	2	3	4	5	6						

Рис. 2.44. Регистр управления оптоэлектронными коммутаторами ячейки

Подполе включает в себя четыре команды управления (*P1-P4*). Команды *P1*, *P2* управляют связью ячейки, соответственно, с входами оптического канала и соседней ячейки.

Следующим шагом развития концепции однородных вычислительных сред может стать замена регистровых каналов связи в ОВС на радиоканалы. Действительно, как показали исследования, значительное число ячеек ОВС (около 30%) не используется в вычислительном процессе, а идет под трассировку информационных потоков, т.е. на создание так называемых регистровых каналов связи, по которым информация перемещается от одного спецпроцессора к другому в соответствии с вложенным в поле ОВС графом задачи.

Уже на ранних этапах работ по созданию программного обеспечения было замечено, что потери от трассировок информационных потоков через

регистровые каналы, образуемые ячейками среды, были бы сведены к нулю, если бы спецпроцессоры обменивались информацией через радиоканалы.

Использование радиоканалов радикально упрощает процедуру укладки графа задачи в поле среды, которая, вообще говоря, сложно формализуется и, следовательно, затрудняет создание эффективной программы транслятора, которая бы в автоматическом режиме могла бы осуществлять оптимальную укладку графа задачи в поле среды заданной геометрии. Если связь между спецпроцессорами осуществлять по радиоканалам, то расположить спецпроцессоры в поле ОВС можно было бы предельно компактно [124].

Использование радиоканалов позволяет отказаться от последовательного соединения программных регистров в длинный регистр строки ячеек пластины и вести настройку ячеек параллельно. Таким образом, вместо длинных регистров настройки можно было бы использовать автономные регистры программирования ячеек, что не только снимает проблемы надёжности длинных регистров, но и позволяет оперативно перенастраивать фрагменты поля ОВС. Радиоканал ячейки даёт возможность с минимальной задержкой сообщать программе, отвечающей за организацию вычислительного процесса в поле ОВС, о координатах дефектных ячеек, которые выявляются самотестированием ячейки.

Преимущества радиоэлектронной коммутации в наибольшей степени проявляются при создании ОВС на пластине, когда необходима организация информационных связей между ячейками, расположенными на разных пластинах [136].

В ОВС с радиоканалами отсутствуют шины, которые должны использоваться в ОВС с оптической связью для установления быстрой связи между удалёнными ячейками. В ОВС с радиоканалами отпадает необходимость различать ячейки на смежные, сопряжённые и связанные. Связь между ними устанавливается в общем порядке, как и между удалёнными ячейками, которые находятся или на одной, или на разных пластинах ОВС.

Глава 3. Структурно-процедурная организация вычислений в мультимиконвейерных структурах

3.1. Принципы функционирования мультимакроконвейерных вычислительных структур

Как показано во второй главе, к преимуществам мультимикроконвейерных вычислительных структур на ОВС следует отнести обеспечение максимального темпа конвейерной обработки, а также простоту их технической реализации в виде СБИС и на пластине. Эти преимущества достигаются за счет однобитных операций и, как следствие, максимального упрощения вычислительной ячейки.

Однако элементарность ячейки мультимикроконвейерных структур порождает основные недостатки этих структур: трудоемкость программирования и необходимость использования огромного числа вычислительных ячеек при решении сложных задач. Эти недостатки мультимикроконвейерных вычислительных структур на ОВС привели к тому, что в настоящее время примеры их реального приложения практически отсутствуют.

Как показано в первой главе, недостатков мультимикроконвейерных структур лишены в значительной степени мультимакроконвейерные вычислительные структуры, основные принципы построения которых предложены в работах [50,72,80]. Эти принципы в настоящее время нашли широкое применение на практике. Поэтому последующие главы монографии посвящены теоретическому развитию и практической реализации мультимакроконвейерных вычислений.

В отличие от мультимикроконвейерных вычислительных структур, которые реализуются на простейших однобитных вычислительных ячейках, мультимакроконвейерные структуры строятся на так называемых *макропроцессорах* (МАП), которые способны реализовывать крупные операции

с многоразрядными операндами [48]. Макропроцессор представляет собой некоторый набор элементарных процессоров (ЭП), объединяемых в единый программно-неделимый вычислительный ресурс с помощью локального пространственного коммутатора (K_1) (рис. 3.1). Внутренний коммутатор макропроцессора, как правило, соединяет по полному графу все информационные входы и выходы элементарных процессоров, а также внешние информационные входы X , внешние информационные выходы Z макропроцессора. Настройка процессора на крупную функционально законченную операцию (макрооперацию) производится с помощью блока макроопераций (БМ). Одновременно макропроцессор может реализовывать только одну макрооперацию. Недействующие в макрооперации элементарные процессоры будут простаивать.

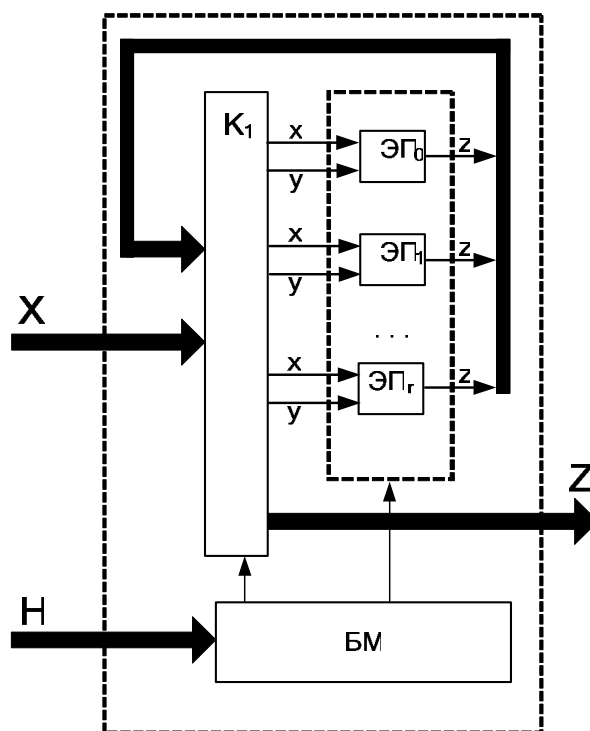


Рис.3.1. Структурная схема макропроцессора

В свою очередь, каждый элементарный процессор в общем случае может представлять собой некоторое арифметико-логическое устройство (АЛУ), дополненное блоком синхронизации операндов (БО) и блоком команд (БК) (рис. 3.2).

```

graph LR
    x((x)) --> BO[БО]
    y((y)) --> BO
    BO --> x_star((x*))
    BO --> y_star((y*))
    x_star --> ALU[АЛУ]
    y_star --> ALU
    h((h)) --> BK[БК]
    BK --> BO
    BK --> ALU
    ALU --> z((z))
  
```

Таблица 3.1

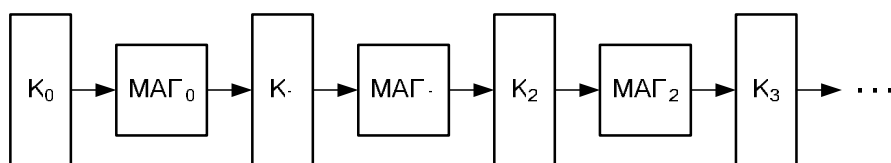
134

Продолжение табл.3.1

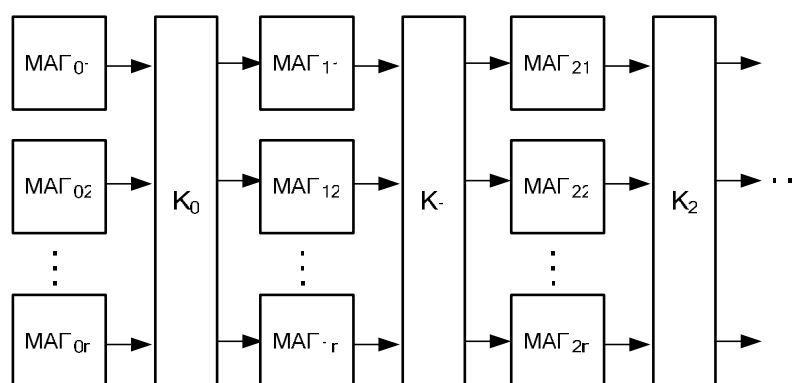
№	Наименование макрооперации	Структурная схема	Мнемоника	Алгоритм операции
1	2	3	4	5
5	Ротор		ROT	$rot(F) = \left(\frac{dFz}{dy} - \frac{dFy}{dz}\right)i + \left(\frac{dFx}{dz} - \frac{dFz}{dx}\right)j + \left(\frac{dFy}{dx} - \frac{dFx}{dy}\right)k$
6	Интегрирование методом Рунге-Кутты		ИРК	$y_{k+1} = y_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ $k_1 = f(x_k, y_k)\Delta x$ $k_2 = f\left(x_k + \frac{\Delta x}{2}, y_k + \frac{k_1}{2}\right)\Delta x$ $k_3 = f\left(x_k + \frac{\Delta x}{2}, y_k + \frac{k_2}{2}\right)\Delta x$ $k_4 = f(x_k + \Delta x, y_k + k_3)\Delta x$
7	Быстрое преобразование Фурье по частоте		БПФЧ	$\dot{Y}_1 = \dot{X}_1 + \dot{X}_2$ $\dot{Y}_2 = (\dot{X}_1 - \dot{X}_2)\dot{W}$

коммутатор, обеспечивающий различные варианты соединения макропроцессоров друг с другом.

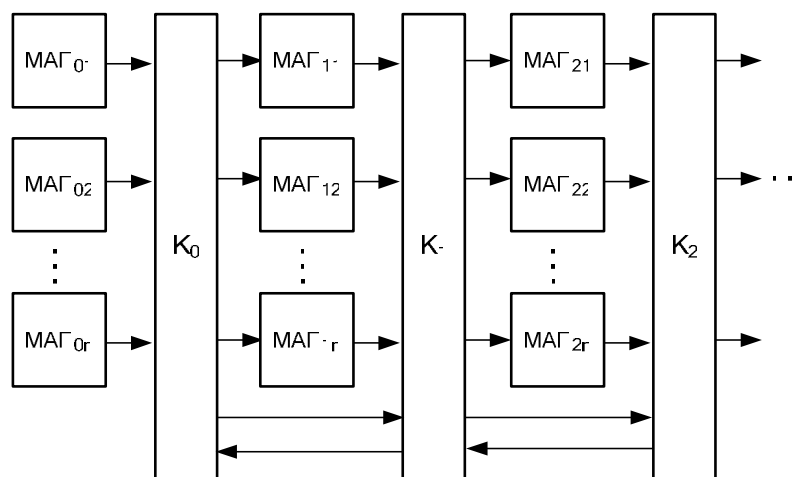
Примеры построения мультимакроконвейерных вычислительных структур с различными вариантами организации системного коммутатора показаны на рис.3.3.



а)



б)



в)

Рис. 3.3. Примеры организации системного коммутатора

Кроме того, в состав мультимакроконвейерной структуры должна входить распределенная память, обеспечивающая возможности параллельной выдачи массива входных данных на входы макропроцессоров и записи результатов вычислений с выходов макропроцессоров.

Обобщенная схема мультимакроконвейерной структуры приведена на рис. 3.4. В ее состав входят: множество макропроцессоров (МАП), состоящих из элементарных процессоров (ЭП), соединяемых между собой с помощью локального коммутатора K_1 ; распределенной памяти (РП), включающей в свой состав контроллер распределенной памяти (КРП) и множества каналов РП, а также системный коммутатор K_2 , предназначенный как для соединения макропроцессоров с друг с другом в конвейерные цепочки, так и для их параллельного доступа к каналам распределенной памяти.

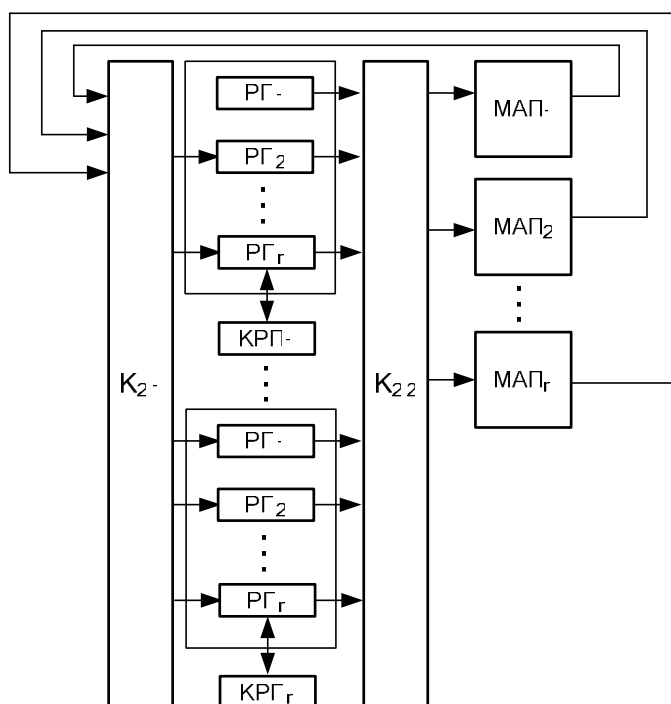


Рис. 3.4. Обобщенная схема мультимакроконвейерной структуры

Коммутатор K_2 конструктивно может быть разделен на две секции: $K_{2.1}$, обеспечивающий соединение МАП и РП, и $K_{2.2}$, обеспечивающий соединение РП и МАП.

На рис. 3.5 представлена схема отображения информационного графа решаемой задачи в мультимакроконвейерную вычислительную структуру. Внутренние дуги информационного графа задачи отображаются на множество коммутационных связей, реализуемых коммутатором K_1 . Каждая операционная вершина информационного графа задачи реализуется с помощью макропроцессора. При этом операционной вершине графа должна быть приписана макрооперация, входящая в набор макроопераций, структурно

реализуемых с помощью элементарных процессоров, входящих в состав макропроцессора.

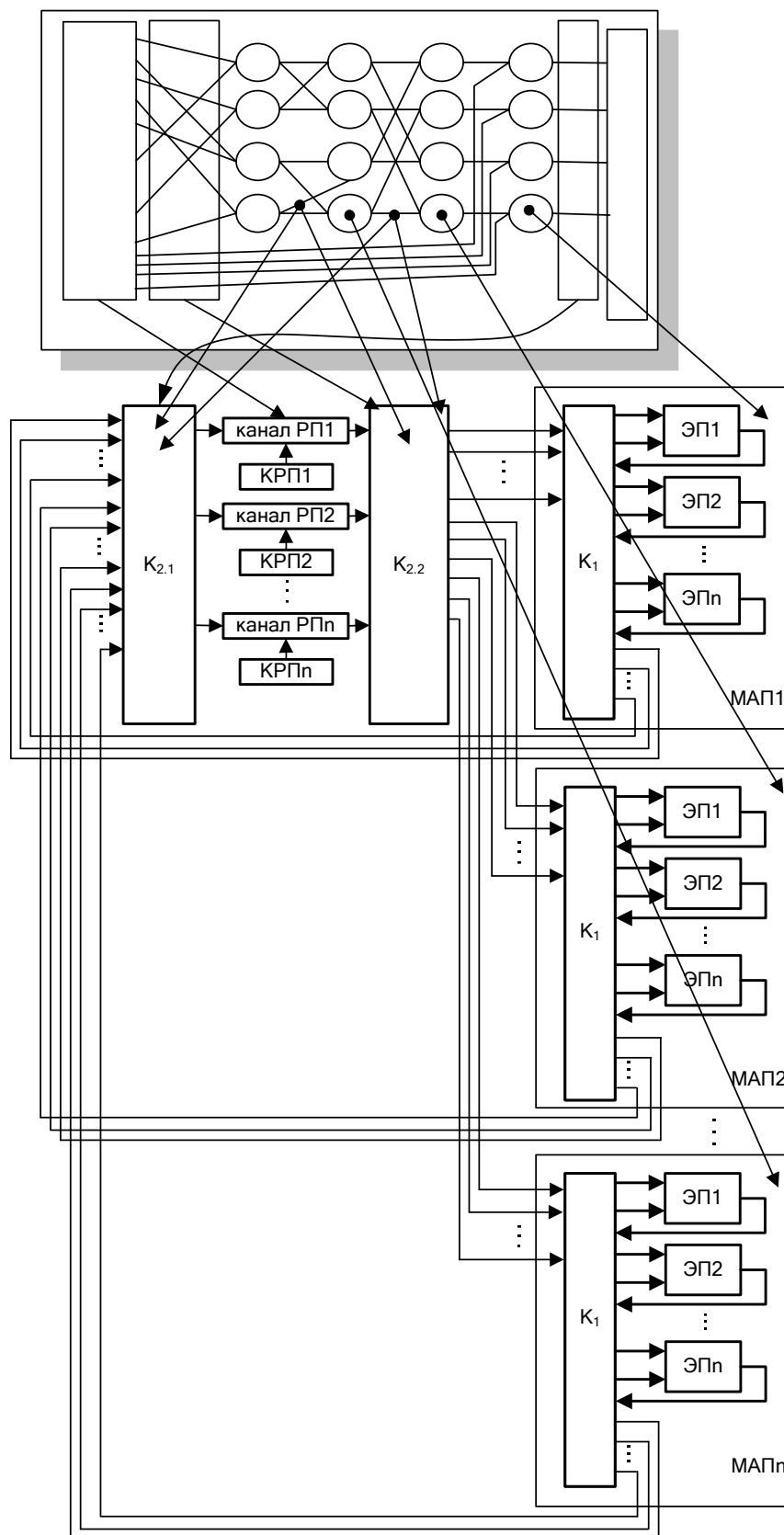


Рис. 3.5. Схема отображения информационного графа задачи в мультимакроконвейерную вычислительную структуру

Следует отметить, что в отличие от стандартного микропроцессора элементарный процессор макропроцессора не управляет процессом обработки информации, а лишь реализует соответствующую ему операцию над операндами, поступающими на его входы. Множество входных и выходных дуг информационного графа реализуется коммутатором $K_{2,2}$, обеспечивающим подключение каналов распределенной памяти к тем или иным входам и выходам макропроцессоров. При этом массивы входных данных отображаются в соответствующие каналы распределенной памяти, процедура обращения к которым реализуется с помощью контроллера распределенной памяти (КРП).

При программировании мультимакроконвейера в каждый макропроцессор посылается макрокоманда, содержащая информацию об элементарной операции, выполняемой каждым элементарным процессором, входящим в его состав, а также о соединении входов и выходов локального коммутатора K_1 . Для настройки межмакропроцессорных соединений, а также соединения входов и выходов макропроцессоров с каналами распределенной памяти используется команда, задающая соединения входов и выходов коммутатора K_2 . Для организации процедур чтения и записи в каналы распределенной памяти служит оператор обращения к распределенной памяти, в котором заданы параметры цикла чтения и записи.

По сравнению с мультимикроконвейерной организацией рассматриваемая мультимакроконвейерная вычислительная структура обладает рядом преимуществ. Во-первых, за счет использования набора крупных макроопераций можно существенно упростить процесс программирования вычислительной структуры, поскольку в этом случае программирование будет осуществляться не в терминах однобитных микроопераций, как в мультимикроконвейерной вычислительной структуре, а в терминах крупных макроопераций конкретной предметной области.

В качестве примера рассмотрим задачу построения радиолокационной карты [4]. Информационный граф задачи представлен на рис. 3.6.

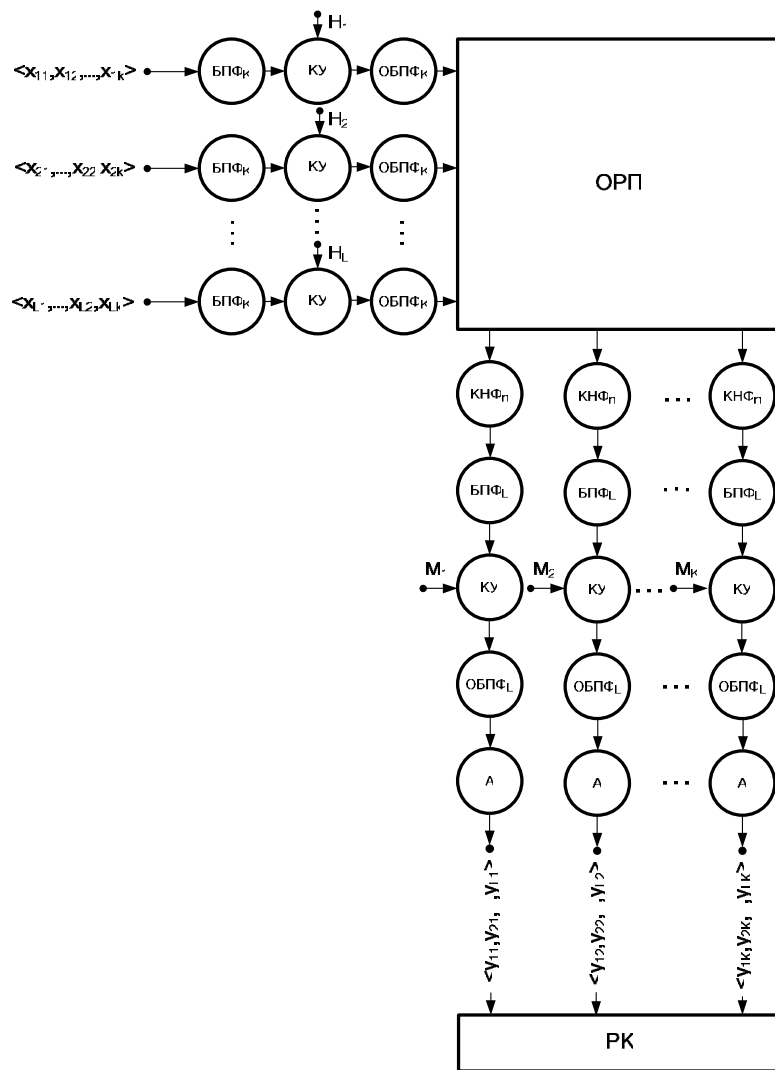


Рис.3.6. Информационный граф задачи построения радиолокационной карты

С РЛС информация поступает параллельно по L каналам, соответствующим строкам. В каждом канале последовательно осуществляются быстрое преобразование Фурье размерности K (БПФ_к), комплексное умножение на коэффициенты H_i ($i=1,2,\dots,L$) и обратное быстрое преобразование Фурье размерности K (ОБПФ_к). Результаты обработки поступают на блок ортогональной распределенной памяти (ОРП), реализующий транспонирование данных.

Дальнейшая обработка ведется параллельно по K каналам, соответствующим столбцам. В каждом канале последовательно осуществляются комплексная нерекурсивная фильтрация размерности n (КНФ_п), быстрое преобразование Фурье размерности K (БПФ_к), комплексное умножение (КУ) на коэффициенты M_i ($i=1,2,\dots,K$) и обратное быстрое

преобразование Фурье размерности L (ОБПФ_L), а также вычисление амплитуды сигнала (A). Совокупность значений амплитуд образует радиолокационную карту (РК).

Данный информационный граф может быть реализован в мультимакроконвейерной вычислительной структуре с помощью библиотеки макроопераций, предназначенных для решения задач цифровой обработки сигналов. Каждой вершине графа ставятся в соответствие определенная макрооперация, коммутационная структура или макрообращение к множеству каналов распределенной памяти. Благодаря этому информационный граф без семантического разрыва отображается на архитектуру мультимакроконвейерной структуры.

Однако здесь следует учесть одно важное обстоятельство. Чем крупнее будут макрооперации, приписанные вершинам информационного графа задачи, тем сложнее будет их оптимальное отображение в структуру макропроцессоров. Иными словами, чем крупнее будут макрооперации, тем менее эффективно будет использоваться вычислительный ресурс структур, поскольку часть элементарных процессоров макропроцессоров может быть просто не задействована в вычислительном процессе.

Очевидно, что этого недостатка можно избежать только путем уменьшения сложности макроопераций, приписанных вершинам информационного графа. В пределе, каждой вершине информационного графа задачи может быть приписана элементарная операция, реализуемая элементарным процессором. В этом случае можно обеспечить максимальное использование всего вычислительного ресурса в вычислительном процессе. Таким образом, пользователь мультимакроконвейерной структуры находится перед альтернативой: укрупнение макроопераций ведет к упрощению программирования структуры, но одновременно снижает эффективность использования ее вычислительного ресурса, и наоборот.

Вторым преимуществом, отличающим мультимакроконвейерную вычислительную структуру от мультимикроконвейерной, является наличие в ее

составе двухкаскадной коммутационной системы, состоящей из локальных коммутаторов макропроцессоров и системного коммутатора. Такая организация коммутационной системы позволяет обеспечить ее широкие коммутационные возможности без привлечения для этих целей ресурсов основных вычислительных узлов элементарных процессоров, в то время как в мультимикроконвейерных структурах до 30% вычислительных ячеек используется не для вычислений, а для организации транзита данных.

Поскольку последующие главы монографии посвящены мультимакроконвейерным вычислительным структурам, то будем использовать установившийся термин «реконфигурируемые мультиконвейерные вычислительные структуры» и обозначать их аббревиатурой РМВС.

3.2. Преобразование информационных графов в структурно-процедурные программы

В предыдущем параграфе при изложении основной идеи организации реконфигурируемой структуры был рассмотрен частный случай, когда информационный граф задачи полностью размещается на аппаратном ресурсе системы. В реальности, как правило, это не так, поскольку аппаратного ресурса не хватает для размещения сложных информационных графов прикладных задач. Как показано в параграфе 1.5, в этом случае целесообразно использовать структурно-процедурную организацию вычислений. Рассмотрим особенности структурно-процедурной организации вычислений более подробно.

В общем виде информационный граф можно представить в виде

$$G = \langle (X, A_x), (Q, A_q), (Y, A_y) \rangle,$$

где X – множество входных вершин;

Y – множество выходных вершин;

Q – множество операционных вершин.

Объединение множества входных дуг A_x , операционных дуг A_q и выходных дуг A_y описывает информационные связи в задаче.

Для любой дуги $\langle u, v \rangle$, принадлежащей A_X , вершина $u \in X$, а вершина $v \in Q$. Для любой дуги $\langle u, v \rangle$, принадлежащей Q , вершина $u \in Q$ и вершина $v \in Q$. Для любой дуги $\langle u, v \rangle$, принадлежащей Y , вершина $u \in Q$, вершина $v \in Q$. Каждой операционной вершине множества Q должна быть приписана некоторая макрооперация. Как показано в предыдущем параграфе, от того, что будет пониматься под макрооперацией, зависит оптимальность размещения информационного графа задачи в реконфигурируемой вычислительной структуре. Наилучший вариант размещения может быть достигнут в случае, если каждой вершине множества Q будет приписана элементарная операция, входящая в набор операций, реализуемых элементарным процессором. Поэтому в дальнейшем, для общности рассуждений, будем считать, что вершине $q_j \in Q$ информационного графа приписана элементарная операция $o_j \in O$, где O – множество элементарных операций, реализуемых элементарным процессором.

Если число элементарных процессоров в реконфигурируемой структуре не меньше числа операционных вершин Q , и все дуги множеств A_X , A_Q и A_Y можно реализовать с помощью локальных коммутаторов K и системного коммутатора K_1 , то информационный граф задачи тривиальным образом реализуется в реконфигурируемой структуре (рис. 3.7).

Операционная вершина графа в данном случае будет соответствовать элементарному процессору, настроенному на выполнение определенной арифметико-логической операции. Входная информационная вершина должна соответствовать каналу распределенной памяти, в который предварительно помещены исходные данные, а выходная информационная вершина – каналу распределенной памяти, в котором по окончании процесса решения задачи окажутся результаты вычислений.

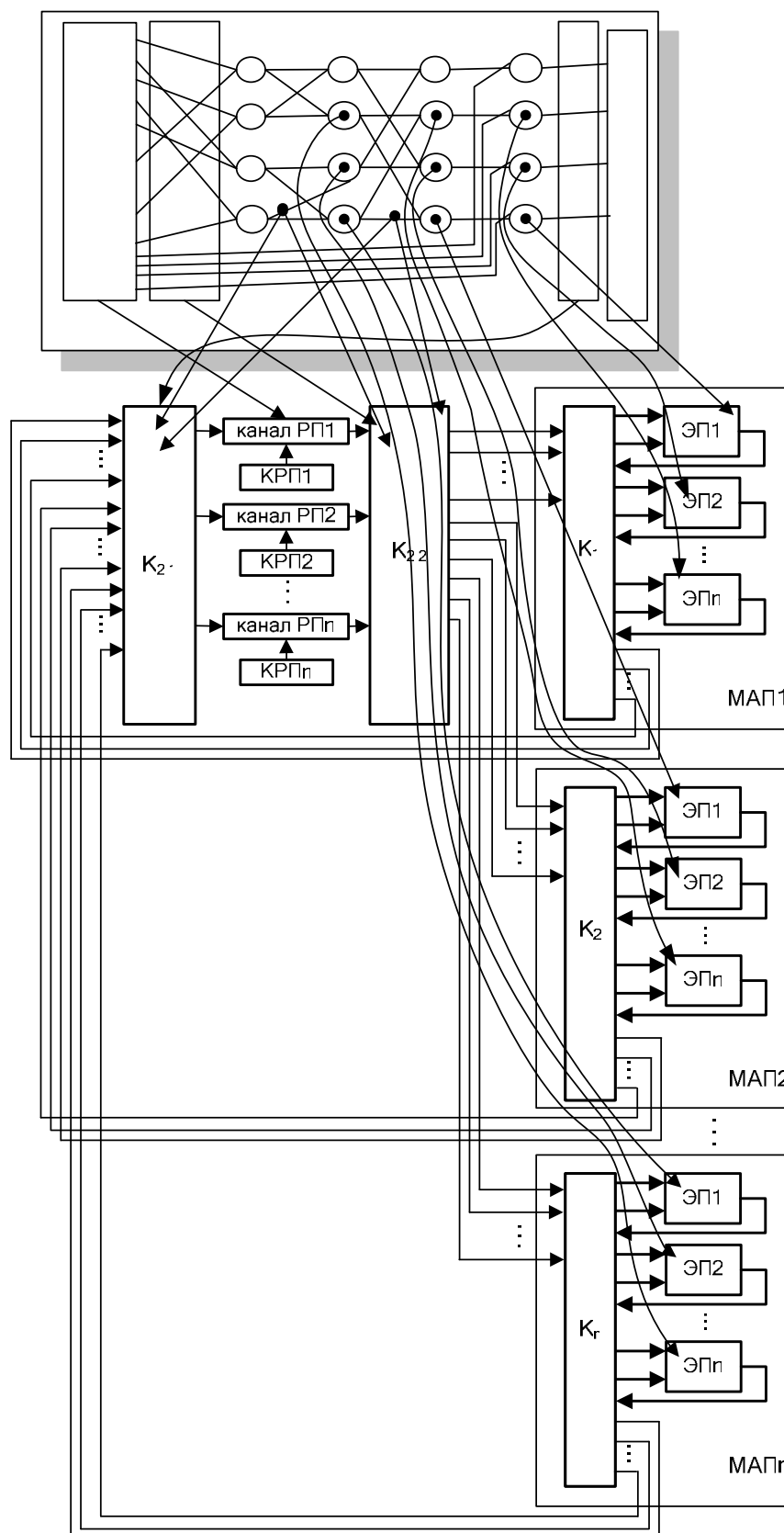


Рис.3.7. Реализация информационного графа задачи в реконфигурируемой структуре

Такую организацию вычислений, при которой весь информационный граф задачи отображается на имеющийся ресурс мультимакроконвейерной

структуры, будем называть *структурной организацией вычислений*. Очевидно, что при структурной организации вычислений достигается минимальное время обработки потока входных данных, поскольку в рамках мультимакроконвейерной структуры формируется специализированный вычислитель, ориентированный на решение текущей задачи. Однако на практике обеспечить структурную организацию вычислений не представляется возможным, поскольку число операционных вершин информационного графа задачи, как правило, оказывается значительно больше числа элементарных процессоров в реконфигурируемой структуре.

Очевидный выход из этой ситуации заключается в сегментировании информационного графа задачи на непересекающиеся подграфы, которые затем могут быть последовательно структурно реализованы. Задача, описываемая в виде информационного графа G , может быть представлена в виде множества крупных функционально законченных вычислительных фрагментов P_i

$$G = \bigcup_{i=1}^M P_i,$$

таких, что если существуют вершина $a \in P_i$ и вершина $b \in P_j$, и существует хотя бы одна дуга $\langle a, b \rangle$, то $i \leq j$. Будем называть дуги, соединяющие подграфы, *внешними*, а дуги, принадлежащие подграфу P_i , - *внутренними*.

Подграфы информационного графа задачи, последовательно реализуемые на реконфигурируемой структуре, принято называть *кадрами* [50], а механизм последовательного обхода информационного графа задачи подграфами – *структурно-процедурной программой*.

Определение 3.1. *Под структурно-процедурной организацией вычислений понимается функционирование реконфигурируемой мультимакроконвейерной вычислительной структуры по единой управляющей программе, при этом минимальной и программно-неделимой конструкцией является кадр, который реализуется структурно.*

Можно сказать, что реконфигурируемые структуры со структурно-процедурной организацией вычислений являются гибридом Фон-неймановской

архитектуры и архитектуры потока данных. Такая организация вычислений обеспечивает детерминизм выполнения программы, что в общем случае недостижимо в многопроцессорных системах, построенных по традиционной мультипроцедурной архитектуре, и высокую эффективность параллельных вычислений на широком классе задач.

Кадр определяется как совокупность элементарных процессоров и блоков распределенной памяти, объединенных с помощью перестраиваемой коммутационной системы в единый вычислительный ресурс, который реализует функциональные преобразования входных информационных потоков в выходные [50]. Как правило, функционирование кадра производится в режиме потока операндов [46]. Можно утверждать, что кадр является специализированной вычислительной структурой, разработанной программистом для аппаратного решения фрагмента задачи.

Кадр часто описывается как объект, определенный в виде тройки $\langle Q, I, O \rangle$, где Q - граф кадра, вершинами которого являются операционные вершины, соответствующие элементарным процессорам, настроенным на выполнение операций, которые в ходе реализации кадра не изменяются;

I - множество (кортеж) информационных потоков, однозначно связанных со входными вершинами;

O - множество информационных потоков, однозначно связанных с выходными вершинами.

На рис. 3.8 представлено графическое изображение кадра.

Входной информационный поток представляет собой вектор независимых потоков $I = \langle I_1, I_2, \dots, I_n \rangle$, каждому элементу которого I_j взаимно однозначно соответствует поток входных вершин функционального графа кадра. Аналогично, выходной поток представляет собой вектор независимых потоков $O = \langle O_1, O_2, \dots, O_n \rangle$, каждому элементу которого O_i взаимно однозначно соответствует поток выходных вершин графа того же кадра. Элементы входного информационного потока последовательно подаются на входную

вершину. Элементы выходного информационного потока последовательно считываются с выходной вершины.

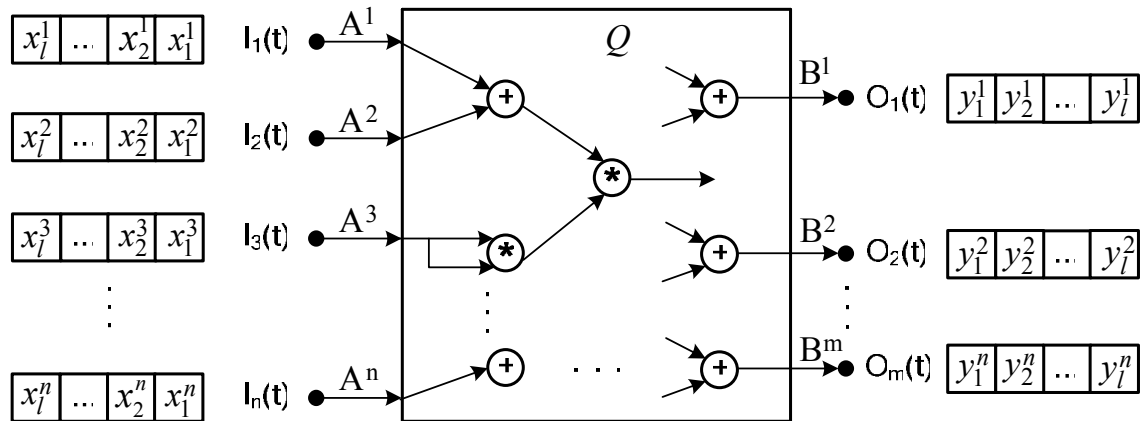


Рис.3.8. Графическое изображение кадра

Определение 3.2. Элементарным кадром будем называть кадр, в котором на каждый вход функционального графа кадра подается единственный операнд и с каждого выхода снимается один результат.

Несложно заметить, что вышеприведенное определение кадра соответствует кортежу элементарных кадров $K = \langle K_1, K_2, \dots, K_l \rangle$, где l - размер информационных потоков на входе (выходе) кадра.

Тогда

$$K = \langle K_i = \langle Q, X_i, Y_i \rangle, i = \overline{1, n} \rangle.$$

Множества входных и выходных вершин элементарного кадра K_i можно определить как функцию от времени следующим образом:

$$\begin{aligned} X_i &= I(t_i), \\ Y_i &= O(t_i). \end{aligned}$$

Следует отметить, что функциональный граф Q всех элементарных кадров кортежа — изоморфный. На рис. 3.9 представлено графическое изображение кортежа изоморфных кадров, в данном случае информационно независимых.

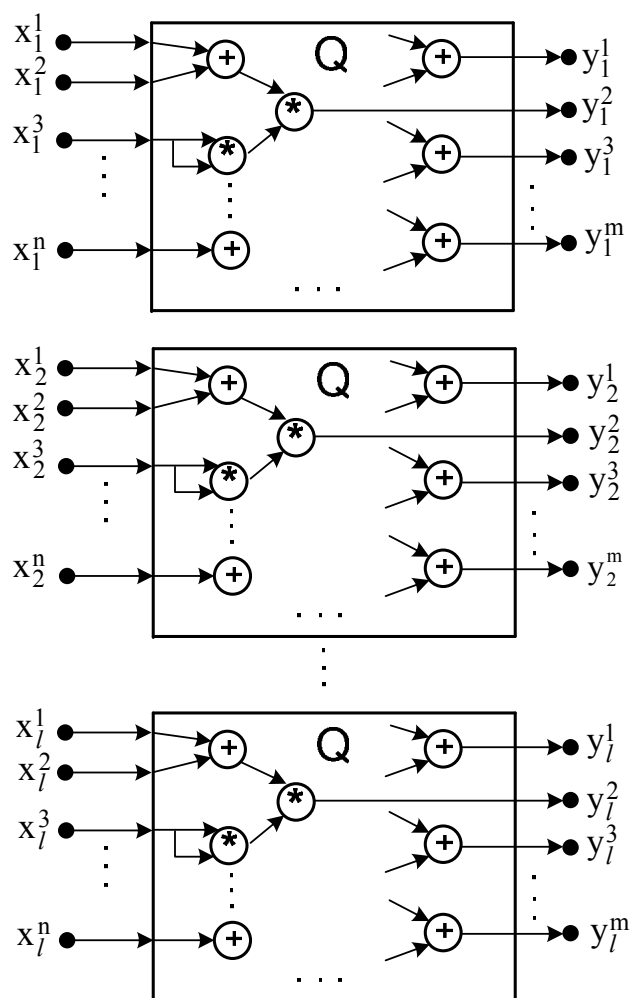


Рис. 3.9. Графическое изображение кортежа изоморфных кадров

Разумеется, справедливо и обратное преобразование: множество изоморфных элементарных кадров можно представить в виде объединенного кадра.

На рис. 3.10 показано преобразование в кадровую форму кортежа изоморфных базовых подграфов (элементарных кадров). При выполнении данного преобразования операционный граф F остается без изменений, а множество входных и выходных информационных вершин базовых подграфов заменяется кортежами входных и выходных информационных вершин X и Y соответственно.

Правило упорядочивания элементарных кадров в кортеж однозначно определяет порядок входных и выходных информационных вершин.

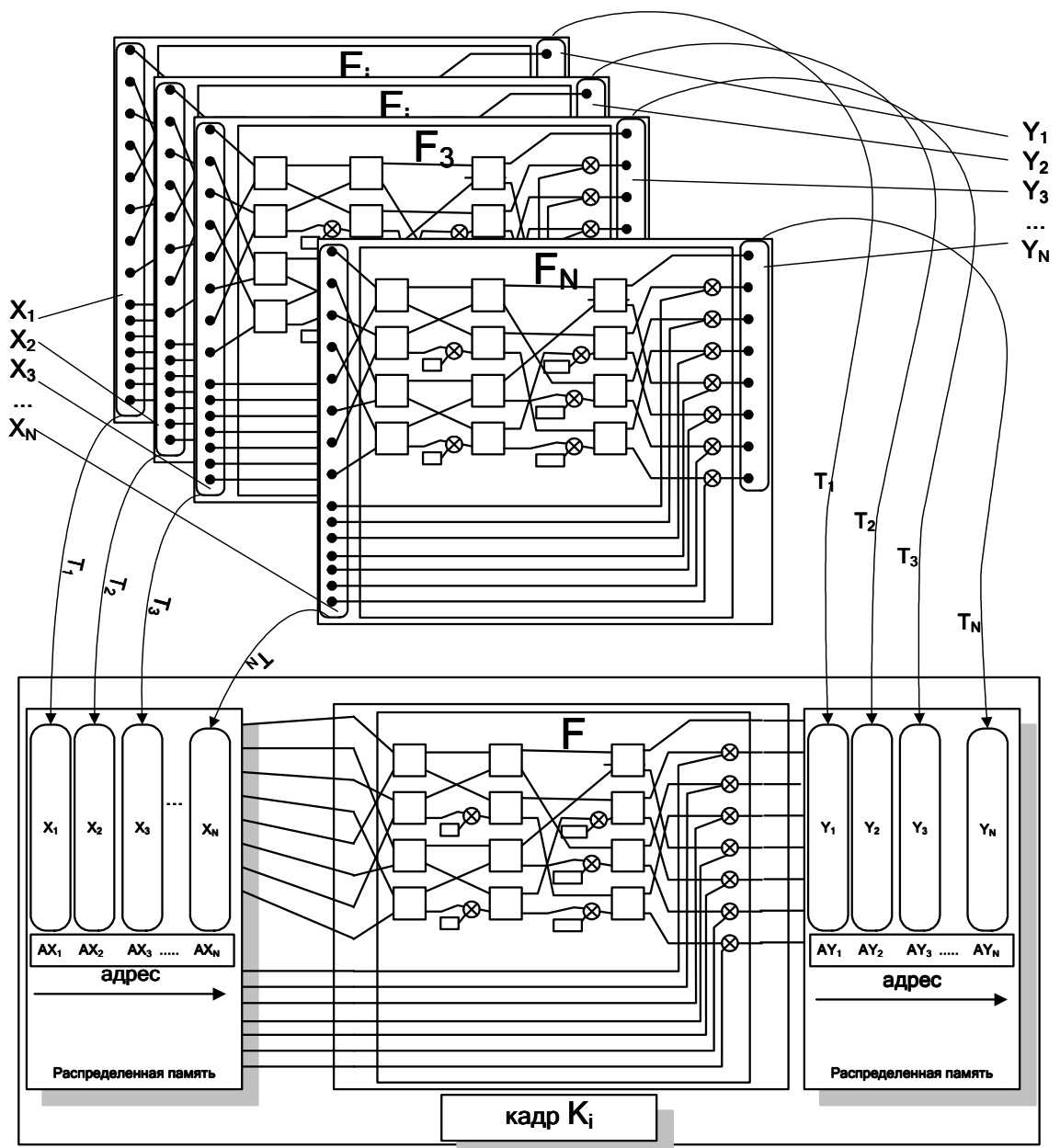


Рис. 3.10. Преобразование в кадровую форму кортежа изоморфных базовых подграфов

Обращаясь к кортежу X и Z по индексу, можно получить требуемый элемент и, следовательно, информационные потоки можно определить как функции последовательного обращения (зависимого от времени) к информационным массивам:

$$I(t_i) = X_i,$$

$$O(t_i) = Y_i.$$

Таким образом, информационные потоки, поступающие на входы кадровых структур, определяются не временными соотношениями задачи, а

информационными зависимостями алгоритма, в которых временные зависимости являются только частными случаями.

Такой подход позволяет значительно расширить применение структурных и структурно-процедурных методов организации параллельных вычислений для задач различных проблемных областей и увеличить эффективность распараллеливания, то есть существенно сократить время и снизить стоимость решения задачи на многопроцессорной системе.

Определение 3.3. *Под информационной зависимостью двух подграфов G_1 и G_2 графа задачи G понимается наличие пути в информационном графе задачи, соединяющем вершину $a \in G_1$ и вершину $b \in G_2$. Два подграфа G_1 и G_2 будут информационно независимыми, если в информационном графе задачи не существует ни одного пути, соединяющего любые вершины, принадлежащие G_1 и G_2 , соответственно.*

Правило упорядочивания подграфов в кортеж однозначно определяет порядок поступления операндов и получения результатов. В этой связи очень важным является соответствие входных и выходных дуг изоморфных подграфов. Данное утверждение можно расширить на произвольное количество информационно-независимых изоморфных подграфов. Упорядочить множество информационных графов в кадрах можно различными способами. После упорядочивания множества информационно-независимых изоморфных подграфов в кортеж существует единственное правило преобразования кортежа подграфов в кадр.

Дополнительным условием, при котором множество подграфов может быть представлено в виде кадра, является наличие рекуррентных правил отображения дуг подграфов g_i :

$$F_R : (x, a_X)_i \rightarrow (x, a_X)_{i+1}, \quad i = 1, 2, \dots, N-1, \quad (3.1)$$

$$F_W : (y, a_Y)_i \rightarrow (y, a_Y)_{i+1}, \quad i = 1, 2, \dots, N-1, \quad (3.2)$$

где x_i - множество входных вершин i -го графа;

a_{Xi} - множество входных дуг i -го графа;

y_i - множество выходных вершин i -го графа;

a_{yi} - множество выходных дуг i -го графа.

Следует отметить, что изоморфные графы, которые преобразуются в кадр, должны быть реализуемыми, то есть к моменту инициализации кадра все информационные элементы, соответствующие входным вершинам всех информационных графов, должны быть получены и находиться в каналах распределенной памяти в соответствии с функцией чтения.

Пусть существуют два изоморфных подграфа G_1 и G_2 , которые не являются информационно независимыми, и имеется единственная дуга, начальная вершина которой $a_1 \in G_1$, а конечная вершина $b_2 \in G_2$, и при этом не существует другого пути информационной зависимости между подграфами G_1 и G_2 . При таких условиях подграфы G_1 и G_2 можно упорядочить в кадр, причем имеется единственное правило упорядочивания $\langle G_1, G_2 \rangle$, определенное дугой $\langle a_1, b_2 \rangle$.

В этом случае производится дополнительное преобразование информационного графа. Формируется дуга, соединяющая вершину a и вершину b . На эту дугу вводится начальный элемент, обеспечивающий реализацию обратной связи. Начальное значение определяется вершиной дуги, входящей в граф G_1 . Вершина b_1 соответствует вершине b_2 . Графы G_1 и G_2 изоморфны. Дуги $\langle x_{b_1}, b_1 \rangle$ и $\langle a_1, b_2 \rangle$ удаляются из графа кадра. Пример преобразования представлен на рис. 3.11.

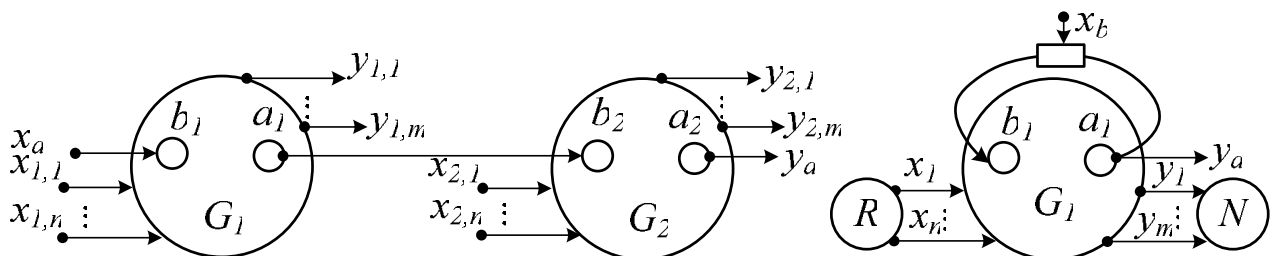


Рис. 3.11. Пример преобразования связанных информационных графов в кадр

Если имеется множество изоморфных подграфов $G_1, G_2, \dots, G_n$, являющихся информационно зависимыми только по смежным дугам α_i , соединяющим подграфы G_i и G_{i+1} , то подграфы $G_1, G_2, \dots, G_n$ могут быть преобразованы в кадр.

Если вершина a_1 соединена с входной вершиной X_a , то такие подграфы могут быть преобразованы в кадр, причем правило упорядочивания подграфов определяется путем, связывающим подграфы $\langle G_1, G_2, \dots, G_n \rangle$. Пример такого графа показан на рис.3.12.

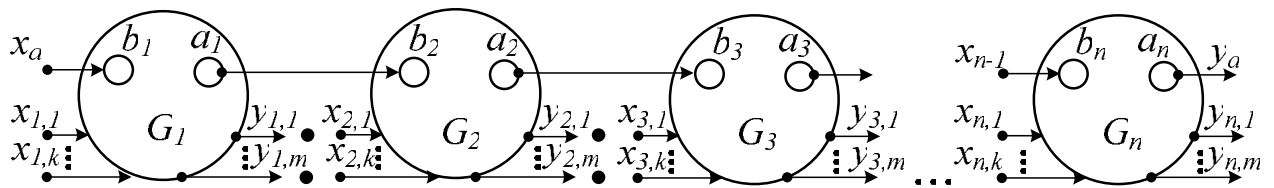


Рис. 3.12. Граф, содержащий изоморфные взаимосвязанные подграфы

Следует отметить, что информационных дуг, соединяющих подграфы, может быть множество. Связи могут следовать не только через смежные фрагменты, но все они должны быть описаны функциональной зависимостью между подграфами согласно выражениям (3.1) и (3.2). В более общем случае должна быть выделена функциональная зависимость между начальной вершиной $a_i \in G_i$ и конечной вершиной $b_{f(i)} \in G_{f(i)}$. Причем связь может быть не только между одиночными подграфами, но и некоторой совокупностью подграфов между собой.

3.3. Преобразование задачи в структурно-процедурную форму

Очевидно, что связь между последовательно реализуемыми кадрами может быть только через оперативную память, которая для многопроцессорных вычислительных систем со структурно-процедурной организацией вычислений должна быть распределенной и многоканальной. Каждой входной дуге a_j кадра ставится в соответствие функция чтения r_j , обеспечивающая поступление информационных потоков на входы кадра из каналов распределенной памяти. Аналогично, каждой выходной дуге b_j кадра должна

быть поставлена в соответствие функция записи w_j , обеспечивающая сохранение информационных потоков с выходов кадра в каналах распределенной памяти.

Будем называть *информационным кадром* следующую тройку:

$$K = \langle R, Q, W \rangle,$$

где R - узел чтения;

Q – структурно-реализованный подграф задачи;

W – узел записи.

Здесь

$$R = \bigcup_{j=1}^{N(A)} \langle a_j, r_j \rangle,$$

где $N(A)$ - мощность множества входных дуг кадра A .

Узел записи представляет собой объединение упорядоченных пар

$$W = \bigcup_{j=1}^{N(B)} \langle b_j, w_j \rangle,$$

где $N(B_j)$ - мощность множества выходных дуг кадра.

При разбиении информационного графа на подграфы, соответствующие кадрам, необходимо выполнить ряд условий, невыполнение которых может существенно снизить эффективность параллельных вычислений или даже привести к алгоритмическим конфликтам. Дополнительным условием разбиения информационного графа задачи на структурно-реализуемые подграфы является требование, чтобы каждый кадр мог быть представлен в виде множества подграфов

$$P_i = \bigcup_{j=1}^N S_{ij}$$

Подграфы S_{ij} будем называть *слоями*. При этом дуги, соединяющие слои, будем называть *межслойными дугами*, а дуги, соединяющие операционные вершины одного слоя, - *внутрислойными дугами*.

Каждый подграф S_{ij} можно представить в виде множества подграфов

$$S_{ij} = \bigcup_{k=1}^K g_{ij,k},$$

при этом все подграфы $g_{i,j,k}$ должны быть изоморфны друг другу. Подграфы $g_{i,j,k}$ будем называть *базовыми подграфами*. Кроме того, внутрислойные дуги должны принадлежать какому-нибудь базовому подграфу. Если существует дуга $\langle a, b \rangle \in S_{ij}$, то вершина $a \in g_{i,j,k}$ и вершина $b \in g_{i,j,k}$. Как следствие, не существует дуг, соединяющих вершины, принадлежащие базовым подграфам одного слоя.

Базовый подграф $g_{ij,k} \subset S_{ij}$ тогда и только тогда, когда существует, по крайней мере, одна дуга $\langle a, b \rangle$ такая, что $a \in q_{ij-1p}$, $b \in q_{ijk}$. Подобное представление графа в виде множества подграфов будем называть *секвенцией графа*. Секвентированный граф вычислительного процесса решения задачи может быть представлен в виде

$$P_i = \Phi_{i,j}(F_{i,j,k}(g_{i,j,k})),$$

где $F_{i,j,k}$ – некоторая функция, отображающая дуги базового подграфа $g_{i,j,k}$ на базовый подграф $g_{i,j,k+1}$;

$\Phi_{i,j}$ – функция, отображающая множество дуг слоя $S_{i,j}$ на множество дуг слоя $S_{i,j+1}$.

Множество функций F является описанием параллельного выполнения базовых подграфов. Множество функций Φ определяет информационную зависимость между слоями.

Если в результате выполнения секвенции в подграфе P_m подграфы $g_{i,n,k} \subset S_{i,n}$ и $g_{i,m,k} \subset S_{i,m}$, $k = 1, 2, \dots, K$ изоморфны для всех значений n и m , $n = 1, 2, \dots, N$, $m = 1, 2, \dots, N$ и существуют обобщающие функции отображения базовых подграфов внутри слоя и функции отображения слоев друг на друга, то такое разбиение графа вычислительного процесса решения задачи G на подграфы P_i будем называть *функциональным секвентированием*. В таком случае каждый подграф P_i может быть представлен в виде

$$P_i = \Phi_i(F_i(g_i)).$$

При функциональном секвентировании подграфы P_i являются функционально-регулярными, а граф вычислительного процесса решения задачи, представленный в этом виде, будет называться *функционально-регулярной формой графа* вычислительного процесса решения задачи.

Функционально-регулярная форма графа позволяет минимизировать аппаратные и программные затраты при решении задачи на многопроцессорной системе и эффективно преобразовать задачу в кадровую форму, обеспечивающую ее эффективную реализацию. Идеальным является случай, когда граф вычислительного процесса решения задачи G может быть представлен в виде единственного разбиения $G = \Phi(F(g))$. В этом случае задача может быть представлена в виде одного кадра. Подробнее преобразования кортежа изоморфных подграфов в кадр рассмотрены в [80].

При реализации кадровых структур в реконфигурируемой мультиконвейерной структуре функция чтения преобразуется в функцию адресации чтения R_A и функцию коммутации чтения R_K , а функция записи, соответственно, в функцию адресации чтения W_A и функцию коммутации чтения W_K . Данные функции описывают процедуры параллельного и последовательного доступа к данным, размещенным в памяти реконфигурируемых структур.

Для эффективной реализации параллельных вычислений целесообразно, чтобы память реконфигурируемой структуры состояла из множества сегментов. При этом данные, требующие параллельного обращения, должны быть расположены в разных сегментах памяти. В результате массивы данных являются двумерными. Первое измерение определяет номер сегмента N_K , а второе – адрес ячейки в канале A_C . Для параллельного обращения к группе данных, размещенных в разных сегментах памяти, требуется задать вектор двумерных адресов $W = \{\langle N_{K1}, A_{C1} \rangle, \langle N_{K2}, A_{C2} \rangle, \dots, \langle N_{KL}, A_{CL} \rangle\}$.

Одной из основных задач синтеза кадровой формы является размещение данных в сегментной памяти. При этом, исходя из кадровой структуры задачи, а также с учетом конфигурации системы, то есть с учетом количества сегментов памяти, топологии коммутатора и количества элементарных процессоров, следует выбрать рациональные варианты размещения данных в памяти, которые исключали бы конфликты параллельного обращения.

Чтобы реализовать бесконфликтное параллельное обращение к каналам распределенной памяти PMBC, необходимо сформулировать набор ограничений, позволяющих выбрать рациональные варианты размещения данных в сегментированной памяти. В соответствии с выбранным вариантом распараллеливания задачи и конфигурацией вычислительной системы определяется семейство вариантов размещения множества данных, используемых в кадрах задачи. На основе этих ограничений может быть выбран вариант размещения элементов многомерных массивов в распределенной памяти вычислительной структуры. С помощью определения порядка перечисления элементов многомерных массивов формируется структура каждого кадра задачи. На основании функций чтения и сформированной структуры данных могут быть реализованы рекуррентные выражения для функции адресации, а также выражения для функции коммутации в каждом кадре.

Система ограничений может включать принципиальные ограничения, невыполнение которых не позволяет реализовать выбранный вариант распараллеливания в реконфигурируемой структуре, а может включать ограничения, лишь снижающие эффективность структурно-процедурной реализации задачи.

Ограничения параллельного доступа можно сформулировать следующим образом. Если функция чтения или функция записи ставит в соответствие моменту времени t два операнда d_1 и d_2 , то они должны быть расположены в разных каналах распределенной памяти. Например, вершине d_1 функция F_s

должна ставить в соответствие адрес $b_1 = \langle N_{K1}, A_{C1} \rangle$, а вершине d_2 функция F_S должна определить адрес $b_2 = \langle N_{K2}, A_{C2} \rangle$, при этом $N_{K1} \neq N_{K2}$.

Если функция записи w_i кадра K_i ставит в соответствие моменту времени t_1 операнд d_1 , а функция чтения r_j кадра K_j ставит при t_2 операнд d_2 , и при этом операнды d_1 и d_2 являются информационно-эквивалентными, то они должны быть расположены в одной ячейке памяти. В этом случае функция F_S ставит в соответствие информационным вершинам d_1 и d_2 адрес $b_1 = \langle N_{K1}, A_{C1} \rangle$.

Количество каналов распределенной памяти, которому соответствует множество переменных с уникальными именами, не должно превышать некоторое наперед заданное значение $C_{\max}$. Ограничение накладывается на множество элементарных процессоров, которые соответствуют операционным вершинам кадров задачи. Мощность множества вершин в кадре задачи не должна превышать некоторое, наперед заданное значение $N_{\max}$, определяемое числом процессоров в графе вычислительной системы.

Одной из основных задач организации эффективных структурно-процедурных вычислений является задача преобразования информационного графа задачи в кадровую форму таким образом, чтобы обеспечить минимальное время решения задачи, что достигается минимизацией числа кадров и выбором максимального потока данных, обрабатываемых в кадрах.

После преобразования информационных подграфов в кадровую форму задача представляется в виде графа кадров G_K . В данном графе каждой вершине ставится в соответствие кадр $K_K = \langle R_K, Q_K, W_K \rangle$. Если между подграфами задачи P_K и P_m существует информационная зависимость, описываемая множеством дуг $D_{k,m}$, то при преобразовании графа задачи в граф кадров множество дуг $D_{k,m}$ заменяется дугой информационной зависимости K_k, K_m . Граф кадров может быть, в свою очередь, представлен в параллельной форме. Пример представления задачи представлен на рис.3.13. В графе кадров

функции чтения и функции записи каждого кадра ставится в соответствие область определения T_k – множество моментов времени $t_{i,k}$.

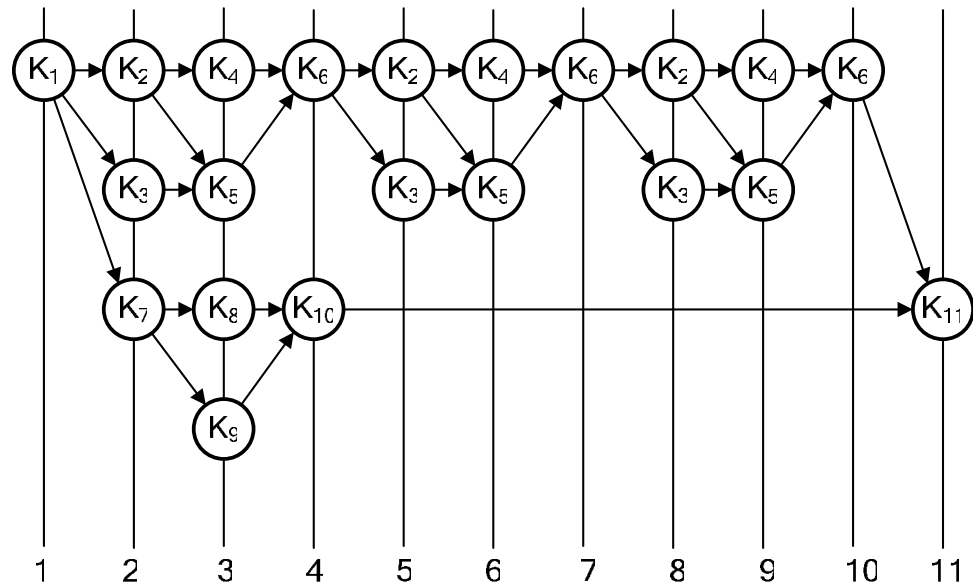


Рис. 3.13. Граф задачи, представленный в кадровой ярусно-параллельной форме

Структурно-процедурная организация вычислений накладывает на реализацию графа кадра следующее ограничение: в реконфигурируемых структурах одновременно может быть реализован только один кадр, что обеспечивает фон-неймановский детерминизм программы. При такой организации вычислительного процесса легко преобразовать граф кадров в структурно-процедурную форму. В самом деле, кадр можно рассматривать как некоторый функциональный оператор. Тогда можно выделить повторяемые или итерируемые участки вычислений и реализовать их в виде циклов.

После преобразования информационного графа в структурно-процедурную форму граф задачи приобретает окончательный вид. Помимо дуг информационной зависимости появляются управляющие дуги, однозначно устанавливающие порядок вызова кадровых структур, а также вершины-решатели [93,94], определяющие, какая из альтернативных ветвей алгоритма должна быть выполнена. При этом в графе структурно-процедурной формы задачи кадры формируются в особые группы, называемые *уровнями*.

Для того чтобы кадры a и b находились в одном уровне, необходимо выполнение двух условий. Во-первых, кадры a и b должны быть информационно независимы, то есть не должен существовать путь по дугам информационной зависимости, соединяющий вершины a и b . Во-вторых, путь по дугам управления, соединяющий кадры a и b , не должен проходить через вершину-решатель. Таким образом, кадры будут находиться в одном уровне тогда и только тогда, когда они могут быть выполнены одновременно.

На рис.3.14 представлен пример структурно-процедурной формы задачи. Здесь пунктирными линиями указаны управляющие дуги, сплошными линиями – информационные. Вершина P_1 является вершиной-решателем. В отличие от ярусно-параллельной формы информационного графа, где множества вершин, расположенные на одном ярусе, могут быть реализованы параллельно, кадры, принадлежащие одному уровню, реализуются только последовательно.

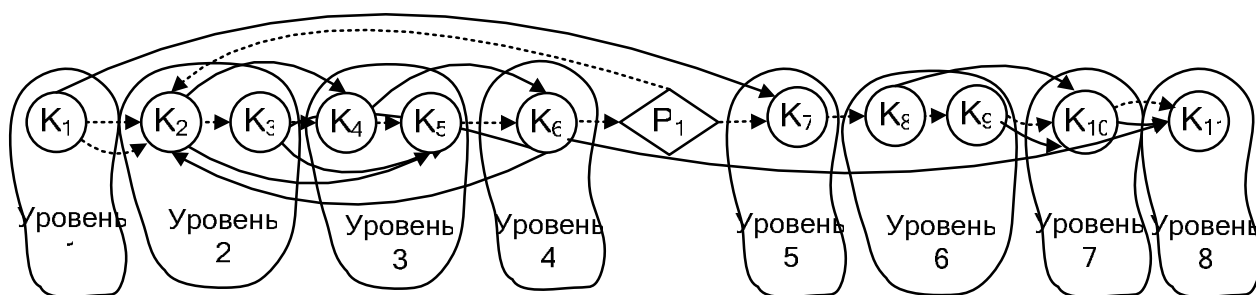


Рис. 3.14. Пример структурно-процедурной формы задачи

При анализе кадровой структуры задач различных проблемных областей можно выделить типовые процедуры обращения к каналам распределенной памяти - макрообращения, а также типовые процедуры параллельного доступа – макрокоммутации и типовые конструкции структурной реализации фрагментов задач – макрооперации.

При создании параллельных программ программист может использовать готовые макрообращения и макрокоммутации совместно с макрооперациями. Подобный подход значительно упрощает процесс программирования задачи и позволяет оперативно создавать эффективные параллельные программы решения задач различных проблемных областей.

Компоненты кадров задачи естественным образом вкладываются в структуру многопроцессорной системы с программируемой архитектурой (рис.3.15).

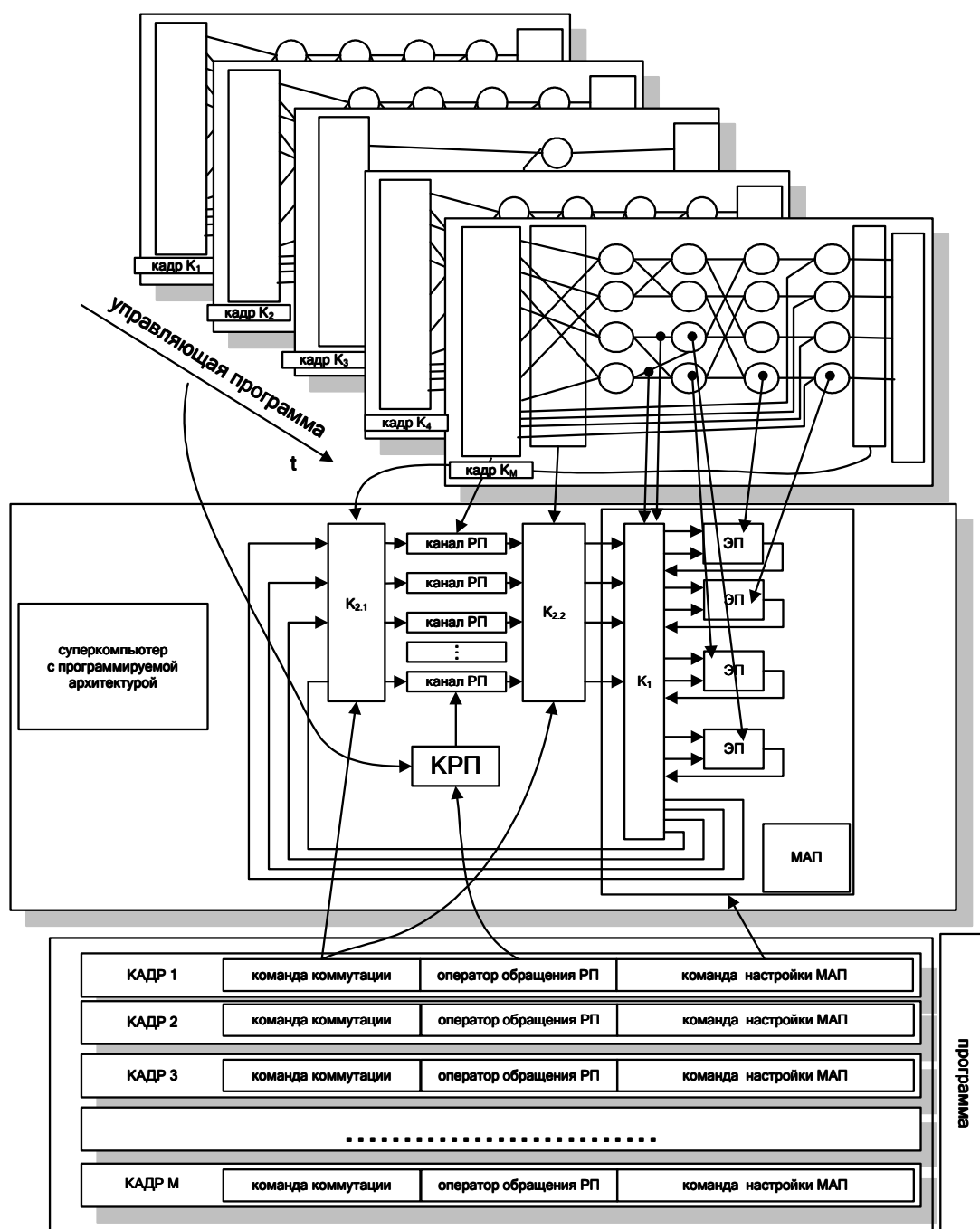


Рис.3.15. Структура системы с программируемой архитектурой

Операционные дуги базового подграфа отображаются во множество коммутационных связей, реализуемых в коммутаторе макропроцессора, множество операционных вершин базового фрагмента – во множество элементарных процессоров. Множество входных и выходных дуг базового фрагмента реализуется во множестве коммутационных связей коммутаторов.

