Цифровая Обработка Сигналов: Методы и Средства (Digital Signal Processing: Methods and Instruments)

Book · January 2001

CITATIONS
0

READS
2,033

3 authors, including:

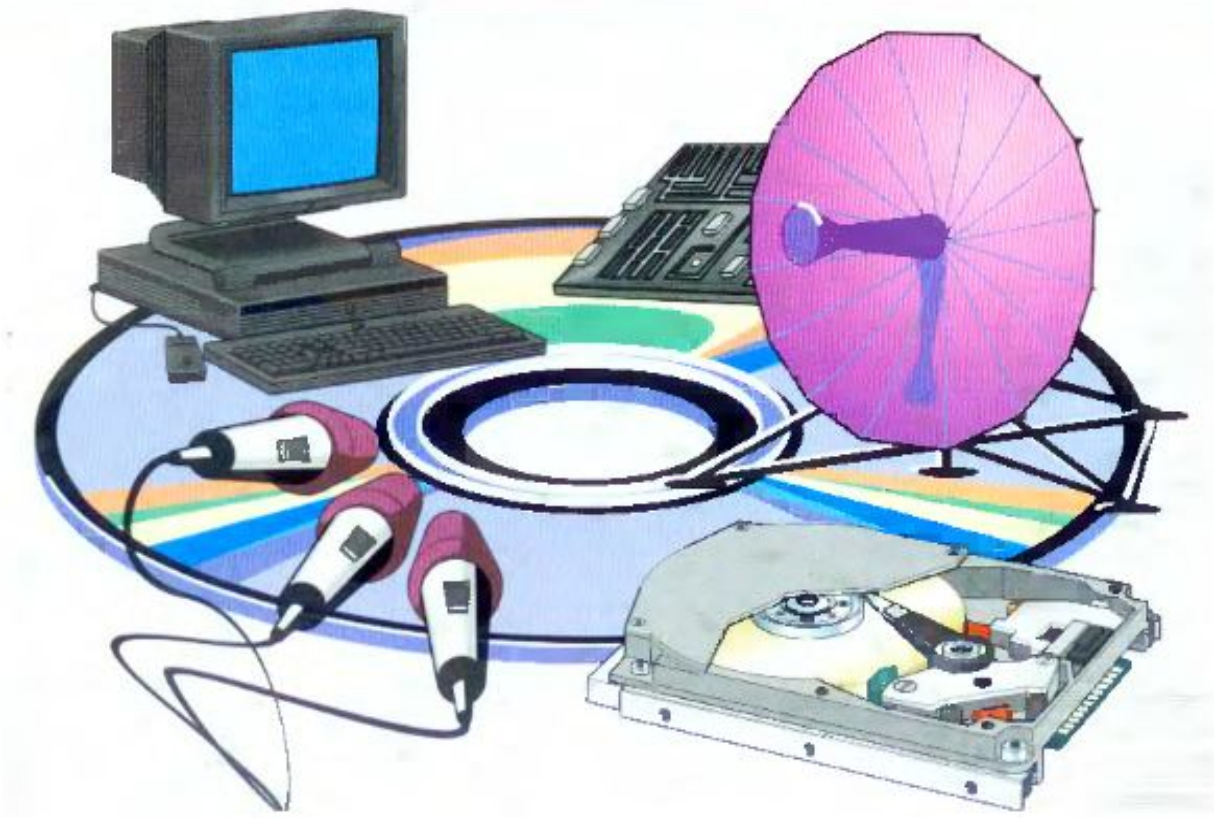
Vladimir Bondarev
Sevastopol State University
25 PUBLICATIONS **22** CITATIONS

SEE PROFILE

В. Бондарев, Г. Трёстер, В. Чернега

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ:

методы и средства



В. БОНДАРЕВ, Г. ТРЁСТЕР, В. ЧЕРНЕГА

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ:

МЕТОДЫ И СРЕДСТВА

Рекомендовано Министерством образования Украины
в качестве учебного пособия для студентов вузов,
обучающихся по направлению 0804 - «Компьютерные науки»

1999, 2001

ББК 32.811.3+32.973.2
Б811
УДК 621.391.26.083.92+681.325.5-181.4

Рецензенты:
канд. техн. наук доцент Новацкий А.А., д-р физ.-мат. наук Суворов А. М.

Бондарев В.Н., Трёстер Г., Чернега В.С.
Б811 **Цифровая обработка сигналов: методы и средства**: Учеб. пособие для вузов.- Севастополь: Изд-во СевГТУ,1999.-398с.: ил.

ISBN 966-7473-15-5

Бондарев В.Н., Трёстер Г., Чернега В.С.
Б811 **Цифровая обработка сигналов: методы и средства**: Учеб. пособие для вузов. 2-е изд. – Х.: Конус, 2001.-398с.: ил.
ISBN 966-7636-14-3

Книга содержит обширную информацию по методам и программно-аппаратным средствам цифровой обработки сигналов. В первой части книги изложены элементы теории сигналов и систем, основы цифрового спектрального анализа и синтеза цифровых фильтров, методы цифровой обработки речевых и аудио сигналов, методы обработки изображений. Во второй части книги рассмотрены архитектурные особенности цифровых процессоров обработки сигналов (ЦПОС), подробно освещаются вопросы программирования ЦПОС с фиксированной запятой TMS320C2x фирмы Texas Instruments. На примере ЦПОС фирмы Analog Devices описаны возможности 32-разрядных ЦПОС с плавающей запятой.

Книга предназначена в качестве учебного пособия для студентов вузов, обучающихся по направлениям: «Компьютерные науки», «Компьютерная инженерия», «Компьютеризированные системы автоматики и управления» и др. , а также для широкого круга лиц, интересующихся вопросами построения подсистем цифровой обработки сигналов в распределенных информационных мультимедиа системах.

ISBN 966-7473-15-5

ISBN 966-7636-14-35

ПРЕДИСЛОВИЕ

Современный этап развития информационных технологий характеризуется бурным развитием распределенных информационных мультимедиа систем (РИМС). РИМС революционизируют образ жизни человека. Их развитие направлено на создание электронного мира, в котором люди смогут эффективно работать, учиться, отдыхать, решать ежедневные задачи своего жизнеобеспечения. Сегодня РИМС предоставляют доступ к электронным библиотекам, позволяют смотреть и слушать видео и радио программы, обеспечивают проведение видеоконференций и многое другое. На очереди создание интерактивных РИМС, в которых предусматриваются голосовые интерфейсы, интерактивное видео и игры и пр.

РИМС интегрируют телекоммуникационные, вычислительные и информационные ресурсы и предназначены для сбора, хранения, обработки и передачи информации в самых разных формах ее представления - текст, числовые данные, изображения, аудио, видео и пр. Примером РИМС являются сети ПЭВМ и рабочих станций, соединенные с серверами мультимедиа информации посредством Internet.

Использование в РИМС, наряду с традиционными информационными архивами, мультимедийных баз данных требует введения в учебные планы подготовки специалистов в области информационных технологий дисциплин, изучающих как методы цифрового представления и обработки различных сигналов, так и вычислительные средства, с помощью которых данная обработка выполняется. Совместное рассмотрение методов и средств цифровой обработки сигналов (ЦОС) должно способствовать более глубокому пониманию сути процессов, протекающих в РИМС, и формированию необходимых практических навыков по разработке конкретных технологических цепочек преобразования информации в РИМС на базе ЦОС.

Учебные пособия на эту тему на территории СНГ практически не издавались. Имеющаяся учебная литература ориентирована преимущественно на специалистов радиотехнического направления и была издана до начала 90-х годов, т.е. почти 10 лет назад. Она не в полной мере учитывает специфику подготовки специалистов в области информационных

технологий, а также современный уровень развития средств ЦОС. В ней не описываются современные методы обработки звуковых сигналов и изображений, отсутствует последовательное изложение вопросов программирования цифровых процессоров обработки сигналов (ЦПОС) - важных элементов РИМС реального времени.

Цель издания учебного пособия - дать систематический обзор основных методов ЦОС, включая современные методы обработки звука и изображений, и изложить вопросы их практической реализации на основе СБИС ЦПОС, выпускаемых ведущими фирмами мира.

Книга базируется на соответствующих курсах лекций, читаемых авторами в Федеральной высшей технической школе (ETH) г.Цюриха и в Севастопольском государственном техническом университете. Книга будет полезной студентам, обучающимся по направлениям “Компьютерные науки”, “Компьютерная инженерия”, “Компьтеризированные системы автоматики и управления”, а также специалистам, занимающимся разработкой разнообразных систем с ЦОС.

Введение, главы 2,3,4,7 написаны В.Н. Бондаревым, главы 1 и 6 написаны В.Н. Бондаревым и Г. Трёстером,. главы 5 и 8 - В.С. Чернегой.

Авторы выражают признательность администрации Института электроники ETH за предоставленную возможность работы над книгой, а также рецензентам – доценту кафедры автоматики и управления в технических системах Национального технического университета Украины (Киевский политехнический институт) канд. техн. наук А.А. Новацкому и заведующему отделом морских информационных систем и технологий Морского гидрофизического института НАН Украины д-ру физ.-мат. наук А.М. Суворову за доброжелательную критику и полезные замечания, которые способствовали улучшению книги. Особую благодарность авторы выражают д-ру техн. наук, проф. В.К. Маригодову за его большой труд по редактированию рукописи.

Отзывы и предложения просьба направлять по адресу: Украина, 335053, г. Севастополь, Студгородок, Севастопольский государственный технический университет, НМЦ.

Цюрих-Севастополь
Июнь 1999г.

ВВЕДЕНИЕ

Под сигналом в общем случае понимают физический процесс, который осуществляет перенос информации во времени и пространстве. Сигналы описываются математическими моделями, отражающими общие свойства различных по физической природе процессов. Обработкой сигнала называется преобразование сигнала с целью представления информации, содержащейся в сигнале, в наиболее удобной форме. Обработка сигналов может осуществляться с помощью различных технических средств: аналоговых (непрерывных), цифровых, гибридных (представляющих комбинацию аналоговых и цифровых средств). Первоначально обработка сигналов осуществлялась с помощью аналоговых средств. Примерно с конца 80-х годов в лидеры выходит цифровая обработка.

Цифровая обработка сигналов (ЦОС) основана на представлении сигналов в виде последовательностей чисел и может осуществляться либо с помощью универсальных цифровых ЭВМ, либо с помощью специальных цифровых процессоров обработки сигналов (ЦПОС). Реализация ЦОС в реальном масштабе времени требует, как правило, применения ЦПОС. Обработка сигналов в ЭВМ и ЦПОС выполняется на основе алгоритмов или процедур.

Причины использования цифровых методов обработки сигналов вряд ли нуждаются в перечислении. Цифровые системы мало чувствительны к параметрам окружающей среды. Они могут быть адаптивными и их легко перепрограммировать. Цифровые сигналы можно хранить в неизменном виде неограниченное время. Цифровые алгоритмы легко переносятся с оборудования одного изготовителя на оборудование другого изготовителя и пр.

ЦОС развивается уже свыше 50 лет. За это время созданы эффективные алгоритмы обработки сигналов, прогрессивные технологии производства БИС ЦПОС, расширено применение ЦОС. Если на первых этапах ЦОС применялась в основном в военных системах, то сегодня диапазон приложений ЦОС весьма широк. Он охватывает следующие области: распознавание и синтез речи, сжатие речи, засекречивание речи, идентификация дикторов по голосу, сжатие и редактирование аудио

сигналов, синтез аудио сигналов, передача данных, подавление шумов, сжатие изображений, синтез и распознавание изображений, сжатие и обработка видеосигналов, спектральный анализ, подавление эхо-сигналов в системах телекоммуникаций, управление приводами магнитооптических дисков, акустика, сейсмика, робототехника и др.

В течение последних 10 лет наблюдается интенсивное внедрение технологий ЦОС в компьютеризированные системы обработки информации. Благодаря этому последние становятся мультимедийными, т.е. наряду с классическими типами данных, хранящимися и обрабатывающимися в таких системах, появляется возможность хранения и обработки аудио и видео данных. Это способствует кардинальному изменению взаимодействия человека с информационными системами. Они становятся по настоящему “дружественными”, значительно расширяется перечень и повышается качество предоставляемых ими информационных услуг.

Выход ЦОС на рынок мультимедийных информационных систем стимулирует дальнейшее развитие производства ЦПОС. Начиная с 1988 года объем производства ЦПОС увеличивался каждый год на 40%. По прогнозам к 2007 году объем продаж БИС ЦПОС составит 20 млрд. долл.

Вышесказанное предполагает детальное знакомство специалистов в области компьютерных информационных технологий с методами и средствами ЦОС. По этой причине в последние время дисциплина “Цифровая обработка сигналов” вводится в учебные планы подготовки специалистов по информационным технологиям во многих вузах. Однако до настоящего времени не издавалось учебного пособия, ориентированного на эту категорию читателей.

Цель данной книги - дать систематический обзор основных методов ЦОС и их практической реализации на основе БИС ЦПОС. Книга ориентирована на студентов старших курсов, обучающихся по направлениям: «Компьютерные науки”, “Компьютерная инженерия”. В ней широко используются математические модели сигналов. Однако описание методов ЦОС ведётся скорее на качественном и содержательном уровне, чем на уровне строгих математических доказательств. Содержащийся в книге учебный курс следует рассматривать как начальный этап на пути к специализации в области технологий ЦОС.

Книга состоит из введения и восьми глав. В первой главе вводятся основные понятия и определения ЦОС. В данную главу включены некоторые вопросы теории непрерывных сигналов и систем, важные с точки зрения последующего изучения методов ЦОС. В главе рассматриваются основополагающие вопросы теории ЦОС, такие как: дискретизация, спектральные преобразования, цифровая фильтрация,

квантование и масштабирование , многомерные цифровые сигналы и системы.

Во второй главе описываются наиболее важные методы цифрового спектрального анализа, которые позволяют выявить частотный состав исследуемых сигналов.

Третья глава посвящена рассмотрению методов синтеза одномерных и двумерных цифровых фильтров. Приводятся примеры синтеза конкретных фильтров и фрагменты соответствующих программ.

В четвертой главе подробно рассматриваются основные методы обработки речевых и аудио сигналов. Описываются особенности слуховой системы человека и его голосового аппарата, используемые при кодировании, сжатии и синтезе речи и аудио сигналов.

Пятая глава посвящена рассмотрению вопросов обработки изображений. Здесь рассматриваются вопросы представления изображений и операции над ними, сегментация и сжатие изображений.

В шестой главе рассматриваются основные характеристики ЦПОС и их архитектурные особенности. Приводятся сравнительные данные наиболее популярных семейств ЦПОС.

Седьмая глава посвящена изучению ЦПОС с фиксированной запятой на примере СБИС TMS320C25 фирмы Texas Instruments. Подробно рассматривается архитектура и система команд, технология написания программ с использованием типовых алгоритмических конструкций и структур данных - массивов, стеков, очередей. Приводятся многочисленные примеры и программы цифровой фильтрации и быстрого преобразования Фурье.

В восьмой главе рассматриваются архитектурные особенности и система команд ЦПОС с плавающей запятой на примере СБИС ADSP21xxx фирмы Analog Devices. Приводится система команд, примеры программ основных алгоритмов ЦОС и примеры построения многопроцессорных систем.

Глава 1
ЭЛЕМЕНТЫ ТЕОРИИ СИГНАЛОВ И СИСТЕМ

1.1 Классификация сигналов

Слово “сигнал” ведет свое происхождение от латинского слова “signum” (знак). Знаки используются для передачи и хранения информации. Совокупность знаков, представляющих ту или иную информацию, называют *сообщением. Сигнал* - это некоторый физический процесс, являющийся носителем сообщения. Примерами сигналов могут служить ток в цепи микрофона, воспринимающего речь, яркость луча электронно-лучевой трубки телевизора при приеме изображения и др.

В теории сигналов обычно рассматривают не сам физический процесс, переносящий сообщение, а его математическую модель, являющуюся формой аналитического описания сигнала. Чаще всего сигналы представляют функциональными зависимостями, в которых аргументом является время t или некоторая пространственная переменная s. Функции, описывающие сигналы, могут принимать как вещественные, так и комплексные значения. В дальнейшем для обозначения сигналов будем использовать функциональные зависимости вида $u(t), f(s)$ и т.п.

Сигнал, описываемый функцией одной переменной, называется *одномерным*, а сигнал, описываемый функцией M независимых переменных $(M \geq 2)$, - *многомерным*. Например, яркость изображения $I(x, y)$ - двумерный сигнал.

Иной признак классификации сигналов основан на возможности или невозможности точного предсказания значений сигнала в любой момент времени или в любой точке пространственной координаты. Соответственно сигналы, для которых возможно указанное предсказание, называются *детерминированными*, а сигналы, для которых невозможно точно предсказать значения, называют *случайными*. Случайные сигналы описываются случайными функциями, значения которых при каждом данном значении аргумента представляются случайными величинами. Случайную функцию времени называют *случайным процессом*. При одном наблюдении случайного процесса получают определенную функциональную зависимость, которую называют *реализацией*. Примером реализации случайного процесса может служить отрезок сигнала $u(t)$, зареги-

стрированный на выходе микрофона при произнесении какого-либо шипящего звука (“с”, “ш” и т.п.). Примером детерминированного сигнала является гармоническое колебание $u(t) = A\cos(\omega t)$, для которого можно точно определить значение в любой момент времени.

Сигналы, значения которых изменяются непрерывно при изменении непрерывной временной t или пространственной переменной s, называются *непрерывными сигналами* (рис.1.1,а). Непрерывные сигналы часто называют *аналоговыми* (континуальными) сигналами.

Наряду с непрерывным способом передачи и преобразования сигналов, в настоящее время широко применяют дискретные способы, в которых в том или ином виде используется дискретизация сигналов. *Дискретизация* сигналов состоит в замене “непрерывных” значений теми или иными дискретными значениями и может быть осуществлена по времени (или пространственной переменной), по уровню, или по времени и уровню одновременно.

Дискретизация сигнала по времени соответствует выделению значений сигнала при фиксированных значениях t (рис.1.1,б). Обычно моменты взятия значений (отсчетов) сигнала отстоят друг от друга на величину T_0 (или S_0), называемую соответственно интервалом или шагом дискретизации. Сигнал, получаемый в ходе указанного процесса, называется дискретным сигналом.

