Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронно-вычислительных средств

М. В. Качинский, А. А. Петровский

СТРУКТУРНЫЙ СИНТЕЗ ПАРАЛЛЕЛЬНО-ПОТОЧНЫХ БПФ-ПРОЦЕССОРОВ РЕАЛЬНОГО ВРЕМЕНИ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-40 02 02 «Электронные вычислительные средства»

Минск БГУИР 2014

УДК 004.717(076)
ББК 32.973.26-018.2я73
К30

Рецензенты:
кафедра информационных систем и технологий
Белорусского национального технического университета
(протокол №10 от 03.06.2013 г.);

заведующий кафедрой информатики учреждения образования
«Минский государственный высший радиотехнический колледж»,
кандидат технических наук, доцент Ю. А. Скудняков

Качинский, М. В.

К30 Структурный синтез параллельно-поточных БПФ-процессоров реального времени : учеб.-метод. пособие / М. В. Качинский, А. А. Петровский. – Минск : БГУИР, 2014. – 58 с. : ил.

ISBN 978-985-543-018-7.

Рассматриваются вопросы структурного синтеза параллельно-поточных БПФ-процессоров для систем многоканальной обработки процессов в реальном времени.

Предназначено для студентов, изучающих дисциплину «Проектирование электронных вычислительных средств с динамически реконфигурируемой архитектурой».

УДК 004.717(076)
ББК 32.973.26-018.2я73

ISBN 978-985-543-018-7

Содержание

ВВЕДЕНИЕ

В настоящее время основными проблемами проектирования вычислительных средств на FPGA (Field-Programmable Gate Arrays) являются следующие: синтез специальных блоков аппаратных средств – IP-ядер (IP cores – Intellectual Property cores), программное обеспечение для синтеза многопроцессорных архитектур, проектирование на системном уровне для автоматической реализации заданного приложения на гетерогенной архитектуре FPGA.

Первое появление программируемых логических интегральных схем (ПЛИС) характеризовалось относительной нехваткой (по сегодняшним меркам) логических элементов на одном устройстве. Это означало, что только небольшое число важных функциональных блоков может быть реализовано на кристалле. Однако природа их программируемости позволила сделать их идеальными основными устройствами в составе многокристальных ASIC (Application-Specific Integrated Circuit) платформ обработки данных. Низкий уровень сложности означал, что одна архитектура может быть реализована с достаточно высокой производительностью и функциональностью при помощи большого числа логических элементов.

С введением новых поколений устройств FPGA, таких, как Xilinx XC4000, Virtex и Altera Stratix, плотность уровня реализуемой логики продолжала увеличиваться по закону Мура. Это в сочетании с потенциально исключительно высоким уровнем параллелизма означает, что FPGA постепенно превратилась из простой логики в сложный компонент, способный реализовать широкий класс алгоритмов цифровой обработки сигналов (ЦОС). Такая эволюция способствовала появлению высокопроизводительных IP-ядер алгоритмов ЦОС. Данная тенденция сохранялась и при введении высокопроизводительных компонентов, таких, как умножители и сумматоры. Увеличение сложности разработки на данных устройствах привело к появлению уровня регистровых передач (RTL), объединяющего средства проектирования, такие, как Xilinx ISE, и инструментарий для синтеза структур алгоритмов ЦОС в программируемых FPGA.

С появлением Virtex TM-2 Pro и Stratix FPGA традиционные программируемые логические устройства были пополнены встраиваемыми микропроцессорами, таким, как PowerPC в Virtex TM-2 Pro, и высокоскоростными последовательными интерфейсами, такими, как Xilinx Rocket I/O. С этих пор FPGA стали одним из основных компонентов системы ЦОС, способных вести независимую обработку данных в соответствии с алгоритмами ЦОС, обеспечивая высокую производительность решения задачи и гибкость структурной реализации. Таким образом, FPGA стали центром системы ЦОС, а не только дополнительным ускорителем алгоритмов ЦОС.

Существенный скачок в производительности систем ЦОС потребовал значительных усовершенствований средств проектирования на базе FPGA. Высокая производительность и функциональные возможности FPGA, соизмеримые с такими решениями, как SoC (System-on-a-Chip) в ASIC, обусловили разработку новых концепций построения программного обеспечения, организации

связи функциональных модулей в составе системы и системной интеграции нескольких FPGA в гетерогенную систему, объединенную сетью высокоскоростного интерфейса.

В данном учебно-методическом пособии рассматривается подход к проектированию систем мультимедиа для решения задач обработки данных в реальном времени, приводятся примеры использования данного подхода при проектировании встраиваемых процессоров ЦОС на основе параллельно-поточной архитектуры.

1. БЫСТРОЕ ПРОТОТИПИРОВАНИЕ СИСТЕМ ЦОС

1.1. Методология проектирования систем ЦОС

Эволюция вычислительных архитектур, сопровождаемая неустанным ростом в технологии интеграции кремния, ведет к развитию ряда новых технологических реализаций систем ЦОС. Такие платформы могут объединять на кристалле как один или несколько микропроцессоров, так и гетерогенные системы на кристалле SoC, а также ПЛИС. Развитие вычислительных архитектур для SoC уже в течение ряда лет опережает возможности проектировщиков в реализации систем ЦОС на SoC. Данный факт называют разрывом проектирования и производства. Основной причиной такого разрыва является ограничивающий фактор в промышленности, стремящийся к реализации массовости SoC. Это вынуждает производителей автоматизированных сред проектирования значительно пересмотреть концепции проектирования систем на кристалле [9].

Современные реализации платформ ЦОС состоят из смеси гетерогенных архитектур для обработки, включая блок микроконтроллера, например с архитектурой фон Неймана, высокопроизводительные блоки, такие, как микропроцессоры с VLIW архитектурой, и специализированные аппаратные средства для эффективного выполнения задач. Эволюция современных ПЛИС показывает, что они также могут являться потенциальными кандидатами в реализации платформ как самостоятельные системы либо в качестве дополнения к существующим программным платформам. Встраиваемые платформы на основе ПЛИС предлагают совершенно новые и более совершенные подходы для решения задач, которые разработчик может применить при построении архитектуры системы обработки.

Широкий диапазон архитектур и методов обработки, реализуемых в системах ЦОС, делает внедрение таких систем на текущем уровне абстракции проектирования трудной задачей. Следовательно, использование когерентных структур, позволяющих быстро переводить поведенческое описание системы непосредственно на аппаратуру, является критичным. Концепция модельно-ориентированного проектирования встраиваемых систем в настоящее время наиболее популярна. Эта концепция, являющаяся обобщением многочисленных конкретных типов методов и средств проектирования, стимулирует использование семантически четко определенных языков моделирования для построения поведенческих алгоритмов. Семантика каждого из этих типов моделей

затем используется для оперативного выбора варианта реализации системы. Таким образом, существует два важных аспекта в проектировании: особенности модели вычислений, определенных в спецификации алгоритма, и методология, используемая для уточнения этих моделей, приводящая к требуемой реализации.

Традиционный подход к проектированию систем ЦОС вовлекает многочисленные группы инженеров, включая группу разработки алгоритма, группу программистов, группу разработки аппаратной части и группу реализации проекта. Когда разработка алгоритма завершена, группа программистов реализует алгоритм в среде моделирования и затем определяет требования к аппаратуре. Аппаратная группа проекта выполняет аппаратную реализацию. Наконец, группа реализации проекта объединяет аппаратные и программные средства в единую систему. Этот традиционный процесс разработки занимает много времени, потому что проектировщики алгоритма часто не имеют доступа к аппаратным средствам, которые фактически уже созданы. Процесс быстрого прототипирования объединяет фазы проектирования, устраняя потенциально узкие места, разрешая инженерам видеть результаты и быстро повторять решения без использования дорогих аппаратных средств. Необходимым условием проведения быстрого прототипирования является возможность автоматической генерации исполняемого кода для целевого процессора.

Методология проектирования систем ЦОС включает три основные категории [9]:

- программно-аппаратное проектирование;
- функционально-архитектурное проектирование;
- проектирование, ориентированное на соответствующую вычислительную платформу.

При проектировании встраиваемых систем ЦОС на базе универсальных процессоров цифровой обработки сигналов (*программно-аппаратное проектирование*) важно понимать имеющий место дуализм: аппаратные средства – программное обеспечение. Суть этого дуализма сводится к следующему: любой процесс, реализуемый программным путем, может быть преобразован в эквивалентный процесс, реализуемый аппаратными средствами, и наоборот. Ни аппаратные средства, ни программное обеспечение не существуют независимо, поэтому указанное преобразование никогда не может быть завершено полностью. Всегда существуют программы управления аппаратными средствами и аппаратура, при помощи которой реализуются программы. Это утверждение основано на опыте, и его принимают как аксиому. Можно привести ряд разнообразных характеристик аппаратных и программных средств. Изменения алгоритма поразному сказывается на аппаратуре и программном обеспечении. Обычно программная реализация алгоритма позволяет вносить значительные изменения, в то время как возможность изменения существующего аппаратного представления алгоритма практически отсутствует. Зачастую разнится и способ выполнения алгоритма при этих подходах: обработка посредством аппаратных средств является непрерывным параллельным одношаговым процессом, программная

же обработка, как правило, состоит из ряда последовательных дискретных шагов. Взаимосвязь аппаратных средств и программного обеспечения первоначально основывалась на соображениях экономического характера. В этой связи определяющим было правило: аппаратные средства обеспечивают быстродействие, программное обеспечение – дешевизну, вследствие чего доминировали простые, преимущественно программные системы с относительно низким быстродействием. В процессе развития за последние годы указанные соображения претерпели существенные изменения. Динамическая связь (экономического характера) между аппаратными и программными средствами явилась следствием быстрых технологических изменений, произошедших в рассматриваемой области. Объединение аппаратных и программных средств привело к изменению, как самих систем ЦОС, так и инструментальных систем для их разработки и отладки.

Функционально-архитектурное проектирование подразумевает постепенное нисходящее проектирование [9]:

I. Задание спецификации на систему ЦОС, в которую кроме функциональных требований включаются, например, требования реального масштаба времени (частотный диапазон), максимальная потребляемая мощность, ограничения максимальной стоимости и т. д.

II. Моделирование и тестирование:

1. Выбор алгоритмического обеспечения, удовлетворяющего заданной спецификации.
2. Моделирование, например с использованием MATLAB.
3. Верификация алгоритмов. Если требования спецификации не выполняются, то производится корректировка алгоритмического обеспечения или в крайнем случае спецификации.

III. Структурный синтез:

1. Генерация архитектур процессора:

а) выбор архитектурного решения процессора, соответствующего алгоритмическому обеспечению;

б) разработка модели выбранного архитектурного решения процессора;

в) оценка эффективности архитектурного решения.

2. Отображение алгоритмов на выбранную в п. III, подп.1 архитектуру процессора:

а) отображение требований спецификации к модели архитектурного решения процессора, полученной в п. III, подп. 1 б;

б) установка синхронизации вычислительных процессов (составление расписания) по результатам оценки эффективности архитектурного решения процессора в п. III, подп. 1 в;

в) моделирование и верификация. Если требования верификации не удовлетворяются, то уточняется расписание процессов (переход к п. III, подп. 2 б) или к п. III, подп. 1.

IV. Аппаратный синтез:

1. Задание модели синхронизации вычислительного процесса (управляющего автомата), на основании которой определяется модель программного обеспечения и генерируются требуемые программы, а также составляется модель аппаратного обеспечения и синтезируется соответствующая аппаратура.
2. Моделирование полученного процессора. Если требования спецификации не удовлетворяются, то корректируется модель аппаратного обеспечения или уточняется спецификация.

V. Конец.

Для современной вычислительной техники характерно развитие систем ЦОС, оптимальных для той или иной специальной области применения. По мере совершенствования элементов микроэлектроники разнообразие этих систем растет и их специализация во многих случаях сужается, а масштабы применения расширяются, и они становятся для соответствующих областей обработки универсальными.

С учетом этого схема функционально-архитектурного проектирования может быть модифицирована: заранее выбирается фиксированное архитектурное решение процессора и его модель, например, на базе аппаратных сетей Петри, и для требуемого задания из некоторого множества заданий находится расписание вычислительных процессов на модели архитектуры процессора, удовлетворяющее требованиям спецификации. Таким образом, определяется модель синхронизации вычислительного процесса (управляющего автомата). Такое решение имеет ряд достоинств: нет необходимости для каждого задания искать новое архитектурное решение процессора, что сокращает время проектирования; есть возможность создать, например, виртуальную систему ЦОС, адаптируемую под конкретного пользователя.

Проектирование, ориентированное на соответствующую вычислительную платформу, представляет собой классический подход к разработке систем ЦОС, в соответствии с которым алгоритм отображается на относительно фиксированную структуру вычислительной платформы (хотя она может быть адаптирована в ряде конкретных случаев). Вычислительная платформа считается «гибкой» интегральной схемой, где настройка для конкретного применения достигается за счет «программирования» одного или нескольких компонентов кристалла.

1.2. Основные средства проектирования встраиваемых систем ЦОС

Ключевым аспектом всех подходов к проектированию систем ЦОС (в частности, ориентированных на соответствующую вычислительную платформу и функционально-архитектурное проектирование) является использование определенного языка для описания модели реализации алгоритма.

Ряд инструментальных систем, таких, как Artisan (Arti-San – Software Tools Ltd 2004) и Rhapsody от I-Logix, могут синтезировать встроенный код. MATLAB – интегрированная среда реального времени предлагает возможности

генерации кода для ряда целевых задач непосредственно из графического описания системы Simulink. Наличие средств структурного синтеза для ПЛИС относит MATLAB к перспективным системам проектирования гетерогенной системы. Альтернативой выступают системы Rhapsody и Statemate от I-Logix, которые дают возможность по генерации кода из различных предметно-ориентированных языков описания систем высокого уровня. Хотя каждая из них предлагает определенные решения по выполнению быстрой реализации, возможность построения полной системы для ПЛИС отсутствует.

Графический подход разработки IP-ядер обеспечивает генерацию VHDL или Verilog HDL кодов из блок-схемы для аппаратного обеспечения. Полученный код может быть подан на инструмент синтеза аппаратных средств для реализации алгоритмов ЦОС в ПЛИС или ASIC. Данный подход по-прежнему требует, чтобы проектировщик был тесно связан с процессом проектирования, осуществлял контроль специфических аспектов проектирования и был в состоянии выполнить постобработку полученного проекта на ПЛИС. Кроме того, в распоряжении проектировщика имеются также стандартные библиотеки IP-ядер. Проектирование системы должно осуществляться с учетом знания аппаратной платформы для обеспечения эффективной реализации алгоритмов ЦОС.

Инструментальные средства такого типа имеются в популярных платформах для обработки сигналов, таких, как Simulink или MATLAB, которые позволяют выполнить реализацию высокоуровневого описания на ПЛИС путем генерации VHDL или Verilog HDL кода.

Основным средством архитектурного синтеза для генерации RTL уровня архитектуры IP-ядра является инструмент Synplify. Разрабатываемая модель алгоритма в Simulink постепенно улучшается с учетом архитектуры ПЛИС шаг за шагом:

1) отправной точкой является идеальная модель алгоритма, которая описывается имеющимися компонентами системы проектирования для формирования эквивалентной модели;

2) на этапе синтеза структуры алгоритма выполняется автоматическая оптимизация структуры средствами интегрированной среды, которая позволяет включить конвейеризацию, внести дополнительную синхронизацию и векторизацию при синтезе архитектуры;

3) в результате получается структура алгоритма, которую легко перенести на RTL уровень для реализации на устройстве с помощью традиционных языков синтеза RTL уровня.

Язык C широко используется для описания алгоритмов ЦОС из-за большого количества проверенных открытых исходных кодов реализации таких алгоритмов, которые существуют в приложениях. Проектирование на языке C дает значительный прирост производительности по сравнению с проектированием на традиционном Verilog или VHDL RTL за счет более высокого уровня абстракции. Язык C имеет свои ограничения в полном использовании архитектур

ПЛИС, т. к. LUT, сдвиговый регистр и другие компоненты ПЛИС должны приниматься во внимание при реализации высокопроизводительных проектов.

Хотя язык *C*/*C*++ является популярным выбором в качестве языка для синтеза, но пользователи чаще выбирают MATLAB из-за следующих факторов [9]. Во-первых, MATLAB обеспечивает более высокий уровень абстракции, и требуется меньше времени на разработку, чем на языке *C*. Во-вторых, MATLAB имеет четкое определение сигнала, блока обработки, богатый набор библиотечных функций, связанных с матричными операциями. Выявление параллелизма в MATLAB легче, чем в *C*, т. к. алгоритмы ЦОС в MATLAB выражаются в виде матричных операций, которые очень легко поддаются распараллеливанию.

2. ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ ИНТЕГРАЛЬНЫЕ СХЕМЫ С АРХИТЕКТУРОЙ FPGA

2.1. Введение в структуры FPGA

В качестве элементной базы для быстрого создания прототипов цифровых устройств идеально подходят программируемые логические интегральные схемы. Для таких устройств в англоязычной литературе используется термин Programmable Logic Device (PLD). Главной особенностью таких устройств является то, что арифметические и логические функции, реализуемые устройством, задаются путем программирования этого устройства.

Для создания прототипов цифровых устройств наиболее удобны ПЛИС с возможностью многократного программирования пользователем в конструктивно законченной системе. Такая возможность позволяет многократно использовать одну отладочную плату на базе соответствующего кристалла ПЛИС для прототипирования различных систем. Сложность прототипируемых систем ограничивается только логической емкостью кристалла ПЛИС, наличием требуемого числа выводов микросхемы для организации ввода/вывода информации и наличием достаточного количества дополнительных специализированных ресурсов в составе ПЛИС (умножителей, блоков памяти, высокоскоростных приемопередатчиков (трансиверов) и т. д.).

Первые ПЛИС появились в конце 60-х гг. прошлого века. Они имели небольшую логическую емкость (небольшое количество логических вентилей), и с их помощью можно было реализовать только простые цифровые устройства. Программирование этих ПЛИС осуществлялось однократно, и проект другой цифровой системы в таких устройствах не мог быть размещен.

В настоящее время основными классами ПЛИС являются CPLD (Complex Programmable Logic Device – сложное программируемое логическое устройство) и FPGA (Field Programmable Gate Array – программируемая пользователем вентильная матрица). Оба класса имеют возможность программирования в конструктивно законченной системе через интерфейс от компьютера с установленной инструментальной средой проектирования.

Основу CPLD составляют программируемые логические блоки, называемые макроячейками (macrocells). Каждая макроячейка обеспечивает выполнение логической и (или) регистровой функции. В состав CPLD входят матрицы элементов И либо ИЛИ. С помощью переключающей матрицы осуществляется коммутация входных и выходных сигналов макроячеек и подключение их к блокам ввода/вывода. Блоки ввода/вывода выполняют интерфейсные функции между внутренними логическими сигналами ПЛИС и сигналами на внешних выводах микросхемы. Информация для настройки макроячейки на конкретную логическую функцию и требуемых коммутаций входных и выходных сигналов макроячеек хранится в энергонезависимой памяти. Помимо архитектурных и технологических отличий CPLD по сравнению с FPGA имеют значительно меньшую логическую емкость (число эквивалентных логических вентилей в CPLD и FPGA может отличаться более чем на три порядка), поэтому для целей прототипирования наиболее целесообразно использовать ПЛИС с архитектурой FPGA.

В отличие от CPLD микросхемы FPGA не содержат матриц элементов И либо ИЛИ. Основу FPGA составляют вентильные матрицы – матрицы конфигурируемых логических блоков (CLB) или логических ячеек. На рис. 1 представлена упрощенная структура ПЛИС с архитектурой FPGA.

Матрица конфигурируемых логических блоков располагается в средней части кристалла. По периферии кристалла располагаются блоки ввода/вывода (IOB). Каждый IOB обеспечивает интерфейс внутренних сигналов ПЛИС с внешним выводом микросхемы. Между CLB располагаются вертикальные и горизонтальные линии связей и программируемые ключи (PS). Программируемый ключ обеспечивает соединение одной горизонтальной линии связи с одной вертикальной линией связи. Совокупность линий связи и программируемых ключей образуют трассировочные ресурсы FPGA (route). Трассировочные ресурсы позволяют соединять входы и выходы CLB и IOB между собой в требуемом порядке. Количество и архитектура CLB трассировочных ресурсов и IOB для различных FPGA сильно различаются. Это необходимо учитывать при выборе производителя FPGA и конкретных микросхем.

Для трассировки сигналов синхронизации используются специальные трассировочные ресурсы, обладающие значительно меньшей задержкой, чем трассировочные ресурсы общего назначения. Это позволяет иметь небольшие временные отклонения фронтов сигнала синхронизации в разных частях кристалла FPGA.