Массивы данных отображаются на множество адресов распределенной памяти. Управляющая программа смены кадров и процедуры обращения к каналам распределенной памяти осуществляется в контроллере распределенной памяти. Таким образом, программист, синтезируя кадры, тем самым создает ряд специализированных вычислителей. Математическая формулировка ряда задач обеспечивает возможность непосредственного преобразования алгоритма в кадровую форму.

3.4. Преобразование функционально-регулярных информационных графов

Функционально-регулярный граф может быть реализован в виде единственного кадра, который представляет собой структурно-реализованный базовый подграф или множество взаимосвязанных базовых подграфов, через которые следует поток данных. Элемент потока данных соответствует множеству информационных вершин на входах структурно-реализуемого подграфа. Элемент потока результата соответствует множеству информационных вершин на выходах структурно-реализованного подграфа. Поэтому преобразование функционально-регулярного графа осуществляется достаточно просто. Пример функционально-регулярного подграфа представлен на рис. 3.16.

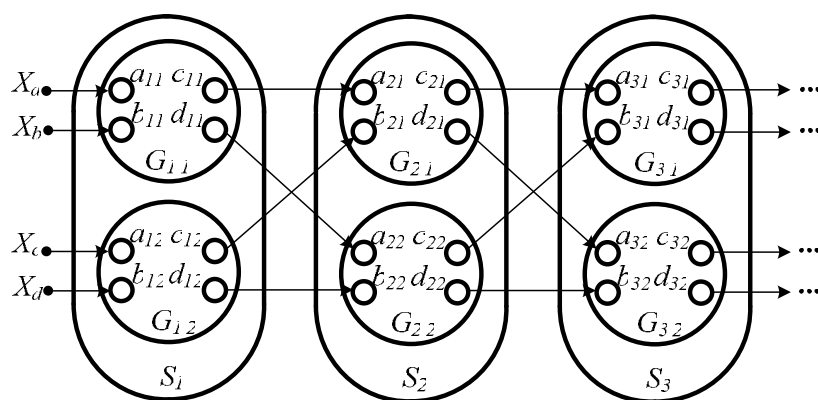


Рис. 3.16. Фрагмент функционально-регулярного подграфа

При объединении информационно-зависимых базовых подграфов в кадр формирование обратных связей происходит не во всех случаях, а только тогда, когда ресурса системы достаточно для структурной реализации множества подграфов и время следования операндов одного слоя через структуру меньше времени заполнения конвейера.

Если ресурса, выделенного для решения задачи, достаточно только для структурной реализации одного базового подграфа, то кортеж подграфов, преобразованный в кадр, можно представить следующим образом: $\langle G_{1,1}; G_{1,2}; G_{2,1}; G_{2,2}; G_{3,1}; G_{3,2} \dots \rangle$ или $\langle G_{1,2}; G_{1,1}; G_{2,2}; G_{2,1}; G_{3,2}; G_{3,1} \dots \rangle$. В первую очередь реализуются базовые подграфы, принадлежащие одному слою. Реализация рекурсивных связей потребуется тогда и только тогда, когда

$$t_{pipeline}(a, b, \mathbf{G}) + t_{setsw} > 2\tau,$$

где $t_{pipeline}(a, b, \mathbf{G})$ - глубина заполнения конвейера при структурной реализации графа $\mathbf{G}$;

t_{setsw} - время переключения источников информационных потоков;

τ - время следования операнда.

Если прохождение двух операндов больше или равно времени появления информации, то реализовывать обратную связь не обязательно. Разумеется, обратная связь реализуется, но уже через память, что не приводит к увеличению времени решения задачи.

Время выполнения кадра можно определить по следующей формуле

$$T_k = t_{pipeline}(\mathbf{Q}) + t_{setup} + \tau N,$$

где $t_{pipeline}(\mathbf{Q})$ - время заполнения конвейера кадра;

t_{setup} - время настройки на кадр;

N - размер потока данных, следующих через кадр;

τ - время следования операнда.

Время следования операнда можно оценить по формуле:

$$\tau = \max(t_i(\mathbf{Q}), t_0),$$

где t_0 - время следования операнда в потоке;

t_l - максимальная величина задержки по обратной связи в графе $\mathcal{Q}$.

При большой величине N временем заполнения конвейера и временем настройки на кадр можно пренебречь. Время вычисления кадра можно оценить как

$$T_k \approx t_0 N.$$

Однако если существует рекурсивная связь, то время выполнения кадра увеличивается до величины

$$T_k \approx t_l(\mathcal{Q})N.$$

При достаточно больших структурных фрагментах это время может многократно превысить минимальное время выполнения кадра. Для случая, когда слой фрагмента информационного графа, преобразуемый в кадр, содержит два базовых подграфа, получим, что для нерекурсивной реализации необходимо, чтобы выполнялось неравенство

$$2t_0 \geq T_p.$$

Величину T_p определим как время, необходимое для записи операнда и переключения информационных потоков коммутатором:

$$T_p = t_{\text{setup}} + t_{\text{setsw}} + \tau.$$

В более общем случае, когда слой фрагмента информационного графа, преобразуемого в кадр, содержит n базовых подграфов, требование рекурсивной связи можно записать как выполнение условия

$$t_0 \geq \frac{T_p}{\lceil n/p \rceil}, \quad (3.3)$$

где p – число одновременно реализуемых базовых подграфов.

Поэтому целесообразно повышать степень распараллеливания p только до тех пор, пока справедливо неравенство (3.3). В противном случае время решения задачи существенно увеличится. В самом деле, если слой фрагмента содержит n базовых подграфов, то при реализации всех базовых подграфов слоя время решения задачи составит

$$T_k = t_{\text{pipeline}}(\mathcal{Q}) + t_{\text{setup}} + t_l L,$$

где L – число слоев.

Если реализуется только некоторое число подграфов m , то

$$T_k \approx t_Q + t_0 L \lceil n/p \rceil,$$

$$t_l = m t_0.$$

Таким образом, для эффективной организации вычислений должны выполняться соотношения

$$L m t_0 \leq t_0 L \lceil n/p \rceil$$

или

$$m \leq \lceil n/p \rceil.$$

Как правило, $\sqrt{N} \leq \lceil n/p \rceil$, что позволяет организовывать структурные вычисления без обратных связей в графе кадра.

Функционально-регулярный граф может быть реализован в виде единственного кадра, который представляется как поток данных, следующих через аппаратно реализованный подграф задачи. Поскольку подграфы g в кортеже являются информационно-независимыми и могут быть реализованы параллельно, то при преобразовании функционально-регулярного графа в кадровую форму возможно реализовать кадровые структуры для различной степени распараллеливания.

3.5. Преобразование в кадровую форму решения задачи Пуассона

В качестве примера рассмотрим преобразование в кадровую форму, точнее, задачи численного решения в прямоугольной области D дифференциального уравнения в частных производных. Пусть необходимо решить дифференциальное уравнение Пуассона

$$\frac{\partial^2 U}{\partial^2 x} + \frac{\partial^2 U}{\partial^2 y} = f(x, y) \quad (3.4)$$

с граничными условиями первого рода.

Рассмотрим численное решение дифференциального уравнения в узловых точках равномерной сетки. После преобразования уравнения (3.4) в конечно-

разностную форму [138] непрерывные функции $U(x, y)$ и $f(x, y)$ заменяются множеством значений в узлах сетки $U_{i,j}$ и $f_{i,j}$, и формируется система линейных алгебраических уравнений. Для решения системы используем итерационный метод одновременной нижней релаксации [105], когда k -е приближение сеточной функции $U_{i,j}^k$ в узловых точках (x_i, y_j) вычисляется по рекуррентной формуле

$$U_{i,j}^k = c_1(U_{i,j-1}^{k-1} + U_{i,j+1}^{k-1}) + c_2(U_{i-1,j}^{k-1} + U_{i+1,j}^{k-1}) + c_3 U_{i,j}^{k-1} - f_{i,j},$$

где c_1, c_2 и c_3 – коэффициенты разностного уравнения.

Вычисления по этой формуле производятся в диапазоне индексов $i = 1, 2, \dots, N-1$, $j = 1, 2, \dots, M-1$. Величины M и N определяются, соответственно, горизонтальными и вертикальными размерами области D и шагом сетки. Итерационный процесс необходимо продолжать до тех пор, пока для всех значений i и j не будет выполнено неравенство

$$|U_{i,j}^r - U_{i,j}^{r-1}| \leq \varepsilon,$$

где ε – константа, определяющая точность решения.

Таким образом, вычислительный процесс решения задачи может быть представлен как упорядоченная последовательность функционально-законченных вычислительных фрагментов $\langle P^1, P^2, \dots, P^k \rangle$, где k – номер последней итерации. В результате выполнения k -го фрагмента P^k формируется множество $U^k = \{U_{i,j}^k, i = 1, 2, \dots, N-1; j = 1, 2, \dots, M-1\}$. Элементы множества U^k являются входными данными для фрагмента P^{k+1} , в котором вычисляется множество $U^{k+1} = \{U_{i,j}^{k+1}, i = 1, 2, \dots, N-1; j = 1, 2, \dots, M-1\}$.

Если представить вычислительный процесс решения задачи в виде вычислительного графа G , то каждому фрагменту P^k соответствует информационный подграф S^k . Информационная зависимость между фрагментами вычислений P^k и P^{k+1} определяется множеством дуг A^k , соединяющих подграфы S^k и S^{k+1} . Для любой дуги, принадлежащей множеству A^k , операционная вершина-источник принадлежит подграфу S^k , а

операционная вершина-приемник принадлежит подграфу S^{k+1} . Таким образом, информационный граф G данной задачи можно представить в виде кортежа информационно-зависимых подграфов $\langle S^1, S^2, \dots, S^{k-end} \rangle$.

При ограниченном числе элементарных процессоров в реконфигурируемой структуре необходимо перейти от параллельной формы задачи к структурно-процедурной, то есть кадровой форме. Для того чтобы представить задачу в кадровой форме, необходимо исходный информационный граф разрезать на множество изоморфных подграфов, которые реализуются последовательно. При этом количество операционных вершин в каждом подграфе не должно превышать количество процессоров, выделенных для решения задачи, а коммутационная система должна реализовать соединения между элементарными процессорами в соответствии с дугами подграфа.

Для преобразования кортежа подграфов в кадровую форму необходимо, чтобы все подграфы кортежа были изоморфны и существовало правило Φ , по которому подграф кортежа отображается на другой подграф кортежа. Кроме того, подграф кортежа желательно представить в виде множества информационно-независимых базовых подграфов и определить правило, по которому базовые подграфы отображаются друг на друга. Данные правила позволяют эффективно преобразовать информационный граф в кадровую форму и создать масштабируемую программу, выполняемую при различных вариантах структурно-процедурного распараллеливания вычислений.

Для данной задачи выполняются оба условия. Поскольку все вычислительные фрагменты P^k идентичны, то все информационные подграфы S^k изоморфны. Правило отображения Φ однозначно определено множеством информационных дуг A^k . Каждый информационный подграф S^k может быть представлен в виде множества изоморфных базовых подграфов $\{g_{i,j}^k, i = 1, 2, \dots, N-1, j = 1, 2, \dots, M-1\}$. Информационный базовый подграф $g_{i,j}^k$ соответствует вычислению значения $U_{i,j}^k$ в узле сетки. При решении задачи математической физики методом одновременных смещений базовые подграфы,

принадлежащие одному подграфу S^k , являются информационно независимыми, поскольку в подграфе S^k не существует дуги, источником которой является операционная вершина одного базового подграфа, а приемником - операционная вершина другого базового подграфа. Информационная независимость базовых подграфов $g_{i,j}^k$, принадлежащих одному подграфу S^k , позволяет создать масштабируемую параллельную программу для решения задачи математической физики.

Для структурной реализации базового подграфа в реконфигурируемой структуре выделяются элементарные процессоры, количество которых соответствует числу операционных вершин базового подграфа. Число вершин базового подграфа определяется количеством арифметических и логических операций. Дуги базового подграфа реализованы в коммутационной системе реконфигурируемой структуры. В соответствии с методами программирования задач математической физики структурно-реализованный базовый подграф, соответствующий вычислениям, осуществляемым в узле сеточной области, будем называть далее *узловым процессором*. Базовый подграф задачи решения уравнения Пуассона показан на рис. 3.17.

Каждый подграф $g_{i,j}^k$ имеет шесть входных информационных вершин

$$\langle f_{i,j}, U_{i,j}^{k-1}, U_{i,j-1}^{k-1}, U_{i,j+1}^{k-1}, U_{i-1,j}^{k-1}, U_{i+1,j}^{k-1} \rangle$$

и одну выходную информационную вершину $U_{i,j}^k$.

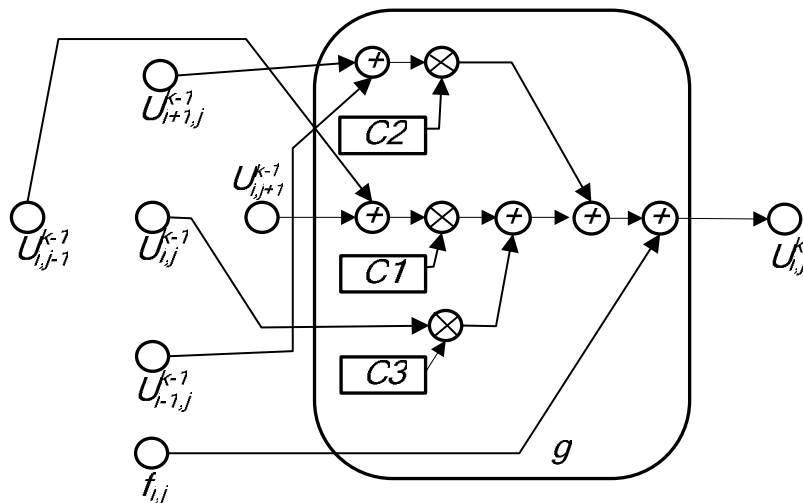


Рис. 3.17. Базовый подграф задачи решения уравнения Пуассона

Промежуточные результаты вычислений на k -й итерации $U_{i,j}^k$ записываются в распределенную память реконфигурируемой структуры. Каждому значению приближения сеточной функции $U_{i,j}^k$ в информационном графе однозначно соответствует одноименная вершина. При включении информационной вершины $U_{i,j}^k$ в граф каждая дуга, принадлежащая множеству A^k , заменяется парой дуг, инцидентных вершине $U_{i,j}^k$, причем для первой дуги информационная вершина $U_{i,j}^k$ является вершиной-приемником, а для второй дуги данная информационная вершина является вершиной-источником. На рис. 3.18 показан информационный граф задачи решения уравнения Пуассона, разрезанный на множество базовых подграфов $g_{i,j}^k$.

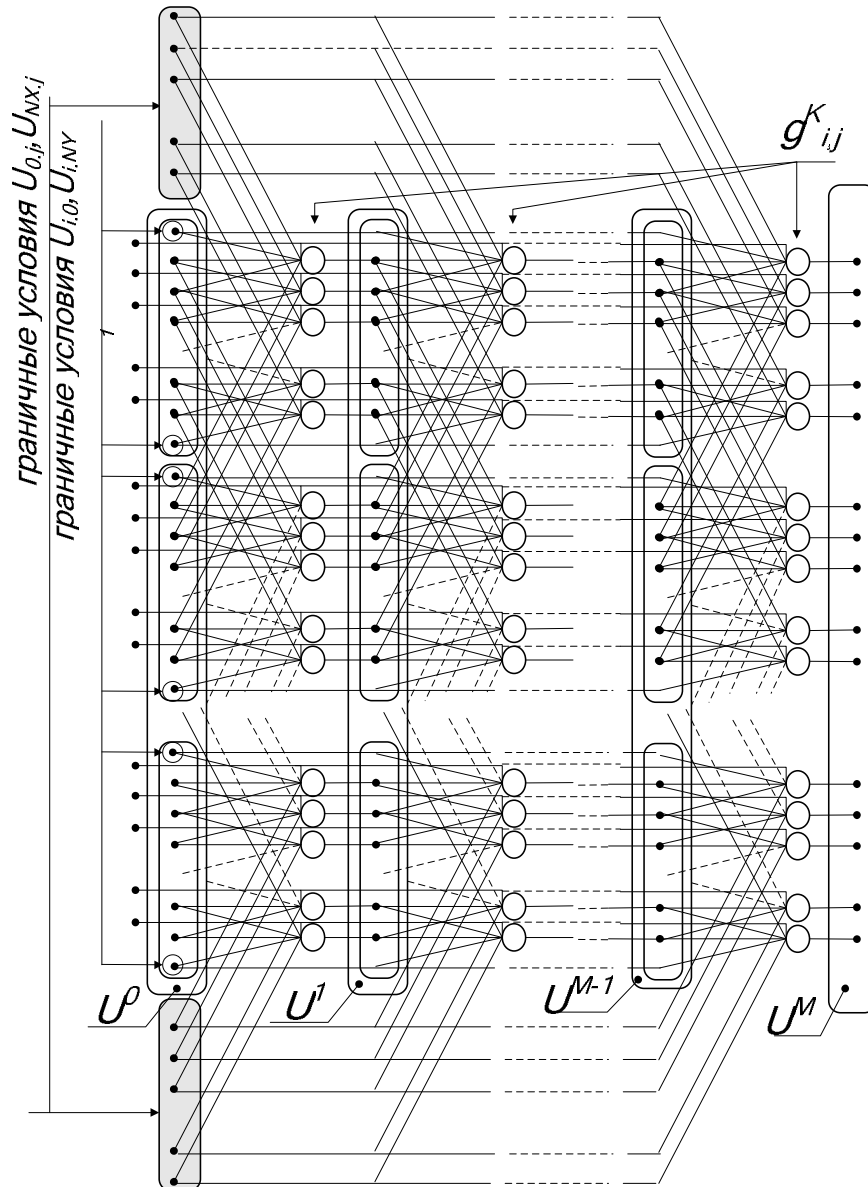


Рис. 3.18. Информационный граф задачи решения уравнения Пуассона

Для реализации задачи математической физики на L узловых процессорах каждый информационный подграф S^k представляется в виде множества изоморфных подграфов $\{F_m, m = 0, 1, \dots, q-1\}$. Величина q определяется выражением

$$q = (M-1) \left\lceil \frac{N-1}{L} \right\rceil.$$

Каждый подграф F_m представляет собой множество из L базовых подграфов (узловых процессоров).

В дальнейшем такую группу базовых подграфов будем называть L -узловым процессором. Обозначим X_m и Y_m соответственно множество входных и выходных информационных вершин подграфа F_m . Множество X_m представляет собой объединение подмножеств, элементами которых являются входные информационные вершины базовых подграфов, принадлежащих подграфу F_m . Аналогично элементами множества Y_m являются выходные информационные вершины базовых подграфов, принадлежащих подграфу F_m .

Для синтеза кадра множество подграфов $\{F_m\}$ необходимо преобразовать в кортеж, однозначно определяющий порядок реализации операций L -узлового процессора. При этом формируется отношение, ставящее в соответствие множество входных вершин X_m времени t_m выполнения операции L -узлового процессора. Обратное отображение будет являться функцией чтения. Аналогично можно определить функцию записи. В дальнейшем при синтезе процедур обращения к данным, размещенным в каналах распределенной памяти, функция чтения (записи) преобразуется в функцию адресации чтения (записи) и функцию коммутации чтения (записи). Функция адресации определяет последовательное обращение к данным в одном канале памяти, а функция коммутации – параллельное обращение к группе данных, расположенных в различных каналах памяти.

При преобразовании задачи в кадровую форму создается структура данных, которая накладывает ряд ограничений на возможность реализации того

или иного варианта распараллеливания задачи. Так, при использовании метода одновременных смещений для данных U^k и результатов вычисления значений сеточной функции U^{k+1} необходимо выделять различные области распределенной памяти M_1 и M_2 . В этом случае при выполнении нечетных фрагментов P^{2k+1} , $k = 0, 1, 2, \dots$, данные $U_{i,j}^{2k}$ поступают на входы L -узлового процессора из области памяти M_1 , а результаты вычислений $U_{i,j}^{2k+1}$ записываются в область памяти M_2 . При выполнении фрагментов P^{2k} , $k = 0, 1, 2, \dots$, данные $U_{i,j}^{2k-1}$ поступают на входы L -узлового процессора из области памяти M_2 , а результаты $U_{i,j}^{2k}$ записываются в область памяти M_1 . Каждой итерации будет соответствовать кадр C , а весь итерационный процесс может быть представлен в виде цикла, в теле которого выполняется кадр C .

Другим ограничением является бесконфликтное размещение данных в распределенной памяти, гарантирующее, что в один момент времени не придется обращаться к данным, расположенным в одном канале по разным адресам. Размещение данных в памяти реконфигурируемых структур определяется вариантом формирования подграфов F и их упорядочиванием в кортеже.

При произвольном способе разрезания графа S^k на подграфы $\{F_m\}$ множество входных вершин X_m может содержать до $6L$ информационных вершин. В этом случае для того, чтобы обеспечить параллельное поступление данных на входы L -узлового процессора, потребуется до $6L$ каналов распределенной памяти, а функции адресации и коммутации могут быть сложны для реализации в реконфигурируемых структурах.

В то же время для данной задачи существует ряд рациональных вариантов разрезания графа S^k на подграфы $\{F_m\}$, которые упрощают процедуры адресации и коммутации. Наибольший интерес представляет вариант разрезания графа S^k на подграфы $\{F_m\}$, когда подграф F_m соответствует вычислениям в соседних узлах сетки, принадлежащих одному столбцу, и содержит базовые подграфы

$$F_m = \varphi\{g_{r+p,s}^k\}, \quad (3.5)$$

где индексы r и s функционально связаны с индексом m . Здесь и далее индекс p принимает значения $0, 1, 2, \dots, L-1$. Подграфы множества (3.5) соответствуют вычислению множества значений сеточной функции $\{U_{r+p,s}^k\}$ в s -м столбце, начиная с r -й строки сетки.

Рассмотрим обобщенную формулу, которая позволяет перейти от одномерной индексации некоторой последовательности групп объектов, определяемой значением x , к двумерной индексации столбцов и строк $\langle y_1, y_2 \rangle$ при размещении этих групп в прямоугольной области. При этом первая группа, включающая в себя K объектов, размещается в начальном столбце y_2^0 , начиная со строки y_1^0 . Следующая группа размещается в столбце с номером $y_2^0 + 1$ и т.д. После размещения очередной группы объектов в последнем столбце следующая группа будет размещена в начальном столбце, начиная со строки $y_1^0 + K$. При таком размещении групп объектов номер строки y_1 и номер столбца y_2 связаны с одномерным индексом x следующими соотношениями:

$$y_1 = y_1^0 + K \left\lceil \frac{x + x_0}{Z} \right\rceil, \quad (3.6)$$

$$y_2 = y_2^0 + \text{mod}_Z(x + x_0), \quad (3.7)$$

где $\lceil * \rceil$ – операция округления в меньшую сторону;

$\text{mod}_a(b)$ – остаток от деления a/b ;

y_1^0 и y_2^0 – начальные значения индексов;

Z – количество элементов в строке;

x_0 – базовое смещение.

Использование базового смещения позволяет привести значение столбца y_2 к величине y_2^0 и значение строки y_1 к величине y_1^0 , если начальное значение x отличается от нуля.

Нетрудно заметить, что формулы (3.6) и (3.7) могут быть использованы для вычисления индексов r и s соответственно. Очевидно, что переменная x

соответствует индексу m подграфа F_m , а переменные y_1 и y_2 - индексам i и s соответственно. При вычислении значений сеточной области обрабатываются только внутренние узлы сетки, поэтому количество обрабатываемых узлов, принадлежащих одной строке сетки, равно $M - 1$. Коэффициент K определяет количество базовых подграфов множества (3.5) и равен L . Поскольку обрабатываемые узлы сетки имеют начальные номера, равные 1 по строке и по столбцу, а индексация базовых подграфов соответствует индексам узлов сетки, то начальные смещения y_0^1 и y_0^2 равны 1. Базовое смещение $x_0 = 0$, поскольку индексация подграфов F_m начинается с $m = 0$. Таким образом, функциональная зависимость одномерного индекса m подграфа F_m с индексами i и s определяется выражениями

$$\begin{aligned} r &= 1 + L \left[\frac{m}{M - 1} \right], \\ s &= 1 + \text{mod}_{M-1}(m). \end{aligned} \quad (3.8)$$

При решении уравнения Пуассона нами используется 5-точечный шаблон. Назовем точку шаблона (x_i, y_j) центральной, точки (x_{i-1}, y_j) и (x_{i+1}, y_j) – верхней и нижней, точки (x_i, y_{j-1}) и (x_i, y_{j+1}) – левой и правой. Подобные обозначения можно ввести для входных информационных вершин базового подграфа $g_{i,j}^k$. Входная информационная вершина $U_{i,j}^{k-1}$ является центральной, а вершины с индексами $(i - 1, j), (i + 1, j), (i, j - 1)$ и $(i, j + 1)$ – соответственно, верхней, нижней, левой и правой.

При объединении подмножеств входных информационных вершин, принадлежащих подграфам множества (3.8), одноименные информационные вершины «поглощаются». Уникальными являются верхняя вершина подграфа $g_{r,s}^k$, нижняя вершина подграфа $g_{r+L-1,s}^k$, а также левые и правые вершины всех подграфов $\{g_{r+p,s}^k\}$. В то же время нижняя вершина подграфа $g_{r+p,s}^k$ является одноименной с центральной вершиной подграфа $g_{r+p+1,s}^k$, а верхняя вершина подграфа $g_{r+p,s}^k$ является одноименной с центральной вершиной подграфа

$g_{r+p-1,s}^k$. Таким образом, при объединении информационных вершин в множество X_m включаются центральные вершины $\{U_{r+p,s}^{k-1}\}$, левые вершины $\{U_{r+p,s-1}^{k-1}\}$, правые вершины $\{U_{r+p,s+1}^{k-1}\}$, а также уникальные вершины $U_{r-1,s}^{k-1}$ и $U_{r+L,s}^{k-1}$.

Множество выходных информационных вершин Y_m подграфа F_m включает в себя вершины $\{U_{r+p,s}^k\}$. Эти выходные вершины являются центральными входными вершинами подграфов F_m , принадлежащих S^{K+1} .

При преобразовании кортежа подграфов $K_F = \langle F_0, F_1, \dots, F_{q-1} \rangle$ в кадр $C = \langle R, Q, W \rangle$ множества входных информационных вершин X_m заменяются функцией чтения $r(t)$, а множества выходных информационных вершин Y_m заменяются функцией записи $w(t)$. Функция чтения $r(t)$ ставит в соответствие моменту t_m множество информационных вершин X_m . Аналогично, функция записи $w(t)$ ставит в соответствие моменту t_m множество выходных информационных вершин Y_m . Компонент Q кадра представляет собой граф, вершины и дуги которого соответствуют операционным вершинам и операционным дугам подграфа F_m .

В соответствии с принятым порядком следования подграфов F_m в кортеже K_F выражение для значений функции чтения $r(t_m)$ имеет вид

$$r(t_m) = \{f_{r+p,s}\}, U_{r-1,s}^{k-1}, \{U_{r+p,s-1}^{k-1}\}, \{U_{r+p,s}^{k-1}\}, \{U_{r+p,s+1}^{k-1}\}, U_{r+L,s}^{k-1}.$$

Таким образом, в каждом такте t_m для параллельного вычисления множества значений функции $\{U_{r+p,s}^k\}$ на входы L -узлов процессора требуется параллельно подать $4L+2$ данных. Если все данные в такте t_m должны поступать на обработку непосредственно из каналов памяти, то минимальное количество каналов памяти будет равно $4L+2$.

Если количество каналов распределенной памяти ограничено, то может возникнуть ситуация, когда каналов памяти будет недостаточно для того, чтобы задействовать все процессоры реконфигурируемой структуры. В этом случае будет снижена эффективность решения задачи. В то же время для данной

задачи существует возможность уменьшения числа каналов распределенной памяти.

Разобьем вычислительный фрагмент P^k на множество однотипных фрагментов p_i^k . Фрагмент p_i^k соответствует обработке узлов сетки, размещенных в столбцах с первого по $(M-1)$ -й. В этом случае кадр $C\langle R, Q, W \rangle$ будет разрезан на $(N-1)/L$ кадров C_i . Функция чтения r_i в кадре C_i представляет собой функцию чтения исходного кадра C , определенную на интервале $T_i = [t_a, t_b]$, где $a = (r-1)(M-1)$, $b = r(m-1)-1$. Аналогично, функция записи $w_i(t)$ в кадре C_r представляет собой функцию записи $w(t)$ исходного кадра C , определенную на интервале T_r .

Для некоторого множества $\{H_i\}, i = 1, 2, \dots, N_H$ введем функцию $Z(H_i)$, которая ставит в соответствие элементу H_i элемент H_{i-1} за исключением H_1 :

$$Z(H_i) = \begin{cases} H_{i-1}, & \text{если } i > 1, \\ G, & \text{если } i = 1. \end{cases} \quad (3.9)$$

На основе введенной функции определим композицию $Z(Z(H_i)) = Z^2(H_i)$:

$$Z^2(H_i) = \begin{cases} H_{i-2}, & \text{если } i > 2, \\ G_1, & \text{если } i = 2, \\ G_2, & \text{если } i = 1. \end{cases} \quad (3.10)$$

Аналогичным образом можно определить функции $Z(H_i)$ и $Z^2(H_i)$, аргументом которых является вектор H_i .

Обозначим $U_{r,s+1}^{k-1} = \{U_{r+p,s+1}^{k-1}\}$ и $f_{r,s} = \{f_{r+p,s}\}$. Тогда значения функции $r(t_m)$ можно записать следующим образом:

$$r(t_m) = \{f_{r,s}, Z(U_{r-1,s+1}^{k-1}), Z^2(U_{r,s+1}^{k-1}), Z(U_{r,s+1}^{k-1}), U_{r,s+1}^{k-1}, Z(U_{r+L,s+1}^{k-1})\}.$$

Отсюда видно, что нет необходимости задействовать $4L+2$ канала для реализации функции чтения. В такте t_m достаточно параллельно прочитать из памяти данные $f_{r,s}$ и $U_{r,s+1}^{k-1}$, а также данные $U_{r-1,s+1}^{k-1}$ и $U_{r+L,s+1}^{k-1}$. Поступление на входы L -узлов процессора данных $U_{r,s}^{k-1}$ и $U_{r,s-1}^{k-1}$ осуществляется с помощью буферной памяти, реализующей функции Z и Z^2 . Таким образом, для

вычисления значений сеточной функции $U_{r,s}^{k-1}$ в такте t_m данные $U_{r,s+1}^{k-1}$ подаются на входы L -узлов процессора непосредственно из каналов памяти. Данные $U_{r,s}^{k-1}$ и $U_{r,s-1}^{k-1}$ подаются из буферной памяти, в которую они были записаны в тактах t_{m-1} и t_{m-2} , соответственно.

Одним из вариантов реализации буферной памяти является использование элементов задержки на один операнд. Задержка на два операнда может быть реализована путем последовательного соединения двух элементов задержки на один операнд. В этом случае данные $U_{r,s}^{k-1}$ подаются на входы L -узлов процессора с задержкой на один операнд, а данные $U_{r,s-1}^{k-1}$ — с задержкой на два операнда. Поскольку данные $U_{r-1,s}^{k-1}$ и $U_{r+L,s}^{k-1}$ должны поступать на входы L -узлов процессора одновременно с данными $\{U_{r+p,s}^k\}$, то они следуют с задержкой на один операнд. Для реализации подобного поступления данных требуются две векторные операции задержки $z1$ и $z2$, а также две операции задержки $z3$ и $z4$.

Особым является случай, когда необходимо вычислить значения сеточной функции в первом столбце $U_{r,l}^{k-1}$. В этом случае в элементы задержки $z1$ и $z2$ должны быть предварительно занесены данные $U_{r,0}^{k-1}$ и $U_{r,l}^{k-1}$, а в элементы $z3$ и $z4$ — соответственно, данные $U_{r-1,1}^{k-1}$ и $U_{r+L,1}^{k-1}$. Тогда при вычислении значений $U_{r,1}^k$ данные $U_{r,0}^{k-1}$ и $U_{r,l}^{k-1}$ поступят на входы L -узлов процессора из элементов задержки $z1$ и $z2$, данные $U_{r-1,1}^{k-1}$ и $U_{r+L,1}^{k-1}$ — соответственно из элементов задержки $z3$ и $z4$, а данные $U_{r,2}^{k-1}$ — из каналов памяти. Таким образом, при введении элементов задержки функция чтения в кадрах модифицируется. На рис. 3.19 показана граф-схема L -узлов процессора, содержащего операции задержки z .

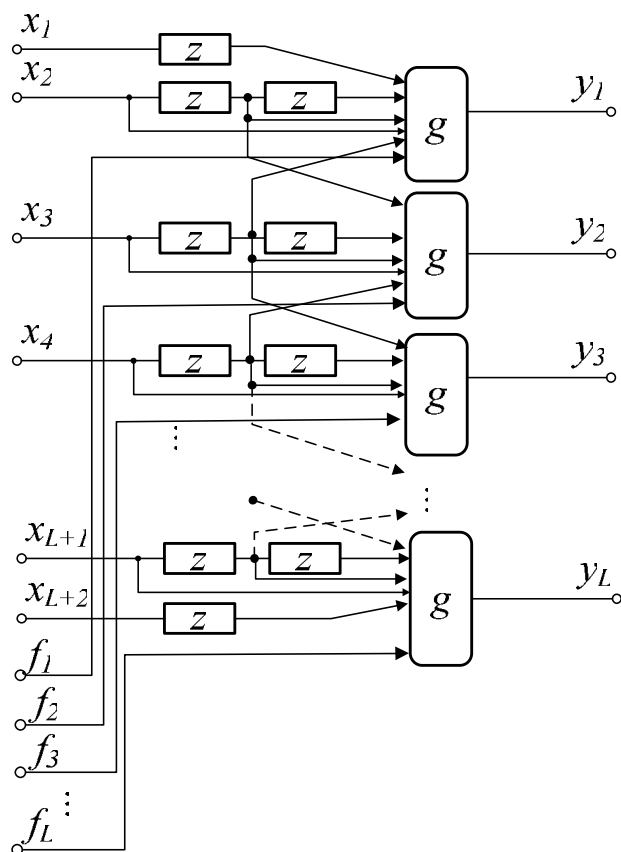


Рис. 3.19. Граф-схема L -узлового процессора

Для того чтобы обеспечить параллельное поступление данных на входы L -узлового процессора с элементами задержки на операнд, минимальное количество каналов памяти должно быть $2L + 2$.

При этом в $L + 2$ каналах памяти записаны значения сеточной функции U , а в других L каналах — значения функции f . Результаты вычислений параллельно записываются в L каналов памяти. Однако для реализации функции чтения один и тот же вход L -узлового процессора необходимо подключать к различным каналам памяти. Количество вариантов подключения определяется как наименьший общий делитель от количества каналов памяти, выделенных для хранения значений сеточной функции U , и числа узловых процессоров.

Таким образом, увеличение числа каналов для размещения данных $U_{i,j}^{k-1}$ до $2L$ позволяет использовать только две коммутации. Всего при реализации данного варианта распараллеливания требуется $3L$ каналов распределенной

памяти. При формировании структуры данных целесообразно разместить элементы $U_{i,j}^{k-1}$, имеющие одинаковый индекс j , в одном канале памяти. В этом случае не требуется менять коммутацию при чтении из каналов памяти элементов одной строки.

В соответствии с функцией чтения данные читаются из памяти в порядке возрастания индекса i , поэтому для упрощения процедуры адресации элементы строки сетки должны быть размещены в канале памяти в естественном порядке. Естественный порядок размещения данных позволяет получить простые рекуррентные формулы для вычисления последовательности адресов чтения и записи данных.

Формулы (3.9) и (3.10) дают возможность перейти к двумерной адресации данных $\langle N_K, A_C \rangle$ в каналах распределенной памяти. В данном случае объектом, размещаемым в каналах распределенной памяти, является строка сеточной области, включающая в себя $(M+1)$ элементов. Переменная x соответствует номеру строки i , а переменные y_1 и y_2 - индексам N_K и A_C соответственно. Строки сеточной области размещаются в $2L$ каналах распределенной памяти.

Выбор базового смещения x_0 определяется номером канала, начиная с которого размещаются строки сетки. Если размещение строк сетки начинать с канала N_K^0 , то базовое смещение равно нулю. Тогда компоненты кортежа $\langle N_K, A_C \rangle$, соответствующего данному $U_{i,j}^{k-1}$, можно определить по формулам

$$A_C = (A_C^0 + j) + (M+1) \left\lceil \frac{i}{2L} \right\rceil, \quad (3.11)$$

$$N_K = N_K^0 + \text{mod}_{2L}(i).$$

Структура данных позволяет легко определить порядок подключения каналов памяти к входам L -узлового процессора в кадрах C_r в соответствии с размещением элементов сеточной функции U в каналах распределенной памяти. При выполнении кадров C_i , где $i = 2[(r-1)/2L]$, в такте t_m , принадлежащем интервалу T_i , на первые $L+2$ информационных входа L -узлового процессора требуется параллельно подать множество данных

$$\langle U_{r-1,s+1}^{k-1}, U_{r,s+1}^{k-1}, U_{r+L,s+1}^{k-1} \rangle.$$

При этом $r = (i-1)L + 1$. Здесь и далее положение элементов в кортеже соответствует номеру входа L -узлового процессора. Из формулы (3.11) можно получить номера каналов чтения, которые подключаются к входам L -узлового процессора в этих кадрах, а также номера каналов записи, подключенные к выходам L -узлового процессора.

Таким образом, первый вход L -узлового процессора подключен к каналу с номером N_C^0 , из которого поступают данные $U_{r-1,s+1}^{k-1}$. Следующие L входов L -узлового процессора подключены к каналам с номером $N_C^0 + j$, где $j = 0, 1, 2, \dots, L-1$. Последний вход подключен к каналу памяти с номером $N_C^0 + L$. Вычисленные значения $U_{r,s}^k$ записываются в каналы с номерами $N_C^0 + j + 1$. Аналогичным образом определяются номера каналов распределенной памяти, которые подключаются к входам L -узлового процессора в кадрах C_i для нечетных значений i , при этом $r = (i-1)L + L + 1$. На рис. 3.20 показаны кадры C_2 и C_4 , в которых используется описанная выше схема структурно-процедурных вычислений.

Параметры циклов чтения (записи) находятся в простой функциональной зависимости от числа узловых процессоров L и от размеров сеточной области M и N , что позволяет их легко модифицировать. Программная реализация процедуры вычисления значений сеточной функции представляет собой цикл по индексу s , в теле которого расположены четыре кадра $K1 - K4$.

В кадрах $K1$ и $K3$ элементы задержки z инициализируются начальными значениями, необходимыми для вычисления значения сеточной функции. В кадрах $K2$ и $K4$ вычисляются значения сеточной функции для индексов $s = 2, 3, \dots, M-1$. Одновременно с расчетом значений сеточной функции формируется предикат P_{out} в соответствии с условием (3.10) и записывается в отдельный канал памяти. После выхода из цикла выполняется условный переход по значению предиката P_{out} .

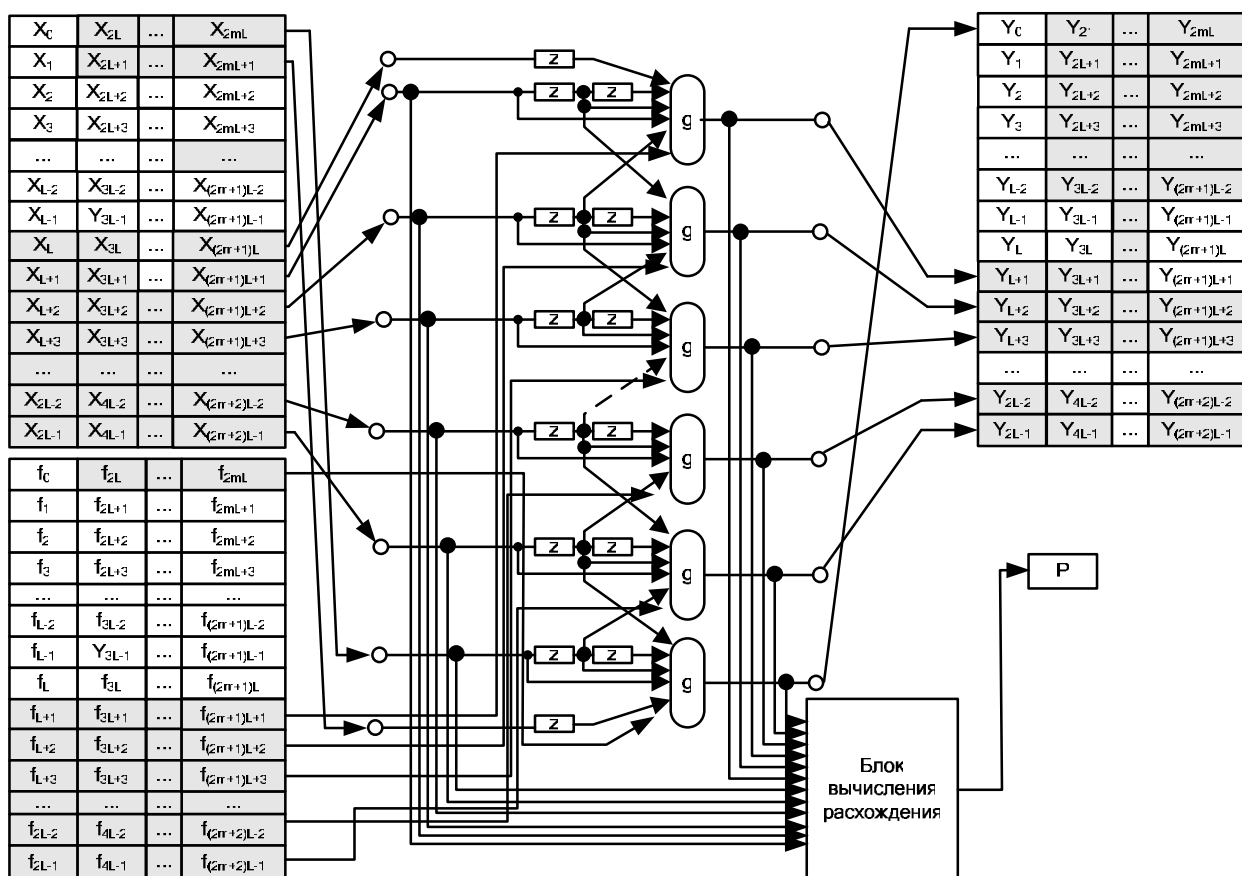


Рис. 3.20. Структурно-процедурные вычисления кадров C_2 и C_4

Для данной задачи возможны и другие варианты организации вычислений, которые определяются способом разрезки информационного графа на подграфы. При этом кадровая форма задачи будет определяться с учетом конкретных особенностей архитектуры реконфигурируемых структур, в том числе, количеством каналов памяти, топологией коммутационной системы, количеством элементарных процессоров и способами организации адресных вычислений.

Структурно-процедурная организация вычислений обеспечивает высокую эффективность решения задачи на реконфигурируемых структурах, поскольку передача данных между компонентами вычислительной системы совмещена с процессом обработки данных на множестве узловых процессоров, а кадровая форма задачи позволяет организовать вычислительный конвейер, глубина которого определяется количеством обрабатываемых данных.

Время решения задачи на РМВС можно оценить следующей формулой:

$$T_p \approx N_{it} \left\{ M[(N-2)/L] t_{op} + 2(t_{pipeline} + t_{setup}) \right\},$$

где N_{it} – количество итераций;

t_{op} – время обработки одного операнда;

$t_{pipeline}$ – задержка конвейера;

t_{setup} – время настройки компонентов вычислительной системы.

При увеличении размеров сеточной области доля времени, определяемая $t_{pipeline}$ и t_{setup} , уменьшается, а эффективность приближается к единице. Таким образом, гарантируется близкий к линейному рост реальной производительности реконфигурируемых структур при увеличении числа параллельно функционирующих узловых процессоров. Рассмотренный пример реализации задачи на реконфигурируемых структурах является достаточно простым, однако он в полной мере демонстрирует возможности технологии преобразования задач в структурно-процедурную форму решения. Технология структурно-процедурного программирования позволяет без семантического разрыва выполнить последовательность преобразований от исходного представления задачи до ее параллельной программы и оперативно разработать эффективные параллельные программы решения широкого класса задач различных проблемных областей.

Полученные результаты можно расширить на случай, когда информационный граф задачи состоит из нескольких подграфов $G_1, G_2, \dots, G_n$, каждый из которых является функционально-регулярным, при этом подграфы $G_1, G_2, \dots, G_n$ не изоморфны друг другу. В этом случае каждый подграф $G_1, G_2, \dots, G_n$ будет преобразован в отдельный кадр.

Следует отметить, что функции чтение/запись могут быть более сложными, чем в предыдущем случае. Например, коммутация каналов распределенной памяти и структурно-распределенного графа может не оставаться неизменной. Структурно-процедурные алгоритмы решения задач математической физики подробно описаны в работах [52-54, 70-73].

Ниже рассмотрен процесс преобразования в кадровую форму более сложной задачи – процедуры свертки в частотной области с использованием алгоритма быстрого преобразования Фурье (БПФ) [74, 100].

3.6. Преобразование в структурно-процедурную форму информационных графов нерегулярной структуры

Можно выделить несколько основных типов информационных графов, для отображения которых в структурно-процедурную форму требуется выполнение специальных информационно-эквивалентных преобразований. К таким графам следует отнести информационные графы, которые хоть и имеют детерминированную структуру и могут быть представлены в виде кортежа информационно-зависимых подграфов, но функциональная зависимость между базовыми подграфами явно не выражена. Этот тип информационных графов характерен тем, что количество слоев в подграфе заранее определено, однако каждый слой содержит разное количество базовых подграфов, при этом существует правило отображения базовых подграфов внутри каждого слоя, а правило отображения слоев либо является слишком общим, либо определяется для каждого конкретного варианта данной задачи. Второй тип графа представляют собой задачи, у которых количество базовых подграфов, принадлежащих слою, одинаково, но правила отображения базовых подграфов внутри каждого слоя различны, а правило отображения слоев определяется конкретной реализацией данной задачи. Кроме того, существуют задачи, графы вычислительного процесса решения которых содержат различное и заранее неизвестное количество базовых подграфов, принадлежащих слою. При этом мощность слоя, содержащего базовые подграфы, зависит от результатов вычислений на предыдущих слоях графа задачи.

Тривиальным решением структурно-процедурной организации вычислений для вышеперечисленных типов графов является построение параллельного информационного графа, имеющего прямоугольную ярусно-

параллельную форму, с последующим преобразованием этого графа в кадровую форму. Однако данный путь неэффективен, поскольку такие графы вычислительного процесса решения задачи обладают функциональной избыточностью: большинство базовых подграфов соответствует «пустым» операциям, когда результаты выполнения таких операций тождественно равны ее входным данным [77].

Если процедура секвенции не позволяет сразу получить функционально-регулярные графы, обеспечивающие рациональную параллельную программу, то для таких графов возможно провести специальные информационно-эквивалентные преобразования и получить структурно-процедурную форму в виде последовательности кадров. При этом возможно обеспечить вычислительный конвейер и получить достаточно высокую эффективность решения задач на реконфигурируемых структурах.

Для нерегулярных задач алгоритм представляется в абсолютно-параллельной форме - в виде графа вычислительного процесса. Затем устраняется функциональная избыточность графа вычислительного процесса и формируется промежуточный функционально-нерегулярный граф. И в завершение предлагается выполнить специальное преобразование - векторизацию информационных вершин в графе [78]. Это преобразование существенно отличается от векторизации циклов и процедур [20,60,102,137]. При выполнении данной операции множество непересекающихся подмножеств информационных вершин, в каждом из которых существует ациклический путь, соединяющий в определенном порядке информационные вершины, или другое правило детерминированного упорядочивания вершин заменяется специальным объектом – *вершиной-массивом*. Вершина соответствует расположению структуры данных в каналах распределенной памяти вычислительной системы. Каждое подмножество информационных вершин, соединенных ациклическим путем в информационном графе или отобранное другим правилом упорядочивания, будет расположено в отдельном канале распределенной памяти. Адреса, по которым будут расположены вершины в

канале, взаимно однозначно соответствуют номеру вершины на пути, соединяющему эти вершины в графе, или номеру в кортеже упорядочивания. После выполнения операции векторизации информационные дуги, источниками или приемниками которых являлись векторизированные информационные вершины, исключаются и заменяются дугами, источником или приемником которых является вершина-массив.

В векторизованном графе появляются адресные дуги, которые формулируются по следующему правилу: началом адресной дуги является информационная адресная вершина с уникальным именем, соответствующим адресу информационной вершины, а концом - метка информационной дуги, выходящей (входящей) из вершины-массива. Адрес соответствует номеру канала распределенной памяти, то есть номеру упорядоченного в кортеж подмножества информационных вершин.

Каждая адресная дуга имеет признак t , который соответствует временной метке обращения по адресу ячейки в канале распределенной памяти.

Пусть V_i - множество информационных вершин графа P_i , представленного в виде кортежа слоев $\langle S_{i,1}, S_{i,2}, \dots, S_{i,k} \rangle$ информационно-зависимых подграфов

$$S_{i,j} = \langle (Vx_{i,j}, Ax_{i,j}), (Vq_{i,j}, Aq_{i,j}), (Vy_{i,j}, Ay_{i,j}) \rangle,$$

где $Vx_{i,j}$ - множество входных вершин;

$Ax_{i,j}$ - множество входных дуг;

$Vq_{i,j}$ - множество операционных вершин;

$Aq_{i,j}$ - множество дуг, соединяющих операционные вершины базового подграфа $q_{i,j}$;

$Vy_{i,j}$ - множество выходных вершин;

$Ay_{i,j}$ - множество выходных дуг.

Множество V_i может быть представлено в виде объединения двух непересекающихся подмножеств VX_i и VY_i :

$$VX_i = Vx_{i,1} \cup Vx_{i,2} \cup Vx_{i,3}, \dots, \cup Vx_{i,K},$$

$$VY_i = Vy_{i,1} \cup Vy_{i,2} \cup Vy_{i,3}, \dots, \cup Vy_{i,K}.$$

При выполнении процедуры векторизации множество информационных вершин VX_i заменяется вершиной-массивом W_x , а множество информационных вершин VY_i - вершиной-массивом W_y . Из вершины W_x выходит множество дуг AX . Каждая дуга $a \in AX$ имеет следующее описание:

$$a = \langle W_x, \langle F(x), t \rangle, b, p \rangle,$$

где W_x - имя вершины-массива;

$\langle F(x), t \rangle$ - пара признаков;

b - имя операционной вершины;

p - номер выхода.

Признак $F(x)$ является адресным признаком, однозначно соответствующим имени вершины $x \in VX_i$, а t - временная метка, которая определяет принадлежность информационной вершины x подграфу $S_{i,t}$. Аналогично можно определить дуги AY , входящие в вершину W_y :

$$a = \langle b, p, W_y, \langle F(y), t \rangle \rangle.$$

В ряде случаев достаточно использования однократной векторизации информационных вершин для того, чтобы получить функционально регулярный граф и, тем самым, выделить правила отображения F и Φ .

На рис. 3.21 приведен пример W -графа, полученного при выполнении процедуры первичной векторизации информационных вершин для функционально-нерегулярного графа формирования гистограммы.

Данная процедура широко используется при решении многих задач обработки сигналов и изображений. Здесь операционная вершина S соответствует операции инкремента. После преобразования информационного избыточного графа операция над данными производится по адресу элемента массива гистограмм. Адресом служит функция $F(x_i)$, которая определяет: к

какому элементу гистограммы относится пришедший операнд x_i . Полученные результаты записываются в промежуточную вершину-массив WX .

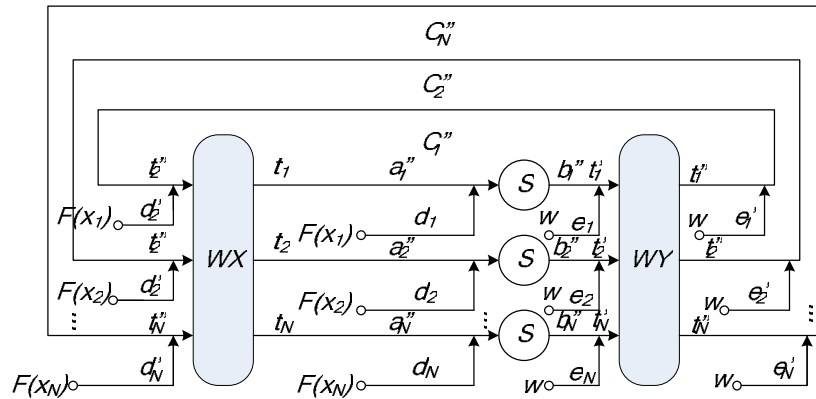


Рис. 3.21. Пример W -графа, полученного при векторизации вершин

При решении ряда задач, например задачи трассировки [87], применения однократной векторизации информационных вершин недостаточно, поэтому необходимо использовать процедуру вторичной векторизации. В отличие от первичной векторизации здесь в вершину-массив преобразуются адресные признаки $F(x)$ и $F(y)$, которые можно рассматривать как информационные адресные вершины.

На рис. 3.22 показан W -граф, который получен в результате проведения вторичной векторизации.

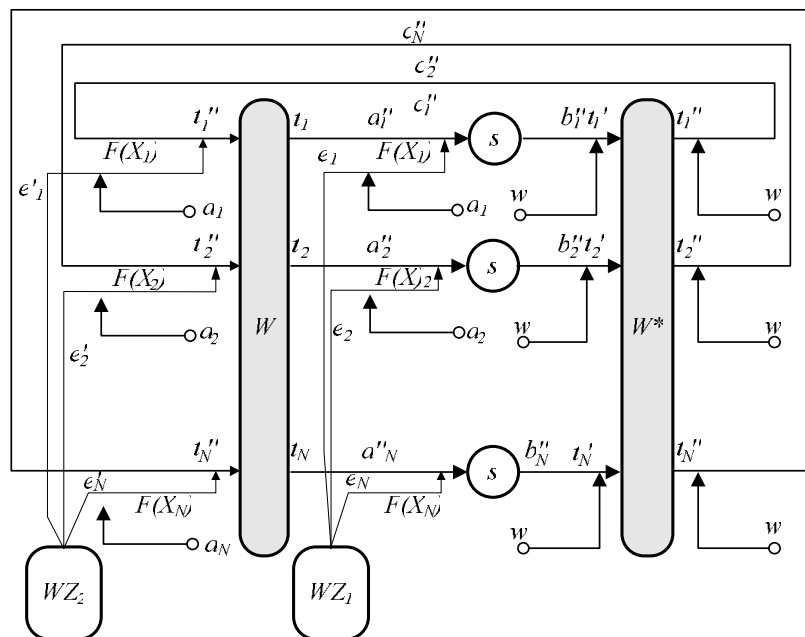


Рис. 3.22. Вторичная векторизация адресных вершин в W -графе

Здесь WZ_1 и WZ_2 - адресные вершины-массивы, которые получены путем векторизации первичных адресных вершин. В результате вторичной векторизации адресных вершин возможно получить функционально-регулярный граф, в котором правила отображения дуг подграфов определены через правила отображения адресных вершин $a_i \rightarrow a_{i+1}$.

Операция векторизации информационных вершин позволяет представить задачу в виде функционально-регулярных информационных подграфов, в которых правила отображения базовых подграфов друг на друга определены через правила отображения адресных вершин. Данные информационные подграфы преобразуются в кадры, реализуемые для произвольной степени распараллеливания. Особенность формирования бесконфликтной структуры данных и функции адресации и коммутации для подобных задач заключается в том, что процедура векторизации информационных вершин, за исключением некоторых особых случаев, предполагает использование косвенной адресации.

Таким образом, для ряда сложных задач, информационные графы которых имеют нерегулярную структуру, использование процедуры векторизации информационных и адресных вершин позволяет получить эффективную структурно-процедурную реализацию. Примером подобных задач является задача волновой трассировки, рассмотренная в [75,79].

3.7. Структурно-процедурная реализация задачи трассировки

Трассировка соединений элементов сверхбольших интегральных схем (СБИС) является одним из важнейших этапов проектирования микроэлектронного изделия. От качества трассировки существенно зависят параметры разрабатываемого устройства. Современные микроэлектронные технологии позволяют создавать СБИС со степенью интеграции свыше 10^8 элементов на кристалле. Конкуренция на рынке микроэлектронных технологий требует сокращения сроков разработки, при этом временные затраты на разработку топологии СБИС становятся сравнимыми со временем, необходимым на подготовку производства. Поэтому актуальным является

внедрение средств высокопроизводительной вычислительной техники для сокращения сроков разработки топологии СБИС.

В последние годы большой интерес разработчиков СБИС проявляется к параллельным вычислительным системам, потенциальная производительность которых на несколько порядков превосходит производительность однопроцессорных рабочих станций. Ведутся активные исследования в направлении создания параллельных алгоритмов решения задач САПР СБИС на многопроцессорных системах различных архитектур.

В то же время структура задачи трассировки такова, что при ее реализации на многопроцессорных системах с распределенной памятью требуется интенсивный обмен между процессорами. При этом оказывается, что время, необходимое для организации параллельных вычислений, существенно превосходит время вычислений. Вследствие этого отношение реальной производительности системы при решении задачи трассировки к пиковой производительности системы составляет всего 0,05-0,1 [73,142]. Альтернативой известным параллельным алгоритмам является структурно-процедурная организация вычислений, которая позволяет существенно повысить эффективность распараллеливания задачи трассировки.

В настоящее время существуют различные алгоритмы трассировки, наиболее универсальным из которых является *волновой алгоритм* Ли [146]. По сравнению с другими, более скоростными и экономичными алгоритмами, такими как лучевой и канальный, алгоритм Ли имеет два серьезных преимущества. Во-первых, он позволяет найти кратчайшую трассу между соединяемыми элементами, если таковая существует. Во-вторых, обеспечивает разводку максимально возможного количества трасс. Учитывая резерв производительности многопроцессорных систем, реализация параллельного волнового алгоритма трассировки оправдана.

Решение задачи трассировки с использованием волнового алгоритма предполагает два этапа. На первом осуществляется поиск возможных решений. На втором - выбирается решение, удовлетворяющее условиям поставленной

задачи. Наиболее трудоемким и сложным для распараллеливания является процесс распространения волны, поэтому остановимся на этом этапе алгоритма.

Модель кристалла СБИС для трассировки представляется множеством дискретов D . Каждому дискрету $d_{i,j}$ из D присвоен весовой коэффициент $W_{i,j}$, определяющий особенности расположения дискрета на модели поля. Если дискрет свободен, то $W_{i,j} = 0$. Если дискрет занят, то ему соответствует отличное от нуля число. В начальный момент времени все элементы $W_{i,j}$ обнуляются.

Источником трассы является точка с координатами $\langle i_b, j_b \rangle$. Вокруг источника трассы для смежных элементов формируется волна при условии, что смежные элементы свободны. В этом случае соответствующие элементы массива W получают значение единицы. Далее для элементов массива W , которые изменили свое значение, исследуются смежные элементы, и если такие элементы свободны, им присваивается номер текущего волнового фронта. Процесс продолжается до тех пор, пока волна не распространится до конечной точки трассы, определяемой координатами $\langle i_e, j_e \rangle$. В общем случае на k -й итерации формируется волновой фронт, определяемый множеством пар координат на дискретном поле. Здесь и далее считается, что шаг дискретизации постоянный по всей области трассировки. Это позволяет использовать масштабированное единичное приращение координат.

Обозначим B_k – множество координат дискрет, принадлежащих k -му волновому фронту. Введем некоторое отношение $Z(a,b)$, указывающее, что элемент a является смежным с элементом b . В простейшем случае дискреты с координатами (X_i, Y_i) и (X_j, Y_j) будем называть *смежными*, если для них выполняется условие $|X_i - X_j| + |Y_i - Y_j| = 1$.

Элементы множества A_k – координат дискрет, смежных с элементами множества B_k , определяются следующей реляционной формулой:

$$A_k = \{f \mid Z(f, b) \wedge B_{k-1}(af)\}.$$

Здесь и далее под $B(a)$ понимается принадлежность элемента a , представляющего собой пару координат $\langle x, y \rangle$, множеству B . Множество элементов следующего волнового фронта B_k определяется по формуле

$$B_k = \{f \mid (W_f = 0) \wedge A_k(f)\}.$$

Элементам массива W , номера которых принадлежат B_{k+1} , присваивается значение $k+1$. Основная идея организации параллельных вычислений для данной задачи заключается в том, что на каждом шаге алгоритма все элементы множества A_k могут быть вычислены одновременно и независимо друг от друга. Таким же образом параллельно могут быть вычислены все значения координат, принадлежащих множеству B_k .

Информационный граф процедуры распространения числовой волны представляет собой кортеж информационных слоев $S = \langle S_0, S_1, \dots, S_M \rangle$. Слой S_k соответствует процедуре формирования элементов k -го волнового фронта. Слой может быть представлен в виде множества изоморфных информационных базовых подграфов g , каждый из которых соответствует операции формирования значений $W_{i,j}^k$ в соответствии с выражением

$$W_{i,j}^k = (k+1) \& P\{W_{i,j}^{k-1} = 0\} \vee W_{i,j}^{k-1} \& P\{W_{i,j}^{k-1} \neq 0\},$$

где $W_{i,j}^k$ - значение элемента массива W на k -й итерации;

$P\{.\}$ – обозначение предиката.

Базовые подграфы реализуются не для всего массива W , а только для элементов $\langle i, j \rangle \in A_k$, поэтому количество базовых подграфов слоя S_k и топология связей между входными и выходными информационными вершинами слоя определяются значениями массива W на предыдущей итерации. Немодернизированные элементы массива W переписываются с предыдущей итерации, что соответствует дуге, соединяющей информационные вершины $W_{i,j}^{k-1}$ и $W_{i,j}^k$. Вследствие этого информационный граф алгоритма является функционально-нерегулярным, поскольку в общем случае не

существует единого правила $\Phi: S_{k-1} \rightarrow S_k$ отражения множества слоев друг на друга.

В то же время для эффективной реализации вычислений целесообразно уменьшить количество кадров, чтобы обеспечить большой поток данных, следующий через конвейер вычислительной структуры. Для этой цели необходимо преобразовать граф задачи в функционально-регулярную форму с помощью ряда информационно-эквивалентных преобразований.

На рис. 3.23 приведен фрагмент информационного графа решения задачи. Здесь для простоты восприятия показаны смежные вершины только по одной координате.

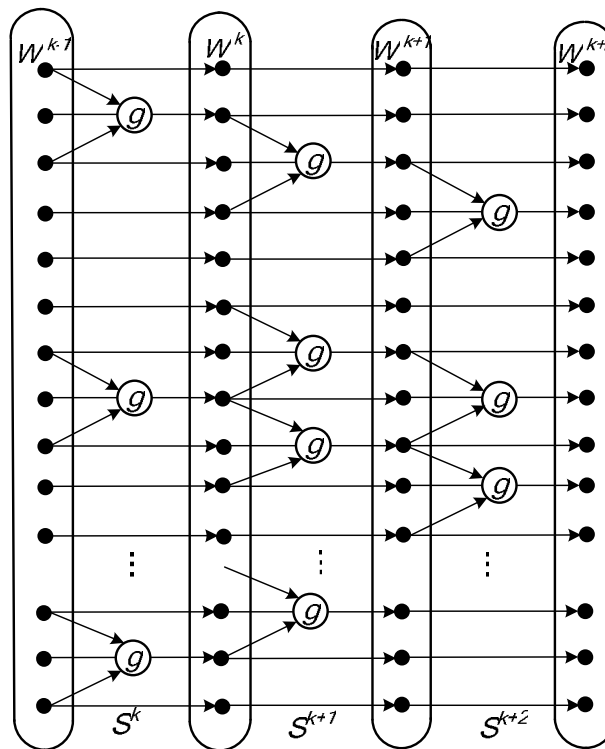


Рис. 3.23. Фрагмент информационного графа решения задачи

Одним из основных методов приведения графа задачи к функционально-регулярному виду является операция векторизации. Для упорядочивания информационных вершин в подмножестве используется путь, соединяющий вершины монтажного поля $W_{i,j}^k$ с одинаковыми координатами $\langle i, j \rangle$ различных слоев k .

На рис. 3.24 представлен векторизованный подграф задачи, соответствующий построению k -го волнового фронта. Каждый подграф g_i содержит несколько подграфов f , соответствующих смещению элемента волнового фронта в соответствии с отношением смежности z по нескольким направлениям. В простейшем случае направления будет четыре. Из множества B^{k-1} формируется множество A^k , элементы которого являются адресом для вершины-массива коэффициентов рабочего поля W . Подграфы g_2 формируют элементы нового волнового фронта B^k . Для каждого нового волнового фронта строится аналогичный граф.

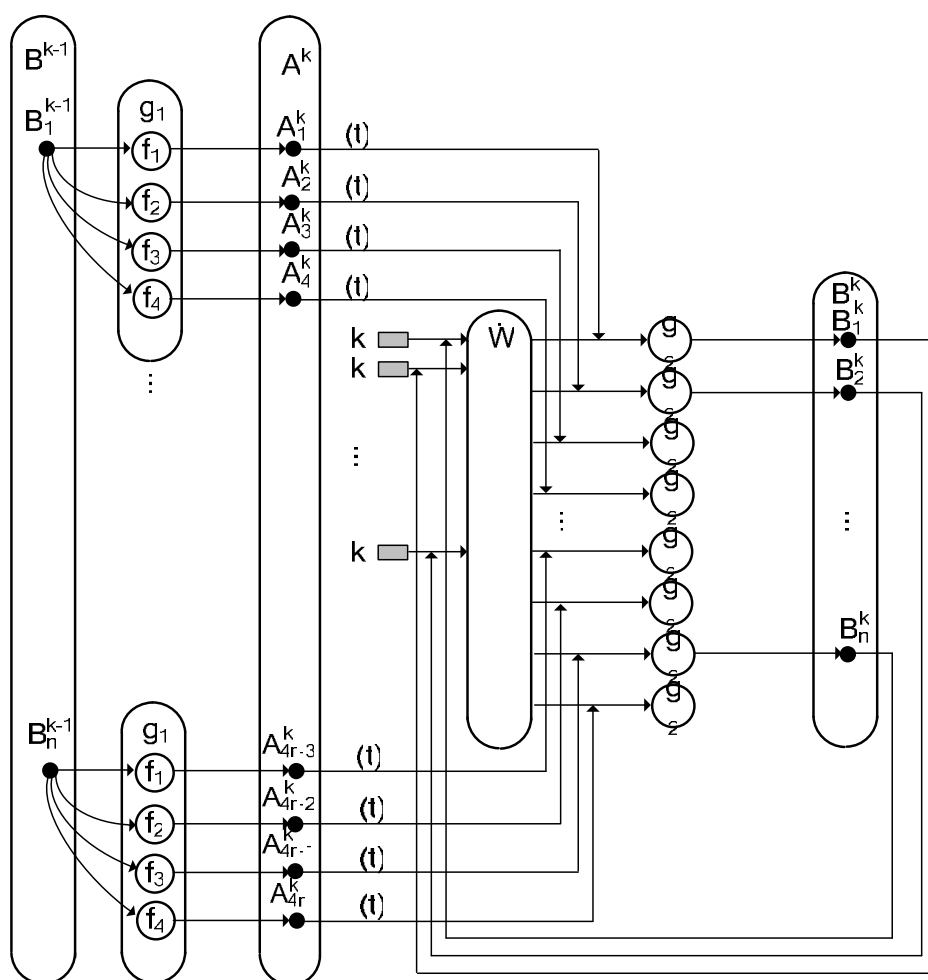


Рис. 3.24. Векторизованный граф задачи

После первичной векторизации граф задачи волновой трассировки остается функционально нерегулярным, так как отсутствует правило упорядочивания адресных вершин для различных волновых фронтов, которые

определяются данными задачи. Проведение операции вторичной векторизации позволяет привести граф задачи в функционально регулярную форму.