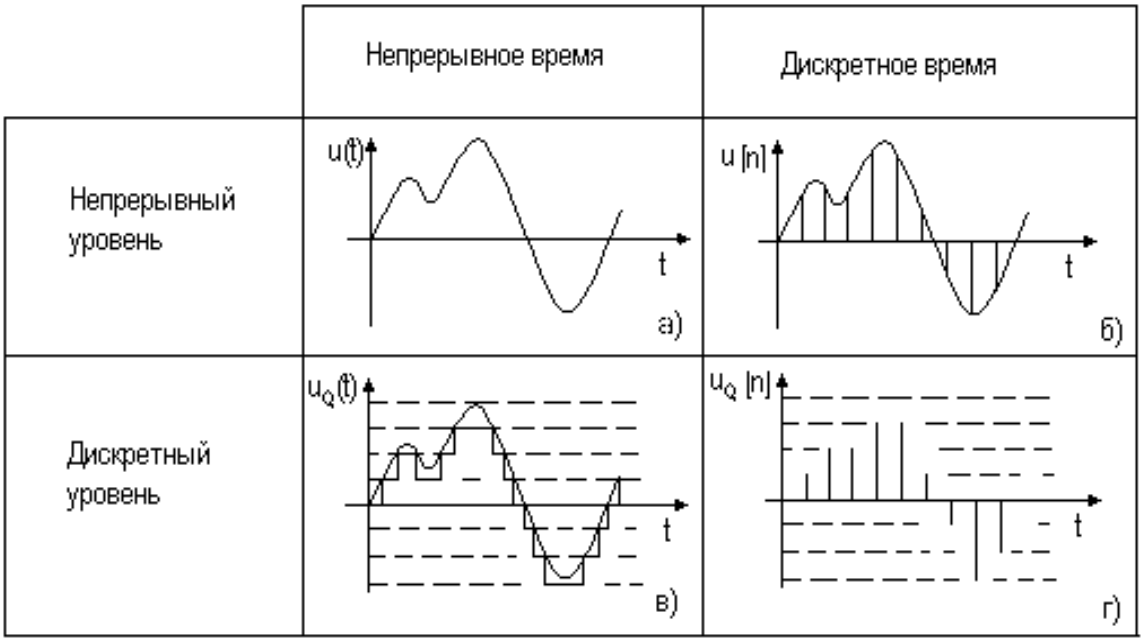


Рис.1.1. Дискретизация и квантование сигналов

Дискретный сигнал представляется множеством точек nT_0, в каждой из которых определено отсчетное значение сигнала $u[n] = u(nT_0)$. Таким образом, дискретизация по времени заменяет непрерывную функцию $u(t)$ решетчатой, которая определяется совокупностью выделенных ординат или дискрет $u[n]$, следующих через равные интервалы T_0. Отметим, что для различия обозначений непрерывных и дискретных функций в литературе используются соответственно круглые (.) и квадратные [.] скобки.

Дискретизация сигнала по уровню, которая чаще называется *квантованием по уровню* (рис.1.1,в), соответствует выделению значений сигнала при достижении им заранее фиксированных уровней. Уровни обычно отстоят друг от друга на постоянную величину, называемую шагом квантования по уровню.

Одновременная дискретизация сигнала по времени и по уровню соответствует выделению в заданные моменты времени значений сигнала, ближайших к заранее фиксированным уровням (рис.1.1, г). Сигналы, дискретизированные как по времени, так и по уровню, называются *цифровыми*. Цифровые сигналы отличаются от дискретных тем, что для них отсчетные значения представлены в виде чисел. Для формирования цифровых сигналов применяют аналого-цифровые преобразователи (АЦП), которые выполняют дискретизацию аналогового сигнала по времени и уровню, а затем кодируют уровень сигнала, используя ту или иную систему счисления. Цифровые сигналы находят все большее применение при решении различных задач, поскольку могут обрабатываться с помощью программируемых цифровых вычислительных устройств, обладающих известными достоинствами [31].

1.2 Преобразования Фурье непрерывных сигналов

1.2.1 Разложение периодических сигналов в ряд Фурье

В теории и практике обработки сигналов часто встречаются сигналы, которые могут рассматриваться как периодические. Сигнал *x(t)* называется периодическим, если для него выполняется условие

$$x(t) = x(t + nT), \tag{1.1}$$

где *n*- целое число, а T- период сигнала.

Примером простейшего периодического сигнала является гармоническое колебание

$$x(t) = A\cos(\omega t - \varphi). \tag{1.2}$$

Такой сигнал представляет гармонику, которая характеризуется амплитудой A, круговой частотой ω и начальной фазой φ . Нетрудно убедиться, что гармоника имеет период $T = 2\pi / \omega$. Сложение гармоник вида (1.2) с кратными частотами ω, 2ω, 3ω,… приводит к образованию периодического сигнала $x(t)$ с периодом, равным периоду первой гармоники $T = 2\pi/\omega$. В качестве примера рассмотрим сигнал

$$x(t) = \cos t + \cos 2t + 0{,}5\cos 3t\,. \qquad (1.3)$$

Суммарный сигнал $x(t)$ будет периодическим (рис 1.2). Периодическим будет и сигнал

$$x(t) = \sum_{\kappa=1}^{\infty} A_k \cos(\, k\omega t - \phi_k\,)\,, \qquad (1.4)$$

состоящий из бесконечного числа гармоник. Период этого сигнала также равен T .

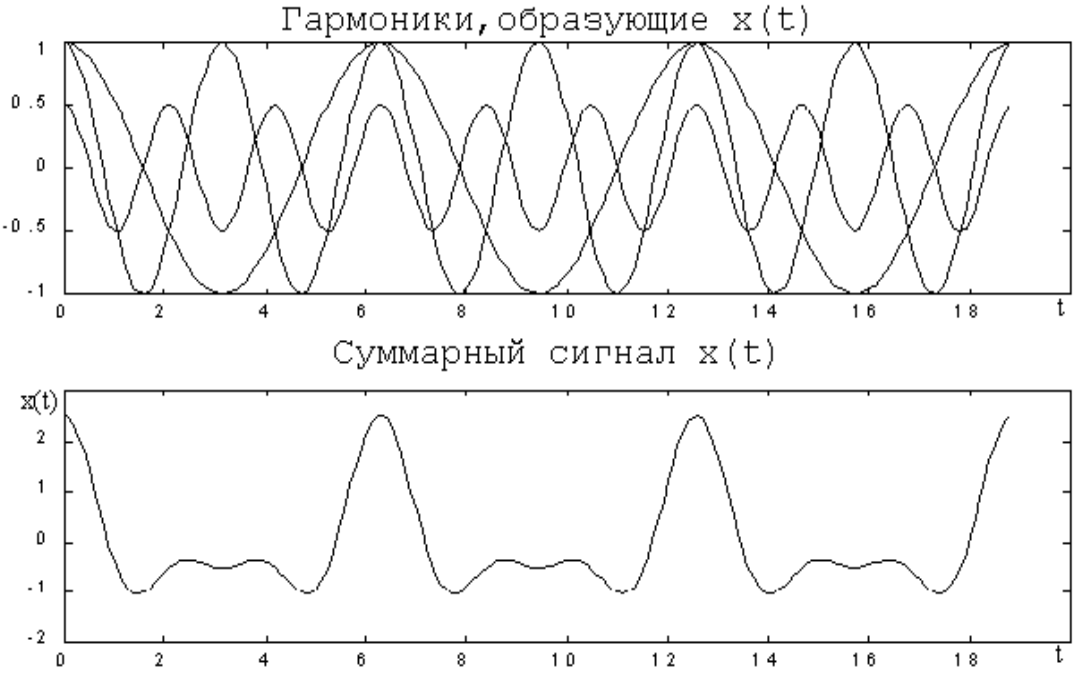


Рис.1.2. Разложение периодического сигнала

В теории сигналов доказывается, что любой периодический сигнал $x(t)$, имеющий на периоде конечное число точек разрыва первого рода, может быть представлен в виде (1.4) [1,5].

Иными словами, периодический сигнал $x(t)$ сложной формы может быть разложен на элементарные гармонические колебания с амплитудой A_k, частотой $k\omega_1$ и начальной фазой φ_k . Здесь ω_1 - круговая частота первой гармоники.

Если учесть, что $A_k \cos(k\omega_1 t - \varphi_k) = A_k \cos k\omega_1 t \cos\varphi_k + A_k \sin k\omega_1 t \sin\varphi_k$ и ввести обозначения $a_k = A_k \cos\varphi_k$, $b_k = A_k \sin\varphi_k$, то выражение (1.4) можно переписать в более удобной форме:

$$x(t) = a_0 / 2 + \sum_{k=1}^{\infty} (a_k \cos k\omega_1 t + b_k \sin k\omega_1 t) . \qquad (1.5)$$

Здесь $A = a_0 / 2$ представляет нулевую гармонику. Выражение (1.5) представляет разложение периодического сигнала $x(t)$ в *ряд Фурье*. При этом коэффициенты ряда Фурье могут быть определены по следующим формулам [1,5]:

$$a_0 = \frac{2}{T} \int_{-T/2}^{T/2} x(t) dt , \qquad (1.6)$$

$$a_k = \frac{2}{T} \int_{-T/2}^{T/2} x(t) \cos k\omega_1 t dt , \qquad (1.7)$$

$$b_k = \frac{2}{T} \int_{-T/2}^{T/2} x(t) \sin k\omega_1 t dt . \qquad (1.8)$$

Легко заметить, что амплитуда A_k и начальная фаза φ_k в разложении (1.4) равны:

$$A_k = \sqrt{a_k^2 + b_k^2} \quad \text{и} \quad \varphi_k = arctg(b_k / a_k) .$$

Если $x(t)$ является четной функцией на интервале $[-T/2, T/2]$, то произведение $x(t)\cos k\omega_1 t$ будет представлять собой четную функцию, а произведение $x(t)\sin k\omega_1 t$ - нечетную. В этом случае коэффициенты b_k ряда Фурье равны нулю, так как интеграл (1.8) от нечетно-симметричной функции будет равен нулю. И наоборот, если $x(t)$ является нечетной функцией на интервале $[-T/2, T/2]$, то равны нулю коэффициенты a_k. Это свойство коэффициентов a_k и b_k широко используется на практике.

Совокупность коэффициентов A_k и φ_k разложения (1.4) представляет *амплитудный и фазовый частотные спектры* периодического сигнала $x(t)$. Графическое построение, наглядно интерпретирующее коэффициенты ряда Фурье, называется *спектральной диаграммой* (рис.1.3). Различают амплитудные и фазовые диаграммы. На рис 1.3 по оси абсцисс отложены номера (частоты) гармоник, а по оси ординат значения коэффициентов A_k или φ_k .

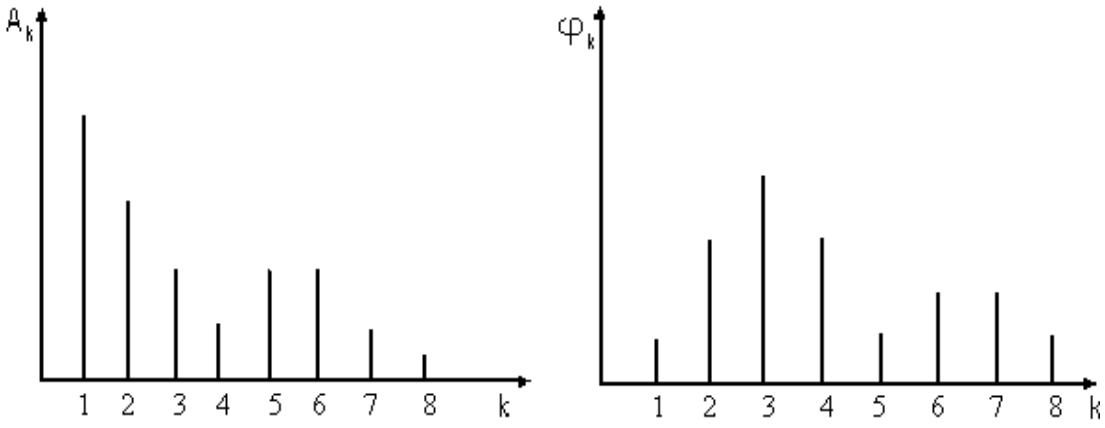


Рис. 1.3. Спектральные диаграммы периодического сигнала

Отметим важное свойство системы функций

$$\cos\omega_1 t,\ \sin\omega_1 t,\ \cos 2\omega_1 t,\ \sin 2\omega_1 t,\ ...,\ \cos k\omega_1 t,\ \sin k\omega_1 t, \qquad (1.9)$$

используемых в разложении (1.5). Это свойство состоит в том, что интеграл, взятый от произведения любых двух функций на периоде $T=2\pi/\omega_1$, равен нулю, т.е.

$$\int_{\rho}^{\rho+T}\cos k\omega_1 t dt = 0;\ \int_{\rho}^{\rho+T}\sin k\omega_1 t dt = 0;.\ \int_{\rho}^{\rho+T}\sin k\omega_1 t\cdot\sin l\omega_1 t dt = 0;$$
$$\int_{\rho}^{\rho+T}\cos k\omega_1 t\cdot\cos l\omega_1 t dt = 0;\ \int_{\rho}^{\rho+T}\sin k\omega_1 t\cdot\cos l\omega_1 t dt = 0,$$

где p - действительное число; l - натуральное число. Указанное свойство называется *ортогональностью*, а разложение (1.5) называют разложением *x(t)* по ортогональному базису функций вида (1.9). Базис (1.9) не является единственным. Возможно разложение сигнала и по другим ортогональным базисам. Способы построения ортогональных базисных функций исследуются в математике.

Рассмотрим разложение $x(t)$ по системе ортогональных комплексных экспоненциальных функций [1,5].

Перепишем (1.5) в компактной симметричной форме, воспользовавшись представлением гармонических функций в виде суммы комплексных экспонент в соответствии с формулами Эйлера. Тогда получим

$$x(t)=\frac{a_0}{2}+\sum_{k=1}^{\infty}\left(a_k\frac{e^{jk\omega_1 t}+e^{-jk\omega_1 t}}{2}+b_k\frac{e^{jk\omega_1 t}-e^{-jk\omega_1 t}}{2j}\right).$$

Введем новые коэффициенты $c_k = (a_k - jb_k)/2$. Коэффициенты c_k можно определить и при отрицательных индексах k. Причем $c_{-k} = (a_k + jb_k)/2$, так как коэффициенты a_k четны, а b_k нечетны относительно индексов. Распространив суммирование в (1.5) и на отрицательные индексы и обозначив $c_0 = a_0/2$, получим

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_1 t} . \qquad (1.10)$$

Формула (1.10) представляет *комплексный ряд Фурье*. Коэффициенты c_k такого ряда определяются выражением

$$c_k = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-jk\omega t} dt . \qquad (1.11)$$

В общем случае коэффициенты c_k имеют комплексные значения. Представим эти коэффициенты в экспоненциальной форме: $c_k = A_k e^{j\cdot\varphi_k}$ и $c_{-k} = A_k e^{-j\cdot\varphi_k}$. Тогда $c_k^* = c_{-k}$, где * - обозначает комплексно-сопряженное значение. В этом случае пара слагаемых разложения (1.10) с положительными и отрицательными значениями индексов представляет гармонику

$$A_k e^{j(k\omega_1 t + \varphi_k)} + A_k e^{-j(k\omega_1 t + \varphi_k)} = 2A_k \cos(k\omega_1 t + \varphi_k) ,$$

которая характеризуется амплитудой $2A_k e^{j\varphi_k} = C_k$.

Отметим, что $C_k = 2c_k$, k=1,2,… и $C_0 = c_0$. Совокупность коэффициентов C_k или c_k образует *комплексный частотный спектр сигнала*. Такие спектры обычно строят только для положительных значений индексов k.

Формулы (1.10) и (1.11) образуют пару преобразований Фурье. Формула (1.10) представляет произвольный сигнал $x(t)$ в виде суммы гармоник с комплексной амплитудой $2c_k$. Формула (1.11) позволяет выполнить обратные преобразования. Иными словами, между периодическими сигналами и их частотными спектрами существует взаимно однозначное соответствие. Выражение (1.10) позволяет перейти от спектра к сигналу, а выражение (1.11) – от сигнала к спектру. Поэтому при решении различных задач можно операции над периодическими сигналами $x(t)$

заменять операциями над частотными спектрами, характеризующими эти сигналы. Это дает возможность исследовать свойства сигналов не только во временной области, анализируя непосредственно сигнал $x(t)$, но и в частотной, оперируя коэффициентами c_k .

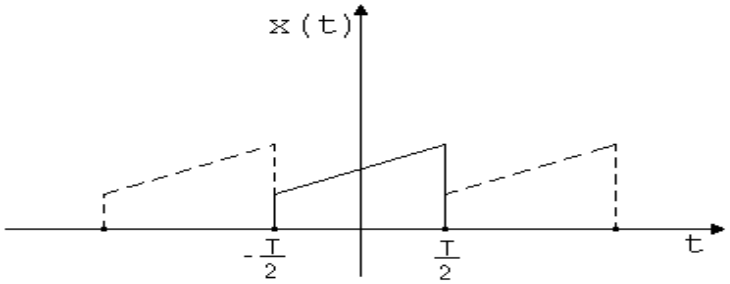


Рис.1.4. Периодическое продолжение непериодического сигнала

В виде суммы гармонических составляющих могут быть представлены не только периодические сигналы, но и непериодические сигналы, заданные на интервале $[-T/2, T/2]$. Разложение в ряд Фурье непериодического сигнала совпадает с разложением в ряд Фурье сигнала, периодически продолженного за пределы интервала $[-T/2, T/2]$. Сигнал, полученный в результате такого продолжения, будет периодическим с периодом T . На интервале $[-T/2, T/2]$ этот сигнал совпадает с исходным непериодическим сигналом $x(t)$ (рис.1.4).

Пример 1.1 Найдем спектр последовательности видеоимпульсов (рис.1.5).

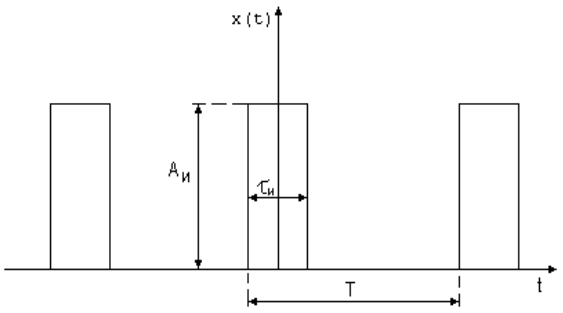


Рис.1.5. Последовательность видеоимпульсов

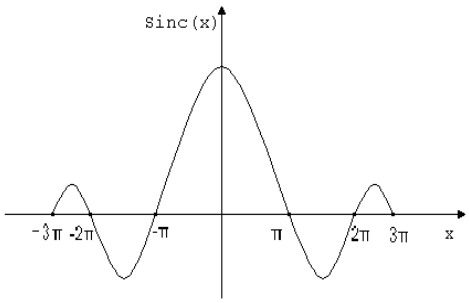


Рис. 1.6. Функция sinc(x)

Воспользуемся формулой (1.11). Выполнив необходимые подстановки, получим

$$c_k = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\cdot e^{-jk\omega_1 t}dt = \frac{1}{T}\int_{-\tau/2}^{\tau/2} A\cdot e^{-jk\omega_1 t}dt = \frac{A}{T}\cdot\frac{e^{-jk\omega_1 t}}{-jk\omega_1}\Bigg|_{\frac{-\tau}{2}}^{\frac{\tau}{2}} =$$

$$= \frac{A}{T} \cdot 2 \frac{\sin k\omega_1 \frac{\tau}{2}}{k\omega_1} = \frac{A}{\pi q} \cdot \frac{\sin k\pi/q}{k/q},$$

где $T = \frac{2\pi}{\omega_1}$ и $q = T/\tau$ - скважность последовательности.