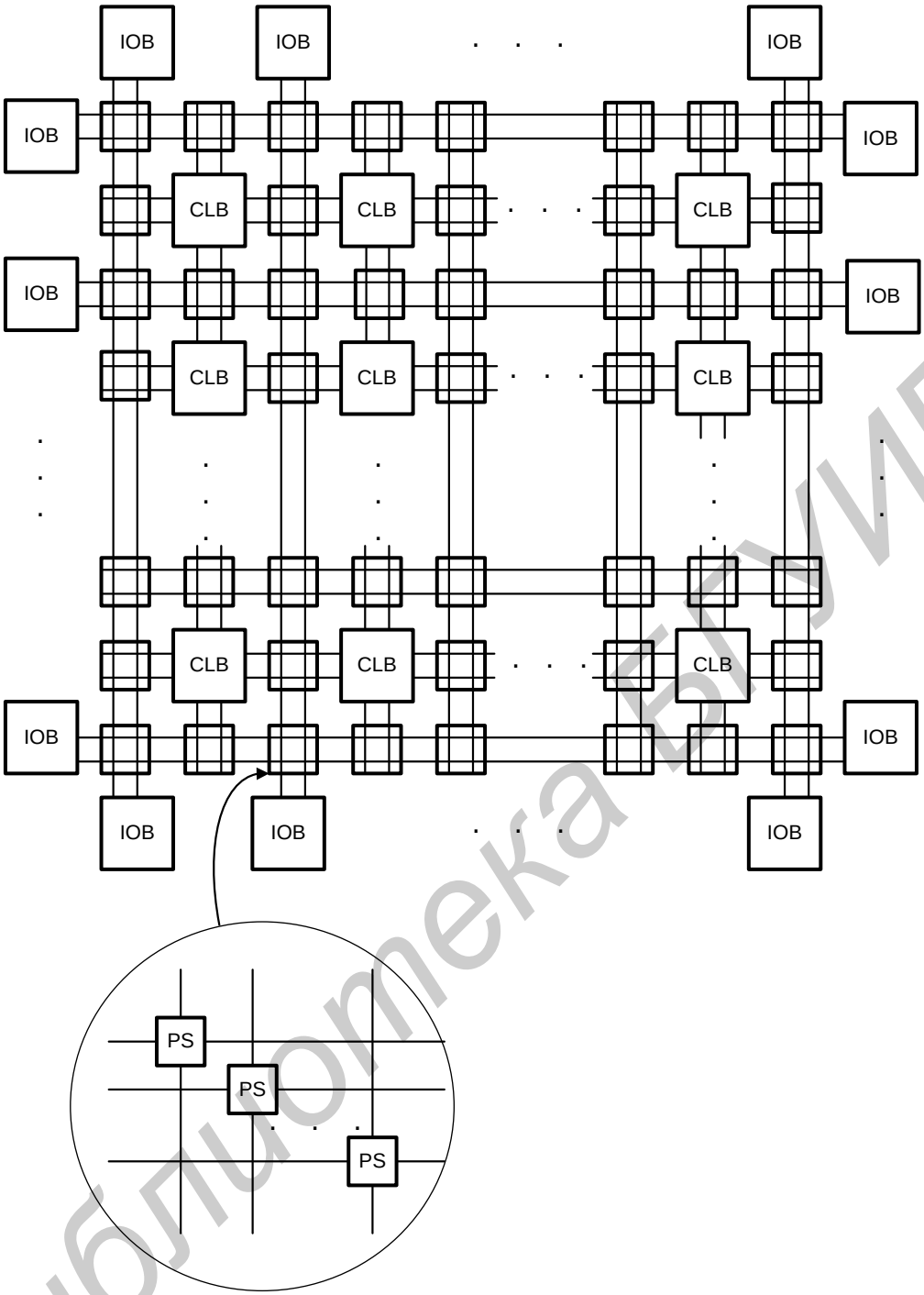

Рис. 1. Упрощенная структура ПЛИС с архитектурой FPGA

При создании больших проектов возможна ситуация, когда имеющиеся трассировочные ресурсы могут явиться «узким местом» разрабатываемой цифровой системы. В этом случае при анализе критических путей можно будет увидеть, что задержка на трассировочных ресурсах значительно превышает задержку в CLB. Кроме того, инструментальные средства проектирования могут использовать CLB не по прямому назначению для реализации логических функций, а для пропуска через них сигналов (route through) для доступа к еще свободным трассировочным ресурсам.

Настройка CLB на определенную логическую функцию и PS на требуемую коммутацию производится с помощью процедуры конфигурирования FPGA. Технологически конфигурирование FPGA может быть реализовано различными способами. В большинстве случаев конфигурационная последовательность бит загружается в специальную конфигурационную память. В настоящее время имеются следующие варианты конфигурирования FPGA [9]:

1. На базе статической конфигурационной памяти (SRAM-based FPGA). Такой вариант использует большинство FPGA. В этом случае конфигурационная память представляет собой массив триггеров-защелок (latches). Для SRAM-based FPGA при выключении электропитания содержимое конфигурационной памяти не сохраняется, поэтому такие FPGA должны конфигурироваться после включения электропитания перед началом работы из внешнего источника. Время конфигурирования зависит от кристалла ПЛИС и частоты, с которой происходит конфигурационный процесс, и составляет несколько миллисекунд. Конфигурирование может производиться под управлением самой FPGA из внешнего источника, например внешней флэш-памяти (Master mode), либо процессом конфигурирования может управлять внешнее устройство, например микропроцессор (Slave mode). Большинство FPGA также имеет для конфигурирования стандартный JTAG интерфейс. По технологии статической конфигурационной памяти выполнены FPGA фирмы Xilinx семейств Virtex и Spartan и фирмы Altera семейств Stratix и Cyclone. Есть также FPGA со статической конфигурационной памятью и внутренней флэш-памятью (например, семейства Spartan-3AN фирмы Xilinx и LatticeXP фирмы Lattice Semiconductors). В этом случае нет необходимости в наличии дополнительного кристалла внешней флэш-памяти, и отсутствует возможность несанкционированного копирования конфигурационной последовательности.

2. На базе конфигурационной флэш-памяти (flash-based FPGA). В этом случае конфигурационная последовательность хранится во внутренней флэш-памяти и отсутствует необходимость ее занесения в SRAM память. Такая технология обеспечивает меньшее энергопотребление и чувствительность к радиации. Этот вариант конфигурирования используется в FPGA фирмы Actel семейств Igloo и ProASIC3. В качестве достоинства этого варианта следует указать отсутствие возможности несанкционированного копирования конфигурационной последовательности.

3. Antifuse-based FPGA. Принцип работы antifuse состоит в следующем. До конфигурирования antifuse не проводит электрический ток, а при программировании antifuse начинает проводить электрический ток (принцип действия обратен принципу действия плавкого предохранителя). Такой принцип имеет семейство Axcelerator фирмы Actel. FPGA с такой технологией могут быть запрограммированы один раз и поэтому не могут быть использованы для многократного прототипирования различных цифровых систем.

Из рассмотренных вариантов современные SRAM-based FPGA имеют наибольшую логическую емкость, но требуют внешнюю энергонезависимую

память и имеют большое энергопотребление. Для защиты от несанкционированного копирования конфигурационной последовательности большинство современных SRAM-based FPGA имеют возможность шифрования этой последовательности.

Архитектура конфигурируемого логического блока, а также его название, отличаются для различных производителей ПЛИС и различных семейств FPGA. Наиболее часто CLB реализуется на основе просмотровых таблиц (Look Up Table – LUT). При этом в состав CLB может входить более одной LUT.

Обычно LUT строится на основе SRAM с организацией $2^k \times 1$ бит (k – число входов LUT) и набора мультиплексоров для выбора требуемого бита. SRAM хранит LUT – маску, заносимую при конфигурировании FPGA. LUT является функциональным генератором, позволяющим реализовать любую логическую функцию k переменных. Для реализации такой функции помимо SRAM емкостью 2^k бит требуется мультиплексор 2^k в 1. Этот мультиплексор обычно реализуется как пирамида мультиплексоров 2 в 1. В качестве примера на рис. 2 приведена типичная архитектура LUT4, которая содержит 16-битную SRAM и мультиплексор 16 в 1. Входами LUT являются *A*, *B*, *C* и *D*.

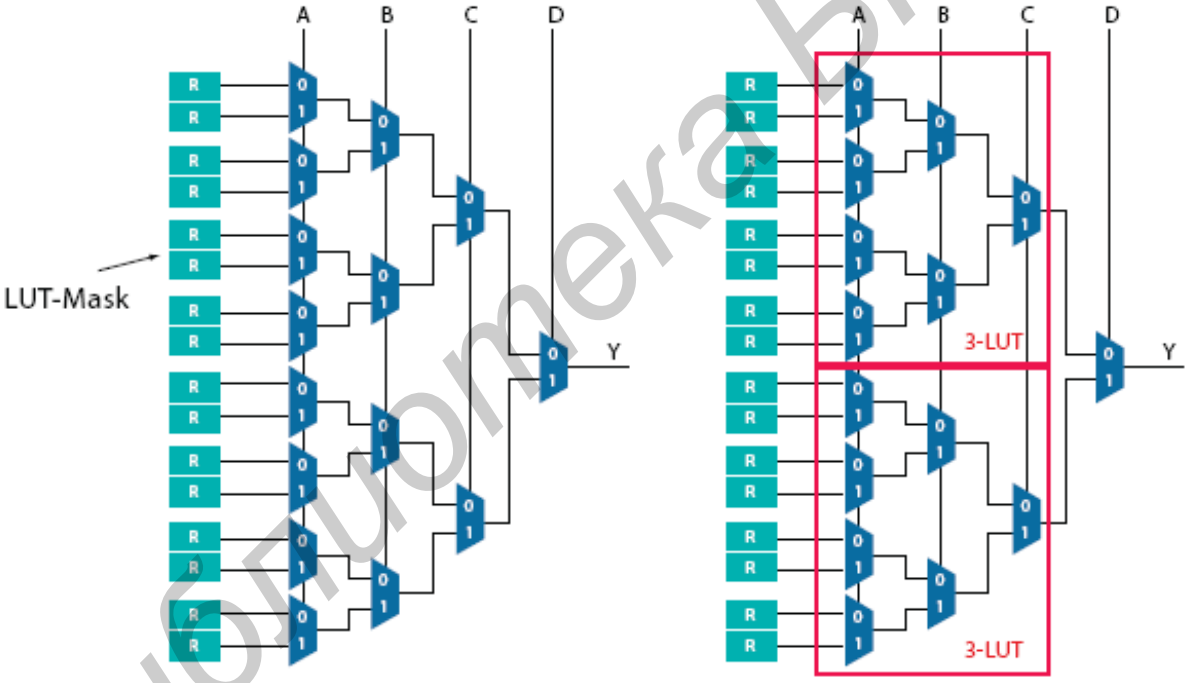

Рис. 2. Типичная архитектура LUT4

LUT с большим числом входов обычно строятся на основе LUT с меньшим числом входов и мультиплексоров. Например, LUT4 может быть реализована на базе двух LUT3, объединенных мультиплексором 2 в 1 (см. рис. 2).

Число входов LUT является важным параметром при выборе FPGA, поскольку оно непосредственно влияет на число уровней логики (число LUT, включенных последовательно), требуемых для реализации необходимой логической функции, и, следовательно, влияет на величину комбинационной задержки при реализации этой функции.

Современные FPGA обычно имеют LUT с 4–6 входами, позволяющими эффективно реализовывать функции 4–6 переменных. При реализации функций меньшего числа переменных (например двух переменных) такие LUT будут использоваться неэффективно. В этом случае целесообразно по возможности провести математические преобразования реализуемого алгоритма для получения последовательно используемых функций малого числа переменных. Например, две последовательные функции двух переменных могут быть сведены к одной функции трех переменных, требующей один уровень LUT вместо двух.

Помимо LUT в конфигурируемый логический блок включается синхронный *D*-триггер, предназначенный для хранения значения функции, реализуемой LUT (рис. 3).

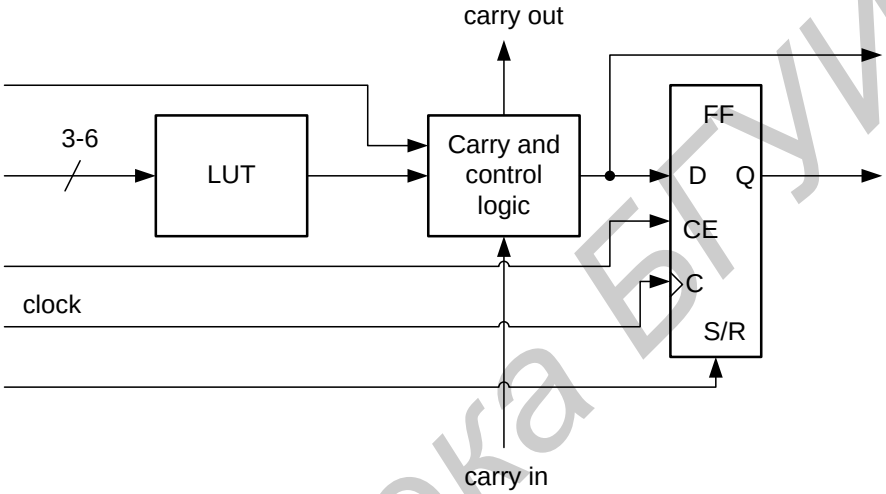

Рис. 3. Архитектура логического блока

С помощью логики управления, включающей мультиплексоры, на вход *D*-триггера может быть подключен либо выход LUT, либо внешний вход логического блока. Последняя возможность позволяет реализовывать многоразрядные регистровые структуры.

Выходной триггер логического блока обычно может быть сконфигурирован либо как *D*-триггер, либо как триггер-защелка (latche). Дополнительно триггер обычно имеет входы начальной установки и вход разрешения переключения по синхросигналу.

Выход LUT может быть подключен к выходу логического блока, минуя триггер, что позволяет реализовать последовательные комбинационные схемы большого числа переменных.

Помимо мультиплексоров в состав логического блока обычно включается схема ускоренного переноса, используемая при реализации сумматоров.

На такую архитектуру хорошо отображаются конвейерные вычисления, когда на LUT реализуются вычисления ступени конвейера, а в триггере фиксируется результат работы ступени. Если вычисления реализуются на одном уровне LUT, то будет получена максимально возможная для данной FPGA производительность поточного процессора.

Блок ввода/вывода IOB реализует программируемый двунаправленный интерфейс между внутренней логикой кристалла FPGA и выводом корпуса микросхемы (pad). Упрощенная схема блока ввода/вывода приведена на рис. 4.

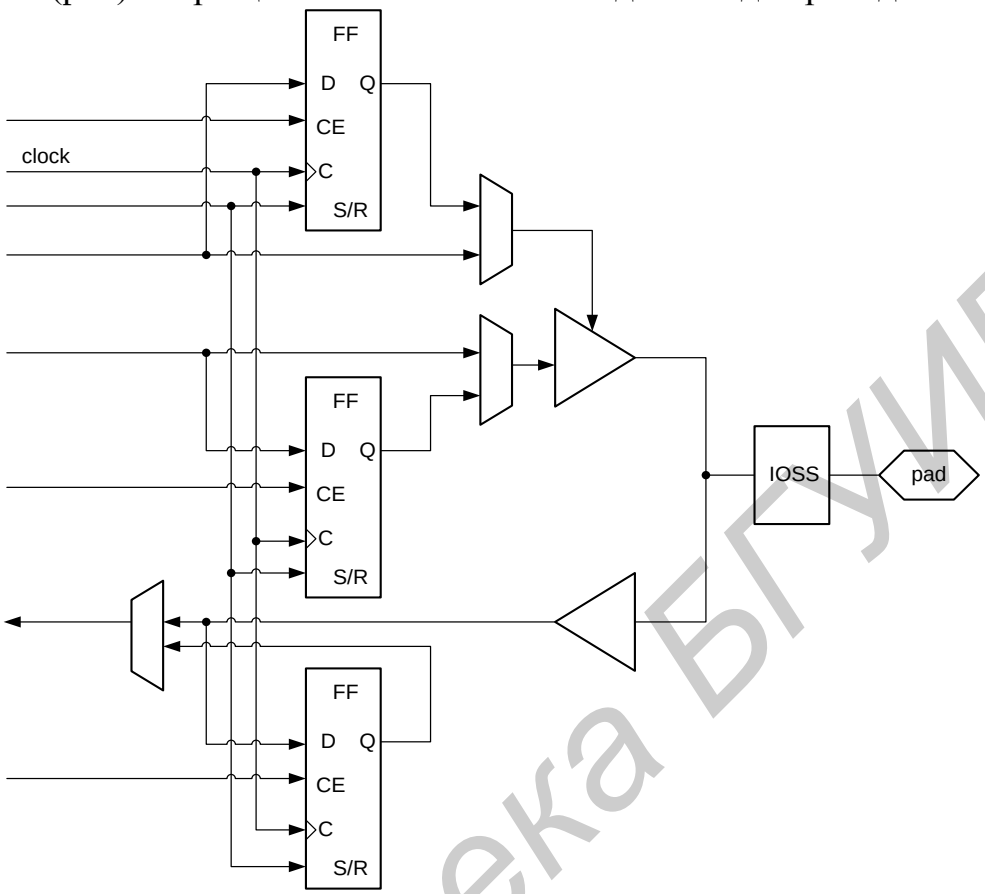

Рис. 4. Упрощенная схема блока ввода/вывода

Каждый IOB содержит входной и выходной буферы для обеспечения двунаправленного приема и передачи, триггеры для хранения входных и выходных значений, мультиплексоры и схему реализации стандарта ввода/вывода (IOSS). IOSS обеспечивает поддержку электрических уровней стандартов ввода/вывода (например LVTTL, LVCMOS, SSTL и др.). Кроме этого, IOB большинства производителей могут быть использованы попарно для осуществления поддержки некоторых дифференциальных стандартов (LVDS, DIFF_HSTL и др.).

IOB может быть запрограммирован как:

– входной блок без триггера или с триггером;

– выходной блок без триггера или с триггером, с буфером с тремя состояниями или без него, управляемый непосредственно или через триггер;

– двунаправленный блок.

Помимо рассмотренных ресурсов FPGA производители ПЛИС размещают на кристалле и некоторые другие ресурсы. Наиболее часто кристаллы FPGA дополнительно содержат:

– блочную память;

– блоки для реализации алгоритмов цифровой обработки сигналов;

– блоки для управления тактовой частотой.

Блочная память может использоваться в проектах для хранения относительно большого количества данных. Каждый блок памяти представляет собой синхронное ОЗУ емкостью несколько килобит в зависимости от производителя и семейства FPGA. Количество блоков памяти также зависит от семейства FPGA и конкретного кристалла ПЛИС этого семейства. Блоки можно каскадировать и таким образом реализовывать массивы памяти большей емкости. Чаще всего блок памяти имеет двухпортовую организацию.

При работе с двухпортовой памятью необходимо иметь в виду, что возможно возникновение коллизий при одновременной записи данных через один порт и чтении данных через другой порт по одному адресу (для одной ячейки памяти). Как разрешается данная ситуация, определяется разработчиком ПЛИС. Обычно запись будет выполнена правильно, а операция чтения даст неопределенный результат.

Блоки памяти обычно имеют настраиваемую разрядность шин данных и соответственно шин адреса, причем эта разрядность может быть разной для разных портов. В этом случае битовый массив данных блока памяти трактуется разделенным на ячейки данных разной разрядности при адресации по разным портам.

Блоки для реализации алгоритмов цифровой обработки сигналов в простейшем случае представляют собой матричные умножители. В некоторых FPGA такие блоки могут быть достаточно сложными устройствами, дополнительно включающими аккумуляторы, схемы сравнения и другие цифровые узлы.

Блоки для управления тактовой частотой также имеют различные возможности в FPGA разных семейств и разных производителей. В качестве наиболее распространенных можно указать следующие функции, поддерживаемые блоками для управления тактовой частотой:

– устранение сдвига фаз синхронизирующих импульсов. Достигается за счет выравнивания фаз выходного тактового сигнала, вырабатываемого блоком для управления тактовой частотой, и этого же сигнала, полученного через обратную связь;

– синтез требуемых значений тактовых частот. Из входного тактового сигнала одной частоты блок для управления тактовой частотой может сгенерировать на выходе тактовый сигнал с другой частотой. Это достигается путем умножения и/или деления частоты на целочисленный коэффициент допустимого диапазона;

– фазовый сдвиг тактового сигнала.

Наиболее известные производители ПЛИС: Actel (http://www.actel.com), Altera (http://www.altera.com), Atmel (www.atmel.com), Lattice Semiconductors (http://www.latticesemi.com), Quick logic (http://www.quicklogic.com), Xilinx (http://www.xilinx.com). В странах СНГ наиболее широко используются FPGA фирм Altera и Xilinx.

2.2. Средства проектирования цифровых систем на базе FPGA

Для разработки цифровых систем на базе FPGA имеются средства проектирования как фирм – изготовителей FPGA, так и сторонних производителей.

Независимо от конкретной системы проектирования при разработке цифровой системы на базе FPGA необходимо выполнить следующие этапы:

1) создать новый проект, указав семейство и тип FPGA;

2) разработать исходное описание проектируемого устройства в текстовой форме на HDL (Hardware Description Language), схемотехнической или алгоритмической форме;

3) провести проверку проекта методом функционального (поведенческого) моделирования;

4) выполнить синтез устройства. При этом имеется возможность задания различных стратегий синтеза (по производительности или по занятым ресурсам кристалла ПЛИС);

5) выполнить размещение и трассировку проекта в кристалле FPGA. При этом имеется возможность задания различных стратегий размещения (по производительности или по занятым ресурсам);

6) провести окончательную верификацию проекта методом временного моделирования с учетом задержек переключения логических элементов и элементов памяти и задержек на трассировочных ресурсах кристалла;

7) создать конфигурационную последовательность;

8) загрузить конфигурационные данные проекта в кристалл (выполнить программирование ПЛИС).

Из языков описания аппаратуры фактически стандартными стали языки VHDL и Verilog, проектирование на которых поддерживается всеми САПР для ПЛИС.