При дополнительной векторизации в векторизованном графе множество подмножеств информационных адресных вершин B_k заменяется вершиной-массивом B . Каждой информационной дуге ставится в соответствие адресная дуга, источником которой является информационная адресная вершина M_B . Для упорядочивания элементов подмножеств адресных вершин использовались временные метки. В один канал распределенной памяти могут быть отобраны только такие адресные вершины, временные метки которых различны.

Векторизованный информационный граф, полученный в результате выполнения операций векторизации адресных вершин, является функционально-регулярным, так как для всех слоев данного графа существует единое правило отображение вершин друг на друга, не зависящее от результатов преобразования информации.

Непосредственная структурно-процедурная реализация полученного векторизованного графа в виде одного кадра невозможна, поскольку при такой реализации вычислений потребуется одновременно обращаться к одним и тем же каналам распределенной памяти по разным адресам для чтения и записи. Функция чтения и функция записи ставят одному моменту времени входное данное и результат вычислений. Поэтому процедура построения волнового фронта реализуется в виде двух кадров. В первом кадре по координатам элементов волнового фронта B_k вычисляются координаты смежных элементов A_k , производится чтение значений массива W и формируются координаты нового волнового фронта B_{k+1} . Во втором кадре элементам массива W , адреса которых принадлежат B_{k+1} , присваивается вес $k+1$. В начальный момент времени вершина-массив содержит один элемент, соответствующий начальной координате трассы Z_1^1 . Данный элемент находится по адресу PI_1 .

После считывания элемента Z_1^1 с помощью структурно реализованного подграфа g_1 , состоящего из четырех подграфов f_1, f_2, f_3, f_4 , каждый из которых соответствует определенному направлению смежности из начальной координаты, формируются четыре координаты предполагаемого второго волнового фронта $q_1^2, q_2^2, q_3^2, q_4^2$. Эти информационные вершины являются началом соответствующих адресных дуг, по которым производится обращение к вершине-массиву W . Считанные из вершины-массива W элементы поступают на вход структурно реализованных подграфов g_2 , и если элемент, считанный по адресу q_1^2 , равен нулю, то формируется элемент, представляющий собой реальную координату второго волнового фронта a_1^2 . В противном случае элемент волнового фронта не формируется.

Аналогично обрабатываются остальные элементы волнового фронта. Сформированные координаты волнового фронта $a_1^2, a_2^2, \dots, a_{m_2}^2$, записываются в массив Mb по адресам $d_1^2, d_2^2, \dots, d_{m_2}^2$. Кроме того, элементы волнового фронта сами являются адресами, по которым в вершину-массив W принимается константа, соответствующая второму волновому фронту. Следует отметить, что запись в вершину-массив Mb и в вершину-массив W производится в момент времени, не совпадающий с процессом чтения. Для k -го волнового фронта считываются из вершины-массива координаты волнового фронта $d_1^k, d_2^k, \dots, d_{m_k}^k$, которые с помощью массива подграфов f_1, f_2, f_3, f_4 преобразуются в координаты следующего волнового фронта $q_1^{k+1}, q_2^{k+1}, \dots, q_{4m_k}^{k+1}$.

Элементы волнового фронта с помощью множества структурно-реализованных подграфов g_2 модернизируются в элементы волнового фронта $a_1^{k+1}, a_2^{k+1}, \dots, a_{n_{k+1}}^{k+1}$. По адресам этих элементов производится запись в вершину-массив W . Сами элементы записываются в массив Mb . Процесс продолжается до тех пор, пока один из элементов волнового фронта достигнет конечной точки трассы.

Актуальным является не только создание параллельной программы решения задачи трассировки, но и обеспечение ее выполнения для различных

вариантов распараллеливания таким образом, чтобы задача могла быть решена на различных конфигурациях реконфигурируемых структур. При степени распараллеливания задачи L матрица коммутационного поля W может быть размещена в L каналах распределенной памяти. Каждому элементу матрицы соответствует пара декартовых координат $\langle x, y \rangle$. При размещении в каналах распределенной памяти необходимо осуществить переход из декартовой системы координат в так называемую коммутационно-мемориальную систему координат (A, K) , где A - адрес в канале памяти, а K - номер канала памяти.

Преобразования пары координат $\langle x, y \rangle$ из декартовой системы в коммутационно-мемориальную производятся по формулам

$$K = \text{mod}_{L_y} A = x + N \lfloor y / L \rfloor,$$

где N - число элементов в строке в матрице коммутационного поля,
 $\lfloor \dots \rfloor$ - операция округления в меньшую сторону.

Для хранения координат волнового фронта требуется L -канальный массив памяти B . Для того чтобы адресоваться к элементу матрицы, имеющему коммутационно-мемориальные координаты $\langle K, A \rangle$, нет необходимости хранить в массиве памяти его декартовы координаты и структурно реализовать их преобразование в коммутационно-мемориальную систему координат. В массив B достаточно записать адрес A и номер канала K .

При параллельной реализации для L элементов волнового фронта операциям приращения координат в декартовом базисе соответствуют определенные преобразования в коммутационно-мемориальном базисе координат. Приращение по координате x : $x' = x \pm 1$ соответствует аналогичному приращению адреса $A' = A \pm 1$. Положительное приращение по координате y соответствует следующим модификациям адреса и номера канала:

$$K' = K \oplus_L 1,$$

$$A' = \begin{cases} A + N, & \text{если } K = L - 1, \\ A, & \text{если } K \neq L - 1. \end{cases}$$

Отрицательное приращение по координате y соответствует следующим модификациям адреса и номера канала:

$$K' = K \ominus_L 1,$$

$$A' = \begin{cases} A - N, & \text{если } K = 0, \\ A, & \text{если } K \neq 0. \end{cases}$$

Так как степень распараллеливания может быть меньше, чем число подмножеств векторизованных информационных вершин, то в общем случае возможно возникновение конфликтов при обращении к каналам распределенной памяти. Для того чтобы одновременно не происходило чтения или записи информации по разным адресам в одном канале памяти, можно синтезировать «диспетчера», который будет препятствовать появлению такой конфликтной ситуации. Однако для реализации аппаратного диспетчера требуется больше процессоров, чем собственно для вычислений элементов волнового фронта, что приводит к существенному снижению эффективности распараллеливания.

Разрешение требований бесконфликтного обращения к распределенной памяти приводит к необходимости разделения каждого кадра процедуры распространения волнового фронта на четыре фрагмента (по количеству направлений). На рис. 3.25 направлению соответствует подграф f_i .

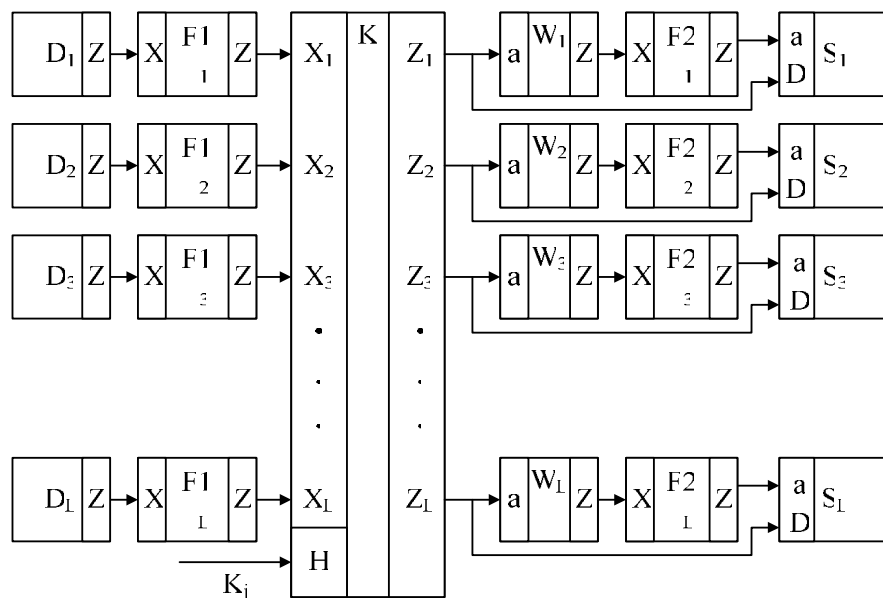


Рис. 3.25. Структурная схема кадра K_{2i}

Рассмотрим структуру кадра, в котором формируются координаты нового волнового фронта.

Здесь D_i – каналы распределенной памяти для хранения координат волнового фронта, S_i – каналы распределенной памяти для записи формируемых координат волнового фронта с номером $k+1$. Для каналов $D_i, i = 0, 1, \dots, L$ выполняется процедура чтения векторов адресов. На структурно реализованных фрагментах (макрооперации) $F1$ и коммутаторе K выполняется преобразование адресов и номеров каналов, соответствующее единичному приращению координат в прямоугольном базисе. Преобразованные адреса подаются на адресные входы каналов распределенной памяти W_i . С выходов каналов памяти, отведенных для хранения массива W , данные поступают на макрооперации $F2$, формирующей адреса записи.

Адрес записи $AW_{m,i}$ для канала S_i определяется по формуле

$$AW_{m,i} = A_m \& P\{W_{m,i} = 0\};$$

$$A_{m,i} = A_{m-1,i} + P\{W_{m,i} = 0\}.$$

По этим адресам координаты волнового фронта по входам D записываются в каналы памяти S_i . После завершения процедуры формирования волнового фронта в буферной ячейке каждой макрооперации $F2$ содержится адрес последней задействованной ячейки памяти. Вычтя из адреса последней задействованной ячейки начальный адрес, получим число занятых ячеек v_i . При выполнении кадра обработки k -го волнового фронта параметр цикла чтения, задающий количество данных, читаемых из D_i , определяется как $C_{k+1} = \max_i(v_i)$.

Более подробно структурно-процедурная организация вычислений при решении задачи трассировки рассмотрена в [79]. В этой работе проведен комплексный анализ эффективности решения задачи волновой трассировки на реконфигурируемых вычислительных структурах.

Глава 4. Реконфигурируемые мультиконвейерные вычислительные структуры на основе ПЛИС

4.1. Модульно-наращиваемая реализация реконфигурируемых мультиконвейерных вычислительных структур

Реконфигурируемые мультиконвейерные вычислительные структуры целесообразно строить по принципу модульной наращиваемости из однотипных *базовых модулей*. Базовые модули, из которых строятся реконфигурируемые структуры, с одной стороны, несут на себе конструктивные ограничения, а с другой стороны, должны в полной мере сохранить все особенности архитектуры. Поэтому требования и ограничения, оказывающие влияние на характеристики базового модуля реконфигурируемой системы, можно условно разделить на архитектурные и конструктивно-технологические.

К архитектурным требованиям относятся параметры системы, которые должны найти свое отражение в основных характеристиках базового модуля таким образом, чтобы при построении системы реализовать все отличительные особенности реконфигурируемых структур. Базовый модуль реконфигурируемых структур включает некоторое сбалансированное количество элементарных процессоров и блоков памяти, объединенных с помощью полнодоступной системы быстрых каналов и способных реализовать, желательно целиком, базовые подграфы различных задач. Исходя из требований операционной системы, базовый модуль должен представлять собой программно-неделимый аппаратный ресурс при организации вычислительного процесса в реконфигурируемых структурах.

Конструктивные решения базового модуля определяются достигнутым уровнем технологии. Параметры базового модуля зависят от многих факторов: элементной базы, которая определяет вычислительные возможности базового модуля; размеров и технологических норм печатной платы; характеристик

разъемов и соединителей для организации внешних связей; подсистемы электропитания и т.д. Все это накладывает определенные ограничения на архитектурные возможности базовых модулей. Однако, как это будет показано ниже, в современных реконфигурируемых структурах вычислительных ресурсов базовых модулей, как правило, достаточно для реализации базовых подграфов многих задач различных предметных областей [50,51].

Сформулированные архитектурные и конструктивные требования и ограничения учитываются при создании универсальных реконфигурируемых структур, которые предназначены для решения широкого круга задач. Для проблемно-ориентированных реконфигурируемых структур определяющим может стать требование обязательной реализации в пределах модуля базовых подграфов данной проблемной области. При этом базовый модуль может быть конструктивно реализован на нескольких печатных платах.

Укрупненная структура базового модуля показана на рис. 4.1.

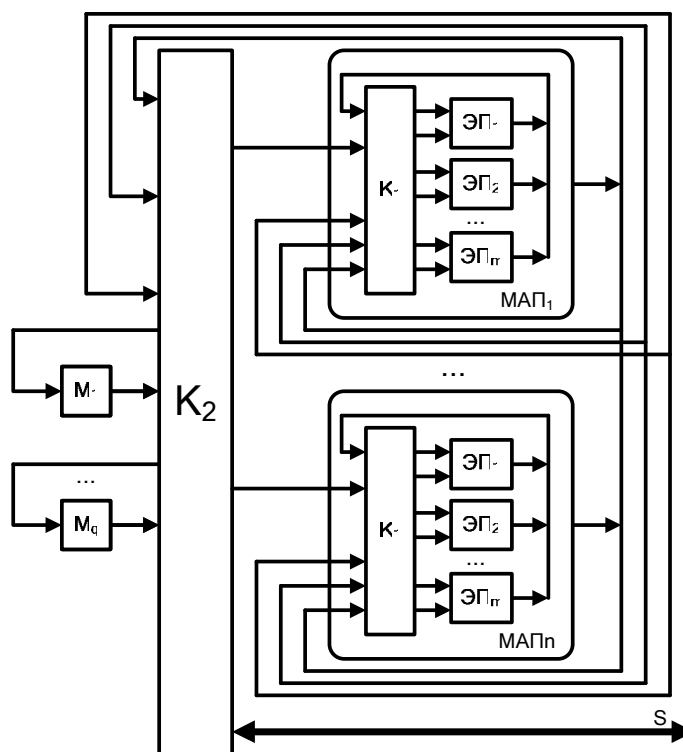


Рис. 4.1. Укрупненная структура базового модуля

В состав базового модуля (БМ) входит некоторое множество макропроцессоров (МАП) и блоков памяти (М), объединенных с помощью пространственной коммутационной системы K_2 . Каждый макропроцессор, в

свою очередь, содержит m элементарных процессоров (ЭП), объединенных коммутатором K_1 . Таким образом, в базовом модуле, показанном на рис. 4.1, имеет место двухуровневая коммутационная система. Первый уровень устанавливает связи между элементарными процессорами внутри макропроцессора, а второй уровень – связи между макропроцессорами внутри базового модуля.

Наращивание производительности путем объединения нескольких базовых модулей в вычислительный блок возможно несколькими путями. Первый из них заключается в объединении блоков и создании более производительных структур путем введения дополнительных уровней коммутации между все более крупными объединениями базовых модулей (рис.4.2).

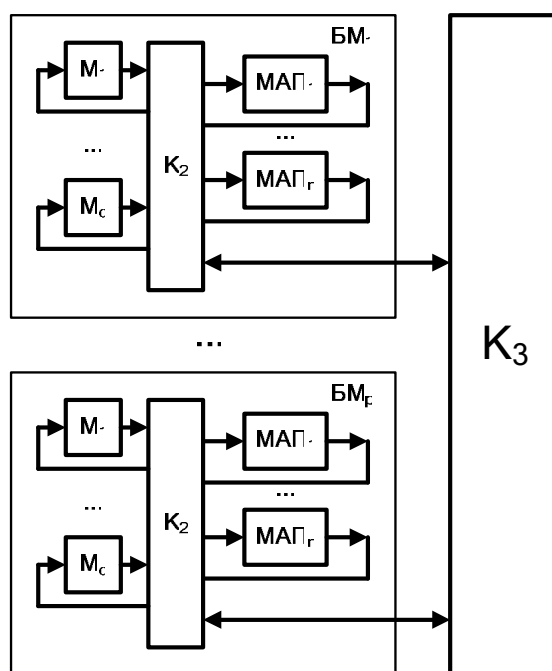


Рис.4.2. Обобщенная структура модульно-наращиваемой многопроцессорной системы с иерархической коммутацией

К недостаткам этого метода можно отнести необходимость дополнительных конструктивных элементов для размещения верхних уровней коммутационной системы.

Другой путь заключается в объединении базовых модулей посредством использования ресурсов внутримодульного коммутатора K_2 , как это показано на рис.4.3. В этом случае можно обойтись одним типом базового модуля. Однако удаленные друг от друга базовые модули будут соединяться по довольно длинным цепочкам и по все более узким каналам. Возможны также любые сочетания рассмотренных вариантов.

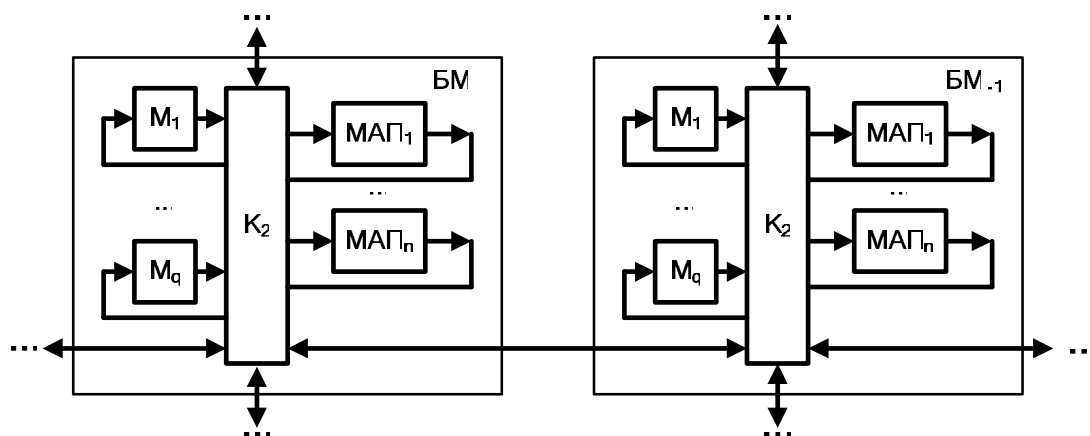


Рис.4.3. Обобщенная структура модульно-наращиваемой многопроцессорной системы на базе внутримодульного коммутатора

Типовая структура базового модуля показана на рис.4.4. В состав базового модуля включены макропроцессоры $МАП_1$ – $МАП_n$, каждый из которых, в свою очередь, содержит элементарные процессоры $ЭП_1$ – $ЭП_m$, объединенные коммутаторами K_1 , блоки распределенной памяти, каждый из которых управляется соответствующим контроллером распределенной памяти (КРП). Все перечисленные компоненты подключены к пространственному коммутатору K_2 , который выполняет не только функции коммутации между ними, но и осуществляет функции коммутации внешних информационных каналов базового модуля. Макропроцессоры, распределенная память и коммутатор K_2 участвуют в организации и выполнении вычислительного процесса как в базовом модуле, так и в реконфигурируемой структуре.

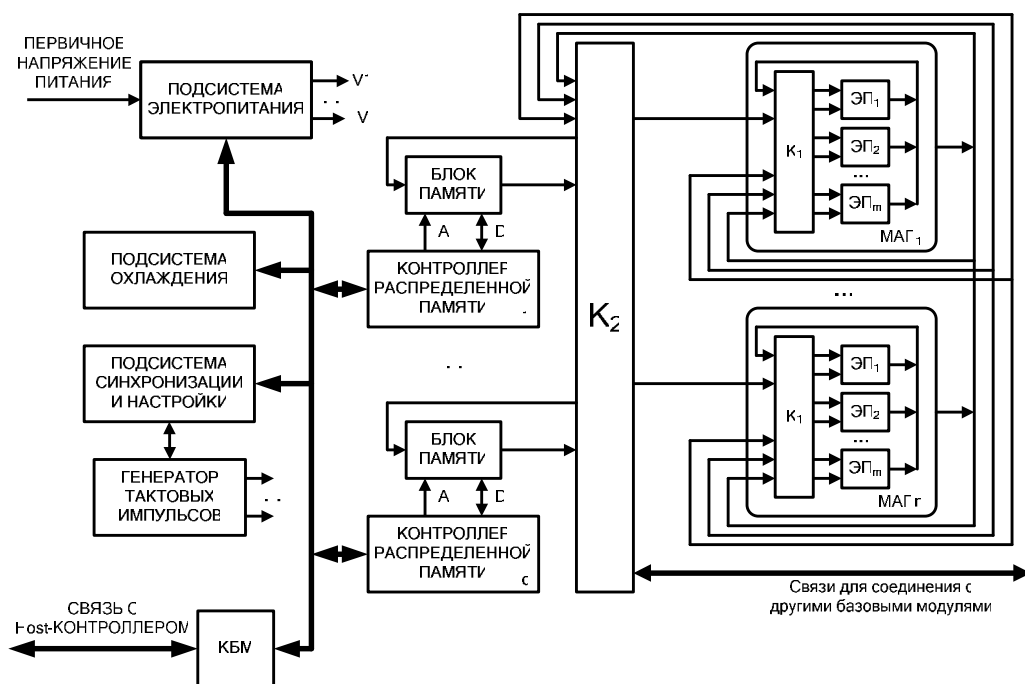


Рис.4.4. Типовая структура базового модуля реконфигурируемой структуры

Контроллер базового модуля (КБМ) выполняет функции управления и контроля всех систем базового модуля. Через этот контроллер осуществляются загрузка входных данных и выгрузка результатов решения из блоков распределенной памяти, а также загрузка программ работы контроллеров распределенной памяти. На базовом модуле располагаются также и вспомогательные подсистемы синхронизации, электропитания и охлаждения.

Центральное место в организации вычислительного процесса отводится контроллерам распределенной памяти. Контроллеры памяти работают с фрагментами параллельной программы, которые загружены в их блоки памяти. Исполняя параллельную программу, контроллеры распределенной памяти участвуют в настройке элементарных процессоров на выполнение необходимых операций и создают с помощью коммутаторов K_1 и K_2 необходимые каналы связи между ними, тем самым реализуя в пределах базового модуля мультиконвейерную вычислительную структуру, соответствующую базовому подграфу задачи.

Одна часть контроллеров распределенной памяти, исполняя фрагменты параллельной программы, организует и синхронизирует потоки данных, подаваемые в созданные вычислительные структуры. Другая часть

контроллеров выполняет функции приема результатов вычислений. Базовый модуль с реализованным подграфом показан на рис. 4.5.

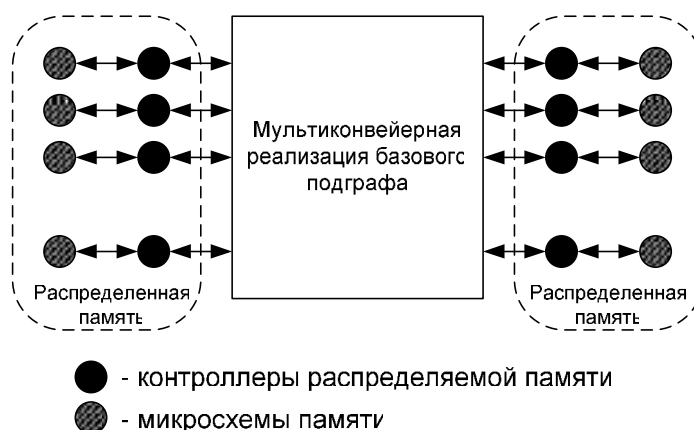


Рис. 4.5. Организация потоков данных через базовый подграф

Для корректного исполнения параллельной программы в базовых модулях контроллеры распределенной памяти совместно с контроллерами других модулей осуществляют синхронную перестройку компонентов системы на выполнение нового кадра. Вычисления в последующем кадре не будут начаты до тех пор, пока во всех базовых модулях не закончится выполнение текущего кадра.

Система команд контроллера памяти обеспечивает эффективную аппаратную поддержку структурно-процедурных вычислений и позволяет гибко реализовывать различные программные конструкции: вложенные циклы, пересчет параметров циклов (начальное значение адреса обращения, приращение и константа цикла), обращение к программе, а также подстановку значений параметров циклов из памяти. Система команд макропроцессора обеспечивает реализацию большого набора арифметико-логических операций, вычисление знаковых функций и предикатов, а также ряда одноместных операций, связанных с преобразованием форматов операндов. Коммутационная система обеспечивает организацию процедур параллельного обращения к каналам распределенной памяти базового модуля, а также информационный обмен между компонентами базовых модулей.

С помощью коммутаторов и программируемых коннекторов базовые модули могут объединяться в блоки. На рис. 4.6 и рис. 4.7 представлены

структурные схемы различных вариантов объединения блоков базовых модулей в более крупные блоки – блоки второго уровня.

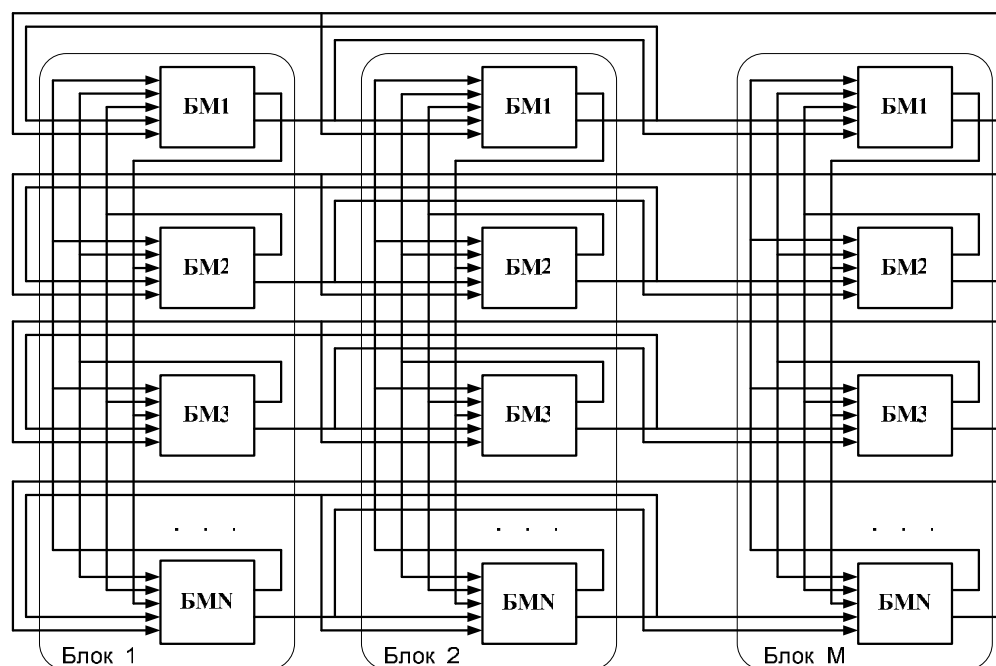


Рис. 4.6. Объединение блоков базовых модулей в блок второго уровня с помощью ортогональной коммутации

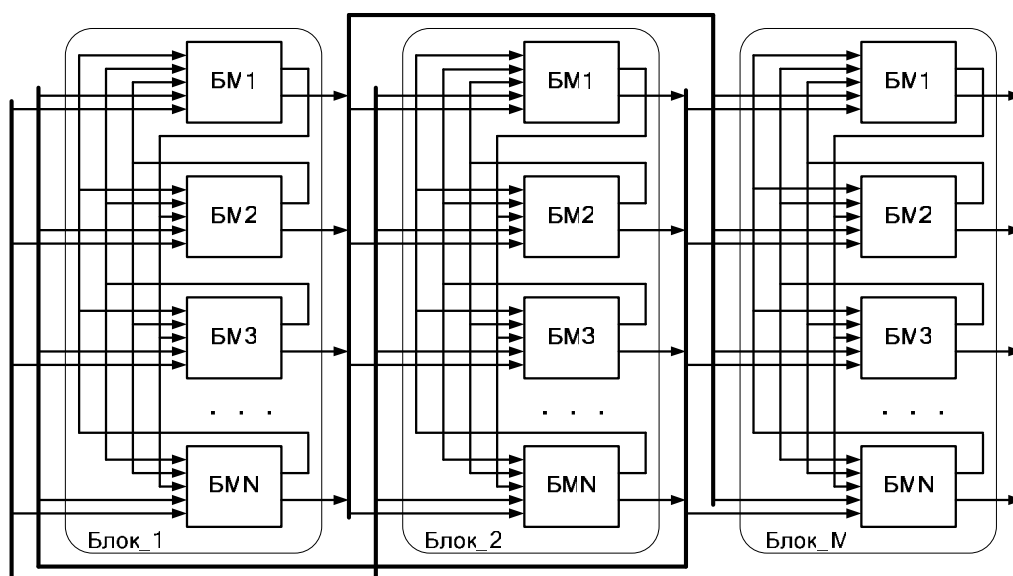


Рис. 4.7. Объединение блоков базовых модулей в блок второго уровня с помощью иерархической коммутации

На структурной схеме (рис.4.6) связи между блоками осуществляются с помощью коммутационной системы, соединяющей по полному графу одноименные базовые модули (ортогональная коммутация). На структурной схеме (рис.4.7) блоки связаны иерархической коммутационной системой.

Пример построения модульной реконфигурируемой структуры с ортогональной коммутационной системой [50] посредством регистровых коннекторов (РК) показан на рис. 4.8.

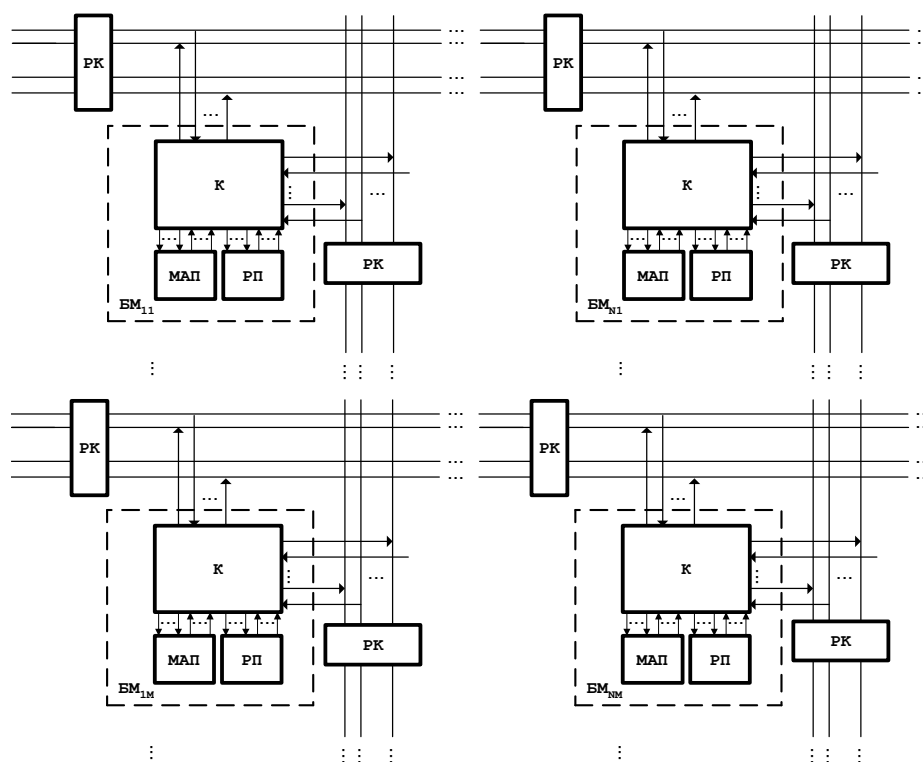


Рис. 4.8. Структурная схема модульной реконфигурируемой структуры с ортогональной коммутацией базовых модулей

Базовые модули (БМ) объединяются с помощью N вертикальных и M горизонтальных q -канальных информационных шин. В каждом базовом модуле эти шины связаны с внутренним коммутатором (К), позволяющим подключать каналы горизонтальных и вертикальных шин данных к группе процессоров в макропроцессоре (МАП) и каналам распределенной памяти данных (РП). Кроме того, коммутатор обеспечивает соединение процессоров и каналов распределенной памяти внутри базового модуля. Количество каналов q , в общем случае, определяется в соответствии с числом макропроцессоров, элементов распределенной памяти и количеством распределенных входов и выходов базового модуля.

Внутренние коммутаторы всех базовых модулей образуют матричный коммутатор системы размерностью $M \times N$ q -канальных шин, объединяющий в единое решающее поле процессоры и распределенную память всех базовых

модулей. Макропроцессор в базовом модуле $БМ_{ij}$ ($i = 1, 2, \dots, M$; $j = 1, 2, \dots, N$) имеет непосредственный ортогональный доступ к каналам распределенной памяти РП, расположенным в соответствующих базовых модулях горизонтальной линейки $БМ_{i1}, \dots, БМ_{iN}$, а также в базовых модулях вертикальной линейки $БМ_{1j}, \dots, БМ_{Mj}$. Доступ к остальным блокам достигается с помощью транзитной коммутации. В такой структуре заложены возможности линейного модульного наращивания. Вместе с тем в его архитектуре просматриваются и механизмы иерархической организации коммутации.

Действительно, если $q > Mkr$, где M – число блоков памяти, k – число информационных входов базового модуля; r – число информационных выходов, то коммутатор вертикальной линейки обеспечивает полноступную коммутацию между макропроцессорами всех базовых модулей линейки, причем эквивалентная ширина каналов обмена в вертикальных и горизонтальных шинах, выделяемых для каждого макропроцессора, соответствует его максимальной пропускной способности. Если $M < q < Mkr$, то полноступность коммутации в рамках вертикальной линейки сохраняется, однако количество эквивалентных каналов обмена b , выделяемых для каждого макропроцессора, будет соответствовать величине [80]

$$b = \left\lceil \frac{q}{\left\lfloor \frac{M}{k+r} \right\rfloor} \right\rceil.$$

При объединении ресурсов нескольких базовых модулей в блок из m базовых модулей вертикальной линейки количество эквивалентных каналов обмена на каждый макропроцессор внутри блока будет определяться величиной $[q/m]$. В случае, когда $q < M$, каналы обмена между макропроцессорами начинают выполнять пространственно-временную коммутацию, разделяя вертикальную шину для обменов во времени.

Другим вариантом ортогонального коммутатора является непосредственное соединение по принципу близкодействия выводов внутренних коммутаторов, как показано на рис. 4.9.

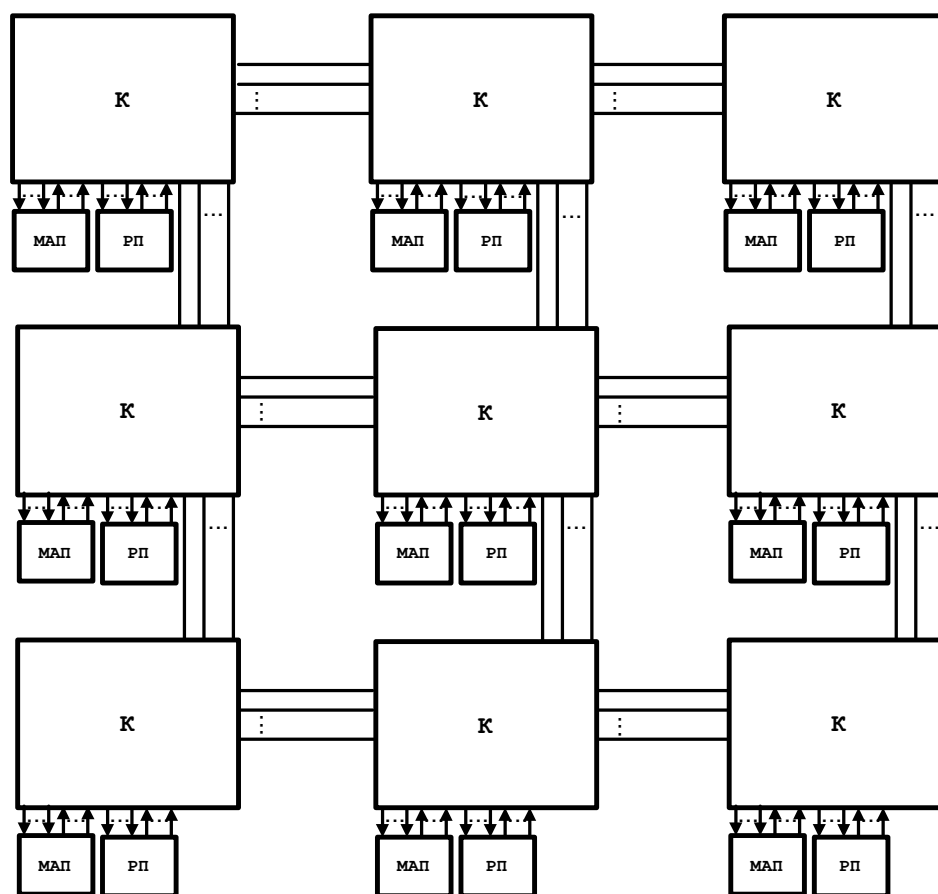


Рис. 4.9. Непосредственное соединение базовых модулей

Такой вариант является более регулярным, что позволяет повысить тактовые частоты межмодульных обменов.

Принцип модульной наращиваемости позволяет увеличить производительность реконфигурируемой структуры при увеличении количества базовых модулей. При этом, как будет показано в шестой главе, обеспечивается ресурсонезависимое и отказоустойчивое программирование, параметризуемое относительно вычислительного ресурса (количества базовых модулей), выделяемого для решения задачи. Задача может быть решена на любой конфигурации вычислительной структуры. Выход из строя одного или нескольких базовых модулей не влечет за собой прекращения решения задачи и необходимости ее полной перетрансляции, а лишь несколько замедляет процесс решения [50].

4.2. Элементная база реконфигурируемых мультиконвейерных вычислительных структур

Техническая реализация идеи настройки архитектуры базового модуля реконфигурируемой структуры на отображение кадров информационного графа задачи требует наличия соответствующей элементной базы, обеспечивающей возможности реализации такой настройки на аппаратном уровне. Эта элементная база должна удовлетворять следующим основным критериям:

- иметь высокую интеграцию, позволяющую реализовать крупные фрагменты вычислений;
- иметь сравнительно низкую стоимость при мелкосерийном производстве вычислительных систем;
- обладать возможностью создания специализированных вычислительных структур;
- обеспечивать создание различных схем распараллеливания вычислений;
- обеспечивать перестройку сформированных вычислительных структур при переходе от одних алгоритмов и схем распараллеливания к другим;
- иметь поддержку разработки технических решений в виде систем автоматизированного проектирования.

Рассмотрим современную элементную базу на соответствие этим критериям.

4.2.1. Заказные СБИС

Заказные СБИС можно условно разделить на полностью заказные микросхемы и микросхемы на базе библиотечных узлов. Полностью заказные микросхемы являются максимально оптимизированными по структуре, быстродействию, площади кристалла, потребляемой мощности. Такие микросхемы используются при массовом производстве аппаратуры. Заказные микросхемы на базе библиотечных узлов предполагают использование при их проектировании как элементарных типовых узлов, например, регистров,

счетчиков, дешифраторов, так и более сложных узлов – шинных формирователей, интерфейсных схем и т.п., которые содержатся в базе типовых узлов САПР. Это позволяет при сохранении практически оптимальных характеристик по быстродействию и занимаемой площади кристалла несколько снизить стоимость проектирования структуры СБИС и, что самое главное, значительно сократить время разработки. Такие микросхемы получили название ASIC (Application Specific Integrated Circuits – Прикладные Специализируемые Интегрируемые Схемы).

К недостаткам заказных СБИС можно отнести высокую стоимость при мелкосерийном производстве, длительное время разработки, модернизации и отладки. Но главное - заказные СБИС обладают неизменной структурой, которую практически невозможно адаптировать к изменившемуся алгоритму работы. Любое, даже незначительное изменение алгоритма, требует вмешательства разработчиков на уровне транзисторных элементов и их связей, что влечет за собой существенные трудовые и финансовые затраты. Переделки фактически эквивалентны разработке нового проекта. Поэтому построение мультиконвейерных вычислительных структур на заказных СБИС нецелесообразно.

4.2.2. Базовые матричные кристаллы

С целью дальнейшего снижения сроков и стоимости проектирования были разработаны специальные полузаказные СБИС, получившие название базовых матричных кристаллов (БМК). Аббревиатура БМК отражает основную суть такого рода микросхем – набор типовых простейших элементов располагается в узлах прямоугольной решетки (матрицы). В БМК, помимо типовых простейших элементов, содержатся средства соединения элементов между собой - трассы, а также типовые интерфейсные узлы с настраиваемыми свойствами. Для формирования законченной цифровой схемы в БМК достаточно создать соединения между контактными окнами элементов, то есть соединить их, используя как имеющиеся готовые трассы, так и соединения,

создаваемые пользователем. При этом для задания соединений, как правило, достаточно одного фотошаблона, создаваемого заказчиком с использованием соответствующей САПР. Еще одним достоинством БМК является то, что не требуется проведения комплексных испытаний конкретного технического решения, реализованного в БМК, поскольку разработанная посредством САПР схема, как правило, работоспособна, а сам БМК аттестован для соответствующих применений. Однако матричное расположение типовых простейших элементов и ограниченные возможности трассировок в БМК приводят к некоторой неоптимальности и избыточности технического решения по сравнению с заказными СБИС.

БМК обладают сопоставимым быстродействием с заказными СБИС. Переделки, связанные с изменениями или уточнением алгоритмов в БМК, сделать значительно проще и дешевле, чем в заказной СБИС, поскольку эти изменения касаются не транзисторных структур, а связей между ними. Разработка фотошаблонов, с помощью которых выполняются связи между логическими блоками, поддерживается эффективными САПР. В то же время переделки БМК практически эквивалентны разработке нового проекта, хотя не столь трудоемки и затратны, как в заказных СБИС. Это дало возможность использовать БМК в качестве элементной базы ранних РМВС. Однако затраты, связанные с созданием новых фотошаблонов и перевыпуском микросхем даже в случае незначительных изменений, вносимых разработчиками РМВС, делают нерентабельным использование БМК в современных реконфигурируемых мультиконвейерных вычислительных структурах.

4.2.3. Системы-на-кристалле

Прогресс в области производства интегральных схем, позволивший реализовывать на кристалле десятки и сотни миллионов транзисторов, неизбежно приводит к противоречию между универсальностью и специализацией. С одной стороны, универсальная СБИС, то есть микропроцессор, обеспечивает массовость производства и, следовательно,

низкую стоимость. В то же время универсальная СБИС для большинства применений обладает большой избыточностью и поэтому приводит к неоптимальным техническим решениям по быстродействию и потребляемой мощности, особенно в массовой аппаратуре. С другой стороны, современные СБИС позволяют практически всю электронную схему разрабатываемого устройства разместить в пределах одной микросхемы, что дает значительный эффект с точки зрения экономии энергии, повышения быстродействия и надежности, снижения затрат на производство и т.п. Однако это приводит к специализации СБИС, а, следовательно, к их удорожанию, поскольку специализированная СБИС не может производиться большими сериями. Компромиссным решением стало создание нового класса СБИС – систем-на-кристалле (SoC – System-on-Chip).

В систему-на-кристалле может включаться множество сложных, функционально законченных блоков, таких как универсальные процессоры, память, различного рода контроллеры и сопроцессоры, интерфейсные узлы и многое другое. Помимо функционально законченных блоков в SoC, как правило, включаются и программируемая логика типа FPGA (Field Programmable Gate Array) или области типа БМК, в которых пользователь может создать те или иные недостающие компоненты.

В основе методологии проектирования SoC лежит принцип использования IP-ядер, разрабатываемых либо пользователем, либо специализированными фирмами – поставщиками IP- ядер. Многократное использование IP-ядер позволяет существенно сократить сроки создания специализированной СБИС – SoC и существенно удешевить производство даже при ограниченных партиях микросхем.

Компоненты системы-на-кристалле могут быть реализованы или в виде заказных специализированных фрагментов, или могут быть реализованы в программируемой части микросхемы – в области FPGA. В первом случае IP-ядра задаются в виде топологии и реализуются как заказные фрагменты, являющиеся аналогами сложных электронных компонентов, во втором случае –

в виде VHDL описания как фрагмент программируемой логической интегральной схемы.

В обоих случаях системы-на-кристалле являются заказными СБИС с точки зрения специализации их структуры, поскольку структура большинства IP-ядер не может быть изменена в процессе эксплуатации. Это является основным недостатком, снижающим эффективность применения систем-на-кристалле в качестве элементной базы реконфигурируемых вычислительных структур.

4.2.4. Программируемые логические интегральные схемы

Основными преимуществами программируемых логических интегральных схем (ПЛИС) являются: возможность реализации сложных параллельных алгоритмов, наличие средств САПР, позволяющих провести полное моделирование системы, возможность программирования или изменения конфигурации непосредственно в системе, совместимость различных проектов при их переводе на языки описания аппаратуры - VHDL, AHDL, Verilog и др.

История развития архитектур ПЛИС берет свое начало с конца 1970-х годов, когда на рынке появились ПЛИС с программируемыми матрицами "И" и "ИЛИ" – архитектуры FPLA (Field Programmable Logic Array) и FPLS (Field Programmable Logic Sequencers). Недостатком их было слабое использование ресурсов программируемой матрицы "ИЛИ".

Идя по пути совершенствования архитектуры, в конце 1980-х годов разработчики ПЛИС предложили более простую и изящную архитектуру программируемой матричной логики PAL (Programmable Array Logic) и GAL (Gate Array Logic) – ПЛИС фирм Intel, Altera, AMD, Lattice и др. с небольшой степенью интеграции, имеющую программируемую матрицу "И" и фиксированную матрицу "ИЛИ". Другим подходом к уменьшению избыточности программируемой матрицы "ИЛИ" являлась программируемая макрологика. Построенные по данной архитектуре микросхемы содержали

единственную программируемую матрицу "И-НЕ" или "ИЛИ-НЕ", но за счет многочисленных инверсных обратных связей были способны формировать сложные логические функции.

В начале 1980-х годов на мировой рынок микроэлектронных изделий выходят три ведущие фирмы-производители ПЛИС. В июне 1983 года основана фирма Altera Corporation, в феврале 1984 - компания Xilinx Inc., в 1985 году – Actel Corporation. Эти три компании в настоящее время занимают до 80% всего рынка ПЛИС и являются основными идеологами их применения. Если ранее ПЛИС являлись одним из множества продуктов, выпускаемых такими гигантами как Intel, AMD и др., то, начиная с середины 1980-х годов, на рынке ПЛИС происходит специализация и законодателями мод являются фирмы, специализирующиеся только на разработке и производстве ПЛИС.

С появлением новых производителей возникли и новые архитектуры. ПЛИС CPLD (Complex Programmable Logic Devices) имеют довольно высокую степень интеграции и содержат несколько логических блоков, объединенных коммутационной матрицей. Каждый логический блок представляет собой программируемую матрицу "И" и фиксированную матрицу "ИЛИ". К этому классу относятся ПЛИС семейств MAX7000 фирмы Altera, XC9500 фирмы Xilinx, а также большое число микросхем других производителей (Atmel, Vantis, Lucent и др.).

Настоящая революция в устройствах программируемой логики произошла с появлением ПЛИС, имеющих архитектуру Field Programmable Gate Array (FPGA). К FPGA относятся ПЛИС XC2000, XC3000, XC4000, Spartan фирмы Xilinx, АСТ1, АСТ2 фирмы Actel, а также семейства FLEX8000 фирмы Altera, некоторые ПЛИС Atmel и Vantis.

В ПЛИС FPGA множество конфигурируемых логических блоков (Configurable Logic Blocks (CLBs)) объединяется с помощью матрицы соединений. Логические блоки таких ПЛИС состоят из одного или нескольких относительно простых логических элементов, в основе которых лежат 4-входовая таблица преобразования (Look-up table, LUT), программируемый

мультиплексор, D-триггер, а также цепи управления. Характерными для FPGA-архитектур являются элементы ввода-вывода, позволяющие реализовать двунаправленный ввод/вывод, третье состояние и т.п. Особенностью современных ПЛИС FPGA является возможность тестирования узлов с помощью порта JTAG (B-scan), а также наличие внутреннего генератора (Osc) и схем управления последовательной конфигурацией.

Дальнейшее развитие архитектуры FPGA привело к появлению ПЛИС, имеющих встроенные реконфигурируемые модули памяти, позволяющие использовать ПЛИС без применения внешних запоминающих устройств. Благодаря большой логической емкости, удобной архитектуре, достаточно высокой надежности и удачному соотношению цена/логическая емкость, данные ПЛИС удовлетворяют разнообразным требованиям, возникающим у разработчика.

В последние годы, наряду с заказными SoC, ПЛИС также становятся основой для "систем-на-кристалле". IP-ядра этих систем разрабатываются отдельно и могут многократно использоваться в различных проектах. Окончательная структура SoC-микросхемы на базе ПЛИС выполняется на основе IP-ядер с помощью программ автоматизации проектирования электронных устройств.

Идеология построения SoC подстегнула ведущих производителей ПЛИС к выпуску в конце 1998 – начале 1999 года изделий с эквивалентной емкостью миллион эквивалентных вентилях и более. Примером новых семейств ПЛИС, пригодных для реализации SoC, является семейство ApEX20K фирмы Altera.

Сходной архитектурой обладает и семейство Virtex фирмы Xilinx, которое характеризуется широким разнообразием высокоскоростных трассировочных ресурсов, наличием выделенного блочного ОЗУ, развитой логикой ускоренного переноса. ПЛИС данной серии обеспечивают высокие скорости межкристального обмена – до 200 МГц (стандарт HSTL IV). Кристаллы серии Virtex за счёт развитой технологии производства и

усовершенствованного процесса верификации имеют достаточно низкую стоимость (до 40% от эквивалентной стоимости серии XC4000XL).

Увеличение эквивалентной логической емкости ПЛИС привело к тому, что в 1998–1999-х годах началось изменение отношения к программному обеспечению САПР ПЛИС со стороны как разработчиков программного обеспечения, так и пользователей. Если до конца 1990-х годов основным средством описания проекта являлся ввод схемы с помощью графических редакторов с использованием библиотек стандартных логических примитивов - логических элементов, простейших комбинационных и последовательностных функциональных узлов, аналогов стандартных интегральных схем малой и средней степени интеграции, то в настоящее время разработчиками широко используются языки описания аппаратуры (Hardware Description Languages) для реализации алгоритмов на ПЛИС. Причем в современных САПР поддерживаются как стандартизованные языки описания аппаратуры типа VHDL, Verilog, HDL, так и языки описания аппаратуры, разработанные компаниями-производителями ПЛИС специально для использования только в своих САПР и учитывающие архитектурные особенности конкретных семейств ПЛИС. Примером может служить AHDL (Altera Hardware Description Languages), поддерживаемый САПР MAX PLUS II и Quartus компании Altera. HDL-языки являются очень удобным средством описания различного рода интерфейсов, однако при реализации сложных вычислений разработчик не имеет возможности влиять на процедуру отображения написанного на HDL кода на физические ресурсы кристалла, что отрицательно сказывается на эффективности реализации. Кроме того, очевидна наглядность схемотехнического описания, поэтому при реализации сложного вычислительного алгоритма представляется предпочтительным использование графического редактора.

Фирма Xilinx предлагает IP-ядра для часто используемых элементов, включая блоки цифровой обработки сигналов, шинные интерфейсы, процессоры и периферийные устройства. Использование пакета LogiCORE™

фирмы Xilinx и аналогичных программ сторонних производителей для создания IP-ядер сокращает время разработки проектов, уменьшает риски и позволяет получать наивысшую производительность. В дополнение к этому система генерации IP-ядер CORE Generator™ позволяет разработчикам реализовать собственные IP-ядра с предсказуемыми и воспроизводимыми временными характеристиками. Данный продукт имеет простой пользовательский интерфейс для генерации параметризованных IP-ядер, оптимизированных для FPGA Xilinx.

Семейство Virtex-IV фирмы Xilinx использует усовершенствованную архитектуру на основе модульных блоков (Advanced Silicon Modular Block, ASMBL). Семейство содержит три подсемейства: LX, FX и SX, которые в совокупности удовлетворяют потребностям самых разных областей применения ПЛИС. Внутри ПЛИС реализовано несколько аппаратных ядер, предназначенных для выполнения ряда типовых задач. Эти ядра включают процессоры PowerPC с новым интерфейсом APU, модули Ethernet (Tri-Mode Ethernet MAC), последовательные передатчики со скоростями передачи от 622 Мбит/с до 11,1 Гбит/с, модули мониторинга напряжения и температуры, выделенные секции DSP, высокоскоростные цепи распределения тактового сигнала и большое количество интерфейсных блоков. Базовые блоки Virtex-4 представляют собой улучшенные версии подобных блоков, имеющих в предыдущих семействах Virtex: Virtex, Virtex-E, Virtex-II, Virtex-II Pro и Virtex-II Pro X, что обеспечивает совместимость «снизу вверх» с существующими проектами. Устройства Virtex-IV выполнены по 90-нм техпроцессу с использованием медных проводников и 300-мм пластин.

ПЛИС Virtex-IV представляет собой регулярную структуру, основными элементами которой являются:

- блоки ввода/вывода IOB;
- конфигурируемые логические блоки CLB;
- секции блочной памяти Block SelectRAM;
- блоки встроенных модулей цифровой обработки сигналов DSP;

- цифровые модули управления синхронизацией DCM;
- трассировочные ресурсы.

Конфигурируемые логические блоки CLB соединены с переключающей матрицей для доступа к главной трассировочной матрице и состоят из четырех связанных секций Slice, сгруппированных в пары SLICEM и SLICEL. Каждая пара имеет независимую цепь переноса, кроме того, в паре SLICEM имеется цепь сдвига.

В состав каждой секции CLB входят два логико-функциональных генератора, два запоминающих элемента, конфигурируемых как D-триггеры, широкофункциональные мультиплексоры MuxF5 и MuxFX, логика ускоренного переноса и каскадирования, а также логика для арифметических вычислений.

Функциональные генераторы конфигурируются как четырехходовые таблицы преобразования look-up tables (LUT), способные реализовывать любые четырехходовые булевы функции, при этом задержка распространения сигнала при прохождении через LUT не зависит от реализованной функции. Каждый функциональный генератор в паре секций SLICEM может быть сконфигурирован как элемент распределенной памяти RAM 16x1-бит или как 16-битовый сдвиговый регистр SRL без использования доступных в секции D-триггеров, то есть каждый LUT может выполнять задержку последовательности данных от 1 до 16 тактов. Для получения больших задержек возможно каскадирование LUT с помощью линий ShiftIn, ShiftOut. В одном блоке CLB можно получить до 64 тактов задержки. Подобное программирование задержек позволяет создавать сбалансированные конвейерные структуры. Все функциональные генераторы могут быть сконфигурированы и как элементы ПЗУ ROM.

Мультиплексоры эффективно комбинируют таблицы преобразования в пределах CLB, а также между различными CLB, позволяя реализовывать различные логические функции даже с большим количеством переменных. Функции до девяти переменных могут быть реализованы в одной секции Slice. Логика для арифметических вычислений представляет собой элемент XOR,

позволяющий реализовывать двухбитовое суммирование в пределах Slice. Дополнительный элемент AND улучшает эффективность трассировки операции умножения.

В 2006 году фирма Xilinx анонсировала выпуск нового семейства ПЛИС Virtex-V. Данные устройства выполнены с соблюдением технологических норм 65 нм с медными проводниками и напряжением питания ядра 1 В. В настоящее время пользователю уже доступны ПЛИС одной серии Virtex-V емкостью до 8000 тыс. «эквивалентных» ячеек.

Архитектура ПЛИС претерпела кардинальные изменения. Логико-функциональные генераторы теперь представляют собой 6-входовую таблицу истинности, которая может конфигурироваться как 32-разрядный сдвиговый регистр или блок памяти 64x1. Каждый функциональный генератор имеет два выхода и поэтому может конфигурироваться не только как произвольная 6-входовая булева функция, но и как две произвольно определенных 5-входовых булевых функции при условии наличия у них общих входов. Это позволит в сложных проектах существенно уменьшить число последовательно соединяемых логических ячеек, реализующих функцию многих переменных, а, значит, и уменьшает задержку распространения сигнала, что ведет к росту тактовой частоты проекта.

Изменения коснулись и блочной памяти, объем базового блока которой составляет теперь 36 Кбит, причем каждый блок может быть использован как два независимых блока по 18 Кбит. В блоках DSP стало доступно перемножение 25-разрядного числа на 18-разрядное, добавлены независимый 48-разрядный вход и возможность суммирования трех операндов в пределах одного блока. Любопытные изменения произошли в схеме трассировочных ресурсов: в дополнение к вертикальным и горизонтальным трассировочным линиям в Virtex-V предусмотрены диагональные соединения между логическими ячейками. Появился новый вариант модуля формирования тактового сигнала – Clock Manager Tile (CMT), в котором имеются два уже

знакомых по Virtex-4 устройства DCM (цифровых) и одно устройство фазовой автоподстройки частоты PLL (аналоговое).

Максимальная тактовая частота ПЛИС семейства Virtex-V достигла 550 МГц, а потребляемая мощность снизилась примерно на 40% по сравнению с Virtex-IV.

Таким образом, быстрое развитие технологии ПЛИС в последние годы предоставило разработчикам вычислительных систем поистине уникальные возможности. Последние разработки ведущих производителей ПЛИС позволяют реализовывать в одной микросхеме самые разнообразные схемотехнические решения - от универсальных микропроцессоров, схем цифровой обработки сигналов, нейронных сетей, схем различных интерфейсов до самых изощренных специализированных вычислителей. Причем все это многообразие технических решений может быть в любой момент изменено пользователем в зависимости от потребности при решении тех или иных задач. Поэтому можно сделать однозначный вывод о том, что ПЛИС являются наиболее перспективной элементной базой для построения реконфигурируемых мультиконвейерных вычислительных структур.

4.3. Принципы построения базовых модулей реконфигурируемых

мультиконвейерных вычислительных структур на основе ПЛИС

Использование ПЛИС как основного элемента при построении базовых модулей реконфигурируемых мультиконвейерных вычислительных структур позволяет придать им новые качества, связанные с интеграцией целого ряда функциональных блоков (см. рис. 4.4) в поле логических ячеек ПЛИС. Типовая компоновка базового модуля на основе ПЛИС показана на рис. 4.10.

Базовый модуль на основе ПЛИС несет в себе все характерные признаки завершенной реконфигурируемой системы [55]. В то же время он обладает возможностями для комплексирования с другими такими же базовыми модулями с целью наращивания производительности.

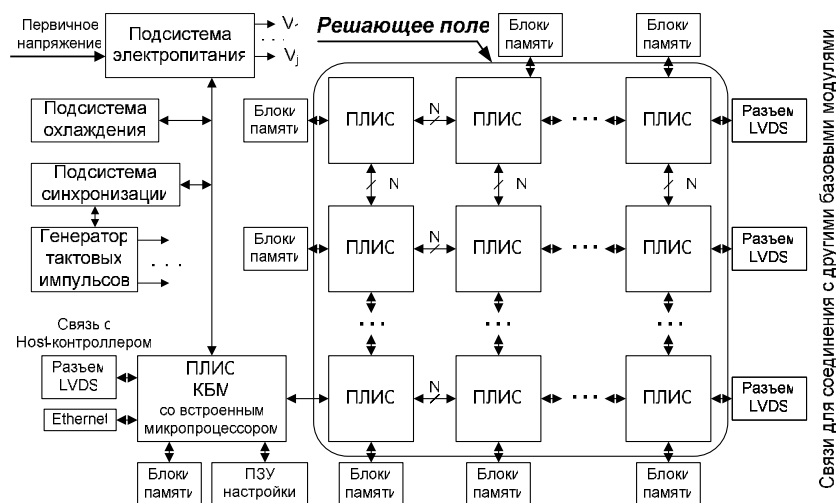


Рис. 4.10. Структура базового модуля реконфигурируемой системы на основе ПЛИС

Основные вычислительные возможности базового модуля сосредоточены в решающем поле, которое содержит некоторое множество ПЛИС большой степени интеграции. В решающем поле создаются вычислительные и коммутационные структуры, показанные на рис.4.4: макропроцессоры $МАП_1 - МАП_n$, каждый из которых, в свою очередь, содержит несколько элементарных процессоров $ЭП_1 - ЭП_m$, объединенных коммутаторами K_1 ; внутримодульный коммутатор K_2 ; контроллеры распределенной памяти для управления блоками распределенной памяти. При этом сами блоки распределенной памяти выполняются на типовых микросхемах ОЗУ SRAM или SDRAM необходимого объема и быстродействия.

Поскольку в решающем поле реализуются функциональные узлы, которые располагаются в ресурсах нескольких или даже всех микросхем, то необходимо стремиться к тому, чтобы максимально снизить эффект границ между кристаллами ПЛИС. Снижение межкристального эффекта границ достигается двумя путями. Первый из них предполагает использование ПЛИС максимальной интеграции, которые позволяют размещать все более крупные фрагменты функциональных узлов. Вторым путем заключается в создании регулярных и однородных связей между ПЛИС. С этой целью микросхемы располагаются в узлах двумерной решетки и соединяются между собой ортогональной системой связей по близкодействию. Такая система связей

позволяет существенно упростить печатную плату и улучшить ее частотные характеристики, поскольку связи между соседними микросхемами не превышают единиц сантиметров. Данные между дальними микросхемами передаются по транзитным каналам через промежуточные микросхемы, используя систему ортогональных связей.

Связи между ПЛИС решающего поля базового модуля целесообразно выполнять на основе стандарта LVDS (Low Voltage Differential Signaling – дифференциальные сигналы низкого напряжения). Преимуществами стандарта LVDS являются низкая потребляемая мощность выходных каскадов, низкий уровень создаваемых электромагнитных излучений, невосприимчивость к синфазным электромагнитным помехам и наличие в микросхемах современных ПЛИС аппаратной поддержки для организации высокоскоростных передач данных на основе стандарта LVDS. Физически каждая линия связи представляет собой пару дифференциальных полосковых проводников, с обоих концов подключенных к специальным выводам микросхем. Темп передачи данных по каждой двухпроводной линии, в зависимости от реализации, может составлять несколько гигабит в секунду.

Следующим конструктивным уровнем, где необходимо принимать специальные меры для снижения эффекта границ, являются базовые модули.

Поскольку при реализации мультиконвейерных вычислительных структур больших базовых подграфов может не хватить ресурсов одного базового модуля, то возникает необходимость реализации таких структур в пределах двух и более базовых модулей. При этом важно сохранить темп передачи данных из микросхем решающего поля одного базового модуля непосредственно в ПЛИС другого базового модуля. Для сохранения темпа передачи в межмодульных обменах целесообразно также использовать двухпроводные каналы LVDS. Это позволяет передавать большие информационные потоки между модулями в процессе их совместной работы. Фактически каналы на основе LVDS связывают внутримодульные

коммутаторы, что позволяет реализовывать различные схемы межмодульных соединений.

Единый тип интерфейса между ПЛИС как в пределах одного базового модуля, так и между различными базовыми модулями, обеспечивает одинаковые скорости обмена и снижает эффект границ в пределах общего схемотехнического ресурса мультиконвейерных вычислительных структур.

Блоки распределенной памяти используются для реализации информационных вершин базового подграфа при подаче входных данных и записи результатов вычислений. Для создания эффективных вычислительных структур в пределах базового модуля необходимо оптимальное соотношение количества ПЛИС, количества блоков распределенной памяти и их объема. Для различных задач, решаемых на мультиконвейерных вычислительных структурах, это соотношение различно. В то же время для мультиконвейерных вычислительных структур не требуется запоминания множества промежуточных данных, так как они передаются для дальнейшей обработки в последующие ступени конвейерного вычислителя без промежуточного запоминания. Это снижает требования к общему объему памяти на базовом модуле. Для мультиконвейерных вычислительных структур более критичным является количество блоков распределенной памяти, а не объем каждого блока или общий объем памяти. Помимо распределенной памяти, реализованной на типовых микросхемах ОЗУ, при создании вычислительных структур широко используется внутрикристальная память ПЛИС.

Еще одной особенностью базовых модулей на основе ПЛИС является реализация контроллеров распределенной памяти (КРП) не в виде отдельных устройств, как это показано на рис.4.4, а в виде отдельных функциональных устройств в ресурсе решающего поля. Реализация распределенной памяти в базовом модуле на основе ПЛИС показана на рис. 4.11.

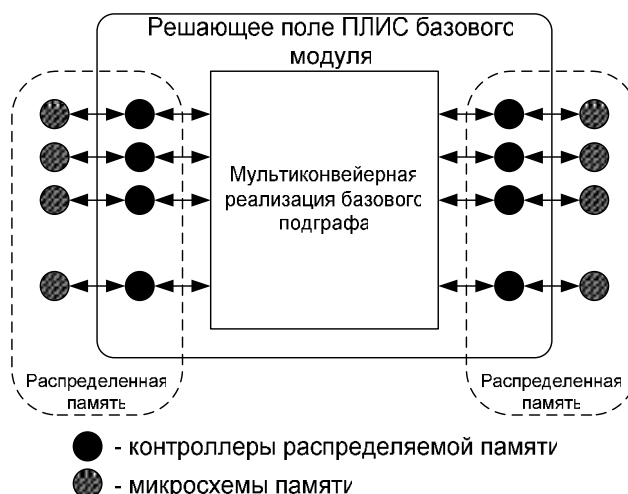


Рис. 4.11. Реализация распределенной памяти в базовом модуле на основе ПЛИС

Большое количество ПЛИС высокой степени интеграции в составе базового модуля требует специальных технических решений для ускорения загрузки в них конфигурационных файлов. Технология ПЛИС предусматривает несколько вариантов загрузки конфигурации: из конфигурационного ПЗУ; от внешнего источника; последовательно бит за битом или параллельно-последовательно в виде последовательности байтов. На базовых модулях реконфигурируемых систем могут применяться различные сочетания этих вариантов.

Поскольку настройка ПЛИС решающего поля выполняется непосредственно контроллером базового модуля или при его участии, то его настройку целесообразно проводить из конфигурационного ПЗУ, причем инициализация настройки происходит по включении питания. Это обеспечит одновременную готовность всех базовых модулей реконфигурируемой структуры к загрузке конфигурации ПЛИС решающего поля.

Настройку конфигурации ПЛИС решающего поля целесообразно выполнять параллельно-последовательно, так как это позволит в несколько раз ускорить загрузку микросхемы по сравнению с последовательным методом загрузки. Дальнейшее ускорение загрузки возможно на пути создания нескольких параллельных каналов настройки. Пусть решающее поле базового модуля содержит $n \times m$ ПЛИС, и пусть также время байтной параллельно-

последовательной загрузки конфигурационного файла в одну микросхему составляет t . Если загрузку вести параллельно во все микросхемы, то время составит t . Если загрузку вести поочередно по одному каналу, то время загрузки всех микросхем составляет $n \times m \times t$. Если микросхемы разбить, например, на n групп из m микросхем каждая и загрузку групп вести по n параллельным каналам, а в пределах группы последовательно, то время загрузки составит $m \times t$. Выбор варианта загрузки конфигурации решающего поля зависит от многих параметров базового модуля и от требований, предъявляемых ко времени готовности и составу аппаратных и программных средств реконфигурируемой структуры.

4.4. Реализация базовых модулей реконфигурируемых

мультиконвейерных вычислительных структур на основе ПЛИС

Как уже отмечалось, реконфигурируемые мультиконвейерные вычислительные структуры целесообразно строить из однотипных конструктивных элементов – базовых модулей. Такой подход позволяет унифицировать оборудование, упростить и удешевить разработку и производство реконфигурируемых вычислительных структур. Покажем, что конструктивные особенности базовых модулей существенно влияют на характеристики реконфигурируемых структур и поэтому архитектурным решениям, положенным в их основу, необходимо уделять особое внимание. Рассмотрим эволюцию технических решений, использованных при реализации базовых модулей на основе ПЛИС.

4.4.1. Структуры базовых модулей первого поколения

Рассмотрим структурную схему базового модуля первого поколения с иерархической системой коммутации. Секция макропроцессоров представляет собой группу из восьми ПЛИС, которые объединяются в МАП с помощью внутреннего коммутатора. Все элементарные процессоры МАП соединяются по

полному графу [50]. Блок распределенной памяти с помощью коммутаторов K_1 и K_2 подключается к макропроцессору. Выходы и входы макропроцессора и блока распределенной памяти связаны по полному графу. Коммутатор K_3 используется для выбора выходных каналов блока распределенной памяти и макропроцессора для коммутации на третьем уровне. Третий уровень коммутации – межмодульный - реализуется с помощью коммутатора K_4 и коннекторов C_1 , C_2 и C_3 . Соединение с другими базовыми модулями можно было выполнить по схеме, показанной на рис. 4.8.

Данный модуль был разработан в НИИ МВС (г. Таганрог) и реализован в 1996 году на основе базовых матричных кристаллов, а в 1998 году – на основе ПЛИС-технологии [49]. На рис. 4.12 показана структурная схема базового модуля первого поколения, построенного на ПЛИС.