Перепишем полученное выражение в виде

$$c_k = \frac{A}{q} \cdot \frac{\sin k\pi/q}{k\pi/q}.$$

Форма спектра последовательности видеоимпульсов соответствует функции $\sin(x)/x$, которая обозначается *sinc(x)* и имеет вид, показанный на рис.1.6.

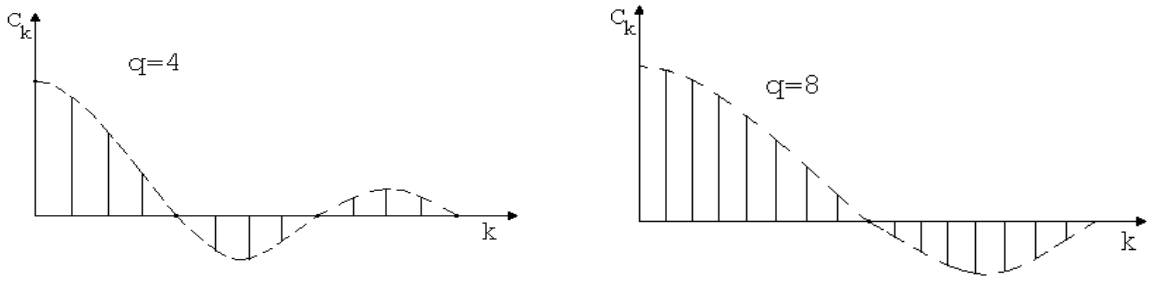


Рис. 1.7. Спектр последовательности видеоимпульсов

Найденный спектр является дискретным и существует только для частот $k\omega_1$. Амплитуды частотных составляющих спектра пропорциональны значениям функции $\sin c\,(k\pi/q)$ и являются действительными значениями.

Отметим, что последовательность более коротких импульсов (с большими значениями q) имеет более насыщенный спектр. Спектр последовательности видеоимпульсов для $q = 4$ и $q = 8$ показан на рис.1.7.

1.2.2 Интегральное преобразование Фурье

Рассмотренное выше представление сигналов в виде ряда Фурье справедливо для периодических сигналов. На практике нередко приходится рассматривать непериодические сигналы. Обобщим ряд Фурье на случай непериодических сигналов.

Рассмотрим сигнал $x_p(t)$, который представляет собой периодическую последовательность импульсов произвольной формы $x(t)$, следующих с периодом T. Тогда

$$x_p(t) = \sum_{n=-\infty}^{\infty} x(t - nT), \qquad (1.12)$$

где функция $x(t)$ описывает один импульс. Согласно (1.11) коэффициенты ряда Фурье для $x_p(t)$ на интервале [*-T/2, T/2*] равны

$$c_k = \frac{1}{T}\int_{-T/2}^{T/2} x_p(t)e^{-jk\omega_1 t}dt \quad . \tag{1.13}$$

Учитывая, что сигнал $x_p(t)$ на интервале [-T/2, T/2] представлен одним импульсом $x(t)$, а за пределами этого интервала $x(t)=0$, перепишем (1.13) в виде

$$c_k = \frac{1}{T}\int_{-\infty}^{\infty} x(t)e^{-jk\omega_1 t}dt \quad . \tag{1.14}$$

Из (1.14) следует, что при $T = const$ коэффициенты c_k зависят только от интеграла

$$X(k\omega_1) = \int_{-\infty}^{\infty} x(t)e^{-j\mathrm{k}\omega_1 t}\,dt \quad . \tag{1.15}$$

Комплексная функция частоты $X(k\omega_1)$ является спектральной характеристикой одиночного импульса *x(t).* Из сравнения (1.14) и (1.15) следует, что

$$c_k = \frac{X(k\omega_1)}{T} \quad . \tag{1.16}$$

Пределы интегрирования в (1.15) являются бесконечными. Это можно трактовать как разложение в ряд одиночного импульса на бесконечном интервале времени $(-\infty,\infty)$. Тогда сигнал *x(t)* можно записать в форме (1.10), если положить $T \to \infty$

$$x(t) = \lim_{T\to\infty}\sum_{k=-\infty}^{\infty} c_k e^{jk\omega_1 t} = \lim_{T\to\infty}\sum_{k=-\infty}^{\infty}\frac{X(k\omega_1)}{T}\cdot e^{jk\omega_1 t} \quad . \tag{1.17}$$

Подставляя значение $T = 2\pi/\omega$, получаем

$$x(t) = \lim_{T\to\infty}\frac{1}{2\pi}\sum_{k=-\infty}^{\infty} X(k\omega_1)e^{jk\omega_1 t}\omega_1 \quad . \tag{1.18}$$

При $T \to \infty$ частота первой гармоники $\omega_1 = 2\pi/T$ становится бесконечно малой. Приращение частоты ω_1 при переходе к соседней гармонике можно отождествить с дифференциалом $d\omega$. Под знаком суммы в (1.18) частоты гармоник принимают дискретные значения. Если же $T \to \infty$, то частоты гармоник становятся бесконечно близкими. Введем обозначение $\omega = k\omega_1$. В этом случае в выражении (1.18) операция суммирования переходит в операцию интегрирования. Выражения (1.15) и (1.18) принимают вид:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) \cdot e^{-j\omega t} dt \,, \tag{1.19}$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) \cdot e^{j\omega t} d\omega \,. \tag{1.20}$$

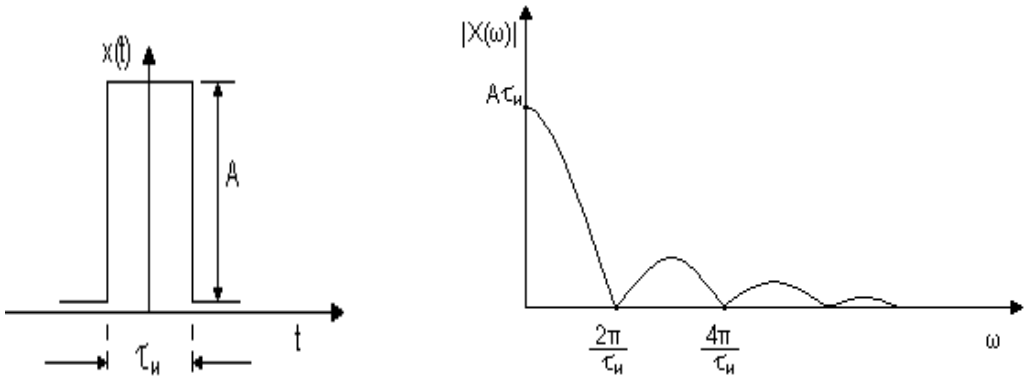
Формулы (1.19) и (1.20) представляют непериодический сигнал $x(t)$, заданный на интервале $(-\infty,\infty)$, соответственно в частотной и временной областях и называются *прямым и обратным интегральным преобразованием Фурье*. В теории сигналов доказывается, что интегральное преобразование Фурье существует не для всех сигналов, а только для *абсолютно интегрируемых*, т.е.

$$\int_{-\infty}^{\infty} |x(t)| dt < \infty \,.$$

Функция $X(\omega)$ характеризует спектральный состав сигнала $x(t)$ и называется *спектральной плотностью* сигнала $x(t)$. Такое название обусловлено тем, что для непериодического сигнала $x(t)$ (т.е. при $T \to \infty$) частотный интервал между смежными гармониками стремится к нулю, следовательно, преобразование (1.20) представляет собой разложение сигнала на сумму бесконечного числа гармоник, амплитуды которых бесконечно малы.

Таким образом, если с помощью ряда Фурье можно периодический сигнал разложить на бесконечное число гармоник с частотами, принимающими дискретные значения, то интегральное преобразование Фурье представляет непериодический сигнал в виде бесконечного числа гармоник, частоты которых бесконечно близки.

Пример 1.2 Определим спектральную плотность прямоугольного импульса амплитудой A и длительностью τ (рис 1.8,а). Воспользуемся формулой (1.19).



а) б)

Рис.1.8. Прямоугольный импульс и его спектральная плотность

После подстановки значений получим:

$$X(\omega)=\int_{-\infty}^{\infty} Ae^{-j\omega t}dt=\int_{-\tau/2}^{\tau/2} Ae^{-j\omega t}dt=\frac{A}{-j\omega}\cdot e^{-j\omega t}\Big|_{-\tau/2}^{\tau/2}=\frac{2A}{\omega}\cdot\sin(\omega\tau/2)=A\tau\cdot\frac{\sin(\omega\tau/2)}{\omega\tau/2}. \quad (1.21)$$

Таким образом, спектральная плотность прямоугольного импульса - вещественная функция частоты и имеет вид, показанный на рис.1.8,б.

Пример 1.3 Определим спектральную плотность сигнала, заданного выражением

$$x(t) = Ae^{-\alpha\cdot t}, \text{ где } \alpha\geq 0, \quad t\geq 0. \quad (1.22)$$

Используя интегральное преобразование Фурье, получаем

$$X(\omega)=\int_{0}^{\infty} Ae^{-(\alpha+j\omega)t}dt=-\frac{A}{\alpha+j\omega}\cdot e^{-(\alpha+j\omega)t}\Big|_{0}^{\infty}=\frac{A}{\alpha+j\omega}. \quad (1.23)$$

Спектральная плотность экспоненциального импульса - комплексная функция частоты. Модуль и аргумент функции соответственно равны:

$$|X(\omega)|=\frac{A}{\sqrt{\alpha^2+\omega^2}}, \qquad \varphi(\omega)=-arctg\frac{\omega}{\alpha}. \quad (1.24)$$

Пример 1.4 Определим спектральную плотность сигнала, описываемого дельта-функцией Дирака. Такая функция широко используется в теории сигналов и вводится следующим образом

$$\delta(t) = \begin{cases} \infty, t=0 \\ 0, t\neq 0 \end{cases}. \quad (1.25)$$

Дельта-функция всюду равна нулю, за исключением точки $t=0$, в которой она обращается в бесконечность. Дельта-функция обладает двумя важными свойствами:

1) интеграл от дельта-функции равен единице, т.е.

$$\int_{-\infty}^{\infty}\delta(t)dt=1; \tag{1.26}$$

2) интеграл от произведения дельта-функции и некоторого непрерывного сигнала $x(t)$ равен значению сигнала $x(t)$ в той точке, где сосредоточена δ -функция

$$x(t_0)=\int x(t)\delta(t-t_0)dt. \tag{1.27}$$

Указанное свойство называется фильтрующим свойством дельта-функции.

Для нахождения спектральной плотности сигнала $x(t)=A\cdot\delta(t)$ найдем преобразование Фурье

$$X(\omega)=\int_{-\infty}^{\infty}A\delta(t)e^{-j\omega t}dt\ . \tag{1.28}$$

В соответствии с фильтрующим свойством дельта-функции приведенный интеграл будет равен значению функции $Ae^{-j\omega t}$ в точке $t=0$, т.е. $X(\omega)=A$. Отсюда следует, что сигнал, описываемый дельта-функцией, имеет равномерный спектр. Приведем еще два важных соотношения для дельта-функции. Обратное преобразование Фурье для дельта-функции $(A=1)$

$$\delta(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}e^{j\omega t}d\omega\ . \tag{1.29}$$

Формально заменив переменную интегрирования ω на t, получим

$$\delta(\omega)=\frac{1}{2\pi}\int_{-\infty}^{\infty}e^{j\omega t}dt\ . \tag{1.30}$$

1.2.3 Свойства интегрального преобразования Фурье

Преобразование Фурье устанавливает взаимосвязь между представлением сигналов во временной и частотной области. При этом желательно знать как изменится представление сигнала в частотной области, если сигнал подвергнуть некоторым преобразованиям во временной области, и наоборот. Указанные соответствия устанавливаются с помощью свойств преобразования Фурье. Приведем их без доказательств [1,2,18].

1 *Свойство вещественной и мнимой частей спектральной плотности действительных сигналов.* Вещественная часть спектральной плотности $X_R(\omega)$ является четной функцией частоты, а мнимая часть $X_I(\omega)$ - нечетной функцией, т.е.

$$X_R(-\omega)=X_R(\omega)\quad \text{и}\quad X_I(-\omega)=-X_I(\omega)\ . \tag{1.31}$$

Это легко показать, подставив в интеграл (1.20) $X(\omega)=X_R(\omega)-j\cdot X_I(\omega)$ и заменив комплексную экспоненту суммой тригонометрических функций.

2 *Свойство линейности.* Спектральная плотность линейной комбинации нескольких сигналов

$$x(t)=a_1x_1(t)+a_2x_2(t) \tag{1.32}$$

представляет собой такую же линейную комбинацию спектральных плотностей

$$X(\omega)=a_1X_1(\omega)+a_2X_2(\omega). \tag{1.33}$$

3 *Свойство запаздывания.* Преобразование Фурье сигнала $x(t)$, запаздывающего на время τ_0, равно

$$x(t-\tau_0)\leftrightarrow X(\omega)e^{-j\omega\tau_0}. \tag{1.34}$$

Иными словами, задержка сигнала на время τ_0 приводит к умножению его спектральной плотности на комплексное число $e^{-j\cdot\omega\cdot\tau_0}$. Так как это число имеет единичный модуль, то при сдвигах сигнала во временной области амплитудный спектр не изменяется, а фазовый спектр получает приращение $\omega\tau_0$, линейно зависящее от частоты.

4 *Свойство масштабирования.* Преобразование Фурье сигнала $x(t)$ при изменении масштаба времени в k раз равно

$$x(kt)\leftrightarrow\frac{1}{k}X(\frac{\omega}{k}). \tag{1.35}$$

Отсюда следует, что “сжатие” исходного сигнала во времени в k раз ($k>1$) приводит к “растяжению” спектральной плотности по оси частот в k раз. Например, спектральная плотность прямоугольного импульса длительностью τ “сжатого” в k раз определяется выражением

$$X(\omega)=\frac{A\tau}{k}\cdot\frac{\sin\omega\tau/(2k)}{\omega\tau/(2k)}.$$

Если частота, соответствующая первому нулю спектральной плотности исходного импульса, равна $\omega=2\pi/\tau$, то частота первого нуля спектральной плотности “сжатого” импульса равна $2\pi k/\tau$, т.е. в k раз выше. А это

означает, что при “сжатии” сигнала во времени происходит расширение спектральной плотности сигнала.

5 *Спектральная плотность производной*. Интегральное преобразование Фурье производной сигнала $x(t)$ определяется соотношением

$$x'(t) \leftrightarrow j\omega X(\omega)\,. \tag{1.37}$$

Из (1.37) следует, что дифференцирование сигнала приводит к росту значений спектральной плотности в области высоких частот.

6 *Спектральная плотность интеграла* сигнала *x(t)* определяется соотношением

$$\int_{-\infty}^{t} x(\tau)d\tau \leftrightarrow X(\omega)/j\omega\,. \tag{1.38}$$

Интегрирование сигнала приводит к уменьшению значений спектральной плотности исходного сигнала в области высоких частот.

7 *Преобразование Фурье произведения сигналов* $x(t)\cdot u(t)$ определяется соотношением

$$x(t)\cdot u(t) \leftrightarrow \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\xi)\cdot U(\xi-\omega)d\xi\,. \tag{1.39}$$

Интеграл, стоящий в правой части, называют сверткой функций $X(\omega)$ и $U(\omega)$ и обозначают $X(\omega)*U(\omega)$. Тогда

$$x(t)\cdot u(t) \leftrightarrow \frac{1}{2\pi} X(\omega)*U(\omega)\,. \tag{1.40}$$

Таким образом, произведению сигналов во временной области соответствует свертка спектральных плотностей сигналов в частотной области.

8 *Преобразование Фурье свертки сигналов* во временной области определяется с помощью выражения

$$\int_{-\infty}^{\infty} x(t-\tau)\cdot u(\tau)d\tau \leftrightarrow X(\omega)\cdot U(\omega) \tag{1.41}$$

или

$$x(t)*u(t) \leftrightarrow X(\omega)\cdot U(\omega)\,.$$

Аналогичные свойства могут быть установлены и для ряда Фурье.

В заключение рассмотрим вычисление спектральных плотностей некоторых сигналов.

Пример 1.5 Определим спектральную плотность комплексного экспоненциального сигнала $x(t) = \exp(j\omega_0 t)$. Этот сигнал не является абсолютно интегрируемым, поэтому к нему нельзя применить непосредственно прямое преобразование Фурье. Чтобы найти спектральную плотность $X(\omega)$ комплексного экспоненциального сигнала, воспользуемся следующим приемом. Запишем обратное преобразование Фурье от $X(\omega)$

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega = e^{j\omega_0 t}$$

и подберем функцию $X(\omega)$ так, чтобы приведенное равенство превратилось тождество. Равенство обращается в тождество, если предположить, что $X(\omega) = 2\pi\delta(\omega - \omega_0)$. Действительно , выполнив указанную подстановку , получим

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} 2\pi\delta(\omega - \omega_0)e^{j\omega t}dt = e^{j\omega_0 t}.$$

Таким образом, спектральная плотность комплексного экспоненциального сигнала отображается дельта-функцией, сосредоточенной в точке ω_0, т.е. $2\pi\delta(\omega - \omega_0)$.

Пример 1.6 Найдем спектральную плотность гармонического колебания $x(t) = A\cos\omega_0 t$. Гармонический сигнал не является абсолютно интегрируемым. Поэтому найдем его спектральную плотность, используя результат предыдущего примера. В соответствии с формулой Эйлера

$$x(t) = A\cos\omega_0 t = A\left(e^{j\omega_0 t} + e^{-j\omega_0 t}\right)/2.$$

Отсюда следует, что спектральная плотность гармонического сигнала равна сумме спектральных плотностей комплексных экспоненциальных сигналов $e^{j\omega_0 t}$ и $e^{-j\omega_0 t}$, т.е.

$$X(\omega) = A\pi\left[\delta(\omega-\omega_0)+\delta(\omega+\omega_0)\right]. \qquad (1.42)$$

На рис.1.9 изображен график спектральной плотности гармонического сигнала.

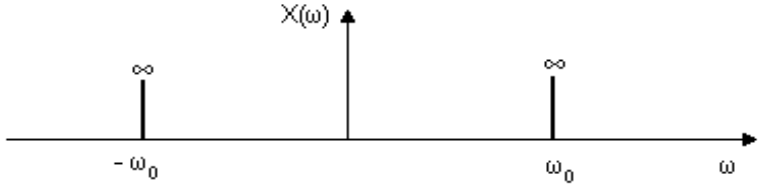


Рис.1.9. Спектральная плотность гармонического сигнала

Пример 1.7 Найдем спектральную плотность сигнала $x(t)\cos\omega_0 t$. Воспользуемся свойством преобразования Фурье

$$x(t)\cdot u(t) \leftrightarrow \frac{1}{2\pi} X(\omega) * U(\omega).$$

В рассматриваемом случае $u(t) = \cos\omega_0 t$ и

$$U(\omega) = \pi\left[\delta(\omega-\omega_0)+\delta(\omega+\omega_0)\right].$$

Тогда спектральная плотность произведения равна

$$\frac{1}{2}\int_{-\infty}^{\infty} X(\xi)\cdot\left[\delta(\xi-(\omega-\omega_0))+\delta(\xi-(\omega+\omega_0))\right]d\xi = \frac{1}{2}[X(\omega-\omega_0)+X(\omega+\omega_0)]. \quad (1.43)$$

Таким образом, при умножении сигнала *x(t)* на $\cos\omega_0 t$ спектральная плотность $X(\omega)$ смещается влево и вправо на значение ω_0. Отметим, что умножение сигнала *x(t)* на $\cos\omega_0 t$ соответствует амплитудной модуляции.