Фирма Altera предлагает пакет Quartus II для автоматизированного проектирования на ПЛИС. Имеется бесплатная версия пакета Web Edition. В бесплатной версии имеются ограничения по используемым кристаллам ПЛИС (поддерживается проектирование только кристаллов небольшой логической емкости) и отсутствуют некоторые программные инструменты. Пакет позволяет реализовать полный цикл разработки цифровых устройств на основе ПЛИС, включающий этапы создания исходных описаний проекта, синтеза, моделирования, размещения и трассировки, а также программирования кристалла.

В качестве языков исходного описания проектов поддерживаются VHDL, Verilog и AHDL. Кроме того, имеется возможность схемного описания проекта (Schematic File).

Среди дополнительных средств пакета Quartus II можно указать следующие программные инструменты:

– TimeQuest Timing Analyzer. Анализатор временных ограничений проекта. Позволяет провести анализ задержек по всем путям проекта или между указанными элементами с формированием отчетов;

– ModelSim. Внешняя программа (фирмы Mentor Graphics) для поведенческого и временного моделирования (с учетом задержек переключения элементов и задержек на трассировочных ресурсах), позволяющая провести верификацию проекта;

– MegaWizard. Средство для параметризованного описания устройств проекта, ускоряющее создание сложных узлов;

– Chip planner. Позволяет просмотреть и при необходимости отредактировать размещение и топологию проекта в кристалле ПЛИС.

Фирма Xilinx предлагает САПР ISE (Integrated Sowtware Environment) для разработки цифровых систем на базе ПЛИС. Также имеется бесплатная версия ISE WebPACK, которая имеет ограничения по используемым кристаллам ПЛИС и некоторым программным инструментам.

Программные средства ISE представляют собой систему сквозного проектирования, которая реализует полный цикл разработки цифровых устройств на основе ПЛИС, включающий этапы создания исходных описаний проекта, синтеза, моделирования, размещения и трассировки, а также программирования кристалла. ISE поддерживает различные виды исходного описания проектируемых устройств (графическое, в форме принципиальных схем или диаграмм состояний; текстовое, с использованием языков описания аппаратуры VHDL и Verilog). Для верификации проектов может использоваться либо встроенный симулятор, либо ModelSim.

ISE располагает комплектом вспомогательных программных средств, позволяющих повысить эффективность процесса проектирования, включающим анализатор временных характеристик проекта Timing Analyzer, интерактивный графический редактор размещения Floorplanner (в более поздних версиях PlanAhead), модуль оценки потребляемой мощности XPower, модуль для редактирования топологии FPGA Editor, модуль для отладки проекта в кристалле ПЛИС ChipScope.

Среди программных средств сторонних производителей можно отметить LeonardoSpectrum фирмы Mentor Graphics. Это средство логического синтеза VHDL и Verilog проектов, которое может быть в равной степени использовано для синтеза CPLD, FPGA или ASIC. Поддерживает ПЛИС Xilinx, Altera, Actel, Lattice и других поставщиков PLD и FPGA.

Рассмотренные системы автоматизированного проектирования обладают сопоставимыми возможностями и позволяют провести полный цикл проектирования цифровой системы на базе FPGA.

3. АЛГОРИТМЫ ВЫЧИСЛЕНИЯ ВЕКТОРНОГО ДПФ ДЛЯ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Определение 1. Под *M*-мерным векторным дискретным процессом понимают вектор

$$X = [x_0(n)\ \ x_1(n) \dots x_{M-1}(n)]^{\mathrm{T}}, \qquad (1)$$

компоненты которого $x_m(n),\ m = 0, \dots, M-1;\ n = 0, \dots, N-1$ являются числовыми последовательностями $\{x_m(n)\}$ длиной N.

В общем случае входные данные системы многоканальной обработки сигналов в реальном времени могут быть структурированы в виде *D*-мерного вектора

$$Z = [z_0(n)\ z_1(n) \dots z_{D-1}(n)]^{\mathrm{T}}, \quad (2)$$

где $z_d(n) = x_{2d}(n) + jx_{2d+1}(n), d = 0, \dots, D-1;\ D = M/2$, если M – четное, и $D = (M+1)/2$, если M – нечетное (при этом принимается, что $x_M(n) = 0$), который определяет *D*-мерный комплексный векторный дискретный процесс Z.

Определение 2. Под векторным дискретным преобразованием Фурье (ДПФ) понимают отображение VDFT вектора Z в вектор

$$U = [u_0(k)\ u_1(k) \dots u_{D-1}(k)]^{\mathrm{T}}, \quad (3)$$

компоненты которого $u_d(k),\ d = 0, \dots, D-1;\ k = 0, \dots, N-1$ представляют собой ДПФ последовательностей $\{z_d(n)\}$ длиной N, являющихся компонентами *D*-мерного векторного процесса Z,

VDFT: $Z \to U$ такое, что для любого n, k, d,

$n = 0, \dots, N-1; k = 0, \dots, N-1; d = 0, \dots, D-1;$

$u_d(k) = \mathrm{DFT}[z_d(n)]$,

где $\mathrm{DFT}[z_d(n)]$ – ДПФ последовательности $\{z_d(n)\}$ длиной N, т. е. покомпонентное преобразование *D*-мерного векторного процесса Z.

При реализации векторного ДПФ возникает проблема, каким образом выполнять совместно множество независимых одномерных ДПФ, составляющих по определению векторное ДПФ? Возможны два подхода к решению этой проблемы. Первый, тривиальный, подход лежит на поверхности и следует непосредственно из определения векторного ДПФ. Этот подход рассматривает вычисление ДПФ *D*-мерного векторного процесса как выполнение D независимых одномерных ДПФ формата N и дает следующие четыре основных алгоритма вычисления ДПФ *M*-мерного действительного векторного процесса X.

В первых двух алгоритмах одновременно выполняется один алгоритм БПФ по основанию 2 последовательно для всех компонентов векторного процесса.

Алгоритм 1. БПФ *M*-мерного действительного векторного процесса

Вход. Вектор $X = [x_0(n)\ x_1(n) \dots x_{M-1}(n)]^{\mathrm{T}}$ размером $M = 2^{q+1}$, компоненты которого $x_i(n),\ i = 0, \dots, M-1;\ n = 0, \dots, N-1$ являются действительными последовательностями длиной $N = 2^l$.

Выход. Вектора $\mathrm{Re}\, y_i = \left[\mathrm{Re}\left(y_i(0)\right) \dots \mathrm{Re}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$ и $\mathrm{Im}\, y_i = \left[\mathrm{Im}\left(y_i(0)\right) \dots \mathrm{Im}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$, $i = 0, \dots, M-1$, представляющие собой соответственно действительную и мнимую части ДПФ $y_i(k),\ k = 0, \dots, N/2-1$ компонентов $x_i(n)$ вектора X.

1. Сформировать комплексный *D*-мерный вектор Z вида (2), где $D = 2^q$.

2. Выполнить прямой алгоритм БПФ по основанию 2 компонента $z_d(n)$, $d = 0,\ n = 0, \dots, N-1$ вектора Z, т. е. получить ДПФ $u_d(k),\ d = 0,\ k = 0, \dots, N-1$ компонента $z_d(n)$ вектора Z.

3. Повторить шаг 2 для $d = 1, \dots, D-1$.

4. Выполнить разрядно-реверсивную перестановку массивов $u_d(k),\ d = 0, \dots, D-1;\ k = 0, \dots, N-1$ путем следующего преобразования двоичного представления индекса k:

$$\rho(k) = \{b_1, b_2, \dots, b_l\}, \quad (4)$$

где $k = \{b_l,\ b_{l-1}, \dots, b_1\} = b_l 2^{l-1} + b_{l-1} 2^{l-2} + \dots + b_1$ представляет собой двоичный адрес элемента в массиве; $b_i = \{0, 1\}$.

5. Сформировать вектора $\mathrm{Re}\, U$ и $\mathrm{Im}\, U$ вида

$$\begin{aligned} &\mathrm{Re}\, U = [\mathrm{Re}\, u_0\ \ \mathrm{Re}\, u_1 \dots \mathrm{Re}\, u_{D-1}]^{\mathrm{T}}, \\ &\mathrm{Im}\, U = [\mathrm{Im}\, u_0\ \ \mathrm{Im}\, u_1 \dots \mathrm{Im}\, u_{D-1}]^{\mathrm{T}}. \end{aligned} \tag{5}$$

6. Определить вектора $\mathrm{Re}\, y_i$ и $\mathrm{Im}\, y_i$, $i = 0, \dots, M-1$ по выражениям

$$\begin{aligned} &[\mathrm{Re}\, y_0\ \ \mathrm{Im}\, y_1\ \mathrm{Re}\, y_2\ \ \mathrm{Im}\, y_3 \dots \mathrm{Re}\, y_{2D-2}\ \mathrm{Im}\, y_{2D-1}]^{\mathrm{T}} = \\ &= \left(\left(I_N + E_{N/2} \otimes B'\right) \otimes I_D\right) \mathrm{Re}\, U, \\ &[\mathrm{Re}\, y_1\ \ \mathrm{Im}\, y_0\ \mathrm{Re}\, y_3\ \ \mathrm{Im}\, y_2 \dots \mathrm{Re}\, y_{2D-1}\ \mathrm{Im}\, y_{2D-2}]^{\mathrm{T}} = \\ &= \left(\left(I_{N/2} \otimes B'' + E_N\right) \otimes I_D\right) \mathrm{Im}\, U, \end{aligned} \tag{6}$$

где I_n – единичная матрица размером $n \times n$; E_n – матрица перестановки размером $n \times n$ $E_{\mathrm{n}} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & & 1 & 0 \\ & & \vdots & & \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}$; $B' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $B'' = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$; $\otimes$ – прямое произведение матриц (произведение Кронекера).

Алгоритм 2. БПФ M-мерного действительного векторного процесса

Вход. Вектор $X = [x_0(n)\ x_1(n) \dots x_{M-1}(n)]^{\mathrm{T}}$ размером $M = 2^{q+1}$, компоненты которого $x_i(n)$, $i = 0, \dots, M-1$; $n = 0, \dots, N-1$ являются действительными последовательностями длиной $N = 2^l$.

Выход. Вектора $\mathrm{Re}\, y_i = \left[\mathrm{Re}\left(y_i(0)\right) \dots \mathrm{Re}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$ и $\mathrm{Im}\, y_i = \left[\mathrm{Im}\left(y_i(0)\right) \dots \mathrm{Im}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$, $i = 0, \dots, M-1$, представляющие собой соответственно действительную и мнимую части ДПФ $y_i(k)$, $k = 0, \dots, N/2-1$ компонентов $x_i(n)$ вектора X.

1. Сформировать комплексный D-мерный вектор Z вида (2), где $D = 2^q$.

2. Выполнить разрядно-реверсивную перестановку компонентов $z_d(n)$, $d = 0, \dots, D-1$; $n = 0, \dots, N-1$ вектора Z путем преобразования (4) двоичного представления индекса n.

3. Выполнить обратный алгоритм БПФ по основанию 2 компонента $z_d(n)$, $d = 0$, $n = 0, \dots, N-1$ вектора Z, т. е. получить ДПФ $u_d(k)$, $d = 0$, $k = 0, \dots, N-1$ компонента $z_d(n)$ вектора Z.

4. Повторить шаг 3 для $d = 1, \dots, D-1$.

5. Сформировать вектора $\mathrm{Re}\, U$ и $\mathrm{Im}\, U$ вида (5).

6. Определить вектора $\mathrm{Re}\, y_i$ и $\mathrm{Im}\, y_i$, $i = 0, \dots, M-1$ по выражениям (6).

В следующих двух алгоритмах все множество D компонентов комплексного векторного процесса разбивается на p, $p = 2, 4, 8, \dots, D$ подмножеств, и преобразования компонентов каждого подмножества выполняются одновременно с помощью своего алгоритма БПФ по основанию 2. При этом внутри каждого подмножества преобразования выполняются последовательно.

Алгоритм 3. БПФ *M*-мерного действительного векторного процесса

Вход. Вектор $X = [x_0(n)\, x_1(n) \ldots x_{M-1}(n)]^{\mathrm{T}}$ размером $M = 2^{q+1}$, компоненты которого $x_i(n),\ i = 0, \ldots, M-1;\ n = 0, \ldots, N-1$ являются действительными последовательностями длиной $N = 2^l$.

Выход. Вектора $\mathrm{Re}\, y_i = \left[\mathrm{Re}\left(y_i(0)\right) \ldots \mathrm{Re}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$ и $\mathrm{Im}\, y_i = \left[\mathrm{Im}\left(y_i(0)\right) \ldots \mathrm{Im}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$, $i = 0, \ldots, M-1$, представляющие собой соответственно действительную и мнимую части ДПФ $y_i(k),\ k = 0, \ldots, N/2-1$ компонентов $x_i(n)$ вектора X.

1. Сформировать комплексный *D*-мерный вектор Z вида (2), где $D = 2^q$.

2. Разбить множество компонентов $z_d(n),\ d = 0, \ldots, D-1;\ n = 0, \ldots, N-1$ вектора Z на $p,\ p = 2,\ 4,\ 8, \ldots, D$ подмножеств по D/p компонентов в каждом подмножестве путем отображения

$$z_{i,j}(n) = z_d(n), \qquad (7)$$

где $i = 0, \ldots, p-1$ – номер подмножества; $j = 0, \ldots, D/p-1$ – номер компонента в подмножестве, с помощью преобразования индексов

$$i = [d/p], \qquad j = d \bmod p, \qquad (8)$$

где $[a]$ – означает целую часть числа a.

3. Выполнить одновременно p прямых алгоритмов БПФ по основанию 2 массивов $z_{i,j}(n),\ i = 0, \ldots, p-1;\ j = 0$, т. е. получить p ДПФ $u_{i,j}(k),\ k = 0, \ldots, N-1$ массивов $z_{i,j}(n)$.

4. Повторить шаг 3 для $j = 1, \ldots, D/p-1$.

5. Объединить p подмножеств массивов $u_{i,j}(k)$ в множество ДПФ $u_d(k)$ компонентов $z_d(n)$ вектора Z путем обратного отображения

$$u_d(k) = u_{i,j}(k) \qquad (9)$$

с помощью преобразования индексов

$$d = ip + j. \qquad (10)$$

6. Выполнить разрядно-реверсивную перестановку массивов $u_d(k),\ d = 0, \ldots, D-1;\ k = 0, \ldots, N-1$ путем преобразования (4) двоичного представления индекса k.

7. Сформировать вектора $\mathrm{Re}\, U$ и $\mathrm{Im}\, U$ вида (5).

8. Определить вектора $\mathrm{Re}\, y_i$ и $\mathrm{Im}\, y_i,\ i = 0, \ldots, M-1$ по выражениям (6).

Алгоритм 4. БПФ *M*-мерного действительного векторного процесса

Вход. Вектор $X = [x_0(n)\, x_1(n) \ldots x_{M-1}(n)]^{\mathrm{T}}$ размером $M = 2^{q+1}$, компоненты которого $x_i(n),\ i = 0, \ldots, M-1;\ n = 0, \ldots, N-1$ являются действительными последовательностями длиной $N = 2^l$.

Выход. Вектора $\mathrm{Re}\, y_i = \left[\mathrm{Re}\left(y_i(0)\right) \ldots \mathrm{Re}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$ и $\mathrm{Im}\, y_i = \left[\mathrm{Im}\left(y_i(0)\right) \ldots \mathrm{Im}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$, $i = 0, \ldots, M-1$, представляющие собой соответственно действительную и мнимую части ДПФ $y_i(k),\ k = 0, \ldots, N/2-1$ компонентов $x_i(n)$ вектора X.

1. Сформировать комплексный *D*-мерный вектор Z вида (2), где $D = 2^q$.

2. Выполнить разрядно-реверсивную перестановку компонентов $z_d(n)$, $d = 0, \dots, D-1$; $n = 0, \dots, N-1$ вектора Z путем преобразования (4) двоичного представления индекса n.

3. Разбить множество компонентов $z_d(n), d = 0, \dots, D-1; n = 0, \dots, N-1$ вектора Z на p, $p = 2, 4, 8, \dots, D$ подмножеств по D/p компонентов в каждом подмножестве путем отображения (7) с помощью преобразования индексов (8).

4. Выполнить одновременно p прямых алгоритмов БПФ по основанию 2 массивов $z_{i,j}(n)$, $i = 0, \dots, p-1$; $j = 0$, т. е. получить p ДПФ $u_{i,j}(k)$, $k = 0, \dots, N-1$ массивов $z_{i,j}(n)$.

5. Повторить шаг 4 для $j = 1, \dots, D/p - 1$.

6. Объединить p подмножеств массивов $u_{i,j}(k)$ в множество ДПФ $u_d(k)$ компонентов $z_d(n)$ вектора Z путем обратного отображения (9) с помощью преобразования индексов (10).

7. Сформировать вектора Re U и Im U вида (5).

8. Определить вектора Re y_i и Im y_i, $i = 0, \dots, M-1$ по выражениям (6).

Второй подход к вычислению векторного ДПФ базируется на том факте, что алгоритм БПФ позволяет вычисление ДПФ D-мерного векторного процесса свести к обработке одного одномерного массива размером DN. Основой для этого служат следующие две теоремы.

Теорема 1 [1]. Пусть элементы

$z_d(n), d = 0, \dots, D-1; n = 0, \dots, N-1$

D комплексных последовательностей $\{z_d(n)\}$ длиной $N = 2^l$ структурированы в виде одномерного массива $z(n^*)$ размером DN следующим образом:

$$z(n^*) = z_d(n), \qquad n^* = nD + d, \qquad n^* = 0, \dots, DN-1. \tag{11}$$

Тогда, применив прямой алгоритм БПФ по основанию 2 к массиву $z(n^*)$ и остановив его после выполнения l итераций, получим на выходе D ДПФ размером N исходных последовательностей.

Доказательство теоремы 1 приведено в [1].

Аналогичная теорема может быть доказана и для обратного алгоритма БПФ.

Теорема 2. Пусть элементы

$z_d(n), d = 0, \dots, D-1; n = 0, \dots, N-1$

D комплексных последовательностей $\{z_d(n)\}$ длины $N = 2^l$ структурированы в виде одномерного массива $z(n^*)$ размером DN следующим образом:

$$z(n^*) = z_d(n), \qquad n^* = dN + n, \qquad n^* = 0, \dots, DN-1. \tag{12}$$

Тогда, применив обратный алгоритм БПФ по основанию 2 к массиву $z(n^*)$ и остановив его после выполнения l итераций, получим на выходе D ДПФ размером N исходных последовательностей.

Доказательство. ДПФ массива $z(n^*)$ имеет вид

$$u(k) = \frac{1}{DN} \sum_{n^*=0}^{DN-1} z(n^*) w_{DN}^{-n^*k}, k = 0, \dots, DN-1. \tag{13}$$

Пусть $k = \alpha D + \beta$, $\alpha = 0, \dots, N-1$; $\beta = 0, \dots, D-1$. Тогда

$$u(k) = \frac{1}{DN}\sum_{d=0}^{D-1}\sum_{n=0}^{N-1} z(dN+n)w_{DN}^{-k(dN+n)}. \quad (14)$$

В выражении (14)

$$w_{DN}^{-k(dN+n)} = w_{DN}^{-(\alpha D+\beta)(dN+n)} = w_N^{-\alpha n} w_{DN}^{-\beta(dN+n)}. \quad (15)$$

Подставив (15) в выражение (14), получим

$$u(k) = \frac{1}{D}\sum_{d=0}^{D-1}\left[\frac{1}{N}\sum_{n=0}^{N-1} z(dN+n)w_N^{-\alpha n}\right] w_{DN}^{-\beta(dN+n)} = \quad (16)$$

$$= \frac{1}{D}\sum_{d=0}^{D-1} u'(dN+\alpha)w_{DN}^{-\beta(dN+\alpha)},$$

где

$$u'(dN+\alpha) = \frac{1}{N}\sum_{n=0}^{N-1} z(dN+n)w_N^{-\alpha n}, \quad (17)$$

$\alpha = 0, \dots, N-1;\ \beta = 0, \dots, D-1.$

Величины $u'(dN+\alpha)$ представляют собой результат первых l итераций обратного алгоритма БПФ. При фиксированном значении d N значений $u'(dN+\alpha)$ являются ДПФ размером N исходной последовательности $\{z_d(n)\}$, т. е. $u_d(k),\ k = 0, \dots, N-1$, что и требовалось доказать.

Так как векторное ДПФ представляет собой множество независимых одномерных преобразований размером N, то из теорем 1 и 2 следует утверждение 1.

Утверждение 1. Пусть с помощью преобразования индексов

$$n^* = nD + d, \quad (18)$$

или

$$n^* = dN + n, \quad (19)$$

где $n^* = 0, \dots, DN-1;\ n = 0, \dots, N-1;\ d = 0, \dots, D-1;\ N = 2^l$, D-мерный векторный процесс Z, заданный (2), отображается в одномерный массив $z(n^*)$ следующим образом:

$$z(n^*) = z_d(n), \quad (20)$$

где $z_d(n)$ – компоненты векторного процесса Z.

Тогда результат выполнения первых l итераций $u(k^*),\ k^* = 0, \dots, DN-1$ соответственно прямого или обратного алгоритма БПФ по основанию 2 массива $z(n^*)$ представляет собой ДПФ исходного D-мерного векторного процесса Z (векторное ДПФ). При этом ДПФ $u_d(k)$ компонентов D-мерного векторного процесса Z получаются путем обратного отображения одномерного массива $u(k^*)$ размером DN:

$$u_d(k) = u(k^*), d = 0, \dots, D-1;\ k = 0, \dots, N-1 \quad (21)$$

с помощью преобразования индексов соответственно

$$d = k^* \bmod D, \qquad k = \left[\frac{k^*}{D}\right], \quad (22)$$

или

$$d = \left[\frac{k^*}{N}\right], \qquad k = k^* \bmod N, \tag{23}$$

где $[a]$ – целая часть числа a.