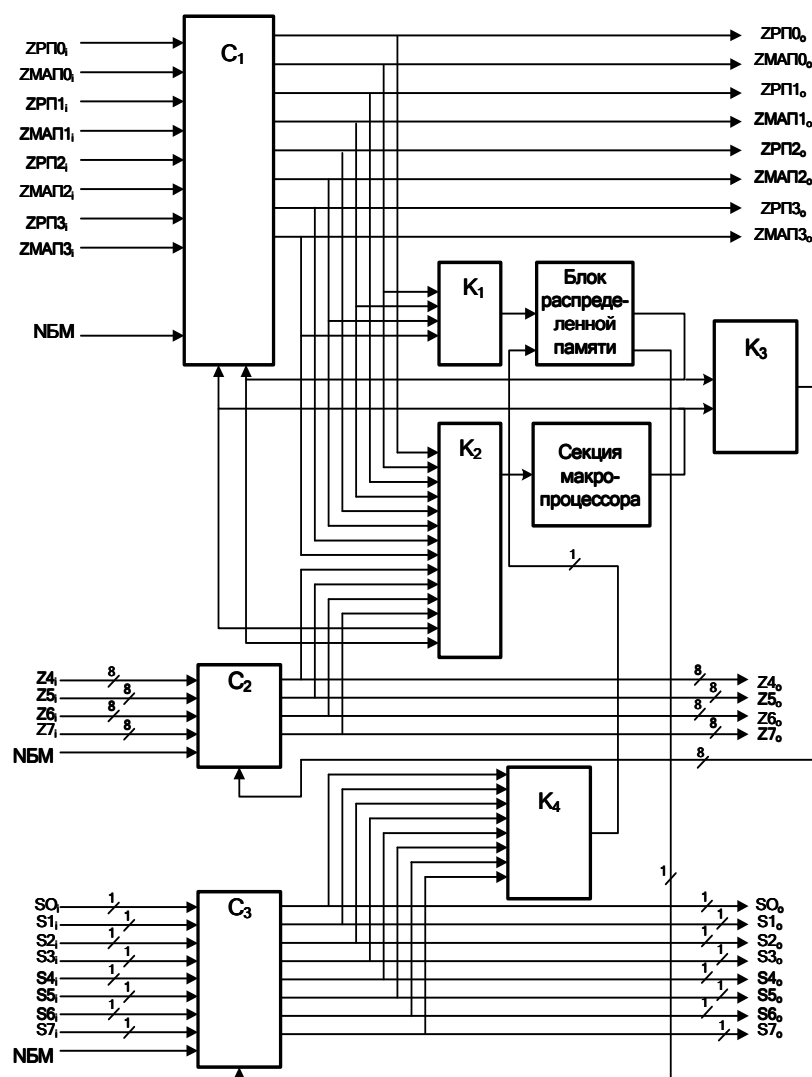


Рис. 4.12. Структурная схема базового модуля

С целью минимизации аппаратных затрат на реализацию коммутационной системы применялись знакоразрядные последовательные коды, обеспечивающие обработку информации старшими разрядами вперед [43, 93]. Обработка и передача данных последовательными кодами позволили на порядок сократить аппаратные затраты на коммутационную систему. Тем не менее, коммутационная система потребовала около 60% общего аппаратного ресурса базового модуля. На рис. 4.13 показана фотография базового модуля первого поколения.

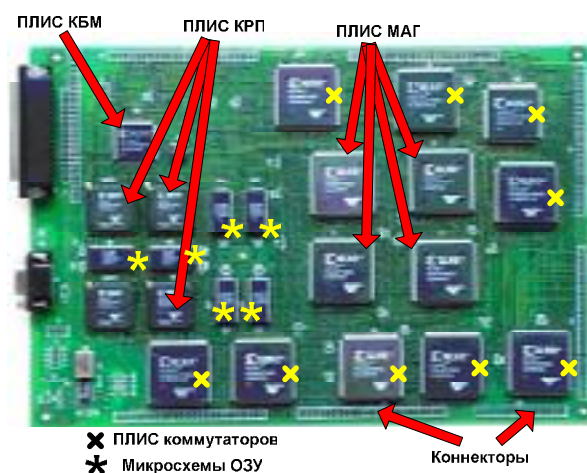


Рис. 4.13. Базовый модуль первого поколения на основе ПЛИС

Реализация макропроцессора на ПЛИС выявила несоответствие последовательного принципа обработки информации с использованием знакоразрядных кодов и архитектуры ПЛИС, что вылилось в снижение эффективности вычислений в макропроцессорах базового модуля.

Снижение эффективности макропроцессора, реализованного на ПЛИС-технологии, по сравнению реализацией на основе базовых матричных кристаллов было вызвано тем, что в структуре ПЛИС аппаратно поддерживается обработка параллельных данных. Реализация вычислений в знакоразрядных кодах потребовала существенно больше ресурсов, чем реализация вычислений в дополнительном коде. Таким образом, переход на ПЛИС-технологию хотя и позволил создать вычислительное устройство на основе современной элементной базы, но при этом производительность базового модуля по сравнению с образцом 1996 года выросла всего в 2 раза при

увеличении тактовой частоты в 4 раза. Поэтому в дальнейших разработках было принято решение использовать традиционные методы обработки данных, представленных параллельными кодами.

4.4.2. Структура базовых модулей второго поколения

Параллельная обработка данных неизбежно влечет за собой увеличение аппаратных затрат на построение коммутационной системы, причем аппаратные затраты пропорциональны количеству разрядов в коммутируемых данных. Поэтому для модуля, работающего с параллельными данными, требовалось значительно сократить расходы оборудования на коммутационную систему. В этой связи была выбрана ортогональная коммутационная система, которая, с одной стороны, обеспечила полноступное соединение элементов системы, а с другой – потребовала существенно меньших затрат по сравнению с иерархическим коммутатором, примененным на базовых модулях первого поколения.

Еще одним фактором, повлиявшим на выбор коммутационной системы, была предопределенность коммутации, которая формирует связи между вершинами базового подграфа. Эта предопределенность связана с детерминированностью вычислительного процесса при структурно-процедурной организации вычислений, принятой в реконфигурируемых структурах, которая детально рассматривалась в главе 3. Детерминизм реализации структурно-процедурных вычислений позволяет уже на этапе трансляции алгоритма определить требуемую коммутацию, которую нужно запрограммировать в коммутаторе базового модуля и в реконфигурируемых структурах в целом. Это дает возможность заранее оценить соответствие коммутационных возможностей аппаратуры и потребностей задачи и учесть это при сегментировании информационного графа.

Ортогональная коммутационная система, принятая для организации связей между ПЛИС базового модуля, существенно проще иерархической и обеспечивает не только соединение соседних ПЛИС, но и возможность

соединения любых ПЛИС базового модуля посредством транзитной передачи данных. Данное свойство обеспечивается созданием в составе каждой ПЛИС внутреннего коммутатора, который подключает внутренние ресурсы ПЛИС к каналам передачи данных и обеспечивает коммутацию всех внешних информационных каналов, подведенных к данной ПЛИС, в том числе для обеспечения транзитной передачи информационных потоков. Ортогональная система связей между ПЛИС базового модуля имеет еще и дополнительное преимущество, так как физические связи между ПЛИС на печатной плате получаются достаточно короткими и одинаковыми по длине и электрическим параметрам, что существенно упрощает разработку самой печатной платы и повышает частотные свойства каналов, а, следовательно, и их быстродействие.

На рис. 4.14. представлена структурная схема базового модуля реконфигурируемой структуры, построенного на основе ортогональной коммутационной системы.

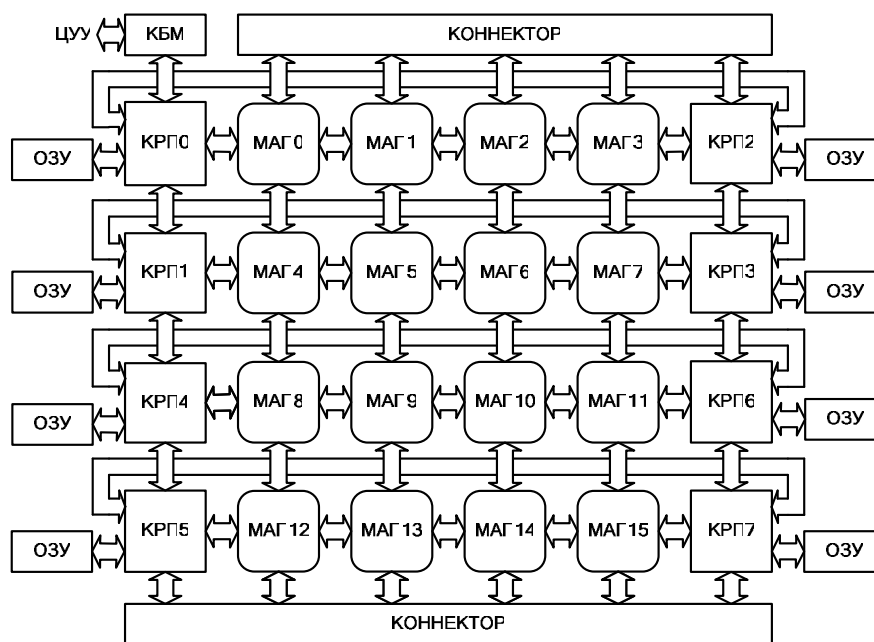


Рис. 4.14. Схема базового модуля с ортогональной коммутационной средой

В состав базового модуля входят:

- шестнадцать макропроцессоров (МАП), каждый из которых содержит четыре элементарных процессора и коммутационный элемент,

обеспечивающий соединение по полному графу элементарных процессоров и внешних каналов связи;

- распределенная память, содержащая восемь контроллеров распределенной памяти (КРП), имеющих в своем составе коммутационный элемент, который обеспечивает соединение по полному графу каналов распределенной памяти и внешних каналов связи;

- микросхемы статического ОЗУ;

- контроллер базового модуля (КБМ).

Таким образом, из состава базового модуля исключены как самостоятельные функциональные элементы: внутримодульный коммутатор, создающий связи между макропроцессорами, коммутаторы, которые соединяют элементарные процессоры в макропроцессоре, коммутатор, соединяющий макропроцессор и распределенную память, и коммутатор внешних связей. Функции этих коммутаторов распределены между частями ортогонального коммутатора, которые расположены во всех ПЛИС базового модуля.

Контроллер базового модуля предназначен для управления базовым модулем и обмена данными с host-контроллером. Распределенная память содержит восемь контроллеров. К каждому контроллеру распределенной памяти подключены микросхемы ОЗУ, представляющие собой блоки распределенной памяти объемом 4 Мбайт каждый. Система команд КРП в сочетании с возможностями коммутационных элементов внутри контроллера обеспечивает возможность организации эффективных высокоскоростных и бесконфликтных процедур доступа к информационным массивам, расположенным в распределенной памяти. Каждый из шестнадцати макропроцессоров содержит четыре элементарных процессора, коммутационный элемент и блок команд, которые обеспечивают эффективную реализацию структурно реализуемых фрагментов задачи.

Базовый модуль реконфигурируемой структуры с ортогональной системой коммутации был разработан в НИИ МВС в 2000 году. Фотография базового модуля представлена на рис. 4.15.

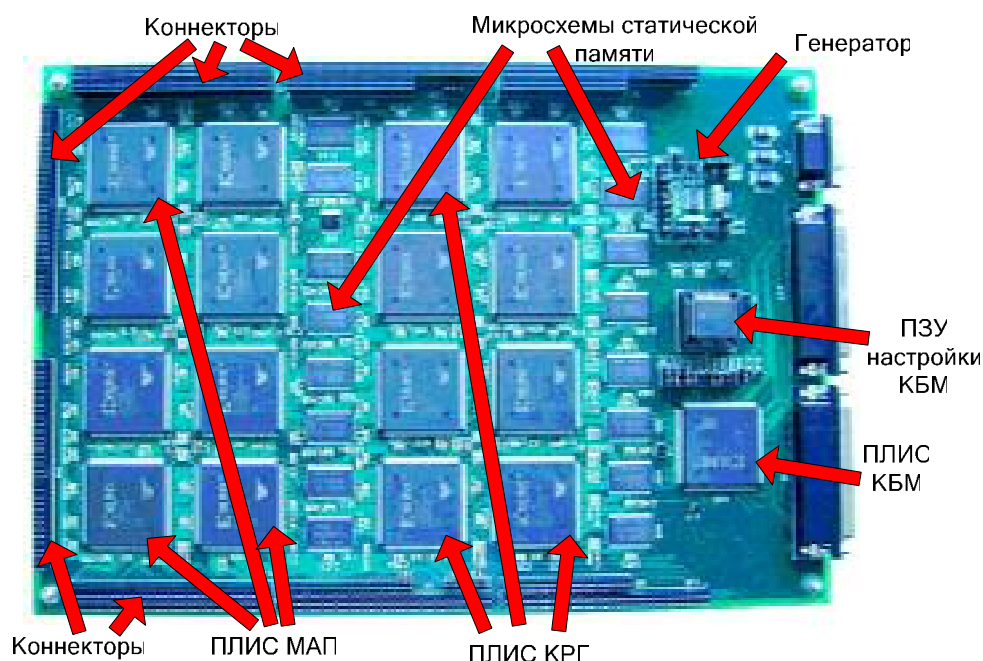


Рис. 4.15. Базовый модуль с ортогональной системой коммутации

Базовый модуль обладал приведенной производительностью $2,5 \cdot 10^{10}$ операций в секунду. При увеличении тактовой частоты по сравнению с базовым модулем первого поколения всего лишь на 25 % (с 40 МГц до 50 МГц) его производительность за счет применения более совершенных архитектурных решений выросла более чем в 150 раз. Число элементарных процессоров возросло в восемь раз. Коннекторы, установленные на базовом модуле, позволяли комплексировать до восьми базовых модулей путем непосредственного соединения без применения кабелей и какого-либо специального оборудования. Комплексированием достигался близкий к линейному рост производительности системы, пропорциональный количеству базовых модулей в пакете. Внешний вид пакета из четырех базовых модулей показан на рис. 4.16.

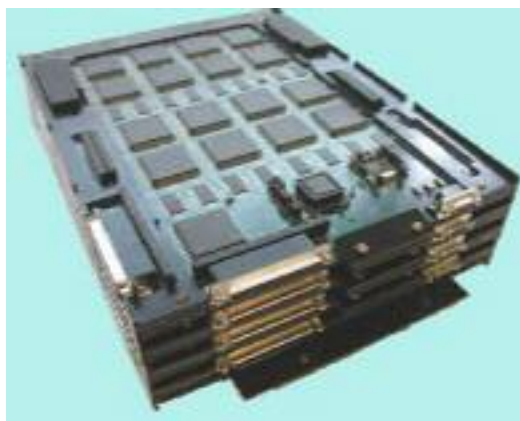


Рис. 4.16. Пакет из четырех базовых модулей второго поколения

Благодаря созданным средствам автоматического размещения базовых подграфов в схемотехническом ресурсе базового модуля сложность программирования практически не увеличилась [55]. Базовый модуль был использован для построения различных проблемно-ориентированных вычислительных систем, построенных на принципе модульного наращивания. Один и тот же базовый модуль был использован для решения разнообразных задач: криптографии, цифровой обработки сигналов, обработки изображений и т.д., для которых требовались не только принципиально различные вычислительные структуры, но и различные форматы представления данных. Форматы данных изменялись от байтного представления в задачах криптографии до форматов данных с плавающей запятой стандарта IEEE754 в алгоритмах цифровой обработки сигналов.

Следует отметить, что макропроцессоры и контроллеры распределенной памяти были жестко привязаны к конкретным микросхемам базового модуля. Каждый макропроцессор был реализован в отдельной микросхеме типа XCV300G объемом 300 тыс. эквивалентных вентилей, а для каждого контроллера распределенной памяти использовались две микросхемы XCV200G объемом 200 тыс. эквивалентных вентилей. Такая жесткая привязка к конкретным микросхемам придавала определенное неудобство при размещении базовых подграфов задачи и несколько снижала эффективность использования оборудования. Эксплуатация изделий на основе базового модуля второго

поколения показала недостаточную эффективность мультиконвейерных вычислительных структур, в которой за отдельными функциональными узлами закреплены жестко заданные функции. Поэтому в дальнейших разработках было принято решение отказаться от жесткой привязки функциональных узлов к конкретным микросхемам базового модуля.

4.4.3. Базовые модули третьего поколения

Основным отличием базовых модулей третьего поколения является использование макрообъектной архитектуры при построении реконфигурируемых структур. Макрообъектом называется архитектурно неделимая совокупность функциональных устройств, объединенных пространственной коммутационной системой, ориентированная на решение задач из конкретной предметной области и размещаемая в одной или нескольких ПЛИС решающего поля. Иными словами, макрообъект - это аппаратно-программная заготовка, которая может доопределяться пользователем в процессе создания конкретного технического решения, а затем тиражироваться в схмотехническом ресурсе ПЛИС базовых модулей в необходимом количестве и соединяться с подобными или другими макрообъектами в вычислительные структуры, которые оптимально соответствуют структуре решаемой задачи. Более подробно макрообъектная архитектура описана в главе 5.

Помимо архитектурных отличий базовые модули третьего поколения обладают и конструктивно-технологическими особенностями, которые характеризуются:

- использованием ПЛИС с интеграцией более четырех миллионов эквивалентных вентилях и частотами работы более 200 МГц;
- использованием ПЛИС со встроенными микропроцессорами типа PPC405;
- применением динамической памяти SDRAM большого объема типа DDR и DDR2 для создания распределенной памяти;

- использованием быстрых LVDS-каналов для связи с другими базовыми модулями;

- применением малогабаритных вторичных источников питания непосредственно на базовом модуле.

В НИИ МВС был разработан целый ряд базовых модулей для реконфигурируемых структур различного назначения.

4.4.3.1. Базовый модуль третьего поколения 16V4-50

В 2005 году был разработан базовый модуль третьего поколения на основе ПЛИС серии Virtex IV.

Фотография базового модуля представлена на рис. 4.17.

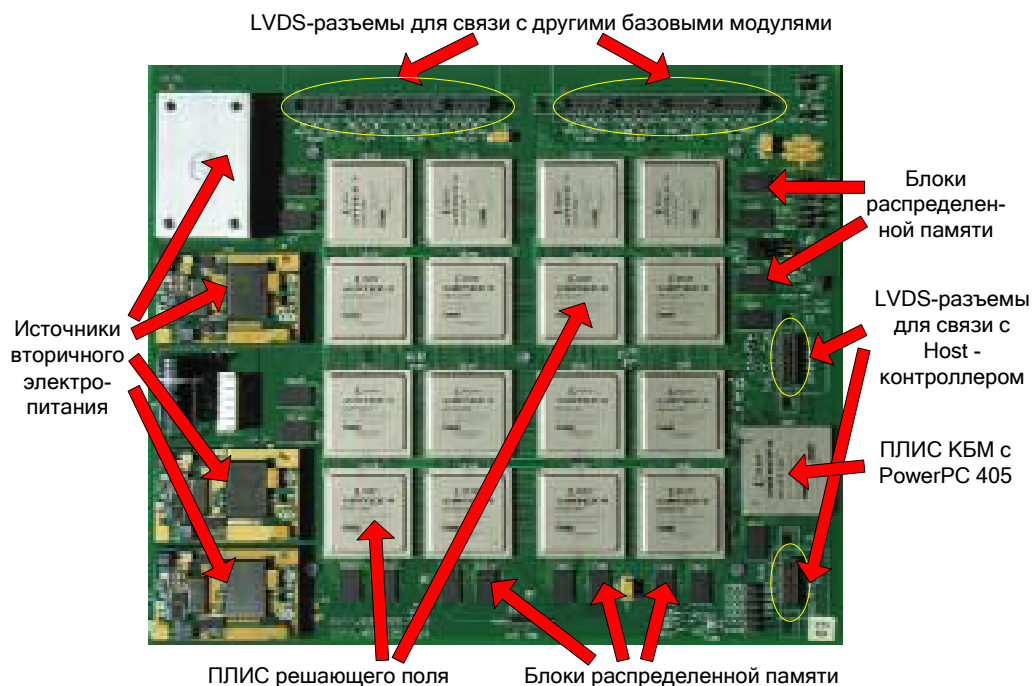


Рис. 4.17. Базовый модуль третьего поколения на основе ПЛИС серии Virtex IV

Характеристики базового модуля:

Число ПЛИС Virtex IV	17
Объем памяти, Гбайт	1
Производительность, Гфлопс	50
Рабочая частота, МГц	200
Количество каналов LVDS	112
Суммарная пропускная способность LVDS каналов, Мбит/сек	22400
Потребляемая мощность, КВт	0,16
Габариты, мм	233,35x280

Базовый модуль 16V4-50 создан на ПЛИС Virtex IV и предназначен для решения задач цифровой обработки сигналов, символьной обработки, решения задач линейной алгебры и математической физики. Каждая ПЛИС содержит 4 миллиона вентиляей. Контроллер базового модуля содержит встроенный процессор PowerPC405. Базовый модуль выполнен в конструктиве Евромеханика 6U и имеет 112 информационных LVDS-каналов для обмена данными с внешними устройствами с суммарной скоростью обмена свыше 22 Гбит/сек. Большая часть ПЛИС сопряжена с внешней распределенной памятью типа SDRAM.

На основе этих модулей создан блок модульно-наращиваемой реконфигурируемой мультиконвейерной вычислительной структуры, представленный на рис. 4.18.



Рис. 4.18. Блок модульно-наращиваемой реконфигурируемой структуры

Блок представляет собой реконфигурируемую структуру и может использоваться как автономно в виде законченного изделия, так и для построения более мощных вычислительных систем. Блок реконфигурируемой структуры имеет следующие характеристики:

Число базовых модулей	4
Объем памяти, Гбайт	4
Пиковая производительность, Гфлопс	200
Потребляемая мощность, КВт	0,9
Объем, л	39,9

Для увеличения вычислительной мощности предусмотрена возможность соединения нескольких блоков. При этом суммарная пиковая

производительность системы увеличивается пропорционально общему количеству задействованных базовых модулей. Для реализации этой возможности блоки объединяются в систему. Общий вид такой системы, реализованной в 19" стойке фирмы Schroff, представлен на рис. 4.19.

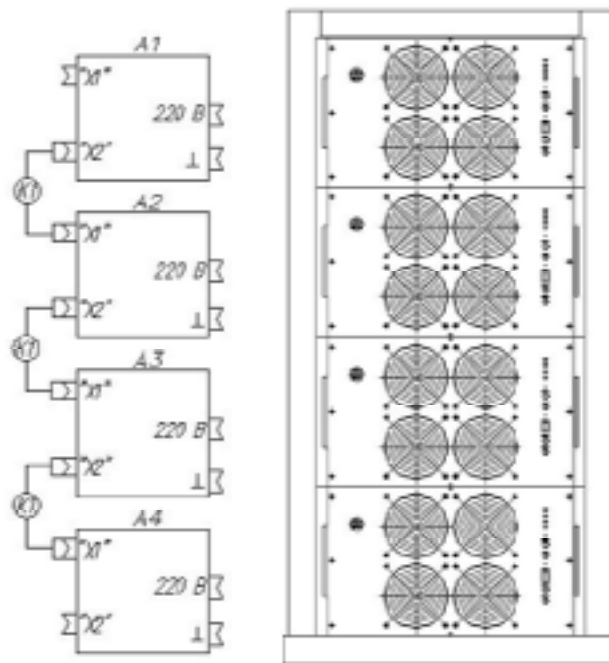


Рис. 4.19. Общий вид реконфигурируемой структуры, реализованной в 19" стойке фирмы Schroff

4.4.3.2. Базовый модуль для решения задач цифровой обработки сигналов

Базовый модуль представляет собой типовой узел реконфигурируемого вычислителя, ориентированный на реализацию структурно-процедурных вычислений, и является мощным вычислительным ресурсом с производительностью до 40 Гфлопс. Базовый модуль позволяет использовать его как автономно, для встраивания в аппаратуру различного назначения, так и для построения высокопроизводительных РМВС.

Базовый модуль имеет два конструктивных исполнения: мезонинное и двухплатное. Структура модуля в мезонинном исполнении приведена на рис. 4.20. Базовый модуль в мезонинном исполнении имеет производительность

64 Гфлопс и получил условное обозначение 8V2P-64. Для подключения мезонина 8V2P-64 используются специальные разъемы, устанавливаемые на плате базового модуля.

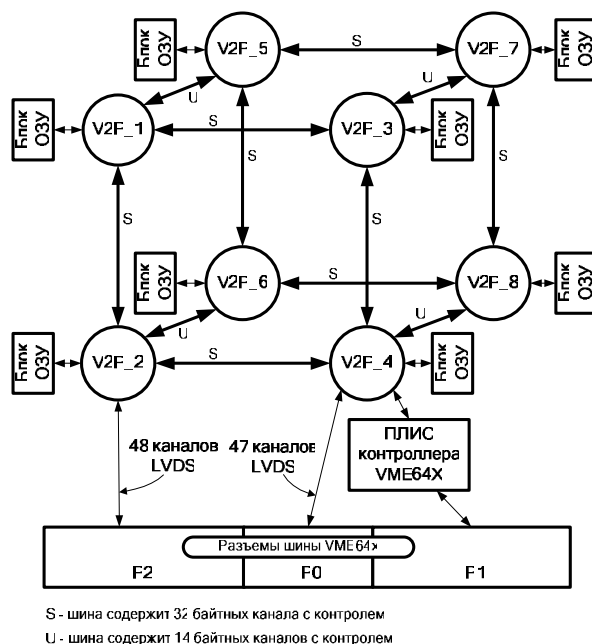


Рис. 4.20. Структура базового модуля в мезонинном исполнении 8V2P-64

Соединение базовых модулей в двухплатном исполнении и их структура показаны на рис. 4.21. Базовый модуль в двухплатном исполнении имеет производительность до 32 Гфлопс и получил условное обозначение 4V2P-32.

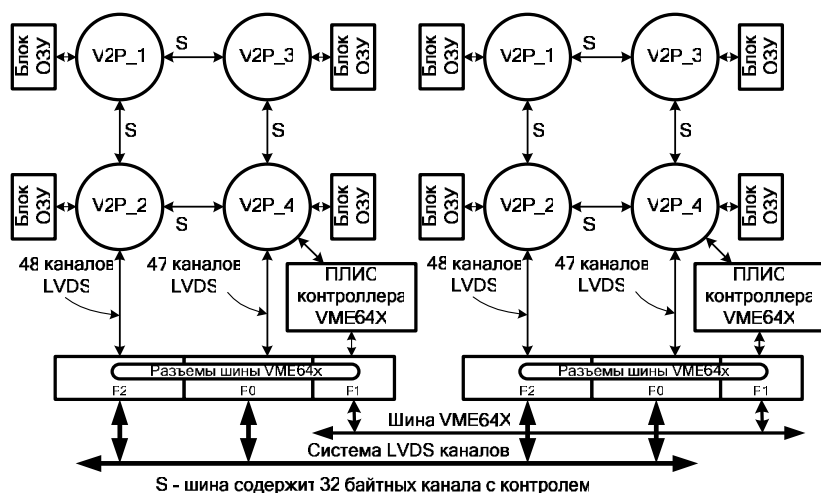


Рис. 4.21. Структура базового модуля в двухплатном исполнении

Основу базового модуля составляют восемь микросхем фирмы Xilinx Virtex II Pro типа XC2VP100-6FF1704C. Эти микросхемы, помимо десяти миллионов логических блоков FPGA, содержат по два микропроцессорных RISC ядра PPC405. Все микросхемы Virtex II Pro в пределах базового модуля соединены ортогональным коммутатором и могут обмениваться данными со скоростью до 3,2 Гбайт в секунду. К каждой микросхеме XC2VP100-6FF1704C подключено собственное ОЗУ типа SDRAM объемом 8М 64-разрядных слов. В составе базового модуля имеется контроллер шины VME64X.

В пределах одного базового модуля пользователь имеет возможность создавать на базе универсальной архитектуры множество специализированных и проблемно-ориентированных виртуальных вычислительных систем, структура которых адекватна решаемым задачам. Синтез виртуальной проблемно-ориентированной системы в рамках универсальной архитектуры осуществляется путем настройки компонентов на структурно реализуемые крупные математические операции, на процедуры макрообращений к каналам распределенной памяти и на коммутационные структуры, обеспечивающие соединения между собой вычислительных блоков и каналов распределенной памяти в соответствии с информационным графом алгоритма. В результате этого достигается адекватность проблемно-ориентированной виртуальной архитектуры структуре решаемой задачи.

Базовый модуль выполнен по стандарту «Евромеханика 6U» и предназначен для работы в составе систем мониторинга цифровой связи. Базовый модуль встраивается в блок с шиной управления VME64X. Связь с другими компонентами системы осуществляется по 95 каналам LVDS.

В базовом модуле реализован ряд подсистем:

- информационная подсистема передачи и распределения данных;
- подсистема синхронизации вычислительного процесса;
- подсистема настройки на исполняемый кадр;
- подсистема настройки конфигурации ПЛИС реконфигурируемого вычислителя;

- подсистема настройки конфигурации ПЛИС через канал JTAG;
- подсистема питания в составе двух вторичных встроенных источников напряжения +1,5В, 80А и +2,5В, 20А, получающих в качестве первичного питания напряжение +48В;
- подсистема обмена информацией с другими модулями реконфигурируемой структуры по каналам LVDS, всего 95 каналов.

Для питания вторичных источников напряжения базовый модуль подключается к первичному источнику питания SP-150-48 с выходным напряжением +48В, 3,2А.

Характеристики базового модуля 4V2P-32:

Количество ПЛИС XC2VP100	4
Объем ОЗУ, Мбайт	256
Реальная производительность, Гфлопс	32
Частота ПЛИС, МГц	200
Потребляемая мощность, Вт	85
Количество каналов LVDS	95
Скорость обмена, Гбит/сек	20
Максимальный размер БПФ	2^{17}
Габариты, мм	233,35 x 160

Внешний вид БМ 4V2P-32 с радиаторами на ПЛИС Virtex II Pro XC2VP100 показан на рис. 4.22.



Рис. 4.22. Базовый модуль 4V2P-32 с радиаторами на ПЛИС Virtex II Pro XC2VP100

В табл. 4.1 приведены сравнительные характеристики модулей цифровой обработки сигналов, выпускаемых зарубежными фирмами, а также базового модуля 4V2P-32.

Таблица 4.1

Параметр	TigerSHARC ADSP-TS101S	TigerSHARC ADSP-TS201S	PowerPC MPC7410	PowerPC MPC7455	4V2P- 32
Частота (МГц)	250	500	500	1000	200
Количество процессоров в модуле	8	8	4	4	4
Пиковая производительность модуля, Gflops	12	24	16	32	32
Шина памяти, бит/МГц	64/83,3	64/100	64/100	64/133	64/160
Количество каналов вв/выв	2PMC+16Links	2PMC+16Links	2PMC	2PMC	95 LVDS
Пиковая скорость каналов вв/выв, МБ/с	5056	9056	1056	1056	20000
Пропускная способность каналов вв/выв, Bytes/flops	0,42	0,38	0,07	0,07	0,64
Количество БПФ1024 в секунду на кристалл	25641	51282	26053	64941	90900
Количество БПФ1024 в секунду на базовый модуль	180000	360000	24000	48000	357000

Как видно из таблицы, большинство характеристик базового модуля 4V2P-32, работающего на частоте 200МГц, не уступает зарубежным образцам, выполненным на микропроцессорах TigerSHARC ADSP-TS201S PowerPC MPC7455 с частотой 500 – 1000МГц. Это достигается за счет того, что мультиконвейерные структуры, реализованные в 4V2P-32, оптимально соответствуют решаемым задачам.

4.4.3.3. Базовый модуль 4V4-25 малогабаритного ускорителя персонального компьютера

Малогабаритный ускоритель персонального компьютера предназначен для решения вычислительно трудоемких фрагментов задач различных предметных областей. Основу ускорителя составляет базовый модуль 4V4-25, выполненный на 18-слойной печатной плате размером 150x190 мм с двухсторонним расположением микросхем. Решающее поле базового модуля содержит четыре ПЛИС XC4VLX80-10FF1148, помещенных в 1148-выводных BGA-корпуса, в каждой из которых имеется по $8 \cdot 10^6$ эквивалентных вентилях. На плате базового модуля также располагаются ПЛИС контроллера XC4VFX60-10FF1152 с микропроцессором PPC405, FLASH-память для хранения конфигурационных данных контроллера, 9 микросхем динамической памяти SDRAM, программируемый тактовый генератор с драйверами, усилители, четыре малогабаритных преобразователя напряжения, один LVDS-разъем и один Ethernet-разъем для соединения с персональным компьютером и другие элементы поверхностного монтажа.

Внешний вид базового модуля 4V4-25 показан на рис. 4.23.

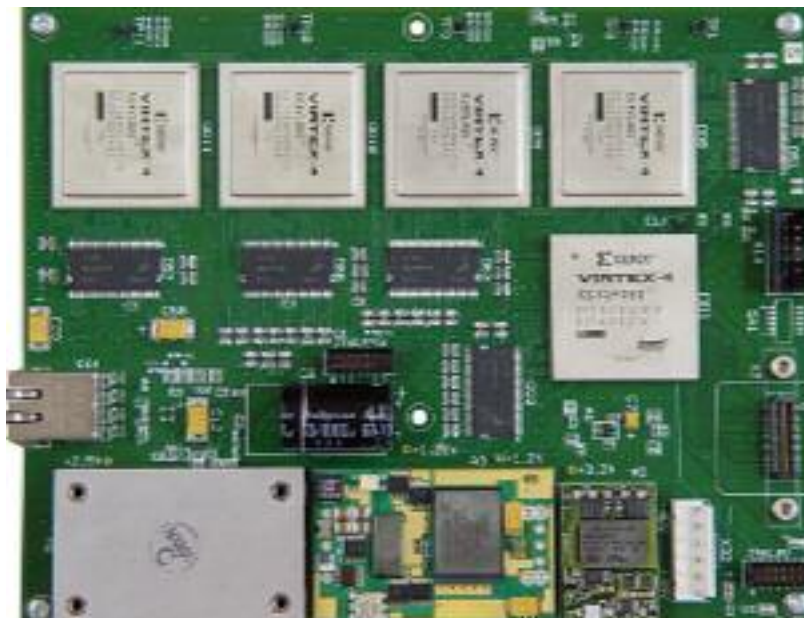


Рис. 4.23. Внешний вид базового модуля 4V4-25

Приведем технические характеристики базового модуля:

Количество ПЛИС решающего поля,шт	4
Число эквивалентных вентилей, млн.	32
Объем динамической памяти, Мбайт	576
Производительность, Гфлопс	25
Частота платы модуля, МГц	160
Скорость обмена данными по каналу Ethernet, Гбит/с	1
Потребляемая мощность, Вт	145
Размеры базового модуля, мм.	150×190

На рис. 4.24 представлена общая структурная схема базового модуля.

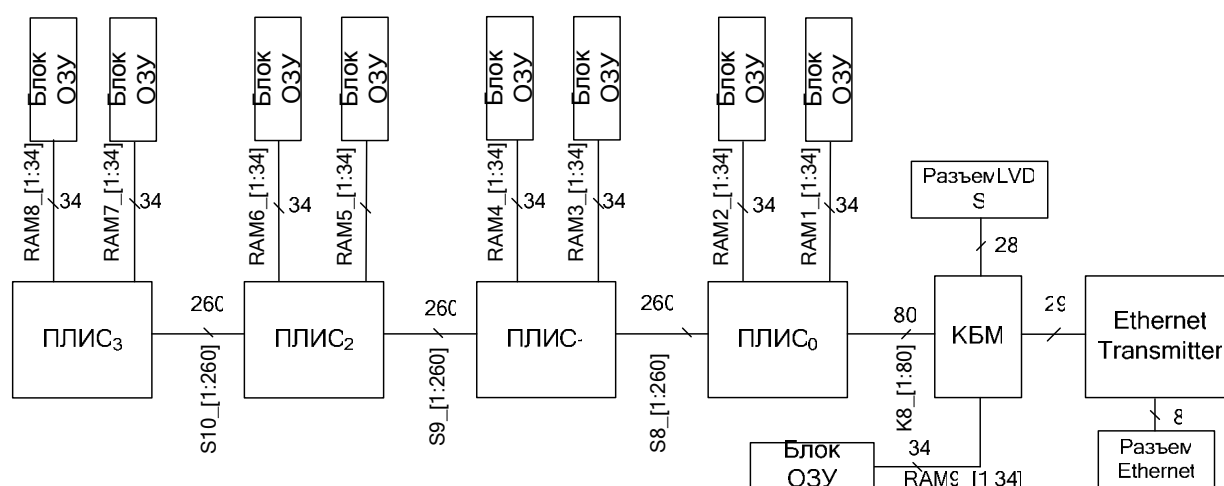


Рис. 4.24. Общая структурная схема базового модуля 4V4-25

На схеме приняты следующие обозначения: КБМ - контроллер базового модуля; Ethernet-Transmitter – приемо-передатчик Ethernet (микросхема Marvell 8E1111).

Контроллер базового модуля предназначен для информационного обмена с персональным компьютером. Контроллер обеспечивает связь между персональным компьютером и микросхемами решающего поля базового модуля по каналу Ethernet или LVDS, а также реализует интерфейс загрузки конфигурации ПЛИС. Следует учитывать, что LVDS-канал предназначен для отладочных целей, является вспомогательным и используется только для наладки изделия.

Внешний вид малогабаритного ускорителя персонального компьютера в корпусе со снятой верхней крышкой показан на рис. 4.25. На микросхемах решающего поля и ПЛИС КБМ установлены разновысотные радиаторы для

отвода тепла. Габаритные размеры корпуса ускорителя персонального компьютера составляют 300x263x55мм.



Рис. 4.25. Малогабаритный ускоритель персонального компьютера

4.4.4. Базовые модули перспективных разработок

В настоящее время в НИИ МВС (г. Таганрог) разрабатываются базовые модули для семейства реконфигурируемых вычислительных структур нового поколения на основе ПЛИС Xilinx серии Virtex 5, которые должны обеспечить рост производительности на 30-40% по сравнению с базовыми модулями третьего поколения [33].

Будет создано семейство реконфигурируемых вычислительных структур:

- РВС-5 – высокопроизводительные реконфигурируемые структуры производительностью более 6 Тфлопс;
- РВС-1Р – реконфигурируемые структуры производительностью более 1 Тфлопс;
- РВС-0.2-РС – рабочая станция производительностью более 200 ГФлопс;
- РУПК-50 и РУПК-25 – ускорители персональных компьютеров производительностью 50 и 25 ГФлопс.

Семейство реконфигурируемых структур будет ориентировано на решение вычислительно трудоемких научно-технических задач в таких областях как символьная обработка информации, защита компьютерных сетей, создание принципиально новых лекарственных препаратов и материалов нового поколения, управление в реальном масштабе времени объектами ядерной энергетики, летательными и космическими аппаратами, расшифровка структуры белка, оптимизация эксплуатации нефтяных месторождений, томографические исследования приповерхностных слоев Земли акустическими и электромагнитными волнами, нейроматематика.

4.4.4.1. Структура базового модуля высокопроизводительных систем

Семейство реконфигурируемых структур строится всего из трех типов базовых модулей: 16V5-75, 16V5-50 и 16S3-25. Базовый модуль 16V5-75, как самый производительный, используется в моделях РВС-5, РВС-1Р и РВС-0.2-РС. Базовые модули 16V5-50 и 16S3-25 входят в состав ускорителей персональных компьютеров РУПК-50 и РУПК-25.

Общие принципы построения базовых модулей одинаковы и поэтому рассмотрим более подробно базовый модуль 16V5-75, который обладает следующими характеристиками:

Производительность (64 разряда), Гфлопс	75
Производительность (32 разряда), Гфлопс	140
Потребляемая мощность, ВА	200
Объем оперативной распределенной памяти, Мбайт	1,25
ПЛИС решающего поля XC5VLX110, шт.	16
Количество эквивалентных вентилях в ПЛИС, шт.	$8 \cdot 10^6$
Тактовая частота, МГц	250
Количество LVDS каналов, шт.	224
Скорость межмодульного обмена, Гбит/сек	22

Решающее поле базового модуля 16V5-75 выполнено на 16-ти ПЛИС Virtex 5 XC5VLX110-2FF1153 фирмы Xilinx. Микросхемы содержат около $8 \cdot 10^6$ эквивалентных вентилях. Микросхемы расположены в узлах двумерной решетки 4 x 4 и соединены между собой ортогональной системой связей по близкодействию. Такая система связей позволяет существенно упростить печатную плату и улучшить ее частотные характеристики, поскольку связи между соседними микросхемами не превышают четыре сантиметра. Данные между дальними микросхемами передаются по транзитным каналам через промежуточные микросхемы, используя систему ортогональных связей.

Структура базового модуля 16V5-75 представлена на рис.4.27.

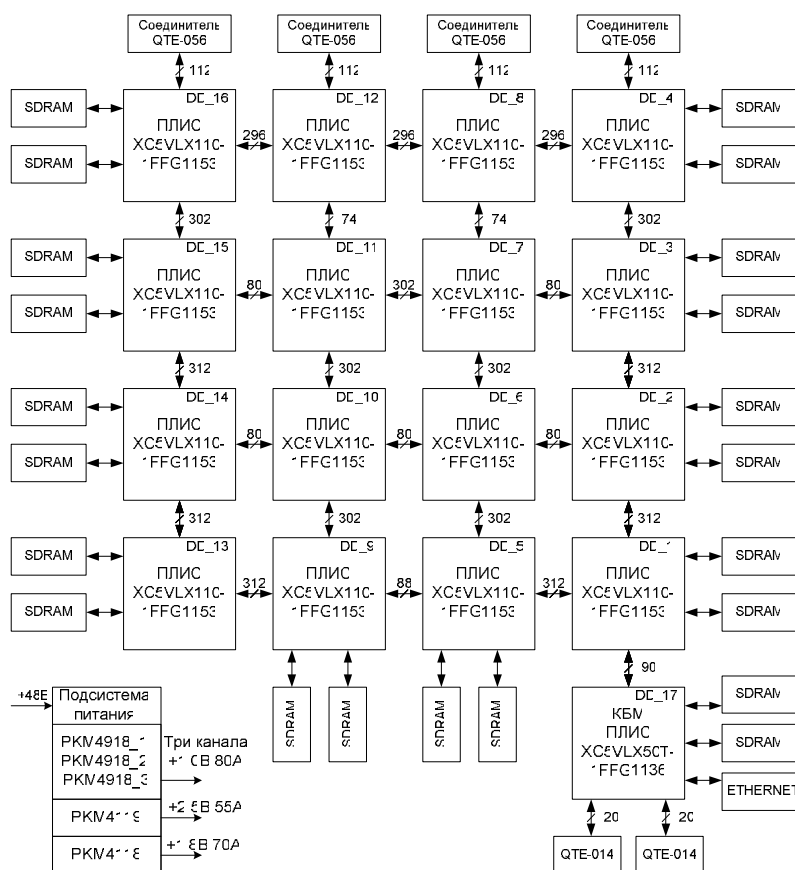


Рис. 4.27. Структура базового модуля 16V5-75

Внешний вид базового модуля 16V5-75 показан на рис. 4.28.



Рис. 4.28. Внешний вид базового модуля 16V5-75

Связи между ПЛИС решающего поля базового модуля реализованы на основе стандарта LVDS. Преимуществами стандарта LVDS являются низкая потребляемая мощность выходных каскадов, низкий уровень создаваемых электромагнитных излучений, невосприимчивость к синфазным электромагнитным помехам и наличие в микросхемах семейства Virtex V аппаратной поддержки для организации высокоскоростных передач данных на основе стандарта LVDS.

Базовый модуль имеет подсистему обмена информацией по LVDS-каналам с другими базовыми модулями. Подсистема обмена информацией предназначена для передачи промежуточных результатов вычислений из микросхем данного модуля в микросхемы решающего поля других базовых модулей. Передача осуществляется посредством специальных кабелей, подключаемых к соединителям типа QTE-056. Единый тип интерфейса между ПЛИС как в пределах одного базового модуля, так и между различными базовыми модулями, обеспечивает одинаковые скорости обмена и снижает эффект границ в пределах общего схемотехнического ресурса реконфигурируемой структуры.

К периферийным микросхемам решающего поля, расположенным на краях двумерной решетки 4x4, подключены микросхемы динамических ОЗУ SDRAM типа DDR2 (Double Data Rate Two Synchronous Dynamic Random

Access Memory), которые образуют распределенную память базового модуля 16V5-75. В состав распределенной памяти входят 20 микросхем MT47H32M16BN фирмы Micron Technology Inc. Каждая микросхема имеет структуру 2^{25} 16-разрядных слов. Общий объем распределенной памяти составляет 1,25 Гбайт. Кроме этого, на базовом модуле установлено еще 128 Мбайт памяти, подключенной к контроллеру базового модуля. Распределенная память предназначена для хранения входных данных, промежуточных и конечных результатов обработки информации. В этой же памяти хранятся фрагменты параллельных программ, ответственных за организацию работы контроллеров распределенной памяти, реализованных в микросхемах решающего поля. Контроллеры распределенной памяти позволяют организовывать множество синхронных потоков данных в решающем поле базового модуля.

Основным назначением контроллера базового модуля являются функции управления всеми его подсистемами, а также передача информации между базовым модулем и управляющим контроллером (ЭВМ типа IBM PC). Помимо этого контроллер базового модуля выполняет загрузку конфигурации ПЛИС решающего поля, участвует в диагностике устройств, загружает входные данные в распределенную память, выгружает результаты обработки, настраивает подсистему синхронизации, собирает и передает информацию о текущем состоянии базового модуля в управляющий контроллер. Контроллер базового модуля выполнен на ПЛИС Virtex V XC5VLX50T-1FFG1136 фирмы Xilinx. Настройка конфигурации этой микросхемы осуществляется по включении питания из энергонезависимой памяти, в которой хранится конфигурационный файл контроллера. Связь контроллера базового модуля с управляющим контроллером осуществляется посредством LVDS-каналов через два соединителя QTE-014, а также с помощью канала Ethernet.

Функция контроля температуры ПЛИС решающего поля осуществляется контроллером базового модуля путем опроса аппаратных блоков System

Monitor, измеряющих температуру кристаллов, которые встроены в ПЛИС семейства Virtex V.

Подсистема электропитания расположена непосредственно на базовом модуле и содержит портативные преобразователи напряжения серии РКМ фирмы Ericsson, на которые подается напряжение +48В от источника первичного питания. При этом ток питания, подводимый к базовому модулю, существенно уменьшается, что приводит к уменьшению сечения подводящих проводников. Для отвода тепла и поддержания необходимых температурных режимов базового модуля предусмотрена комбинированная система охлаждения, которая включает радиаторы на ПЛИС решающего поля и вентиляторы для их обдува.

Таким образом, базовый модуль 16V5-75 представляет собой мощную мультиконвейерную вычислительную структуру производительностью свыше 75 Гфлопс. На основе 16V5-75 могут строиться вычислительные блоки, содержащие от одного до восьми базовых модулей производительностью от 75 до 600 Гфлопс. В то же время базовый модуль обладает достаточной автономностью и может легко комплексоваться с персональным компьютером типа IBM PC в качестве ускорителя и использоваться при решении различных задач.

Практика реальной эксплуатации реконфигурируемых структур, построенных по модульному принципу с использованием базовых модулей на основе ПЛИС, показала что они обладают высокой реальной производительностью, близкой к пиковой производительности, на различных классах решаемых задач и имеют практически линейный рост производительности в зависимости от увеличения числа базовых модулей. При успешном развитии автоматизированных средств проектирования и программирования мультиконвейерных вычислительных структур открываются широкие возможности создания на их основе ускорителей персональных компьютеров и суперкомпьютеров с технико-экономическими

характеристиками, значительно превышающими характеристики существующих кластерных суперЭВМ.

Глава 5. РЕКОНФИГУРИРУЕМЫЕ МУЛЬТИКОНВЕЙЕРНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СТРУКТУРЫ С МАКРООБЪЕКТНОЙ АРХИТЕКТУРОЙ

5.1. Принципы построения реконфигурируемых вычислительных структур на основе макрообъектной архитектуры

Как показано в предыдущих разделах, основой реконфигурируемых мультиконвейерных структур являются макропроцессорные узлы, с помощью которых формируется аппаратный аналог кадра решаемой задачи. В свою очередь, макропроцессор представляет собой некоторый набор элементарных процессоров, объединяемых в единый вычислительный ресурс с помощью коммутатора. При этом понималось, что все элементарные процессоры одинаковы и представляют собой арифметико-логические устройства, реализующие некоторый набор элементарных многоразрядных операций.

Понятно, что использование универсальных однотипных элементарных процессоров (ЭП) приводит к избыточности аппаратных средств реконфигурируемой вычислительной структуры. Действительно, несмотря на то, какую конкретную операцию элементарный процессор должен реализовывать в текущий момент времени, он должен содержать аппаратные средства, обеспечивающие реализацию всего множества элементарных операций. Последнее, в свою очередь, ведет к снижению удельной производительности реконфигурируемой вычислительной структуры.

Частично данную проблему можно решить путем специализации элементарных процессоров. Например, в составе макропроцессора можно использовать элементарные процессоры двух типов – *аддитивный элементарный процессор* (АЭП) и *мультипликативный элементарный процессор* (МЭП), которые представлены на рис. 5.1. Аддитивный элементарный процессор предназначен для выполнения таких операций как сложение, вычитание, сравнение, а мультипликативный процессор может быть

задействован для выполнения операций умножения и деления. Кроме того, каждый из типов элементарных процессоров должен иметь возможность выполнения простейших логических и потоковых операций.

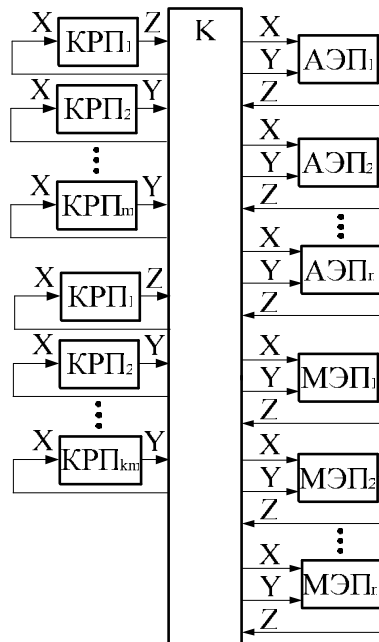


Рис. 5.1. Макропроцессор с двумя типами процессорных элементов

Конечно, трудно представить себе вычислительную задачу, которая состоит исключительно из операций сложения или операций умножения. Поэтому, как правило, в макропроцессоре элементарные процессоры обоих типов должны быть использованы равномерно. Даже если встретится вычислительная задача, использующая исключительно операции одного класса, например, сложение элементов массива, то удельная производительность макропроцессора, содержащего элементарные процессоры различных типов, останется при реализации данной структуры практически такой же, как и удельная производительность макропроцессора, построенного на основе универсальных элементарных процессоров. В первом случае будут простаивать аддитивные элементарные процессоры, во втором случае не будут использованы аппаратные подсистемы универсальных элементарных процессоров, реализующих операции умножения. В случае же равномерного использования операций сложения и умножения удельная производительность макропроцессора, содержащего элементарные процессоры двух типов, возрастет практически в два раза.

Разделение функциональных устройств на более широкий спектр объектов позволит еще более повысить удельную производительность макропроцессора и, как следствие, РМВС в целом, однако нет необходимости в предельной дифференциации функций элементарных процессоров, потому что неизбежной окажется либо простой оборудования при реализации большинства задач, либо потребуется перестройка структуры элементарных процессоров.

Реализация базовых модулей реконфигурируемых структур на основе ПЛИС позволила по-новому взглянуть на данную проблему и предложить подход к ее решению.

Как показано в предыдущей главе, базовый модуль реконфигурируемых вычислительных структур представляет собой некоторое множество ПЛИС, объединенных в одно решающее поле. Иными словами, базовый модуль можно рассматривать как некий аналог суперПЛИС или ПЛИС второго уровня. Например, в базовом модуле, включающем 20 ПЛИС с интеграцией $(8-10) \cdot 10^6$ эквивалентных вентилях, может содержаться до $1,6 \cdot 10^6$ логических блоков, 6-8 тысяч блоков памяти и около 1500 устройств умножения, из которых пользователь может создавать любые вычислительные структуры.

В реконфигурируемых структурах, содержащих десятки базовых модулей, количество логических блоков, потенциально доступных пользователю, достигает нескольких десятков миллионов. Имея в своем распоряжении огромный схемотехнический ресурс, пользователь может реализовать практически любые вычислительные узлы с любыми техническими параметрами. С одной стороны это позволяет не сдерживать разработчика в его стремлении к различным формам реализации, но, с другой стороны, усложняет формализацию процесса разработки.

В рамках такого огромного аппаратного ресурса пользователь, опираясь на самый низкий схемотехнический уровень ПЛИС – логические блоки, блоки памяти, умножители и т. п., с помощью средств САПР ПЛИС может создавать различные вычислительные структуры, реализующие кадры решаемой задачи. Понятно, что такое решение будет оптимальным с точки зрения затрат

аппаратного ресурса, поскольку предполагает для реализации каждого кадра решаемой задачи создание специализированной вычислительной структуры в рамках поля ПЛИС базового модуля. Это обстоятельство делает чрезвычайно привлекательным программирование архитектур базового модуля на схемотехническом уровне ПЛИС.

Однако такой подход имеет ряд недостатков, ограничивающих возможности его практического использования. Во-первых, сам процесс программирования большого поля ПЛИС – очень сложный и трудоемкий процесс, требующий высокой квалификации пользователя и специальных средств программирования, поскольку стандартные средства САПР ПЛИС позволяют программировать схемотехнику только одиночных ПЛИС. Можно, конечно, использовать библиотечные элементы (IP-ядра) отдельных вычислительных элементов при программировании ПЛИС, но такой подход приведет к снижению оптимальности вычислительной структуры и не намного уменьшит трудоемкость разработки.

Наиболее важный недостаток такого подхода заключается в следующем. При структурно-процедурной организации вычислений, когда задача разбивается на последовательность кадров, необходимо обеспечить быструю перестройку базового модуля реконфигурируемой структуры на реализацию следующего кадра последовательности. Процесс же реконфигурирования ПЛИС – очень длительный процесс, занимающий десятки секунд [50]. В результате, длительная смена кадров в вычислительном поле ПЛИС базового модуля, даже в случае заранее подготовленных файлов конфигурации ПЛИС, сведет на нет все преимущества в высокой производительности мультимониторных вычислений. Поэтому чисто схемотехническая реализация кадра решаемой задачи в поле ПЛИС базового модуля целесообразна только в случае однокадровых задач, т.е. задач, информационный граф которых полностью помещается в аппаратный ресурс поля ПЛИС базовых модулей. Этот подход неприменим, если аппаратного ресурса реконфигурируемой

структуры не хватает для реализации всего информационного графа, и необходимо использование метода структурно-процедурных вычислений.

В этом случае целесообразно использование комбинированного подхода, основная идея которого заключается в программировании архитектуры базового модуля на двух уровнях – схмотехническом и системном. На схмотехническом уровне пользователь, используя возможности САПР ПЛИС, может создать специализированные функциональные узлы, ориентированные на проблемную область решаемой задачи, а также коммутационные средства, обеспечивающие возможности их объединения в мультиконвейерные структуры [51]. Иными словами, на схмотехническом уровне создаются проблемно-ориентированные «кубики», с помощью которых можно создавать мультиконвейерные структуры, реализующие кадры прикладных задач некоторой предметной области. Далее, в процессе решения конкретной задачи из данной предметной области осуществляется оперативное системное программирование базового модуля путем объединения «кубиков» в мультиконвейерные структуры, реализующие кадры задачи, с помощью коммутационных средств. Понятно, что при таком подходе эффективность синтезируемых мультиконвейерных структур будет несколько ниже, чем при чисто схмотехнической реализации, но стоимость и время ее разработки будут существенно меньше за счет заранее подготовленных функциональных узлов, а главное, реконфигурирование на реализацию различных кадров будет зависеть только от времени срабатывания коммутационных средств.

В то же время схмотехническое программирование вычислительных элементов не исключает программирование в традиционном смысле этого слова. Необходимость создания процедур для отдельных вычислительных элементов обосновывается экономией ресурсов системы, поскольку их аппаратная реализация может быть очень громоздкой. Как правило, такие элементы реализуют специфические вычислительные фрагменты задач. К подобным фрагментам относятся условные операторы, прежде всего многократно вложенные, так как для их структурной реализации необходимо

выполнять все альтернативные ветви, а не единственную ветвь, как при процедурных вычислениях. Также структурно неэффективно выполняются рекурсивные вычисления с глубокими обратными связями и вычислительные фрагменты, содержащие множество пересылок информации.

Опыт разработки реконфигурируемых структур на базе ПЛИС показывает, что для каждого класса задач можно подобрать определенный набор функциональных узлов, с помощью которых создаются оптимальные мультиконвейерные вычислительные структуры, наиболее эффективно решающие задачи данного класса. При этом наибольшая эффективность структур достигается в том случае, если вычислительные элементы располагаются в объединенном поле логических блоков всего множества ПЛИС базового модуля или в поле логических блоков всего множества ПЛИС всей вычислительной структуры. Иными словами, для создания оптимальных вычислительных структур не должно существовать границ микросхем и границ базовых модулей. Так как физические границы все же существуют, то необходимо так или иначе снизить негативный эффект от их наличия.

Для реализации предложенного выше подхода к программированию реконфигурируемых вычислительных структур на базе ПЛИС введем понятие макрообъекта.

Макрообъектом будем называть архитектурно неделимую совокупность функциональных узлов (объектов), объединенных пространственной коммутационной системой и размещенных в одной или нескольких ПЛИС. При этом для макрообъекта допустимо изменение количества функциональных узлов того или иного типа, параметров узлов (разрядности операндов, числа информационных каналов, системы команд и т.п.), но не их назначения.

Каждый макрообъект может содержать множество разнородных вычислительных функциональных узлов, объединенных между собой некоторой внутренней пространственной коммутационной структурой, которая позволяет соединить элементы в функционально законченное устройство, а также обеспечивает транзит информации между различными макрообъектами.

Кроме того, каждый макрообъект содержит интерфейсный узел для обеспечения корректного взаимодействия между объектами, принадлежащими различным макрообъектам.

Для того чтобы макрообъект мог использоваться при программировании различных мультиконвейерных структур в рамках реконфигурируемой структуры, необходимо заранее, на этапе проектирования, выполнить следующие условия:

- 1) определить номенклатуру функциональных узлов, которые допустимо использовать при создании макрообъектов для различных предметных областей;

- 2) определить параметры функциональных узлов, которые могут изменяться в процессе конструирования макрообъекта. К таким изменяемым параметрам блоков можно отнести: разрядность, количество входов/выходов, типы внешних интерфейсов, систему команд функциональных узлов и т.п.;

- 3) разработать библиотеки функциональных узлов на схемотехническом уровне, которые фактически являются IP-ядрами, созданными в терминах логических схем ПЛИС или в терминах языка описания логических схем VHDL;

- 4) создать набор типовых интерфейсов, посредством которых можно соединить функциональные узлы в мультиконвейерные вычислительные структуры согласно информационному графу алгоритму и которые также являются IP-ядрами;

- 5) определить структуры макрообъектов для различных предметных областей.

Здесь и далее под IP-ядрами понимаются заранее подготовленные схемотехнические решения законченных функциональных узлов, представленные в виде программных блоков языка описания логических схем VHDL, которые пользователь может использовать в качестве заготовок при схемотехническом программировании ПЛИС.

Таким образом, макрообъект является некоторой заготовкой, которая может доопределяться пользователем в процессе создания конкретного технического решения, а затем тиражироваться в схемотехническом ресурсе ПЛИС базовых модулей в необходимом количестве и соединяться с подобными или другими макрообъектами в мультиконвейерные вычислительные структуры, которые оптимально соответствуют структуре решаемой задачи. Можно сформулировать и другое определение макрообъекта, которое отражает его суть с точки зрения программиста реконфигурируемой системы. *Макрообъект* – это виртуальное проблемно-ориентированное устройство с изменяемыми параметрами, которое является заготовкой при программировании архитектуры реконфигурируемой структуры и может быть реализовано в поле логических блоков ПЛИС базового модуля.

В главе описываются примеры создания макрообъектов для таких проблемных областей как математическая физика, цифровая обработка сигналов, линейная алгебра. На этих примерах читатель может составить более развернутое представление о макрообъектной архитектуре реконфигурируемых систем. Кроме того, в конце главы приводится описание так называемого универсального макрообъекта, с помощью которого могут решаться задачи различных проблемных областей. Понятно, что универсальный макрообъект не обладает эффективностью проблемно-ориентированных макрообъектов при построении мультиконвейерных вычислительных структур. Однако такой макрообъект обладает и рядом положительных свойств:

- универсальный макрообъект располагается в одной микросхеме ПЛИС;
- структура макрообъекта не изменяется в зависимости от размещения в ПЛИС базового модуля;
- для этого макрообъекта, как правило, имеются формальные методы представления задачи в структурно-процедурной форме.

Перечисленные свойства существенно облегчают программирование тем пользователям, которые недостаточно глубоко владеют другими, более эффективными способами программирования реконфигурируемых систем.

5.2. Обобщенная структура макрообъекта

Рассмотрим обобщенную структуру макрообъекта. Проведенные исследования показали, что в состав макрообъектов, создаваемых для различных предметных областей, как правило, входят следующие объекты: элементарный процессор (ЭП), контроллер распределенной памяти (КРП), преобразователь информации (ПИ), динамический коммутатор (ДК), блоки памяти различного назначения (П), процедурное АЛУ (ПАЛУ) и локальное устройство управления (ЛУУ). Структура унифицированного макрообъекта представлена на рис. 5.2.

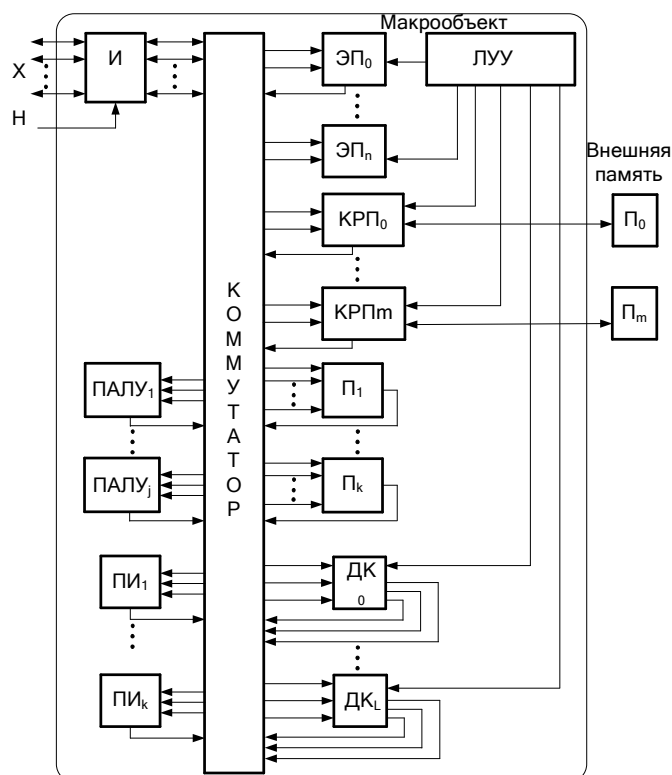


Рис. 5.2. Структура унифицированного макрообъекта

Элементарный процессор осуществляет обработку пары операндов в соответствии с кодом операции, а также обеспечивает синхронизацию потоков операндов и реализацию операций с константой.

Процедурное АЛУ выполняет те вычисления, которые неэффективно реализовывать структурно. При этом их число должно быть настолько велико,

чтобы обеспечить обработку элементов информационного потока, выходящих (входящих) из структурных компонентов макрообъектов. Число процедурных АЛУ может достигать нескольких сотен, что требует существенного сокращения системы команд подобных элементов и, как следствие, уменьшение аппаратных затрат на их реализацию. Предполагается, что процедурные АЛУ будут проектироваться для каждой задачи или, по крайней мере, для проблемной области.

Преобразователь информации обеспечивает реализацию фиксированной функции. Следует отметить, что функциональное преобразование может быть выполнено над группой операндов. В качестве примеров можно рассматривать скалярное произведение векторов, численное интегрирование, умножение матриц, решение системы логических уравнений. При этом в преобразователе информации синхронизация информационных потоков, как правило, не обеспечивается.

Совокупность элементарных процессоров, настроенных на выполнение определенных арифметико-логических операций и преобразователей информации, входы которых соединены внутренней пространственной коммутационной структурой в соответствии с информационным графом решаемой задачи, образуют мультиконвейерную вычислительную структуру, реализующую обработку информационных массивов в соответствии с алгоритмом решения задачи или ее фрагмента. В вычислительную структуру могут быть включены дополнительные узлы и блоки, например, буферы данных для синхронизации информационных потоков или компараторы-переключатели.

Информационные потоки в мультиконвейерной вычислительной структуре образуются с помощью контроллеров распределенной памяти, которые организуют скоростные параллельные обращения к сегментированной внутренней и внешней памяти макрообъекта. Внутренняя память контроллера памяти подразделяется на ряд сегментов, каждый из которых имеет определенное функциональное назначение. Сегмент операторов содержит

управляющую программу. Сегмент параметров содержит параметры операторов группового обращения к блокам распределенной памяти. Параметры передаются и обрабатываются адресными процессорами. Внутренняя память контроллера содержит также сегмент макрокоманд, предназначенный для настройки вычислительных блоков. В сегмент макрокоманд входят команды элементарных процессоров и функциональных узлов, а также адреса статических коммутаторов и операции динамических коммутаторов. Для реализации подпрограмм во внутреннюю память контроллера включен сегмент стека, содержащий адреса вызова подпрограмм и слова состояния контроллера распределенной памяти.

Динамические коммутаторы обеспечивают переключение информационных потоков между источниками и приемниками, причем изменение адреса источника (приемника) может производиться произвольно с каждым данным. Совокупность статических и динамических коммутаторов обеспечивает возможность создания разнообразных вычислительных структур для решения различных задач из проблемной области, для которой создавался данный макрообъект.

На рис. 5.3 приведен вариант структурной схемы динамического коммутатора.

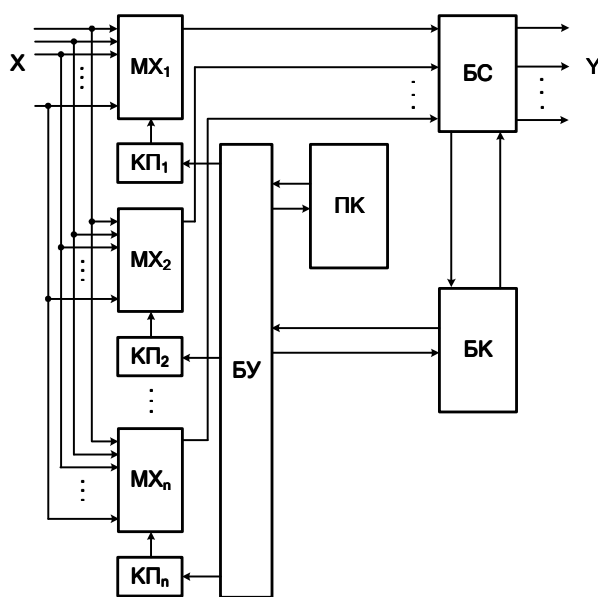


Рис. 5.3. Структурная схема динамического коммутатора

В состав динамического коммутатора входят мультиплексоры. Каждый мультиплексор (МХ) управляется коммутационным процессором (КП), который функционирует по независимой программе. Для синхронизации работы множества коммутационных процессов вводится блок управления (БУ), который функционирует по единой управляющей программе, хранящейся в памяти команд (ПК).

Блок динамической коммутации может осуществлять контроль информационных потоков при реализации режимов резервирования. Анализ информационных потоков выполняет блок контроля (БК).

В макрообъекте имеется локальное устройство управления, которое управляет архитектурой перестраиваемого параллельного конвейера и делает систему практически универсальной. В качестве устройства управления предлагается использовать универсальные микропроцессоры Microblaze или Power PC. Данные процессоры реализуют функции распределенной операционной системы, обеспечивают загрузку информации в макрообъекты и, тем самым, устраняют проблему «узкого горла» связи между host-машиной и множеством макрообъектов.

5.3. Вычислительные блоки макрообъектов

5.3.1. Представление данных в макропроцессоре

При создании функциональных узлов для системы с макрообъектной архитектурой используются разнообразные форматы данных. Однако наибольший интерес для эффективных аппаратных реализаций широкого применения представляет обработка данных в формате с плавающей запятой в стандарте IEEE-754 с одинарной и двойной точностью. Стандарт IEEE-754 устанавливает единое представление и способы обработки информации для чисел с плавающей запятой в современных компьютерах.