1.2.4 Обобщенный ряд Фурье и системы базисных функций

Рассмотрим разложение сигнала $x(t)$ на интервале $[-T/2, T/2]$ в ряд Фурье

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j\omega_1 kt}. \quad (1.44)$$

В общем случае выражение (1.44) представляет собой разложение сигнала $x(t)$ по некоторой системе функций $\{\varphi_k(t)\} = \{\exp(-j\omega_1 kt)\}$ и может быть записано в форме *обобщенного ряда Фурье*

$$x(t) = \sum_k c_k \varphi_k(t). \quad (1.45)$$

Если предположить, что сигнал $x(t)$ - это вектор, то выражение (1.45) можно интерпретировать как разложение вектора по некоторому базису, а коэффициенты c_k можно рассматривать как проекции вектора на координатные оси, заданные системой функций $\{\varphi_k(t)\}$, образующих базис. Для того чтобы разложение (1.45) было возможно, система функций $\{\varphi_k(t)\}$ должна удовлетворять определенным условиям.

Во-первых, сигнал $x(t)$ должен принадлежать множеству квадратично-интегрируемых на отрезке T сигналов, т.е.

$$\int_T [\, x\,(t)\,]^{\,2}\, dt \;<\; \infty \quad . \tag{1.46}$$

Такое множество сигналов образует пространство сигналов и обозначается $L^2(T)$. Отметим, что в качестве отрезка квадратичной интегрируемости могут рассматриваться как конечные, так и бесконечные интервалы. Соответственно образуются различные виды пространств: $L^2(-\infty,\infty)$, $L^2[-1,1]$ и т.д. Пространство L^2 замкнуто относительно линейных операций, т.е. если $x_1(t) \in L^2$ и $x_2(t) \in L^2$, то $\alpha \cdot x_1(t) + \beta \cdot x_2(t) \in L^2$, поэтому его называют *линейным векторным пространством*. Сигналы $x_1(t)$ и $x_2(t)$ рассматриваются как векторы в линейном пространстве, для которых определено *скалярное произведение*

$$(x_1, x_2) = \int_T x_1(t)\, x_2(t)\, dt \;<\; +\infty \tag{1.47}$$

и *норма* (длина вектора)

$$\|x\| = \left\{ \int_T \left[x\,(t) \right]^2 dt \right\}^{1/2} \quad . \tag{1.48}$$

Для скалярного произведения справедливо соотношение, называемое неравенством Коши-Буняковского:

$$|(x_1, x_2)| \le \|x_1\| \cdot \|x_2\| . \tag{1.49}$$

Отношение

$$\frac{(x_1, x_2)}{\|x_1\| \cdot \|x_2\|} = \cos\Theta \tag{1.50}$$

представляет косинус угла между сигналами (векторами) $x_1(t)$ и $x_2(t)$. Два сигнала (вектора) $x_1(t)$ и $x_2(t)$ будут являться *ортогональными*, если скалярное произведение этих сигналов равно нулю, т.е. (x_1, x_2)=0.

Во-вторых, функции $\{\varphi_k(t)\}$ должны быть попарно ортогональны, т.е.

$$(\varphi_i, \varphi_k) = \begin{cases} 0, & i \ne k \\ 1, & i = k \end{cases} . \tag{1.51}$$

Система функций (1.51) имеет единичную норму и образует *ортонормированный базис.*

Если указанные выше условия выполняются, то коэффициенты обобщенного ряда Фурье могут быть найдены следующим образом. Умножим левую и правую часть (1.45) на базисную функцию $\varphi_i(t)$ и проинтегрируем произведение на интервале T. В результате этого получим

$$\int_T x(t)\cdot\varphi_i(t)dt = \sum_k c_k \int_T \varphi_k(t)\cdot\varphi_i(t)dt \qquad (1.52)$$

или

$$\int_T x(t)\varphi_i(t)dt = \sum_k c_k(\varphi_k,\varphi_i)\ . \qquad (1.53)$$

Скалярное произведение функций (φ_k,φ_i) равно нулю для всех i и k, кроме $i=k$. Коэффициенты обобщенного ряда Фурье могут быть найдены из выражения

$$c_k = \frac{1}{(\varphi_k,\varphi_k)}\int_T x(t)\varphi_k(t)dt = \frac{(x,\varphi_k)}{\|\varphi_k\|^2}\ . \qquad (1.54)$$

Для ортонормированного базиса $c_k=(x,\varphi_k)$.

Обобщенный ряд Фурье (1.45) содержит бесконечное число членов. На практике приходится ограничивать ряд конечным числом членов *N*. Это приводит к появлению ошибки аппроксимации

$$\varepsilon(t) = x(t) - \sum_{k=1}^{N} c_k\varphi_k(t)\ . \qquad (1.55)$$

Обычно рассматривают норму ошибки

$$\|\varepsilon\| = \left\{\int_T [\varepsilon(t)]^2\,dt\right\}^{1/2}\ . \qquad (1.56)$$

Одним из важных свойств систем базисных функций является *полнота*. Базисные функции образуют полную систему, если норма ошибки аппроксимации представляет убывающую функцию от *N*. Для полной системы базисных функций, выбирая *N* достаточно большим, можно снизить норму ошибки до приемлемой величины.

Рассмотренное ранее разложение сигнала в ряд Фурье по системе тригонометрических функций с кратными частотами не является единственно возможным. Можно построить и другие системы ортогональных функций. Интерес к поиску других систем функций обусловлен тем, что норма ошибки аппроксимации для иных систем базисных функций может стать меньше при одном и том же числе членов ряда. Кроме того,

тригонометрический базис не всегда удобен с точки зрения технической реализации. Выбор базиса в значительной степени обусловлен спецификой решаемых задач.

Рассмотрим примеры систем ортогональных функций.

Пример 1.8 *Система мультипликативно - ортогональных функций*

Рассмотрим последовательность неперекрывающихся прямоугольных импульсов, имеющих значения в диапазоне от 0 до 1 (рис.1.10).

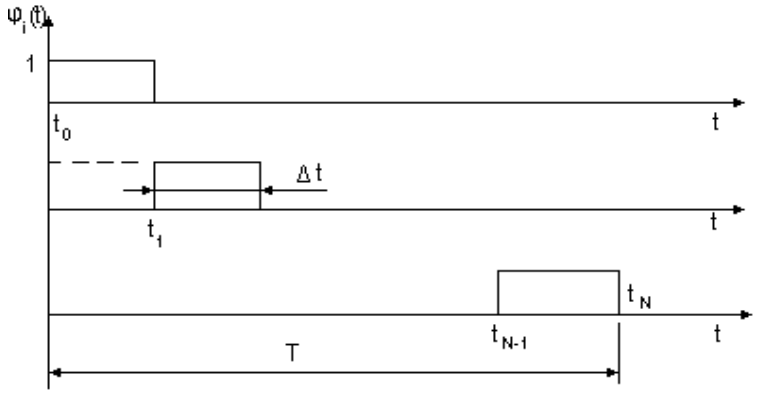


Рис 1.10. Мультипликативно-ортогональные функции

Так как любые два прямоугольных импульса не перекрываются по времени, то рассматриваемая система функций ортогональна. Действительно,

$$(\varphi_k, \varphi_i) = \int_0^T \varphi_k(t)\varphi_i(t)dt = 0 \qquad (1.57)$$

и

$$\|\varphi_k\|^2 = (\varphi_k, \varphi_k) = \int_0^T \varphi_k^2(t)dt = \int_{\Delta t(k-1)}^{\Delta t \cdot k} 1 \cdot dt = \Delta t = \frac{T}{N}. \qquad (1.58)$$

Коэффициенты ряда Фурье в мультипликативно-ортогональном базисе будут равны

$$c_k = \frac{(x, \varphi_k)}{\|\varphi_k\|^2} = \frac{N}{T}\int_0^T x(t)\varphi_k(t)dt = \frac{N}{T}\int_{\Delta t(k-1)}^{\Delta t \cdot k} x(t)dt. \qquad (1.59)$$

С помощью рассматриваемого базиса сигнал $x(t)$ может быть представлен в виде ряда

$$x(t) = \sum_{i=1}^{N} c_k \varphi_k(t). \qquad (1.60)$$

Система мультипликативно-ортогональных функций является полной только для подмножества ступенчатых сигналов с шириной ступени Δt.

Пример 1.9 *Полиномиальные системы базисных функций*
Такие системы строят на основе ортогональных полиномов. На практике широкое распространение получили системы базисных функций, построенные на основе полиномов Чебышева, Лежандра, Лагерра и др.[18]. Рассмотрим систему базисных функций, построенную на основе полиномов Чебышева. Эта система функций обеспечивает быстрое уменьшение нормы ошибки аппроксимации с ростом N.

Обозначим через $T_k(t)$ - полином Чебышева k - го порядка, задаваемый с помощью рекуррентного выражения

$$T_k(t) = 2t \cdot T_{k-1}(t) - T_{k-2}(t) \ , \tag{1.61}$$

где $T_0(t) = 1$ и $T_1(t) = t$.

Ниже приведены примеры полиномов Чебышева:

$$T_0(t) = 1;\ T_1(t) = t;\ T_2(t) = 2t^2 - 1;\ T_3(t) = 4t^3 - 3t;\ T_4(t) = 8t^4 - 8t^2 + 1, \tag{1.62}$$

Полиномы Чебышева обладают важным свойством: из всех многочленов *k*-й степени с одинаковыми коэффициентами при старшей степени аргумента они менее всего уклоняются от нуля на интервале $(-1, 1)$. В то же время при $|t| >> 1$ полиномы Чебышева резко увеличивает свои значения (рис.1.11).

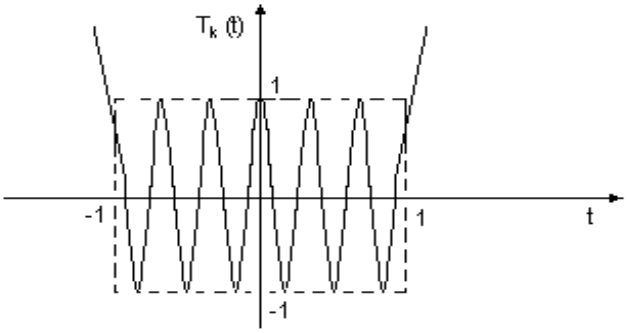


Рис.1.11 Поведение полиномов Чебышева

На интервале $(-1, 1)$ полиномы Чебышева описываются соотношением

$$T_k(t) = \cos(k \cdot \arccos t) \ , \tag{1.63}$$

т.е. значения полинома не превышают единицы и представляют косинусную функцию. Число полуволн полинома на интервале $(-1, 1)$ на единицу меньше порядка полинома. При $t >> 1$ значения полинома асимптотически равны

$$T_k(t) = 2^{k-1} t^k \ . \tag{1.64}$$

На основе полиномов Чебышева строится система ортогональных функций Чебышева. Функции Чебышева образуют полную систему ортогональных функций на интервале $(-1, 1)$ с весом

$$\rho(t) = \frac{1}{\sqrt[4]{1-t^2}} \ . \tag{1.65}$$

Система ортогональных функций Чебышева представляет множество

$$\{\varphi_k(t)\} = \left\{ \frac{1}{\sqrt[4]{1-t^2}} T_k(t) \right\}. \tag{1.66}$$

Условие ортогональности функций Чебышева:

$$(\varphi_k, \varphi_i) = \int_{-1}^{+1} \frac{1}{\sqrt{1-t^2}} T_k(t) T_i(t) dt = \begin{cases} 0, & k \neq i \\ \pi/2, & k = i \neq 0 \\ \pi, & k = i = 0 \end{cases}. \tag{1.67}$$

Отсюда следует, что коэффициенты обобщенного ряда Фурье с учетом нормировки вычисляются по формулам:

$$c_0 = \sqrt{\frac{1}{\pi}} \int_{-1}^{1} x(t) \frac{1}{\sqrt[4]{1-t^2}} dt \ ,$$

$$c_k = \sqrt{\frac{2}{\pi}} \int_{-1}^{1} x(t) \frac{T_k(t)}{\sqrt[4]{1-t^2}} dt \ . \tag{1.68}$$

Сигнал $x(t)$ в этом случае представляется в виде ряда

$$x(t) = c_0 \sqrt{\frac{1}{\pi}} \frac{1}{\sqrt[4]{1-t^2}} + \sum_{k=1}^{\infty} c_k \sqrt{\frac{2}{\pi}} \frac{T_k(t)}{\sqrt[4]{1-t^2}}. \tag{1.69}$$

Функции Чебышева удовлетворяют условию наилучшей аппроксимации $x(t)$ в смысле минимума среднего квадрата ошибки с весом $1/\sqrt{1-t^2}$. Аппроксимация произвольного сигнала функциями Чебышева позволяет добиться равномерного распределения погрешности и избежать ее накопления к концу интервала разложения.

Пример 1.10 *Непрерывные базисные системы функций Уолша*

Функции Уолша являются кусочно-постоянными знакопеременными функциями, которые определяют на интервале $[-0{,}5;\ 0{,}5]$ или $[0, 1]$. Интервал определения функций Уолша задается совокупностью $N = 2^n$, $n = 1, 2, \ldots$ равных подынтервалов. Функции Уолша образуют базисную систему $\{W_\alpha(Z)\}$, где α - номер функции Уолша, $\alpha = 0, 1, 2, \ldots$ $Z \in [-0{,}5;\ 0{,}5]$ или $[0, 1]$. Существуют различные системы функций Уолша, отличающиеся способом упорядоченности. Наиболее часто на практике применяют следующие системы функций Уолша: Пэли, Адамара и Хармута.

Для описания различных систем функций Уолша используют функции Радемахера $r_k(z)$:

$$r_0(z) = 1, \qquad r_k(z) = sign(\sin 2^k \pi z). \tag{1.70}$$

Функции Радемахера ортогональны и нормированы, но не образуют полную систему. Дополняя систему функций Радемахера определенными произведениями этих же функций, можно построить полные системы функций Уолша.

Рассмотрим систему функций Уолша - Пэли. Любую функцию такой системы можно записать в форме

$$W_\alpha(z) = \prod_{k=1}^{n} r_k(z)^{\alpha_k}, \tag{1.71}$$

где $\alpha_k = 0$ или 1 и соответствует коэффициентам двоичного разложения номера функции Уолша

$$\alpha = \sum_{k=1}^{n} \alpha_k 2^{k-1}. \tag{1.72}$$

В тех случаях, когда α представляет степень числа 2 (т.е. содержит только один единичный коэффициент α_k), функция Уолша совпадает с одной из функций Радемахера.

Рассмотрим систему функций Уолша - Пэли для $N = 8$ и $z \in [0, 1]$. В этом случае

$$\begin{gathered} W_0(z) = r_0(z); \quad W_1(z) = r_1(z); \quad W_2(z) = r_2(z); \\ W_3(z) = r_1(z) \cdot r_2(z); \quad W_4(z) = r_3(z); \quad W_5(z) = r_1(z) \cdot r_3(z); \\ W_6(z) = r_2(z) \cdot r_3(z); \quad W_7(z) = r_1(z) \cdot r_2(z) \cdot r_3(z). \end{gathered} \tag{1.73}$$

Графики функций Уолша-Пэли изображены на рис.1.12.

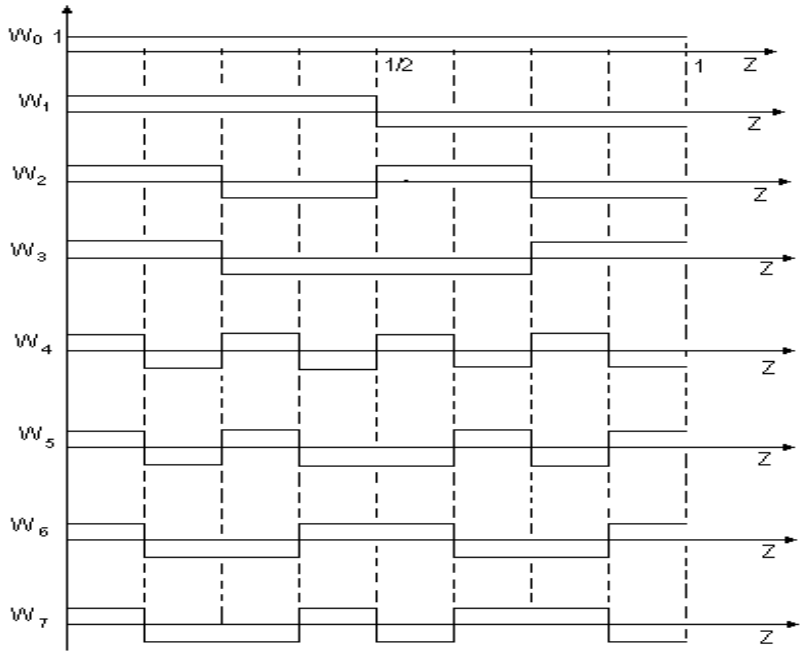


Рис. 1.12. Функции Уолша – Пэли

Ряд Фурье - Уолша сигнала $x(t)$, заданного на интервале $[0, T]$, будет иметь вид

$$x(t) = \sum_{\alpha=0}^{\infty} c_\alpha W_\alpha\left(\frac{t}{T}\right), \tag{1.74}$$

где

$$c_\alpha = \frac{1}{T}\int_0^T x(t)\cdot W_\alpha\left(\frac{t}{T}\right)dt \ . \qquad (1.75)$$

Так как функция Уолша имеет N участков с постоянными значениями, то

$$c_\alpha = \frac{1}{T}\sum_{i=0}^{N-1}\int_{t_i}^{t_{i+1}} x(t)\cdot W_\alpha\left(\frac{t}{T}\right)dt = \frac{1}{T}\sum_{i=0}^{N-1} W_\alpha\left(\frac{i}{N}\right)\int_{t_i}^{t_{i+1}} x(t)\,dt \ , \qquad (1.76)$$

где i - определяет номер участка постоянства функции Уолша .

1.3 Непрерывные системы

1.3.1 Импульсная и частотная характеристики системы

В системах обработки сигналов можно выделить: *вход*, предназначенный для подачи сигналов; *выход*, откуда обработанные сигналы поступают для дальнейшего использования; *внутренние переменные*, характеризующие состояние системы. Входной $x(t)$ и выходной сигнал (реакция системы) $y(t)$ системы обычно представляют собой скалярные функции времени. Однако в общем случае входные и выходные сигналы представляются в виде векторов:

$$\vec{X}(t) = \{x_1(t),\, x_2(t),\, ...,\, x_m(t)\}, \qquad (1.77)$$

$$\vec{Y}(t) = \{y_1(t),\, y_2(t),\, ...,\, y_n(t)\}. \qquad (1.78)$$

Система обработки сигналов, имеющая *m* входов и *n* выходов называется *многомерной*.