Утверждение 1 имеет два важных следствия [2].

Следствие 1. Вычисление векторного ДПФ (множества D одномерных преобразований формата N) сводится к вычислению одного одномерного DN-точечного ДПФ.

Следствие 2. Вычисление одного DN-точечного ДПФ, где $D = 2^q$, $N = 2^l$, позволяет при минимальном количестве дополнительной аппаратуры вычислять от 1 до $DN/2$ преобразований соответственно длиной от DN до 2, изменяя только число выполняемых итераций в алгоритме БПФ по основанию 2 соответственно от $q + 1$ до 1.

Второй подход позволяет сформулировать следующие два алгоритма вычисления ДПФ M-мерного действительного векторного процесса X, заданного определением 1.

Алгоритм 5. БПФ M-мерного действительного векторного процесса [3, 4]

Вход. Вектор $X = [x_0(n)\, x_1(n) \ldots x_{M-1}(n)]^{\mathrm{T}}$ размером $M = 2^{q+1}$, компоненты которого $x_i(n)$, $i = 0, \ldots, M-1$; $n = 0, \ldots, N-1$ являются действительными последовательностями длиной $N = 2^l$.

Выход. Вектора $\mathrm{Re}\, y_i = \left[\mathrm{Re}\big(y_i(0)\big) \ldots \mathrm{Re}\big(y_i(N/2-1)\big)\right]^{\mathrm{T}}$ и $\mathrm{Im}\, y_i = \left[\mathrm{Im}\big(y_i(0)\big) \ldots \mathrm{Im}\big(y_i(N/2-1)\big)\right]^{\mathrm{T}}$, $i = 0, \ldots, M-1$, представляющие собой соответственно действительную и мнимую части ДПФ $y_i(k)$, $k = 0, \ldots, N/2-1$ компонентов $x_i(n)$ вектора X.

1. Сформировать комплексный D-мерный вектор Z вида (2), где $D = 2^q$.

2. Выполнить с помощью преобразования индексов (18) отображение (20) вектора Z в одномерный массив $z(n^*)$ размером DN.

3. Выполнить l итераций обратного алгоритма БПФ по основанию 2 массива $z(n^*)$, в результате чего получить одномерный массив $u(k^*)$, $k^* = 0, \ldots, DN-1$.

4. Выполнить l-ю разрядно-реверсивную перестановку массива $u(k^*)$ путем следующего преобразования двоичного представления индекса k^*:

$$\rho_{(l)}(k^*) = \{b_{q+1},\, b_{q+l-1}, \ldots,\, b_{l+1},\, b_1,\, b_2, \ldots,\, b_l\}, \tag{24}$$

где $k^* = \{b_{q+1},\, b_{q+l+1}, \ldots,\, b_1\} = b_{q+1}2^{q+l-1} + b_{q+l-1}2^{q+l-2} + \cdots + b_1$ – представляет собой двоичный адрес элемента в массиве; $b_i = \{0,\ 1\}$.

5. Получить ДПФ $u_d(k)$, $d = 0, \ldots, D-1$; $k = 0, \ldots, N-1$ компонентов $z_d(n)$ вектора Z путем обратного отображения (21) одномерного массива $u(k^*)$ с помощью преобразования индексов (22).

6. Сформировать вектора $\mathrm{Re}\, U$ и $\mathrm{Im}\, U$ вида (5).

7. Определить вектора $\mathrm{Re}\, y_i$ и $\mathrm{Im}\, y_i$, $i = 0, \ldots, M-1$ по выражениям (6).

Алгоритм 6. БПФ *M*-мерного действительного векторного процесса

Вход. Вектор $X = [x_0(n)\, x_1(n) \ldots x_{M-1}(n)]^{\mathrm{T}}$ размером $M = 2^{q+1}$, компоненты которого $x_i(n),\ i = 0, \ldots, M-1;\ n = 0, \ldots, N-1$ являются действительными последовательностями длиной $N = 2^l$.

Выход. Вектора $\mathrm{Re}\, y_i = \left[\mathrm{Re}\left(y_i(0)\right) \ldots \mathrm{Re}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$ и $\mathrm{Im}\, y_i = \left[\mathrm{Im}\left(y_i(0)\right) \ldots \mathrm{Im}\left(y_i(N/2-1)\right)\right]^{\mathrm{T}}$, $i = 0, \ldots, M-1$, представляющие собой соответственно действительную и мнимую части ДПФ $y_i(k),\ k = 0, \ldots, N/2-1$ компонентов $x_i(n)$ вектора X.

1. Сформировать комплексный *D*-мерный вектор Z вида (2), где $D = 2^q$.

2. Выполнить с помощью преобразования индексов (19) отображение (20) вектора Z в одномерный массив $z(n^*)$ размером DN.

3. Выполнить l-ю разрядно-реверсивную перестановку массива $z(n^*)$ путем преобразования (24) двоичного представления индекса n^*

4. Выполнить l итераций обратного алгоритма БПФ по основанию 2 массива $z(n^*)$, в результате чего получить одномерный массив $u(k^*)$, $k^* = 0, \ldots, DN-1$.

5. Получить ДПФ $u_d(k),\ d = 0, \ldots, D-1;\ k = 0, \ldots, N-1$ компонентов $z_d(n)$ вектора Z путем обратного отображения (21) одномерного массива $u(k^*)$ с помощью преобразования индексов (23).

6. Сформировать вектора $\mathrm{Re}\, U$ и $\mathrm{Im}\, U$ вида (5).

7. Определить вектора $\mathrm{Re}\, y_i$ и $\mathrm{Im}\, y_i$, $i = 0, \ldots, M-1$ по выражениям (6).

Шаг 2 алгоритма 1, шаг 3 алгоритмов 2, 3 и 5, шаг 4 алгоритмов 4 и 6 наиболее эффективно реализуются на основе параллельно-поточной структуры (ППБПФ-структуры). При этом требуется решить следующие две основные задачи:

1) определить оптимальное количество вычислительных элементов на ступени в параллельно-поточном БПФ-процессоре (ППБПФ-процессоре) $m_{\text{опт}}$, обеспечивающее при заданной структуре и элементной базе арифметического устройства (АУ) базовой операции (БО) вычисление векторного ДПФ в реальном масштабе времени с минимальной структурной сложностью;

2) определить топологию ППБПФ-процессора.

4. ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНО-ПОТОЧНЫХ БПФ-ПРОЦЕССОРОВ ПО ОСНОВАНИЮ 2

В поточном процессоре БПФ параллельно работают $l = \log_2 N$, где N – формат преобразования, вычислительных элементов (ВЭ) по одному на каждую итерацию алгоритма. При этом каждый ВЭ за время выполнения преобразования $T_{\text{пр}}$ вычисляет $N/2$ базовых операций алгоритма БПФ. Поточная структура построения процессора БПФ основывается на том факте, что каждый ВЭ может начать вычисления после того, как завершено вычисление не всех, а некоторого числа предшествующих базовых операций. В такой поточной структуре ВЭ выполняет базовую операцию алгоритма БПФ с прореживанием по времени или по частоте с одним комплексным умножением и двумя комплексными сложе-

ниями. После того, как первоначальный набор данных проходит через процессор, все ВЭ оказываются непрерывно заняты и работают с одной и той же скоростью. Так как данные поступают в процессор по двум линиям, то один полный N-точечный набор может быть введен за $N/2$ тактов, и время прохождения данных через одну ступень процессора равно $T_{\text{ст}} = 2\Delta t$, где Δt – период дискретизации. Следовательно, частота передачи данных в поточном процессоре БПФ с основанием 2 в два раза меньше частоты дискретизации.

Основные характеристики поточного процессора БПФ с основанием 2 приведены в табл. 1.

Таблица 1

Основные характеристики поточного процессора БПФ с основанием 2

Параметр	Оценка
1. Количество входов	2
2. Количество ступеней	$\log_2 N$
3. Количество ВЭ на ступени	1
4. Общее количество ВЭ	$\log_2 N$
5. Объем промежуточной памяти, слов	$N - 2$
6. Задержка, тактов	$\log_2 N + \frac{N}{2} - 1$
7. Время выполнения базовой операции (длительность такта)	$2\Delta t$

Процессор БПФ может строиться и по другой поточной структуре, которая состоит из трех типов модулей:

1) ВЭ, который выполняет малоточечное ДПФ и необходимые коммутации данных;

2) комплексного умножителя или модуля поворота CORDIC, который служит для умножения на поворачивающие множители;

3) сдвигового регистра для промежуточного хранения данных.

При заданном быстродействии элементной базы и структуре ВЭ время выполнения базовой операции фиксировано, а значит, ограничена максимальная допустимая скорость обработки в поточном процессоре БПФ. Требуемая степень параллелизма вычислений может быть достигнута путем использования для построения процессора БПФ **частично-параллельной структуры – параллельно-поточной**. Параллельно-поточная структура по аппаратурной сложности занимает промежуточное положение между поточной и матричной структурами и имеет достаточное количество m параллельно работающих ВЭ на каждой ступени для достижения требуемого увеличения быстродействия. При параллельном соединении m ВЭ на каждой ступени логические схемы параллельно-поточного процессора могут работать в m раз медленнее, чем в однопоточном, т. к. каждый ВЭ вычисляет в m раз меньше базовых операций соответствующей итерации.

Количество параллельно работающих на ступени ВЭ m может быть произвольным. Нижний предел m равен единице, что соответствует однопоточному процессору БПФ, верхний – $N/2$, что соответствует матричному процессору. Однако удобно ограничить m значениями, равными степени 2, т. к. в этом случае каждый ВЭ выполняет одинаковое число базовых операций, и вычислительная нагрузка равномерно распределяется между всеми ВЭ.

В общем случае при использовании m параллельно работающих вычислительных элементов на ступени каждый ВЭ выполняет базовые операции с номерами, отличающимися на $N/2m$. При этом ВЭ итерации с 0 до $\log_2 m$ соединяются непосредственно без промежуточной памяти, а ВЭ итераций с $\log_2 m$ до $\log_2 N$ образуют m меньших независимых поточных БПФ длиной N/m каждый.

По сравнению с поточным процессором БПФ управление промежуточной памятью в параллельно-поточном процессоре проще, т. к. количество ступеней, для которых обязательно наличие промежуточной памяти, меньше, а управление памятью является параллельным, что требует только одного множества управляющих сигналов.

Таким образом, параллельно-поточная структура процессоров БПФ хорошо соответствует алгоритму БПФ, что позволяет:

– значительно упростить взаимосвязи в процессоре БПФ;

– варьируя параметр m, получить архитектуры с любой производительностью;

– при реализации процессора в микроэлектронном исполнении использовать небольшое количество однотипных микросхем.

В табл. 2 приведены основные характеристики параллельно-поточного процессора БПФ с основанием 2.

Таблица 2

Основные характеристики ППБПФ процессора с основанием 2

Параметр	Оценка
1. Количество входов	$2m$
2. Количество ступеней	$\log_2 N$
3. Количество ВЭ на ступени	m
4. Общее количество ВЭ	$m \log_2 N$
5. Объем промежуточной памяти, слов	$N - 2m$
6. Задержка, тактов	$\log_2 N + \frac{N}{2m} - 1$
7. Время выполнения базовой операции (длительность такта)	$2m\Delta t$

Матричная структура представляет собой $\log_2 N$-уровневую структуру, на каждом уровне которой все базовые операции преобразования вычисляются параллельно. В такой структуре все преобразования реализуются параллельно в одной матричной схеме. Матричные процессоры обладают очень большой скоростью вычислений, однако даже при умеренном количестве D и длине N преобразований требуют значительного объема аппаратных средств.

Для вычисления дискретного преобразования Фурье могут использоваться однородные вычислительные структуры: однородные среды и систолические структуры.

Однородные среды представляют собой комплекс из однородной вычислительной среды (ОВС) и однородной запоминающей среды (ОЗС). ОВС построена в виде матрицы идентичных вычислительных ячеек (ВЯ), каждая из которых соединена с четырьмя ближайшими соседями. ВЯ электрически перепрограммируется на одну из общего списка возможных арифметико-логических и коммутационных операций. ОЗС построена аналогичным образом, но в узле регулярной матрицы стоит запоминающая ячейка (ЗЯ), которая представляет собой регистр сдвига с электрически перепрограммируемой величиной сдвига.

Систему обработки на однородных средах строят традиционно, сохраняя принцип однородности и регулярности связей в рамках всего устройства, что обеспечивает ему универсальность и способность к перестройке в широком классе задач. Но такая архитектура системы накладывает ограничения на класс реализуемых алгоритмов:

– максимальная простота узла обработки;

– локальность связей между узлами;

– минимальное число разнотипных узлов;

– максимальная возможность распараллеливания и конвейеризации.

Таким требованиям наиболее полно отвечают алгоритмы цифровой обработки сигналов (ЦОС), реализуемые непосредственно через операцию умножения вектора на матрицу и матрицы на матрицу. Быстрые же алгоритмы ЦОС не удовлетворяют приведенным выше требованиям, и их реализация в традиционной структуре становится неэффективной. В частности, алгоритм БПФ требует большого числа нелокальных передач, которые приводят к существенным аппаратурным затратам на коммутацию при реализации на однородных средах.

Организация вычислительного процесса в параллельно-поточном процессоре БПФ определяется используемыми алгоритмом вычисления ДПФ *M*-мерного векторного процесса и способом параллельного выполнения базовых операций в алгоритме БПФ по основанию 2. С точки зрения выполнения условий реального масштаба времени при одинаковом количестве вычислительных элементов в ППБПФ-процессоре все шесть алгоритмов, рассмотренных в предыдущем разделе, эквивалентны. Однако они различаются организацией вычисления БПФ (выполнения базовых операций алгоритма БПФ по основанию 2) отдельных компонентов векторного процесса, структурной сложностью реализации ППБПФ-процессора, а также временем и задержкой вычисления векторного ДПФ.

При использовании алгоритмов 1 и 2 преобразования компонентов векторного процесса выполняются последовательно на одном ППБПФ-процессоре с m вычислительными элементами на ступени. Организация вычислительного процесса в обратном ППБПФ-процессоре с одновременным выполнением m последовательных базовых операций для $D = 4$, $N = 8$ и $m = 2$ приведена в табл. 3.

Таблица 3

Порядок выполнения базовых операций в обратном ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 2 для $D = 4$, $N = 8$ и $m = 2$. Одновременно выполняются m последовательных базовых операций

Ступень	АУ	Такты										
		1	2	3	4	5	6	7	8	9	10	11
0	0	0.0	0.2	1.0	1.2	2.0	2.2	3.0	3.2	×	×	×
	1	0.1	0.3	1.1	1.3	2.1	2.3	3.1	3.3	×	×	×
1	0	–	0.0	0.2	1.0	1.2	2.0	2.2	3.0	3.2	×	×
	1	–	0.1	0.3	1.1	1.3	2.1	2.3	3.1	3.3	×	×
2	0	–	–	–	0.0	0.2	1.0	1.2	2.0	2.2	3.0	3.2
	1	–	–	–	0.1	0.3	1.1	1.3	2.1	2.3	3.1	3.3

Примечания:

– базовые операции, выполняемые над данными предыдущей выборки векторного процесса;

× базовые операции, выполняемые над данными последующей выборки векторного процесса.

В табл. 3 обозначение i, j означает j-ю базовую операцию, выполняемую над данными i-го компонента векторного процесса. В этом случае каждый вычислительный элемент ППБПФ-процессора выполняет последовательно все закрепленные за ним базовые операции БПФ одного компонента векторного процесса и только после этого переходит к обработке следующего компонента. При этом организация вычислительного процесса в ППБПФ-процессоре при выполнении векторного ДПФ не отличается от организации вычисления одного БПФ одномерного массива.

Оценки структурной сложности реализации ППБПФ-процессора и задержки вычисления векторного ДПФ при организации вычислений по алгоритму 1 или 2 совпадают с соответствующими оценками для параллельно-поточной структуры при выполнении одномерного БПФ и определяются следующими выражениями:

– сложность структуры:

$$C_{\text{ВБПФ1}} = C_{\text{АУ}} m \log_2 N + 2C_{\text{П}}(N - 2m) + 2C_{\text{К}} m \left(\log_2 \left(\frac{N}{m}\right) - 1\right), \qquad (25)$$

где $C_{\text{АУ}}, C_{\text{П}}, C_{\text{К}}$ – сложность арифметического устройства базовой операции, ячейки локальной памяти и коммутатора соответственно;

– задержка вычисления векторного ДПФ, тактов:

$$P_{\text{ВБПФ1}} = \frac{N}{2m} + \log_2 N - 1. \qquad (26)$$

Оценки латентности и времени вычисления векторного ДПФ в этом случае определяются следующим образом:

– латентность, тактов:

$$L_{\text{ВБПФ1}} = \frac{ND}{2m}; \qquad (27)$$

– время вычисления векторного ДПФ, тактов:

$$T_{\text{ВБПФ1}} = (D + p)\frac{ND}{2m} + \log_2 N - 2. \quad (28)$$

При использовании алгоритмов 3 и 4 БПФ компонентов каждого подмножества выполняются последовательно на своем ППБПФ-процессоре с m/p вычислительными элементами на ступени, где m – количество вычислительных элементов, выполняющих базовые операции одноименной итерации алгоритма БПФ по основанию 2; $p = 2,\ 4,\ 8, \ldots, D$ – число подмножеств разбиения компонентов векторного процесса. Организация вычислительного процесса в ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 4 для $D = 4$, $N = 8$, $p = 2$, $m = 2$ приведена в табл. 4. В этом случае m вычислительных элементов одной ступени ППБПФ-процессора выполняют одновременно по m/p базовых операций соответствующей итерации БПФ p различных компонентов векторного процесса. При этом организация вычислительного процесса в каждом ППБПФ-процессоре, вычисляющем преобразования одного подмножества компонентов векторного процесса, при выполнении векторного ДПФ не отличается от организации вычисления одного БПФ одномерного массива.

Оценка латентности при организации вычисления векторного ДПФ по алгоритму 3 или 4 совпадает с соответствующей оценкой (27), оценки остальных параметров ППБПФ-процессора в этом случае определяются следующим образом:

– сложность структуры:

$$C_{\text{ВБПФ2}} = C_{\text{АУ}} m \log_2 N + 2C_{\text{П}}(Np - 2m) + 2C_{\text{К}} m \log_2 \left(\frac{Np}{2m}\right); \quad (29)$$

– время вычисления векторного ДПФ, тактов:

$$T_{\text{ВБПФ2}} = (D + p)\frac{N}{2m} + \log_2 N - 2; \quad (30)$$

– задержка вычисления векторного ДПФ, тактов:

$$P_{\text{ВБПФ2}} = \frac{Np}{2m} + \log_2 N - 1. \quad (31)$$

Таблица 4

Порядок выполнения базовых операций в ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 4 для $D = 4$, $N = 8$, $p = 2$, $m = 2$

Ступень	АУ	Такты												
		1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0.0	0.1	0.2	0.3	1.0	1.1	1.2	1.3	×	×	×	×	×
	1	–	–	0.0	0.1	0.2	0.3	1.0	1.1	1.2	1.3	×	×	×
	2	–	–	–	–	–	0.0	0.1	0.2	0.3	1.0	1.1	1.2	1.3
1	0	2.0	2.1	2.2	2.3	3.0	3.1	3.2	3.3	×	×	×	×	×
	1	–	–	2.0	2.1	2.2	2.3	3.0	3.1	3.2	3.3	×	×	×
	2	–	–	–	–	–	2.0	2.1	2.2	2.3	3.0	3.1	3.2	3.3

Примечания:

– базовые операции, выполняемые над данными предыдущей выборки векторного процесса;

× базовые операции, выполняемые над данными последующей выборки векторного процесса.

При использовании алгоритмов 5 и 6 для вычисления векторного ДПФ организация вычислительного процесса в ППБПФ-процессоре зависит от способа параллельного выполнения базовых операций в алгоритме БПФ по основанию 2.

I. Алгоритм 5:

1. Параллельно выполняются m базовых операций с номерами, различающимися на $DN/2m$.

В этом случае при $m < N/2$ m базовых операций с номерами, различающимися на $N/2m$, выполняются последовательно для всех компонентов векторного процесса, т. е. одноименные базовые операции выполняются сразу для всех компонентов. При $m = N/2$ последовательно для всех компонентов выполняется полностью одноименная итерация алгоритма БПФ. Следует отметить, что для $m \leq N/2$ на каждой ступени ППБПФ-процессора одновременно выполняются базовые операции только одного компонента векторного процесса. При $m > N/2$ полностью выполняется одноименная итерация алгоритма БПФ для $2m/N$ компонентов. Таким образом, БПФ всех компонентов векторного процесса выполняются одновременно в отличие от первых двух случаев, где БПФ одного или p компонентов выполняются полностью от начала до конца, и только затем происходит переход к обработке следующих компонентов векторного процесса. Организация вычислительного процесса в ППБПФ-процессоре для рассматриваемого случая приведена в табл. 5 для $D = 4, N = 8, m = 2$.