Для определенности рассмотрим реализацию функциональных узлов, использующих формат представления чисел с плавающей запятой. Формат

представления чисел с плавающей запятой представлен 32-битными числами – 1 бит для знака мантиссы, 8 бит – для порядка, 23 бита – для мантиссы. Формат предполагает наличие «скрытого» (старшего) бита мантиссы, так что мантисса на самом деле представлена 24 битами.

5.3.2. Обобщенная структурная схема функциональных узлов

с плавающей запятой

Обобщенная структурная схема функциональных узлов с плавающей запятой, работающих в соответствии со стандартом IEEE-754, представлена на рис. 5.4 и состоит из трех блоков: блока контроля входных операндов, блока выполнения арифметических операций и блока формирования результата.



Рис. 5.4. Обобщенная структурная схема функциональных узлов с плавающей запятой

Блок контроля входных операндов анализирует и преобразует входные данные в соответствии со стандартом IEEE-754. Так как вычисления выполняются над 24-разрядными мантиссами, а на входы поступают 23-разрядные мантиссы, то старший разряд нормализованной мантиссы необходимо восстановить. Суть восстановления заключается в следующем: при поступлении нормализованного числа, о чем свидетельствует его порядок, не равный нулю, старший разряд мантиссы восстанавливается как единица. Если число денормализовано и порядок числа равен нулю, то старший разряд мантиссы устанавливается в ноль. Денормализованные числа могут быть результатом операций сложения, вычитания, умножения.

Блок арифметических операций выполняет заданную операцию с плавающей запятой и формирует признаки переполнения порядка. Блок

формирования результата преобразует результат в соответствии со стандартом IEEE-754.

5.3.3. Функциональный узел сложения чисел с плавающей запятой

Сложение и вычитание чисел с плавающей запятой осуществляется функциональным узлом, выполненным в виде IP-ядра. Упрощенная структура функционального узла сложения чисел показана на рис. 5.5.

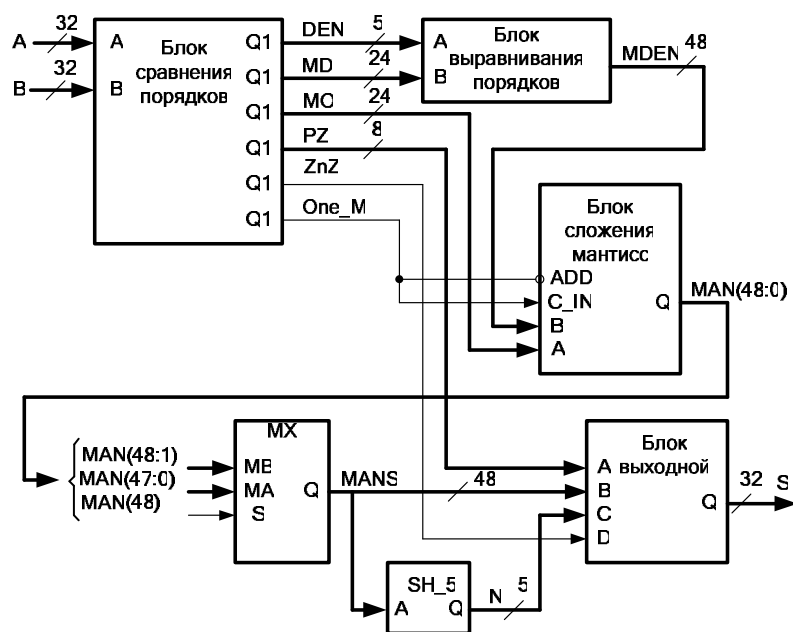


Рис. 5.5. Структурная схема сумматора SMFP 32

На вход сумматора параллельно поступают операнды A, B с учетом восстановления скрытой единицы. Согласно алгоритму сложения (вычитания) чисел с плавающей запятой в блоке выравнивания порядков вычисляется разность порядков DEN, при этом по знаку разности порядков в соответствии с алгоритмом выбираются мантиссы с большим порядком MO и с меньшим порядком MD и порядок результата PS.

Операция выравнивания порядков мантисс заключается в денормализации мантиссы с меньшим порядком MD и выполняется на блоке выравнивания порядков.

Вычисление мантиссы результата организовано таким образом, что обе мантиссы MO и MDEN поступают в прямых кодах и, если знаки одинаковые, то выполняется сложение мантисс, а результату присваивается знак,

соответствующий знаку мантисс на входе блока. Если же мантиссы имеют разные знаки, то мантисса результата формируется как разность, где в качестве уменьшаемого выступает мантисса большая по абсолютной величине, а в качестве вычитаемого – мантисса меньшая по абсолютной величине и знаку разности присваивается знак мантиссы, большей по абсолютной величине. Далее выполняется нормализация мантиссы результата. При выполнении округления результата выбран вариант округления в направлении ближайшего целого.

5.3.4. Функциональный узел умножения чисел с плавающей запятой

Умножение чисел с плавающей запятой осуществляется функциональным узлом, выполненным в виде IP-ядра. Операция умножения производится только над нормализованными числами. Упрощенная схема функционального узла умножения показана на рис. 5.6.

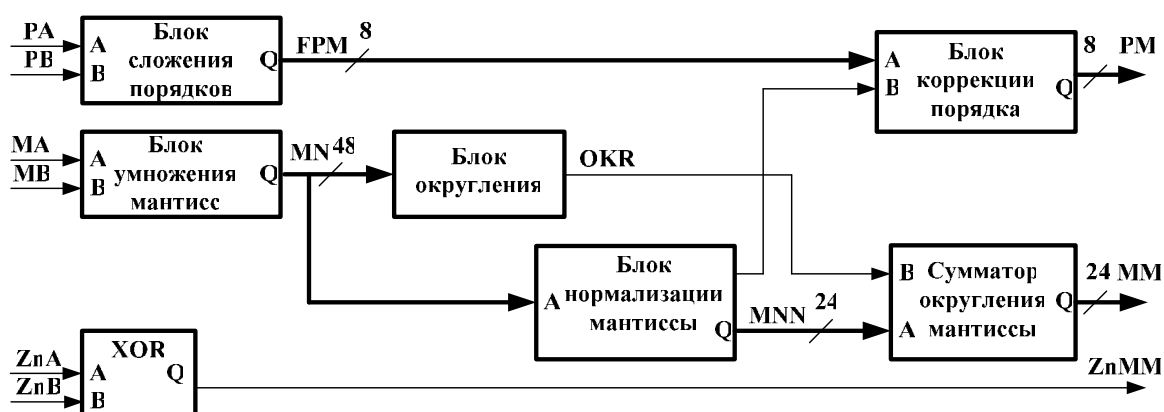


Рис. 5.6. Структурная схема умножителя MULT 32

Порядок результата вычисляется как алгебраическая сумма порядков сомножителей с помощью блока сложения порядков. Умножение двух 24-разрядных мантисс осуществляется на блоке умножения мантисс. Мантисса произведения содержит 48 разрядов. Далее выполняются нормализация мантиссы произведения и коррекция порядка. В блоке округления определяется значение бита округления мантиссы произведения. На выход умножителя выдается знак мантиссы, порядок и мантисса произведения.

Рассмотренные функциональные узлы сложения и умножения в формате плавающей запятой одинарной точности дают представление о принципах построения вычислительных узлов для реконфигурируемых структур, поэтому опускаем рассмотрение более сложных по своей структуре функциональных узлов деления, извлечения квадратного корня, а также функциональных узлов двойной точности.

5.4. Макрообъект для решения задач математической физики сеточными методами

Рассмотрим в качестве примера структуру макрообъекта, ориентированного на решение задач математической физики, и использование этого макрообъекта для создания проблемно-ориентированной структуры в пределах базового модуля [139].

Любые макрообъекты, ориентированные на решения задач из той или иной предметной области, строятся на основе фиксированного набора библиотечных функциональных узлов. Набор узлов определяется для каждой предметной области и отражает присущие ей особенности. В такой набор входят функциональные узлы, выполняющие обработку информации, блоки внутреннего ОЗУ определенной структуры для хранения промежуточных данных и временного согласования потоков данных внутри макрообъекта, специальные функциональные узлы (например, для слежения за процессом сходимости итерационных процессов), интерфейсные узлы для согласования с другими макрообъектами и т.п. В дальнейшем функциональные узлы, входящие в состав макрообъекта, будем называть *объектами*. В частности, количество и номенклатура объектов для макрообъекта, ориентированного на решение задач математической физики, может определяться теми вычислениями, которые необходимо выполнять в узле сеточной области. Такой макрообъект содержит функциональные узлы, набор блоков задержки, блок для расчета погрешности, блок граничных условий, контроллеры распределенной

памяти, а также коммутационную среду для связи элементов между собой. Все объекты параметризуемы и могут менять свою внутреннюю структуру в зависимости от решаемой задачи. Количество объектов и коммутационная структура также могут изменяться и задаются набором параметров. Структура макрообъекта для задач математической физики представлена на рис. 5.7.

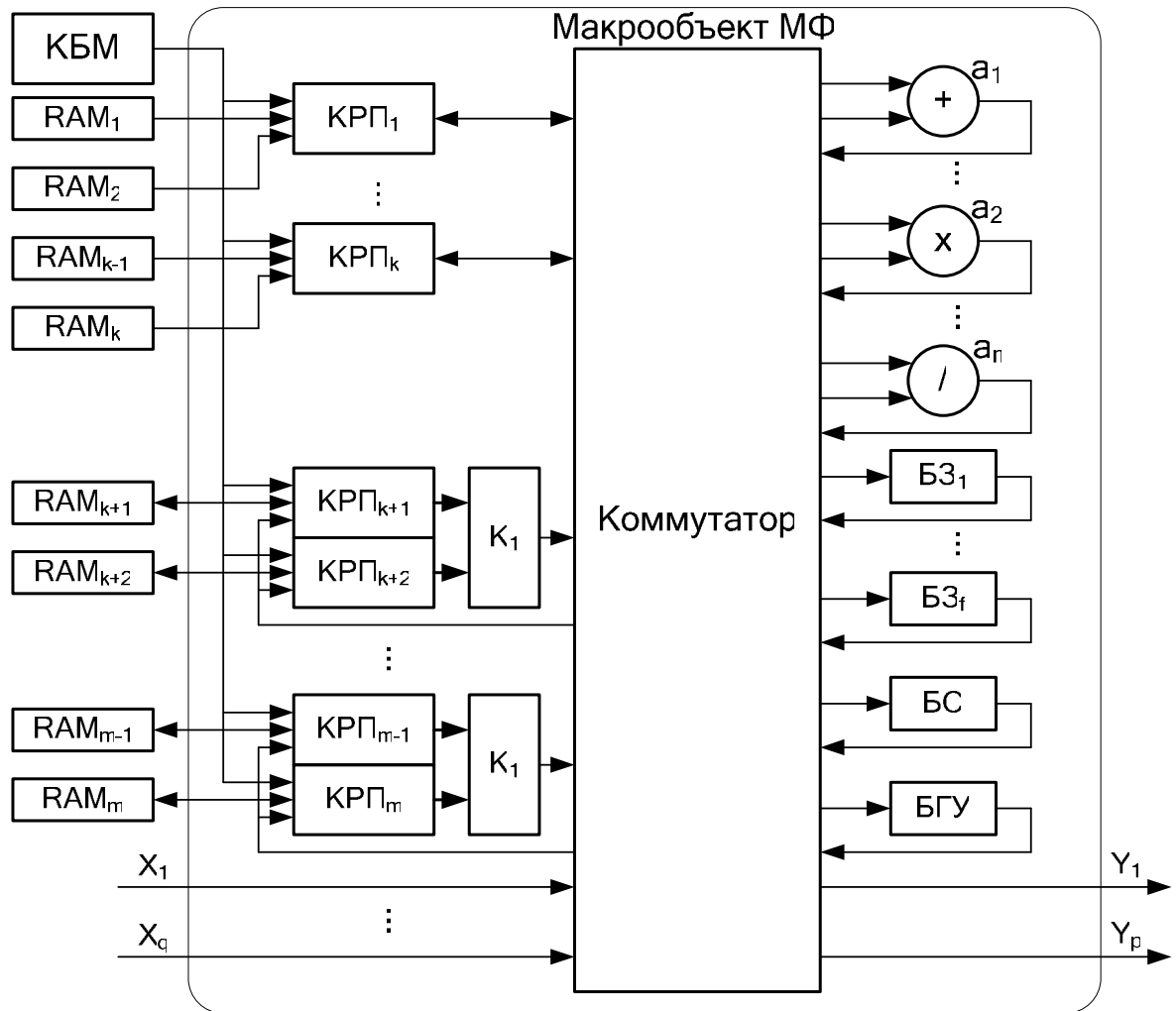


Рис. 5.7. Структура макрообъекта для задач математической физики

Макрообъект для задач математической физики включает в себя следующие объекты:

- функциональные узлы $a_1 \dots a_k$ (сумматоры, умножители, делители и т.д.);
- блоки задержек $БЗ_1 \dots БЗ_{k1}$;
- блок сходимости БС;
- блок граничных условий БГУ;
- динамические коммутаторы данных $K1, K2$;
- контроллеры распределенной памяти КРП;

- интерфейсные узлы приема данных из других макрообъектов $X_1 \dots X_q$;
- интерфейсные узлы передачи данных в другие макрообъекты $Y_1 \dots Y_p$.

При помощи функциональных узлов $a_1 \dots a_k$ и коммутатора создается вычислительная структура для расчетов в узле сетки. Количество объектов и связи между ними могут варьироваться и задаются при помощи загружаемых параметров. Контроллеры распределенной памяти предназначены для реализации в макрообъекте структуры распределенной памяти. К каждому контроллеру подключены блоки внешней памяти, которые не являются элементами макрообъекта. Блоки задержек представляют собой память FIFO и обеспечивают синхронизацию вычислений, задерживая данные на определенное количество тактов. Количество блоков задержки и количество тактов, на которое задерживаются данные, могут изменяться и зависят от конкретной реализации.

Коммутатор обеспечивает отображение информационного графа вычислений в узле сетки на структуру макрообъекта. Настраиваясь один раз, коммутатор в процессе выполнения задачи не изменяется, однако для различных задач математической физики должна создаваться своя коммутационная структура. Входные и выходные интерфейсные узлы макрообъекта $X_1 \dots X_q$ и $Y_1 \dots Y_p$ обеспечивают объединение макрообъектов в более мощные вычислительные структуры. Такие объединенные вычислительные структуры выполняют расчеты в нескольких смежных узлах сетки, тем самым повышая производительность реконфигурируемой мультиконвейерной вычислительной структуры.

Данный макрообъект ориентирован на решение задач математической физики сеточными методами [52-54]. Поскольку процесс решения является итерационным, в макрообъект введен *блок сходимости*. В блоке сходимости рассчитывается погрешность на очередном шаге алгоритма и выносятся решение об окончании или продолжении работы.

Блок граничных условий, размещенный в макрообъекте, предназначен для работы с граничными точками сеточной области и состоит из устройства

управления и мультиплексора (Mx). Структура блока граничных условий в совокупности с другими элементами макрообъекта показана на рис. 5.8.

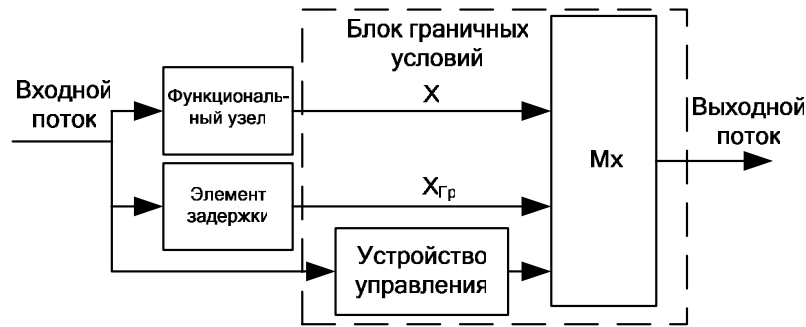


Рис. 5.8. Структура блока граничных условий

Поток входных данных поступает в функциональный узел на элемент задержки и устройство управления, и если входные данные являются граничными точками, то они без изменения передаются на выход.

В качестве примера решения задачи на реконфигурируемой структуре с макрообъектной архитектурой рассмотрим решение задачи однофазной фильтрации жидкости в пористой среде. Данная задача описывается дифференциальными уравнениями вида

$$\nabla(\lambda(p)(\nabla p - \rho_{sc} g \nabla D)) = 0. \quad (5.1)$$

Для решения этого уравнения применим метод конечных разностей. Выбор метода обусловлен его универсальностью.

Метод конечных разностей заключается в следующем. Область непрерывного изменения аргумента заменяется дискретным множеством точек или узлов, называемым *сеткой*. Вместо функций непрерывного аргумента рассматриваются функции дискретного аргумента, определенные в узлах сеточной области и называемые *сеточными функциями*. Производные, входящие в дифференциальное уравнение, аппроксимируются соответствующими разностными соотношениями. Дифференциальное уравнение при этом заменяется системой алгебраических уравнений [138]. Начальные и краевые условия заменяются разностными начальными и краевыми условиями для сеточной функции. Тогда решение задачи фильтрации сводится к решению системы

$$p_{i,j,k} = \frac{1}{\frac{\lambda_{i+1,j,k} + \lambda_{i-1,j,k}}{2\Delta x^2} + \frac{\lambda_{i,j+1,k} + \lambda_{i,j-1,k}}{2\Delta y^2} + \frac{\lambda_{i,j,k+1} + \lambda_{i,j,k-1}}{2\Delta z^2}} \times$$

$$\times \left(\frac{\lambda_{i+1,j,k} (p_{i+1,j,k} - \rho_{sc} g(D_{i+1,j,k} - D_{i,j,k})) + \lambda_{i-1,j,k} (p_{i-1,j,k} + \rho_{sc} g(D_{i,j,k} - D_{i-1,j,k}))}{2\Delta x^2} + \right.$$

$$+ \frac{\lambda_{i,j+1,k} (p_{i,j+1,k} - \rho_{sc} g(D_{i,j+1,k} - D_{i,j,k})) + \lambda_{i,j-1,k} (p_{i,j-1,k} + \rho_{sc} g(D_{i,j,k} - D_{i,j-1,k}))}{2\Delta y^2} +$$

$$\left. + \frac{\lambda_{i,j,k+1} (p_{i,j,k+1} - \rho_{sc} g(D_{i,j,k+1} - D_{i,j,k})) + \lambda_{i,j,k-1} (p_{i,j,k-1} + \rho_{sc} g(D_{i,j,k} - D_{i,j,k-1}))}{2\Delta z^2} \right). \quad (5.2)$$

Для решения системы линейных алгебраических уравнений (5.2) может быть применен метод одновременной верхней релаксации, итерационная формула которого имеет вид

$$p_{i,j,k}^{(n)} = \omega F(p_{i+1,j,k}^{(n-1)}, p_{i-1,j,k}^{(n-1)}, p_{i,j+1,k}^{(n-1)}, p_{i,j-1,k}^{(n-1)}, p_{i,j,k+1}^{(n-1)}, p_{i,j,k-1}^{(n-1)}) + (1 - \omega) p_{i,j,k}^{(n-1)}, \quad (5.3)$$

где F – функция, описываемая выражением (5.2),

ω – итерационный параметр, $0 < \omega < 2$,

n – номер итерации.

Термин «одновременный» означает тот факт, что полученные новые значения давления $p_{i,j,k}$ будут использованы только в следующей итерации. Это ведет к увеличению числа итераций, однако дает возможность распараллелить процесс вычислений.

Условием завершения процесса вычислений является неравенство

$$\max_{i,j,k} |p_{i,j,k}^{(n)} - p_{i,j,k}^{(n-1)}| < \varepsilon, \quad (5.4)$$

где ε – заданная точность.

5.5. Пример решения задачи математической физики с помощью макрообъекта

Рассмотрим решение задачи фильтрации жидкости на базовом модуле третьего поколения 16V4-50, содержащем 16 ПЛИС с интеграцией $4 \cdot 10^6$

эквивалентных вентилях. Макрообъект, описанный выше, может быть реализован в четырех ПЛИС базового модуля.

Реализация данной вычислительной схемы на базовом модуле 16V4-50 возможна в двух вариантах. Первый вариант заключается в следующем: на базовом модуле располагаются четыре независимых макрообъекта, каждый в своем квадранте. В этом случае макрообъекты работают независимо и обрабатывают в общем случае произвольные узлы сеточной области. При этом макрообъекты не обмениваются данными между собой. Второй вариант предусматривает совместное использование макрообъектов, расположенных в четырех квадрантах базового модуля. В этом случае четыре квадранта базового модуля обрабатывают одновременно четыре соседние точки сеточной области.

Остановимся на первом варианте вычислений. Решение задачи реализуется в четырех ПЛИС, соединенных между собой и составляющих квадрант в базовом модуле. Рассмотрим более подробно структуру каждой ПЛИС макрообъекта. Для этого в формуле (5.2) введем следующие обозначения:

$$\begin{aligned}
 A &= \frac{1}{\frac{\lambda_{i+1,j,k} + \lambda_{i-1,j,k}}{2\Delta x^2} + \frac{\lambda_{i,j+1,k} + \lambda_{i,j-1,k}}{2\Delta y^2} + \frac{\lambda_{i,j,k+1} + \lambda_{i,j,k-1}}{2\Delta z^2}}, \\
 B &= \left(\frac{\lambda_{i+1,j,k} (p_{i+1,j,k} - \rho_{sc} g(D_{i+1,j,k} - D_{i,j,k})) + \lambda_{i-1,j,k} (p_{i-1,j,k} + \rho_{sc} g(D_{i,j,k} - D_{i-1,j,k}))}{2\Delta x^2} \right), \\
 C &= \frac{\lambda_{i,j+1,k} (p_{i,j+1,k} - \rho_{sc} g(D_{i,j+1,k} - D_{i,j,k})) + \lambda_{i,j-1,k} (p_{i,j-1,k} + \rho_{sc} g(D_{i,j,k} - D_{i,j-1,k}))}{2\Delta y^2}, \\
 E &= \frac{\lambda_{i,j,k+1} (p_{i,j,k+1} - \rho_{sc} g(D_{i,j,k+1} - D_{i,j,k})) + \lambda_{i,j,k-1} (p_{i,j,k-1} + \rho_{sc} g(D_{i,j,k} - D_{i,j,k-1}))}{2\Delta z^2}. \quad (5.5)
 \end{aligned}$$

На рис. 5.9а,б,в,г представлены обобщенные структуры, запрограммированные в четырех ПЛИС для вычисления значений А, В, С, Е соответственно по формуле (5.5).

Рассмотрим второй вариант реализации, в котором четыре квадранта базового модуля обрабатывают одновременно четыре соседние точки сеточной области.

Вычислительная структура задачи для связанных вычислений, подразумевающих решение единой задачи четырьмя взаимосвязанными квадрантами, представлена на рис. 5.11.

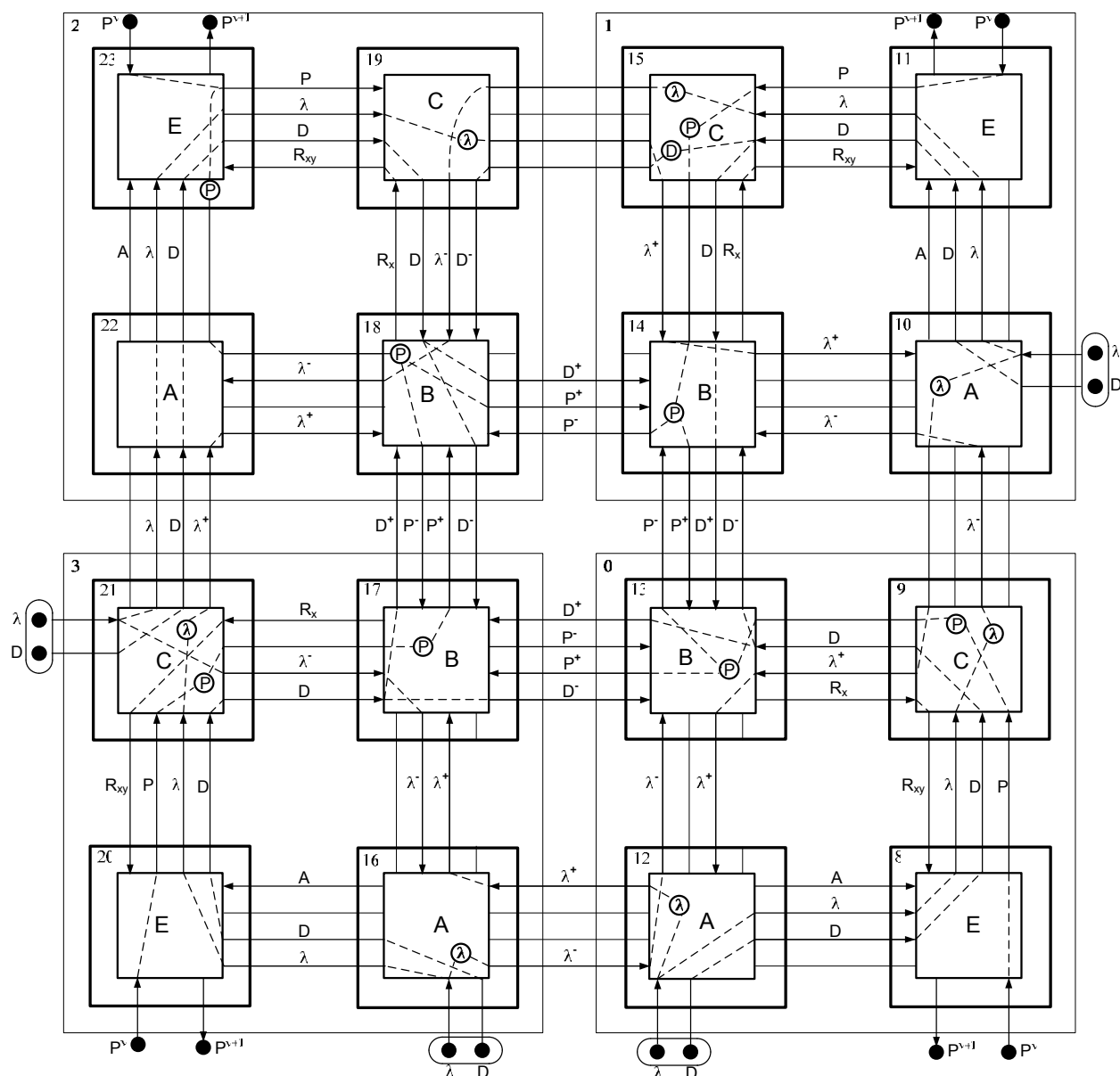


Рис. 5.11. Реализация связанных вычислений в базовом модуле

На основе макрообъекта может быть построено большое количество вычислительных структур для решения различных задач математической физики. Пользователь имеет возможность самостоятельно задавать состав макрообъекта, изменять его структуру под различные вычислительные схемы, а

также изменять свойства ранее созданных элементов, доопределяя библиотечные элементы.

5.6. Макрообъект для решения задач цифровой обработки сигналов

Основными алгоритмами цифровой обработки сигналов являются алгоритм цифровой свертки и алгоритм дискретного преобразования Фурье, поскольку на их основе можно получить множество прикладных алгоритмов, направленных на решение большинства частных случаев [19,98,100,107].

Цифровая свертка с конечной импульсной характеристикой определяется выражением [100]

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k), \quad n = 0, 1, \dots, N-1. \quad (5.6)$$

Алгоритмы дискретного прямого и обратного преобразований Фурье [98]

$$S(k) = \sum_{n=0}^{N-1} s(n)W^{n-k}, \quad W = e^{-j\frac{2\pi}{N}}, \quad k = 0, 1, \dots, N-1, \quad (5.7)$$

$$s(n) = \frac{1}{N} \sum_{k=0}^{N-1} S(k)W^{-nk}, \quad W = e^{-j\frac{2\pi}{N}}, \quad n = 0, 1, \dots, N-1 \quad (5.8)$$

имеют как самостоятельное значение при вычислении спектра сигнала, так и используются в алгоритмах быстрых свертки, вычисления корреляционных функций и т.п. Спектр может быть получен посредством дискретного прямого преобразования Фурье (ДПФ) по формуле (5.7).

Для вычисления прямого и обратного ДПФ существуют эффективные алгоритмы – это так называемые быстрые алгоритмы преобразования Фурье – БПФ для вычисления прямого ДПФ и ОБПФ для вычисления обратного ДПФ. Алгоритмы БПФ, как правило, выполняются над вектором входных данных, длина которых является степенью двойки $N = 2^p$, $p = 1, 2, 3, \dots$. Будем в дальнейшем приписывать к аббревиатуре БПФ цифру N , которая определяет размер преобразования над вектором длины N . В этом случае алгоритм БПФ, определенный над массивом $N = 2^9 = 512$, будет обозначаться как БПФ512.

Рассмотрим конвейерную реализацию свертки. Структура для конвейерной реализации алгоритма свертки (5.6) приведена на рис. 5.12.

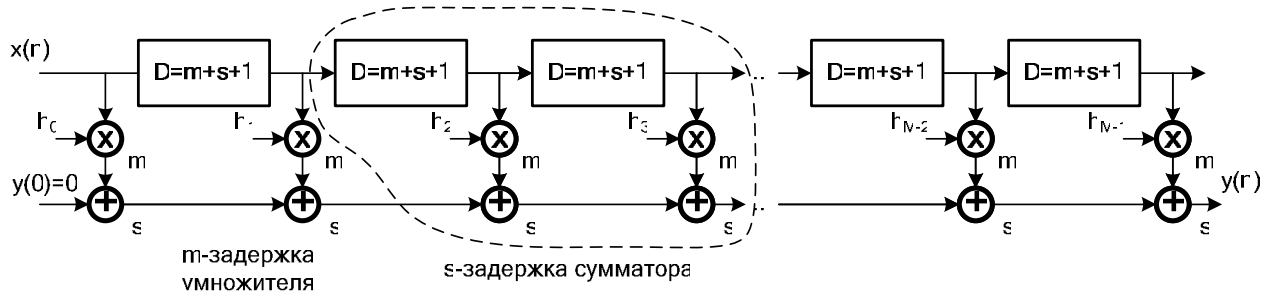


Рис. 5.12. Структурная реализация алгоритма свертки

Очевидно, что повторяющимся фрагментом для алгоритма свертки будет участок конвейерной схемы, выделенный пунктиром.

Рассмотрим реализацию алгоритма БПФ. На основании приведенных формул (5.7) и (5.8) можно сделать вывод, что количество входных данных равно количеству результатов обработки. Это касается как алгоритмов фильтрации, так и алгоритмов быстрых ортогональных преобразований. Утверждение справедливо при условии, что множество коэффициентов фильтров $\{h_i\}, i \in 0, 1, \dots, M-1$ и множество коэффициентов $\{W^{n \cdot k}\}, n, k \in 0, 1, \dots, N-1$ хранятся или вырабатываются в самом вычислительном устройстве, реализующем алгоритм, а не поступают из внешних источников. Утверждение имеет место также и для любых функционально законченных фрагментов упомянутых алгоритмов. Под функционально законченным фрагментом алгоритма будем понимать такую его часть, суперпозиция которых позволяет реализовать весь алгоритм. Для алгоритма БПФ512 таким фрагментом может являться, например, алгоритм БПФ8.

Приведем математическое описание фрагментов алгоритмов БПФ. Базовая операция БПФ по основанию 2 с прореживанием по частоте выполняется по следующему алгоритму:

$$\begin{aligned} \dot{A}_{t+1} &= \dot{A}_t + \dot{B}_t, \\ \dot{B}_{t+1} &= (\dot{A}_t - \dot{B}_t) \dot{W}_t, \\ \dot{W}_t &= \dot{W}_6 \times \dot{W}_8. \end{aligned} \quad (5.9)$$

Отдельно для реальных и мнимых частей можно записать:

$$\begin{aligned}
 \operatorname{Re}[\dot{A}_{t+1}] &= \operatorname{Re}[\dot{A}_t] + \operatorname{Re}[\dot{B}_t]; & \operatorname{Im}[\dot{A}_{t+1}] &= \operatorname{Im}[\dot{A}_t] + \operatorname{Im}[\dot{B}_t], \\
 \operatorname{Re}[\dot{B}_{t+1}] &= (\operatorname{Re}[\dot{A}_t] - \operatorname{Re}[\dot{B}_t]) \cdot \operatorname{Re}[\dot{W}_t] - (\operatorname{Im}[\dot{A}_t] - \operatorname{Im}[\dot{B}_t]), \\
 \operatorname{Im}[\dot{B}_{t+1}] &= (\operatorname{Re}[\dot{A}_t] - \operatorname{Re}[\dot{B}_t]) \cdot \operatorname{Im}[\dot{W}_t] + (\operatorname{Im}[\dot{A}_t] - \operatorname{Im}[\dot{B}_t]), \\
 \operatorname{Re}[\dot{W}_t] &= \operatorname{Re}[\dot{W}_\sigma] \cdot \operatorname{Re}[\dot{W}_\sigma] - \operatorname{Im}[\dot{W}_\sigma] \cdot \operatorname{Im}[\dot{W}_\sigma], \\
 \operatorname{Im}[\dot{W}_t] &= \operatorname{Re}[\dot{W}_\sigma] \cdot \operatorname{Im}[\dot{W}_\sigma] + \operatorname{Im}[\dot{W}_\sigma] \cdot \operatorname{Re}[\dot{W}_\sigma].
 \end{aligned} \tag{5.10}$$

По этим же формулам может выполняться и базовая операция алгоритма обратного БПФ [110]. Для этого необходимо использовать комплексно-сопряженные коэффициенты $\dot{W}_t$. Комплексное сопряжение выполняется путем изменения знака у мнимой части коэффициентов $\dot{W}_t$.

Рассмотрим получение коэффициентов $\dot{W}_t$. Коэффициент $\dot{W}_t$ может поступать в каждую базовую операцию из внешнего ПЗУ объема $N/2$ комплексных слов. При структурной организации вычислений, когда одновременно выполняется несколько базовых операций БПФ, необходимо одновременно выбирать такое же количество различных коэффициентов $\dot{W}_t$. Поэтому для каждой базовой вычислительной структуры необходимо иметь свое ПЗУ.

Другие подходы заключаются в вычислении коэффициента $\dot{W}_t$. Один из способов вычисления, описанный в [100], заключается в рекурсивном вычислении коэффициентов исходя из коэффициента с минимальным углом $W = e^{-j\frac{2\pi}{N}}$. Однако, рекурсивный способ вычисления имеет быстро накапливающуюся погрешность и поэтому применим только для малых N .

Для структурно-процедурных реализаций БПФ наиболее приемлемым является метод, описанный в [111,113]. Коэффициент $\dot{W}_t$ вычисляется из базового и вспомогательного коэффициентов $\dot{W}_\sigma$ и $\dot{W}_\sigma$ по формуле $\dot{W}_t = \dot{W}_\sigma \times \dot{W}_\sigma$. Коэффициенты $\dot{W}_\sigma$ и $\dot{W}_\sigma$ считываются из ПЗУW. Для хранения коэффициентов $\dot{W}_\sigma$ и $\dot{W}_\sigma$ участвующих в получении коэффициента $\dot{W}_t$ преобразования БПФ размера N , необходимы две таблицы – таблица

базовых и таблица вспомогательных коэффициентов. Коэффициенты, размещаемые в базовой и вспомогательной таблицах коэффициентов, могут быть получены по следующим формулам:

$$\dot{W}_o = \exp(-j 2\pi p N_2 / N_M), \quad \dot{W}_e = \exp(-j 2\pi q / N_M),$$

$$p = 0, 1, \dots, N_1 - 1, \quad q = 0, 1, \dots, N_2 - 1, \quad N_1 \times N_2 = N_M / 2,$$

где N_M - максимально возможный размер БПФ, на который рассчитан макрообъект.

Такой способ формирования коэффициентов позволяет сократить объем ПЗУ W по сравнению с традиционным подходом в $N_M / (2(N_1 + N_2))$ раз. Для $N_M = 2^{16}$, $N_1 = 2^7$ и $N_2 = 2^8$ объем ПЗУ сокращается в 85 раз [110, 111].

Рассмотрим использование коэффициентов $\dot{W}_t$ более подробно. Предположим, что в составе макропроцессора имеется таблица с полным набором коэффициентов $\dot{W}_t = \exp(-j 2\pi n / N_M)$, $n = 0, 1, \dots, (N/2)-1$, необходимым для выполнения алгоритма БПФ размера N . Обращение к элементам этой таблицы, назовем ее традиционной таблицей $\dot{W}_t$, ведется по адресу $A = n$ и, в результате, считывается коэффициент $\dot{W}_t[A] = \exp(-j 2\pi A / N_M)$. Алгоритм получения адресов A для считывания коэффициентов из традиционной таблицы зависит от порядка выполнения базовых операций алгоритма БПФ. Так как структура вычислительного устройства ориентирована на конвейерное выполнение алгоритма БПФ, то порядок выполнения базовых операций, а следовательно, значение адреса A определяется конвейерным алгоритмом, описанным в [113].

Вычислительная структура, соответствующая обычной базовой операции, где коэффициент $\dot{W}_t$ подается из внешнего источника, показана на рис. 5.13.

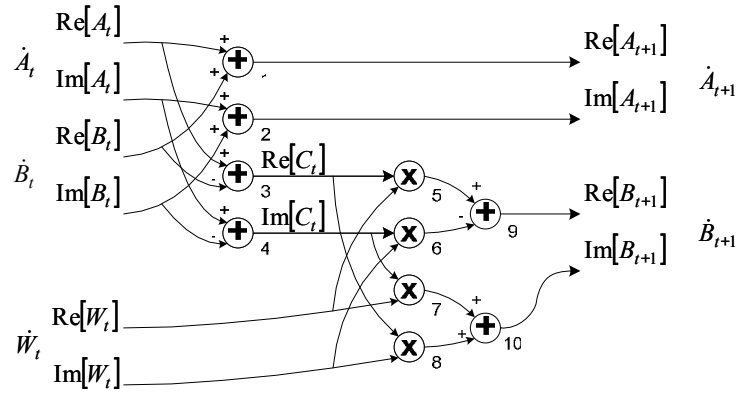


Рис. 5.13. Вычислительная структура обычной базовой операции

Вычислительная структура, соответствующая модифицированной базовой операции, где коэффициент $\dot{W}_t$ вырабатывается из $\dot{W}_o$ и $\dot{W}_e$, показана на рис. 5.14.

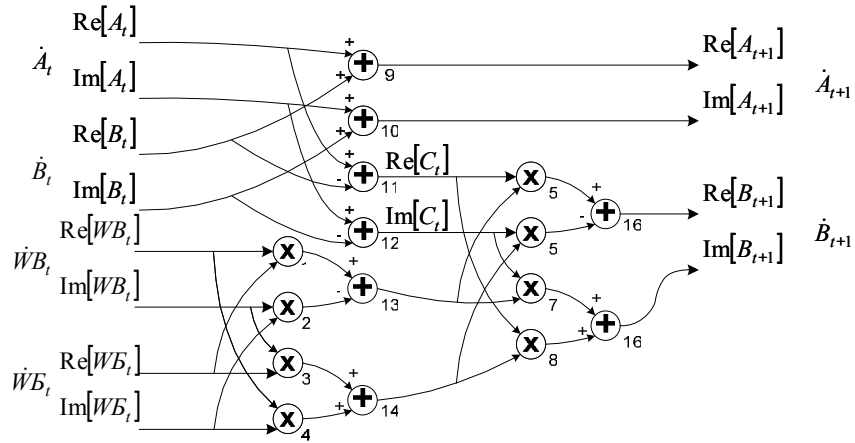


Рис. 5.14. Вычислительная структура модифицированной базовой операции

Выполнение быстрой свертки с использованием алгоритмов БПФ и ОБПФ предполагает умножение спектра входного сигнала на коэффициенты спектра ядра свертки. Поскольку алгоритм БПФ на последней итерации использует коэффициенты $\dot{W}_t = 1$, то выполнение базовой операции на последней итерации БПФ может быть совмещено с умножением на коэффициенты ядра свертки. С этой целью рассмотрим выполнение такой совмещенной с умножением упрощенной базовой операции БПФ.

Упрощенная базовая операция с умножением на коэффициенты ядра свертки $\dot{H}_i$ и $\dot{H}_{i+1}$ определяется выражением:

$$\begin{aligned}\dot{A}_{t+1} &= (\dot{A}_t + \dot{B}_t) \cdot \dot{H}_i \\ \dot{B}_{t+1} &= (\dot{A}_t - \dot{B}_t) \cdot \dot{H}_{i+1},\end{aligned}\tag{5.11}$$

или, отдельно для реальных и мнимых частей:

$$\begin{aligned}
 \operatorname{Re}[\dot{A}_{t+1}] &= (\operatorname{Re}[\dot{A}_t] + \operatorname{Re}[\dot{B}_t]) \cdot \operatorname{Re}[\dot{H}_t] - (\operatorname{Im}[\dot{A}_t] + \operatorname{Im}[\dot{B}_t]) \cdot \operatorname{Im}[\dot{H}_t], \\
 \operatorname{Im}[\dot{A}_{t+1}] &= (\operatorname{Re}[\dot{A}_t] + \operatorname{Re}[\dot{B}_t]) \cdot \operatorname{Im}[\dot{H}_t] + (\operatorname{Im}[\dot{A}_t] + \operatorname{Im}[\dot{B}_t]) \cdot \operatorname{Re}[\dot{H}_t], \\
 \operatorname{Re}[\dot{B}_{t+1}] &= (\operatorname{Re}[\dot{A}_t] - \operatorname{Re}[\dot{B}_t]) \cdot \operatorname{Re}[\dot{H}_{t+1}] - (\operatorname{Im}[\dot{A}_t] - \operatorname{Im}[\dot{B}_t]) \cdot \operatorname{Im}[\dot{H}_{t+1}], \\
 \operatorname{Im}[\dot{B}_{t+1}] &= (\operatorname{Re}[\dot{A}_t] - \operatorname{Re}[\dot{B}_t]) \cdot \operatorname{Im}[\dot{H}_{t+1}] + (\operatorname{Im}[\dot{A}_t] - \operatorname{Im}[\dot{B}_t]) \cdot \operatorname{Re}[\dot{H}_{t+1}].
 \end{aligned} \tag{5.12}$$

Упрощенная базовая операция отличается от обычной тем, что в ней коэффициент $\dot{W}_t$ равен единице. Поэтому вместо вычисления $\dot{W}_t = \dot{W}_o \times \dot{W}_e$ и умножения на коэффициент $\dot{W}_t = 1$ можно выполнить умножение на коэффициенты ядра свертки $\dot{H}_t$ и $\dot{H}_{t+1}$. Упрощенные базовые операции с умножением на коэффициенты ядра свертки $\dot{H}_t$ и $\dot{H}_{t+1}$ используются только в алгоритмах быстрой свертки и совмещают в себе выполнение базовой операции на последней итерации алгоритма БПФ и умножение на коэффициенты ядра свертки.

Вычислительная структура, соответствующая упрощенной базовой операции с умножением на коэффициенты ядра свертки $\dot{H}_t$ и $\dot{H}_{t+1}$, показана на рис. 5.15.

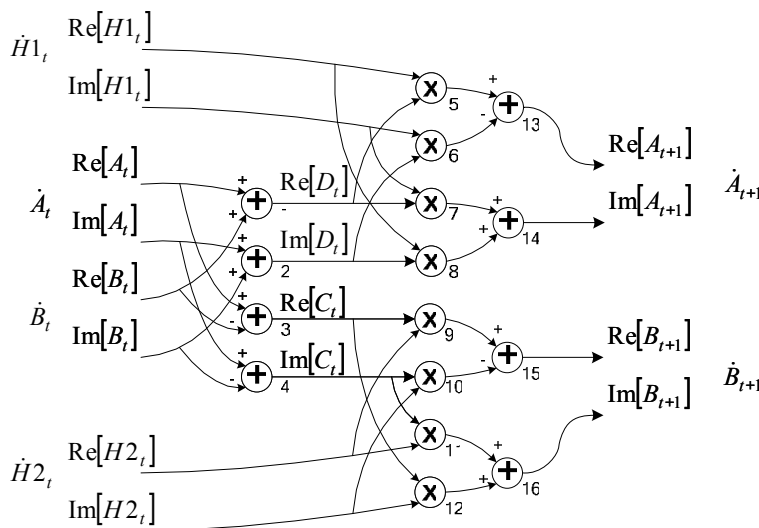


Рис. 5.15. Вычислительная структура упрощенной базовой операции

Рассмотрим организацию вычислительного процесса в конвейерном вычислительном блоке (КВБ), содержащем два сумматора и два умножителя и выполняющем расчеты согласно (5.10) и (5.12). Будем полагать:

- источником входных данных является 32-разрядное ОЗУ, в котором реальная и мнимая части комплексных чисел записаны по последовательным адресам. Адресом комплексного числа считается адрес A реальной части числа. В этом случае мнимая часть числа имеет адрес $A+1$;

- входные данные для базовой операции $\text{Re}[\dot{A}_t]$, $\text{Im}[\dot{A}_t]$, $\text{Re}[\dot{B}_t]$ и $\text{Im}[\dot{B}_t]$ подаются на обработку за четыре такта;

- результаты базовой операции $\text{Re}[\dot{A}_{t+1}]$, $\text{Im}[\dot{A}_{t+1}]$, $\text{Re}[\dot{B}_{t+1}]$ и $\text{Im}[\dot{B}_{t+1}]$ выдаются также за четыре такта;

- коэффициенты $\dot{W}_t$ рассчитываются в конвейерном блоке из $\dot{W}_o$ и $\dot{W}_e$ по формулам (5.10). Коэффициенты $\dot{W}_o$ и $\dot{W}_e$ считываются из внутреннего ПЗУ блока;

- коэффициенты $\dot{H}_t$ и $\dot{H}_{t+1}$ считываются из внутреннего ОЗУ.

В этом случае для подачи в конвейерный вычислительный блок двух комплексных чисел $\dot{A}_t$ и $\dot{B}_t$, являющихся исходными данными для вычисления базовой операции БПФ, и вывода результатов вычислений $\dot{A}_{t+1}$ и $\dot{B}_{t+1}$ в формате 32-разрядной плавающей запятой необходимо четыре такта. За четыре такта по двум каналам одновременно возможно передать четыре комплексных числа $\dot{A}_t$ и $\dot{B}_t$ и вывести результаты вычислений по упрощенной базовой операции. Таким образом, формат данных и порядок обменов по внешним связям конвейерного блока определяют требования к организации вычислительной части блока. Конвейерный блок должен выполнять вычисления по формулам (5.10) и (5.12) за четыре такта. Это позволит уравнивать время конвейерного вычисления и время передачи входных данных и выгрузки результатов по внешним каналам блока. Такая вычислительная структура является оптимальной в том смысле, что в ней обработка и передача информации по каналам согласованы и не требуют дополнительных аппаратных средств выравнивания и синхронизации.

Вычислительная часть конвейерного блока содержит два сумматора и два умножителя с плавающей запятой, каждый из которых представляет

собой конвейерное вычислительное устройство, выполняющее в потоковом режиме операции сложения и умножения за один такт. Помимо устройств умножения и сложения в состав блока входят регистры, двухпортовые ОЗУ, мультиплексоры, внутренний коммутатор и ПЗУ. Структурная схема конвейерного вычислительного блока показана на рис. 5.16.

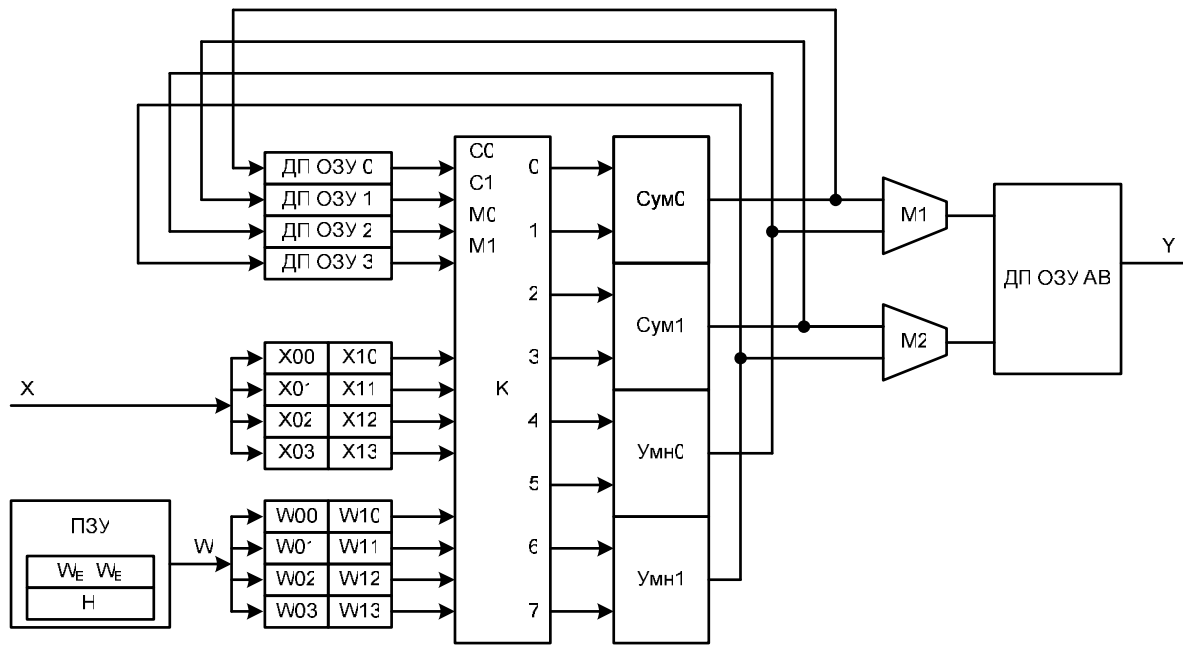


Рис. 5.16. Структурная схема конвейерного вычислительного блока

Учитывая тот факт, что базовая операция с вычислением коэффициента $\dot{W}_i$ (5.10) и упрощенная базовая операция с умножением на коэффициенты ядра свертки $\dot{H}_i$ и $\dot{H}_{i+1}$ (5.12) содержат по 8 операций сложения и по 8 операций умножения, можно утверждать, что конвейерное выполнение каждой из них занимает 4 такта работы блока.

Построение конвейерных структур БПФ требует наличия конвейерного ОЗУ между устройствами, выполняющими последовательные итерации алгоритма БПФ [100,111]. Эти ОЗУ вносят задержки между ступенями конвейера. Конвейерное ОЗУ может быть реализовано на базе блочной памяти ПЛИС микросхем семейства Virtex в виде двухпортовых банков памяти ДП ОЗУАВ. Комплексные результаты вычислений подаются в ОЗУАВ и записываются по адресам, вырабатываемым блоком адресации. Через некоторое количество тактов, которое определяется в вычислительном

конвейере, из ОЗУАВ начинается считывание комплексных выходных данных по адресам, которые формирует блок адресации ОЗУАВ. Глубина конвейерной памяти может программироваться в зависимости от номера итерации БПФ.

Без потери общности будем полагать, что глубина конвейерного сумматора составляет четыре такта, а глубина конвейерного умножителя – три такта. Временная диаграмма выполнения вычислений конвейерным вычислительным блоком (см. рис. 5.16) в соответствии с формулами (5.10), представлена на рис. 5.17.

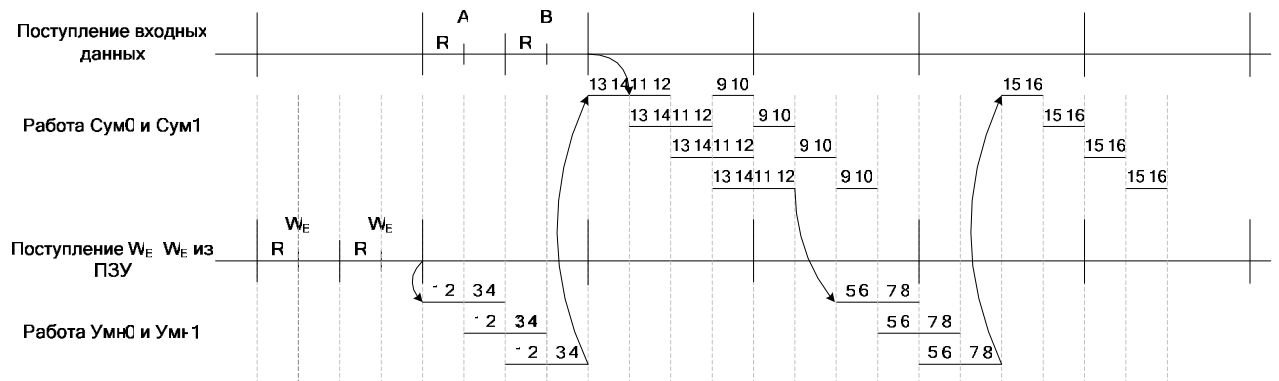


Рис. 5.17. Временная диаграмма выполнения вычислений

На рисунке большими делениями показаны четырехтактные интервалы подачи в конвейерный блок действительных и мнимых частей для $\dot{A}_t$, $\dot{B}_t$, $\dot{W}_o$ и $\dot{W}_s$, пунктиром - такты работы. Цифрами указаны номера арифметических операций согласно рис. 5.14.

Управление работой конвейерного вычислительного блока осуществляется микропрограммным устройством управления. Микрокоманды циклически считываются из ОЗУ микропрограмм устройства управления. Цикл считывания, как видно из рис. 5.17, занимает двадцать два такта. Микропрограмма начинается с записи реальных и мнимых частей коэффициентов $\dot{W}_o$ и $\dot{W}_s$ и заканчивается выдачей в ОЗУ АВ результатов вычислений в соответствии с информационным графом, показанным на рис. 5.14.

На рис. 5.18 показана временная диаграмма выполнения на упрощенной базовой операции с умножением на комплексные коэффициенты ядра свертки по формулам (5.12) согласно информационному графу, представленному на рис. 5.15.

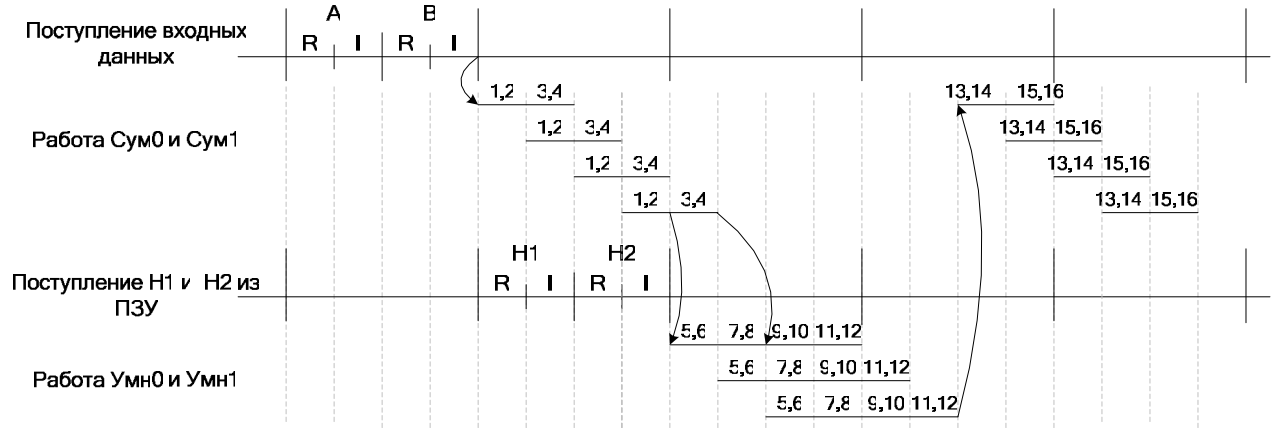


Рис. 5.18. Временная диаграмма выполнения упрощенной базовой операции

Микрокоманды циклически считываются из ОЗУ микропрограмм устройства управления. Цикл считывания для упрощенной базовой операции, согласно рис. 5.18, занимает девятнадцать тактов. Микропрограмма начинается с записи реальных и мнимых частей коэффициентов $\dot{A}_i$ и $\dot{B}_i$ и заканчивается выдачей в ОЗУ АВ результатов вычислений в соответствии с информационным графом, показанным на рис. 5.15.

Конвейерный вычислительный блок, показанный на рис. 5.16, выполняет следующие типовые операции цифровой обработки сигналов:

1. Базовая операция БПФ с использованием коэффициента $\dot{W}_i$ согласно информационному графу, показанному на рис. 5.13.
2. Базовая операция БПФ с расчетом коэффициента $\dot{W}_i$ из коэффициентов $\dot{W}_6$ и $\dot{W}_8$ согласно информационному графу, показанному на рис. 5.14.
3. Упрощенная базовая операция БПФ с умножением на коэффициенты ядра свертки $\dot{H}_i$ и $\dot{H}_{i+1}$ согласно (5.12).
4. Упрощенная базовая операция БПФ с вычислением квадрата модуля комплексных чисел $\dot{A}_{i+1}$ и $\dot{B}_{i+1}$.
5. Двухзвенная секция нерекурсивного фильтра согласно рис. 5.12.

6. Рекурсивное звено фильтра первого порядка.

7. Рекурсивное звено фильтра второго порядка. Отметим, что конвейерная реализация рекурсивных фильтров второго порядка может приводить к потере устойчивости фильтра и необходимо применять специальные меры, направленные на сохранение устойчивости [81,112].

8. Базовая операция в безызбыточном алгоритме БПФ действительного массива с разделением элементов и с расчетом коэффициента $\dot{W}_i$ согласно информационному графу, показанному на рис. 5.19.

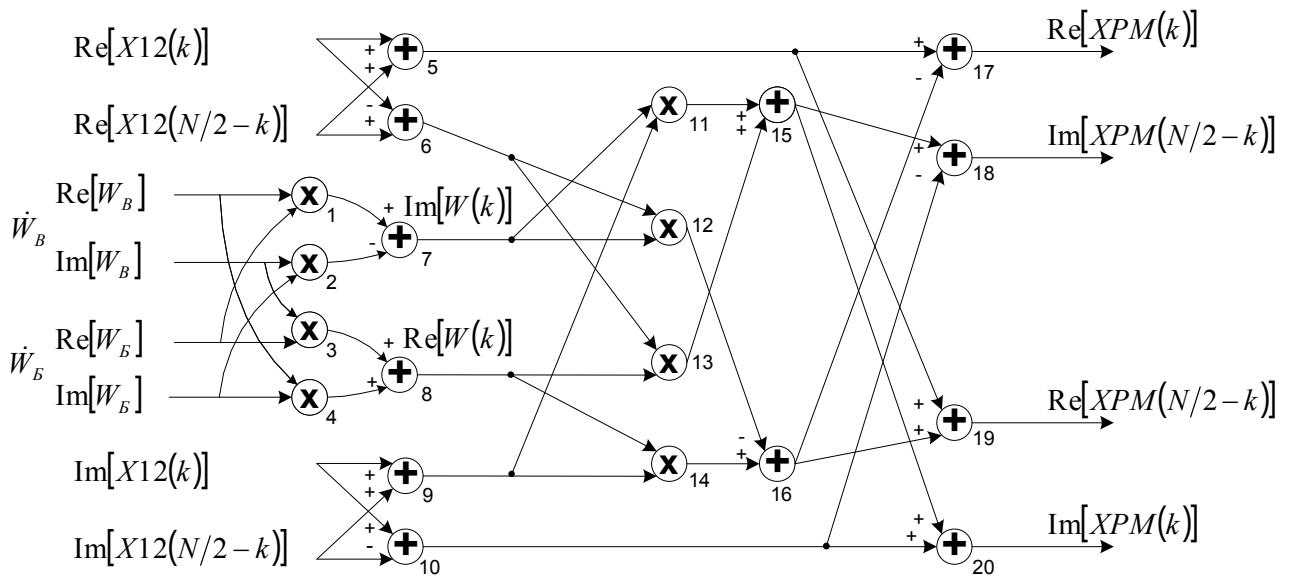


Рис. 5.19. Базовая операция в безызбыточном алгоритме БПФ действительного массива с разделением элементов

Таким образом, конвейерный вычислительный блок является универсальным вычислительным узлом, выполняющим целый ряд алгоритмов цифровой обработки сигналов. Поэтому используя этот блок в качестве основного вычислительного узла, можно предложить структуру макрообъекта цифровой обработки сигналов, представленную на рис. 5.20.

Контроллер внешних связей предназначен для создания каналов обмена данными и каналов распространения управляющей и настроечной информацией между макрообъектами базового модуля.

Блок преобразования кодов выполняет функции согласования форм представления информации между различными по своему назначению

функциональными узлами, входящими в систему цифровой обработки сигналов.

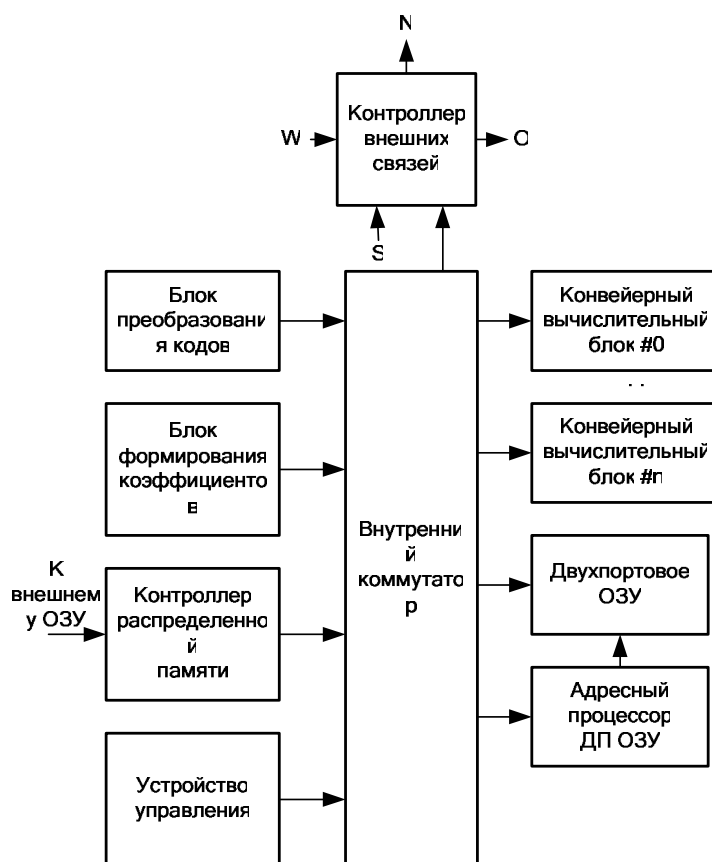


Рис. 5.20. Структура макрообъекта цифровой обработки сигналов

Как известно, современные аналогово-цифровые преобразователи (АЦП) не формируют кодов в формате плавающей запятой стандарта IEEE-754, а формируют информацию, представленную в смещенных или в прямых кодах. Это делается с целью повышения скорости работы и упрощения АЦП. Некоторые узлы цифровой обработки сигналов используют для представления данных дополнительные коды различной разрядности в формате фиксированной запятой. Фиксированная запятая используется в цифровой обработке сигналов с целью увеличения быстродействия функциональных узлов и упрощения их структуры. Разрядность обрабатываемых данных может меняться в определенных пределах с целью обеспечения необходимой точности вычислений и динамического диапазона представления информации на различных этапах обработки.

Блок формирования коэффициентов предназначен для хранения и выработки коэффициентов, необходимых для выполнения алгоритмов цифровой обработки сигналов. Коэффициенты могут быть постоянными (коэффициенты БПФ, коэффициенты, отражающие специфику аппаратуры и трактов передачи данных, и т.д.) и не постоянные (коэффициенты адаптивных фильтров, коэффициенты, зависящие от состояния окружающей среды, и т.д.). Постоянные коэффициенты, как правило, хранятся в устройствах постоянной памяти. Переменные коэффициенты изменяются в зависимости от условий обработки и поступают либо извне, либо вырабатываются в самом макрообъекте, и тогда в качестве устройства хранения коэффициентов выступает оперативная память. Поэтому в составе макрообъектов различного назначения могут использоваться различные блоки формирования коэффициентов. В состав блока коэффициентов входят и аппаратные средства адресации устройств хранения данных.

Контроллер распределенной памяти является основным активным узлом, формирующим потоки данных в мультиконвейерных вычислительных структурах. Использование контроллера памяти в качестве стандартного узла для взаимодействия с памятью сохраняет преемственность по программному обеспечению при использовании описанного макрообъекта в задачах цифровой обработки сигналов.

Конвейерный вычислительный блок необходим для выполнения основных вычислительных операций алгоритмов цифровой обработки сигналов. Принципы организации конвейерного блока макрообъекта цифровой обработки сигналов были описаны выше. Поскольку один блок может реализовывать ограниченный фрагмент информационного графа задачи цифровой обработки сигналов, например, фрагмент нерекурсивного фильтра или базовую операцию БПФ, то для реализации более крупных фрагментов алгоритмов используется несколько конвейерных блоков. Поэтому в состав макрообъекта цифровой обработки сигналов включается необходимое их количество.

В качестве примера рассмотрим структуру макрообъекта цифровой обработки сигналов ЦОС-7.1, в состав которого входит пять конвейерных блоков и который реализован в ПЛИС Virtex II Pro XC2VP100. Параметры базовых модулей 8V2P-64 и 4V2P-32, в которых были реализованы макрообъекты ЦОС-7.1, описаны в подразделе 4.4.3.2.

Структура макрообъекта ЦОС-7.1 показана на рис. 5.21.

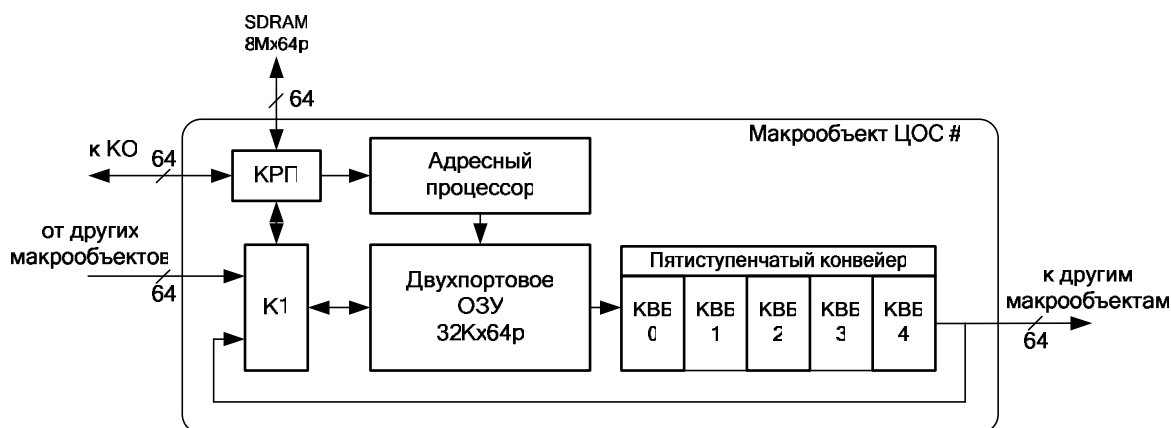


Рис. 5.21. Структура макрообъекта цифровой обработки сигналов

Двухпортовое ОЗУ и адресный процессор двухпортового ОЗУ предназначены для оперативного хранения промежуточных результатов вычислений. В качестве внешнего ОЗУ зачастую используется память с динамическим принципом хранения информации - SDRAM, DDR, DDR2, обеспечивающая экономичное хранение информации большого объема. Однако блочный принцип доступа к информации в динамических ОЗУ не позволяет реализовывать нестандартные типы адресации данных, которые используются в алгоритмах цифровой обработки сигналов для перестановки элементов цифровых последовательностей. Для этого в состав макрообъекта вводится ОЗУ с произвольным доступом, которое в совокупности с адресным процессором позволяет делать необходимые перестановки элементов цифровых последовательностей [113, 114].

Внутренний коммутатор предназначен для создания информационных каналов между всеми составными частями макрообъекта.

Устройство управления используется для передачи управляющей и настроечной информации во все компоненты макрообъекта, синхронизации их работы и организации вычислительного процесса.

Макрообъект включает следующие объекты: контроллер распределенной памяти; коммутатор каналов данных; адресный процессор; двухпортовое ОЗУ объемом 32К 64-разрядных слов; вычислительный конвейер из пяти конвейерных блоков.

Адресный процессор ориентирован на работу с двухпортовыми ОЗУ в составе макрообъекта. Основными задачами адресного процессора являются:

- обмен данными между SDRAM и двухпортовым ОЗУ под управлением контроллера распределенной памяти;
- обеспечение различных режимов адресации данных, в том числе и ориентированных на выполнение алгоритмов цифровой обработки сигналов.