Если входной и выходной сигналы, а также состояния системы определены в каждый момент времени t и время непрерывно, то система называется *непрерывной*. Если указанные сигналы и состояния определены в дискретные моменты времени, система называется *дискретной.* Ниже рассматриваются основные понятия непрерывных систем. Такие системы исторически первыми применялись для обработки сигналов. Многие понятия, которые будут введены, имеют аналогию в дискретных и цифровых системах обработки сигналов.

Связь между сигналами *x(t)* и *y(t)* можно задать посредством системного оператора $O\{.\}$, выполняющего преобразование входного сигнала в выходной

$$y(t) = O\{x(t)\} \ . \qquad (1.79)$$

Система называется *стационарной*, если её выходная реакция не зависит от момента подачи входного сигнала $x(t)$, т.е.

$$y(t \pm t_0) = O\{x(t \pm t_0)\}, \quad \text{при любом } t. \qquad (1.80)$$

Если оператор системы таков, что выполняется *принцип суперпозиции*:

$$O\{x_1(t) + x_2(t)\} = O\{x_1(t)\} + O\{x_2(t)\} \qquad (1.81)$$

и

$$O\{\alpha \cdot x(t)\} = \alpha \cdot O\{x(t)\}, \qquad (1.82)$$

где α - произвольное число, то система называется *линейной*. Для линейной системы реакция на сумму сигналов $x_1(t)$ и $x_2(t)$ равна сумме реакций на каждый из указанных сигналов.

Строго говоря, все системы обработки сигналов являются нелинейными. Однако при определенных допущениях, они могут описываться линейными уравнениями. Замечательная особенность линейных систем состоит в том, что для них, благодаря принципу суперпозиции, легко определяется выходной сигнал по заданному входному сигналу.

Импульсной характеристикой системы называется функция $h(t)$, представляющая реакцию системы на входной сигнал, заданный дельта-функцией Дирака, т.е.

$$h(t) = O\{\delta(t)\}. \qquad (1.83)$$

Знание $h(t)$ позволяет формально решить любую задачу о прохождении детерминированного сигнала через линейную систему. Для линейной системы связь между входным и выходным сигналами при известной импульсной характеристике и нулевых начальных условиях задается *интегралом Дюамеля* [1,5,18]:

$$y(t) = \int_{-\infty}^{\infty} x(\tau) h(t - \tau) d\tau \qquad (1.84)$$

или

$$y(t) = \int_{-\infty}^{\infty} x(t - \tau) h(\tau) d\tau. \qquad (1.85)$$

Из (1.84) и (1.85) следует, что выходной сигнал линейной системы представляет свертку входного сигнала $x(t)$ и импульсной характеристики $h(t)$.

Для *физически реализуемых* систем $h(t) = 0,\ t < 0$. Это означает, что реакция системы на дельта-импульс не может возникнуть до момента подачи этого импульса на вход системы. Легко видеть, что для физически реализуемых систем интеграл Дюамеля обращается в ноль при $\tau > t$, поэтому (1.84) можно переписать в виде:

$$y(t) = \int_{-\infty}^{t} x(\tau)h(t-\tau)d\tau = x(t) * h(t) . \qquad (1.86)$$

Формула (1.86) имеет ясный физический смысл: линейная система выполняет взвешенное интегрирование всех мгновенных значений сигнала *x(t),* поступивших на вход к моменту времени *t*. Поэтому импульсную характеристику часто называют *весовой* функцией системы.

Импульсная характеристика линейной стационарной системы полностью определяет ее поведение и позволяет исследовать систему во временной области. Для исследования линейных систем в частотной области используют *частотную характеристику* $H(j\omega)$. Частотная $H(j\omega)$ и импульсная характеристика *h(t)* линейной стационарной системы связаны между собой парой преобразований Фурье [1,5,18]:

$$H(j\omega) = \int_{-\infty}^{\infty} h(t)e^{-j\omega t}dt , \qquad (1.87)$$

$$h(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(j\omega)e^{j\omega t}d\omega . \qquad (1.88)$$

Для линейных систем гармонические сигналы являются *собственными*, т.е. при прохождении гармонического сигнала через линейную систему он не изменяет своей формы. Иными словами, реакция линейной системы на гармонический входной сигнал есть тоже гармонический сигнал. Поэтому функция $H(j\omega)$ имеет простую интерпретацию. Она представляет коэффициент передачи гармонического сигнала с частотой ω со входа линейной системы на ее выход.

В общем случае $H(j\omega)$ имеет комплексные значения и связывает спектральные плотности входного и выходного сигнала простой зависимостью

$$Y(j\omega) = H(j\omega)X(j\omega) . \qquad (1.89)$$

Частотная характеристика может быть записана в показательной форме

$$H(j\omega) = \left|H(j\omega)\right| \cdot e^{j\varphi(\omega)} , \qquad (1.90)$$

где $\left|H(j\omega)\right|$ - *амплитудно-частотная характеристика* (АЧХ); $\varphi(\omega)$ - *фазо* -

- *частотная характеристика* (ФЧХ).

Так как импульсная характеристика $h(t)$ - вещественная функция, то из свойств преобразования Фурье следует, что

$$H(j\omega) = H^{*}(-j\omega) . \qquad (1.91)$$

Таким образом, АЧХ является четной, а ФЧХ - нечетной функцией частоты.

Пример 1.10 Определим импульсную характеристику $h(t)$ системы по заданной частотной характеристике. Частотная характеристика определяется уравнением (рис.1.13):

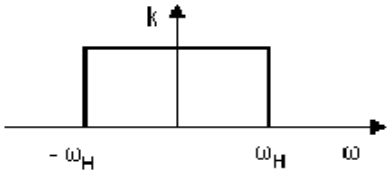


Рис.1.13. АЧХ

$$H(j\omega)=\begin{cases}0, & \omega<-\omega_H\\ k, & -\omega_H\le\omega\le\omega_H\\ 0, & \omega>\omega_H\end{cases}. \quad (1.92)$$

и соответствует идеальному фильтру нижних частот. Применив к $H(j\omega)$ обратное преобразование Фурье, получим

$$h(t)=\frac{k}{2\pi}\int_{-\omega_H}^{\omega_H} e^{j\omega t}d\omega=\frac{k\,\omega_H}{\pi}\cdot\frac{\sin\omega_H t}{\omega_H t}. \quad (1.93)$$

Так как $h(t)$ симметрична относительно точки $t=0$, то это свидетельствует о физической нереализуемости линейной системы с частотной характеристикой (1.92).

Далеко не каждая функция $H(j\omega)$ соответствует физически реализуемой системе. Существует *критерий Пэли - Винера*, который устанавливает, что для физически реализуемой системы должен существовать интеграл [1]:

$$\int_{-\infty}^{\infty}\frac{\ln\left|H(j\omega)\right|}{1+\omega^2}d\omega<+\infty. \quad (1.94)$$

Отсюда следует, что системы, АЧХ которых обращаются в нуль при некоторых частотах, физически нереализуемы.

1.3.2 Преобразование Лапласа и передаточная функция системы

Иная возможность описания линейных стационарных систем основана на использовании дифференциальных уравнений, которые также устанавливают соответствие между сигналами на входе и выходе системы:

$$a_n\frac{d^n y}{dt^n}+a_{n-1}\frac{d^{n-1}y}{dt^{n-1}}+\ldots+a_1\frac{dy}{dt}+a_0 y=b_m\frac{d^m x}{dt^m}+b_{m-1}\frac{d^{m-1}x}{dt^{m-1}}+\ldots+b_1\frac{dx}{dt}+b_0 x. \quad (1.95)$$

Для решения дифференциальных уравнений широко применяют операторный метод, основанный на преобразовании Лапласа. *Преобра-*

зование Лапласа аналогично преобразованию Фурье и задается парой уравнений[1,5,18]:

$$X(p) = \int_0^{\infty} x(t) e^{-pt} dt \quad , \tag{1.96}$$

$$x(t) = \frac{1}{2\pi j} \int_{c-j\cdot\infty}^{c+j\cdot\infty} X(p) e^{pt} dp \quad , \tag{1.97}$$

где $p = (\delta + j\omega)$ - комплексная частота. В этом случае сигнал $x(t)$ называют оригиналом, а функцию $X(p)$ - его изображением по Лапласу. Большинство свойств преобразования Лапласа соответствует аналогичным свойствам преобразования Фурье. Преобразование Лапласа определено только для сигналов, тождественно равных нулю при $t < 0$ и удовлетворяющих условию

$$|x(t)| \le k \cdot e^{\alpha t} \quad , \quad k, \alpha \text{ - положительные числа.} \tag{1.98}$$

Взяв преобразование Лапласа от обеих частей (1.95) с учетом $x'(t) \leftrightarrow pX(p)$, получим

$$(a_n p^n + a_{n-1} p^{n-1} + ... + a_1 p + a_0) Y(p) = (b_m p^m + b_{m-1} p^{m-1} + ... + b_1 p + b_0) X(p). \tag{1.99}$$

Отсюда отношение *Y(p)* к *X(p)* равно

$$H(p) = \frac{Y(p)}{X(p)} = \frac{b_m p^m + b_{m-1} p^{m-1} + ... + b_0}{a_n p^n + a_{n-1} p^n + ... + a_0} . \tag{1.100}$$

Данное отношение является важнейшей характеристикой линейной системы и называется *передаточной функцией.*

Корни уравнений

$$N(p) = a_n p^n + a_{n-1} p^{n-1} + ... + a_1 p^1 + a_0 = 0 \tag{1.101}$$

и

$$M(p) = b_m p^m + b_{m-1} p^{m-1} + ... + b_1 p^1 + b_0 = 0 \tag{1.102}$$

называются соответственно *полюсами и нулями* передаточной функции.

Передаточная функция представляет собой оператор, который преобразует входное воздействие линейной системы в выходную реакцию системы, при этом $m \le n$.

Если входное воздействие *x(t)* представляет собой дельта-импульс $\delta(t)$, то с учетом того, что его изображение по Лапласу равно единице, получаем следующее выражение

$$L\{h(t)\} = Y(p) = H(p), \qquad (1.103)$$

где $L\{\cdot\}$ - оператор преобразования Лапласа.
Отсюда импульсная характеристика будет равна

$$h(t) = L^{-1}\{H(p)\}. \qquad (1.104)$$

Таким образом, импульсная характеристика определяется через передаточную функцию с помощью обратного преобразования Лапласа $L^{-1}\{\cdot\}$, т.е. *h(t)* является оригиналом для $H(p)$. На практике для нахождения $h(t)$ по известной передаточной функции обычно используется формула

$$h(t) = \sum_{k=1}^{n} \frac{M(p)}{N'(p)} e^{p_k t}, \qquad (1.105)$$

где p_k - полюсы передаточной функции $H(p)$. Формула (1.105) справедлива, когда полюсы p_k не являются кратными. Возможен также переход от передаточной функции *H(p)* к частотной характеристике $H(j\omega)$. Для этого необходимо учесть, что $p = \sigma + j\omega$ и положить $\sigma = 0$.

Устойчивой называется система, которая способна возвращаться в исходное состояние после всякого выхода из него в результате какого-либо воздействия. Системы, используемые на практике, должны быть устойчивыми. Устойчивой является линейная система, у которой все полюсы передаточной функции имеют отрицательные действительные части. Нули передаточной функции могут иметь как отрицательные, так и положительные действительные части. Если все полюсы и нули передаточной функции имеют отрицательные или равные нулю действительные части, то система называется *минимально-фазовой*. Минимально-фазовые системы дают минимальный фазовый сдвиг φ при любой частоте ω по сравнению с системами, имеющими такую же АЧХ и у которых указанное выше условие в отношении полюсов и нулей передаточной функции не выполняется (такие системы называются *неминимально-фазовыми*). Между АЧХ и ФЧХ минимально-фазовой системы имеется взаимно однозначное соответствие. Для неминимально-фазовых систем такого соответствия не существует.

1.3.3 **Аналоговые фильтры**

Теория линейных стационарных непрерывных систем широко используется при построении различных частотно-избирательных цепей. А*налоговым фильтром* называется частотно-избирательная цепь, обеспечивающая пропускание сигналов в определенных полосах частот и по-

давление в других. Область частот, в которой фильтр пропускает сигнал, называют *полосой пропускания*, а область частот, в которой ослабление входного сигнала велико - *полосой задерживания*.

Фильтры применяют для выделения требуемого сигнала из смеси полезных и нежелательных сигналов. В зависимости от взаимного расположения полосы пропускания и полосы задерживания различают следующие виды фильтров: нижних частот (ФНЧ), верхних частот (ФВЧ), полосовые (ПФ), режекторные (заграждающие). АЧХ указанных фильтров изображены на рис.1.14 в порядке упоминания. Кроме этих четырех основных типов фильтров при обработке сигналов находят применение: амплитудные корректоры (АК), способные в некоторой полосе частот осуществлять как усиление, так и ослабление сигналов; фазовые корректоры (ФК), у которых АЧХ не зависит от частоты, а ФЧХ может меняться по заданному закону.

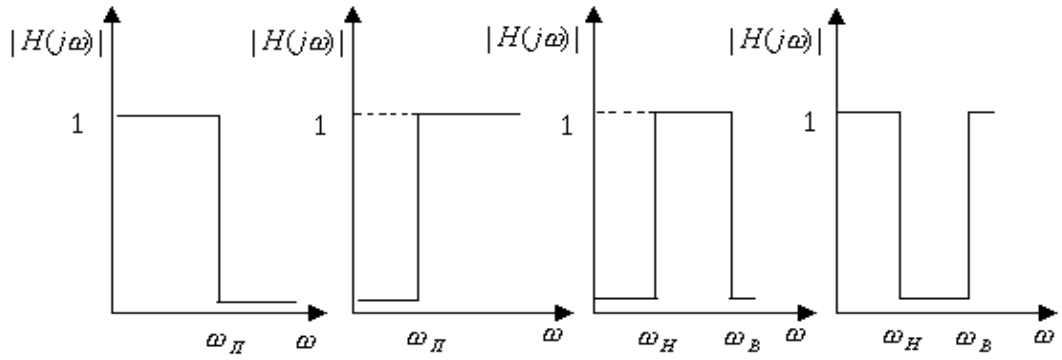


Рис.1.14. АЧХ типовых фильтров

Потребности практики предъявляют различные требования к форме АЧХ и параметрам фильтров. При проектировании фильтров задают границы полос пропускания ω_1 (рис.1.15) и задерживания ω_2, затухание в полосе задерживания H_2 и коэффициент передачи в полосе пропускания H_1, допуск на отклонение АЧХ от желаемого вида в полосе пропускания Δ_1. Отклонение реальной АЧХ фильтра от желаемой в полосе пропускания называют неравно- мерностью АЧХ. Закон изменения АЧХ в переходной области $\omega_1 \ldots \omega_2$ не задается, обычно коэффициент передачи H_1 равен 1.

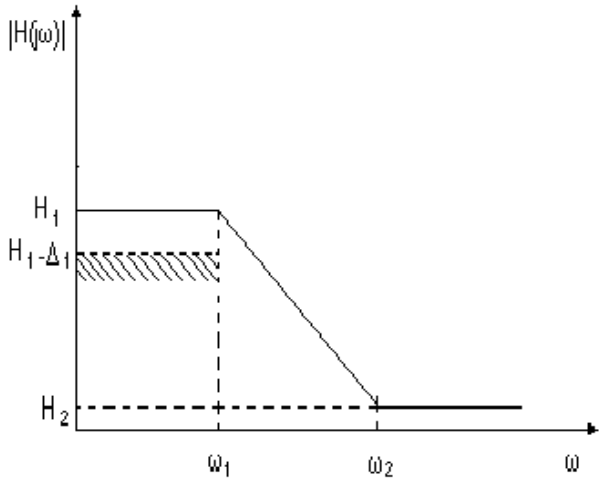


Рис.1.15. Требования к АЧХ

В некоторых случаях задание на проектирование может содержать дополнительные требования к фазо-частотной характеристике фильтра в полосе пропускания.

Требуемая частотная характеристика фильтра в виде кусочно-линейной аппроксимации, показанная на рис.1.15, нереализуема. Поэтому при синтезе фильтров на первом этапе выполняют аппроксимацию желаемой АЧХ, подбирая передаточную функцию фильтра таким образом, чтобы она была реализуемой и удовлетворяла требованиям, заданным при проектировании [6,13,19]. На практике широко применяют аппроксимацию АЧХ полиномами Баттерворта, Чебышева и Бесселя. С целью унификации общепринято применять нормирование по частоте, приводящее расчет различных типов фильтров (ФВЧ, ФНЧ, ПФ, РФ) к нормированному фильтру нижних частот с граничной частотой пропускания (*частотой среза*) $\omega_1 = 1$. В дальнейшем переход от низкочастотного фильтра-прототипа к требуемому фильтру выполняют с помощью специальных преобразований частоты.

Обычно при проектировании фильтров *неравномерность* в полосе пропускания R и *затухание* в полосе задерживания A задают в логарифмическом масштабе (децибелах):

$$R = 20 \cdot \log\left(\frac{1}{1-\Delta_1}\right) \qquad (1.106)$$

и

$$A = 20 \cdot \log\left(\frac{1}{H_2}\right). \qquad (1.107)$$

Рассмотрим различные способы аппроксимации аналоговых фильтров [6,13,19].

ФНЧ Баттерворта имеет АЧХ, которая определяется соотношением (рис.1.16)

$$\left|H(j\omega)\right|^2 = \frac{1}{1+(\omega/\omega_1)^{2n}} \quad , \qquad (1.108)$$

где ω_1 - частота среза фильтра; n - порядок фильтра.

На частоте $\omega = \omega_1$ коэффициент передачи фильтра Баттерворта равен $1/\sqrt{2}$. В логарифмическом масштабе это соответствует ослаблению в 3 дБ. Производные от функции (1.108) по частоте в точке $\omega = 0$ равны нулю. Поэтому фильтр Баттерворта называют фильтром с максимально плоской АЧХ.

Порядок n фильтра Баттерворта при заданных значениях R и A можно определить из соотношения [13]:

$$n \geq \frac{\log \sqrt{(10^{0,1A}-1)/(10^{0,1R}-1)}}{\log(\omega_2/\omega_1)}. \qquad (1.109)$$

Передаточная функция нормированного фильтра нижних частот n -го порядка описывается с помощью соотношения

$$H(p) = 1/N(p) = 1\Big/\prod_{k=1}^{n}(p-p_k), \qquad (1.110)$$

где *N(p)* - полином Баттерворта n -го порядка, p_k - полюсы передаточной функции $H(p)$. Полюсы p_k для фильтров Баттерворта определяются из соотношения [5]:

$$p_k = \cos\pi\frac{n-1+2\cdot k}{2n} + j\sin\pi\frac{n-1+2\cdot k}{2n}. \qquad (1.111)$$

В табл.1.1 приведены некоторые полиномы *N(p)* нормированных фильтров Баттерворта.