Таблица 5

Порядок выполнения базовых операций в прямом ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 5 для $D = 4$, $N = 8$, $m = 2$. Одновременно выполняются m базовых операций с номерами, различающимися на $DN/2m$

Ступень	АУ	Такты													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0.0	1.0	2.0	3.0	0.1	1.1	2.1	3.1	×	×	×	×	×	×
	1	0.2	1.2	2.2	3.2	0.3	1.3	2.3	3.3	×	×	×	×	×	×
1	0	–	0.0	1.0	2.0	3.0	0.1	1.1	2.1	3.1	×	×	×	×	×
	1	–	0.2	1.2	2.2	3.2	0.3	1.3	2.3	3.3	×	×	×	×	×
2	0	–	–	–	–	–	–	0.0	1.0	2.0	3.0	0.1	1.1	2.1	3.1
	1	–	–	–	–	–	–	0.2	1.2	2.2	3.2	0.3	1.3	2.3	3.3

Примечания:

– базовые операции, выполняемые над данными предыдущей выборки векторного процесса;

× базовые операции, выполняемые над данными последующей выборки векторного процесса.

Оценка латентности при организации вычисления векторного ДПФ таким образом совпадает с соответствующей оценкой (27), оценки остальных параметров ППБПФ-процессора в этом случае определяются следующим образом:

– сложность структуры:

$$C_{\text{ВБПФЗ}} = C_{\text{АУ}} m \log_2 N + 2C_{\text{П}} D(N - 2m) + 2C_{\text{К}} m \log_2 \left(\frac{N}{2m}\right), \quad m \le N/2; \qquad (32)$$

$$C_{\text{ВБПФЗ}} = C_{\text{АУ}} m \log_2 N, m > N/2;$$

– время вычисления векторного ДПФ, тактов:

$$T_{\text{ВБПФЗ}} = D\left(\frac{N}{m} - 1\right) + \log_2 N - 1, m \le N/2; \qquad (33)$$

$$T_{\text{ВБПФЗ}} = \left(\frac{DN}{2m} - 1\right) + \log_2 N, m > N/2;$$

– задержка вычисления векторного ДПФ, тактов:

$$P_{\text{ВБПФЗ}} = D\left(\frac{N}{2m} - 1\right) + \log_2 N, m \le N/2; \qquad (34)$$

$$P_{\text{ВБПФЗ}} = \log_2 N, m > N/2.$$

2. Параллельно выполняются m последовательных базовых операций.

В этом случае при $m < D$ одновременно выполняются m одноименных базовых операций по одной для каждого из m последовательно расположенных компонентов векторного процесса. При этом одноименные базовые операции выполняются для всех компонентов векторного процесса последовательно группами по m компонентов. При $m = D$ одноименные базовые операции выполняются одновременно для всех компонентов векторного процесса. Если $m > D$, то одновременно выполняется по m/D последовательных базовых операций для всех компонентов векторного процесса. В последних двух случаях, когда $m \ge D$, порядок выполнения базовых операций такой же, как и при вычислении по алгоритму 3 при $m \ge p = D$. Организация вычислительного процесса в прямом ППБПФ-процессоре при вычислении векторного ДПФ по алгоритму 5 с одновременным выполнением m последовательных базовых операций приведена в табл. 6 для $D = 4$, $N = 8$, $m = 2$.

Таблица 6

Порядок выполнения базовых операций в прямом ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 5 для $D = 4, N = 8, m = 2$. Одновременно выполняются m последовательных базовых операций

Ступень	АУ	Такты															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0.0	2.0	0.1	2.1	0.2	2.2	0.3	2.3	×	×	×	×	×	×	×	×
	1	1.0	3.0	1.1	3.1	1.2	3.2	1.3	3.3	×	×	×	×	×	×	×	×
1	0	–	–	–	–	–	0.0	2.0	0.1	2.1	0.2	2.2	0.3	2.3	×	×	×
	1	–	–	–	–	–	1.0	3.0	1.1	3.1	1.2	3.2	1.3	3.3	×	×	×
2	0	–	–	–	–	–	–	–	–	0.0	2.0	0.1	2.1	0.2	2.2	0.3	2.3
	1	–	–	–	–	–	–	–	–	1.0	3.0	1.1	3.1	1.2	3.2	1.3	3.3

Примечания:

– базовые операции, выполняемые над данными предыдущей выборки векторного процесса;

× базовые операции, выполняемые над данными последующей выборки векторного процесса.

Оценка латентности при организации вычисления векторного ДПФ рассмотренным способом совпадает с соответствующей оценкой (27), оценки остальных параметров ППБПФ-процессора в этом случае определяются следующим образом:

– сложность структуры:

$$C_{\text{ВБПФ4}} = C_{\text{АУ}} m \log_2 N + 2C_{\text{П}} D(N-2) + 2C_{\text{К}} m \log_2\left(\frac{N}{2}\right), \quad m \le D;$$
$$C_{\text{ВБПФ4}} = C_{\text{АУ}} m \log_2 N + 2C_{\text{П}}(DN - 2m) + 2C_{\text{К}} m \log_2\left(\frac{DN}{2m}\right), \quad m > D; \tag{35}$$

– время вычисления векторного ДПФ, тактов:

$$T_{\text{ВБПФ4}} = D(N-1) + \log_2 N - 1, m \le D;$$
$$T_{\text{ВБПФ4}} = \frac{DN}{m} + \log_2 N - 2, m > D; \tag{36}$$

– задержка вычисления векторного ДПФ, тактов:

$$P_{\text{ВБПФ4}} = \frac{D(N-2)}{2m} + \log_2 N, m \le D;$$
$$P_{\text{ВБПФ4}} = \frac{DN}{2m} + \log_2 N - 1, m > D. \tag{37}$$

II. Алгоритм 6:

1. Параллельно выполняются базовые операции с номерами, различающимися на $DN/2m$.

В этом случае при $m < D$ одновременно выполняются m одноименных базовых операций по одной для каждого из m компонентов векторного процесса, различающихся номерами на D/m. При $m = D$ одновременно выполняется по одной одноименной базовой операции для всех компонентов векторного процесса. Причем порядок выполнения базовых операций при $m \le D$ такой же, как и при вычислении по алгоритму 4 при $m = p \le D$. Если $m > D$, то одновременно выполняется по m/D базовых операций с номерами, различающимися на $N/2m$, для всех компонентов векторного процесса. В этом случае порядок выполнения базовых операций также совпадает с порядком, имеющим место при вычислении по алгоритму 4, но при $m > p = D$. Таким образом, с точки зрения организации вычисления векторного ДПФ в ППБПФ-процессоре алгоритмы 4 и 6 с параллельным выполнением m базовых операций, различающихся номерами на $DN/2m$, эквивалентны при $m = p \le D$ и $m > p = D$. Организация вычислительного процесса в обратном ППБПФ-процессоре при вычислении векторного ДПФ по алгоритму 6 с параллельным выполнением базовых операций, различающихся номерами на $DN/2m$, приведена в табл. 7 для $D = 4$, $N = 8, m = 2$.

Оценка латентности при организации вычисления векторного ДПФ рассмотренным способом совпадает с соответствующей оценкой (27), оценки остальных параметров ППБПФ-процессора в этом случае определяются следующим образом:

– сложность структуры:

$$\begin{aligned} &C_{\text{ВБПФ5}} = m\left(C_{\text{АУ}}\log_2 N + 2C_{\text{П}}(N-2) + 2C_{\text{К}}\log_2\left(\frac{N}{2}\right)\right),\\ &m \le D;\\ &C_{\text{ВБПФ5}} = C_{\text{АУ}}m\log_2 N + 2C_{\text{П}}(DN-2m) + 2C_{\text{К}}m\log_2\left(\frac{DN}{2m}\right),\\ &m > D; \end{aligned} \tag{38}$$

– время вычисления векторного ДПФ, тактов:

$$\begin{aligned} &T_{\text{ВБПФ5}} = \frac{N}{2}\left(\frac{D}{m}+1\right) + \log_2 N - 1, m \le D;\\ &T_{\text{ВБПФ5}} = \frac{DN}{m} + \log_2 N - 2, m > D; \end{aligned} \tag{39}$$

– задержка вычисления векторного ДПФ, тактов:

$$\begin{aligned} &P_{\text{ВБПФ5}} = \frac{N}{2} - 1 + \log_2 N, m \le D;\\ &P_{\text{ВБПФ4}} = \frac{DN}{2m} + \log_2 N - 1, m > D. \end{aligned} \tag{40}$$

Таблица 7

Порядок выполнения базовых операций в обратном ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 6 для $D = 4, N = 8, m = 2$. Одновременно выполняются m базовых операций с номерами, различающимися на $DN/2m$

Ступень	АУ	Такты												
		1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0.0	0.1	0.2	0.3	1.0	1.1	1.2	1.3	×	×	×	×	×
	1	2.0	2.1	2.2	2.3	3.0	3.1	3.2	3.3	×	×	×	×	×
1	0	–	–	0.0	0.1	0.2	0.3	1.0	1.1	1.2	1.3	×	×	×
	1	–	–	2.0	2.1	2.2	2.3	3.0	3.1	3.2	3.3	×	×	×
2	0	–	–	–	–	–	0.0	0.1	0.2	0.3	1.0	1.1	1.2	1.3
	1	–	–	–	–	–	2.0	2.1	2.2	2.3	3.0	3.1	3.2	3.3

Примечания:

– базовые операции, выполняемые над данными предыдущей выборки векторного процесса;

× базовые операции, выполняемые над данными последующей выборки векторного процесса.

2. Параллельно выполняются m последовательных базовых операций.

В этом случае при $m < N/2$ последовательно выполняются группы из m базовых операций БПФ одного компонента векторного процесса. При этом внутри группы базовые операции выполняются параллельно. После вычисления всех базовых операций БПФ обрабатываемого компонента происходит переход к выполнению базовых операций БПФ следующего компонента векторного процесса. При $m = N/2$ одновременно выполняются все базовые операции соответствующей итерации алгоритма БПФ для одного компонента векторного процесса. Порядок выполнения базовых операций при $m \le N/2$ такой же, как и при вычислении векторного ДПФ по алгоритму 2. Если $m > N/2$, то параллельно выполняются все базовые операции одной итерации алгоритма БПФ одновременно для $2m/N$ компонентов векторного процесса с последовательными номерами. В этом случае порядок выполнения базовых операций такой же, как и при вычислении по алгоритму 3 или 4 при $m > N/2$, $p = 2m/N$. Таким образом, с точки зрения организации вычисления векторного ДПФ в ППБПФ-процессоре с параллельным выполнением $m \le N/2$ последовательных базовых операций алгоритмы 2 и 6 эквивалентны. Организация вычислительного процесса в обратном ППБПФ-процессоре при вычислении векторного ДПФ по алгоритму 6 с одновременным выполнением m последовательных базовых операций при $D = 4, N = 8, m = 2$ приведена в табл. 8.

Таблица 8

Порядок выполнения базовых операций в обратном ППБПФ-процессоре при вычислении векторного ДПФ в соответствии с алгоритмом 6 для $D = 4$, $N = 8$ и $m = 2$. Одновременно выполняются m последовательных базовых операций

Ступень	АУ	Такты										
		1	2	3	4	5	6	7	8	9	10	11
0	0	0.0	0.2	1.0	1.2	2.0	2.2	3.0	3.2	×	×	×
	1	0.1	0.3	1.1	1.3	2.1	2.3	3.1	3.3	×	×	×
1	0	–	0.0	0.2	1.0	1.2	2.0	2.2	3.0	3.2	×	×
	1	–	0.1	0.3	1.1	1.3	2.1	2.3	3.1	3.3	×	×
2	0	–	–	–	0.0	0.2	1.0	1.2	2.0	2.2	3.0	3.2
	1	–	–	–	0.1	0.3	1.1	1.3	2.1	2.3	3.1	3.3

Примечания:

– базовые операции, выполняемые над данными предыдущей выборки векторного процесса;

× базовые операции, выполняемые над данными последующей выборки векторного процесса.

Оценка латентности при организации вычисления векторного ДПФ рассмотренным способом совпадает с соответствующей оценкой (27), оценки остальных параметров ППБПФ-процессора в этом случае определяются следующим образом:

– сложность структуры:

$$C_{\text{ВБПФ6}} = C_{\text{АУ}} m \log_2 N + 2C_{\text{П}}(N - 2m) + 2C_{\text{К}} m \log_2 \left(\frac{N}{2m}\right), \quad m \le N/2; \qquad (41)$$

$$C_{\text{ВБПФ6}} = C_{\text{АУ}} m \log_2 N, m > N/2;$$

– время вычисления векторного ДПФ, тактов:

$$T_{\text{ВБПФ6}} = \frac{N}{2m}(D + 1) + \log_2 N - 2, m \le N/2; \qquad (42)$$

$$T_{\text{ВБПФ6}} = \frac{DN}{2m} + \log_2 N - 1, m > N/2;$$

– задержка вычисления векторного ДПФ, тактов:

$$P_{\text{ВБПФ6}} = \frac{N}{2m} - 1 + \log_2 N, m \le N/2; \qquad (43)$$

$$P_{\text{ВБПФ6}} = \log_2 N, m > N/2.$$

Анализ различных структур ППБПФ-процессора, получаемых при организации вычисления векторного ДПФ в соответствии с алгоритмами 1–6, показывает, что при одинаковом значении латентности все структуры имеют также одинаковое общее число вычислительных элементов и различаются только суммарным объемом локальной памяти и количеством коммутаторов, т. е. общим объемом и организацией промежуточной памяти. С этой точки зрения

наилучшими являются алгоритмы 1, 2 и 6 при параллельном выполнении групп последовательных базовых операций, т. к. они приводят к минимальному общему объему промежуточной памяти и, следовательно, минимальной структурной сложности ППБПФ-процессора. При этом предпочтение следует отдать алгоритму 6, т. к. в этом случае вычисление векторного ДПФ сводится к выполнению одного усеченного алгоритма БПФ одномерного массива, что приводит к более простой общей организации вычислительного процесса в ППБПФ-процессоре. Алгоритм 5 является самым неэффективным, т. к. при организации вычисления векторного ДПФ по этому алгоритму при любом из двух возможных способов параллельного выполнения базовых операций общий объем промежуточной памяти в ППБПФ-процессоре оказывается практически в D раз большим, чем при использовании алгоритмов 1, 2, 6. Кроме того, этот алгоритм характеризуется наибольшими значениями времени и задержки вычисления векторного ДПФ ППБПФ-процессором.

5. СТРУКТУРНОЕ ОПИСАНИЕ ПАРАЛЛЕЛЬНО-ПОТОЧНЫХ БПФ-ПРОЦЕССОРОВ

5.1. Язык структурного описания параллельно-поточных БПФ-процессоров

Параллельно-поточную структуру можно рассматривать как применение к входному потоку данных последовательности операторов, которые преобразуют одномерный массив данных в двумерный и выполняют базовую операцию алгоритма БПФ по основанию 2 [5]. За основу при описании ППБПФ-структуры приняты система обозначений и подход, предложенные в работах [5, 6]. В пособии рассматривается подход, модифицированный и расширенный авторами применительно к описанию структуры ППБПФ-процессора для вычисления векторного ДПФ [7, 8, 9].

Определим индекс элемента двумерного массива в виде

$$[x, y] = [\{a_s a_{s-1} \dots a_1\}, \{b_t b_{t-1} \dots b_1\}], \quad (44)$$

где $x = a_s 2^{s-1} + a_{s-1} 2^{s-2} + \dots + a_1$ – номер столбца; $y = b_t 2^{t-1} + + b_{t-1} 2^{t-2} + \dots + b_1$ – номер строки; $b_i = \{0, 1\}$, определяющие положение данного элемента в массиве.

Операторы, описывающие параллельно-поточную структуру, определяются по их воздействию на индексы элементов данных и задают некоторую перестановку элементов в массиве.

Оператор $\mu_{(j,i)}$ разбивает каждую строку исходного массива на 2^j строки, обрабатывая входные данные блоками по 2^i столбцов, и определяется следующим образом:

$$\begin{aligned} &\mu_{(j,i)}[x, y] = \mu_{(j,i)}[\{a_s a_{s-1} \dots a_1\}, \{b_t b_{t-1} \dots b_1\}] \\ &= \left[\{a_s \dots a_{i+1} a_{i-j} \dots a_1\}, \{b_t \dots b_1 a_i \dots a_{i-j+1}\}\right], \\ &\mu_{(j,i)}^{-1}[x, y] = \left[\{a_s \dots a_{i-j+1} b_j \dots b_1 a_{i-j} \dots a_1\}, \{b_t \dots b_{j+1}\}\right], \\ &\mu_{(i)} = \mu_{(1,i)}, \mu_{(i)}^{-1} = \mu_{(1,i)}^{-1}, j \le i \le s. \end{aligned} \quad (45)$$

На рис. 5 показан пример применения оператора $\mu_{(1,2)}$ к вектору $z = [z_0, z_1, \dots, z_7]$, состоящему из восьми элементов.

$$(z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0) \xrightarrow{\mu_{(1,2)}} \begin{pmatrix} z_5 & z_4 & z_1 & z_0 \\ z_7 & z_6 & z_3 & z_2 \end{pmatrix}$$

Рис. 5. Применение оператора $\mu_{(1,2)}$ к вектору $z = [z_0, z_1, \dots, z_7]$

Графическое представление воздействия операторов $\mu_{(i,j)}$ и $\mu_{(i,j)}^{-1}$ на битовое представление индексов элемента двумерного массива $[x, y]$ показано на рис. 6.

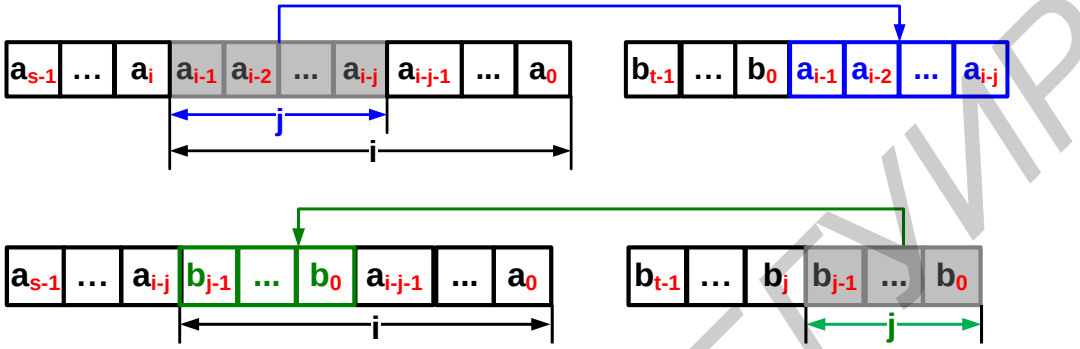

Рис. 6. Воздействие операторов $\mu_{(i,j)}$ и $\mu_{(i,j)}^{-1}$ на битовое представление индексов элемента двумерного массива $[x, y]$

Оператор B описывает базовую операцию алгоритма БПФ по основанию 2. Действия выполняются над каждой парой данных столбца, при этом результат базовой операции с суммой помещается в строку с меньшим значением индекса, а результат базовой операции с разностью – в строку с большим индексом.

Оператор $\delta_{(i)}$ разбивает каждую строку исходного массива данных на 2^i строки по следующему правилу:

$$\delta_{(i)}[x, y] = [\{a_s \dots a_{i+1}\}, \{b_t \dots b_1 a_i \dots a_1\}], i \le s. \quad (46)$$

Оператор $T_{(i,j)}$ выполняет перестановку элементов двумерного массива данных внутри блоков, состоящих из строк и столбцов, по следующему правилу:

$$T_{(i,j)}[x, y] = \left[\{a_s \dots a_{j+1} b_i \dots b_1\}, \{b_t \dots b_{i+1} a_j \dots a_1\}\right], \quad (47)$$
$$j \le s, i \le t.$$

Оператор $R_{(i)}$ через каждые 2^i строки меняет местами четные и нечетные строки в блоках данных размером 2^i строки и определяется следующим образом:

$$R_{(i)}[x, y] = [x, \{b_t \dots b_2 (b_1 \oplus b_{i+1})\}], i \le t, \quad (48)$$

где $\oplus$ означает операцию ИСКЛЮЧАЮЩЕЕ ИЛИ.

Оператор $\varphi_{i,j}$ осуществляет перестановку данных внутри каждого столбца массива данных по следующему правилу:

$$\varphi_{(i,j)}[x, y] = \left[x, \{b_t \dots b_{i+j+1} (b_{i+j} \oplus a_i) b_{i+j-1} \dots b_1\}\right], \quad (49)$$
$$\varphi_{(i)} = \varphi_{(i,0)}, i \le s, j \le t - s.$$

Относительно рассмотренных операторов справедливо следующее тождество:

$$\delta_{(j)} T_{(j,i-j)} = \mu_{(i-j,i)}, j \le i, \quad (50)$$

доказательство которого приведено в работе [5].

При описании параллельно-поточной структуры будем придерживаться соглашения, согласно которому операторы записываются слева направо, т. е. $\pi_1\pi_2[x,y] = \pi_2(\pi_1[x,y])$. Это соглашение связано с принятым направлением перемещения данных в структуре и делается для того, чтобы последовательности операторов точно соответствовали изображениям структур.