Укрупненная структурная схема адресного процессора приведена на рис. 5.22.

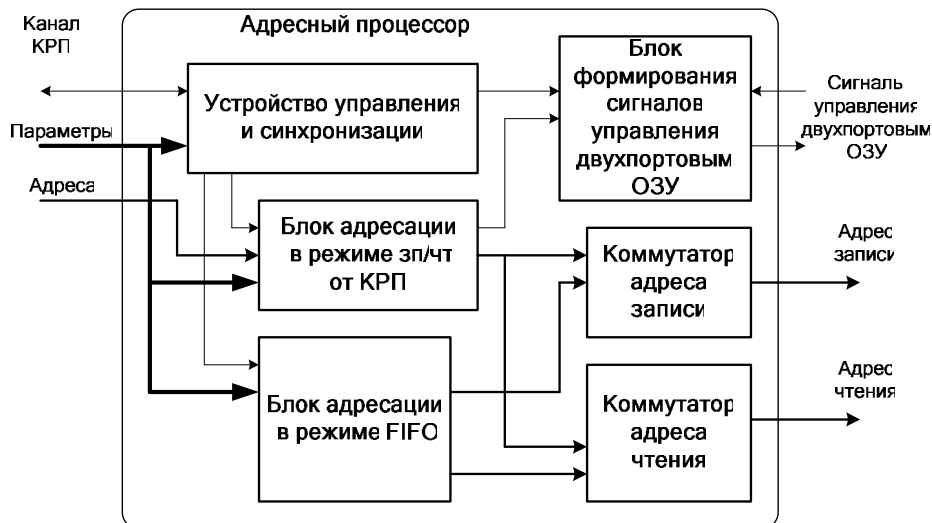


Рис. 5.22. Укрупненная структурная схема адресного процессора

Блок адресации FIFO предназначен для формирования потоков адресов при считывании и записи данных из/в двухпортовое ОЗУ при выполнении алгоритмов БПФ, ОБПФ, быстрой свертки, алгоритмов безызбыточного БПФ. Для формирования адресов в алгоритмах цифровой обработки сигналов при конвейерной организации вычислений требуются довольно сложные

вычисления. Процедурный способ формирования адреса требует иногда нескольких десятков машинных тактов для получения очередного адреса. Поэтому в блоке адресации предусмотрена аппаратная поддержка вычисления адресов с тем, чтобы в каждом машинном такте формировался очередной адрес.

Функциональная схема блока адресации в режиме FIFO показана на рис. 5.23.

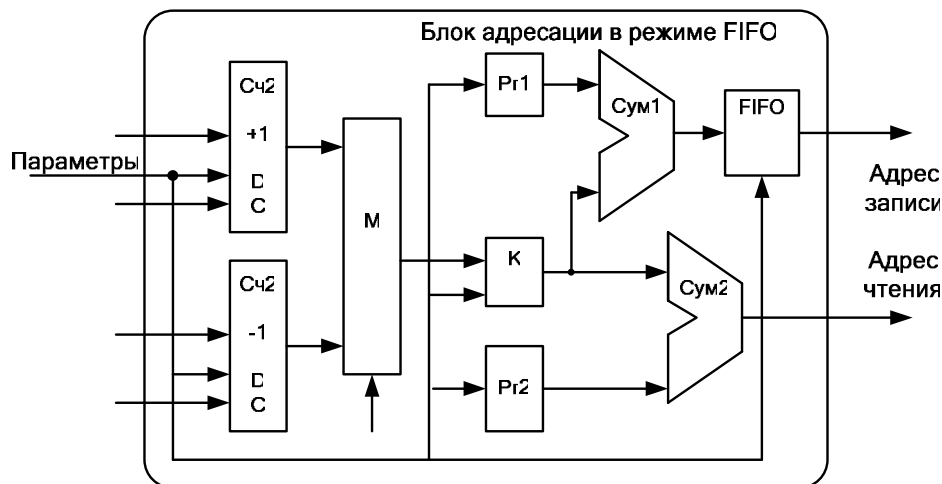


Рис. 5.23. Функциональная схема блока адресации в режиме FIFO

С целью экономии объема двухпортового ОЗУ предусматривается возможность записи результатов обработки информации в те же адреса, из которых было проведено чтение в этом же кадре. Это налагает определенные ограничения на способы чтения данных из ОЗУ. В адресном процессоре предусмотрены несколько режимов адресации. Адресный процессор, благодаря наличию коммутатора, позволяет формировать любые типы адресации, которые могут быть получены путем коммутации разрядов адреса. Теоретическое обоснование такого подхода к формированию адресов в быстрых алгоритмах цифровой обработки сигналов дано в [114]. Поскольку предложенная структура адресного процессора позволяет записывать результаты обработки на те же места, с которых было проведено считывание данных на обработку, то это позволяет сократить объем двухпортового ОЗУ макрообъекта в два раза.

Производительность разработанного макрообъекта ЦОС-7.1 при тактовой частоте 200 МГц для алгоритма БПФ1024 составляет 8 Гфлопс. Предусмотрена возможность комплексирования нескольких макрообъектов ЦОС-7.1 в единую вычислительную структуру с пропорциональным увеличением производительности.

5.7. Макрообъект для решения задач линейной алгебры

5.7.1. Структура макрообъекта для решения задач линейной алгебры

Отличительной особенностью задач линейной алгебры является наличие стандартных вычислительных конструкций, таких как сумма парных произведений, а также отдельных элементов матриц, которые необходимы при вычислениях на каждой итерации алгоритма. Будем в дальнейшем называть такие элементы *ведущими элементами*.

Структура макрообъекта для решения задач линейной алгебры показана на рис. 5.24.

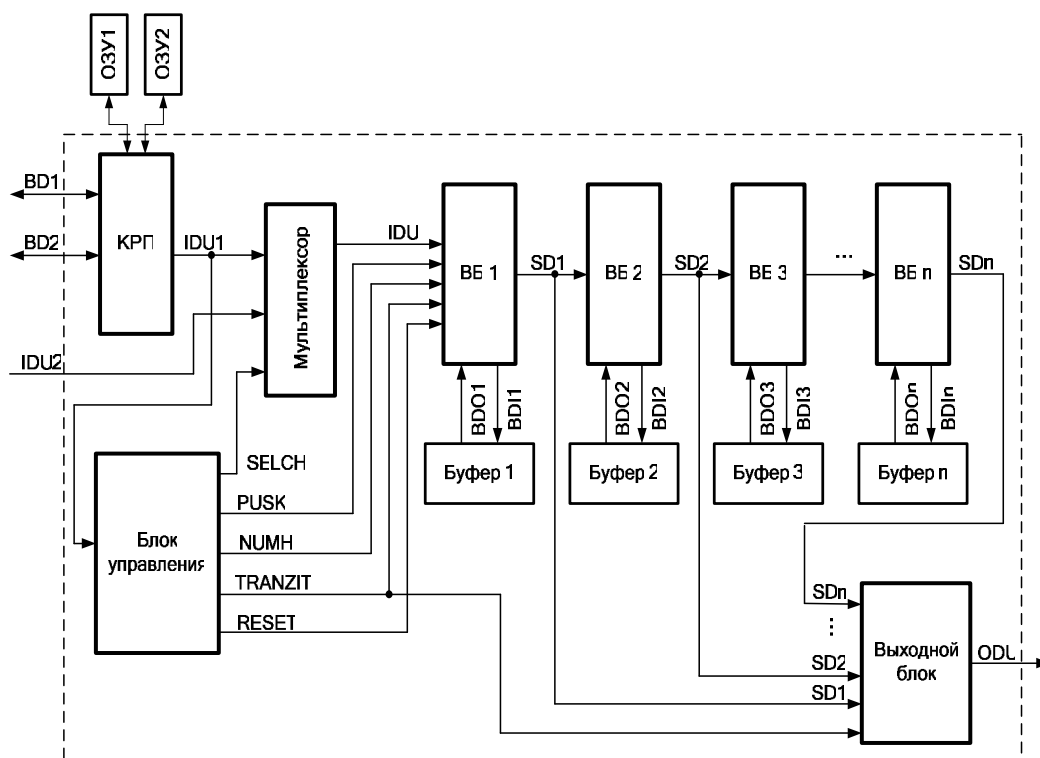


Рис. 5.24. Структура макрообъекта для задач линейной алгебры

На рисунке приняты следующие обозначения:

ВБ 1...ВБ n – вычислительные блоки;

КРП – контроллер распределенной памяти.

Вычислительные блоки выполняют основные функции по обработке данных, поступающих из ОЗУ1 или ОЗУ2. Вычислительные блоки в алгоритмах линейной алгебры имеют структуру, приведенную на рис. 5.25.

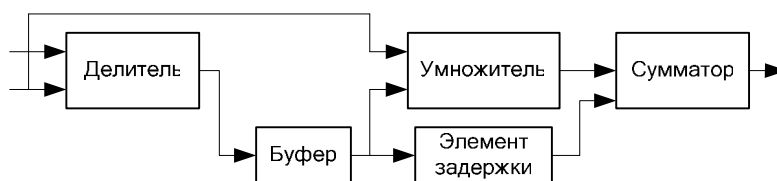


Рис. 5.25. Вычислительный блок для задач линейной алгебры

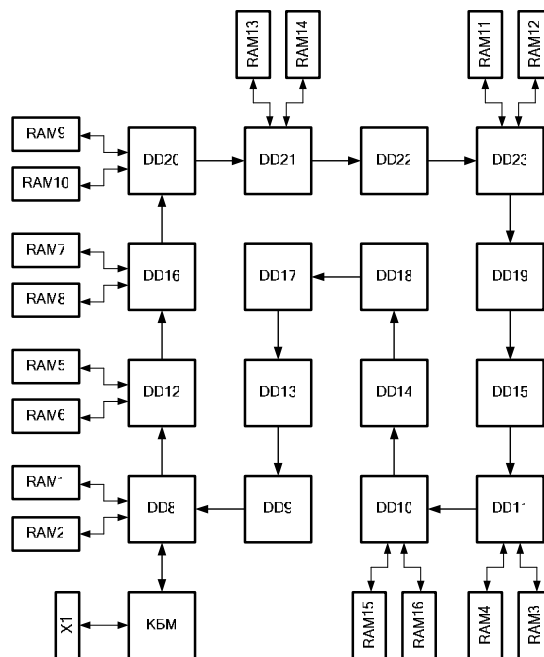
Основной операцией в задачах линейной алгебры является операция суммирования парных произведений [22], которая присутствует в структуре объекта. Количество объектов, задействованных при выполнении какого-либо алгоритма, может изменяться. Также могут быть изменены не только количество объектов, но и их свойства, которые зависят от загружаемых параметров.

Все операции с памятью берет на себя контроллер распределенной памяти, который также осуществляет взаимодействие функциональных узлов между собой и памятью. К контроллеру распределенной памяти подключены, как правило, два блока внешней памяти, в одном из которых хранятся исходные данные, а в другом записываются результаты вычислений. Данные на вход вычислительных блоков подаются с выхода мультиплексора, функционирование которого определяет блок управления. Буферы представляют собой блоки FIFO, хранящие ведущие элементы.

5.7.2. Реализация алгоритма решения СЛАУ методом Гаусса

Рассмотрим пример реализации алгоритма решения системы линейных уравнений методом Гаусса. Конвейерная реализация алгоритма является

наиболее приемлемой для данного класса задач, поэтому алгоритм реализован в виде конвейерной схемы. Для реализации алгоритма был выбран базовый модуль третьего поколения 16V4-50, в рамках которого была задействована конфигурация, показанная на рис. 5.26.



*Рис. 5.26. Конфигурация аппаратных средств
в рамках базового модуля 16V4-50*

В ПЛИС решающего поля были реализованы все фрагменты алгоритма решения системы линейных уравнений методом Гаусса, такие как «прямой ход» алгоритма Гаусса, «обратный ход» алгоритма Гаусса, нормализация ведущей строки, обработка коэффициентов последующих строк, транзит коэффициентов ведущих строк внутри решающей структуры.

Макрообъект функционирует в трех режимах:

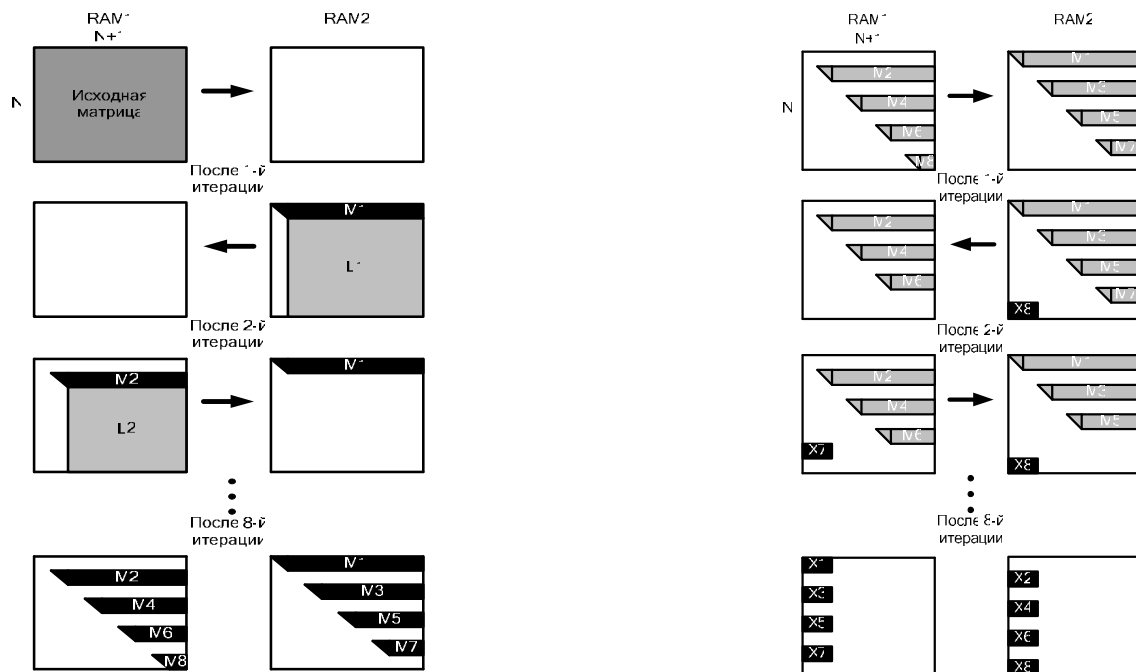
Режим 1. Прямой ход Гаусса или преобразование расширенной матрицы к треугольному виду.

В начальный момент времени исходная матрица размерности $N \times (N + 1)$ расположена в распределенной памяти. В каждой паре блоков памяти один блок занят исходными данными и является источником, а второй - является приемником. В приемнике сохраняются рассчитанные данные после прохождения одной итерации алгоритма. Затем источник и приемник под

управлением контроллера памяти меняются местами и начинается следующая итерация алгоритма. Количество итераций определяется количеством ступеней обработки K . Следовательно, количество итераций может быть определено по формуле

$$I = \frac{N \times (N + 1)}{K}. \quad (5.13)$$

При прохождении очередной итерации в память-приемник записываются K ведущих строк и $N - K$ ведомых строк. На рис. 5.27 показано расположение данных в памяти до начала работы алгоритма, в процессе обработки и после завершения восьми итераций. Как видно из рис. 5.27а, в начале работы данные, проходя через обрабатывающие блоки, попадают из памяти RAM1 в память RAM2, при этом сначала в память записываются ведущие строки M1, а потом ведомые L1. Рассчитанные ведущие строки в дальнейшей работе прямого хода алгоритма участия не принимают. Пустое пространство в массиве памяти резервируется для дальнейшей работы алгоритма.



а) прямой ход алгоритма Гаусса

б) обратный ход алгоритма Гаусса

Рис. 5.27. Расположение данных в памяти

Режим 2. Обратный ход Гаусса или преобразование треугольной матрицы к диагональной.

Схема работы обратного хода алгоритма аналогична прямому, с той лишь разницей, что в памяти будут храниться не массивы данных, а результаты расчетов искомых коэффициентов, что отражено на рис. 5.27 б. При обратном ходе алгоритма рассчитанные ведущие строки, начиная с последней, проходя через решающие блоки схемы, преобразуются в искомые коэффициенты. Это и есть результат работы алгоритма. Данный подход имеет место как при использовании одной пары блоков памяти, так и для нескольких пар.

Режим 3. Транзит коэффициентов ведущих строк.

Этот режим необходим для того, чтобы осуществлять обработку матрицы, строки которой размещены более чем в одной паре ОЗУ. Пусть, например, одна половина матрицы лежит в RAM1, а вторая - в RAM5. Поскольку во время одной итерации через решающую схему должна пройти вся матрица при условии, что она разбита на два или более блоков памяти, необходимо обеспечить соответствие коэффициентов рассчитанных ведущих строк данным, находящимся в памяти RAM5. На рис. 5.28 показана структура кольцевого буфера.

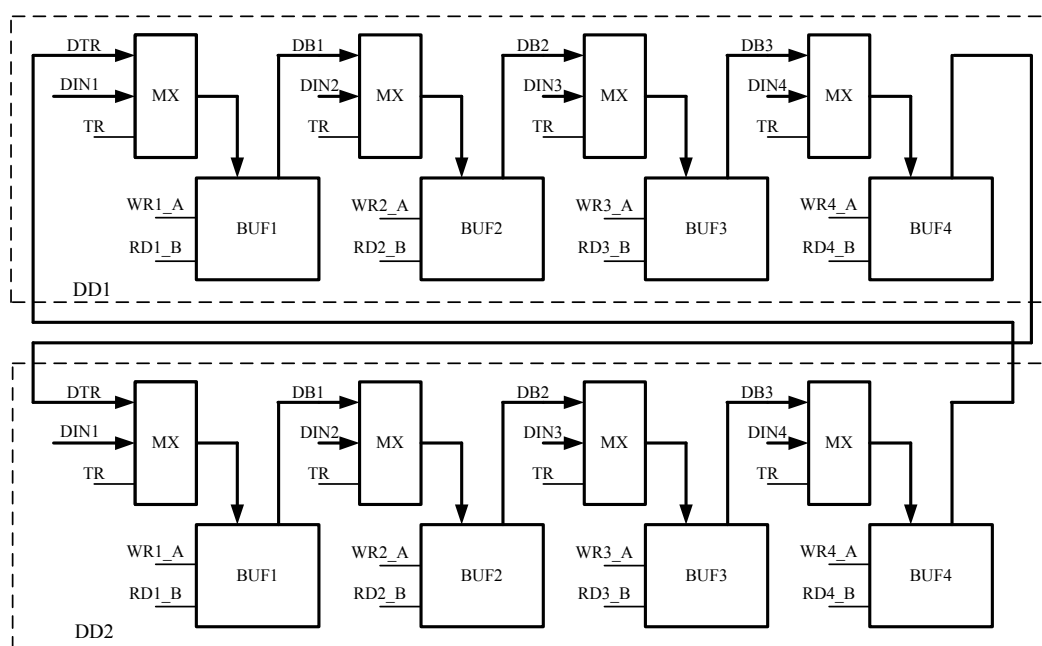


Рис. 5.28. Структура кольцевого буфера ПЛИС DD1 и DD2

Блоки BUF 1- BUF 4 представляют собой двухпортовую память. Обмен данными осуществляется через мультиплексоры MX, которые коммутируют транзитные шины. В режиме прямого или обратного хода буферы записывают и хранят данные, рассчитанные в решающих блоках. В режиме транзита мультиплексор переключает буферы на транзитные шины DTR и ODU, замыкая их в кольцевой буфер. Таким образом, можно переместить необходимое количество данных одновременно из всех буферов ПЛИС DD1 в ПЛИС DD2 и т.д. Рассчитанные коэффициенты ведущих строк располагаются в нужных буферах, и управление всей цепью ступеней алгоритма передается следующему блоку, который продолжает обработку расположенных в нем данных. Объем памяти, необходимый для реализации алгоритма, определяется как $2N^2$ слов, размер слова равен 4 байта. При $N = 10^4$, объем памяти составит 800 Мбайт. При $N = 10^5$ объем памяти составит 80 ГБ. Время решения задачи на базовом модуле 16V4-50 (см. подраздел 4.4.3.1) при $N = 10^4$ и частоте 160MHz составляет 49 с. Для решения задачи СЛАУ размерностью $N = 10^5$ необходим объем распределенной памяти, равный 80 Гбайт. Время решения СЛАУ задачи размерностью на одном базовом модуле 16V4-50 составляет $N = 10^5 - 13,5$ часов.

5.8. Универсальный макрообъект

Особое место среди проблемно-ориентированных макрообъектов занимает универсальный макрообъект или, иными словами, макропроцессор (МАП). У универсального макрообъекта имеются определенные свойства, выделяющие его из множества проблемно-ориентированных и специализированных макрообъектов. Приведем особенности универсального макрообъекта:

- структура макропроцессора содержит неизменное количество вычислительных ядер - элементарных процессоров (ЭП), обладающих определенными свойствами;

- параметры элементарных процессоров (разрядность операндов, система команд, состав входных и выходных шин) не изменяются в зависимости от предметной области;

- параметры внутреннего коммутатора универсального макропроцессора не изменяются в зависимости от предметной области;

- с точки зрения программиста макропроцессор является программно-неделимой единицей, то есть изменение команд для всех элементарных процессоров и коммутатора выполняется одновременно в момент смены макрооперации.

Макропроцессор сохраняет свою структуру независимо от размещения в той или иной ПЛИС базового модуля. Неизменность структуры макропроцессора компенсируется универсальной системой команд для элементарных процессоров и наличием программируемого коммутатора, позволяющим соединять элементарные процессоры в различные конвейерно-параллельные структуры. Это дает возможность для различных классов задач сформировать эффективные параллельно-конвейерные вычислительные структуры. Конечно, эффективность этих структур будет несколько ниже, чем при реализации таких же структур из макрообъектов, не имеющих перечисленных выше ограничений. Однако меньшая эффективность компенсируется тем, что требуется всего один конфигурационный файл для всех ПЛИС базового модуля и реконфигурируемой вычислительной системы в целом, что приводит к снижению стоимости разработки.

5.8.1. Структура макропроцессора

Универсальный макропроцессор предназначен для структурной реализации крупных функционально законченных фрагментов задач. В общем случае макропроцессор состоит из множества внутренних элементарных процессоров и программируемого коммутатора каналов, который обеспечивает связь между элементарными процессорами, а также между элементарными

процессорами и выводами макропроцессора. Более подробно элементарный процессор описан в параграфе 5.8.4.

Существуют различные варианты построения внутреннего программируемого коммутатора макропроцессора. Очевидно, что коммутации по полному графу могут быть реализованы в макропроцессоре с ограниченным набором элементарных процессоров.

Рассмотрим макропроцессор, в котором соединения внутренних элементов и выводов производятся по полному графу. Пусть, для определенности, в состав макропроцессора входит четыре элементарных процессора и имеется по одному входу/выходу в каждом из ортогональных направлений. Граф коммутации в этом случае будет содержать 16 вершин. Функциональная схема макропроцессора с коммутацией по полному графу показана на рис. 5.29.

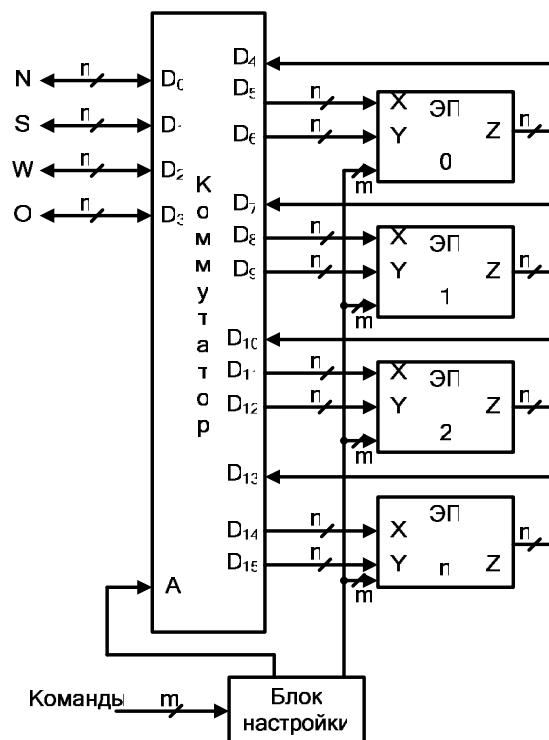


Рис. 5.29. Схема макропроцессора с коммутацией по полному графу

Коммутатор обеспечивает соединения между:

- входами и выходами элементарных процессоров;
- выводами макропроцессора, каждый из которых в общем случае является двунаправленным;

- выводами макропроцессора, запрограммированными на вход, и входами элементарных процессоров;

- выходами элементарных процессоров и выводами макропроцессора, запрограммированными на выход.

Макропроцессоры с коммутацией по полному графу реализовывались в базовых модулях второго поколения, так как ПЛИС, на которых они строились, обладали ограниченным схмотехническим ресурсом, не позволяющим реализовать макропроцессор с большим количеством элементарных процессоров.

5.8.2. Макропроцессор с каскадной коммутационной структурой

Появление современных ПЛИС, содержащих несколько миллионов эквивалентных вентилях, позволяет реализовать макропроцессоры, в составе которых может быть реализовано от 16 до 40 элементарных процессоров и до четырех 32-разрядных каналов передачи данных по каждому из направлений. Коммутационный граф в этом случае будет содержать 128 вершин. Создать коммутатор, соединяющий по полному графу все 128 вершин каналами шириной 32 разряда, на современном этапе развития ПЛИС не представляется возможным. Поэтому необходимо каким-то образом ограничить сложность коммутатора. В качестве альтернативы полному коммутатору предлагается использовать каскадный коммутатор, который обеспечивает соединение элементов внутри макрообъекта по определенным правилам. Для конкретности будем предполагать, что в состав макропроцессора входят 16 элементарных процессоров и 16 внешних каналов передачи данных – по четыре канала по каждому из ортогональных направлений.

Элементарные процессоры макрообъекта предлагается сгруппировать в каскадные линейки L0-L3. В каждой линейке будут расположены по четыре элементарных процессора – два аддитивного типа и два мультипликативного типа. Каскады макропроцессоров соединены последовательно, образуя кольцо, поэтому выходы элементов линейки L3 поступают на входы элементов L0.

Внешние выводы макрообъекта распределяются по четыре на каждое направление N, S, W, O.

Поскольку коммутационная структура макрообъекта ограничена и не позволяет соединять элементы несмежных каскадов непосредственно друг с другом, целесообразно ввести такой внутренний элемент макрообъекта как транзитная шина. В макрообъекте для каждого каскада имеется по четыре транзитных шины T0-T3, которые обеспечивают транзитную передачу данных между каскадами и выводами макрообъекта. В зависимости от реализуемой макрооперации каждый вывод макрообъекта может быть запрограммирован как на вход, так и на выход. С помощью транзитных шин информация может проходить через макрообъект без числовой обработки. Каскады коммутатора K0-K3 обеспечивают связи между элементарными процессорами, транзитными шинами и выводами макрообъекта.

Функциональная схема макропроцессора без контроллера распределенной памяти представлена на рис. 5.30.

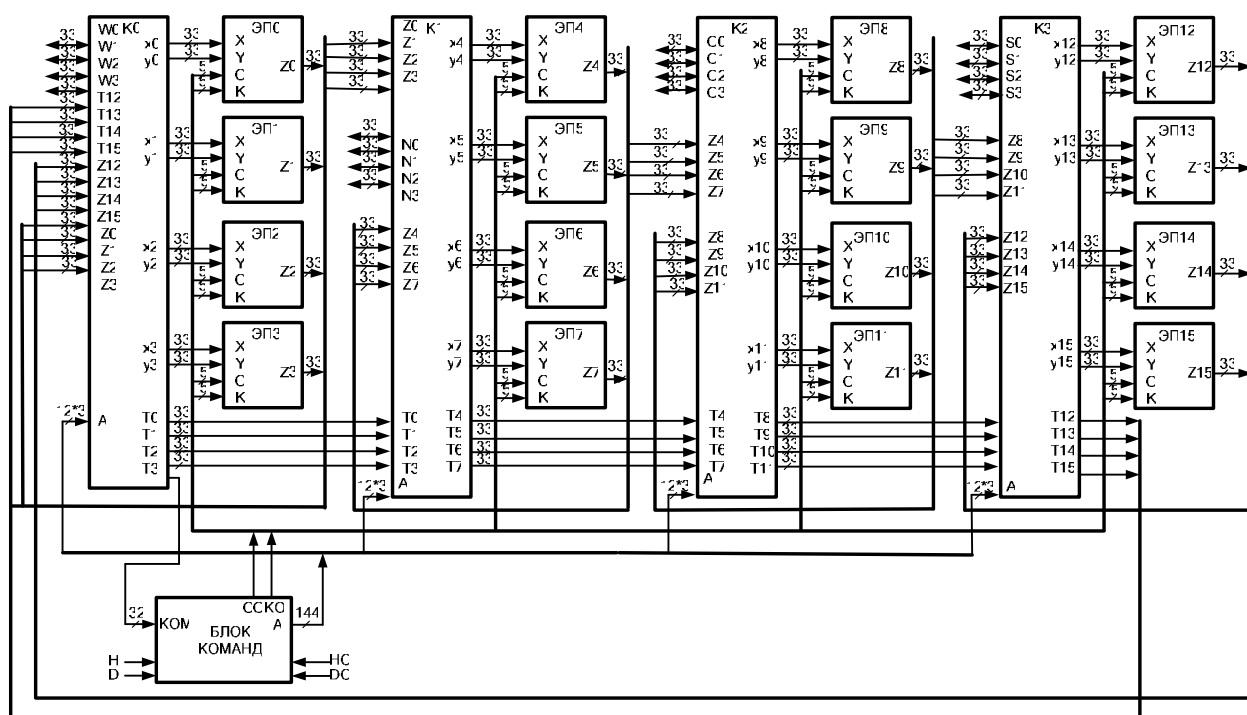


Рис. 5.30. Функциональная схема макропроцессора без контроллера памяти

Если к ПЛИС, в которой размещается макропроцессор, подключены микросхемы ОЗУ, образующие распределенную память, то эти ОЗУ подключаются к внешним каналам данных соответствующего направления. В

этом случае в состав макропроцессора включается такое же количество контроллеров распределенной памяти. Количество подключаемых ОЗУ, как правило, не превышает двух на одну ПЛИС. Функциональная схема макропроцессора с включенными в его состав двумя контроллерами распределенной памяти показана на рис. 5.31. Процессор содержит:

- два контроллера распределенной памяти КРП0-КРП1;
- двенадцать элементарных процессоров ЭП0-ЭП11;
- четыре каскада коммутатора К0-К3;
- блок команд.

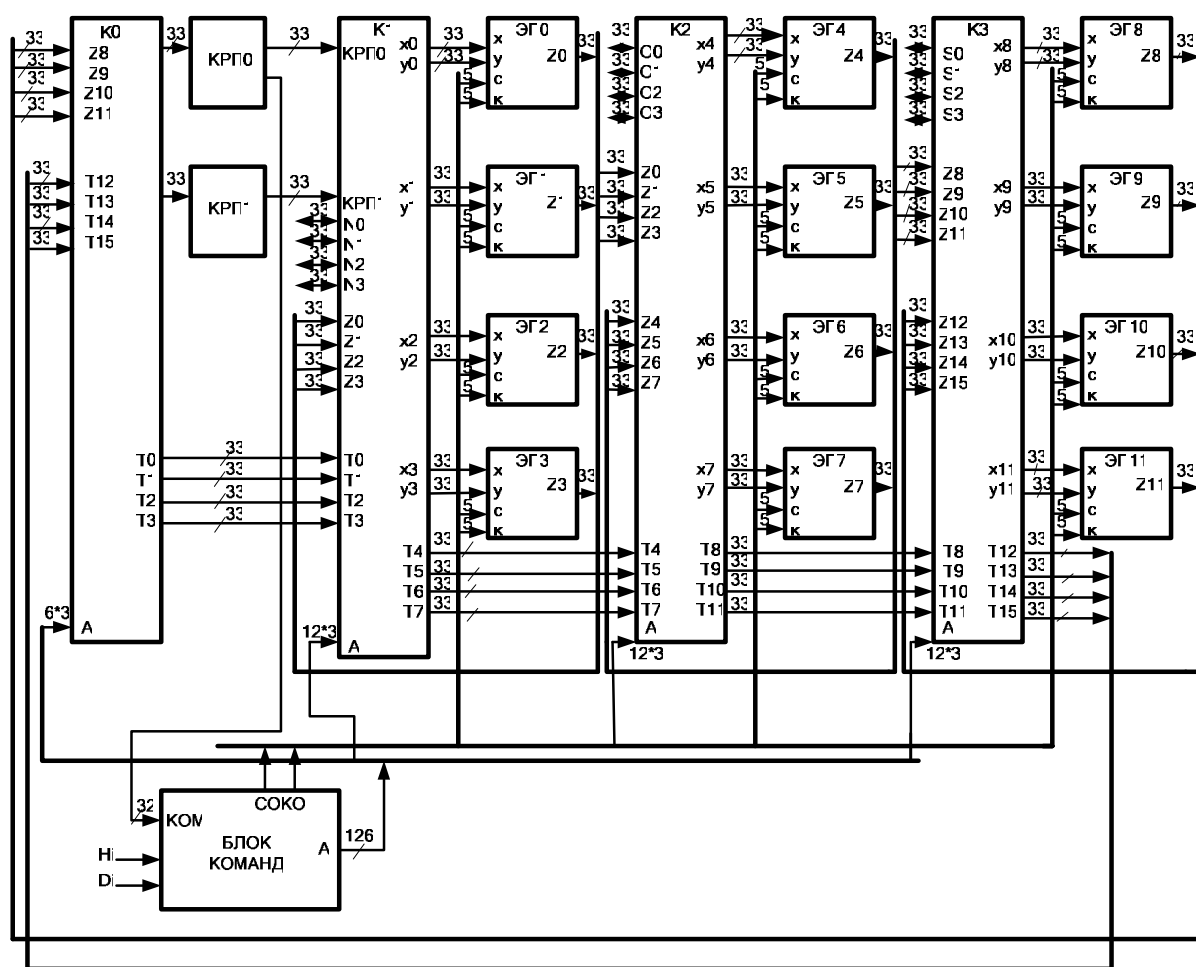


Рис. 5.31. Функциональная схема макропроцессора с двумя контроллерами памяти

ОЗУ подключаются к структуре макропроцессора через контроллер памяти. Посредством каскадов коммутатора данные из ОЗУ через контроллер

могут передаваться во все компоненты макропроцессора, а также транзитом на выходные шины.

5.8.3. Каскадный коммутатор макропроцессора

Функционально коммутатор макропроцессора можно представить состоящим из трех типов 33-разрядных мультиплексоров.

Мультиплексоры первого типа обеспечивают связь выходов элементарных процессоров с выводами макрообъекта. На входы мультиплексоров поступает информация с выходов элементарных процессоров, а также транзитных шин.

Мультиплексоры второго типа обеспечивают связь выходов различных элементов макропроцессора с входами элементарных процессоров рассматриваемого каскада макрообъекта. На входы мультиплексоров поступает информация с соответствующих входов макрообъекта, выходов элементарных процессоров предшествующей линейки, транзитных шин, расположенных в предшествующем каскаде, а также выхода соответствующего элементарного процессора, расположенного в рассматриваемом каскаде.

Мультиплексоры третьего типа обеспечивают связь выходов различных элементов макропроцессора с транзитными шинами рассматриваемого каскада макрообъекта. На входы мультиплексоров поступает информация с соответствующих входов макрообъекта, выходов элементарных процессоров предшествующего каскада, а также соответствующих транзитных шин.

Всего коммутатор макропроцессора содержит 16 мультиплексоров первого типа, 32 мультиплексора второго и 16 мультиплексоров третьего типа. По сравнению с коммутационной структурой по полному графу предлагаемая каскадная коммутационная структура требует существенно меньше ресурсов для коммутаторов, обеспечивающих соединения между элементами макрообъекта, в результате чего больший объем ресурсов можно использовать для вычислений. Однако нельзя не отметить, что программирование макрообъектов с каскадными коммутационными структурами вызывает некоторые сложности.

5.8.4. Элементарный процессор

Элементарный процессор предназначен для выполнения арифметических и логических операций над двумя операндами, поступающими на его входы в формате 32-разрядных чисел с плавающей запятой. Каждый операнд сопровождается однобитным маркером. Наличие маркера подтверждает передачу числа по каналу данных. Фактически передача данных по каналам производится 33-разрядными словами. В состав элементарного процессора входят блок операндов, арифметико-логическое устройство (АЛУ) и преобразователь. Структурная схема элементарного процессора приведена на рис. 5.32.

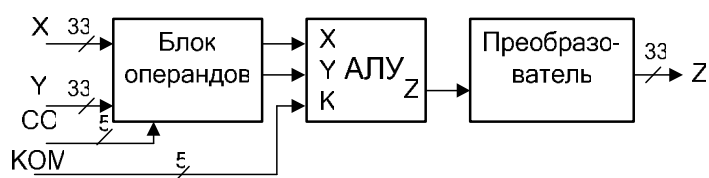


Рис. 5.32. Структурная схема элементарного процессора

АЛУ предназначено для выполнения арифметических и логических операций, а также для вычисления предикатов сравнения операндов, поступающих на его входы. Список операций, выполняемых АЛУ, можно разделить на четыре группы: арифметические операции, логические операции, операции сравнения и специальные операции.

Список арифметических операций включает: сложение, вычитание, умножение, деление, остаток от деления, подсчет числа единиц в числе.

Список логических операций: поразрядная конъюнкция, поразрядная дизъюнкция, сложение по модулю 2, логический сдвиг, циклический сдвиг.

Список операций сравнения: больше, не меньше, равно, не равно.

Список специальных операций: масштабирование, синхронизация данных, очистка буферной памяти.

С помощью приведенного списка операций макропроцессора пользователь может реализовать макрооперации из различных предметных областей. Существует возможность изменять и дополнять список операций, выполняемых арифметико-логическим устройством элементарного процессора.

Основное назначение блока операндов - выравнивание потоков данных на входе АЛУ. В ходе вычислений потоки операндов, поступающие на входы элементарных процессоров, могут проходить пути, имеющие различную операционную задержку, что приводит к опережению или запаздыванию одного потока относительно другого. Блок операндов на основании анализа маркеров обоих потоков выполняет функции задержки опережающего потока данных на нужное количество тактов, так что на входы АЛУ потоки приходят одновременно. Блок операндов синхронизирует рассогласованные потоки операндов с помощью буферной памяти, в которой накапливаются опережающие операнды. Существует два режима работы блока операндов:

- режим синхронного прихода операндов, когда операнды в элементарные процессоры поступают синхронно;
- режим синхронизации с памятью, когда операнды в элементарные процессоры приходят рассогласованно.

5.8.5. Блок команд макропроцессора

Блок команд записывает и хранит макрооперацию на время ее выполнения в макропроцессоре. Макрооперация содержит коды операции каждого элементарного процессора, адреса входных операндов, адреса для транзитных шин и адреса выходов макрообъекта. Блок команд, структурная схема которого представлена на рис. 5.33, состоит из двадцати двух сдвиговых регистров. Каждый регистр содержит четыре байта. Первые шестнадцать регистров хранят команды элементарного процессора.

Каждая команда содержит код операции (два байта) и адреса входных операндов для элементарного процессора (байт). Два регистра хранят адреса шестнадцати транзитных шин. Остальные четыре регистра предназначены для хранения адресов источников 16-ти внешних выводов. Загрузка команды осуществляется параллельно по четыре байта в регистр и за двадцать два такта макрокоманда последовательно заносится в регистровую память процессора по входу W0.

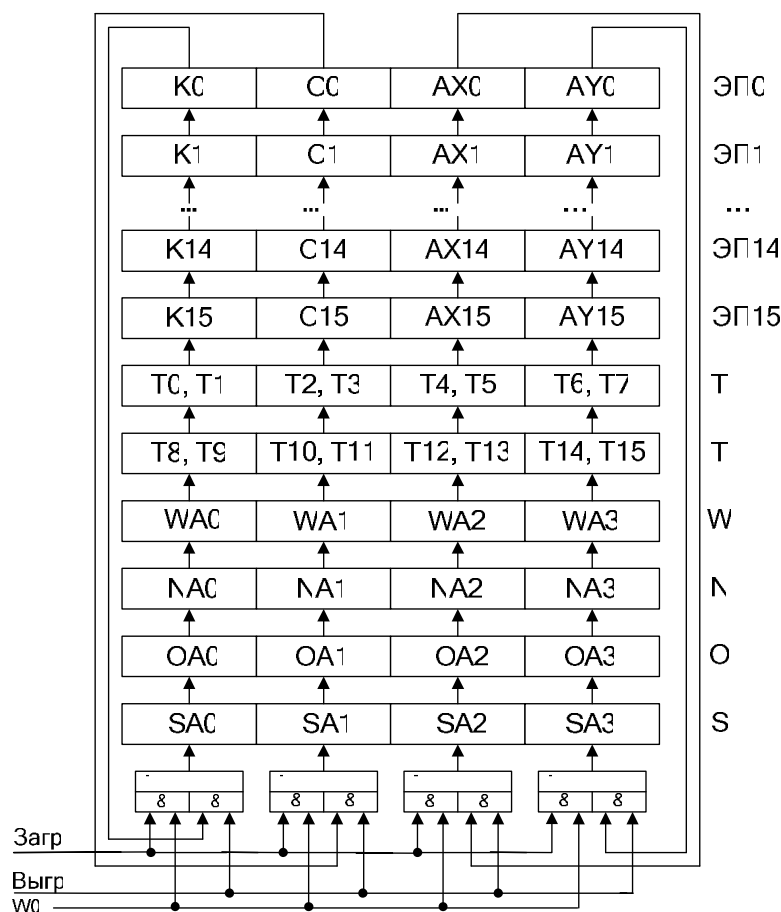


Рис. 5.33. Структурная схема блока команд

Блок команд для обоих видов макрообъектов одинаков, но для макрообъекта, в состав которого входит контроллер распределенной памяти, поля для ЭП0-ЭП3 не используются. Для корректной загрузки команд обязательным условием является единица в шестом бите поля K0.

5.8.6. Блок операндов

В реконфигурируемой структуре существуют две независимые подсистемы синхронизации вычислений – система синхронизации кадров и система синхронизации потоков данных. Система синхронизации кадров реализуется на программном уровне и рассматривается в главе 6. Здесь же более подробно рассмотрим синхронизацию потоков операндов в мультиконвейерных структурах, создаваемых из элементарных процессоров.

Система синхронизации потоков данных реализуется в блоках операндов, входящих в состав каждого элементарного процессора. Рассогласование потоков данных на входе процессора возникает вследствие того, что входные

операнды являются результатом вычислений в различных ветвях мультиконвейерной вычислительной структуры, имеющих различную операционную задержку. Блоки операндов осуществляют синхронизацию рассогласованных потоков данных с помощью буферной памяти, внося задержку в опережающий поток данных. Структурная схема блока операндов представлена на рис. 5.34. В состав блока операндов входят блок ввода, блок управления ОЗУ, буферная память – ОЗУ и выходной коммутатор.

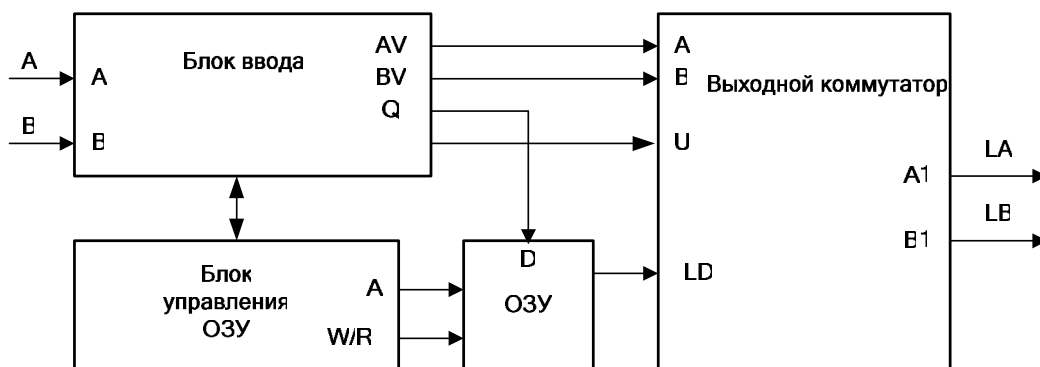


Рис. 5.34. Функциональная схема блока операндов

В блоке ввода формируются режимы синхронизации, маркеры потоков данных A и B и определяется маркер опережающего потока. Если оба потока данных поступили одновременно, то блок операндов выдает их на выходы LA и LB без дополнительной задержки. Если один из потоков появился раньше другого, то опережающий поток данных записывается в ОЗУ до появления второго потока данных. После поступления на вход блока данных запаздывающего потока данных из ОЗУ начинает считываться записанный туда опережающий поток, и на выходы LA и LB оба потока выдаются одновременно с задержкой, равной рассогласованию этих потоков. Таким образом, устраняется рассогласование потоков данных на входе АЛУ элементарного процессора.

Блок операндов, помимо функции синхронизации рассогласованных потоков данных, выполняет и некоторые другие функции, например, функцию хранения и выдачи в элементарный процессор констант или векторов констант. В этом случае константы должны быть предварительно записаны в ОЗУ блока операндов.

В этой главе были рассмотрены структуры нескольких макрообъектов из таких предметных областей, как математическая физика, цифровая обработка сигналов, линейная алгебра. Рассмотрена также структура универсального макрообъекта. При использовании предложенных макрообъектов с помощью формальных методов могут создаваться мультиконвейерные вычислительные структуры, оптимально соответствующие информационному графу решаемой задачи из конкретной предметной области. При этом гарантируются достаточно полная загрузка аппаратного ресурса реконфигурируемой структуры и, как следствие, высокие показатели реальной производительности вычислительной системы.

Глава 6. Системное математическое обеспечение реконфигурируемых мультиконвейерных вычислительных структур

6.1. Структура системного математического обеспечения

Как показано в предыдущих главах, в реконфигурируемых мультиконвейерных вычислительных структурах реализованы структурно-процедурные принципы организации вычислений. При этом каждый кадр представляет собой структурно (аппаратно) реализованный в базовых модулях РМВС подграф информационного графа решаемой задачи, а последовательность отображения кадров выполняется процедурно. Такая структурно-процедурная реализация вычислений в реконфигурируемых мультиконвейерных вычислительных структурах (РМВС) не позволяет использовать стандартные средства программного обеспечения традиционных многопроцессорных вычислительных систем, которые, в основном, ориентируются на реализацию обычных мультипроцедурных вычислений.

Поэтому очень важной проблемой, направленной на расширение круга пользователей РМВС, является создание такого системного программного обеспечения, которое обеспечило бы удобство программирования сложных задач и в идеале по сложности приблизило бы этот процесс к традиционному программированию многопроцессорных систем кластерного типа.

Системное программное обеспечение, которое должно обеспечить пользователям удобство программирования сложных задач для реконфигурируемых мультиконвейерных вычислительных структур, функционально можно разделить на три группы:

- средства разработки прикладных программ;
- средства администрирования вычислительных ресурсов;
- служебные программы и драйверы.

Структура системного программного обеспечения приведена на рис. 6.1.

Если для традиционных вычислительных систем, в том числе многопроцессорных систем, программный компонент реализуется во времени, то для РМВС характерно пространственное программирование, которое сродни схмотехническим решениям. Пользователь должен создавать в рамках реконфигурируемой архитектуры свою проблемно-ориентированную систему.



Рис. 6.1. Структура системного программного обеспечения для РМВС

Известно, что *программирование в пространстве* - создание конфигурационных файлов для программирования ПЛИС - является трудоемким процессом, требующим специальных знаний и навыков. Поэтому основной целью комплекса средств разработки прикладных программ для РМВС является предоставление пользователю возможностей, которые позволили бы создавать прикладные программы без привлечения специальных знаний в области схмотехники ПЛИС и по сложности были бы приближены к традиционному программированию многопроцессорных ЭВМ. Программный комплекс должен обеспечить эффективную реализацию вычислительно трудоемких фрагментов задач различных проблемных областей на произвольном количестве базовых модулей, построенных на ПЛИС. Для этого должны быть созданы удобные средства написания и отладки прикладных программ на языках структурно-процедурных вычислений.

Язык структурно-процедурного программирования низкого уровня Argus v.3.0 представляет собой ассемблер для PMBC, позволяющий осуществлять низкоуровневый доступ к аппаратному обеспечению базовых модулей. Язык Argus v.3.0 необходим для контроля схмотехнических реализаций и создания программ, использующих топологические и архитектурные особенности базовых модулей PMBC. В то же время следует отметить, что программисту, использующему язык Argus, необходимо четко представлять аппаратные возможности вычислительных структур, как, впрочем, и любому программисту, пишущему на языке ассемблера для традиционной MBC. Для того чтобы избавить программиста от необходимости знания аппаратной части PMBC, разработан и используется язык высокого уровня COLAMO.

Язык структурно-процедурного программирования высокого уровня COLAMO v.2.0 позволяет создавать эффективные прикладные программы для PMBC пользователям, недостаточно представляющим архитектурные особенности вычислителя. Язык построен на основе неявного описания параллелизма. Отображение алгоритма на архитектуру PMBC обеспечивает транслятор, который реализует синтаксическую поддержку реконфигурации аппаратной платформы, опираясь на элементы библиотеки масштабируемых IP-ядер. Библиотеки масштабируемых IP-ядер используются для создания вычислительных структур в аппаратном ресурсе PMBC. Транслятор COLAMO v.2.0 осуществляет перевод исходного кода программы с языка высокого уровня в язык ассемблера Argus с последующим созданием структурных и процедурных компонентов параллельной программы для PMBC.

Интегрированная среда разработки Argus IDE предназначена для интерактивной разработки параллельных программ на языках высокого уровня COLAMO и языке ассемблера Argus. Среда Argus IDE обеспечивает эффективную разработку масштабируемых параллельных программ для PMBC в едином языковом пространстве и объединяет в своем составе трансляторы языков COLAMO и Argus. В процессе трансляции параллельной COLAMO-

программы транслятор формирует несколько компонентов, реализуемых в различных узлах РМВС:

- структурный компонент – задает информационный граф алгоритма, описанного в программе, который должен быть реализован в виде вычислительной структуры кадра;
- потоковый компонент – описывает (на языке Argus) потоки данных, которые должны протекать через вычислительную структуру кадра;
- кадровый компонент – описывает (на языке Argus) очередность смены кадров при структурно-процедурной организации вычислений;
- управляющий компонент – задает (на языке C++, Object Pascal, и т.д.) процедуры настройки РМВС, пуска, останова и другие управляющие и сервисные функции.

Если в исходном коде параллельной программы транслятором были найдены ошибки, то список ошибок и предупреждений возвращается среде Argus IDE с целью их устранения программистом.

Система удаленного доступа и управления вычислительными ресурсами РМВС состоит из сервера, поддерживающего очередь заявок и обеспечивающего обработку удаленных заявок на использование вычислительных ресурсов. К функциям системы удаленного доступа относятся функции включения, выключения, остановки и запуска как отдельных базовых модулей, так и блоков, стоек и РМВС в целом.

Драйвер обеспечивает программную поддержку функций доступа к высокоскоростному аппаратному интерфейсу. С целью обеспечения скоростных режимов работы с аппаратным интерфейсом драйвер поддерживает функционирование шины в пакетных режимах работы и обеспечивает механизмы прямого доступа к физической памяти управляющего компьютера. Кроме того, драйвер выполняет поддержку режимов энергосбережения.

Отладчик параллельных программ предназначен для комплексной отладки программ, функционирующих на множестве базовых модулей РМВС. К основным функциям отладчика можно отнести следующие:

- поддержку точек останова для контроллеров памяти, в том числе и условных;
- обеспечение полного доступа к памяти базовых модулей в любой момент времени;
- обеспечение выполнения и возможности настройки команд пуска, останова, выгрузки промежуточных результатов.

Программный интерфейс представляет собой основной набор функций доступа к базовым модулям. Все системное и прикладное программное обеспечение основывается на использовании программного интерфейса. Независимо от низкоуровневой программно-аппаратной реализации функций доступа к базовым модулям программный интерфейс остается без изменений с целью обеспечения переносимости системного и прикладного программного обеспечения. С целью поддержки особенностей различных аппаратных интерфейсов набор функций программного интерфейса может расширяться.

Обобщенная структурная схема взаимодействия компонентов комплекса средств разработки приведена на рис. 6.2.

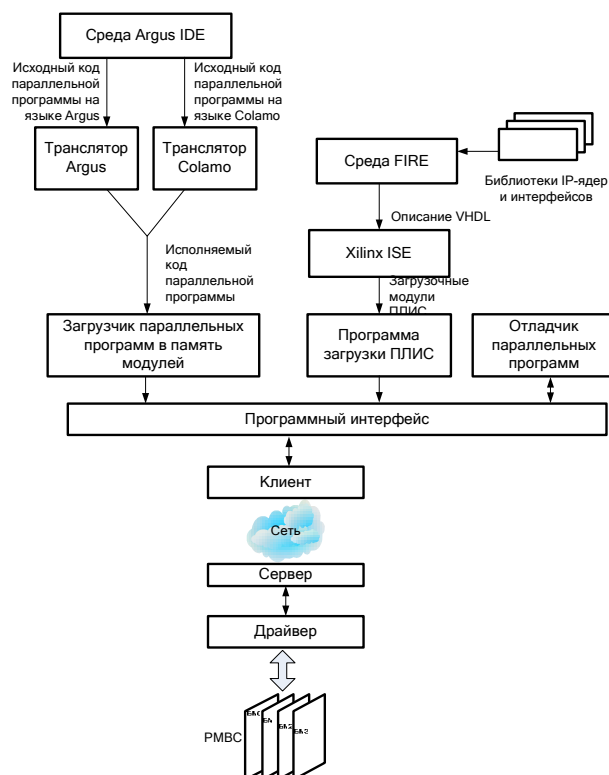


Рис. 6.2. Структурная схема взаимодействия компонентов комплекса разработки программ

Для реализации структурного компонента, полученного в результате трансляции программы с языка COLAMO, среда Argus IDE осуществляет вызов другой среды – Fire!Constructor, которая предназначена для построения прикладных вычислительных структур.

В среде Fire!Constructor на основе библиотеки IP-ядер и интерфейсов реализуется структурный компонент в виде вычислительной структуры кадра. Структурный компонент передается в Fire!Constructor в виде информационного графа, полученного в результате трансляции COLAMO-программы. Вычислительная структура создается из информационного графа путем замены его вершин на библиотечные IP-ядра вычислительных узлов и замены дуг информационного графа соответствующими библиотечными интерфейсами. При этом обязательно учитывается паспорт (описание) базового модуля, для которого выполняется создание вычислительной структуры кадра. Паспорт базового модуля включает информацию о количестве и типах ПЛИС в его составе, о конфигурации и количестве связей между ПЛИС и другими компонентами базового модуля, о наличии и типах микросхем ОЗУ, о количестве и типах внешних каналов связи и т.п. Fire!Constructor автоматически выполняет распределение созданной вычислительной структуры по ПЛИС базового модуля и осуществляет трансляцию этих фрагментов в язык описания аппаратуры (VHDL). Далее с помощью типовых синтезаторов конфигурации ПЛИС (таких как Xilinx ISE) производится создание конфигурационных файлов для всех ПЛИС базового модуля. Загрузка конфигурации всех ПЛИС производится с помощью специальной программы загрузчика.

Среда Fire!Constructor существенно сокращает время разработки и трудоемкость прикладных программ для PMBC. Это происходит за счет выполнения синтезатором ряда трудоемких процедур:

- согласования uscf-файлов совместно работающих ПЛИС;

- автоматической синхронизации информационных потоков при размещении функциональных устройств единого вычислительного контура, входящих в различные ПЛИС;

- автоматического обеспечения сбалансированного размещения функциональных устройств по различным ПЛИС.

Автоматический синтез вычислительной структуры ПЛИС и синтез многокристальных вычислительных структур в пределах базового модуля позволит, в принципе, исключить из процесса создания параллельной программы для РМВС квалифицированных специалистов-схемотехников, необходимых при создании схемотехнических решений для всех ПЛИС базового модуля РМВС в целом.

Потоковый компонент формируется при трансляции COLAMO-программы в виде исполнительного модуля для контроллеров распределенной памяти, которые управляют подключенными к ним ОЗУ и реализуют процедуры чтения входных потоков данных и записи результатов вычислений в процессе реализации кадра. Исполняемые модули загружаются в память контроллеров распределенной памяти либо с помощью отладчика, либо с помощью специальной программы-загрузчика и управляют контроллерами в процессе работы РМВС.

Программный доступ к базовым модулям осуществляется через единый программный интерфейс, позволяющий использовать функции загрузки/выгрузки и управления вычислительным процессом на традиционных языках программирования (C++, Object Pascal, Java и т.д.) с помощью подключаемых к проекту библиотек интерфейса. Программную поддержку аппаратного интерфейса доступа к базовым модулям обеспечивает драйвер базовых модулей.

Удаленный доступ к базовым модулям осуществляется через систему многопользовательского доступа. Система реализует доступ к базовым модулям через набор протоколов прикладного уровня.

По функциональным признакам системное программное обеспечение можно разделить на три группы (рис. 6.3):

- средства разработки и взаимодействия с пользователем;
- среда исполнения;
- средства взаимодействия с вычислительными ресурсами распределенной системы.

Создание параллельных программ осуществляется в интегрированных средах разработки, включающих в себя средства ввода и редактирования исходных текстов и граф-схем, трансляторы в объектное представление, конверторы между графической и текстовой формой описания, средства доступа к базе компонентов и отладчик параллельных программ.

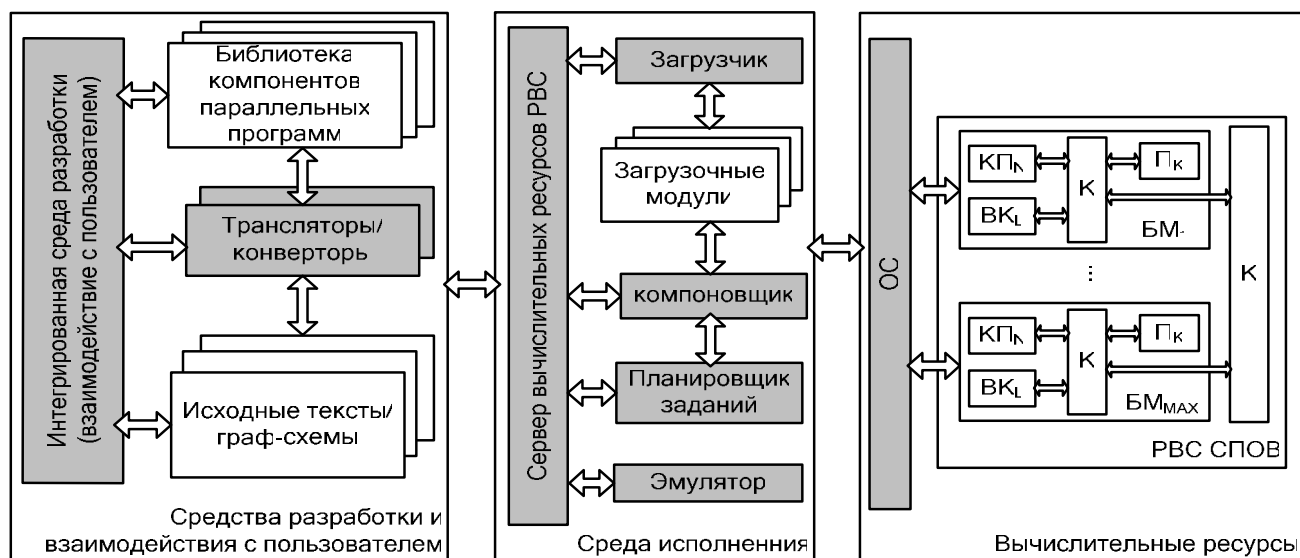


Рис. 6.3. Функциональное деление системного программного обеспечения

Вторая подсистема представляет собой сервер вычислительных ресурсов, в задачи которого входит обеспечение многоуровневого взаимодействия пользовательских процессов с оборудованием РМВС. На уровне прямого доступа к вычислительным ресурсам со стороны прикладных процессов обеспечивается загрузка конфигурации, загрузка исполнительных модулей в контроллеры распределяемой памяти, передача команд управления вычислительным процессом, ввод/вывод информации из каналов распределяемой памяти. Уровень прямого доступа позволяет получить

единственному прикладному процессу монопольные права на взаимодействие с оборудованием базовых модулей реконфигурируемой структуры.

На уровне клиентских запросов со стороны прикладных процессов средой исполнения выполняются функции планирования ресурсов и компоновки масштабируемых модулей в зависимости от конфигурации и текущей занятости ресурсов другими задачами. Данный уровень позволяет взаимодействовать с оборудованием системы сразу несколькими прикладными процессами. Для этого планировщик заданий упорядочивает обработку клиентских запросов, придавая среде исполнения функциональность многоканальной системы массового обслуживания, в которой единицей ресурса является один базовый модуль.

Структура математического обеспечения комплекса реконфигурирования архитектуры состоит из двух подсистем. В первой подсистеме обеспечивается доступ пользователей к среде исполнения и вычислительным ресурсам через комплекс системного и прикладного программного обеспечения. Создание параллельных программ осуществляется в интегрированной среде разработки. Эта среда включает в себя средства ввода и редактирования исходных текстов и граф-схем, трансляторы, отладчик параллельных программ, средства тестирования работоспособности оборудования РМВС и целый ряд других программ. Вторая подсистема представляет собой сервер вычислительных ресурсов, в задачи которого входит обеспечение многоуровневого взаимодействия пользовательских процессов с оборудованием РМВС.

На уровне прямого доступа к вычислительным ресурсам обеспечивается загрузка конфигурации ПЛИС и исполнительных модулей в контроллеры распределенной памяти, передача команд управления вычислительным процессом, ввод/вывод информации из каналов распределяемой памяти. Планировщик заданий упорядочивает обработку клиентских запросов, придавая среде исполнения функциональность многоканальной системы массового обслуживания, в которой единицей ресурса является один базовый модуль.

Для программирования РМВС могут создаваться статические, параметризуемые и масштабируемые параллельные программы. Статическая параллельная программа реализуется на неизменном в процессе исполнения множестве кадров. Для выполнения каждого кадра выделяется фиксированное количество базовых модулей и обеспечивается их взаимодействие. В процессе вычислений в каждом задействованном базовом модуле выполняются процедуры доступа к каналам распределяемой памяти и к каналам внешних связей, что обеспечивает поток данных через вычислительную структуру, состоящую из множества базовых модулей.

Параметризуемая параллельная программа отличается от статической программы тем, что процесс ее исполнения характеризуется множеством параметров, задающих количественные характеристики информационных потоков, что позволяет варьировать параметры реализации задачи, изменяя при этом объемы обрабатываемых данных. Масштабируемая параллельная программа отличается от параметризуемой программы тем, что параметризуется не только глубина обрабатываемых потоков данных, но и структура вычислительной системы.

Разработка масштабируемых структурно-процедурных программ включает в себя создание компонентов структуры параллельного алгоритма, а также процедур доступа к каналам распределяемой памяти и правил формирования в них исходных данных задачи и функций построения загрузочных модулей в зависимости от выделенных для вычислений ресурсов системы.

Разработку компонентов параллельных программ можно разделить на ряд этапов:

- 1) определяются структуры информационных потоков, общий параллельный алгоритм, правила размещения данных в каналах распределяемой памяти;
- 2) выполняется декомпозиция алгоритма на множество компонентов;

3) выполняется разработка и спецификация недостающих компонентов, которые помещаются в базу данных;

4) формируется кадровая форма выполнения задачи;

5) выполняется верификация созданной программы. Если созданная программа не может быть выполнена в силу аппаратных ограничений, то выполняется коррекция общего алгоритма, в связи с этим первоначальные шаги выполняются вновь;

6) выполняется отладка созданной программы.

Начальный этап разработки направлен на анализ алгоритмов решаемой задачи, определение необходимых структур данных. На последующих этапах строится основное решение задачи, исходя из возможных ограничений системы: вычислительных ресурсов системы, ресурсов памяти, пропускной способности каналов передачи данных, возможностей коммутационной системы. Структурная форма задачи переходит в структурно-процедурную, кадровую. Вопросы, связанные с методами преобразования в эффективную кадровую форму статических решений, рассматриваются в работах [62-65]. В предлагаемых методах происходит описание параллельных алгоритмов, на основе которых возможно выполнение параллельных программ различной конфигурации с многоуровневой организацией программирования архитектуры.

6.2. Язык ассемблера

6.2.1. Общие сведения

Язык программирования ARGUS v.2.0 представляет собой низкоуровневый язык, предназначенный для программирования вычислительной системы. Язык структурно-процедурного программирования ARGUS v.2.0 поддерживает реконфигурируемую аппаратную платформу структуры как за счет специальных языковых конструкций, так и за счет

включения параметров конфигурации в процессе трансляции и генерации исполняемого кода.

Программа на языке ARGUS v.2.0 содержит текстовый файл с инструкциями языка. Программы могут быть многомодульными. В минимальном случае программа может состоять из основного файла программы. В общем случае программа состоит из основного файла программы и нескольких подключенных к нему модулей, содержащих описание структурных компонентов программы. Такой подход позволяет сделать процесс написания программ более гибким и простым.

Основная программа содержит описания переменных и последовательности кадров, очередность выполнения которых задается управляющими операторами. Каждый кадр, в свою очередь, состоит из множества вычислительных команд, которые выполняются параллельно и асинхронно. Текст основной программы на языке ARGUS v.2.0 ограничен ключевыми словами PROGRAM и ENDPROG.

Подключаемый файл описывает вычислительную структуру кадра и содержит описание макрообращений, специальных прошивок и описание связей между ними. Порядок следования макрообращений и специальных прошивок не регламентирован. Текст подключаемого файла ограничен ключевыми словами UNIT и ENDUNIT.

Введение таких ограничительных конструкций позволяет структурировать программу, выделить библиотеку макрообращений и специальных прошивок в отдельные модули, которые в случае необходимости могут быть подключены к проекту.

Для определения переменных, описанных пользователем, и операторов языка ARGUS v.2.0 различают постоянные символические имена и символические имена, определяемые пользователем. Постоянные символические имена - это мнемонические обозначения операторов и директив языка. К символическим именам, определяемым пользователем, относятся:

- метки (локальные и глобальные);

- имена макрообращений;
- идентификаторы данных;
- имена базовых модулей.

Символические имена, определяемые пользователем, образуются из буквенно-цифровых символов в соответствии со следующими правилами:

- первый символ не должен быть цифрой;
- нельзя использовать разделительные знаки.

Метка - это символическое обозначение ячейки памяти программ контроллера распределенной памяти, в которую записывается машинный эквивалент оператора контроллера памяти, которому метка предшествует. Определение метки в формате оператора заканчивается символом '!'.

Предложение, написанное на языке ARGUS v.2.0, может располагаться на нескольких строках. Конец предложения определяется символом ';'.

Обычно предложение состоит из трех полей:

- поле метки;
- поле операции;
- поле операндов.

Существует еще поле комментариев, не имеющее строгой привязки, поэтому комментарий можно расположить в любой точке программы.

Поле комментария и поле метки не обязательны.

Поле операндов может отсутствовать.

Комментарий начинается знаком '/' и заканчивается знаком '*'.

Допускаются вложенные комментарии.

В ARGUS v.2.0 предполагаются следующие типы данных:

- *Number* – целое число;
- *Vector* – массив из n-элементов;
- *Stream* – массив из n-элементов;
- *Array* – многомерный массив;
- *ParamAddress* – адрес ячейки памяти.

Арифметические выражения в ARGUS v.2.0 состоят из символических имен, определяемых пользователем, чисел, знаков операций и встроенных функций.

6.2.2. Элементы языка

Формат директивы описания переменных имеет следующий вид:

var имя_переменной : тип_переменной;

Директива инициализации значения переменной описывается следующим образом:

Define имя_переменной = выражение;

Для определения значений элементов векторов существуют две основные формы директивы присваивания.

Форма 1:

define имя_переменной = [выражение];

Форма 2:

define имя_переменной = [выражение,..., выражение];

Если число элементов задано меньше размерности вектора, остальные компоненты будут заполнены:

- машинными нулями, если значения элементов вектора определяются впервые;

- останутся без изменения, если значения элементов вектора определяются не в первый раз.