Таблица 1.1 - Полиномы Баттерворта

Порядок	*N(p)*
1	$1+p$
2	$1+1.414p+p^2$
3	$(1+p)(1+p+p^2)$
4	$(1+0.765p+p^2)(1+1.848p+p^2)$

Фильтр Чебышева нижних частот имеет АЧХ, которая определяется соотношением (рис.1.16)

$$\left|H(j\omega)\right|^2 = \frac{1}{1+\varepsilon^2 T_n^2(\omega/\omega_1)}, \qquad (1.112)$$

где $T_n(x)$ - полином Чебышева n - го порядка; ε - постоянный коэффициент, определяющий амплитуду пульсаций АЧХ в полосе пропускания. Полиномы Чебышева обеспечивают равноволновые пульсации АЧХ в полосе пропускания и более резкий спад ее за граничной частотой ω_1, чем полиномы Баттерворта.

Передаточная функция ФНЧ Чебышева также описывается соотношением (1.110), но полюсы p_k рассчитываются по другим формулам:

$$p_k = j\omega_1\{\cos(\alpha_k)\cosh(\beta_k) - j\sin(\alpha_k)\sinh(\beta_k)\}, \qquad (1.113)$$

где

$$\alpha_k = \frac{(2k+1)\pi}{2n}; \qquad \beta_k = \frac{1}{n}\cdot arsh\,(1/\varepsilon).$$

Коэффициент пульсаций ε при заданном значении R определяется по формуле

$$\varepsilon = \sqrt{10^{0,1R} - 1}\,. \qquad (1.114)$$

Порядок фильтра определяют из соотношения [13]

$$n = \frac{Arch\ (\sqrt{(10^{0.1A} - 1)/(10^{0.1R} - 1)}}{Arch\ (\omega_2/\omega_1)}\,. \qquad (1.115)$$

Инверсный фильтр Чебышева имеет АЧХ, которая определяется соотношением (рис.1.16)

$$\left|H(j\omega)\right|^2 = \frac{\varepsilon\cdot T_n^{\,2}(\omega_2/\omega)}{1+\varepsilon^2 T_n^{\,2}(\omega_2/\omega)}\,. \qquad (1.116)$$

АЧХ такого фильтра монотонно изменяется в полосе пропускания и пульсирует в полосе задерживания.

Полюсы инверсного ФНЧ Чебышева определяются по тем же формулам, что и для “прямого” ФНЧ Чебышева, но с заменой ω_1 на ω_2.

Коэффициент пульсаций равен

$$\varepsilon = \frac{1}{\sqrt{10^{0,1A} - 1}}\,. \qquad (1.117)$$

Порядок фильтра вычисляется по формуле (1.115).

Эллиптические фильтры (фильтры Кауэра) имеют АЧХ, которая имеет вид

$$\left|H(j\omega)\right|^2 = \frac{1}{1+\varepsilon^2\Psi_n^{\,2}(\omega/\omega_1)}\,. \qquad (1.118)$$

АЧХ эллиптического фильтра пульсирует как в полосе пропускания, так и в полосе задерживания (рис 1.16).

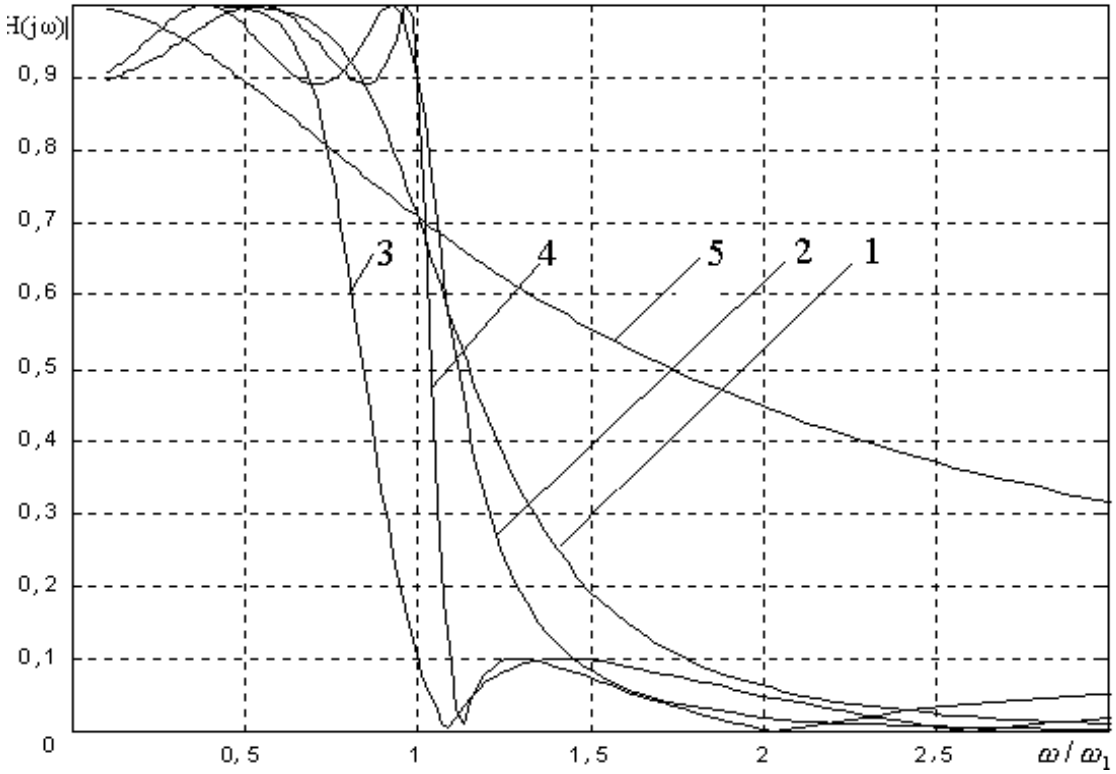


Рис.1.16. АЧХ ФНЧ 4-го порядка (1- Баттерворта; 2- Чебышева; 3- инверсный Чебышева; 4- Кауэра; 5- Бесселя)

Функция $\Psi_n(\omega/\omega_1)$ называется рациональной функцией Чебышева и для четных n имеет вид

$$\Psi_n(x) = \frac{(x^2 - x_1^2)(x^2 - x_2^2)...(x^2 - x_k^2)}{(x^2 x_1^2 - 1)(x^2 x_2^2 - 1)...(x^2 x_k^2 - 1)}, \qquad (1.119)$$

где $k = n/2$; x_1, ..., x_k - параметры, значения которых лежат в диапазоне от 0 до 1. Параметры x_1, ..., x_k выбираются таким образом, чтобы обеспечить равноволновые пульсации функций $\Psi_n(x)$ между нулем и некоторым значением Δ. При этом в полосе пропускания квадрат АЧХ эллиптического фильтра пульсирует между значением 1 и $1/(1+\varepsilon^2\Delta^2)$, а в полосе задерживания – между значением 0 и $1/(1+\varepsilon^2/\Delta^2)$. Соотношение между порядком фильтра n и величинами A, R, ε является более сложным, чем у фильтров Чебышева и поэтому здесь не приводится. В [19] приведены необходимые таблицы для расчетов фильтров Кауэра.

Фильтр Бесселя имеет ФЧХ, приближенную к линейной зависимости. Это позволяет сигналам, проходящим через такой фильтр, не менять своей формы. Передаточная функция фильтров Бесселя определяется формулой

$$H(p) = \frac{k}{D_n(p)}, \qquad (1.120)$$

где $D_n(p)$ - полином Бесселя n - го порядка, который может быть найден с использованием рекурсивных соотношений

$$D_n(p) = (2n-1)D_{n-1}(p) + p^2 D_{n-2}(p), \qquad (1.121)$$
$$D_1(p) = p+1; \quad D_2(p) = p^2 + 3p + 3.$$

Важным свойством фильтров Бесселя, помимо избирательности их АЧХ и линейности ФЧХ, является уменьшение колебательных выбросов на переходной характеристике по сравнению с фильтрами Баттерворта и Чебышева.

1.4 Основные характеристики случайных сигналов

Случайные сигналы описывают случайными функциями. Значение случайной функции при каждом данном значении аргумента - случайная величина. *Случайной величиной* называется величина, которая принимает случайные значения из некоторого множества возможных значений. Случайную функцию времени называют *случайным процессом*. Условно случайный процесс можно представить в виде совокупности реализаций (рис.1.17).

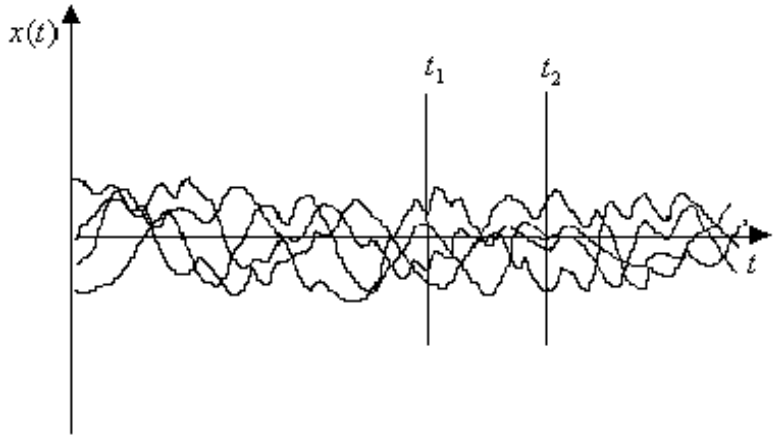


Рис.1.17. Реализации случайного процесса x(t)

Сечение случайного процесса *x(t)* в момент времени $t=t_1$ представляет случайную величину *x*, которая полностью характеризуется одномерной *функцией плотности вероятности p(x)*. Произведение *p(x)dx* - вероятность того, что значение случайной величины будет находиться в

интервале (x, $x+dx$). Используя одномерную плотность вероятности, можно определить важные характеристики случайных сигналов.

Математическое ожидание случайной величины определяется выражением [2]

$$m_x = M\{x\} = \int_{-\infty}^{\infty} xp(x)dx . \qquad (1.122)$$

Математическое ожидание, как и другие рассматриваемые ниже характеристики, не является случайной величиной. Оно соответствует среднему значению случайной величины x .

Математическое ожидание квадрата (средний квадрат) случайной величины x равно

$$M\{x^2\} = \int_{-\infty}^{\infty} x^2 p(x)dx . \qquad (1.123)$$

Дисперсией D_x случайной величины x называется средний квадрат отклонения этой величины от ее математического ожидания

$$D_x = M\{(x-m_x)^2\}. \qquad (1.124)$$

Среднеквадратическое отклонение случайной величины x равно $\sigma_x = \sqrt{D_x}$.

Если рассмотреть сечения случайного процесса $x(t)$ в моменты времени t_1 и t_2, то получим две случайные величины $x(t_1)=x_1$ и $x(t_2)=x_2$. Они полностью характеризуются *двумерной плотностью вероятности* $p(x_1, x_2)$. Произведение $p(x_1, x_2)dx_1dx_2$ - вероятность того, что значения случайной величины x_1 будут находиться в интервале (x_1, x_1+dx_1), а x_2 – в интервале (x_2, x_2+dx_2).

Характеристикой взаимосвязи двух случайных величин является *корреляционный момент*

$$K_{x_1x_2} = M\{(x_1-m_{x_1})(x_2-m_{x_2})\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x_1-m_{x_1})(x_2-m_{x_2})p(x_1,x_2)dx_1dx_2 . \qquad (1.125)$$

Если учесть, что случайные величины x_1 и x_2 определены для двух сечений одного и того же процесса $x(t)$, то можно ввести функцию

$$K_x(t_1,t_2) = M\{(x(t_1)-m_x(t_1))(x(t_2)-m_x(t_2))\}, \qquad (1.126)$$

которую называют *корреляционной функцией* случайного процесса. Часто рассматривают нормированную корреляционную функцию

$$R_x(t_1,t_2)=\frac{K_x(t_1,t_2)}{\sqrt{K_x(t_1,t_1)\cdot K_x(t_2,t_2)}}. \qquad (1.127)$$

Случайный процесс *x(t)* называют *стационарным в широком смысле*, если его статистические характеристики, рассмотренные выше, не зависят от начала отсчета времени, т.е.

$$\begin{aligned} m_x(t) &= m_x; \\ D_x(t) &= D_x; \\ K_x(t_1,t_2) &= K_x(\tau), \end{aligned} \qquad (1.128)$$

где $\tau=t_2-t_1$ - произвольный сдвиг времени.

Легко заметить, что для таких процессов корреляционная функция обладает следующими свойствами [2]:

$$\begin{aligned} &K_x(0)=D_x>0; \\ &K_x(-\tau)=K_x(\tau); \\ &K_x(0)\ge\left|K_x(\tau)\right|; \\ &\left|R_x(\tau)\right|\le 1. \end{aligned} \qquad (1.129)$$

Если $K_x(\tau)=0$ при $\tau\neq 0$, то случайный процесс является некоррелированным.

Связь двух случайных процессов *x(t)* и *y(t)* характеризуется *взаимной корреляционной функцией*

$$K_{xy}(t_1,t_2)=M\{(x(t_1)-m_x(t_1))(y(t_2)-m_y(t_2))\}. \qquad (1.130)$$

Так как функция $K_x(t_1,t_2)$ характеризует взаимосвязь сечений одного и того же случайного процесса, то ее обычно называют *автокорреляционной функцией.*

Рассмотрим стационарный в широком смысле процесс *x(t)*. Введем случайную величину

$$Y_T=\frac{1}{T}\int_0^T x(t)\,dt. \qquad (1.131)$$

Если

$$M\{Y_T\}=m_x, \qquad D\{Y_T\}\to 0 \quad при \quad T\to\infty, \qquad (1.132)$$

то можно записать приближенное равенство

$$m_x\approx\frac{1}{T}\int_0^T x(t)dt. \qquad (1.133)$$

Случайный процесс *x(t)*, для которого выполняются условия (1.132), называется *эргодическим* по отношению к математическому ожиданию. Для эргодического процесса автокорреляционная функция и дисперсия могут быть вычислены по формулам:

$$K_x(\tau) = \frac{1}{T}\int_0^T (x(t) - m_x)(x(t-\tau) - m_x)dt, \qquad (1.134)$$

$$K_x(0) = D_x = \frac{1}{T}\int_0^T (x(t) - m_x)^2 dt. \qquad (1.135)$$

Величину

$$P_x = \lim_{T\to\infty} \frac{1}{T}\int_0^T x^2(t)\,dt \qquad (1.136)$$

называют *мощностью* сигнала *x(t)*.

Определим спектральные характеристики стационарного процесса *x(t)* с математическим ожиданием m_x=0. Найдем спектральную плотность корреляционной функции

$$S_x(\omega) = \int_{-\infty}^{\infty} K_x(\tau)\cdot e^{-j\omega\tau}\,d\tau. \qquad (1.137)$$

Функция $S_x(\omega)$ называется *спектральной плотностью мощности* (СПМ).

Соответственно справедливо и обратное преобразование

$$K_x(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_x(\omega)e^{j\omega\tau}d\omega. \qquad (1.138)$$

Пара преобразований (1.137) и (1.138) называется *преобразованиями Винера – Хинчина*.

Выясним физический смысл СПМ. Из (1.138) следует, что при $\tau = 0$

$$D_x = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_x(\omega)d\omega, \qquad (1.139)$$

т.е. дисперсия случайного процесса *x(t)* пропорциональна площади под кривой $S_x(\omega)$. С другой стороны, при m_x=0 дисперсия равна средней мощности процесса *x(t)*

$$D_x = \lim_{T\to\infty} \frac{1}{T}\int_0^T x^2(t)\,dt. \qquad (1.140)$$

Поэтому величина $S_x(\omega)$ представляет удельную мощность, приходящуюся на спектральную составляющую сигнала *x(t)* в окрестности выбранной частоты ω. Можно показать, что

$$S_x(\omega) = \lim_{T\to\infty} M\left\{\frac{|X_T(\omega)|^2}{T}\right\}, \qquad (1.141)$$

где $X_T(\omega)$ - представляет непосредственное преобразование Фурье реализации сигнала *x(t)* на интервале [0, *T*]. Формула (1.141) позволяет вычислять СПМ через спектральную плотность реализации $X_T(\omega)$.

Поскольку $K_x(\tau)$ - четная функция, то соответствующая спектральная плотность $S_x(\omega)$ будет также четной функцией частоты. Тогда пара преобразований (1.137) и (1.138) может быть записана в форме

$$K_x(\tau) = \frac{1}{\pi}\int_0^\infty S_x(\omega)\cos\omega\tau d\omega\ , \qquad (1.142)$$

$$S_x(\omega) = 2\int_0^\infty K_x(\tau)\cos\omega\tau d\tau\ . \qquad (1.143)$$

Здесь функция $S_x(\omega)$ определена только для положительных частот.

По аналогии с (1.137) и (1.38) можно ввести *взаимную спектральную плотность мощности*

$$S_{xy}(\tau) = \int_{-\infty}^{\infty} K_{xy}(\tau)e^{-j\omega\tau}d\tau\ , \qquad (1.144)$$

где

$$K_{xy}(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{xy}(\omega)\cdot e^{j\omega\tau}d\tau\ . \qquad (1.145)$$

Преобразования Винера – Хинчина играют огромную роль в прикладной теории случайных сигналов, позволяя решать разнообразные задачи.

1.5 Дискретные сигналы и системы

1.5.1 Преобразование спектра при дискретизации сигналов

Рассмотрим равномерную дискретизацию сигнала *x(t)*, выполняемую с помощью импульсного элемента (ИЭ) (рис. 1.18). Как видно из рис. 1.18,а импульсный элемент представляет собой ключ, который периодически замыкается через интервалы времени T_0. Время замкнутого состояния контактов ключа бесконечно мало. В результате на выходе

импульсного элемента формируются отсчеты входного сигнала (рис. 1.18,б).