Прямая поточная структура вычисления 2^l-точечного БПФ описывается следующим образом [5]:

$$FFT^{\text{п}}_{(l)} = \mu_{(l)} B \mu^{-1}_{(l)} \mu_{(l-1)} B \mu^{-1}_{(l-1)} \dots \mu_{(1)} B \mu^{-1}_{(1)}. \quad (51)$$

Операторы $\mu_{(l)}$ и $\mu^{-1}_{(1)}$ описывают буферную память соответственно на входе и выходе поточной структуры, которая на входе отображает последовательный поток данных на две входные линии, а на выходе осуществляет обратное преобразование. Комбинация операторов $\mu^{-1}_{(l-i+1)}\mu_{(l-i)}, i = 1, \dots, l-1$ функционально соответствует промежуточной памяти объемом $2 \times 2^{l-i-1}$ слов между $(i-1)$-й и i-й ступенями поточной структуры, оператор B – арифметическому устройству базовой операции алгоритма БПФ по основанию 2. Так как речь идет только о топологии структуры БПФ-процессора, то в описаниях структур игнорируется тот факт, что АУ БО, описываемые оператором B, используют различные поворачивающие множители. Узнать, какие поворачивающие множители и к каким данным необходимо применять можно непосредственно из описания алгоритма БПФ.

Аналогичным образом представляется обратная поточная структура вычисления 2^l-точечного БПФ:

$$FFT^{\text{о}}_{(l)} = \mu_{(1)} B \mu^{-1}_{(1)} \mu_{(2)} B \mu^{-1}_{(2)} \dots \mu_{(l)} B \mu^{-1}_{(l)}, \quad (52)$$

где операторы $\mu_{(1)}$ и $\mu^{-1}_{(l)}$ описывают соответственно входную и выходную буферные памяти, а комбинация операторов $\mu^{-1}_{(i)}\mu_{(i+1)}, i = 1, \dots, l-1$ функционально соответствует промежуточной памяти объемом $2 \times 2^{i-1}$ слов между $(i-1)$-й и i-й ступенями структуры.

Описание прямой параллельно-поточной структуры процессора БПФ с $m = 2^r$ вычислительными элементами (ВЭ) на ступени, выполняющими одновременно m последовательных базовых операций, получается путем последовательной подстановки в выражение (51) тождественного преобразования $\delta_{(i)} T_{(i,l-r)} \mu^{-1}_{(l-r,l-r+i)}$ для $i = r, 2r-l, 3r-2l, \dots$ (см. выражение (50)) до тех пор, пока $l - r < i$, с последующей декомпозицией операторов $T_{(i,l-r)}$ [5]. В результате прямая параллельно-поточная структура вычисления 2^l-точечного БПФ с $m = 2^r$ ВЭ на ступени, выполняющими m последовательных базовых операций, описывается следующим образом:

$$\begin{aligned} FFT^{\text{п}}_{(l,r)} = {} & \delta_{(r)} \mu_{(l-r)} B \mu^{-1}_{(l-r)} \mu_{(l-r-1)} B \dots \\ & \dots B \mu^{-1}_{(2)} \mu_{(1)} B R_{(r)} \mu^{-1}_{(r)} \varphi_{(r)} \mu_{(r)} R_{(r)} B \dots \\ & \dots B R_{(1)} \mu^{-1}_{(1)} \varphi_{(1)} \mu_{(1)} R_{(1)} B \mu^{-1}_{(1)} \mu^{-1}_{(r,r+1)}. \end{aligned} \quad (53)$$

В выражении (53) комбинация операторов $\delta_{(r)}\mu_{(l-r)}$ разбивает входную последовательность данных на 2^{r+1} строки длиной 2^{l-r-1} и функционально соответствует входной буферной памяти, а операторы $\mu_{(1)}^{-1}\mu_{(r,r+1)}^{-1}$ выполняют обратное преобразование и описывают выходную буферную память. Последовательность операторов $R_{(i)}\mu_{(i)}^{-1}\varphi_{(i)}\mu_{(i)}R_{(i)}$ описывает структуру связей между двумя соседними ступенями ППБПФ-процессора, соединенными непосредственно без промежуточной памяти.

Аналогичным образом представляется обратная параллельно-поточная структура вычисления 2^l-точечного БПФ с $m = 2^r$ ВЭ на ступени, выполняющими m последовательных базовых операций:

$$\begin{aligned} FFT_{(l,r)}^{\text{о}} = {} & \mu_{(r,r+1)}\mu_{(1)}BR_{(1)}\mu_{(1)}^{-1}\varphi_{(1)}\mu_{(1)}R_{(1)}B \ldots \\ & \ldots BR_{(r)}\mu_{(r)}^{-1}\varphi_{(r)}\mu_{(r)}R_{(r)}B\mu_{(1)}^{-1}\mu_{(2)}B \ldots \\ & \ldots B\mu_{(l-r-1)}^{-1}\mu_{(l-r)}B\mu_{(l-r)}^{-1}\mu_{(r,r)}^{-1}, \end{aligned} \quad (54)$$

где комбинации операторов $\mu_{(r,r+1)}\mu_{(1)}$ и $\mu_{(l-r)}^{-1}\mu_{(r,r)}^{-1}$ описывают соответственно входную и выходную буферные памяти.

Таким образом, рассмотренные выше операторы позволяют функционально описать параллельно-поточную структуру. Однако такое описание получается громоздким и неудобным для использования, т. к. в нем присутствуют промежуточные перестановки данных, и последовательность операторов в описании не соответствует однозначно структуре процессора. Для обеспечения взаимно однозначного соответствия последовательности операторов в описании структуре ППБПФ-процессора вводятся два дополнительных оператора.

Оператор $M_{(i)}$ выполняет следующую перестановку элементов двумерного массива данных:

$$M_{(i)}[x,y] = [\{a_s \ldots a_{i+1}b_1a_{i-1} \ldots a_1\},\{b_t \ldots b_2a_i\}], i \le s. \quad (55)$$

Оператор $\beta_{(i)}$ переставляет строки исходного массива данных по следующему правилу:

$$\beta_{(i)}[x,y] = [x,\{b_t \ldots b_{i+2}b_1b_i \ldots b_2b_{i+1}\}], i \le t. \quad (56)$$

Тогда справедливо следующее утверждение.

Утверждение 2. Параллельно-поточная структура вычисления 2^l-точечного БПФ с $m = 2^r$ вычислительными элементами на ступени, выполняющими m последовательных базовых операций, может быть представлена следующим образом:

– прямая ППБПФ-структура:

$$\begin{aligned} FFT_{(l,r)}^{\text{п}} = {} & \delta_{(r)}\mu_{(l-r)}BM_{(l-r-1)}B \ldots BM_{(1)}B\beta_{(r)}B \ldots \\ & \ldots B\beta_{(1)}B\mu_{(1)}^{-1}\mu_{(r,r+1)}^{-1}; \end{aligned} \quad (57)$$

– обратная ППБПФ-структура:

$$\begin{aligned} FFT_{(l,r)}^{\text{о}} = {} & \mu_{(r,r+1)}\mu_{(1)}B\beta_{(1)}B \ldots B\beta_{(r)}BM_{(1)}B \ldots \\ & \ldots BM_{(l-r-1)}B\mu_{(l-r)}^{-1}\mu_{(r,r)}^{-1}. \end{aligned} \quad (58)$$

Доказательство. Нетрудно показать, что

$$M_{(i)} = \mu_{(i+1)}^{-1}\mu_{(i)}, \quad (59)$$

$$M_{(i)} = \mu_{(i+1)}^{-1}\mu_{(i)}, \quad (60)$$

$$\beta_{(i)} = R_{(i)}\mu_{(i)}^{-1}\varphi_{(i)}\mu_{(i)}R_{(i)}. \quad (61)$$

Действительно,

$$\mu_{(i+1)}^{-1}\mu_{(i)}[x,y] = \mu_{(i)}[\{a_s \dots a_{i+1}b_1a_i \dots a_1\},\{b_t \dots b_2\}] =$$
$$= [\{a_s \dots a_{i+1}b_1a_{i-1} \dots a_1\},\{b_t \dots b_2a_i\}] = M_{(i)},$$

и тождество (59) доказано.

Аналогично доказывается тождество (60):

$$\mu_{(i)}^{-1}\mu_{(i+1)}[x,y] = \mu_{(i+1)}[\{a_s \dots a_ib_1a_{i-1} \dots a_1\},\{b_t \dots b_2\}] =$$
$$= [\{a_s \dots a_{i+1}b_1a_{i-1} \dots a_1\},\{b_t \dots b_2a_i\}] = M_{(i)}.$$

Следующим образом доказывается тождество (61):

$$R_{(i)}\mu_{(i)}^{-1}\varphi_{(i)}\mu_{(i)}R_{(i)}[x,y] = \mu_{(i)}^{-1}\varphi_{(i)}\mu_{(i)}R_{(i)}[x,\{b_t \dots b_2(b_1 \oplus b_{i+1})\}]$$
$$== \varphi_{(i)}\mu_{(i)}R_{(i)}[\{a_s \dots a_i(b_1 \oplus b_{i+1})a_{i-1} \dots a_1\},\{b_t \dots b_2\}] =$$
$$= \mu_{(i)}R_{(i)}[\{a_s \dots a_i(b_1 \oplus b_{i+1})a_{i-1} \dots a_1\},$$
$$\{b_t \dots b_{i+2}(b_{i+1} \oplus (b_1 \oplus b_{i+1}))b_i \dots b_2\}].$$

Выражение $b_{i+1} \oplus (b_1 \oplus b_{i+1}) = b_{i+1} \oplus b_1 \oplus b_{i+1} = b_1$, тогда

$$R_{(i)}\mu_{(i)}^{-1}\varphi_{(i)}\mu_{(i)}R_{(i)}[x,y] =$$
$$= \mu_{(i)}R_{(i)}[\{a_s \dots a_i(b_1 \oplus b_{i+1})a_{i-1} \dots a_1\},\{b_t \dots b_{i+2}b_1b_i \dots b_2\}] =$$
$$= R_{(i)}[\{a_s \dots a_ia_{i-1} \dots a_1\},\{b_t \dots b_{i+2}b_1b_i \dots b_2(b_1 \oplus b_{i+1})\}] =$$
$$= [x,\{b_t \dots b_{i+2}b_1b_i \dots b_2((b_1 \oplus b_{i+1}) \oplus b_1)\}].$$

Выражение $(b_1 \oplus b_{i+1}) \oplus b_1 = b_1 \oplus b_{i+1} \oplus b_1 = b_{i+1}$ и, следовательно, $R_{(i)}\mu_{(i)}^{-1}\varphi_{(i)}\mu_{(i)}R_{(i)}[x,y] = [x,\{b_t \dots b_{i+2}b_1b_i \dots b_2b_{i+1}\}] = \beta_{(i)}$, что и требовалось доказать.

Справедливость тождеств (59)–(61) доказывает утверждение 2 в целом.

Из утверждения 2 следует, что оператор $M_{(i)}$ функционально описывает промежуточную память объемом $2 \times 2^{i-1}$ параллельно-поточной структуры, а оператор $\beta_{(i)}$ – структуру связей между двумя соседними ступенями ППБПФ-процессора, соединенными непосредственно без промежуточной памяти.

С помощью введенного оператора $M_{(i)}$ поточная структура вычисления 2^l-точечного БПФ будет описываться следующим образом:

– прямая поточная структура:

$$FFT_{(l)}^{\text{п}} = \mu_{(l)}BM_{(l-1)}B \dots BM_{(1)}B\mu_{(1)}^{-1}; \quad (62)$$

– обратная поточная структура:

$$FFT_{(l)}^{\text{о}} = \mu_{(1)}BM_{(1)}B \dots BM_{(l-1)}B\mu_{(l)}^{-1}. \quad (63)$$

Для описания параллельно-поточной структуры с одновременным выполнением m базовых операций, различающихся номерами на $N/2m$, вводится еще один дополнительный оператор $\sigma_{(i)}$, который переставляет строки исходного массива данных по следующему правилу:

$$\sigma_{(i)}[x,y] = [x,\{b_t \dots b_{i+2} b_i \dots b_1 b_{i+1}\}],$$
$$\sigma_{(i)}^{-1}[x,y] = [x,\{b_t \dots b_{i+2} b_1 b_{i+1} \dots b_2\}], \quad (64)$$
$$i < t.$$

С помощью этого оператора прямая параллельно-поточная структура вычисления 2^l-точечного БПФ с $m = 2^r$ ВЭ на ступени, выполняющими m базовых операций с номерами, различающимися на $N/2m$ $(N = 2^l)$, описывается следующим образом:

$$FFT_{(l,r)}^{\text{п}} = \mu_{(r,l)}\mu_{(l-r)}\sigma_{(r)} B \beta_{(r)} B \dots B\beta_{(1)} B M_{(l-r-1)} B \dots \\ \dots B M_{(1)} B \mu_{(1)}^{-1}\mu_{(r,l)}^{-1}. \quad (65)$$

В выражении (58) комбинация операторов $\mu_{(r,l)}\mu_{(l-r)}\sigma_{(r)}$ описывает входную буферную память, которая отображает последовательный входной поток данных на 2^{r+1} линии, а $\mu_{(1)}^{-1}\mu_{(r,l)}^{-1}$ – выходную буферную память, которая выполняет обратное преобразование данных на выходе.

Аналогичным образом представляется обратная параллельно-поточная структура вычисления 2^l-точечного БПФ с $m = 2^r$ ВЭ на ступени, выполняющими m базовых операций с номерами, различающимися на $N/2m$:

$$FFT_{(l,r)}^{\text{о}} = \mu_{(r,l)}\mu_{(1)} B M_{(1)} B \dots B M_{(l-r-1)} B \beta_{(1)} B \dots \\ \dots B \beta_{(r)} B \sigma_{(r)}^{-1}\mu_{(l-r)}^{-1}\mu_{(r,l)}^{-1}, \quad (66)$$

где комбинации операторов $\mu_{(r,l)}\mu_{(1)}$ и $\sigma_{(r)}^{-1}\mu_{(l-r)}^{-1}\mu_{(r,l)}^{-1}$ описывают соответственно входную и выходную буферные памяти.

Оператор $\sigma_{(i)}$ позволяет также другим образом представить описание (57), (58) параллельно-поточной структуры вычисления 2^l-точечного БПФ с $m = 2^r$ ВЭ на ступени, выполняющими m последовательных базовых операций:

– прямая ППБПФ-структура:

$$FFT_{(l,r)}^{\text{п}} = \delta_{(r)}\mu_{(l-r)} B M_{(l-r-1)} B \dots B M_{(1)} B \beta_{(r)} B \dots \\ \dots B \beta_{(1)} B \sigma_{(r)}\mu_{(1)}^{-1}\mu_{(r,r)}^{-1}; \quad (67)$$

– обратная ППБПФ-структура:

$$FFT_{(l,r)}^{\text{о}} = \delta_{(r)}\mu_{(1)}\sigma_{(r)}^{-1} B \beta_{(1)} B \dots B \beta_{(r)} B M_{(1)} B \dots \\ \dots B M_{(l-r-1)} B \mu_{(l-r)}^{-1}\mu_{(r,r)}^{-1}, \quad (68)$$

где комбинации операторов $\delta_{(r)}\mu_{(l-r)}$ и $\delta_{(r)}\mu_{(1)}\sigma_{(r)}^{-1}$ описывают входную буферную память соответственно в прямой и обратной ППБПФ-структурах, а $\sigma_{(r)}\mu_{(1)}^{-1}\mu_{(r,r)}^{-1}$ и $\mu_{(l-r)}^{-1}\mu_{(r,r)}^{-1}$ – выходную буферную память.

В таком описании ППБПФ-структуры с одновременным выполнением m последовательных базовых операций в отличие от (57) и (58) на входе и выходе используются операторы, выполняющие взаимно обратные преобразования (перестановки), т. к. легко показать, что $\delta_{(r)} = \mu_{(r,r)}$.

5.2. Формализованное описание структуры параллельно-поточных БПФ-процессоров

Описание структуры параллельно-поточного процессора БПФ для обработки векторных процессов определяется алгоритмом, который используется для вычисления векторного ДПФ.

При организации вычисления векторного ДПФ в соответствии с алгоритмами 1 и 2 компоненты размером $N = 2^l$ векторного процесса обрабатываются независимо друг от друга последовательно одним ППБПФ-процессором с $m = 2^r$ вычислительными элементами на ступени. Такой процессор содержит $l = \log_2 N$ ступеней, и его структура описывается одним из выражений (65), (66), (67) или (68). Так же просто описывается структура параллельно-поточного процессора и при использовании алгоритмов 3 и 4. В этом случае вычисляются БПФ одновременно $p = 2^i, i = 1, \dots, \log_2 D$ компонентов векторного процесса на p ППБПФ-процессорах, причем каждый компонент обрабатывается на своем процессоре. При этом структура каждого ППБПФ-процессора так же, как и в первом случае, описывается одним из выражений (65), (66), (67) или (68).

При организации вычисления векторного ДПФ по алгоритмам 5 и 6 $D = 2^q$ компонентов размером $N = 2^l$ векторного процесса структурируются в виде одномерного массива размером $DN = 2^{q+l}$, который обрабатывается одним ППБПФ-процессором с m вычислительными элементами на ступени. Однако в этом случае, в отличие от первых двух, процессор содержит не все $q + 1$ ступеней, как это требуется для вычисления БПФ массива размером DN, а только первые l ступеней. Поэтому описание структуры такого ППБПФ-процессора может быть получено из структурного описания (65), (66), (67) или (68) полного процессора путем его соответствующего усечения.

Утверждение 3. Структура параллельно-поточного процессора БПФ для вычисления ДПФ векторного процесса, состоящего из $D = 2^q$ компонентов размером $N = 2^l$, по алгоритмам 5 и 6 при $m = 2^r$ вычислительных элементах на ступени может быть представлена следующим образом:

1) прямой ППБПФ-процессор:

– одновременно выполняются m последовательных базовых операций:

$$\begin{aligned} FFT^{\text{п}}_{(q,l,r)1} &= \delta_{(r)}\mu_{(q+l-r)}BM_{(q+l-r-1)}B \dots \\ &\dots BM_{(q-r+1)}B\mu^{-1}_{(q-r+1)}\mu^{-1}_{(r,r)}, r \le q; \\ FFT^{\text{п}}_{(q,l,r)1} &= \delta_{(r)}\mu_{(q+l-r)}BM_{(q+l-r-1)}B \dots BM_{(1)}B\beta_{(r)}B \dots \\ &\dots B\beta_{(q+1)}B\sigma^{-1}_{(q)}\sigma_{(r)}\mu^{-1}_{(1)}\mu^{-1}_{(r,r)}, r > q; \end{aligned} \quad (69)$$

– одновременно выполняются базовые операции с номерами, различающимися на $\mathrm{DN}/2m$:

$$\begin{aligned} FFT^{\text{п}}_{(q,l,r)2} &= \mu_{(r,q+l)}\mu_{(q+l-r)}\sigma_{(r)}B\beta_{(r)}B \dots \\ &\dots B\beta_{(1)}BM_{(q+l-r-1)}B \dots \\ &\dots BM_{(q+1)}B\mu^{-1}_{(q+1)}\mu^{-1}_{(r,q+l)}, r < l-1; \\ FFT^{\text{п}}_{(q,l,r)2} &= \mu_{(r,q+l)}\mu_{(q+l-r)}\sigma_{(r)}B\beta_{(r)}B \dots \end{aligned} \quad (70)$$

$$\ldots B\beta_{(r-l+2)}B\sigma^{-1}_{(r-l+1)}\mu^{-1}_{(q+l-r)}\mu^{-1}_{(r,q+l)}, r \geq l-1;$$

2) обратный ППБПФ-процессор:

– одновременно выполняются m последовательных базовых операций:

$$\begin{aligned} FFT^{o}_{(q,l,r)1} &= \delta_{(r)}\mu_{(1)}\sigma^{-1}_{(r)}B\beta_{(1)}B \ldots B\beta_{(r)}BM_{(1)}B \ldots \\ &\ldots BM_{(l-r-1)}B\mu^{-1}_{(l-r)}\mu^{-1}_{(r,r)}, r < l-1; \\ FFT^{o}_{(q,l,r)1} &= \delta_{(r)}\mu_{(1)}\sigma^{-1}_{(r)}B\beta_{(1)}B \\ &\ldots B\beta_{(l-1)}B\sigma^{-1}_{(l-1)}\sigma_{(r)}\mu^{-1}_{(1)}\mu^{-1}_{(r,r)}, r \geq l-1; \end{aligned} \quad (71)$$

– одновременно выполняются базовые операции с номерами, различающимися на $\mathrm{DN}/2m$:

$$\begin{aligned} FFT^{o}_{(q,l,r)2} &= \mu_{(r,q+l)}\mu_{(1)}BM_{(1)}B \ldots \\ &\ldots BM_{(l-1)}B\mu^{-1}_{(l)}\mu^{-1}_{(r,q+l)}, r \leq q; \\ FFT^{o}_{(q,l,r)2} &= \mu_{(r,q+l)}\mu_{(1)}BM_{(1)}B \ldots BM_{(q+l-r-1)}B\beta_{(1)} \ldots \\ &\ldots B\beta_{(r-q)}B\sigma^{-1}_{(r-q)}\mu^{-1}_{(q+l-r)}\mu^{-1}_{(r,q+l)}, r > q, \end{aligned} \quad (72)$$

где комбинации операторов $\delta_{(r)}\mu_{(q+l-r)}$, $\mu_{(r,q+l)}\mu_{(q+l-r)}\sigma_{(r)}$, $\delta_{(r)}\mu_{(1)}\sigma^{-1}_{(r)}$, $\mu_{(r,q+l)}\mu_{(1)}$ описывают входную буферную память в соответствующих структурах, а $\mu^{-1}_{(q-r+1)}\mu^{-1}_{(r,r)}$, $\sigma^{-1}_{(q)}\sigma_{(r)}\mu^{-1}_{(1)}\mu^{-1}_{(r,r)}$, $\mu^{-1}_{(q+1)}\mu^{-1}_{(r,q+l)}$, $\sigma^{-1}_{(r-l+1)}\mu^{-1}_{(q+l-r)}\mu^{-1}_{(r,q+l)}$, $\mu^{-1}_{(l-r)}\mu^{-1}_{(r,r)}$, $\mu^{-1}_{(l)}\mu^{-1}_{(r,q+l)}$, $\sigma^{-1}_{(l-1)}\sigma_{(r)}\mu^{-1}_{(1)}\mu^{-1}_{(r,r)}$, $\sigma^{-1}_{(q)}\sigma_{(r)}\mu^{-1}_{(1)}\mu^{-1}_{(r,r)}$, $\sigma^{-1}_{(r-q)}\mu^{-1}_{(q+l-r)}\mu^{-1}_{(r,q+l)}$ – выходную буферную память.