Особенности векторной арифметики сводятся к нескольким правилам. Если один операнд - вектор, второй операнд – число и результат – вектор, то для каждого из элементов вектора-операнда выполняется арифметическая операция с этим числом.

Описание макрообращений

Макрообращения используются для настройки элементарных связей каналов контроллера распределенной памяти с другими компонентами

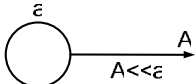
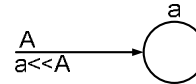
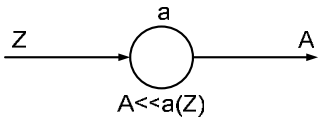
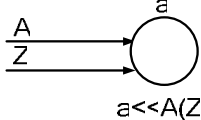
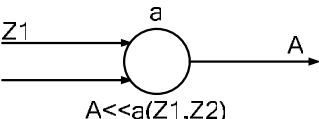
системы. В языке ARGUS v.2.0 существуют два вида макрообращений: реальное и виртуальное. Реальное макрообращение настраивает два канала связи.

Названия каналов должны начинаться со слова CHAN. Затем должен следовать номер канала. Для реального макрообращения номер должен находиться в диапазоне от 0 до 1.

Существует шесть типов команд макрообращения, которые представлены в табл. 6.1.

Все команды одного макрообращения должны иметь одинаковый тип. Завершать макрообращения должна директива ENDMACROMEMORY, которая сообщает ассемблеру об окончании последовательности команд макрообращения. Она требуется, так как команды настройки каналов могут стоять не в порядке возрастания номеров каналов контроллера распределенной памяти, которые они настраивают, а вразброс.

Таблица 6.1

Название типа команды	Формат команды	Графическое представление
Чтение данных	имя_выхода<<имя_канала;	
Запись данных	имя_канала<<имя_входа;	
Чтение данных по типу WHILE	имя_выхода<<имя_канала (имя_входа);	
Запись данных по типу WHILE	имя_канала<<имя_входа (имя_входа);	
Чтение по внешнему адресу (внешнее приращение).	имя_выхода<<имя_канала [имя_входа; имя_входа];	
Запись по внешнему адресу (внешнее приращение).	имя_канала<<имя_входа [имя_входа; имя_входа];	

Пример, демонстрирующий настройку каналов контроллера памяти на чтение данных, используя реальную макрооперацию:

```
RealMacromemory m00(out alpha0,beta0); Virtual;
alpha0<<chan0;
beta0<<chan1;
EndMacroMemory;
```

В этом макрообращении производится настройка двух каналов на чтение данных.

Пример, демонстрирующий использование виртуального макрообращения:

```
VirtualMacroMemory m00(In : alpha0,beta0; out : alpha0,beta0);
EndMacroMemory;
```

alpha0, beta0, alpha0, beta0 – логические имена используемых контроллеров распределенной памяти.

Команда чтения данных

Команда чтения данных предназначена для чтения потоков данных из распределенной памяти по вычисленным адресам. Начальный адрес определяется вектором начальных адресов оператора read или read_mod. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Приращение адресов задает вектор адресных приращений оператора чтения. Прочитанные данные поступают на выход А канала макропамяти.

Команда записи данных

Команда записи данных предназначена для записи потока данных в распределенную память по адресам. Начальные адреса определены вектором начальных адресов оператора write или write_mod. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Приращение адресов задает вектор адресных приращений оператора записи. Данные поступают на вход А канала макропамяти.

Команда чтения данных по типу while

Команда чтения данных по типу *while* предназначена для чтения потока данных из распределенной памяти, размер которого не может быть определен на этапе трансляции задачи, а определяется только на этапе выполнения. Начальные адреса определены вектором начальных адресов оператора *read_while*. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Приращение адресов задает вектор адресных приращений оператора чтения. Данные поступают на выход *A* канала макропамяти. Чтение происходит до тех пор, пока на вход *Z* канала не поступит логическая единица.

Команда записи данных по типу while

Команда записи данных по типу *while* предназначена для записи потоков данных в распределенную память. Начальный адрес определяется вектором начальных адресов оператора *write_while*. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Приращение адресов задает вектор адресных приращений оператора записи. Данные поступают на вход *A* канала макропамяти. Циклы записи происходят до тех пор, пока на вход *Z* канала не поступит логическая единица.

Команда чтения данных по внешнему адресу

Команда чтения данных по внешнему адресу предназначена для чтения потоков данных из распределенной памяти по адресу, две части которого поступают на входы *Z1* и *Z2* канала макропамяти. Для организации чтения данных по внешнему адресу следует использовать оператор *read*, у которого вместо имени вектора начальных адресов используется ключевое слово *extern*. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Данные поступают на выход *A* канала макропамяти.

Команда записи данных по внешнему адресу

Команда записи потоков данных по внешнему адресу предназначена для записи данных в распределенную память по адресу, две части которого поступают на входы *Z1* и *Z2* канала макропамяти. Для организации записи

данных по внешнему адресу следует использовать оператор *write*, у которого вместо имени вектора начальных адресов используется ключевое слово *extern*. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Данные поступают на вход А канала макропамяти.

Команда чтения данных с внешним приращением

Команда чтения данных с внешним приращением предназначена для чтения данных из распределенной памяти по адресу, который является вектором начальных адресов оператора *read*. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Для организации чтения данных с внешним приращением следует использовать оператор *read*, у которого вместо имени вектора адресных приращений используется ключевое слово *extern*. Значение приращения поступает на входы Z1 и Z2 канала макропамяти. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Данные поступают на выход А канала макропамяти.

Команда записи данных с внешним приращением

Команда записи потоков данных с внешним приращением предназначена для записи потоков данных в распределенную память, начальный адрес определяется вектором начальных адресов оператора *write*. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять.

Для организации записи данных с внешним приращением следует использовать оператор *write*, у которого вместо имени вектора адресных приращений используется ключевое слово *extern*. Значение приращения поступает на входы Z1 и Z2 канала макропамяти. Этот оператор должен относиться к той же странице распределенной памяти, в которую загружена макропамять. Данные поступают на вход А канала макропамяти.

Описание структурных компонентов

Описание структурного компонента кадра представляет собой следующую конструкцию:

```
Structure имя_кадра  
in список_формальных_параметров_входов;  
out список_формальных_параметров_выходов;  
совокупность_обращений_к_структурным_конструкциям.  
endstructure;
```

Директива STRUCTURE отмечает начало структурного компонента кадра, в теле которого могут находиться макрооперации, макрообращения или структурные компоненты кадра. Обращение к макрооперации осуществляется по следующему формату:

```
Имя_макрооперации(список_фактических_параметров);
```

В результате подстановки фактических параметров кадра в фактические параметры макрообращения определяются связи между компонентами системы, которые транслятор реализует в пространственно-коммутационной системе. Таким образом, программист может быть освобожден от программирования коммутатора.

Описание кадров

Кадр - это совокупность программных секций контроллера распределенной памяти, выполнение которых должно происходить одновременно. Обычно программа разбивается на множество кадров, которые последовательно вызываются во время работы программы, в минимальном варианте - на один кадр. Если программа должна выполнить переход из одного кадра в другой не через последний оператор, а посреди текущего кадра, то такой переход будет корректен, если все контроллеры памяти сделают аналогичный переход, минуя последний оператор кадра.

Директива CADR отмечает начало кадра. Все внутренние операторы контроллера памяти должны находиться только внутри кадра. Все внешние

операторы, за исключением оператора STOP, должны располагаться только вне кадра. Директивы присваивания, описание типов данных, директивы общего назначения могут находиться как внутри, так и вне кадра. Заканчивается описание кадра директивой ENDC.

6.2.3. Программирование распределенной памяти

Список конструкций программирования контроллера распределенной памяти включает в себя директивы и операторы. Директивы контроллера памяти - это группа конструкций языка ассемблера, которые формируют структуру программ для контроллера на этапе их ассемблирования.

Рассмотрим директивы контроллера распределенной памяти. При решении задачи часть модулей может не использоваться. Директивой MODULE задается тот ресурс, который требуется для решения текущего задания.

Формат директивы MODULE имеет следующий вид:

MODULE BM0[,BM1[,...[,BMi]]];

где $BM_i ::=$ номер базового модуля.

Число указывает абсолютный номер базового модуля.

Каждый базовый модуль содержит несколько ($8 \div 32$) модулей распределенной памяти или страниц. При обращении к странице контроллера распределенной памяти вначале указывается номер базового модуля, которому она принадлежит, а затем ее номер. При загрузке данных и программ в распределенную память необходимо указать конкретный номер страницы памяти, для которого эта информация предназначена. В операторе загрузки должен быть указан номер лишь одной страницы контроллера распределенной памяти. Отмена выбора текущей страницы контроллера распределенной памяти происходит автоматически при выборе новой страницы. Операторы контроллера распределенной памяти подразделяются на внутренние и внешние.

Внутренние операторы

Внутренние операторы контроллера распределенной памяти всегда должны быть расположены только в теле кадра. Эти операторы загружаются

непосредственно в память той страницы, в которой они используются. Операторы составляют три группы: операторы обращения к памяти данных, операторы пересчета параметров и операторы управления.

Операторы обращения могут быть операторами чтения и операторами записи.

В общем случае любой внутренний оператор контроллера распределенной памяти состоит из директивы, определяющей принадлежность оператора к определенному логическому номеру базового модуля и логическому номеру контроллера распределенной памяти и самого внутреннего оператора.

```
Cadr Cadr1
      DMC[№БМ , №КРП]#
      Read a Step 1 repeat 2;
EndCadr;
```

где DMC – директива, определяющая принадлежность оператора;

№БМ – номер базового модуля МВС;

№КРП – номер контроллера распределенной памяти;

READ – внутренний оператор контроллера распределенной памяти.

Операторы обращения могут быть операторами чтения и операторами записи. По типу адресации операторы делятся на операторы с инкрементной адресацией и операторы с пороговой адресацией. По типу данных различаются операторы чтения/записи чисел и операторы чтения/записи макроопераций и макрокоммутаций.

В табл. 6.2 приведены форматы операторов обращения.

Таблица 6.2

М-код	Формат	Действие
READ	Read BHA [step ВАП [repeat ПЦ]];	Чтение чисел с инкрементной адресацией
WRITE	Write BHA [step ВАП [repeat ПЦ]];	Запись чисел с инкрементной адресацией
CONTINUE	Continue step ВАП repeat ПЦ	Дополнительный оператор чтения(записи)
READ_MOD	Read_mod BHA [step ВАП [repeat ПЦ]];	Чтение чисел с пороговой адресацией

М-код	Формат	Действие
WRITE_MOD	Write_mod BHA [step ВАП [repeat ПЦ]];	Запись чисел с пороговой адресацией
READ_WHILE	Read_while BHA [step ВАП];	Чтение чисел по типу While
WRITE_WHILE	Write_mod BHA [step ВАП];	Запись чисел по типу While
READ	Read Extern [repeat ПЦ];	Чтение чисел с внешним адресом
WRITE	Write Extern [repeat ПЦ];	Запись чисел с внешним адресом
READ	Read BHA step Extern [repeat ПЦ];	Чтение чисел с внешним приращением
WRITE	Write BHA step Extern [repeat ПЦ];	Запись чисел с внешним приращением
READ_STRING	Read_Stirng BHA [size N];	Чтение строки
WRITE_STRING	Write_Stirng BHA [size N];	Запись строки
READ	Read BHA TO <i>ИМЯ_УСТРОЙСТВА</i>	Чтение МАОП (МАКОМ) для <i>ИМЯ_УСТРОЙСТВА</i> (МАР0-МАР3)
WRITE	Write BHA TO <i>ИМЯ_УСТРОЙСТВА</i>	Запись МАОП (МАКОМ) для <i>ИМЯ_УСТРОЙСТВА</i> (МАР0-МАР3)
MOVE_REG	Move_Reg [@]BHA, имя_регистра	Чтение регистра
MOVE_REG	Move_Reg имя_регистра; [@]BHA	Запись регистра

Формат оператора чтения с инкрементной адресацией имеет вид

read BHA [step ВАП [repeat ПЦ]], [чтение чисел].

Чтение потока чисел осуществляется одновременно по всем каналам. Количество циклов чтения определяется параметром цикла. Если читается один вектор, то параметры *step* и *repeat* могут отсутствовать. Начальные адреса чисел массива определяет параметр вектора начальных адресов (ВНА). На первой итерации исполнения оператора числа извлекаются из ячеек памяти каналов памяти данных, адреса которых заданы вектором начальных адресов. После выполнения оператора будет осуществлен переход к следующему оператору.

Оператор записи с инкрементной адресацией:

write BHA [step ВАП [repeat ПЦ]]; [запись чисел].

Запись потока чисел осуществляется одновременно по всем каналам. Количество циклов записи определяется параметром цикла (ПЦ). Если

читается один вектор, то параметры *step* и *repeat* могут отсутствовать. Начальные адреса чисел массива определяет параметр ВНА.

На первой итерации исполнения оператора числа помещаются в ячейки памяти каналов памяти данных, адреса которых заданы вектором ВНА. После выполнения оператора будет осуществлен переход к следующему оператору.

Оператор чтения с пороговой адресацией:

`read_mod ВНА [step ВАП [repeat ПЦ]]; [чтение чисел].`

Оператор выполняет чтение чисел с пороговой адресацией. Чтение чисел осуществляется одновременно по всем каналам. Количество циклов чтения/записи определяется параметром цикла. Если во время выполнения оператора чтения с пороговой адресацией текущий адрес данных достиг значения А0, то после этого значению нового адреса присваивается значение регистра BASE.

Оператор записи с пороговой адресацией:

`write_mod ВНА [step ВАП [repeat ПЦ]]; [запись чисел].`

Оператор выполняет чтение/запись чисел с пороговой адресацией. Запись чисел осуществляется одновременно по всем каналам. Количество циклов записи определяется параметром цикла. Если во время выполнения оператора записи с пороговой адресацией текущий адрес данных достиг значения А0, то после этого значению нового адреса присваивается значение регистра BASE.

Оператор CONTINUE:

`continue ВАП [repeat ПЦ];`

Оператор предназначен для организации вложенных циклических обращений к данным. Используется только после операторов чтения (записи) данных.

Оператор чтения по типу WHILE:

`read_while ВНА [step ВАП];`

Оператор выполняет чтение массива данных до тех пор, пока на управляющий вход не поступит логическая единица.

Оператор записи по типу WHILE:

write_while BHA [step ВАП];

Оператор выполняет запись массива данных до тех пор, пока на управляющий вход не поступит логическая единица.

Оператор чтения по внешнему адресу:

Read Extern [repeat ПЦ];

Оператор выполняет чтение данных по адресу, пришедшему по внешнему адресному входу.

Оператор записи по внешнему адресу:

Write Extern [repeat ПЦ];

Оператор выполняет запись данных по адресу, пришедшему по внешнему адресному входу.

Оператор чтения с внешним приращением:

Read BHA Step Extern [repeat ПЦ];

Оператор выполняет чтение данных по заданному адресу с приращением, пришедшим по внешнему адресному входу.

Оператор записи с внешним приращением:

Write BHA Step Extern [repeat ПЦ];

Оператор выполняет запись данных по заданному адресу с приращением, пришедшим по внешнему адресному входу.

Оператор чтения строк:

Read_String BHA [size N];

Оператор производит чтение данных, начиная с начального адреса, размера N. Эти данные представляют собой неразрывную конструкцию, начало и конец которой помечены начальным и конечным маркером (соответственно ВМ и ЕМ), в отличие от чтения чисел, где маркером отмечено каждое число.

Оператор записи строк:

Write_String BHA [size N],

Оператор производит запись данных, начиная с начального адреса, размера N. Эти данные представляют собой неразрывную конструкцию, начало и конец которой помечены начальным и конечным маркером соответственно,

ВМ и ЕМ, в отличие от записи чисел, где маркером отмечено каждое число. Элементы строки передаются плотным потоком, то есть разрыв между элементами строки недопустим. Строки обрабатываются с помощью специальных прошивок, реализованных на схемотехническом уровне.

Помимо рассмотренных операторов, используемых для программирования контроллера распределенной памяти, используются групповые операторы, которые предназначены для работы с массивами адресов.

При использовании групповых операторов программист освобождается от указания директивы принадлежности оператора к определенной странице памяти, для которой будет выполняться данный оператор.

В табл. 6.3 приведены форматы групповых операторов обращения.

Таблица 6.3

М-код	Формат	Действие
GROUPREAD	GroupRead MAC [<Параметры>];	Чтение массива чисел с инкрементной адресацией
GROUPWRITE	GroupWrite MAC [<Параметры>];	Запись массива чисел с инкрементной адресацией

Поле <Параметр> определяет диапазон работы оператора, в котором указывается начальный индекс в массиве адресов и конечный индекс. В качестве параметра MAC для групповых операторов может выступать переменная типа Vector, Stream или Array. Использование в качестве параметра MAC переменной другого типа является синтаксической ошибкой.

Групповой оператор чтения:

GroupRead MAC (StartIndex TO EndIndex);

Оператор выполняет чтение данных по адресам, указанным в поле MAC, начиная с позиции, указанной в поле StartIndex до элемента массива с индексом EndIndex.

Групповой оператор записи:

GroupWrite MAC (StartIndex TO EndIndex);

Оператор выполняет запись данных по адресам, указанным в поле MAC, начиная с позиции, указанной в поле StartIndex по элемент массива с индексом EndIndex.

Операторы чтения регистра

В язык ассемблера введены восемь регистров: регистр базы (BASE), регистр возврата (RETURN_REG), регистр состояния (STATUS), регистр операторов (OPER_SEG), регистр параметров (PARAM_SEG), регистр данных (DATA_SEG), пороговый регистр (MOD_REG) и сегмент макроинструкций (MACRO_SEG). Для получения доступа к содержимому этих регистров в язык ассемблер введены операторы чтения регистров.

Формат оператора чтения регистра:

move_reg [@]ВНА, имя_регистра;

Оператор производит чтение содержимого указанного регистра и записывает полученные данные по адресу, который содержится в векторе начальных адресов (ВНА). Для этого оператора можно использовать косвенную адресацию.

Формат оператора записи регистра:

move_reg имя_регистра,[@]ВНА;

Оператор производит запись данных, хранящихся по адресу, содержащемуся в ВНА, в соответствующий регистр. Для этого оператора можно использовать косвенную адресацию.

Операторы пересчета параметров

Операторы пересчета параметров представлены в табл. 6.4.

Таблица 6.4

Формат	Действие
Сложение add vector1, vector 2;	Пересчет вектора (vector1). Пересчет вектора адресов (приращений адресов).

Формат	Действие
Пересылка Move vector1, vector2; Move vector1,@ vector2;	Пересылка вектора Пересылка вектора по косвенному адресу
Сложение полей оператора AddoSH метка, ВАП; AddoS метка, ВАП; AddoJ метка, ВАП; AddoM метка, ВАП;	Модернизация поля SH Модернизация поля S Модернизация поля J Модернизация поля M
Замещение полей оператора MoveSH метка, ВАП; MoveS метка, ВАП; MoveJ метка, ВАП; MoveM метка, ВАП;	Замещение поля SH Замещение поля S Замещение поля J Замещение поля M
Очистка полей оператора ClrSH метка; ClrS метка; ClrJ метка; ClrM метка;	Очистка поля SH Очистка поля S Очистка поля J Очистка поля M

Оператор сложения:

Add vector1, vector 2.

Оператор выполняет сложение vector1 с vector2 и результат помещает в vector1.

Оператор пересылки

MOVE *приемник, источник*;

Действие оператора состоит в пересылке содержимого операнда источника в операнд приемник. В качестве источника может выступать либо непосредственно вектор данных, либо вектор, содержащий адреса памяти данных, по которым хранятся пересылаемые данные. В первом случае выполнение оператора заключается в том, что содержимое источника

переписывается в приемник. Во втором случае содержимое памяти данных, адреса ячеек которой задает источник, переписывается в приемник.

Оператор сложения поля смещения:

addoSH метка, ВАП;

Оператор выполняет сложение соответствующего поля смещения операнда с нулевым элементом операнда ВАП.

Оператор сложения поля цикла:

addoS метка, ВАП;

Оператор выполняет сложение соответствующего поля цикла операнда с нулевым элементом операнда ВАП.

В данном случае также достаточно задать лишь нулевой элемент вектора, так как остальные элементы ВАП данным оператором использоваться не будут.

Оператор сложения поля метки:

addoJ метка, ВАП;

Оператор выполняет сложение соответствующего поля метки операнда с нулевым элементом операнда ВАП.

Оператор сложения поля модификации:

addoM метка, ВАП;

Оператор выполняет сложение соответствующего поля модификации операнда с нулевым элементом операнда ВАП.

Оператор замещения поля смещения:

moveSH метка, [@]ВАП;

Оператор выполняет замещение поля смещения операнда нулевым элементом операнда ВАП.

Оператор замещения поля цикла:

moveS метка, [@]ВАП;

Оператор выполняет замещение поля цикла операнда нулевым элементом операнда ВАП.

В данном случае также достаточно задать лишь нулевой элемент вектора, так как остальные элементы операнда данным оператором использоваться не будут.

Оператор замещения поля метки:

moveJ метка, [@]ВАП;

Оператор выполняет замещение поля метки операнда нулевым элементом операнда ВАП.

Оператор замещения поля модификации:

moveM метка, [@]ВАП;

Оператор выполняет замещение поля модификации операнда нулевым элементом операнда ВАП.

Оператор очистки поля смещения:

ClrSH метка;

Оператор выполняет очистку поля смещения оператора.

6.2.4. Операторы управления

Группа операторов управления включает следующие операторы:

- оператор безусловного перехода;
- оператор паузы;
- оператор цикла;
- условный оператор.

Операторы управления приведены в табл. 6.5.

Таблица 6.5

Название	Формат	Действие
Оператор безусловного перехода:	Goto J;	Безусловный переход по адресу J
Оператор паузы:	Pause N;	Задержка выполнения программы на N тактов
Оператор цикла:	Loop S [goto J];	Оператор цикла
Условный оператор:	If A goto J; If @A goto J;	Переход по значению ячейки A Переход по значению ячейки по адресу A

Оператор безусловного перехода:

Goto J;

При выполнении оператора безусловного перехода выполняется переход к оператору, помеченному меткой J. Частными случаями оператора goto являются операторы call и return.

Формат оператора call:

call J;

Данный оператор выполняет переход по метке J с сохранением значения адреса в стек. Данный оператор используется для организации подпрограмм.

Формат оператора return:

return;

Данный оператор извлекает адрес из регистра и выполняет переход по полученному адресу. Данный оператор используется для возврата из подпрограмм.

Оператор паузы:

Pause N;

Действие этого оператора заключается в том, что выполняется задержка исполнения программы на N тактов.

Оператор цикла:

Loop S [goto J];

Условный оператор:

If A goto J;

Действие условного оператора заключается в том, что происходит проверка содержимого ячейки памяти A. Если она содержит ненулевое значение, то происходит переход по метке J, иначе выполняется следующий оператор.

Второй вариант условного оператора следующий:

If @A goto J;

Знак @ означает, что используется косвенная адресация.

Формат структуры описания адресной коммутации:

```
acommutation <Имя_Коммутации>;  
chan0 (<Номер Бита_0>, ...<Номер Бита_15>);  
chan1 (<Номер Бита_0>, ...<Номер Бита_15>);  
endacom;
```

Для использования коммутации в исполняемом кадре предусмотрен оператор загрузки коммутации LOADACOM и оператор управления коммутацией LOADBCOM. Формат оператора LOADACOM следующий:

LOADACOM <Имя_Коммутации>.

Коммутация должна быть предварительно описана. Загруженная для данного контроллера распределенной памяти коммутация используется до загрузки новой коммутации или до подачи команды сброса коммутации.

Формат оператора LOADBCOM следующий:

LOADBCOM <Имя_Переменной>.

Оператор LOADBCOM выполняет переключение между внешними шинами, подключенными к контроллеру распределенной памяти.

Для сброса адресной коммутации используется оператор сброса, имеющий следующий формат:

RESETACOM.

В результате восстанавливается естественный порядок бит адреса.

Операторы загрузки и сброса могут также использоваться в групповых операциях для группы контроллеров распределенной памяти.

6.2.5. Внешние операторы контроллера распределенной памяти

В табл. 6.6 приведены форматы внешних операторов контроллера памяти. Отличительная особенность этих операторов в том, что они не могут помечаться метками.

Ввод чисел непосредственно из исходного текста программы выполняется оператором, имеющим следующий формат:

input ВНА [step ВАП] [channels ВВК][repeat КЦ] data {числовой_ряд};

где числовой_ряд ::= набор чисел, разделенных запятыми или набор выражений.

Таблица 6.6

Формат	Действие
Операторы: ввод input ВНА [step ВАП] [channels ВВК] [repeat КЦ] data {числовой_ряд};	Ввод данных непосредственный
input ВНА [step ВАП] [channels ВВК] [repeat КЦ] file "имя_файла";	Ввод данных из внешнего файла
- вывод output ВНА [step ВАП] [channels ВВК] [repeat КЦ] file "имя_файла";	Вывод данных в файл (данные не преобразованы)

Константа цикла здесь задает количество вводимых векторов чисел. Когда требуется вводить числа из отдельного файла, оператор записывается в следующем виде:

input ВНА [step ВАП] [channels ВВК] [repeat КЦ] file "имя_файла";

где имя_файла ::= строка_символов.

Оператор OUT осуществляет вывод в файл содержимого памяти данных.

Формат оператора имеет вид:

output ВНА [step ВАП] [channels ВВК] [repeat КЦ] file "имя_файла";

В языке ARGUS v.2.0 различают два вида операторов ввода/вывода: внешние операторы и внутренние операторы.

Операторы, располагающиеся вне кадра и управляющие работой контроллера распределенной памяти извне, называются *внешними*. Внешнее управление выполняет программа "Загрузчик векторов прерываний" совместно с контроллером обмена.

Внешние операторы делятся на операторы ввода-вывода и операторы управления заданием. Операторы ввода-вывода выполняют загрузку/выгрузку данных в/из памяти данных в контроллер распределенной памяти. Данными могут быть как числа, так и адреса. Загрузка может происходить как во все каналы, так и в определенные. Это ограничение накладывается вектором выбираемых каналов. Числа могут вводиться в двух вариантах: либо непосредственно из исходной программы, либо из внешнего файла. Файл, из

которого будут считываться данные при вводе, должен быть двоичным. Когда данные задаются непосредственно в тексте программы, то они могут быть представлены в любой системе счисления, поддерживаемой языком ARGUS v.2.0. Операторы ввода-вывода могут быть расположены всегда только вне кадра, в так называемом "межкадровом пространстве". Если оператор ввода/вывода стоит до первой директивы CADR, то оператор называется "Оператором начальной загрузки(выгрузки)". Когда оператор ввода/вывода размещен между кадрами, то есть его выполнение должно произойти по окончании некоторого кадра, то оператор называется "Оператором загрузки (выгрузки)". При обнаружении такого оператора ARGUS V.2.0 устанавливает последнему оператору предыдущего кадра признак перехода с ОСТАНОВОМ. Если программа запущена, то после исполнения оператора с таким типом перехода в текущем кадре выполнится ОСТАНОВ во всех контроллерах памяти, и обработчик векторов прерываний выполнит ввод/вывод в/из контроллера, номер которого был задан в программе. По окончании операций ввода/вывода загрузчиком будет выработан общий ПУСК для продолжения работы платы базового модуля. Номер контроллера задается аналогично тому, как и при программировании кадров.

Ввод/вывод может выполняться как во все каналы контроллера памяти, так и по выборочным каналам. Внешние параметры в операторах ввода не допускаются, когда данные вводятся непосредственно. Если же ввод или вывод осуществляются из внешнего файла, то параметры операторов ввода/вывода могут быть и внешними.

Так как константа цикла в операторах ввода/вывода не может быть подвергнута модификации, то в этом поле допускается использовать как идентификаторы, так и числа, и выражения.

Операторы, располагающиеся внутри кадра и управляющие работой контроллера распределенной памяти изнутри, называются *внутренними*. К внутренним управляющим операторам относят оператор чтения данных из канала памяти и оператор записи данных в канал памяти.

6.3. Язык параллельного программирования высокого уровня

COLAMO

Язык высокого уровня COLAMO используется для структурно-процедурной организации вычислений в РМВС. Одной из проблем, с которой сталкивается программист при написании параллельной программы, является семантический разрыв между параллельным алгоритмом и его реализацией на вычислительной системе. Структурно-процедурная организация вычислений позволяет решать эту проблему.

Для эффективной реализации на многопроцессорных системах вычислительных алгоритмов необходим язык программирования высокого уровня (ЯПВУ). Язык должен эффективно реализовываться на различных вычислительных системах, иметь средства для записи алгоритмов без изменения их начальной параллельной структуры [86,123,141]. Язык должен позволять программисту максимально просто описывать различные виды параллелизма в достаточно сжатом виде и содержать мощные конструкции, имеющиеся в традиционных ЯПВУ [122,123].

В известных работах [30,67,120,121,139] рассматриваются новые языки, ориентированные на конкретные вычислительные системы, что, естественно, ограничивает число пользователей, или традиционные процедурные ЯПВУ с расширениями, предназначенные для программирования параллельных вычислений. Изучение этих подходов показало, что существующие методы распараллеливания являются чрезмерно трудоемкими или не позволяют адекватно отобразить алгоритм на данную вычислительную систему, поэтому целесообразна разработка языков, имеющих средства для явного задания независимых частей программы [28].

Сложившаяся методика распараллеливания алгоритмов предусматривает представление задачи в ярусно-параллельной форме (ЯПФ) [94]. Языком программирования такого вида является ОССАМ, получивший широкое распространение в транспьютерных системах и позволяющий описать любую

ЯПФ с помощью инструкций `par`, `seq`, отмечающих участки параллельных и последовательных процессов [121].

Однако программы, описывающие ЯПФ задачи, имеют недостатки, главные из которых - низкая читабельность текста и сложность отладки программ, так как в каждый момент времени выполняется несколько процессов, то есть фактически функционирует несколько программ, которые необходимо связывать друг с другом, что особенно сложно при асинхронной работе элементарных процессоров вычислительной системы. Обычно проблемы синхронизации между процессами разрешаются конструкциями `fork-join` [21], семафорами Дейкстры [29], принципами рандеву [139], определениями приоритетных процессов [121]. Каждый из этих способов имеет свои недостатки, рассмотренные в работе [6].

Получить на практике для любой задачи ее прямоугольную ЯПФ [94] не представляется возможным, что является одной из причин снижения эффективности работы системы. Известные многопроцессорные вычислительные системы с неизменяемой организацией каналов связи между компонентами системы в большинстве случаев не адекватны решаемой задаче. Это обстоятельство существенно затрудняет распараллеливание задачи и значительно снижает эффективность системы. Возможность программирования каналов связи между компонентами многообъектной РМВС позволяет программировать систему так, чтобы она была адекватна решаемой задаче и обеспечивала высокое распараллеливание процесса обработки информации.

Для реконфигурируемой мультиконвейерной вычислительной системы со структурно-процедурной организацией вычислений возможен качественно новый подход. Задача делится на независимые, функционально законченные участки, реализуемые аппаратно. Смена вычислительных структур происходит под управлением программы. Таким образом, в ходе выполнения программы может выполняться только одна вычислительная конструкция.

Значительную сложность в языках программирования высокого уровня для многопроцессорных систем представляет проблема, связанная с

программированием связей между компонентами системы [116]. Например, в языке OCCAM данная задача решается с помощью операторов описания каналов между элементарными процессорами и элементами распределенной памяти [31]. В системе SWARD средствами коммутации являются две машинные команды *send* и *receive*, а также объекты «порт» [122]. Такие подходы, по сути дела, снижают ЯПВУ до уровня машинных языков.

В рассматриваемом языке COLAMO отсутствуют явные формы описания параллелизма. Распараллеливание достигается с помощью объявления типов переменных и индексации элементов массивов. Естественным способом реализации неявного описания параллелизма в программе является правило единственной подстановки [115], которое широко используется в языках потока данных и заключается в следующем: переменная может получить значение в программе только один раз [61]. Данное правило приводит к противоречию с традиционными принципами программирования. В связи с этим делается следующее ограничение: правило единственной подстановки действует в специальных конструкциях предлагаемого языка, описывающих вычислительные структуры, задаваемые пользователями.

Операторы языка программирования COLAMO делятся на три группы:

- операторы описания вычислительных структур;
- операторы управления вычислительным процессом;
- операторы описания данных и программно-аппаратных ресурсов.

Фундаментальным типом вычислительной структуры в данном языке является конструкция "*кадр*". *Кадр* языка соответствует совокупности арифметико-логических команд, выполняемых в различных элементарных процессорах и контроллерах распределенной памяти, соединенных между собой в соответствии с информационной структурой алгоритма таким образом, что вычисления производятся с максимально возможными параллелизмом и асинхронностью [61].

В общем случае кадр задается следующим образом:

CADR name;

P

END name;

где CADR, END - ключевые слова языка;

name - имя кадра;

P - список операторов кадра.

Операторы присваивания, имеющие вид: Name=f, где Name - имя переменной, которой присваивается значение арифметико-логического выражения f, могут быть только внутренними. Операторы безусловного перехода goto lab, где lab - метка оператора, к которой производится переход, могут быть только внешними. Вычислительные структуры рассматриваются как некоторая крупная, программно-неделимая единица, и передавать управление внутри структуры между ее компонентами нельзя.

В теле кадра могут находиться несколько операторов присваивания. Программист не выделяет параллельные участки вычислений. В результате преобразования программы транслятором параллелизм, содержащийся в описании вычислительной системы, извлекается из нее естественным образом на основании взаимосвязей между данными.

В рассматриваемом языке переменные разделяются по способу хранения на мемориальные (MEMory), коммутационные (COMmutation) и регистровые (REGister). *Мемориальной переменной* называется величина, хранящаяся в ячейке памяти и, следовательно, сохраняющая свое значение до очередного присваивания (рис.6.4).

```
Var a,b,c,d,e,f,g : Array [10 : Stream] Mem;  
Cadr  
  For I := 1 to 5 do  
    Begin  
      A[I] := B[I]·C[I] + F[I];  
      D[I] := B[I]·C[I] + E[I]·G[I] + G[I-1];  
    End;  
  EndCadr;
```

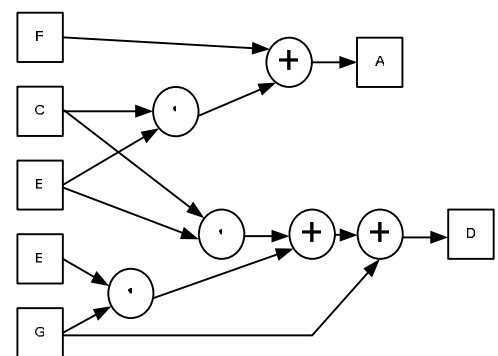


Рис. 6.4. Использование мемориальной переменной

Единственным ограничением, накладываемым на использование мемориальных переменных, является правило единственной подстановки. В данном случае запрещается использование в левых и правых частях вычислительных выражений переменных, расположенных в одинаковых каналах памяти.

Коммутационной переменной называется величина, служащая для описания каналов между элементами системы; значения переменной данного типа недоступны пользователю (рис.6.5).

```

Var a,b,c,d,e,f,g : Array [10 : Stream] Mem;
Var Z : Integer Com;
Cadr
  For I := 1 to 5 do
    Begin
      Z := B[I]·C[I];
      A[I] := Z+F[I];
      D[I] := Z + (E[I]·G[I]) + G[I-1];
    End;
  EndCadr;

```

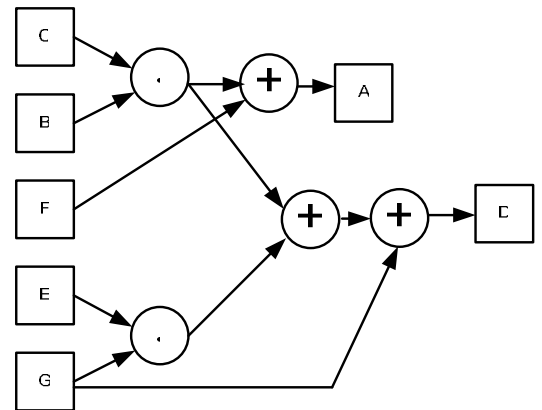


Рис. 6.5. Использование коммутационной переменной

Регистровой переменной называется величина, служащая для хранения информации, значения которой доступны пользователю в любой момент времени (рис.6.6).

```

Var a,b,c,d,e,f,g : Array [10 : Stream] Mem;
Var Z : Integer Reg;
Cadr
  For I := 1 to 5 do
    Begin
      Z := B[I]·C[I];
      A[I] := Z+F[I];
      D[I] := Z + (E[I]·G[I]) + G[I-1];
    End;
  EndCadr;

```

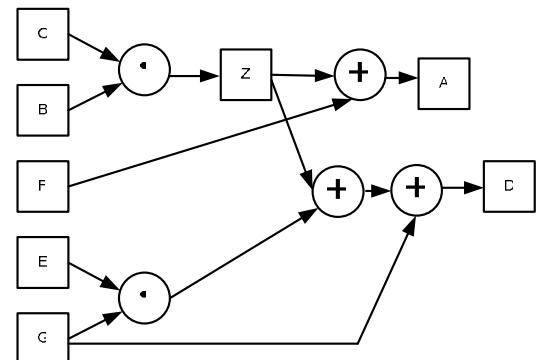


Рис. 6.6. Использование регистровой переменной

Использование регистровых переменных может привести к неоднозначности выполнения программы. Это связано с тем, что все компоненты кадра выполняются асинхронно и независимо, и невозможно однозначно сказать, что выполнится первым – чтение или запись в регистровую переменную, следовательно, неизвестно, корректны будут вычислительные операции в данном случае или нет.

Выйти из данной тупиковой ситуации возможно с помощью разрезки некорректного кадра на несколько кадров. В этом случае порядок выполнения кадров жестко регламентирует значения получаемых переменных:

```
CADR S1;
For I := 1 to 5 do
  Begin
     $M[I] := B[I] \cdot C[I];$ 
  End;
ENDCADR;
CADR S2;
For I := 1 to 5 do
  Begin
     $A[I] := M[I] + F[I];$ 
     $D[I] := M[I] + (E[I] \cdot G[I]) + G[I-1];$ 
  End;
ENDCADR.
```

Подобная конструкция обеспечивает детерминизм получения результатов, но сокращает число арифметико-логических операций в кадре и, как следствие, количество элементарных процессоров (ЭП), реализующих кадр. Это приводит к уменьшению реальной производительности системы при решении задачи и снижает эффективность РМВС.

Введение коммутационной или регистровой переменной позволяет упростить графы вычислительных структур, описанных пользователем на предлагаемом языке.

Следует отметить, что применение в одном кадре вычислительной конструкции, в которой требуется прочитать и записать переменную,

некорректно. Этот принцип легко реализовать для скаляров, но когда используются элементы массива, могут быть определенные трудности.

В самом деле, запись

$$X[i]=f(X[j])$$

на этапе трансляции не может быть признана ни ошибочной, ни корректной, так как невозможно заранее сказать, не встретится ли при выполнении кадра такая ситуация, при которой i будет равно j , которая приведет к нарушению принципа однократного присваивания.

Поскольку одной из задач, поставленной при создании рассматриваемого языка программирования высокого уровня, является взаимно однозначное соответствие вычислительных структур и записей на языке, то мы приходим к парадоксальному выводу, что некоторые недетерминированные конструкции могут быть разрешены к использованию. Эти конструкции отмечаются транслятором предупреждением. Выйти из тупиковой ситуации возможно с помощью разрезания некорректного кадра на несколько кадров, что обеспечивает детерминизм результатов, но приводит, как отмечалось выше, к снижению реальной производительности системы при решении задачи. Введение коммутационных переменных решает рассматриваемую задачу без нежелательных побочных эффектов. Объявление типов переменных производится с помощью оператора DCL, который в общем случае имеет следующий вид:

VAR p1 a1, p2 a2,..., pn an;

где p_i - список переменной i -й группы, для которых установлены соответствующим списком характеристики,

VAR - ключевое слово языка.

Элементы списка переменных разделяются запятыми. Элементы списка характеристик переменных разделяются пробелами. К характеристикам переменных относятся: способ хранения в памяти (мемориальная, регистровая, коммутационная), а также тип переменной (вещественная (REAL), комплексная

(COMPL), символьная (CHAR), адресная (ADDR), логическая (LOGIC), управляющая (CONTR), целая (INTEGER).

Кроме скалярных переменных, в языке допускается использование массивов. В однопроцессорных ЭВМ многомерные массивы представляются одномерными. В PMBC со структурно-процедурной организацией вычислений все массивы представляются двумерными, где первая координата соответствует номерам каналов элементов распределенной памяти, вторая - ячейкам памяти в канале.

В разработанном языке программирования используются векторы (VECTOR) и потоки (STREAM). Аналогичные предложения были реализованы в языке программирования IVTRAN [76]. По умолчанию массивы считаются векторами. Потоками являются массивы переменных, элементы которых могут быть обработаны только последовательно. Векторами являются массивы переменных, элементы которых могут быть обработаны параллельно.

На практике обычно используется параллельно-последовательная обработка массивов информации. Степень параллелизма зависит от конкретного алгоритма и аппаратного ресурса, которым обладает пользователь. Под степенью параллелизма понимается количество элементарных операций, выполняемых одновременно.

Оператор цикла языка имеет следующий вид:

FOR[j=i1] [TO i2] [STEP i3] [WHILE i4] BEGIN p END;

где FOR, TO, STEP, WHILE - ключевые слова языка;

j - имя переменной цикла;

i1 - список начальных значений переменной цикла;

i2 - выражение, определяющее предел приращений переменной j;

i3 - приращение переменной цикла;

i4 - логическое выражение, при невыполнении которого производится выход из цикла;

BEGIN, END - синтаксические скобки, ограничивающие тело цикла;

P - список операторов цикла.

Все записи, находящиеся во внутреннем цикле, мультиплицируются в соответствии с параметрами цикла на вычислительную структуру для векторов или осуществляется периодическое поступление элементов массивов по заданной оператором цикла процедуре для потоков. Таким образом, семантика синтаксически одинаковых конструкций кадра определяется структурой данных, однозначно диктующих правило доступа к элементам массивов.

Внешний по отношению к описанию вычислительной структуры оператор цикла осуществляет циклическое выполнение кадров операторов, в том числе и находящихся в теле цикла. Внешний оператор цикла эквивалентен оператору цикла стандартных языков программирования.

На рис. 6.7 приведены программа, реализующая параллельно-последовательное суммирование массивов с помощью внутренних и внешних операторов цикла, и граф вычислительной системы, в который она будет оттранслирована.

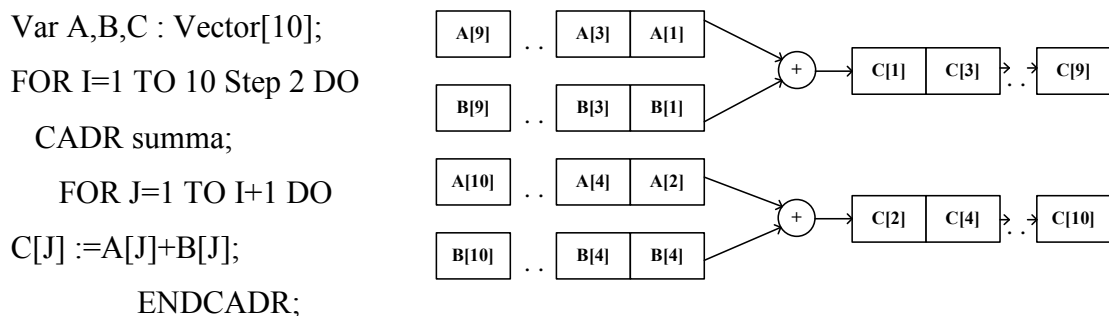


Рис. 6.7. Программа и эквивалентный граф вычислительной структуры

На практике часто встречается необходимость обращения и выборки элементов массива. В общем случае выборка имеет следующий вид:

$$A[l:k:s],$$

где A - имя массива;

l - значение индекса, определяющее начало выборки;

k - значение индекса, определяющее конец выборки;

s - значение шага индекса в выборке.

Все параметры являются позиционными. При пропуске любого параметра, кроме последнего, необходимо ставить определенное количество

двоеточий. По умолчанию начальный индекс выборки равен начальному значению в описании массива, k - конечному значению в описании массива. Например, запись $A[i, 4:8]$ означает, что производится выборка по второй координате массива, начиная с 4-го по 8-й с шагом 1.

В общем случае массив описывается следующим образом:

$$\text{VAR } A [p1, p2, \dots, pn] S,$$

где A - имя массива;

p_i - характеристики i -й размерности массива;

S - тип переменной массива.

Структурное отображение выборки определяется типом обработки компонента массива, по которому производится выборка. Для векторной составляющей массива выборка соответствует определенным номерам каналов распределенной памяти, которые одновременно выставляют (считывают) данные на вычислительную структуру. Для среза потоковой составляющей данные считываются (записываются) во все каналы распределенной памяти, которые соответствуют массивам. Для потоковой составляющей выборка определяет цикл обращения к ячейкам памяти в каналах распределенной памяти, в случае среза - последовательное обращение ко всем элементам распределенной памяти.

Массивы переменных могут быть мемориальные, коммутационные и регистровые. Коммутационные вектора представляют собой группу информационных каналов, которым присвоено одно имя. Коммутационный поток позволяет организовать запаздывание операндов.

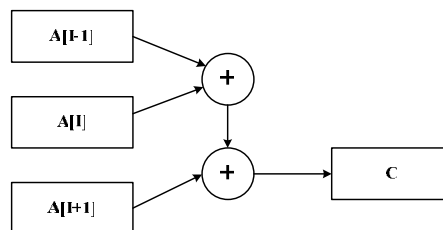
На рис. 6.8 приведены программы на языке программирования высокого уровня и графы вычислительных структур, в которые они будут оттранслированы.

Если в одном кадре для вычисления переменной происходит обращение к нескольким элементам массива, то для вектора это означает определенным образом заданную коммутацию каналов распределенной памяти, а для потоков - запаздывание между элементами массива.

```

Var A : Array[10 : Vector];
Var C : Integer Reg;
CADR a; FOR I=2 TO 9 DO
  C:=A[I]+A[I-1]+A[I+1];
ENDCADR;

```

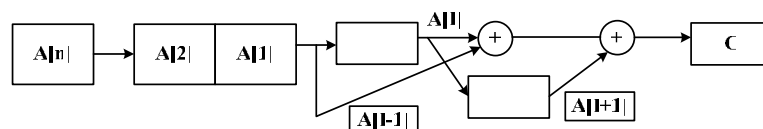


а)

```

VAR A : Array[10 : Stream];
Var C : Integer Reg;
CADR b;
FOR I=2 TO 9 DO
  C:=A[I]+A[I-1]+A[I+1];
ENDCADR;

```



б)

Рис. 6.8. Обращения к элементам вектора и к элементам потока

а) обращение к элементам вектора; б) обращение к элементам потока

Следует отметить, что если в программе встречается, например, следующая ситуация:

```

Var X : Array[100 : Stream];
CADR ST;
FOR I := 1 TO 50 DO
  C := X[i] + X[j];
ENDCADR;

```

то транслятор обязан определить эту ситуацию как ошибочную, поскольку невозможно определить размер буфера для потока X. Разница между индексами элементов потока, к которым происходит обращение в теле кадра, должна быть если не константой, то, по крайней мере, не изменяться во время выполнения кадра.

Пусть требуется вычислить $A_i = (B_i + B_{i-1}) * C$. Программа, реализующая данные вычисления, может выглядеть следующим образом при использовании неявной буферизации:

```

Var A,B,C : Array [4 : Stream];
CADR A;
FOR i=2 TO 4 DO;

```

```

    A[i] := (B[i]+ B[i-1])*C[i];
  END;
ENDCADR;

```

Ниже приведена программа, реализующая тот же пример, но уже с явно заданной буферизацией:

```

  Var A,B,C : Array [4 : Stream];
  Var R REG;
  CADR BO
    R := C[1];
  ENDCADR;
  CADR B;
  FOR i:=2 TO 4 DO
    BEGIN
      A[i] := (B[i]+R)*C[i];
      R := B[i];
    END;
  ENDCADR;

```

(6.1)

На первый взгляд мы имеем дело с нарушением корректности данных: чтение и запись одной переменной происходят в теле кадра, однако идентификатор R описывает регистровую переменную, которая используется специально для организации запаздываний. Регистровая переменная выставит свое значение тогда и только тогда, когда в нее происходит запись, или в момент запуска кадра.

Таким образом, в фрагменте программы (6.1) оператор присваивания $R:=B[i]$ выполняется одновременно с оператором присваивания $A:=(B[i]+R)*C[i]$, при этом переменной R соответствуют значения $B[i-1]$.

Аналогично можно в явном виде записать буферизацию для примера, представленного на рис. 6.8б.

```

  Var A : Array [10 : Stream] Mem;
  Var R1, R2 : Integer REG;
  CADR I;

```

```

R1=A[1];
R2=A[2];
ENDCADR;
CADR C;
FOR i=3 TO 10 DO
BEGIN
  C=A[i]+R1+R2;
  R1=A[i];
  R2=R1;
END;
ENDCADR;

```

Поскольку регистровой переменной соответствует задержка потока на один отсчет, то R1 соответствует A[i-1], а R2 есть задержка отсчета R1 и соответствует A[i-2]. Данный пример демонстрирует частный случай использования регистровых переменных.

Использование правила единственной подстановки в теле кадра приводит к трудностям при организации рекурсивных вычислений. Поскольку рекурсивные вычисления в теле кадра допускаются только для регистровых переменных, это требует определенных навыков при программировании на данном языке. Например, запись

```

VAR A,B : Integer MEM;
CADR a
A:=B;
B:=A;
ENDCADR;

```

является семантически неправильной, поскольку не соответствует принципам программирования потоков данных, так как переменные A и B должны быть одновременно прочитаны и записаны, в то время как для мемориальных переменных в теле кадра допускается только одно обращение к памяти. Кажущаяся последовательность записей тут не выполняется, так как все операторы в теле кадра выполняются параллельно. Запись вида

VAR A,B : Integer REG;

CADR b

A:=B;

B:=A;

ENDCADR;

семантически правильная. По завершении данной программы переменные A и B поменяют свои значения. Непривычность подобного действия объясняется тем, что все операторы в теле кадра выполняются параллельно.

Для реализации ветвления в программах используется условный оператор IF. Оператор IF может быть внутренним или внешним по отношению к кадру. Условный оператор представляет собой конструкцию следующего вида:

IF q THEN a [ELSE b];

где IF, THEN, ELSE - ключевые слова языка;

q - логическое выражение;

a, b - списки операторов, ограниченные синтаксическими скобками BEGIN, END, выполняемые при истинности или ложности q соответственно.

Если списки a или b состоят из одного оператора, синтаксические скобки BEGIN, END могут отсутствовать. Допускаются вложенные конструкции условных операторов:

IF q_1 THEN IF q_2 THEN IF q_3 ELSE d ;

Семантика внешнего условного оператора совпадает с семантикой условных операторов процедурных ЯПВУ.

Внутренний условный оператор выполняется следующим образом: группа операторов a и b , представляющих подграфы графа кадра, выполняются параллельно. Результаты вычислений обеих групп поступают на переключатель, работа которого определяется логическим выражением q . Другими словами, внутренний условный оператор реализует структурный аналог условного перехода. Использование условного оператора в теле кадра позволяет естественным образом реализовать параллелизм, содержащийся в алгоритме, и максимально быстро получить результат.

На рис. 6.9 приведен пример использования внутреннего условного оператора.

```
Var A,B: Array [100 : Stream];
```

```
CADR b;
```

```
FOR I:=1 TO 100 DO
```

```
IF A[I]>0 THEN
```

```
  B[I]:=sin(A[I]);
```

```
ELSE
```

```
  B[I]:=exp(A[I]**2);
```

```
ENDCADR;
```

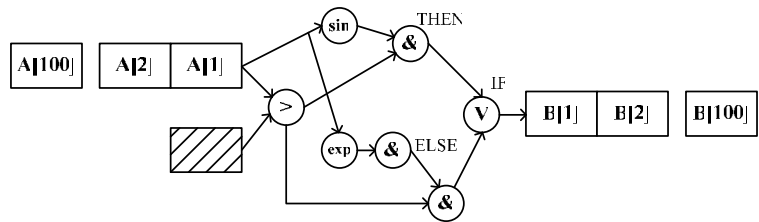


Рис. 6.9. Применение внутреннего условного оператора

Рассматриваемые вычисления могут быть реализованы другим способом.

```
Var A,B : Array [100 : Stream] Mem;
```

```
FOR I=1 TO 100 DO;
```

```
IF A[I]>0 THEN
```

```
  CADR b1;
```

```
    B[I] := sin(A[I]);
```

```
  ENDCADR;
```

```
ELSE
```

```
  CADR b2;
```

```
    B[I] := exp(A[I]**2);
```

```
  ENDCADR;
```

```
END;
```

(6.2)

Здесь условный оператор расположен вне кадра, задача реализуется на двух ВС, причем в процессе выполнения программы производится инициализация одной из них в зависимости от значения логического выражения $A[I]>0$. Данная реализация вычисления проигрывает предыдущей по скорости, но позволяет решить задачу с использованием меньшего количества элементарных процессоров.

Вышеприведенные рассуждения касались только тех случаев, когда во внутреннем условном операторе есть обе альтернативные ветви THEN и ELSE.

Если во внутреннем операторе IF есть только конструкция THEN, то трансляция будет происходить иначе. Оператор IF q THEN $a = b$ будет оттранслирован в различные варианты вычислительных структур в зависимости от типа переменной.

Для коммутационной переменной предыдущего значения не существует, поэтому запись

```
VAR A : Integer com;  
CADR L1  
    A := IF q THEN b;  
ENDCADR;
```

эквивалентна записи $A = q \& b$,

Если не выполнено условие q , то коммутационная переменная получает значение логического нуля.

При реализации параллельных вычислений с помощью кадров возможна ситуация, когда функционально законченные подграфы кадров изоморфны. Здесь целесообразно использовать подкадры, являющиеся структурными аналогами подпрограмм. Оператор объявления подкадра выглядит следующим образом:

```
SUBCADR Name ( F ) <СписокОператоров> ENDSUBCADR;
```

где SUBCADR, ENDSUBCADR - ключевые слова языка;

Name S - имя подкадра;

f - список формальных параметров, ограниченных круглыми скобками.

Обращение к подкадру производится по имени, после которого следует список фактических параметров, ограниченных круглыми скобками. Если подкадр является функцией, то необходимо в операторе описания данных Var указать имя подкадра и тип переменной. В подкадрах не разрешена рекурсия, в том числе и скрытая, то есть из подкадра не разрешается вызывать как собственный подкадр, так и последовательность подкадров, которая завершается вызовом текущего подкадра. Чтобы избежать побочных эффектов подкадров, целесообразно использовать в теле подкадра только

коммутационные переменные. При вызове подкадра происходит сцепление информационных каналов обмена, соответствующих фактическим параметрам, с формальными параметрами подкадра. В целом подкадры оформляются аналогично кадрам. В них запрещено использование операторов передачи управления, операторов описания вычислительных структур.

На рис. 6.10 приведены программа использования подкадров и графы вычислительных структур, в которые данная программа будет оттранслирована.

Несовпадение типов имен формальных и фактических параметров может приводить к ошибкам в работе программы.

Если подкадр имеет несколько выходов, то нельзя подставлять в формальные параметры одно имя фактического параметра.

```

SUBCADR REAL pirsum (A,N);
VAR (A[N], S, C[N/2, N/2] VEC) COM;
M=log2(N); K=1;
FOR I=1 TO N/2 DO
C[1,I]=A[K]+A[K+1];
K=K+2; END;
L=2; FOR J=2 TO M;
FOR I=L TO N/2 STEP L;
C[J,I]=C[J-1, I]+C[J-1, I-L/2]; END;
L=L*2; END; S=C[M,N/2];
RETURN S;END pirsum;

```

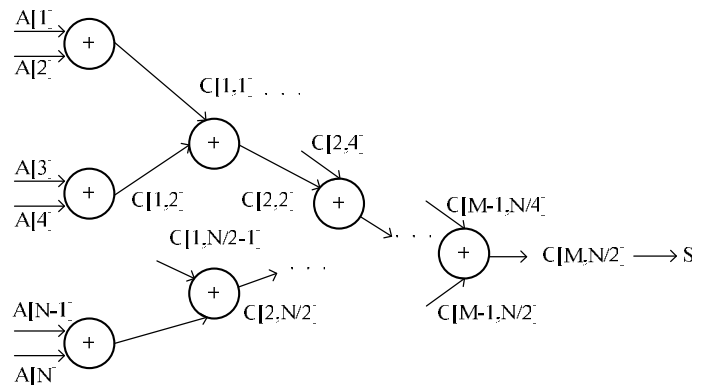


Рис. 6.10. Программа и граф-схема подкадра суммирования элементов массива

Это приводит к семантической ошибке, то есть нарушению правила единственной подстановки в теле кадра. Ниже приведен пример использования подкадров.

```

VAR A,B,C,D,E,F,G : REAL MEM;
SUBCADR BETA (A,B,C,D);
VAR (A,B,C,D) COM;
C=A+B/2;D=A-B*2;
ENDSUBCADR;
SUBCADR ALFA (In : A,B,C);

```

```

VAR (A,B,C) COM;
  C=A*3.14+B*B;
ENDSUBCADR;
CADR A1;
  BETA(A*2,4/B,D,C);
  ALFA(D*C,E*E,F);
ENDCADR;
CADR A2;
VAR K COM;
  BETA (F,F,G,K);
  ALFA (K,E,A);
ENDCADR;

```

В данном случае продемонстрированы две особенности использования подкадров, на которых следует подробнее остановиться: динамическая организация массива и подстановка параллельного параметра. Под динамической организацией массива переменных понимается тот факт, что размерность массива в подкадре *pirsum* определена как переменная VAR A [N], где N – параметр, задаваемый пользователем. Очевидно, что возможно использование как векторного, так и потокового массивов переменных. Подстановка потокового массива производится аналогично передаче в качестве параметров подпрограммы имени массива переменных в традиционных языках. Передаются начальный адрес массива, с которым начинает работать подкадр, и число членов массива, заданного пользователем. Кроме этого, необходимо передать адрес канала распределенной памяти, откуда извлекается данный массив. Таким образом, при вызове подкадра в качестве фактических параметров передается адрес потока в виде упорядоченной пары <Ak, Ag>, где Ak – адрес канала распределенной памяти, Ag – адрес начала данных. При работе с потоком Ak фиксирован, Ag изменяется при обращении к тому или иному элементу массива.

При обращении к подкадру с фактическим параметром векторного потока фиксируется Ag, а Ak изменяется. Адрес канала распределенной памяти является двойным. Во-первых, это адрес канала распределенной памяти, во-

вторых, - совокупность адресов для многокаскадного коммутатора, обеспечивающего доступ с требуемого канала распределенной памяти к решающему полю элементарных процессоров вычислительной структуры. Таким образом, $A_k = \langle A_{kv}, A_{kk} \rangle$, где A_{kv} – адрес канала распределенной памяти, A_{kk} – список адресов для многокаскадного коммутатора, обеспечивающего доступ к требуемому каналу распределенной памяти.

Кроме подкадров, в языке существуют традиционные подпрограммы. В общем случае подпрограмма имеет вид

SUBROUTINE Name (F) P ENDSUBROUTINE;

где SUBROUTINE, ENDSUBROUTINE - ключевые слова языка;

f - список формальных параметров подпрограммы;

P - список операторов подпрограммы.

В тело подпрограммы могут входить несколько кадров и операторы управления. Подпрограмма вызывается по имени, после которого следует список фактических параметров.

При реализации параллельных вычислений с помощью кадров возможна ситуация, когда необходимо использовать неизменяемую вычислительную структуру, которая в процессе выполнения программы при смене кадров не перегружается. Такая вычислительная структура должна представлять собой функционально законченный изоморфный граф. Для реализации неизменяемой вычислительной структуры в COLAMO используется конструкция LET, являющаяся структурным аналогом подкадра. Оператор объявления конструкции LET в РБНФ выглядит следующим образом:

LET <Имя> (<СписокПараметров>) <СписокОператоров> ENDLET;

В отличие от подкадров конструкция LET не может вызываться из кадров или подкадров и является независимой локальной ВС. При объявлении конструкции LET происходит сцепление информационных каналов обмена с формальными параметрами конструкции LET.

Все переменные конструкции LET, кроме помеченных в операторе VAR EXTERN, являются внутренними и не связаны с именами переменных в кадре.

В конструкции LET не разрешена рекурсия, в том числе и скрытая, то есть из конструкции LET не разрешается вызывать как собственную конструкцию LET, так и последовательность подкадров, кадров или подпрограмм. В теле конструкции LET рекомендуется использовать коммутационные переменные. В данной конструкции запрещено использование операторов передачи управления, операторов описания вычислительных структур. В процессе выполнения программы возможно отключение конструкции LET при помощи оператора STOPLET. Оператор отключения конструкции LET в РБНФ выглядит следующим образом:

STOPLET <Имя конструкции LET>;

Для эффективного управления работой кадров в языке COLAMO предусмотрена конструкция MCADR. Конструкция MCADR позволяет описать все объявленные кадры в виде одного массива, доступ к элементам которого осуществляется через индекс массива. Конструкция MCADR в РБНФ выглядит следующим образом:

MCADR : <Имя> [<СписокКадров>];

Использование конструкции MCADR позволит программисту выполнять смену кадров в любой последовательности. Кадры, объявленные в конструкции MCADR, должны быть описаны до объявления конструкции MCADR. Конструкция MCADR может использоваться как в теле кадра, так и вне кадра. При использовании такой конструкции вне кадра выполняется последовательная обработка вызываемых кадров. В случае использования конструкции MCADR в теле кадра выполняется процедура слияния вызываемых кадров в единый кадр, тем самым достигается возможность параллельной обработке вызываемых кадров. Пример использования конструкции MCADR представлен ниже.

Cadr Cadr1

.....

EndCadr;

Cadr Cadr2

.....

```

EndCadr;
.....
Cadr CadrN
.....
EndCadr;

Mcadr : MasCadr [Cadr1, Cadr2,....., CadrN];
// Последовательная обработка вызываемых кадров
For I := 0 to 5 Step 2 do
  MasCadr[I];

// Параллельная обработка вызываемых кадров
Cadr Cadr0
For J := 0 to 5 Step 1 do
  MasCadr[J];
EndCadr;

```

Язык высокого уровня для реконфигурируемой мультиконвейерной вычислительной системы со структурно-процедурной организацией вычислений позволяет эффективно реализовать любой тип параллельной обработки. Использование неявного описания параллелизма, виртуальных систем коммутации и распределенной памяти упрощает программирование. Язык отражает все особенности структурно-процедурной организации вычислительного процесса и позволяет оперативно разрабатывать прикладное программное обеспечение PMBC.

Трансляция языка COLAMO

Трансляция языка COLAMO в язык ассемблера ARGUS заключается в генерации двух компонентов: структурно-процедурной программы на языке ARGUS, реализованной на реконфигурируемой структуре, и управляющей программы, выполненной в host-машине.

В процессе трансляции выполняется проверка входного текста параллельной программы на выявление синтаксических и семантических

ошибок. После проверки входного текста программы транслятор генерирует четыре компонента: структурный, потоковый, процедурный и управляющий.

В результате успешной генерации всех компонентов выполняется объединение структурного, потокового и процедурного компонентов в параллельную структурно-процедурную программу на языке ассемблера ARGUS. Структурный и потоковый компоненты образуют кадровую форму задачи. Процедурный компонент реализует процесс настройки PBC на различные кадры. Управляющий компонент является самодостаточным, представляет собой исполняемую программу (exe-файл) и обеспечивает загрузку и выгрузку программной и числовой информации в PBC из host-машины, а также контролирует процесс решения задачи. Алгоритмическая схема работы транслятора представлена на рис.6.11.

Структурный компонент параллельно-конвейерной программы описывает информационные аппаратно-реализуемые информационные подграфы задачи. При генерации структурного компонента каждая операция или макрооперация, указанная в теле кадра, отображается на отдельный вычислительный ресурс. Аппаратный ресурс, отведенный для выполнения операции (макрооперации), не может быть использован для выполнения других операций в процессе выполнения кадра. Если в программных конструкциях кадра на языке COLAMO используются операторы цикла, переменные которых применяются для индексации массивов с параллельным доступом данных (vector), то в этом случае тело цикла мультиплицируется в структуре PBC в соответствии с параметрами цикла. Следует отметить, что при реализации условного перехода в теле кадра будут аппаратно реализованы все альтернативные ветви вычислений, а также устройства для выбора определенного варианта. Алгоритм генерации структурного компонента представлен на рис.6.12.

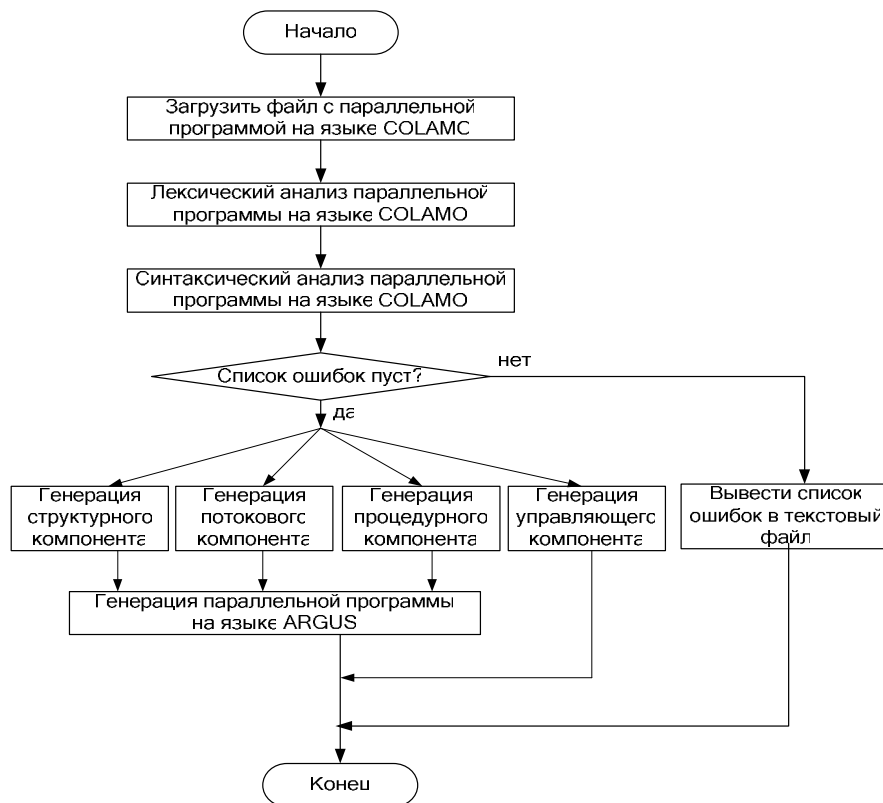


Рис. 6.11. Укрупненный алгоритм работы транслятора с языка COLAMO

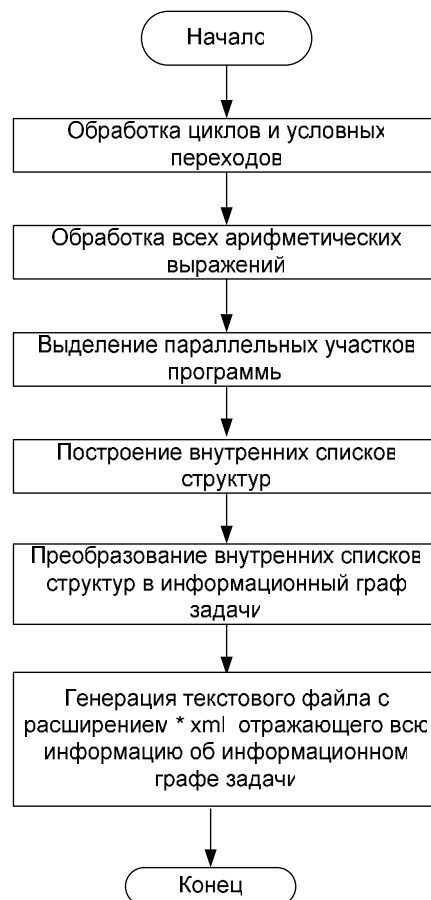


Рис. 6.12. Алгоритм генерации структурного компонента

Пример преобразования параллельной программы в эквивалентный информационный граф представлен на рис. 6.13.

```

VAR a,b,c,d,e,g : Array [2 : Vector 10 :
Stream] Mem;
VAR Z : Integer Com;
Cadr Cadr1
  For I := 0 to 1 do
    For J := 0 to 9 do
      Begin
        A[I,J] := B[I,J] + B[I,J + 3];
        D[I,J] := C[I,J] + (B[I,J] * G[I,J]);
      End;
    EndCadr;
  EndCadr;

```

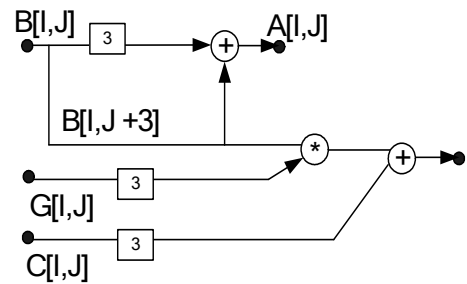


Рис. 6.13. Программа и эквивалентная ей граф-схема

Здесь вершина в виде кружка соответствует входной/выходной вершине графа, которая соотносится с распределенной памятью. Вершины графа, обозначенные светлыми кружками, являются арифметико-логическими операциями, реализованными на элементарных процессорах. Вершины графа, обозначенные прямоугольниками, указывают на задержку информации на определенное число тактов.