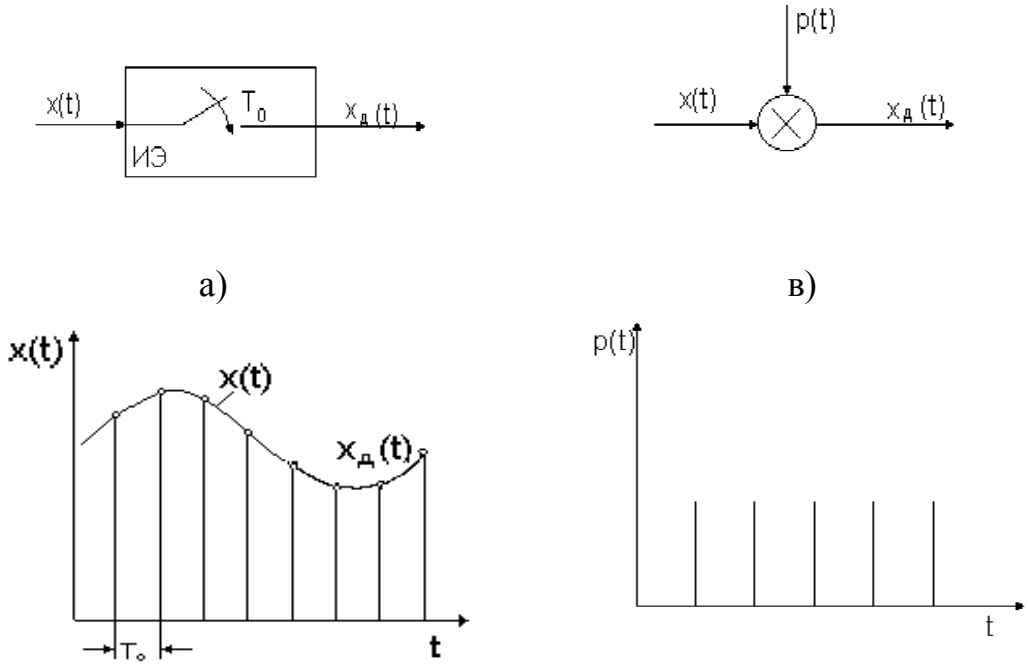


Рис 1.18. Дискретизация непрерывного сигнала

Дискретный сигнал на выходе ИЭ можно представить в виде произведения исходного сигнала *x(t)* и дискретизирующей последовательности импульсов *p(t)* (рис.1.18,в):

$$x_д(t) = x(t)\,p(t) \quad . \tag{1.146}$$

Дискретизирующая последовательность (рис.1.18,г)

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_0) \tag{1.147}$$

состоит из дельта-импульсов, которые следуют с периодом T_0. Разложим $p(t)$ в ряд Фурье по системе комплексных экспоненциальных функций

$$p(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\,2\pi t / T_o} \quad . \tag{1.148}$$

Коэффициенты c_k определятся из соотношения:

$$c_k = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2}\delta(t)e^{-jk2\pi t/T_0}dt = \frac{1}{T_0}. \qquad (1.149)$$

Тогда импульсный сигнал $x_д(t)$, получаемый в результате дискретизации, можно описать следующим образом:

$$x_Д(t) = \sum_{k=-\infty}^{\infty}\frac{1}{T_0}x(t)e^{jk2\pi t/T_0} \quad . \qquad (1.150)$$

Воспользовавшись интегральным преобразованием Фурье, находим выражение для спектральной плотности дискретного сигнала $x_д(t)$

$$X_Д(\omega) = \frac{1}{T_0}\sum_{k=-\infty}^{\infty}X(\omega - k\frac{2\pi}{T_0}) \quad , \qquad (1.151)$$

где $X(\omega)$ - спектральная плотность исходного недискретизированного сигнала $x(t)$.

Таким образом, спектральная плотность $X_д(\omega)$ дискретного сигнала $x_д(t)$, соответствующего непрерывному сигналу $x(t)$, равна сумме спектральных плотностей $X(\omega)$, сдвинутых на величину $k2\pi/T_0$, т. е. спектр дискретного сигнала является периодическим по частоте с периодом $\omega_0=2\pi/T_0$ (рис. 1.19).

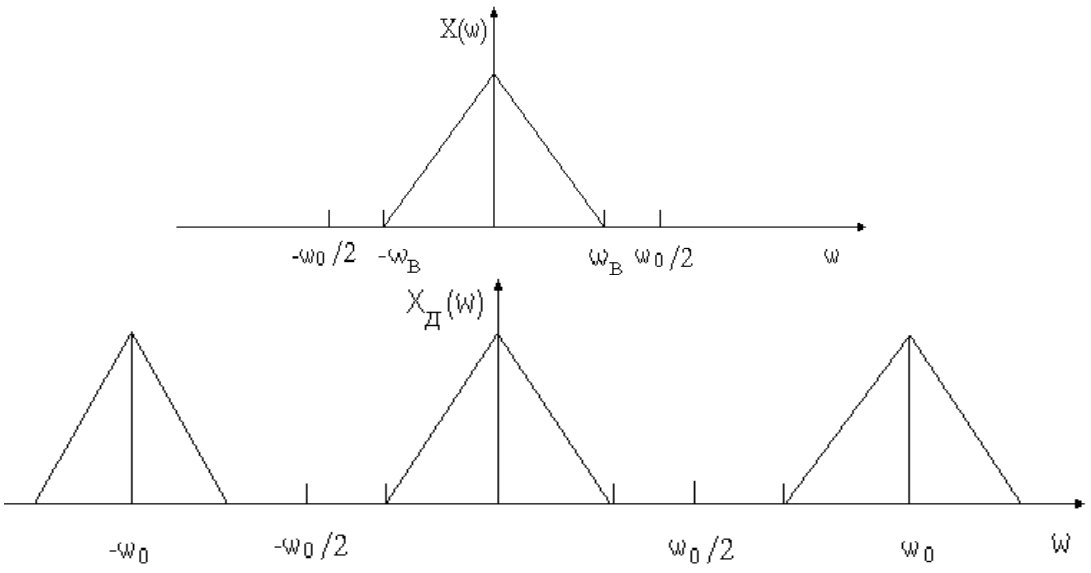


Рис.1.19. Соотношение спектров непрерывного и дискретного сигналов

Для изображенного на рис. 1.19. случая, когда граничная частота спектра исходного непрерывного сигнала удовлетворяет условию $\omega_B \le \omega_0/2$, не происходит наложения соседних членов ряда (1.151). В том случае, когда соотношение $\omega_B \le \omega_0/2$ не выполняется, возникает наложение спектральных составляющих, что приводит к погрешностям при обработке сигналов (рис.1.20).

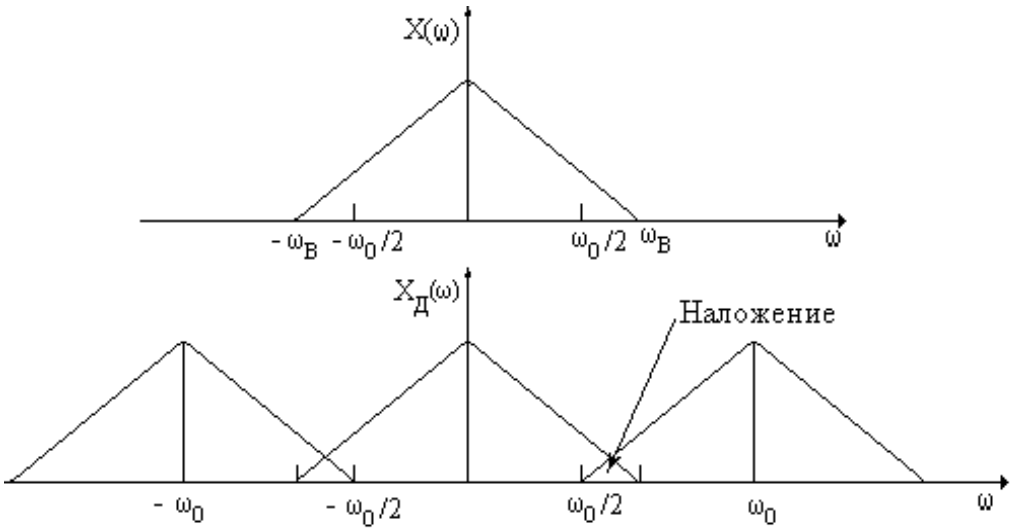


Рис.1.20. Наложение спектральных составляющих

Для исключения наложения спектральных составляющих обычно перед дискретизацией спектр исходного непрерывного сигнала ограничивают по частоте, пропуская его через аналоговый ФНЧ, который задерживает все составляющие, частоты которых превышают значение $\omega_0/2$.

1.5.2 Теорема Котельникова

Теорема Котельникова определяет условия, при которых непрерывный сигнал может быть восстановлен по своим отсчетам без потери информации.

Теорема Котельникова формулируется следующим образом:

1) сигнал *x(t),* спектральная плотность которого не содержит составляющих с частотами выше ω_B, полностью определяется своими мгновенными значениями, отсчитанными в дискретные моменты времени через интервал

$$T_0 = \frac{1}{2F_B} = \pi / \omega_B , \quad \text{где } F_B = \omega_B / 2\pi ;$$

2) сигнал *x(t)* может быть точно восстановлен по своим отсчетным значениям с помощью выражения

$$x(t)=\sum_{n=-\infty}^{\infty}x(nT_0)\frac{\sin\omega_B(t-nT_0)}{\omega_B(t-nT_0)}\,, \qquad (1.152)$$

где $x(nT_0)$ - отсчеты непрерывного сигнала $x(t)$ в точках $t= nT_0$.

Выражение (1.152) можно интерпретировать как ряд Фурье. В этом случае (1.152) соответствует разложению непрерывного сигнала $x(t)$ по системе базисных функций

$$\varphi(t-nT_0)=\frac{\sin\omega_B(t-nT_0)}{\omega_B(t-nT_0)}\,, \qquad (1.153)$$

ортогональных на интервале $(-\infty,\infty)$ [1]. Коэффициенты в разложении (1.152) равны значениям восстанавливаемого сигнала в точках дискретизации.

Каждая из базисных функций $\varphi_n(t)$ сдвинута относительно функции с номерами n+1 или n-1 на время T_0, соответствующее временному интервалу дискретизации, который называют *интервалом Найквиста*. В точках nT_0 сигнал $x(t)$ определяется лишь одним слагаемым ряда (1.152), все остальные члены ряда в этих точках имеют нулевые значения. Следовательно, в моменты времени $t= nT_0$ непрерывный сигнал восстанавливается точно. Между отсчетами $(t\neq nT_0)$ сигнал $x(t)$ восстанавливается точно только в том случае, если суммируются все члены ряда (1.152) и соблюдается условие $T_0=\pi/\omega_B$.

Представление (1.152) справедливо для бесконечных во времени сигналов с конечным (финитным) спектром, ограниченным частотой ω_B. В процессе цифровой обработки реальные сигналы всегда ограничивают во времени. Если сигнал ограничен интервалом наблюдения T, то он лишь приближенно описывается рядом (1.152), состоящим из конечного числа членов $N = T / T_0 = 2F_BT$.

Перепишем (1.152) в виде

$$x(t)=\sum_{n=-\infty}^{\infty}x(nT_0)\varphi(t-nT_0)\,. \qquad (1.154)$$

Выражение (1.154) можно рассматривать как сумму реакций некоторого фильтра на импульсный сигнал $x(nT_0)\cdot\delta(t - nT_0)$. В этом случае $\varphi_n(t)$ соответствует импульсной характеристике фильтра.

Отсюда следует, что для восстановления непрерывного сигнала $x(t)$ по дискретным значениям $x(nT_0)$ необходимо воздействовать импульсами $x(nT_0)\cdot\delta(t - nT_0)$ на устройство с импульсной характеристикой вида (1.153). Подобную импульсную характеристику имеет идеальный фильтр нижних частот с частотой среза ω_B. Следовательно, восстановление не-

прерывного сигнала по его отсчетам в соответствии с теоремой Котельникова должно выполняться идеальным ФНЧ. Однако, как было показано в примере 1.10, такой фильтр нереализуем. Поэтому на практике восстановление непрерывного сигнала по его дискретным отсчетам всегда выполняется с некоторой погрешностью.

1.5.3 Дискретно-непрерывное и дискретное преобразования Фурье

Рассмотренное ранее разложение непрерывного сигнала *x(t)* в комплексный ряд Фурье может быть распространено на дискретный сигнал, описываемый решетчатой функцией *x*[*n*], который, в частности, может быть получен в результате дискретизации непрерывного сигнала *x(t)*. В этом случае значения дискретной последовательности *x*[*n*] будут совпадать с отсчетами $x(nT_0) = x(t)\big|_{t=nT_0}$.

Спектром последовательности *x*[*n*] называют комплексную функцию

$$X(e^{j\omega T_0}) = \sum_{n=0}^{\infty} x(nT_0)e^{-jn\omega T_0} = \sum_{n=0}^{\infty} x[n]e^{-jn\omega T_0} \quad . \quad (1.155)$$

Выражение (1.155) адекватно разложению спектральной функции $X(e^{j\omega T_0})$ в комплексный ряд Фурье (1.10). При этом коэффициенты ряда соответствуют последовательности *x*[*n*] и могут быть вычислены с помощью соотношения

$$x[n] = x(nT_0) = \frac{T_0}{2\pi}\int_{\frac{-\pi}{T_0}}^{\frac{\pi}{T_0}} X(e^{-j\omega T_0})e^{jn\omega T_0}d\omega \quad . \quad (1.156)$$

Формулы (1.155) и (1.156) представляют пару преобразований Фурье. Будем называть их *дискретно-непрерывным преобразованием Фурье* (ДНПФ) [11]. Подчеркивая, что $x[n]$ - дискретная (решетчатая) функция, а $X(e^{j\omega T_0})$ - непрерывная функция частоты ω, $-\pi/T_0 \le \omega \le \pi/T_0$. В литературе часто ДНПФ записывают в иной форме

$$X(e^{j\omega'}) = \sum_{n=0}^{\infty} x[n]e^{-jn\omega'} \quad , \quad (1.157)$$

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{-j\omega'})e^{jn\omega'}d\omega' \quad . \quad (1.158)$$

Преобразования (1.157) и (1.158) могут быть получены из (1.155) и (1.156) заменой $\omega' = \omega \cdot T_0$. С целью упрощения записи в преобразованиях (1.157) и (1.158) ω' записывают как ω. При этом полагают, что частота ω изменяется от $-\pi$ до π. Обычно это не вызывает недоразумений, поскольку диапазон изменения ω легко усматривается из контекста. Если в формулах присутствует произведение ωT_0, то ω принадлежит интервалу $[-\pi / T_0, \pi / T_0]$. Если же в формулах преобразований не фиксируется период дискретизации , то $\omega \in [-\pi, \pi]$.

Преобразования (1.157) и (1.158) обычно применяются в тех случаях, когда не акцентируется внимание на периоде дискретизации . В дальнейшем будем использовать обе формы ДНПФ .

Пара преобразований

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp(-j2\pi kn / N) \quad , \tag{1.159}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \exp(j2\pi kn / N) \quad , \tag{1.160}$$

где $x[n]$ – N-точечная последовательность данных; $X[k]$ – N-точечная последовательность спектральных коэффициентов, называется *дискретным преобразованием Фурье* (ДПФ) [4,16]. Преобразование (1.159) называют *прямым ДПФ*, а преобразование (1.160) – *обратным ДПФ* (ОДПФ).

Введем обозначение $W_N = e^{-j2\pi / N}$. Тогда

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad , \tag{1.161}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad . \tag{1.162}$$

ДПФ выполняется над конечной периодической последовательностью данных $x[n]$, у которой период состоит из N дискретных значений. Последовательность $X[k]$ представляет собой спектр дискретного сигнала $x[n]$ и в соответствии с (1.151) тоже является периодической, т.е. $X[k]= X[k+N]$.

Основные свойства ДПФ соответствуют рассмотренным ранее свойствам интегрального преобразования Фурье и приведены в табл. 1.2 [10].

Из табл.1.2 видно, что если $x[n]$ – действительная последовательность, то спектральные отсчеты $X[k]$, номера которых располагаются симметрично относительно k=0, образуют комплексно сопряженные па-

ры. Следовательно, для анализа амплитудного спектра можно использовать только первую половину отсчетов $X[k]$.

Таблица 1.2 - Свойства ДПФ

Последовательность конечной длины	ДПФ
$a_1x_1[n]+ a_2x_2[n]$	$a_1X_1[k]+ a_2X_2[k]$
$x[n+m]$	$X[k]\cdot e^{j2\pi km/N}=X[k]\cdot W_N^{-km}$
$x^*[n]$ $x^*[-n]$	$X^*[-k]$ $X^*[k]$
$x[n]$ действительная последовательность	$X[k]=X^*[-k]$ Re $X[k]$=Re $X[-k]$ Im $X[k]$= - Im $X[-k]$ $\|X[k]\|=\|X[-k]\|$

1.5.4 Быстрое преобразование Фурье

В соответствии с формулой прямого ДПФ для вычисления N значений спектральных отсчетов $X[k]$ требуется выполнить примерно N^2 умножений и N^2 сложений. С ростом N значительно возрастает время вычисления $X[k]$. Такая ситуация наблюдается и при вычислении ОДПФ. Для ускорения вычислений ДПФ разработаны алгоритмы быстрого преобразования Фурье (БПФ).

Рассмотрим алгоритмы БПФ с основанием 2, которые применяются к последовательностям длины $N=2^k$, k – целое.

Основная идея БПФ состоит в следующем. Последовательность $x[n]$ разбивают на две $N/2$ – точечные последовательности $x_1[n]$ и $x_2[n]$, и находят для них $X_1[k]$ и $X_2[k]$. Затем по значениям $X_1[k]$ и $X_2[k]$ определяют требуемое N – точечное ДПФ $X[k]$. Если последняя операция будет выполняться просто и не потребует сложных вычислений, то для вычисления N – точечного ДПФ потребуется выполнить $(N/2)^2+(N/2)^2=N^2/2$ операций. Если продолжить процесс разбиения последовательностей $x_1[n]$ и $x_2[n]$ на две части и находить для каждой из них свои ДПФ, то можно существенно сократить количество операций.

Рассмотрим ДПФ последовательности $x[n]$:

$$X\ [\ k\ ]\ =\ \sum_{n\ =\ 0}^{N\ -\ 1} x\ [\ n\ ]W_{\ n}^{\ kn} \quad . \qquad (1.163)$$

Разобьем $x[n]$ на две части $x_1[n]$ и $x_2[n]$, содержащие соответственно четные и нечетные члены $x[n]$:

$$x_1[n]=x[2n]\ , \quad n=0,1,2,\ldots..,(N/2-1);$$
$$x_2[n]=x[2n+1],\ n=0,1,2,\ldots..,(N/2-1).$$

Тогда

$$X[k]=\sum_{n=0}^{N-1}x[n]\ \ W_N^{kn}=\sum_{n=0}^{N/2-1}x[2n]\ \ W_N^{2nk}\ +\sum_{n=0}^{N/2-1}x[2n+1]\ \ W_N^{(2n+1)k}\ .$$

Так как $W_N^2=e^{(j2\pi/N)\cdot 2}=e^{j2\pi/(N/2)}=W_{N/2}$, то

$$X[k]=\left\{\sum_{n=0}^{N/2-1}x[2n]\ \ W_{N/2}^{nk}\right\}+\left\{\sum_{n=0}^{N/2-1}x[2n+1]\ \ W_{N/2}^{nk}\right\}\cdot W_N^k\ .$$

Здесь выражения в фигурных скобках представляют прямые $N/2$ – точечные ДПФ от последовательностей $x_1[n]$ и $x_2[n]$. Тогда

$$X[k]=X_1[k]+X_2[k]\cdot W_N^k\quad ,\ 0\le k\le N/2-1\ . \qquad (1.164)$$

Из формулы (1.164) следует, что N – точечное ДПФ $X[k]$ может быть вычислено через два $N/2$ – точечных ДПФ $X_1[k]$ и $X_2[k]$. Следует иметь в виду, что отсчеты ДПФ для последовательностей $x_1[n]$ и $x_2[n]$ повторяются с периодом $N/2$, Поэтому

$$X_1[k+N/2]=X_1[k] \qquad и \quad X_2[k+N/2]=X_2[k]\ . \qquad (1.165)$$

Учитывая равенство $W_N^{k+N/2}=-W_N^k$, из (1.164) получаем выражение для определения второй части последовательности спектральных коэффициентов X[k]:

$$X[k+N/2]=X_1[k]-X_2[k]\cdot W_N^k\ . \qquad (1.166)$$

Формулы (1.164), (1.166) представляют базовую операцию БПФ (так называемую «бабочку»). Схематическое изображение “бабочки” показано на рис.1.21. Здесь кружок в центре обозначает операцию сложения/вычитания. Стрелка обозначает операцию умножения на W_N^k. Жирные точки обозначают регистры, содержащие входные и выходные значения для отдельных этапов БПФ.