Пример структуры ППБПФ-процессора для вычисления векторного ДПФ, построенной в соответствии с выражениями (71) для $D = 4$, $N = 8$, $m = 2$, показан на рис. 7.

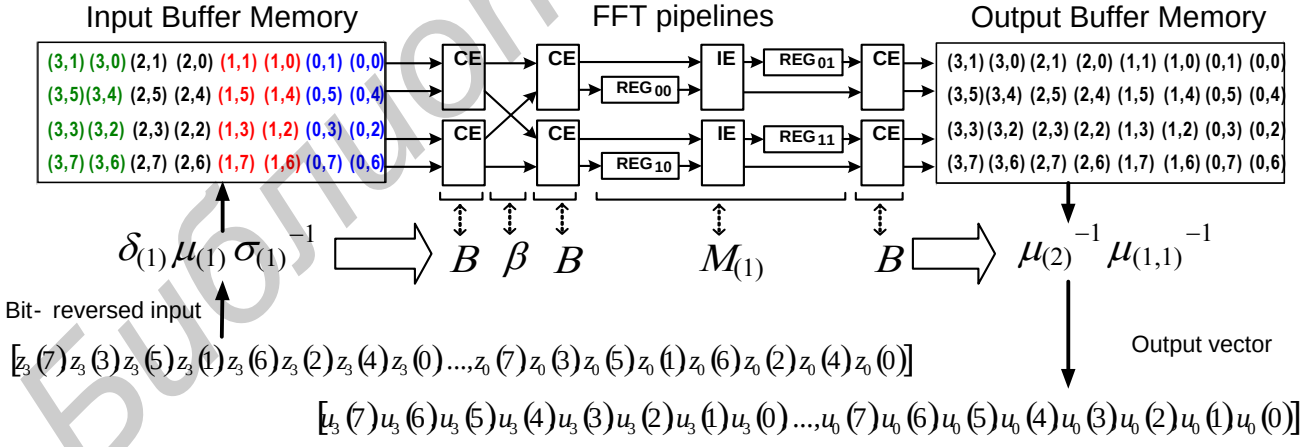

Рис. 7. Структура обратного ППБПФ-процессора для вычисления векторного ДПФ в соответствии с алгоритмом 6 для $D = 4$, $N = 8$, $m = 2$. Одновременно выполняются m последовательных базовых операций

Рассмотренное формализованное описание ППБПФ-структуры является основой для разработки машинно-ориентированного метода синтеза структуры параллельно-поточного процессора БПФ для обработки векторных процессов, а также позволяет проводить исследование процессора на ЭВМ.

6. МЕТОДИКА СИНТЕЗА СТРУКТУР ПАРАЛЛЕЛЬНО-ПОТОЧНЫХ ПРОЦЕССОРОВ БПФ

Анализ основных свойств параллельно-поточной структуры, алгоритмов вычисления векторного ДПФ и организации вычислительного процесса в ППБПФ-процессоре при обработке векторных процессов позволяет сформулировать методику синтеза структуры параллельно-поточного процессора БПФ для обработки векторных процессов.

Основным критерием при синтезе структуры процессора для обработки векторных процессов является обеспечение вычисления векторного ДПФ в реальном масштабе времени с минимальной структурной сложностью. При разработке БПФ-процессора по этому критерию на основе параллельно-поточной структуры, как отмечалось выше, требуется решить следующие две основные задачи структурного синтеза:

1) определить оптимальное количество вычислительных элементов на ступени в ППБПФ-процессоре, обеспечивающее при заданном времени выполнения базовой операции вычисление векторного ДПФ в реальном масштабе времени с минимальной структурной сложностью;

2) для этого значения числа вычислительных элементов на ступени определить топологию структуры ППБПФ-процессора.

Первая задача решается на основе анализа основных свойств параллельно-поточной структуры и организации вычислительного процесса в ППБПФ-процессоре при вычислении векторного ДПФ. Будем рассматривать сложность ППБПФ-структуры как функцию от числа вычислительных элементов на ступени m. Тогда анализ выражений (25), (29), (32), (35), (38) и (41) показывает, что структурная сложность реализации ППБПФ-процессора для вычисления векторного ДПФ является строго возрастающей функцией для всех значений m, для которых определена структура. Из этого однозначно следует, что структура с меньшим числом вычислительных элементов на ступени имеет меньшую сложность. Это обстоятельство позволяет значительно упростить решение первой задачи.

Для заданной скорости поступления входных данных существует множество значений числа вычислительных элементов на ступени в ППБПФ-процессоре A_m, при которых обеспечивается необходимая скорость обработки. Этому множеству A_m соответствует множество структур ППБПФ-процессора A_s, из которых минимальную сложность $C_{\min}$ имеет структура, соответствующая минимальному значению числа вычислительных элементов на ступени $m_{\min}$ из множества A_m. Таким образом, задача определения оптимального, с точки зрения минимума структурной сложности, количества вычислительных

элементов на ступени в ППБПФ-процессоре сводится к нахождению минимального значения числа вычислительных элементов на ступени, которое при заданном времени выполнения базовой операции обеспечивает вычисление векторного ДПФ в реальном масштабе времени. Это значение может быть определено следующим образом.

Условие вычисления векторного ДПФ в реальном масштабе времени для любой структуры имеет вид

$$L_{\text{ВБПФ}} \le NT_{\text{д}}, \tag{73}$$

где $L_{\text{ВБПФ}}$ – латентность вычисления векторного ДПФ; N – размер отдельного компонента векторного процесса; $T_{\text{д}}$ – период дискретизации.

Для всех алгоритмов вычисления векторного ДПФ, рассмотренных в данной главе, значение латентности совпадает и определяется выражением (27). Тогда условие вычисления векторного ДПФ в реальном масштабе времени принимает вид

$$\frac{DN}{2m} T_{\text{БО}} \le NT_{\text{д}}, \tag{74}$$

где $T_{\text{БО}}$ – время выполнения базовой операции арифметическим устройством, откуда

$$m \ge \frac{DT_{\text{БО}}}{2T_{\text{д}}}. \tag{75}$$

Так как возможные значения числа вычислительных элементов на ступени в ППБПФ-процессоре ограничены целой степенью 2, то минимальное число вычислительных элементов на ступени $m_{\min}$ получается из выражения (75) следующим образом:

$$m_{\min} = 2^r, \qquad r = \left]\log_2 \frac{DT_{\text{БО}}}{2T_{\text{д}}}\right[, \tag{76}$$

где $]a[$ означает ближайшее целое число, большее или равное a.

После нахождения минимального числа вычислительных элементов на ступени вторая задача построения структуры ППБПФ-процессора решается просто путем записи для конкретных значений D, N и $m_{\min}$ одного из выражений (65)–(72), описывающих параллельно-поточную структуру для вычисления векторного ДПФ по одному из алгоритмов 1–6.

Следует еще раз отметить, что все структуры, соответствующие алгоритмам 1–6, обеспечивают вычисление векторного ДПФ в реальном масштабе времени при одном и том же минимальном количестве вычислительных элементов на ступени. Однако, как было показано выше, для вычисления векторного ДПФ следует использовать алгоритмы 1, 2 и 6 (последний при параллельном выполнении последовательных базовых операций), т. к. они приводят к минимальной структурной сложности ППБПФ-процессора. При этом предпочтение следует отдавать алгоритму 6, т. к. в этом случае имеем более простую общую организацию вычислительного процесса в ППБПФ-процессоре при вычислении векторного ДПФ.

Таким образом, методику синтеза структуры параллельно-поточного процессора БПФ для обработки векторных процессов можно представить в виде следующего алгоритма.

Алгоритм 7. Синтез структуры ППБПФ-процессора для обработки векторных процессов

Вход. Мерность векторного процесса $D = 2^q$, размер компонента векторного процесса $N = 2^l$, период дискретизации $T_д$, время выполнения базовой операции арифметическим устройством $T_{БО}$.

Выход. Структура $FFT_{(q,l,r)}$ ППБПФ-процессора с $m = 2^r$ вычислительными элементами на ступени.

1. Определить

$$q = \log_2 D. \tag{77}$$

2. Определить число ступеней в ППБПФ-процессоре

$$l = \log_2 N. \tag{78}$$

3. По формулам (76) определить параметр r и число процессорных модулей на ступени m, обеспечивающее вычисление векторного ДПФ в реальном масштабе времени при минимальной структурной сложности процессора.

4. Если $m \le DN/2$ или, что то же самое, $r \le q + l - 1$, то перейти к шагу 5, в противном случае при заданном времени выполнения базовой операции $T_{БО}$ ППБПФ-процессор с заданными параметрами и не может быть построен, и перейти к шагу 14.

5. Определить общее число процессорных модулей в ППБПФ-процессоре

$$m_{общ} = ml. \tag{79}$$

6. Определить число ступеней в ППБПФ-процессоре, имеющих промежуточную память,

$$m_{ст\ ПП} = \begin{cases} l - r - 1, & r < l - 1, \\ 0, & r \ge l - 1. \end{cases} \tag{80}$$

7. Определить общее число модулей промежуточной памяти в ППБПФ-процессоре

$$m_{ПП\ общ} = \begin{cases} (l - r - 1)m, & r < l - 1, \\ 0, & r \ge l - 1. \end{cases} \tag{81}$$

8. Определить общий объем промежуточной памяти в словах в ППБПФ-процессоре

$$V_{ПП\ общ} = \begin{cases} N - 2m, & r < l - 1, \\ 0, & r \ge l - 1. \end{cases} \tag{82}$$

9. Если $m < N/2$ или, что то же самое, $r < l - 1$, то перейти к шагу 10, если $m = N/2$ или $r = l - 1$, перейти к шагу 11, в противном случае – к шагу 12.

10. Записать описание структуры $FFT_{(q,l,r)}$ ППБПФ-процессора путем подстановки значений q, l и r в выражение (65), (66), (67), (68) или (71). Перейти к шагу 13.

11. Записать описание структуры $FFT_{(q,l,r)}$ ППБПФ-процессора путем подстановки значений q, l и r в выражение (65), (66), (67), (68), (70) или (71). Перейти к шагу 13.

12. Записать описание структуры $FFT_{(q,l,r)}$ ППБПФ-процессора путем подстановки значений q, l и r в выражение (70) или (71).

13. Построить структуру ППБПФ-процессора по полученному выражению. При этом операторы в описании структуры соответствуют следующим элементам структуры:

B – вычислительному элементу (арифметическому устройству базовой операции);

$M_{(i)}$ – промежуточной памяти объемом $2 \times 2^{i-1}$ слов;

$\beta_{(i)}$ – непосредственной, без промежуточной памяти, связи между двумя ступенями ППБПФ-процессора.

14. Конец.

Рассмотренный алгоритм позволяет построить структуру ППБПФ-процессора, вычисляющего векторное ДПФ в реальном масштабе времени с минимальной структурной сложностью при заданных скорости поступления входных данных, структуре вычислительного элемента (арифметического устройства) и времени выполнения базовой операции. Дальнейшее упрощение ППБПФ-процессора может быть получено за счет совершенствования структуры вычислительного элемента с целью уменьшения времени выполнения базовой операции.

7. ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНО-ПОТОЧНОГО ПДВП-ПРОЦЕССОРА

В последние десятилетия важное место в цифровой обработке аудиосигналов занимают алгоритмы кодирования (компрессии), позволяющие получать компактное представление аудиосигнала в цифровой форме для эффективной передачи по каналам связи или хранения на некотором носителе [9]. Именно высококачественное кодирование аудиосигналов становится ключевой технологией в развитии цифровых аудиосистем, таких, как системы запоминания и редактирования цифровых аудиосигналов на персональных компьютерах (домашние студии), компьютерные мультимедийные системы, цифровое радиовещание, передача аудиосигналов через узкополосные ISDN каналы и сеть Интернет при проведении теле- или видеоконференций.

Большинство систем компрессии и обработки аудиоданных базируется на анализе информации в частотной области, что приводит к необходимости поиска преобразования, осуществляющего заданную частотно-временную декомпозицию сигнала. В этом смысле большое распространение получило пакетное дискретное вейвлет-преобразование (ПДВП), обладающее целым рядом преимуществ по сравнению с более традиционными методами декомпозиции [9].

Структурная схема процессора показана на рис. 8, которая состоит из вычислительных блоков (ВБ), блоков памяти (БП) и блоков коммутации (БК). Блоки памяти и коммутации представлены на рис. 9.

Вычислительный блок выполняет функцию двухканального банка фильтров, раскладывающего входной сигнал на низкочастотную и высокочастотную составляющие с последующей децимацией по основанию 2 в каждом выходном канале.

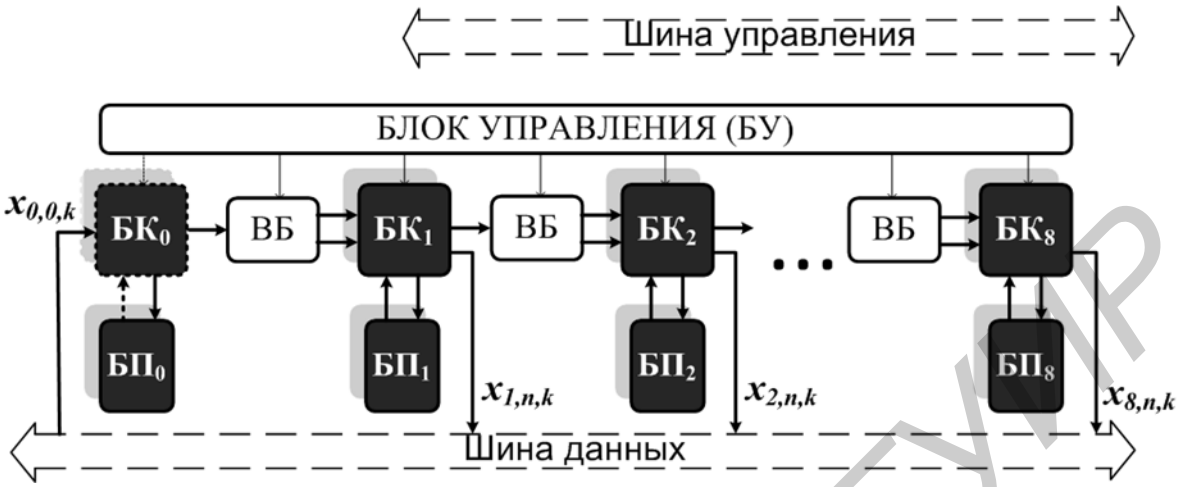

Рис. 8. Структурная схема процессора

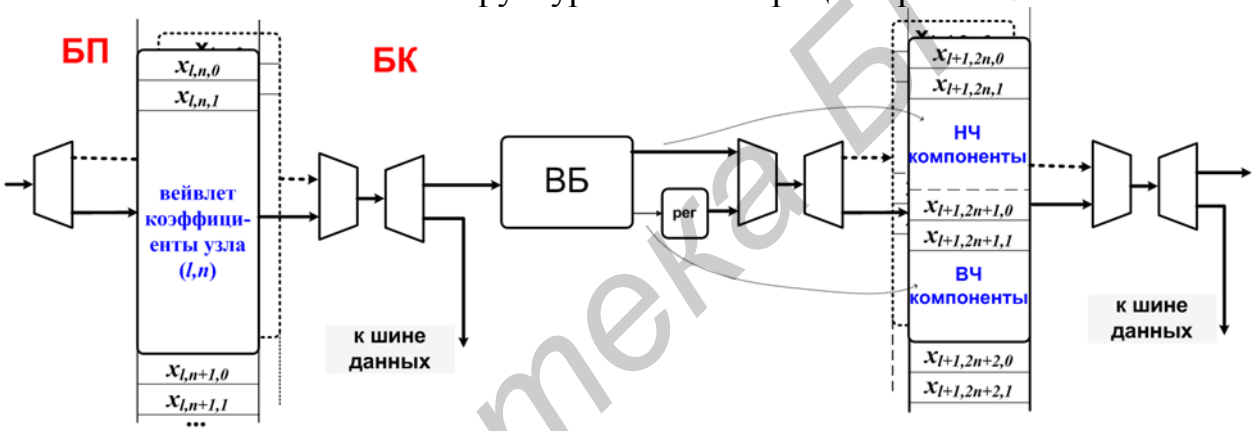

Рис. 9. Блоки памяти и коммутации

Для эффективной реализации алгоритма ПДВП большое распространение получили поточные архитектуры. Рассмотрим в качестве аппаратной реализации для алгоритма ПДВП динамически реконфигурируемый поточный процессор на базе ПЛИС.

Представим алгоритм ПДВП в виде сигнального ориентированного графа. На данном графе входной сигнал x представляется как набор значений $x[n]$, где $n = 0,1,2\ldots$ – это номер входного отсчета. Расчет очередных двух значений с выхода двухканального банка фильтров производится с помощью вычислительного элемента ВЭ, на вход которого поступают два отсчета входного сигнала (либо два промежуточных отсчета, лежащих в одной частотной субполосе). Также для данного графа введены следующие обозначения:

$J = 0\ldots7$ – номер уровня декомпозиции;

$w_{J,k}[n]$, $k = 0\ldots(2^{J+1} - 1)$ – n-й вейвлет-коэффициент k-й субполосы, полученный на J-м уровне декомпозиции;

$G_{k,i}$ – идентификатор i-го ВЭ, обрабатывающего последовательность из k-й субполосы, полученной на предыдущем уровне декомпозиции, т. е. для ВЭ на J-м уровне индекс k может принимать значения от 0 до 2^J.

На каждом уровне декомпозиции J все ВЭ разбиваются на 2^J группы так, чтобы вычислительные элементы в одной группе обрабатывали последовательность только из одной частотной субполосы. В идентификаторе $G_{k,i}$ индекс k обозначает номер группы, а индекс i – номер ВЭ в k-й группе. Таким образом, идентификатор $G_{k,i}$ уникален для каждого ВЭ в рамках одного уровня декомпозиции. В качестве примера на рис. 10 приведен сигнальный ориентированный граф для алгоритма полного трехуровневого ПДВП (т. е. для алгоритма, описывающего структуру полного дерева, в котором присутствуют все ветви).

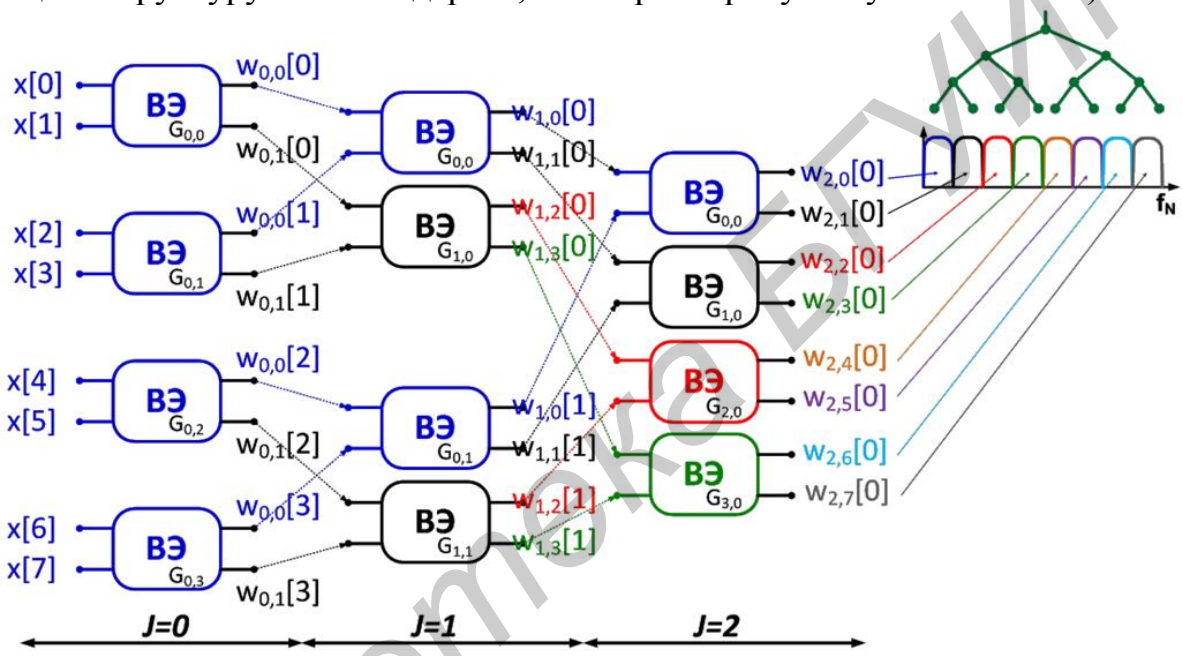

Рис. 10. Сигнальный ориентированный граф для алгоритма полного трехуровневого ПДВП

С помощью представленного графа можно описывать не только алгоритм полного ПДВП, но и алгоритм с произвольной структурой дерева. Для этого необходимо в тех местах, где отсутствуют ветви в структуре дерева, просто убрать ВЭ, а данные, которые должны были бы обрабатываться этими ВЭ, пропустить без изменения на следующий уровень графа.