Потоковый компонент структурно-процедурной программы обеспечивает прохождение информационных потоков через вычислительную структуру, заданную структурным компонентом, а также бесконфликтность параллельной записи и чтения в каналы распределенной памяти РВС.

Алгоритм генерации потокового компонента представлен на рис. 6.14.

Процедура выделения информационных потоков заключается в сортировке информационных вершин графа по принадлежности их к каналам памяти с учетом последовательности обработки.



Рис. 6.14. Алгоритм генерации потокового компонента

На следующем примере представлен фрагмент трансляции параллельной программы в потоковый компонент.

<pre> VAR a,b,c,d,e,g : Array [2 : Vector, 10 : Stream] Mem; VAR Z : Integer Com; Cadr Cadr1 For I := 0 to 1 do For J := 0 to 6 do Begin A[I,J] := B[I,J] + B[I,J + 3]; D[I,J] := C[I,J] + (B[I,J] * G[I,J]); End; EndCadr; EndCadr; </pre>	<pre> Var A,B,C,D,G : Array [2 : Vector 10 : Stream] Mem; Cadr Cadr1 DMC[0.B1] Read B[0] Step 1 repeat 12; DMC[0.C1] Read C[0] Step 1 repeat 12; DMC[0.G1] Read G[0] Step 1 repeat 12; DMC[0.A1] Write A[0] Step 1 repeat 12; DMC[0.D1] Write D[0] Step 1 repeat 12; DMC[0.B2] Read B[1] Step 1 repeat 12; DMC[0.C2] Read C[1] Step 1 repeat 12; DMC[0.G2] Read G[1] Step 1 repeat 12; DMC[0.A2] Write A[1] Step 1 repeat 12; DMC[0.D2] Write D[1] Step 1 repeat 12; EndCadr; </pre>
---	---

Оператор *Read* обеспечивает чтение потока данных из контроллера памяти в специализированную конфигурацию, а оператор *Write* выполняет запись потока данных.

Процедурный компонент программы предназначен для описания последовательности выполнения программы. В языке ассемблера ARGUS ход выполнения программы определяется последовательностью расположения кадров в тексте программы. При трансляции параллельной программы в процедурную составляющую основные действия производятся над межкадровыми операторами языка COLAMO.

Алгоритм генерации процедурного компонента представлен на рис. 6.15.

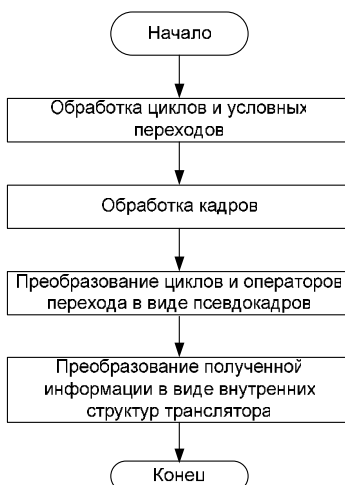


Рис. 6.15. Алгоритм генерации процедурного компонента

На следующем примере представлен фрагмент трансляции параллельной программы в процедурный компонент.

VAR a,b,c,d,e,g : Array [2 : Vector	Program Promer;
10 : Stream] Mem;	Var A,B,C, D : ParamAddress MEM;
VAR Z : Integer Com;	Var N, K, M : Number;
For I:= 0 to 5 Do	Define N = 5; Define K = 6; Define M = 1;
Begin	Label M0, M1;
Cadr Cadr1	M0 : CADR Cadr1
A[0,0] := B[0,0] + B[0,3];	<Потоковые операторы>
D[0,0] := C[0,0] + (B[0,0] *	EndCadr;
G[0,0]);	CADR CadrIF

EndCadr;	If Mem Goto M1;
If A > 0 then	Goto M2;
Cadr Cadr2	EndCadr;
A[0,0] := B[0,0] + B[0,3];	M1 : CADR Cadr2
D[0,0] := C[0,0] + (B[0,0] *	<Потоковые операторы>
G[0,0]);	EndCadr;
EndCadr;	CADR CadrFOR
End;	M2: Loop 4 Goto M0;
	EndCadr;

Псевдокадр CadrIf реализует оператор условного перехода (IF).

Псевдокадр CadrFor реализует внешний цикл программы.

Управляющий компонент программы представляет собой программу на языке программирования однопроцессорных ЭВМ, например Pascal. Результатом трансляции параллельной программы в управляющий компонент является исполняемая программа.

Алгоритм трансляции программы с языка COLAMO в управляющий компонент заключается в трансляции операторов загрузки и выгрузки данных, трансляции кадровых конструкций языка в программу на языке Pascal. Полученный файл компилируется средствами традиционных языков в исполняемую программу.

На следующем примере представлен фрагмент трансляции параллельной программы в управляющую составляющую.

Program Primer;	Program 1;
Var A,B,C,D:array [N : Stream, K :	Uses SysUtils;
Stream]: Mem;	TempLongword : PLongWord;
	Data64 : PInt64; InputFile : File; OutputFile :
ReadFile (A, FF_STR, "C:\Input.txt");	File; Str : String;
	Begin
Cadr Cadr1	LoadFile(D:\Программы\Среда_для_компиля
For I := 0 to N-1 do	topa\Result\OutFile.out);
For J := 1 to K-1 do	{Загрузка данных из файла в

D[I, J] := (A[I,J] + A[I,J-1]) + 5;	распределенную память}
EndCadr;	// Запуск всех КРП
	RunAllDMC(0, 16);
WriteFile (D, FF_STR, "C:\Output.txt");	// Обработка останова
	Repeat
End_Program.	ReadCOPreg(0, 2, TempLongword);
	bit5 := Integer(TempLongword^) and 32;
	until Bit5 = 32;
	{Выгрузка данных из распределенной памяти
	в файл}
	End;

6.4. Среды проектирования параллельно-конвейерных программ

Среда разработки «Argus IDE» позволяет разрабатывать параллельные программы на языке структурно-процедурных параллельных вычислений [50]. К особенностям данной системы можно отнести многооконную организацию пользовательского интерфейса, где используются два типа окон (рис.6.16). К первому типу относятся окна специализированного текстового редактора, предназначенного для ввода исходных текстов параллельных программ, ко второму - множество вспомогательных окон, через которые осуществляется доступ к элементам управления процессом разработки.

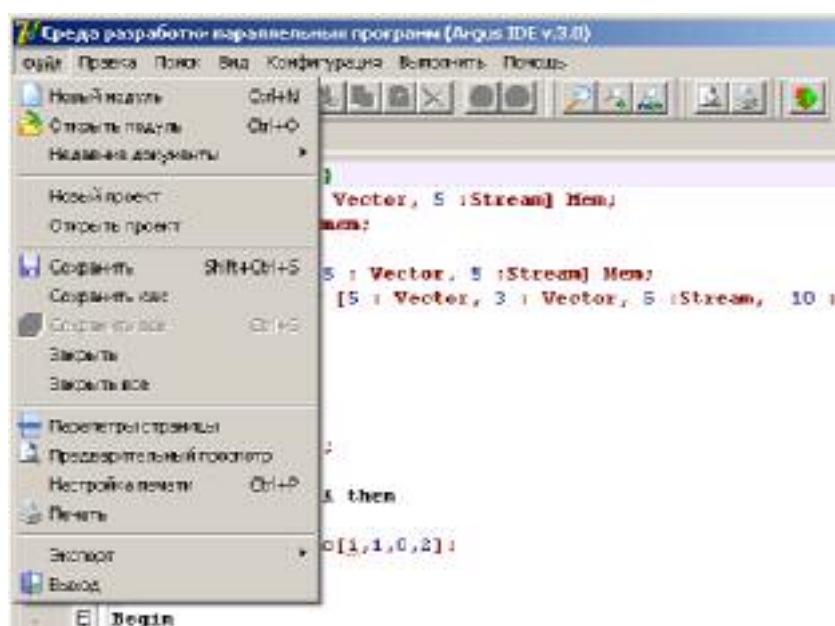


Рис. 6.16. Расположение секций меню “Файл”

К элементам управления и контроля процессом разработки относятся список сообщений разработчику, объектная модель программы, таблица свойств объектов программы и журнал событий. Имеется справочная система как по принципам работы среды разработки, так и по языку структурно-процедурного параллельного программирования «Argus» (рис.6.17).

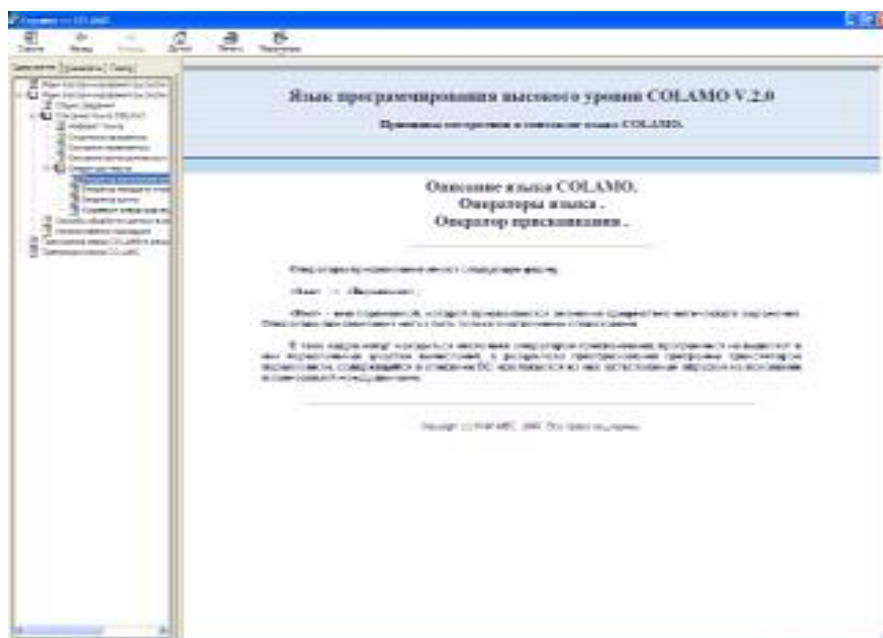


Рис. 6.17. Внешний вид справочной системы

В процессе трансляции после успешно пройденных этапов синтаксического контроля и формирования объектной модели программы происходит обращение к подсистеме отображения графа задачи на архитектуру системы, которая определяет загрузку физических ресурсов базовых модулей в соответствии со структурой задачи (рис. 6.18).

Использование данной подсистемы освобождает разработчика от явного программирования коммутационной системы, что является довольно трудоемким процессом. Язык параллельного структурно-процедурного программирования «Argus» позволяет использовать конструкции как прямого программирования связей между элементами базовых модулей, так и задавать схемы виртуальных компонентов параллельных структурно-процедурных программ, для которых конфигурирование ресурсов будет выполнено автоматически (рис.6.19).

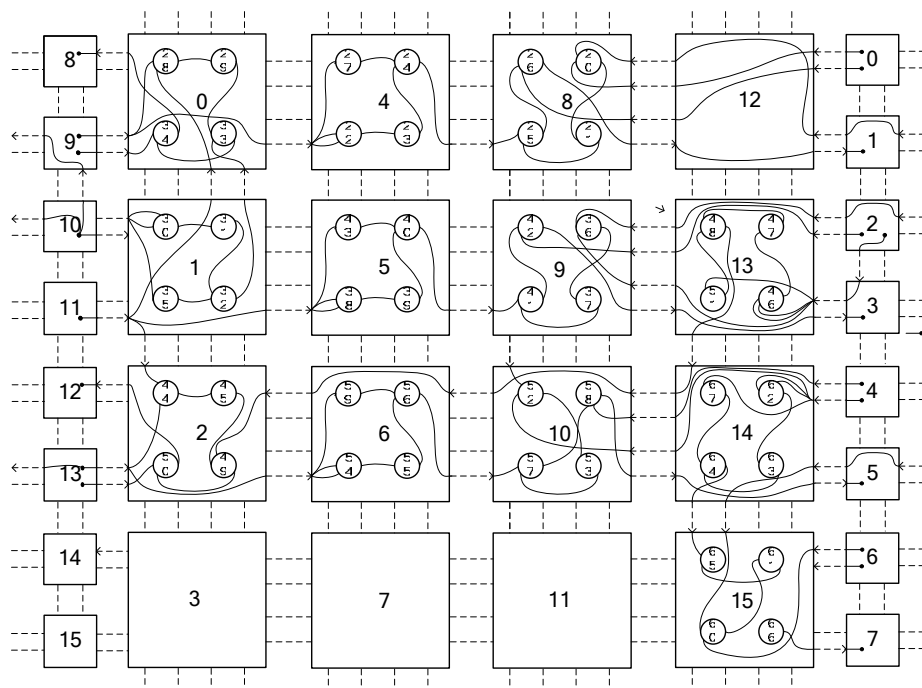


Рис. 6.18. Отображение информационного графа для решения задачи математической физики на базовый модуль МВС с ортогональной коммутационной структурой

Для наглядного представления структурных компонентов параллельно-конвейерных программ РМВС разработана специальная среда.

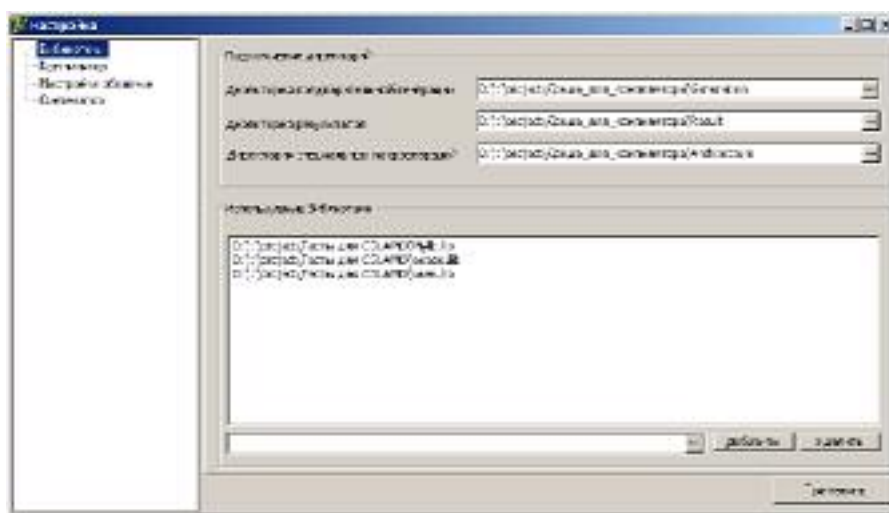


Рис. 6.19. Настройка параметров при выборе пункта “Библиотеки”

Среда разработки «Easyo IDE» позволяет разрабатывать компоненты параллельных программ в виде граф-схем для различной конфигурации. Язык граф-схем используется для эквивалентного описания компонентов параллельных программ, для которых их текстовое представление на языке структурно-процедурного параллельного программирования не является столь

выразительным, как графическое. Граф-схема - это альтернативная вспомогательная форма, которая используется для описания структурной части задачи, в то время как для описания процедурной составляющей используется текстовое описание последовательных программ контроллеров распределенной памяти на языке Argus или COLAMO.

Если текстовый редактор исходных модулей параллельных программ подразумевает последовательный ввод символов, то графический – соответственно, последовательный ввод графических объектов граф-схем компонентов параллельных программ (рис.6.20).

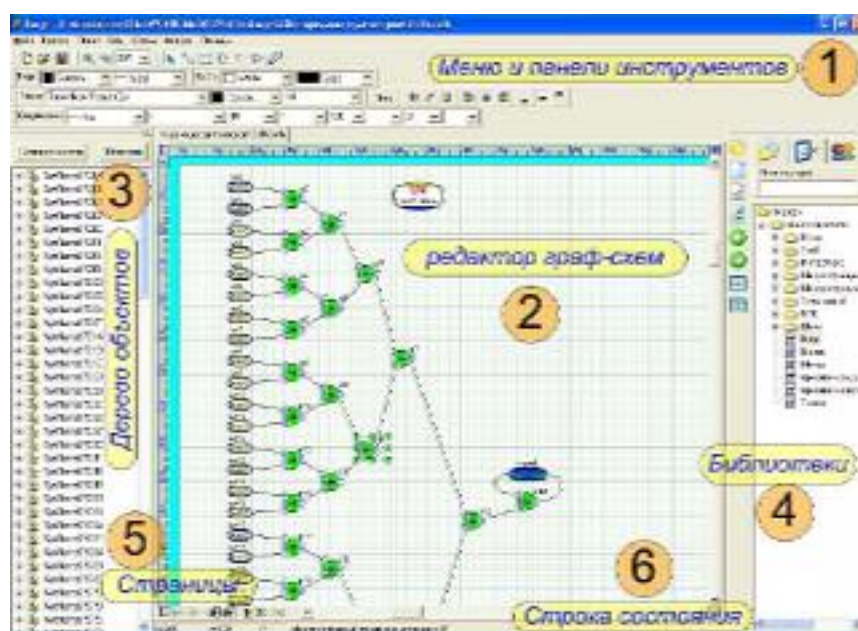


Рис. 6.20. Среда разработки компонентов в виде граф-схем

Для текстового редактора рабочий файл представляет собой список строк, для графического редактора файл граф-схем состоит из множества страниц одинакового размера. Под графическими объектами будем понимать некоторое множество графических примитивов, объединенных в группы. Под графическим примитивом будем понимать некоторую последовательность кривых первого и второго порядков. В качестве графической функции, отображаемой на точках сегмента, используется *кривая Безье*. Изображение примитива, построенное из таких сегментов, может быть замкнутым, когда начальная и конечная точки множества сегментов совпадают.

Для однозначной идентификации у каждого примитива существует текстовый атрибут, задающий класс объекта. Для граф-схем компонентов параллельных программ разработана базовая классификационная схема, на основе которой возможна разработка идентифицируемых объектов граф-схем.

Графические примитивы объединяются в иерархические группы, причем для каждой группы возможно задать как текстовый атрибут, так и классификационный. Объект должен быть классифицирован посредством задания соответствующего текстового атрибута, иначе объект во время трансляции не будет идентифицирован.

Для определения структурных взаимосвязей между объектами граф-схем используется специальный класс незамкнутых графических примитивов, именуемых *связями*. Связи устанавливаются между объектами через задания в их пространстве связующих точек, координаты которых привязаны к координатам владеющих ими объектов.

Для системы «Easyo IDE» разработана базовая библиотека классифицированных объектов, из которых составляются граф-схемы компонентов структурно-процедурных параллельных программ. Быстрый доступ к таким объектам осуществляется через одно из вспомогательных окон на основе технологии «Drag&Drop» [118] (рис.6.21).

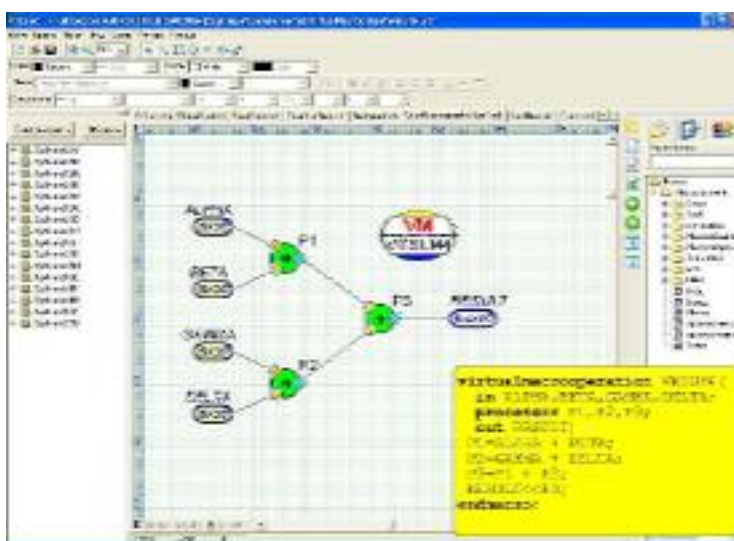


Рис. 6.21. Среда разработки компонентов в виде граф-схем

В процессе разработки граф-схема транслируется в эквивалентное текстовое представление в виде включаемого модуля на языке структурно-процедурных параллельных вычислений. Возможна и обратная трансляция из текстового представления в графическое.

Это обеспечивается на уровне взаимодействия сред разработки «Argus IDE» и «Easy IDE» через подсистему трансляции между двумя эквивалентными представлениями.

Основное назначение среды разработки - ускорение создания структурно-процедурных программ для реконфигурируемых мультиконвейерных вычислительных структур различной конфигурации за счет интеграции и эффективного взаимодействия всех необходимых средств в одну среду. Система разработки параллельных программ представляет их как в текстовом виде, так и в виде граф-схем. Разработчик параллельных программ и программных компонентов может оперировать с их текстовым и графическим представлением посредством специализированных редакторов, а также средств трансляции.

Редактор граф-схем сочетает в себе возможности как универсального графического редактора, так и специализированного, предназначенного для разработки программных компонентов. Для оперативного создания граф-схем параллельных программ разработана библиотека базовых элементов, в которую входят все необходимые для разработки граф-схем «мастера». Библиотека «мастеров» включает как отдельные функциональные устройства: счетчики, сумматоры, умножители, так и аппаратно-реализованные крупные и функционально законченные макрооперации. Пользователи могут создавать собственные библиотеки «мастеров» и использовать их вместо стандартной библиотеки. Ниже перечислены компоненты параллельных программ, представимые в виде граф-схем:

- виртуальная макрооперация;
- реальная макрооперация;
- реальная макропамять;

- векторная макропамять;
- кадр.

Виртуальные макрооперации могут быть реализованы на произвольном ресурсе. Отображение компонентов макроопераций на ячейки ПЛИС, а также трассировку информационной связи реализует специальная утилита транслятора. Реальные макрооперации реализуются на определенном фиксированном аппаратном ресурсе и не могут быть изменены ни пользователем, ни транслятором.

Каждая граф-схема разрабатывается в соответствии с определенной классификацией, которая позволяет интерпретировать те или иные графические объекты. Граф-схема макрооперации состоит из ряда графических объектов, которые представляют:

- входы;
- процессоры;
- выходы;
- тип схемы.

Между графическим и текстовым представлениями компонентов параллельных программ существует однозначное соответствие.

Процесс разработки осуществляется как через основное меню среды, так и через панель инструментов. Меню «Файл», «Правка», «Поиск», «Вид», «Помощь» содержат ряд общепринятых команд для таких систем. Рассмотрим более подробно специализированные функции.

Разработчик может расширять набор библиотечных элементов, внося изменения непосредственно через служебное окно управления. Практически любая пользовательская граф-схема может быть использована в различных проектах в качестве библиотечной конструкции.

Взаимосвязь со средой разработки параллельных программ на языке структурно-процедурных вычислений «Argus» осуществляется через функцию трансляции в текст параллельной программы. В случае успешной трансляции

полученный текст открывается в новом окне редактора среды разработки параллельных программ «Argus IDE».

Отладчик параллельных программ на «Argus Debugger» позволяет осуществлять интерактивную отладку на базовых модулях системы (рис. 6.22).

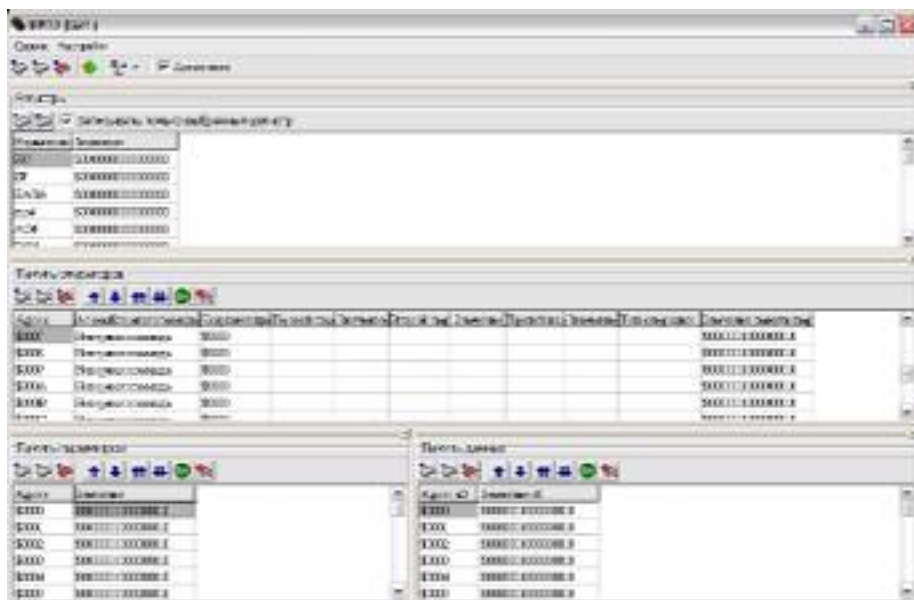


Рис. 6.22. Отладчик параллельных программ «Argus Debugger» - форма КРП

Отличительными особенностями программы является интерактивная возможность доступа к контроллеру распределенной памяти базовых модулей, к регистрам контроллера и сегментам памяти, находящимся под их управлением. Для каждого контроллера памяти организовано свое специализированное окно просмотра и ввода значений регистров и сегментов памяти.

Функции контроля выполнения вычислений базовыми модулями реализованы в меню программы и доступны через набор «горячих клавиш». Через программу «Argus Debugger» возможна загрузка исполняемых модулей параллельных программ и выгрузка их в указанный файл.

Основной задачей среды разработки «Argus IDE» является ускорение процесса разработки параллельных программ различной конфигурации за счет интеграции всех необходимых для этого средств в одну среду, обеспечивающую интерактивную подготовку исходных текстов и загрузочных модулей параллельных решений. Пользовательский интерфейс включает:

1. *Меню и панели инструментов.* Через основное меню программы и панели инструментов осуществляется доступ к основным функциям среды разработки (рис. 6.23).



Рис. 6.23. Меню программы

2. *Специализированный текстовый редактор.* Разработка программ на языке структурно-процедурных вычислений «Argus» осуществляется в специализированном текстовом редакторе (рис. 6.24), отличительными особенностями которого являются:

- соответствующие синтаксису языка «Argus» атрибуты вывода текста;
- интерактивное взаимодействие управляющими элементами, позволяющее осуществлять быструю контекстную навигацию по тексту;
- вывод на печать с сохранением атрибутов вывода;
- поиск и замена строк.

3. *Поля редактора.* Служат для индикации различных атрибутов строк параллельной программы, таких как точка останова, предупреждение или ошибка, точка исполнения.



Рис. 6.24. Специализированный текстовый редактор

4. *Дерево объектов параллельной программы.* Объектная древовидная форма представления параллельной программы формируется в результате трансляции. Такая форма позволяет рассматривать объекты параллельной программы в компактной иерархической форме (рис. 6.25).



Рис. 6.25. Дерево объектов параллельной программы

Также возможна навигация по тексту параллельной программы с помощью данной формы (рис. 6.26).

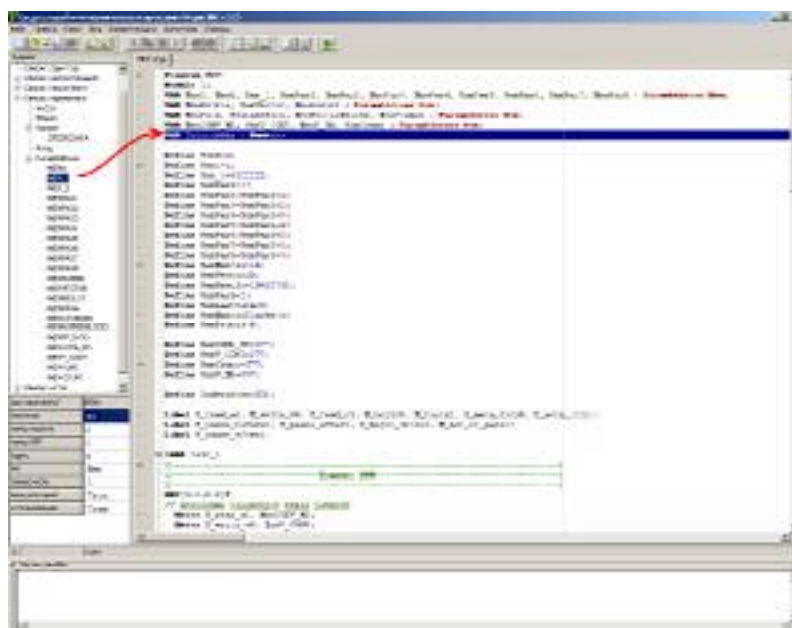


Рис 6.26. Навигация по тексту параллельной программы с помощью дерева объектов

5. *Таблица свойств объектов программы.* Для вывода информации об объектах программы используется таблица свойств, которая соответствует выбранному элементу дерева объектов параллельной программы (рис. 6.27).

Имя параметра	PCML
Включено	10
Номер модуля	0
Номер КОП	3
Адрес	1
Тип	Man
Размерность	1
Уникальность	True
Использование	True

Рис. 6.27. Таблица свойств объектов программы

6. *Информационная панель редактора текста программ.* На панель выводится информация о положении курсора, индикатор режима вставки, индикатор модификации текста (рис 6.28).

7. *Список сообщений пользователю.* Список строк, формируемый во время трансляции, в котором выводится информация об ошибках и предупреждениях. Возможна навигация по тексту программы через список сообщений.

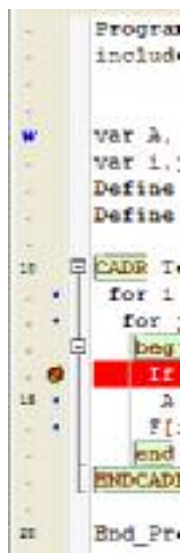


Рис. 6.28. Информационная панель редактора текста программ

Имеется экранная форма системы разработки, где показываются результаты трансляции, объектная модель параллельной программы (рис. 6.29).

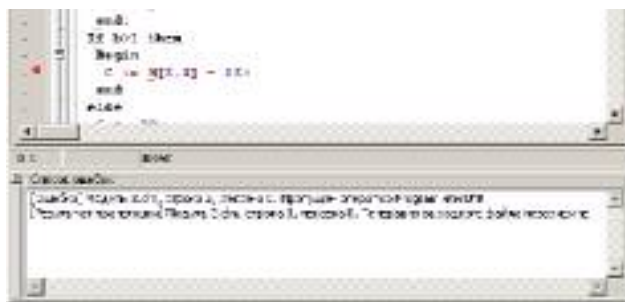


Рис. 6.29. Панель «Список ошибок»

Окно сообщений позволяет переходить к соответствующим позициям в тексте программы. Поля редактора отображают текущее состояние каждой строки. В зависимости от содержания строки могут содержать ошибки или предупреждения. Вспомогательные окна, такие как окно сообщений, дерево объектной модели можно размещать по своему усмотрению. Интерфейс организован стандартным образом, например, как в среде «Microsoft Visual Studio» или «Borland Delphi». Имеется возможность вывода исходных текстов параллельных программ на печать с сохранением атрибутов вывода соответствующих синтаксису описания структурно-процедурных вычислений.

Интегрированная среда разработки программ и программных компонентов на языке структурно-процедурных вычислений «Argus IDE» позволяет одновременно работать с несколькими исходными файлами посредством элемента управления с многостраничной организацией.

Управление процессом разработки осуществляется как через основное меню среды, так и через панель инструментов. Меню «Файл», «Правка», «Поиск», «Вид», «Помощь» содержат общепринятые команды для таких систем. Компиляция исходных модулей загрузки и запуска исполняемых модулей осуществляется через меню «Выполнить» (рис. 6.30).

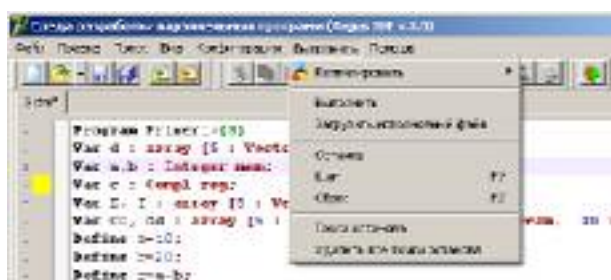


Рис. 6.30. Меню «Выполнить»

Данное меню реализует следующие команды:

1. «*Компилировать*». При выборе данного пункта выполняется процедура компиляции, которая состоит из нескольких этапов:

- выполняется сохранение исходных файлов;
- текущий файл считается основным, выполняется трансляция текущего файла;
- в случае если трансляция прошла успешно, то есть не было ошибок, выполняется генерация загрузочного модуля;
- в случае если список сообщений транслятора не пуст, то активизируется окно сообщений;
- строится объектная модель параллельной программы.

2) «*Выполнить*». Выполняется процедура реализации текущей параллельной программы на вычислительной системе, которая состоит из следующих этапов:

- если нет загрузочного модуля, то выполняется процедура компиляции;
- если дата последней модификации загрузочного модуля более ранняя, чем у исходного файла, то выполняется процедура компиляции;
- если загрузочный модуль создан, то выполняется процедура его загрузки и исполнения на системе.

3) «*Загрузить*». При выборе данного пункта выполняется процедура загрузки исполняемого модуля текущей параллельной программы.

Среды разработки параллельно-конвейерных программ реконфигурируемых мультikonвейерных вычислительных структур обеспечивают широкие сервисные возможности и существенно облегчают работу программиста.

Послесловие

Высокопроизводительные многопроцессорные вычислительные системы предназначены для решения вычислительно трудоемких задач, решение которых на обычных персональных компьютерах либо требует недопустимо больших временных затрат, либо вообще невозможно. Однако в отличие от персональных компьютеров, реальная производительность которых практически пропорциональна тактовой частоте работы процессора, реальная производительность многопроцессорной системы зависит от многих характеристик, в том числе от архитектуры, от организации процессорных узлов, от тактовой частоты и т.д. Поэтому реальная производительность различных многопроцессорных систем при решении одной и той же задачи может отличаться на порядок.

До последнего времени развитие суперкомпьютерных технологий шло в основном по пути «крупноблочной» сборки, когда показатели производительности достигались не за счет качества архитектурных и схемных решений, а путем технологических решений на уровне микропроцессоров и систем коммутации. В результате в настоящее время наибольшее распространение получили многопроцессорные системы кластерного типа, которые строятся на основе широкодоступных компьютерных узлов и коммутационных сетей. Это вызвано их относительной дешевизной и простотой программирования, поскольку для них достаточно легко может быть адаптировано программное обеспечение, отработанное на персональном компьютере. Безусловно, все это открывает хорошие перспективы по использованию кластерных МВС для широкого круга пользователей - исследователей, решающих задачи моделирования, конструирования, вычислительного эксперимента и т.п. В то же время кластерные МВС показывают хорошие характеристики реальной производительности только при решении задач, которые могут быть разбиты на отдельные слабосвязанные фрагменты, каждый из которых может быть поручен своему узлу кластера. При

решении других типов задач реальная производительность кластерных МВС резко падает и, более того, может даже уменьшаться при увеличении числа процессоров в системе.

В результате, в некотором смысле, назрела революционная ситуация: пользователи кластерных систем не могут добиться их высокой реальной производительности при решении своих задач, а производители кластерных МВС не хотят ничего менять. Эта «революционная ситуация» может быть разрешена только путем предоставления пользователю больших возможностей не только в плане процедурного программирования суперЭВМ, но и архитектурного программирования ее аппаратного параллелизма. Именно такими возможностями обладают реконфигурируемые мультиконвейерные вычислительные структуры. Если в кластерной МВС пользователь пытается адаптировать структуру решаемой задачи под архитектуру МВС, то в реконфигурируемых структурах, наоборот, архитектура системы адаптируется пользователем под решаемую задачу. При этом в структуре формируется аппаратный аналог информационного графа решаемой задачи, что в принципе позволяет добиться теоретического максимума реальной производительности при ее решении.

Но за все надо платить. В данном случае платой за высокую реальную производительность являются сложности программирования реконфигурируемых МВС, поскольку в данном случае программирование должно осуществляться не только на процедурном, но и на структурном (аппаратном) уровне. Поэтому реконфигурируемые мультиконвейерные вычислительные структуры наилучшим образом подходят для решения потоковых задач, когда задачу необходимо решать максимально быстро при различных исходных данных, но по единому алгоритму. В этом случае, потратив один раз время на программирование архитектуры реконфигурируемых многопроцессорных вычислительных структур, можно обеспечить максимальную скорость обработки большого массива (потока) данных.

Как известно, история развивается по спирали. Когда 30 лет назад появилась концепция создания реконфигурируемых вычислительных структур, она была воспринята с большим энтузиазмом. Однако в связи с возникшими техническими трудностями на пути практической реализации данной концепции через некоторое время пыл угас, и идея начала потихоньку забываться. Прошло более 20 лет, прежде чем появление ПЛИС высокой степени интеграции вдохнуло новую жизнь в идею создания реконфигурируемых мультиконвейерных структур.

В настоящее время подходу к созданию РМВС на базе ПЛИС уделяется все большее внимание со стороны разработчиков высокопроизводительных вычислительных средств. В частности, по отзывам участников конференции «Supercomputing-2007», состоявшейся в г. Рино (США) в ноябре 2007 года, не менее трети представленных докладов в той или иной мере затрагивало вопросы повышения производительности многопроцессорных систем за счет использования реконфигурируемых узлов, в том числе реализованных на основе ПЛИС. Все это позволяет утверждать, что «ледниковый период» в области архитектурного программирования суперкомпьютеров на уровне аппаратного параллелизма заканчивается. При этом если западные разработчики только недавно начали использовать технологии реконфигурируемых вычислений, то более чем 30-летний опыт отечественных исследований и разработок в данной области дает нам хорошую стартовую позицию и позволяет небезосновательно рассчитывать на завоевание ведущих позиций на мировом рынке реконфигурируемых мультиконвейерных вычислительных структур. И мы очень надеемся, что настоящая монография послужит этому важному с точки зрения стратегического развития России делу.

ЛИТЕРАТУРА

- [1] Аладышев О.С., Дикарев Н.И., Овсянников А.П. и др. СуперЭВМ: области применения и требования к производительности - Известия ВУЗов. Электроника, 2004, №1, 13-17.
- [2] Алексин Ю.П., Бартиш М.Я, Шмойлов В.И. и др. Мультиконвейерные вычислительные структуры на однородных средах - Львов: ФМИ АН УССР, 1986, 74.
- [3] Алмаши А.Г., Резников Д.Ю., Шимкив А.П. Пособие для программиста ОВС. - Львов: НТЦ "Интеграл", 1991, 53.
- [4] Антипов В.Н., Горяинов В.Т., Кулин А.Н. и др. Радиолокационные станции с цифровым синтезированием апертуры антенны / Под ред. Горяинова В.Т. - М.: Радио и связь, 1988, 304.
- [5] Антоненко А.И., Голиченко Е.П., Шмойлов В.И. и др. Система разработки микропрограммных модулей ОВС. - Львов: НТЦ "Интеграл", 1991, 71.
- [6] Бабенко Л.К., Макаревич О.Б., Чефранов А.Г. Принципы описания и организации асинхронных крупноблочных вычислений в многопроцессорных системах - Электронное моделирование, 1988, №3, 13-17.
- [7] Банковский А.Г., Гильбурд М.М., Шмойлов В.И. Однородные вычислительные среды с потоком команд. - Львов: НТЦ "Интеграл", 1989, 63.
- [8] Банковский А.Г., Карпа Н.М., Шмойлов В.И. и др. Библиотека микропрограммных модулей арифметических и логических операций для чисел с фиксированной запятой. - Львов: НТЦ "Интеграл", 1990, Ч.1, 133.
- [9] Банковский А.Г., Карпа Н.М., Шмойлов В.И. и др. Библиотека микропрограммных модулей арифметических и логических операций для чисел с фиксированной запятой. - Львов: НТЦ "Интеграл", 1990, Ч.2, 113.
- [10] Бачериков Г.И., Богачёв М.П., Шмойлов В.И. и др. Мультиконвейерные вычислительные структуры на однородных средах. - Львов: ФМИ АН УССР, 1985, 70.
- [11] Бачериков Г.И., Брановский Ю.С., Шмойлов В.И. и др. Программное обеспечение ОВС. - Львов: ИППММ АН УССР, 1990, 56.
- [12] Богачев М.П. Архитектура вычислительной системы с однородной структурой / В кн. Однородные вычислительные среды. - Львов: ФМИ АН УССР, 1981, 15-21.
- [13] Богачёв М.П., Бачериков Г.И., Геворкян В.И. и др. Однородные вычислительные структуры в системах связи. - Ленинград: Военная академия

связи, 1987, 82.

- [14] Бондарев А.Н., Елисеев В.Г., Шмойлов В.И. и др. Библиотека микропрограммных модулей арифметических операций для комплексных чисел. - Львов: НТЦ "Интеграл", 1991, 107.
- [15] Валиев К.А. Квантовые компьютеры и квантовые вычисления, УФН, 175 (1), 3 (2005).
- [16] Бурцев В.С. Информационно-вычислительные системы с автоматическим распараллеливанием вычислительных процессов - Труды Международной конференции «Параллельные вычисления и задачи управления», 2000.
- [17] Варшавский В.И., Мораховский В.Б. и др. Однородные структуры. - М.: Энергия, 1973, 150.
- [18] Василькевич А.В., Дмитриев А.Г., Шмойлов В.И. и др. Библиотека микропрограммных модулей для реализации задач обработки изображений на ОВС. - Львов: НТЦ "Интеграл", 1991, 225.
- [19] Введение в цифровую фильтрацию / Под ред. Р. Богнера и А. Константинодиса. - М.: Мир, 1976. 216.
- [20] Векторизация программ: теория, методы, реализация / Сборник переводов статей. - М.: Мир, 1991, 246-267.
- [21] Воеводин В.В. Математические модели и методы параллельных процессов. -М.: Наука, 1986, 286.
- [22] Воеводин В.В. Математические основы параллельных вычислений. - М.: Изд-во МГУ, 1991, 345.
- [23] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - С.-Петербург: «БХВ-Петербург», 2002, 599.
- [24] Глазков М.А., Малахов В.В., Савостин М.Ф. Видеопроцессор на ОВС. - Львов: НТЦ "Интеграл", 1990, 47.
- [25] Голиченко Е.П., Костив Ю.В., Шмойлов В.И. и др. Микропрограммные модули арифметических и логических операций для чисел с плавающей запятой. - Львов: НТЦ "Интеграл", 1990, 41.
- [26] Голиченко Е.П., Костив Ю.В., Шмойлов В.И. и др. Библиотека микропрограммных модулей арифметических и логических операций для целых чисел. - Львов: НТЦ "Интеграл", 1991. 210.
- [27] Голиченко Е.П., Фурдыло А.И., Шмойлов В.И. Программирование однородных вычислительных сред. - Львов: НТЦ "Интеграл", 1989, 50.
- [28] Дейкстра Э. Взаимодействие последовательных процессов // В сб. Языки

- программирования / Под ред. Женюи Ф. - М.: Мир, 1972, 9-86.
- [29] Дейкстра Э. Дисциплина программирования. - М.: Мир, 1978, 278.
- [30] Джахани Н. Язык Ада. - М.: Мир, 1988, 540.
- [31] Джоунз Г. Программирование на языке Оккам /Пер. с англ. - М.: Мир, 1989, 208.
- [32] Димитриев Ю.К., Хорошевский В.Г. Вычислительные системы из мини-ЭВМ. - М.: Радио и связь, 1982, 304.
- [33] Дордопуло А.И., Каляев И.А., Левин И.И., Семерников Е.А. Семейство многопроцессорных вычислительных систем с динамически перестраиваемой архитектурой // Материалы Четвертой Международной научной молодежной школы «Высокопроизводительные вычислительные системы». - Таганрог: Изд-во ТТИ ЮФУ, 2007, 68-74.
- [34] Евреинов Э.В. О микроструктуре элементарных машин вычислительной системы. Вычислительные системы - Сб. трудов. Новосибирск, 1962, Вып. 4, 3-16.
- [35] Евреинов Э.В., Косарев Ю.Г. О вычислительных системах высокой производительности - Известия АН СССР, «Техническая кибернетика», 1963, №4.
- [36] Евреинов Э.В. Теоретические основы построения универсальных вычислительных сред. Вычислительные системы - Сб. трудов, Новосибирск, 1965, Вып. 16, 3-72.
- [37] Евреинов Э.В., Косарев Ю.Г. Однородные универсальные вычислительные системы высокой производительности. - Новосибирск: Наука, 1966, 308.
- [38] Евреинов Э.В., Прангишвили И.В. Цифровые автоматы с настраиваемой структурой. - М.: Энергия, 1974, 240.
- [39] Евреинов Э.В., Хорошевский В.Г. Однородные вычислительные системы. - Новосибирск: Наука, 1978.
- [40] Евреинов Э.В. Однородные вычислительные системы, структуры и среды. - М.: Радио и связь, 1981, 208.
- [41] Елейко В.И., Елейко Я.И., Шмойлов В.И. и др. ОВС без резервных процессорных элементов. - Львов: НТЦ "Интеграл", 1991, 34.
- [42] Елейко Я.И., Самчинский А.А., Шмойлов В.И. Построение отказоустойчивой однородной вычислительной среды. - Львов: НТЦ "Интеграл", 1991, 67.
- [43] Исаенко В.А., Кузьо М.Н., Шмойлов В.И. и др. Высокопроизводительный комплекс анализа изображений с процессором на однородной вычислительной

- среде. - Львов: НТЦ "Интеграл", 1991, 60.
- [44] Каляев А.В. Теория цифровых интегрирующих машин и структур. - М.: Сов. Радио, 1970.
- [45] Каляев А.В. Однородные коммутационные регистровые структуры. - М.: Сов. Радио, 1978, 336.
- [46] Каляев А.В. Многопроцессорные системы с программируемой архитектурой. - М.: Радио и связь, 1984, 240.
- [47] Каляев А.В., Станишевский О.Б. Принципы построения программно-аппаратных средств супермакрокомпьютеров // Сб. «Информатика». Сер. Автоматизация проектирования. - Москва, 1990, Вып.2, 13-21.
- [48] Каляев А.В., Станишевский О.Б., Фрадкин Б.Г. Макропроцессорный комплект СБИС // Сборник докладов I Всесоюзной конференции «Однородные вычислительные среды и систолические структуры». - Львов, 1990, Т.2, 33-46.
- [49] Каляев А.В., Левин И.И. Многопроцессорные системы с перестраиваемой архитектурой: концепции развития и применения - Наука – производству, 1999, № 11, 11-19.
- [50] Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. - М.: Янус-К, 2003, 380.
- [51] Каляев А.В., Каляев И.А., Левин И.И. Многопроцессорные вычислительные системы с программируемой архитектурой на основе ПЛИС - Вестник ЮНЦ РАН, 2004, 24-33.
- [52] Каляев В.А., Левин И.И., Фомин С.Ю. Об эффективности моделирования на многопроцессорной системе процессов распространения цунами // Сб. «Многопроцессорные вычислительные структуры». - Таганрог, 1987, Вып. 9, 14-18.
- [53] Каляев В.А., Левин И.И., Фомин С.Ю. Многопроцессорная система для оперативного моделирования гидрофизических процессов // Сб. «Программные и аппаратные средства машинного моделирования». - Москва, 1988, 70-77.
- [54] Каляев В.А., Левин И.И., Фомин С.Ю. Об оценке эффективности решения задач математической физики на многопроцессорных системах - Электронное моделирование, 1989, №6, 11-15.
- [55] Каляев И.А., Левин И.И. Высокопроизводительные модульно-наращиваемые многопроцессорные системы на основе реконфигурируемой элементной базы - Вычислительные методы и программирование. Изд-во Московского

Университета, 2007, Т.8, №1, 181-190.

- [56] Камша Л.С., Камша В.П., Шмойлов В.И. и др. Суперкристалл ОВС с анализом команд. - Львов: НТЦ "Интеграл", 1990, 63.
- [57] Камша В.П., Кузьо М.Н., Шмойлов В.И. и др. ОВС на пластине. - Львов: НТЦ "Интеграл", 1990, 63.
- [58] Кокурин С.Л., Кузьо М.Н., Шмойлов В.И. Разработка однородной вычислительной среды на пластине. - Львов: НТЦ "Интеграл", 1991, 26.
- [59] Корнеев В.В. Архитектура вычислительных систем с программируемой архитектурой. - Новосибирск: Наука, 1983, 116.
- [60] Корнеев В.В. Вычислительные системы - М.: «Гемос АРВ», 2004, 511.
- [61] Котов В.Е. Теория параллельного программирования. Прикладные аспекты - Кибернетика, 1974, № 1, 1-16.
- [62] Кузьо М.Н., Самчинский А.А., Шмойлов В.И. и др. Комплекс обработки изображения на ОВС для контроля СБИС. - Львов: НТЦ "Интеграл", 1990, 47.
- [63] Кузьо М.Н., Самчинский А.А., Федорцов В.Й., Шмойлов В.И. Однородная вычислительная среда для обработки изображений. - Львов: НТЦ "Интеграл", 1991, 62.
- [64] Кузьо М.М., Русин Б.П., Шмойлов В.И. Однорідні середовища - елементна база високопродуктивних обчислювальних систем. - Управляющие системы и машины. - Киев, 2000, №4, 52-62.
- [65] Кузьо М.М., Русын Б.П., Шмойлов В.И. Пульсирующие информационные решётки. // Оптико-електронні інформаційно-енергетичні технології. - Вінниця, 2001, №1, 51-78.
- [66] Кузьо М.М., Русын Б.П., Шмойлов В.И. Однородные вычислительные среды // Оптико-електронні інформаційно-енергетичні технології. - Вінниця, 2001, №2, 19-37.
- [67] Кутепов В.П., Кораблин Ю.П. Язык граф-схем параллельных алгоритмов // Программирование, 1978, № 1, 1-16.
- [68] Кун С. Матричные процессоры на СБИС / Пер. с англ. - М.: Мир, 1991, 672.
- [69] Кухарев Г.А., Шмерко В.П., Зайцева Е.Н. Алгоритмы и систолические процессоры для обработки многозначных данных. - Минск: Наука и техника, 1990, 300.
- [70] Левин И.И. Организация вычислительного процесса для физико-топологического моделирования транзисторных структур СБИС на многопроцессорной системе // Сб. «Математическое моделирование элементов и

- фрагментов БИС». - Рига, 1990, 51.
- [71] Левин И.И. Сопроцессор для структурно-процедурного решения задач математической физики структурно-процедурным методом распараллеливания // Анализ эффективности вычислительных систем / Под ред. Дуброва Я.А. - Львов: Препринт НТЦ «Интеграл», 1991, 14-21.
- [72] Левин И.И. Исследование и разработка проблемно-ориентированной многопроцессорной системы для структурно-процедурного решения задач математической физики. Дисс. на соискание ученой степени канд. техн. Наук. - Таганрог, 1992, 293.
- [73] Левин И.И., Пономарев И.М., Фрадкин Б.Г. Анализ эффективности структурно-процедурной организации вычислительного процесса при решении прикладных задач на МВС // Сборник аннотаций и научных статей. - М., 1995, 23-28.
- [74] Левин И.И., Пономарев И.М. Реализация быстрого преобразования Фурье на многопроцессорной системе со структурно-процедурной организацией вычислений // Изв. Вузов. Электромеханика, 1995, №4, 72-74.
- [75] Левин И.И., Пономарев И.М. Реализация алгоритма волновой трассировки на многопроцессорной системе со структурной организацией вычислений / НИИ многопроцессорных вычислительных систем при Таганрогском государственном радиотехническом университете. - Таганрог, 1995, 49с. Деп. в ВИНТИ 29.05.95, №1553-B95.
- [76] Левин И.И. Язык параллельного программирования высокого уровня для структурно-процедурной организации вычислений // Труды Всероссийской научной конференции. - М.: Изд-во МГУ, 2000, 108-112.
- [77] Левин И.И., Пономарев И.М. Методика организации высокоэффективных параллельных вычислений в многопроцессорных системах // Тезисы международной конференции «Искусственный интеллект - 2000», - Кацивели. Таганрог: Изд-во ТРТУ, 2000, 142-144.
- [78] Левин И.И., Пономарев И.М., Шматок А.В. Методы преобразования параллельных программ под структуру вычислительной системы // Искусственный интеллект. - Донецк: Наука і освіта, 2003, №3, 227-231.
- [79] Левин И.И., Пономарев И.М. Структурно-процедурная реализация задачи трассировки // Искусственный интеллект. - Донецк: Наука і освіта, 2003, №3, 121-129.
- [80] Левин И.И. Методы и программно-аппаратные средства параллельных

- структурно-процедурных вычислений. Дисс. на соискание ученой степени докт. техн. Наук. - Таганрог, 2004, 363.
- [81] Левин И.И. Семерников Е.А. Устойчивость конвейерных рекурсивных фильтров - Вестник Южного научного центра Российской академии наук, 2005, Т.1, Вып. 2, 28-40.
- [82] Левин И.И. Структурно-процедурная организация параллельных вычислений // Материалы Четвертой Международной научной молодежной школы «Высокопроизводительные вычислительные системы». - Таганрог: Изд-во ТТИ ЮФУ, 2007, 49-68.
- [83] Литвинюк А.А. Вычислитель на однородных средах. - Львов: НТЦ "Интеграл", 1988, 39.
- [84] Литвинюк А.А., Погрибная И.М., Рожанковский И.В. и др. Алгоритмы, структуры и библиотека микропрограммных модулей цифровых фильтров с дельта-модуляцией на ОВС. - Львов: НТЦ "Интеграл", 1991, 108.
- [85] Маланина Н.М., Шмойлов В.И. Микропрограммирование в мультikonвейерных вычислительных структурах. - Львов: НТЦ "Интеграл", 1989, 66.
- [86] Майерс Г. Архитектура современных ЭВМ. Кн.1. - М.:Мир,1985, 364.
- [87] Норенков И.П., Маничев В.Б. Системы автоматизированного проектирования электронной и вычислительной аппаратуры / Учеб. пособие для вузов. - М.: Высшая школа, 1983, 272.
- [88] Палагин А.В. Опанасенко В.Н. Реконфигурируемые вычислительные системы. - Киев: Просвіта, 2006, 288.
- [89] Погрибной В.А., Бобало С.И., Кальмук Ю.С., и др. Ориентированные на ОВС методы корреляционного анализа со знаковой дельта-модуляцией. - Львов: НТЦ "Интеграл", 1990, 69.
- [90] Погрибной В.А., Литвинюк А.А., Рожанковский И.В. и др. Ориентированные на ОВС методы нерекурсивной фильтрации с дельта-модуляцией. - Львов: НТЦ "Интеграл", 1990, 76.
- [91] Погрибной В.А., Тимченко А.В., Рожанковский И.В. и др. Цифровая фильтрация сигналов с дельта-модуляцией на ОВС. - Львов: НТЦ "Интеграл", 1991, 65.
- [92] Подрубный О.В., Самчинский А.А., Шмойлов В.И. Построение отказоустойчивой элементной базы ОВС. - Львов: НТЦ "Интеграл", 1990, 36.
- [93] Поспелов Д.А. Арифметические основы вычислительных машин дискретного

- действия. - М.: Высшая школа, 1970, 308.
- [94] Поспелов Д.А. Введение в теорию вычислительных систем. - М.: Сов. Радио, 1972, 280.
 - [95] Прангишвили И.В., Абрамова Н.А., Бабичева Б.В., Игнатущенко В.В. Микроэлектроника и однородные структуры для построения логических и вычислительных устройств. - М.: Наука, 1967, 228.
 - [96] Прангишвили И.В. и др. Однородные микроэлектронные ассоциативные процессоры. - М.: Сов.радио, 1973, 280.
 - [97] Прангишвили И.В., Тодуа Д.А., Абрамова Н.А. и др. ЭВМ ПС-300. - Приборы и системы управления, 1978, №10.
 - [98] Применение цифровой обработки сигналов / Под ред. Э. Оппенгейма. - М.: Мир, 1980, 552.
 - [99] Программирование на параллельных вычислительных системах / Под ред. Р. Бэба. - М.: 1991, 376.
 - [100] Рабинер Л., Гоулд Б. Теория и применение цифровой обработки сигналов. - М.: Мир, 1978, 848.
 - [101] Распараллеливание обработки информации / Под ред. Грицика. - Львов, 1985, Т.2.
 - [102] Ривкин М.Н. Векторные операции для моделирования процедур преобразования данных - Программирование, 1991, №3.
 - [103] Русын Б.П., Кузьо М.Н., Шмойлов В.И. Реконфигурируемые высокопроизводительные системы на однородных средах // Автоматика и вычислительная техника. - Рига, 2000, №3, 72-81.
 - [104] Русин Б.П., Кузьо М.Н., Шмойлов В.И. Пульсирующие информационные решётки - новое поколение однородных вычислительных сред // Автоматика и вычислительная техника. - Рига, 2002, №1, 60-71.
 - [105] Самарский А.А. Введение в численные методы. - М.: Наука, 1987, 280.
 - [106] Самчинский А.А., Седов В.С., Шмойлов В.И. Архитектура однородной вычислительной среды. - Львов: НТЦ "Интеграл", 1991, 228.
 - [107] Сверхбольшие интегральные схемы и современная обработка сигналов // Под ред. С.Гуна, Х.Уайтхауса, Т.Кайлата. - М.:Радио и связь, 1989, 472.
 - [108] Седов В.С., Соболев В.Н., Самчинский А.А. и др. Элементная база ОВС. - Львов: ИППММ АН УССР, 1989, 38.
 - [109] Седов В.С. Матрица однородных процессоров. - Львов: НТЦ «Интеграл», 1991, 48.

- [110] Семерников Е.А., Доронченко Ю.И., Трунов И.Л. Конвейерный макропроцессор ЦОС для многопроцессорных вычислительных систем со структурно-процедурной организацией вычислений // Искусственный интеллект. - Донецк: Наука і освіта, 2004, Т.3, 157-163.
- [111] Семерников Е.А., Доронченко Ю.И. Конвейерный макропроцессор цифровой обработки сигналов со структурно-процедурной организацией вычислений // Вестник компьютерных и информационных технологий. - М: Машиностроение, 2005, №6, 49-55.
- [112] Семерников Е.А. К вопросу устойчивости конвейерных рекурсивных фильтров // Искусственный интеллект. - Донецк: Наука і освіта, 2005, Т.4, 713-723.
- [113] Семерников Е.А., Доронченко Ю.И., Коваленко В.Б. и др. Формирование адресных последовательностей для конвейерных вычислителей БПФ // Искусственный интеллект. - Донецк: Наука і освіта, 2006, Т.3, 68-78.
- [114] Семерников Е.А., Семерникова Е.Е. Перестановки элементов последовательностей в быстрых алгоритмах цифровой обработки сигналов - Вестник Южного научного центра Российской академии наук: Изд-во ЮНЦ РАН, 2006, Т.2, № 4, 25-30.
- [115] Сир Ж.К. Метод потока операндов в многопроцессорных системах типа MIMD // Системы параллельной обработки / Под ред. Ивенса Д. - М.: Мир, 1985, 240-276.
- [116] Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - М.: Мир, 1990, 235.
- [117] Тимченко О.В. Різницеві методи цифрової фільтрації. - Львів: Фенікс, 1999, 388.
- [118] Фаронов В.В. Система программирования Delphi. - Санкт-Петербург: БХВ-Петербург, 2004, 912.
- [119] Фрумкин М.А. Систематические вычисления. - М.:Наука, 1990, 192.
- [120] Фути К., Судзуки Н. Языки программирования и схемотехника СБИС. - М.: Мир, 1988, 224.
- [121] Хоар Ч. Взаимодействующие последовательные процессы. - М.: Мир, 1989, 265.
- [122] Хокни Р., Джессхоуп К. Параллельные ЭВМ. Архитектура, программирование и алгоритмы / Пер.с англ. - М.: Радио и связь, 1986, 392.
- [123] Хуан К. Перспективные методы параллельной обработки и архитектура суперЭВМ - ТИИЭР, 1987, № 10, 4-41.

- [124] Шмойлов В.И. Микропрограммирование в мультиконвейерной вычислительной структуре / В кн. Вычислительные системы, структуры и среды. - Киев: Наукова думка, 1986, 116-164.
- [125] Шмойлов В.И. Организация вычислительного процесса в мультиконвейерной вычислительной структуре. - Львов: НТЦ «Интеграл», 1991, 93.
- [126] Шмойлов В.И. Архитектура однородных вычислительных сред. - Львов: НТЦ «Интеграл», 1993, 289.
- [127] Шмойлов В.И., Слобода М.З. Расходящиеся непрерывные дроби. - Львов: Меркатор, 1999, 820.
- [128] Шмойлов В.И., Русин Б.П., Кузьо М.Н., Заяц И.А. Пульсирующие информационные решётки. - Львов: Меркатор, 1999, 66.
- [129] Шмойлов В.И., Русин Б.П., Кузьо М.Н., Капший О.В. Проектирование пульсирующих информационных решёток. - Львов: Меркатор, 2000, 101.
- [130] Шмойлов В.И., Русин Б.П., Кузьо М.Н. Ячейка пульсирующих информационных решёток. - Львов: Меркатор, 2001, 34.
- [131] Шмойлов В.И., Русин Б.П., Кузьо М.Н. Однородные вычислительные среды и пульсиры. - Львов: Меркатор, 2001, 62.
- [132] Шмойлов В.И., Русин Б.П., Таянов В.А. Обработка изображений в однородных вычислительных средах. - Львов: Меркатор, 2002, 70.
- [133] Шмойлов В.И., Адамацкий А.И., Кузьо М.Н. и др. Пульсирующие информационные решетки. - Львов: Меркатор, 2004, 302.
- [134] Шмойлов В.И., Адамацкий А.И., Кузьо М.Н. и др. Пульсирующие информационные решетки. - Львов: Меркатор, 2004, 302.
- [135] Шмойлов В.И., Русин Б.П., Кузьо М.Н. Пульсирующие информационные решетки – новое поколение однородных вычислительных сред // Управляющие системы и машины. - Киев, 2004, №2, 23-38.
- [136] Шмойлов В.И. Пульсирующие информационные решетки и суперкомпьютеры класса А. - Львов: Меркатор, 2005, 902.
- [137] Штейнберг Б.Я. Распараллеливание рекуррентных циклов с условными операторами - Автоматика и телематика, 1995, №6, 176-184.
- [138] Шуп Т.Е. Прикладные численные методы в физики и технике. - М.: Высшая школа, 1990, 255.
- [139] Элементы параллельного программирования /Под ред. Котова В.Я. - М.: Радио и связь, 1983, 240.
- [140] Bolotski M. Abacus: A Reconfigurable Bit Parallel Architectures. Ph.Dd.Thesis -

- Massachusetts Institute of Technologies, 1996, 126.
- [141] Chamberlin D.D. Parallel Implementation of a'LAU Ph.D. Thesis Report 19, 1971.
 - [142] Ercal and Lee H.C. "Time-efficient maze routing algorithms on reconfigurable mesh architectures", J. of Parallel and Distributed Computing, 44(2):133-140, August 1997.
 - [143] Kaliaev A.V., Kaliaev I.A., Levin I.I. The Base Module of Multiprocessor System with Structural- Procedural Organization of Computing. Parallel Computing Technologies // Proceedings of 4-th International Conference, PaCT-97. - Yaroslavl, Russia, 1997, 394-396.
 - [144] Kuch D.J. ILLIACIV Software and Application Programming // IEEE Jrans. Comput., 1968, v. 6-17, N 8, 758-770.
 - [145] Kung H.T. and Leiserson C.E. Systolic arrays (for VLSI). In Sparse Matriz Symposium, SIAM, 1978, 256-282.
 - [146] Tahsin M. Kurç, Cevdet Aykanat, Fikret Erçal: Parallelization of Lee's Routing Algorithm on a Hypercube Multicomputer. EDMCC 1991: 244-253.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Аддитивный процессор 202
Алгоритм свертки 221
Адресный процессор 233,234
Арифметико-логическое устройство 48

Базовый модуль 161, 164, 182
-- с макрообъектной архитектурой 187
Базовые матричные кристаллы 170
Базовые подграфы 125, 136
Блок граничных условий 216
Блок операндов 247
Библиотека модулей 78
Блок генерации констант 67

Временная диаграмма 43,228
Волновая трассировка 153
Версия модуля 72
Вершина-массив 150

Граф алгоритма 14, 20
– векторизованный 156
– вычислительного процесса 14
– микропрограммного модуля 59
– регулярный 131
Граф-схемы 297

Динамический коммутатор 210
Длина конвейера 23

Задача Пуассона 134
Задача фильтрации 218

Интерфейс программный 252

Информационный граф 14
-- нерегулярный 141
-- быстрого преобразования Фурье 38
Информационный кадр 124

Кадр 118, 119
Каналы ячейки 100
Классификация Флинна 13
Клеточные автоматы 101
Код 68
- прямой 69
- обратный 69
- дополнительный 69
-- модифицированный 69

Коммутаторы 110
– второго уровня 110
– системный 111
- смежные 102
- сопряженные 102

Коммутационная переменная 276

Конвейерная обработка 22,24

Константы 65

Конструкция пульсиров 102

Контроллер распределенной памяти 165

Кортеж изоморфных графов 120,132

Кристаллы FPGA 173

Линейная дельта-модуляция 83
– фильтрация 82

Локальные связи 103

Макрограф алгоритма 34

Макровычислительные устройства 35

Макрообъект 206, 208	Организация вычислительного процесса 59
- первого вида 203	Ортогональная коммутационная среда 167
- унифицированный 239	
- цифровой обработки сигналов 221	
Макрооперация 107	
- виртуальная 299	Память распределенная 168
Макропамять векторная 263	- регистровая 50
Макрообращения 259	Параллельная обработка 51
Макропроцессор 18, 107	Паспорт модуля 60
Математическая модель ячейки 54	Перестраиваемая архитектура 40
Мемориальная переменная 275	Пиковая производительность 13
Метод верхней релаксации 234	Пластина 98
Метод уравнивания осей 59	Потоковые задачи 19
Микрокоманда ячейки 51	Потоковая обработка 21
Микрооперации ячейки 32,51	Преобразования в кадровую форму 121
Микропрограммный модуль 57	Преобразование Фурье 221
-- первой группы 57	-- прямое 222
-- второй группы 58	-- обратное 223
Многопроцессорные системы 15	Программируемые матрицы 172
Мультиконвейерные структуры 35	Программирование в пространстве 250
-- двухкаскадная 112	Псевдокадр 164
Мультимакроконвейер 114	
Мультипликативный процессор 202	Реальная производительность 13
	Регистр динамической памяти 53
Однобитовые процессоры 48	Регистровые каналы 53
Однородная вычислительная среда 32	- переменная 276
- запоминающая среда 50	Реконфигурируемая структура 27
--- с матричной коммутацией 94	-- мультиконвейерная 29
Оконная обработка 82	
Оператор обращения 264	Синтез микропрограммных модулей 53
-- внутренний 263	Система команд ячейки 49
Операции арифметические 78	Системы на кристалле 171
- логические 78	- с программируемой архитектурой, 181
- крупные 80	Систолическая обработка 18
Описание кадров 124	Служебный канал 101
Оптоэлектронные коммутаторы 103	Список TOP-500 6

Способ обработки информации 19

--- конвейерный 22

--- параллельный 21

--- структурный 29

--- структурно-процедурный 37, 40

Среда разработки программ 294

Структурное программирование 31

Структура информационных потоков 262

- коммутационная 276

Ступень конвейера 23, 25

Суперкомпьютер 5

Типы параллельной обработки 294

Транспьютер 46

Узел записи 136

Узловой процессор 155, 156

Форматы операндов 62

Функциональная схема ячейки 48

Функционально-регулярные графы 123, 138

Цифровые фильтры 82

Элементарный процессор 202, 299

Элементы языка 258

Язык программирования Argus 256

-- COLAMO 273

Ярусно-параллельная форма 128

Ячейки среды 48

-- связанные 102

-- смежные 102

-- соседние 102

-- сопряженные 102