На рис. 11 приведен пример для трехуровневого ПДВП с неполной структурой дерева.

С учетом анализа алгоритма ПДВП на основе представленного графа предлагается поточная архитектура процессора ПДВП, показанная на рис. 12.

Данная архитектура позволяет вычислять полное восьмиуровневое ПДВП с использованием одного вычислительного элемента на каждом уровне. Для правильного обмена данными между ВЭ на смежных уровнях вводятся регистровые файлы, работающие по принципу FIFO, а также коммутаторы SW_J.

Работа данных блоков сводится к формированию корректной последовательности данных, поступающих на вход ВЭ на каждом уровне в соответствии с сигнальным ориентированным графом.

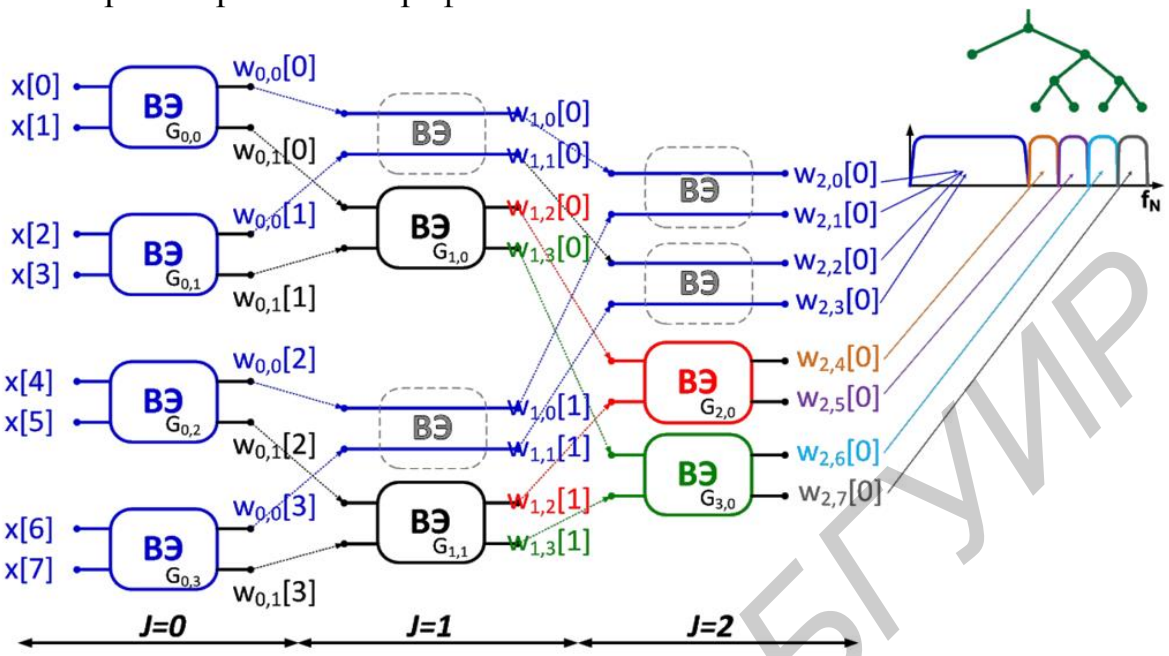

Рис. 11. Сигнальный ориентированный граф алгоритма трехуровневого ПДВП для неполной структуры дерева

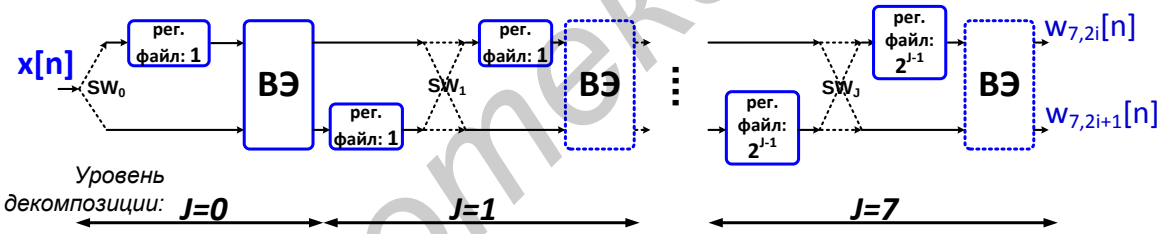

Рис. 12. Поточная архитектура процессора ПДВП с числом уровней декомпозиции, равным 8

На рис. 13 приведен пример графа потока данных на входе и выходе ВЭ первого уровня и далее сформированный поток данных, поступающих на вход ВЭ второго уровня декомпозиции. Коммутатор SW_2 в четные отсчеты времени должен замыкать линии связи в прямом направлении, а в нечетные отсчеты времени – в перекрестном.

На каждом последующем уровне декомпозиции коммутатор SW_J (J = = 2 … 7) переключается с периодичностью в два раза большей, чем коммутатор SW_{J-1} на предыдущем уровне, что в итоге позволяет с учетом введения регистровых файлов обеспечить правильное формирование потока данных.

Таким образом, рассмотренная поточная структура может быть легко масштабируема для любого числа уровней декомпозиции дерева ПДВП, при этом она не требует ввода сложной схемы управления.

Операционный блок процессора должен удовлетворять следующим требования:

– полное функциональное соответствие математической модели на арифметике с фиксированной запятой;

– минимизация аппаратных затрат;

– обеспечение высокой пропускной способности;

– параметризация составных элементов блока с целью ускорения процесса проектирования при изменении требований разработчика.

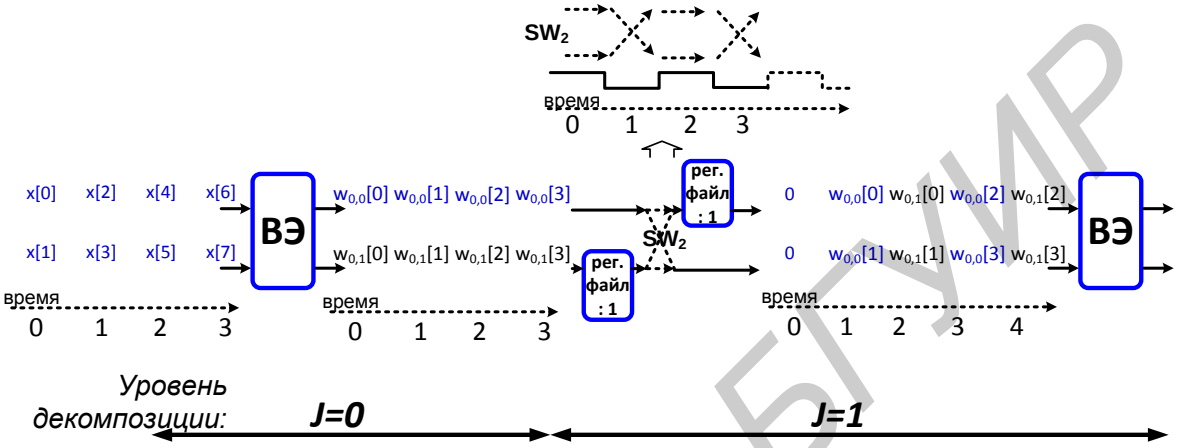

Рис. 13. Граф потока данных между первым и вторым уровнем процессора ПДВП

Последний критерий подразумевает, что результатом разработки должна стать параметризированная модель процессора ПДВП, в которой разработчик может изменять следующие характеристики:

– конкретный тип лестничной структуры (число шагов лестничной структуры, значения коэффициентов b_0, b_1 и др.);

– возможность изменения разрядности основных операционных узлов и регистров для хранения входных, выходных и промежуточных данных, участвующих в вычислениях.

В результате анализа алгоритма предлагается структура операционного блока, изображенная на рис. 14.

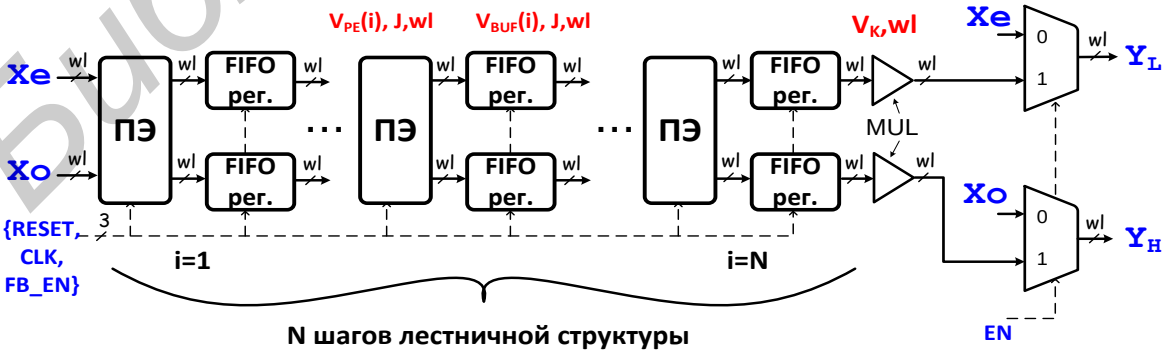

Рис. 14. Структурная схема операционного блока

Вся структура данного блока задается следующими параметрами:

wl – разрядность данных, поступающих на вход, и соответственно разрядность всех арифметических операций и буферов для хранения промежуточных результатов;

J – индекс, задающий уровень декомпозиции, на котором используется данный операционный блок (банк фильтров);

N – число элементарных шагов лестничной структуры;

$V_{PE}(i) \to \{id, type, b_0, ind_x, sh_{x1}, sh_{x2}, [sh_{s2}, b_1, delayed]\}, i = 1 \dots N$ – массив из N векторов, каждый из которых задает параметры i-го элементарного шага лестничной структуры (сами параметры будут рассмотрены ниже);

$V_{BUF}(i) \to \{d_{xe}, d_{xo}\}, i = 1 \dots N$ – массив из N векторов, каждый из которых задает число элементов задержки d_{xe}, d_{xo} соответственно в верхнем и нижнем каналах после i-го элементарного шага;

$V_K \to \{K_1, K_2, n_{ye}, n_{yo}\}$ – вектор, задающий значения квантованных коэффициентов K_1 и K_2 для операции умножения, а также номера старших бит n_{ye}, n_{yo}, начиная с которых будут выданы результаты перемножения сигналов на коэффициенты K_1 и K_2 соответственно.

Принцип работы операционного блока заключается в следующем. На входные шины X_e и X_o поступают четный и нечетный отсчеты входной последовательности, после чего начинается процесс вычисления вейвлет коэффициентов. Результат высчитывается в процессорных элементах ПЭ на основе текущих входных значений и предыдущих значений, хранящихся в буферных блоках. Данные блоки представляют собой регистровые файлы, работающие по принципу FIFO. Число регистров в каждом таком блоке определяется как $n_{xe} \cdot 2^J$ (либо $n_{xo} \cdot 2^J$), где значения n_{xe}, n_{xo} определяют задержку сигнала, соответствующую задержке $z^{-n_{xe}}, z^{-n_{xo}}$ в исходном алгоритме, а значение 2^J определяет число тактов между приходом входных отсчетов из одной субполосы. Данный прием необходим для того, чтобы значения предыстории на выходах блоков FIFO относились к той же субполосе, что и текущие входные сигналы.

Пара результатов, вычисленная в последнем элементе ПЭ, умножается на константы K_1 и K_2 и далее подается на выход операционного блока.

Как уже говорилось ранее, для параметризации i-го процессорного элемента ПЭ формируется специальный вектор $V_{PE}(i)$. Ниже даны разъяснения для всех параметров, включенных в данный вектор.

Параметр id(identifier). Идентификатор, задающий один из двух типов процессорного элемента для реализации элементарного шага лестничной структуры: $id =' PE0'$ при значении коэффициента $b_1^i = 0$ (рис. 15), в противном случае $id =' PE1'$ (рис. 16).

Параметр $type$ задает тип элементарного шага, реализуемого PE: $type = = 's'$, если это прямой шаг, и $type = 't'$, если шаг дуальный.

Параметры b_0, b_1 могут принимать значения от -2^{wl-1} до $+(2^{wl-1}-1)$ и соответствуют значениям mb_0, mb_1 для квантованных коэффициентов b_0^i, b_1^i. В случае $id =' PE0'$ параметр b_1 игнорируется.

Параметр ind_x представляет собой массив из трех чисел, принимающих значения 1, 2 или 3. Данный параметр определяет очередность суммирования сигналов A, s_1 s_2 для ПЭ с $id =' PE1'$. Например, если параметр $ind_x = [2\ 3\ 1]$, то для получения выходного результата сначала выполняется операция $A + s_2$ и далее к получившемуся результату прибавляется значение s_1.

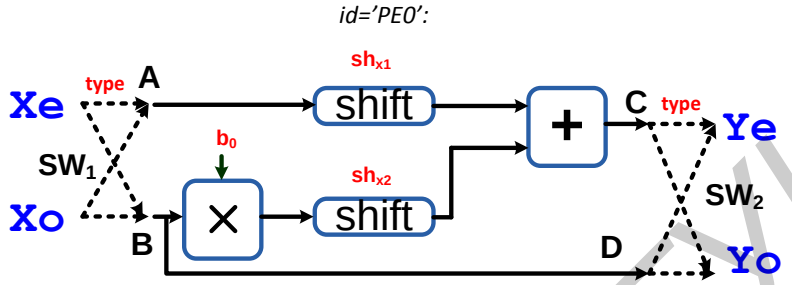

Рис. 15. Структурная схема процессорного элемента при $id =' PE0'$

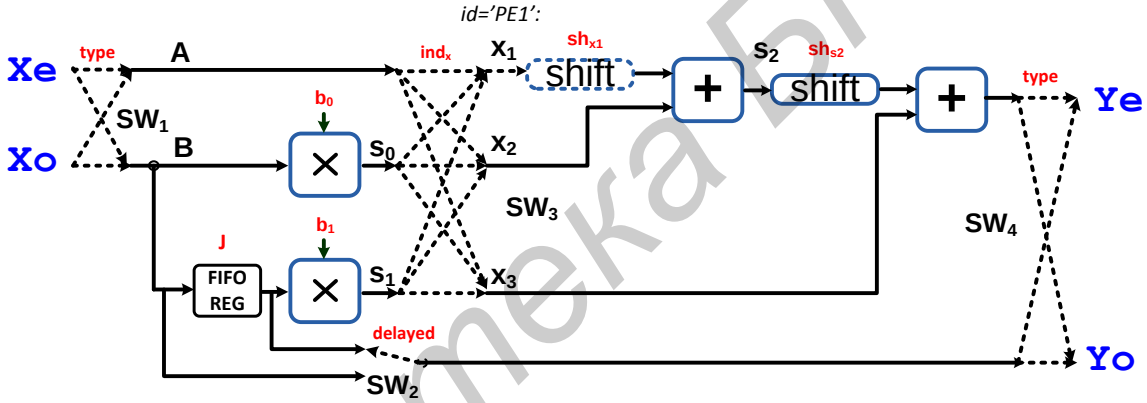

Рис. 16. Структурная схема процессорного элемента при $id =' PE1'$

Параметры $sh_{x1}, sh_{x2}, sh_{s2}$ определяют число арифметических сдвигов для внутренних сигналов процессорного элемента x_1, x_2 и s_2 соответственно.

Параметр $delayed$. Для того чтобы разъяснить назначение данного параметра, рассмотрим участок исходного алгоритма (рис. 17), на котором сначала выполняется прямой шаг $s(z) = b_0 + b_1 z^{-1}$, и далее следует задержка сигнала z^{-1}.

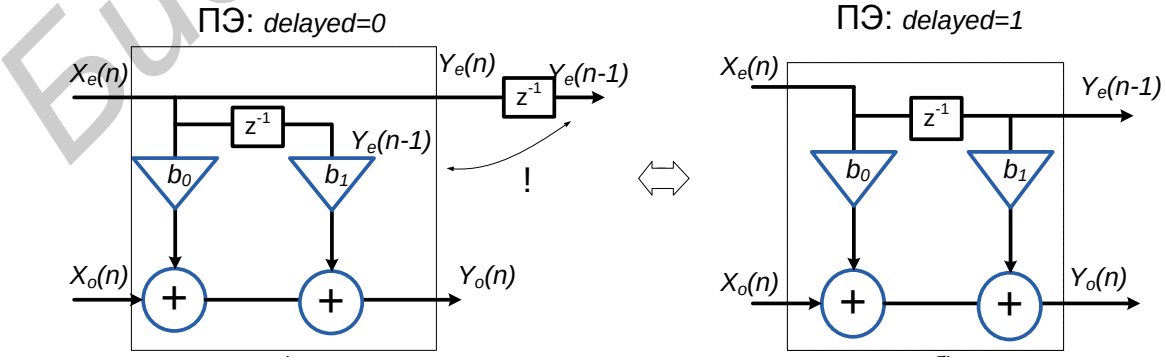

Рис. 17. Структурная схема процессорного элемента PE

Как видно из рисунка, сигнал $Y_e(n-1)$, который должен быть на выходе всей схемы, уже сформирован внутри процессорного элемента. Поэтому при значении $delayed = 1$ на выход ПЭ выдается не сигнал $Y_e(n)$, а сигнал $Y_e(n-1)$, что позволяет сэкономить для данной схемы на *J*-м уровне декомпозиции 2^J регистров.

Отметим, что блоки арифметического сдвига *shift*, а также коммутаторы SW_1, SW_2, SW_3, SW_4 не используют аппаратного ресурса системы. Операция арифметического сдвига вправо сводится к смещению сигнальных линий перед подачей их на вход сумматора. А указанные выше коммутаторы в соответствии с параметрами $type, ind_x, delayed$ реализуются в виде замыкания соответствующих линий.

Таким образом, процессорный элемент для случая $id =' PE0'$ в конечном итоге будет реализован на одном сумматоре и одном умножителе, а процессорный элемент для $id =' PE1'$ – на двух сумматорах, двух умножителях, и регистровом файле, состоящем из 2^J регистров. Все используемые элементы имеют разрядность, равную wl.

ЛИТЕРАТУРА

1. Korn, D. G. Computing the fast Fourier transform on a vector computer / D. G. Korn, J. J. Lambiotte // Math. of Comp. – 1979, July. – V. 33, №147. – P. 977–992.

2. Петровский, А. А. Параллельно-поточные структуры вычисления векторного дискретного преобразования Фурье / А. А. Петровский, М. В. Качинский // Автоматика и вычисл. техника. – 1988. – Вып. 17. – С. 85–91.

3. Качинский, М. В. Параллельно-поточный процессор БПФ и машинно-ориентированный метод построения его структуры / М. В. Качинский, А. А. Петровский // Методы и микроэлектронные средства цифрового преобразования и обработки сигналов. SIAP-89 : тез. докл. конф., Рига, 1989. – Рига : ИЭВТ АН ЛатвССР, 1989. – Т. 1. – С. 181–183.

4. Петровский, А. А. Формализованный метод построения структуры параллельно-поточный процессора БПФ / А. А. Петровский, М. В. Качинский // Распараллеливание обработки информации : тез. докл. 7-й Всесоюзн. шк.-семинара, Львов, 9–14 окт. 1989. – Львов : ФМИ АН УССР, 1989. – Ч. 2. – С. 184–186.

5. Wold, E. H. Pipeline and parallel-pepeline FFT processors for VLSI implementations / E. H. Wold, A. M. Despain // IEEE Trans. on Comput. – 1984, May. – V. C-33, №5. – P. 414–425.

6. Parker, D. S. Notes on shuffle/exchange-type switching networks / D. S. Parker // IEEE Trans. On Comput. – 1980, March. – V. C-29, №3. – P. 213–222.

7. Petrovsky, A. Design method of real-time parallel-pipeline processors computing the vector DFT / A. Petrovsky, M. Kachinsky // Proc. of the Intern. conf. On parallel computing in electrical engineering, PARELEC'98, Bialystok. – Poland, 1998. – P. 242–247.

8. Petrovsky, A. Automated parallel-pipeline structure of FFT hardware design for real-time multidimensional signal processing / A. Petrovsky, M. Kachinsky // Proc. of the 9th European Signal processing conference, EUSIPCO'98, Rhodes. – Greece, 1998. – Vol. 2. – P. 491–494.

9. Петровский, Ал. А. Быстрое проектирование систем мультимедиа от прототипа / Ал. А. Петровский, А. В. Станкевич, Ал. А. Петровский. – Минск : Бестпринт, 2011. – 412 с.

Св. план 2013, поз. 11

Учебное издание

Качинский Михаил Вячеславович
Петровский Александр Александрович

СТРУКТУРНЫЙ СИНТЕЗ ПАРАЛЛЕЛЬНО-ПОТОЧНЫХ БПФ-ПРОЦЕССОРОВ РЕАЛЬНОГО ВРЕМЕНИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. И. Герман*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *М. В. Гуртатовская*

Подписано в печать 17.10.2014. Формат 60х84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,3. Уч.-изд. л. 4,4. Тираж 100 экз. Заказ 239.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6