На рис.1.22 показана схема вычисления 8 – точечного БПФ с использованием двух 4 – точечных преобразований.

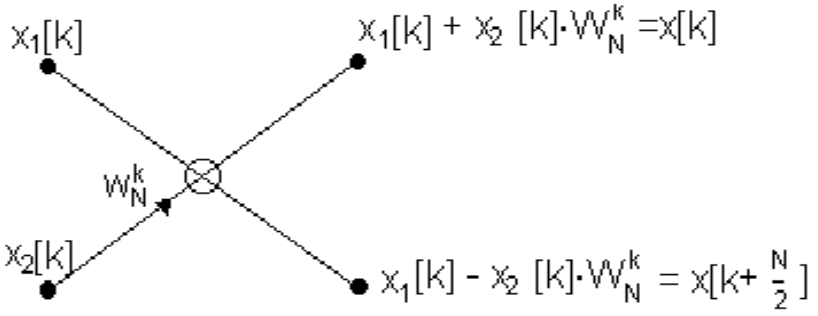


Рис. 1.21. Основная операция БПФ –“бабочка”

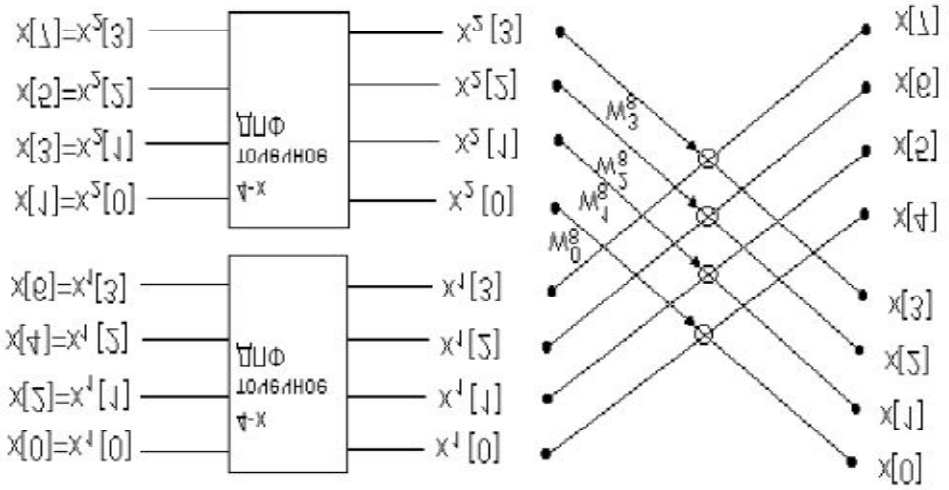


Рис.1.22. Вычисление 8-точечного БПФ

В свою очередь, 4 – точечные преобразования могут быть определены через 2 – точечные, которые вычисляются по формулам:

$$X[0] = x[0] + W_{N/4}^{0} \cdot x[1] = x[0] + x[1] \qquad (1.167)$$

$$X[1] = x[0] + W_{N/4}^{1} \cdot x[1] = x[0] - x[2] \ . \qquad (1.168)$$

На рис.1.23 показан результирующий граф 8 – точечного БПФ.

Описанный алгоритм БПФ называется алгоритмом *с прореживанием по времени*, так как здесь требуется перестановка входных значений $x[n]$ (см. рис.1.23). Алгоритмы, при реализации которых требуется перестановка отсчетов $X[k]$, называются алгоритмами *с прореживанием по частоте*.

Поскольку произведение $X_2[k] \cdot W_N^k$ можно после вычисления запомнить, то для каждой базовой операции БПФ необходимо выполнять только одно умножение $X_2[k]$ на множитель W_N^k. Входные и выходные значения базовых операций обычно хранят в одних и тех же ячейках памяти. На одну базовую операцию приходится одна дополнительная ячейка для хранения произведения $X_2[k] \cdot W_N^k$. Поэтому входная $x[n]$ и выходная

$X[k]$ последовательности, а также промежуточные результаты могут размещаться в одном и том же массиве ячеек памяти. Алгоритм БПФ, использующий указанные возможности, называют *алгоритмом с замещением*.

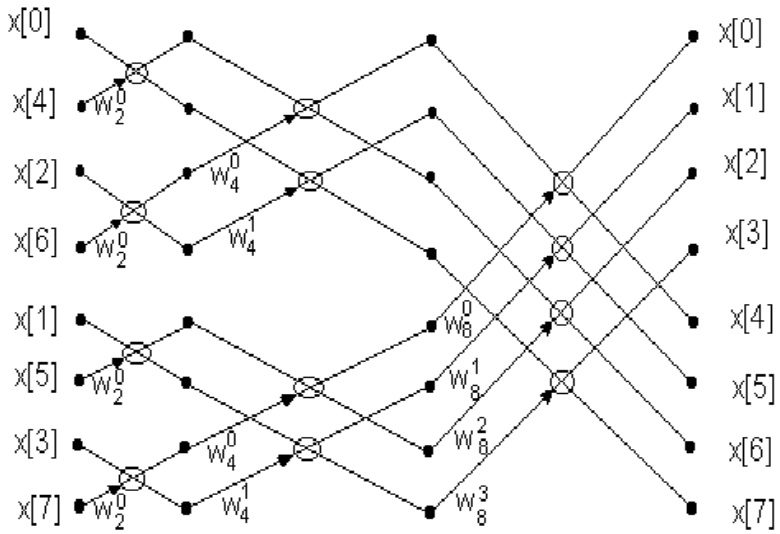


Рис.1.23. Результирующий граф 8-точечного БПФ

Особенностью рассматриваемых алгоритмов БПФ является необходимость перестановки входных или выходных значений. Элементы входной последовательности для алгоритма с прореживанием по времени должны быть расположены в памяти в бит–инверсном порядке. *Бит–инверсный порядок* задается путем «зеркального» отображения двоичных разрядов входной последовательности (табл.1.3).

Таблица 1.3 - Бит-инверсный порядок

N	Двоичный номер	Бит инверсия	Бит-инверсный номер
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

На всех этапах выполнения БПФ используются коэффициенты W_N^k, k=0,1,…N-1. Обычно указанные коэффициенты вычисляют до выполнения БПФ и хранят в таблице, к которой можно обращаться в процессе счета.

Количество этапов БПФ равно $\log_2 N$, количество “бабочек” на каждом этапе - $N/2$. Так как в процессе БПФ используются комплексные числа, то каждая «бабочка» БПФ сопровождается четырьмя операциями умножения и сложения. Поэтому количество парных операций умножение-сложение равно $2N \log_2 N$. При больших N применение алгоритма БПФ существенно сокращает количество операций, требуемых для вычисления ДПФ.

1.5.5 Дискретное косинусное преобразование

Ряд Фурье для любой непрерывной действительной и четной функции содержит только действительные коэффициенты, соответствующие косинусным членам ряда (см.§ 1.2). Распространение данного результата на ДПФ приводит к *дискретному косинусному преобразованию* (ДКП). ДКП широко применяется в системах сжатия сигналов, базирующихся на *методе ортогональных преобразований*. Суть этого метода поясняет рис. 1.24.

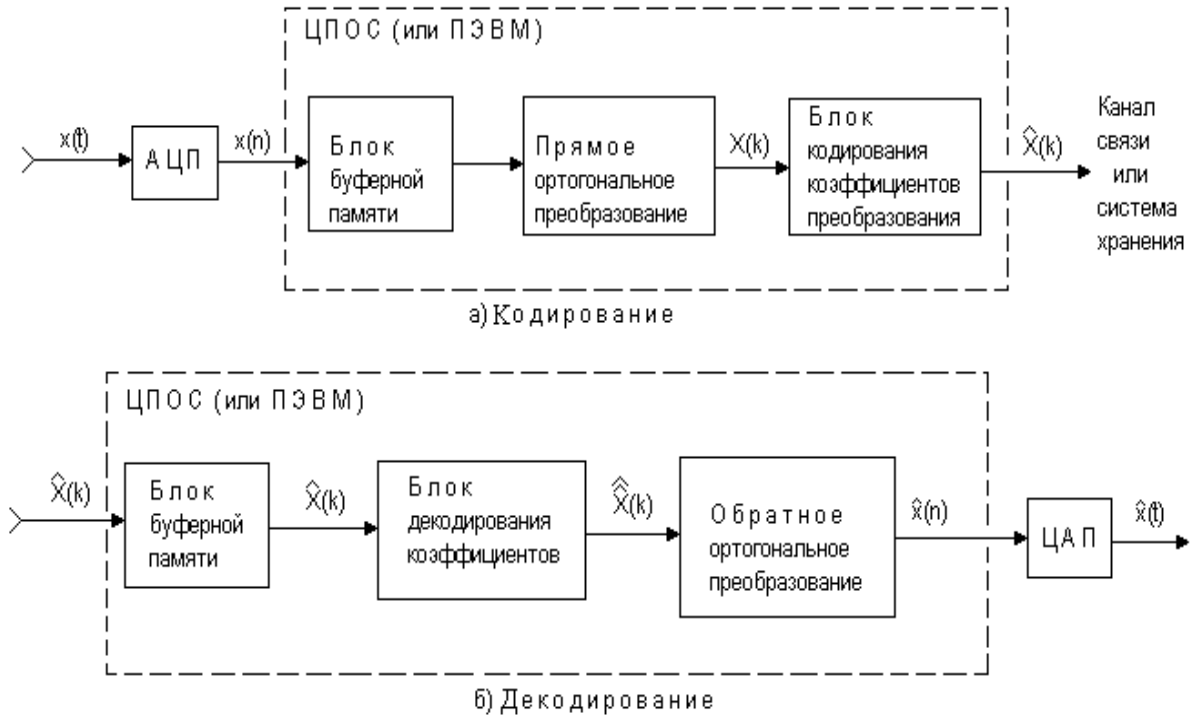


Рис 1.24. Сжатие сигналов на основе ортогональных преобразований

Здесь входной аналоговый сигнал $x(t)$ преобразуется с помощью АЦП в цифровую последовательность $x[n]$, которая запоминается в буферной памяти. Накопленная выборка длины N поступает на блок прямого ортогонального преобразования, где вычисляется последовательность $X[k]$. Прямое ортогональное преобразование должно обладать та-

кими свойствами, чтобы последовательность $X[k]$ имела малое число значимых членов. Кодированию подвергаются только члены $X[k]$, которые переносят существенную информацию. За счет этого и достигается, в основном, сжатие сигнала. После кодирования последовательность $X[k]$ поступает в канал связи или в устройство хранения информации. В декодере выполняются обратные процессы.

Ключевым моментом в реализации метода является выбор ортогонального преобразования, а также выбор способа кодирования коэффициентов $X[k]$. Ортогональное преобразование считается эффективным, если оно обладает быстрым вычислительным алгоритмом, обеспечивает наибольшую концентрацию энергии спектральных составляющих в небольшом числе членов последовательности $X[k]$ и малые искажения при их кодировании и декодировании.

Наиболее полно сформулированным требованиям отвечает ДКП [20]. *Прямое и обратное ДКП* последовательности $x[n]$ имеют вид

$$X[k] = \lambda[k]\sum_{n=0}^{N-1} x[n]\cos\left[\frac{(2n+1)k\pi}{2N}\right], \qquad (1.169)$$

$$x[n] = \sum_{k=0}^{N-1} \lambda[k]\cdot X[k]\cdot\cos\left[\frac{(2n+1)\cdot k\pi}{2N}\right],$$

где N – длина выборки $x[n]$; $X[k]$ – коэффициенты ДКП; $\lambda[k]$ - совокупность весовых функций, причем

$$\lambda[k] = \begin{cases} \sqrt{\dfrac{2}{N}}, & при\ k = 1, 2, \ldots N-1 \\ \dfrac{1}{\sqrt{N}}, & при\ k = 0 \end{cases}. \qquad (1.171)$$

Множество базисных функций ДКП

$$\left\{\frac{1}{\sqrt{N}},\ \sqrt{\frac{2}{N}}\ \cos\left[\frac{(2n+1)k\pi}{2N}\right]\right\} \qquad (1.172)$$

соответствует множеству полиномов Чебышева. Покажем это, для чего найдем нули полинома Чебышева (1.63) степени N. Для этого перепишем его в форме:

$$T_N(z) = \cos(N\cdot\arccos z).$$

Отсюда $\cos(N\cdot\arccos z) = 0$, если $N\cdot\arccos z = \pi/2 + n\pi$ и

$$z = \cos\frac{\pi/2 + n\pi}{N} = \cos\frac{(2n+1)\pi}{2N}\,. \qquad (1.173)$$

Выполним нормировку многочленов Чебышева в соответствии с (1.171):

$$T_0(z) = \frac{1}{\sqrt{N}};$$

$$T_k(z) = \sqrt{\frac{2}{N}} \cdot \cos(k \cdot \arccos\ z), \quad k = 1, 2, \ldots N-1\,. \qquad (1.174)$$

Подставим (1.173) в (1.74) (т.е. вычислим значения полиномов степени k=1, 2, …, N-1 в тех точках, в которых полином N-ой степени дает нули). С учетом этого получим

$$\begin{cases} T_0(z) = \dfrac{1}{\sqrt{N}}, \\ T_k(z) = \sqrt{\dfrac{2}{N}} \cdot \cos\left(k \cdot \arccos\left(\cos\dfrac{(2n+1)\cdot\pi}{2N}\right)\right) = \sqrt{\dfrac{2}{N}}\cos\dfrac{(2n+1)\cdot\pi k}{2N}\,. \end{cases} \qquad (1.175)$$

Множество полиномов (1.175) эквивалентно множеству базисных функций ДКП, т.е. – это множество полиномов Чебышева порядка 0, 1, 2, … N-1, вычисленных в точках, в которых полином N-ой степени дает нули.

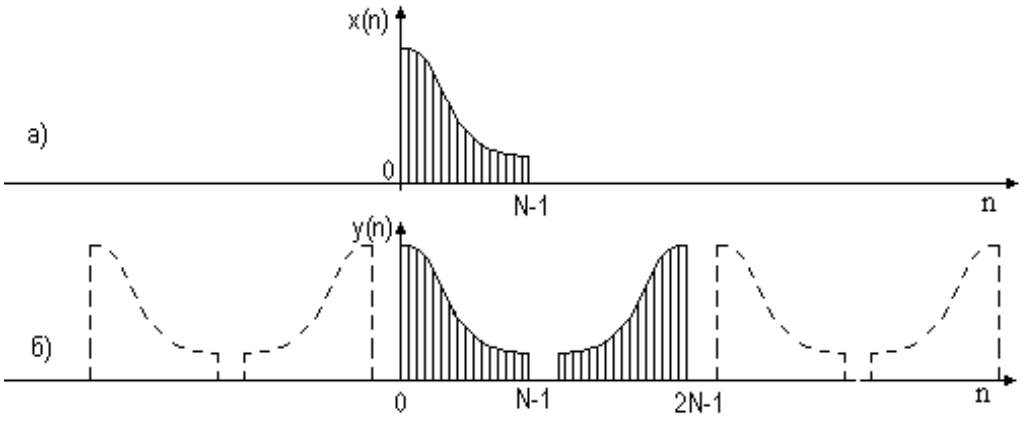


Рис. 1.25. Преобразование входной последовательности

Обратим внимание на то, что коэффициенты $X[k]$ в (1.169) не являются комплексными. Они имеют действительные значения. Кроме этого, в соответствии с общим принципом построения ДКП, последовательность $x[n]$ является действительной и четной. Поэтому ДКП как бы применяется не к исходной выборке $x[n]$ (рис. 1.25,а), а к преобразованной последовательности $y[n]$, являющейся четным расширением $x[n]$:

$$y[n]=\begin{cases}\dfrac{1}{2}x[n], & при\ \ n=0,1,\ldots N-1,\\ \dfrac{1}{2}x[2N-1-n], & n=N,N+1,\ldots 2N-1.\end{cases} \qquad (1.176)$$

Последовательность $y[n]$ должна рассматриваться как периодическая с периодом $2N$ (рис.1.25,б). Данное обстоятельство необходимо учитывать при синтезе сигналов в соответствии с (1.170).

Значения коэффициентов ДПФ $X_F(k)$ $2N$-точечной последовательности $y[n]$ определяются по формуле:

$$X_F[k]=\sum_{n=0}^{2N-1}y[n]e^{-j\frac{2\pi kn}{2N}}=e^{j\frac{\pi k}{2N}}\sum_{n=0}^{N-1}x[n]e^{-j\frac{\pi k(2n+1)}{2N}}. \qquad (1.177)$$

Из (1.169) и (1.177) следует, что коэффициенты ДКП $X_c[k]$ могут быть вычислены через коэффициенты ДПФ $X_F[k]$:

$$X_c[k]=\lambda[k]\,\mathrm{Re}\left\{e^{-j\frac{\pi k}{2N}}X_F[k]\right\}. \qquad (1.178)$$

Наличие связи между коэффициентами Фурье и ДКП позволяет вычислять ДКП с использованием алгоритма БПФ.

Достоинством ДКП является то, что по своей эффективности оно мало отличается от оптимального преобразования Карунена – Лоэва и существенно превосходит преобразование Фурье [12,20].

1.5.6 Дискретное преобразование Лапласа и Z - преобразование

Для дискретной последовательности $x[n]$ вводится понятие дискретного преобразования Лапласа в соответствии с формулой [1]

$$X(e^{pT_0})=\sum_{n=0}^{\infty}x[n]e^{-pnT_0}. \qquad (1.179)$$

Здесь, как и в случае непрерывного преобразования Лапласа, комплексная величина p равна $\sigma+j\omega$. Если $\sigma<\infty$, то ряд (1.179) сходится, и последовательности $x[n]$ соответствует изображение $X(e^{pT_0})$, являющееся функцией величины e^{pT_0}.

При анализе и синтезе систем дискретной обработки сигналов большое распространение получило z–преобразование, которое связано с дискретным преобразованием Лапласа и вытекает из него.

Прямое z *–преобразование* $X(z)$ последовательности $x[n]$ определяется формулой [1,4,5]

$$X(z) = Z\{x[n]\} = \sum_{n=0}^{\infty} x[n]z^{-n} \quad , \tag{1.180}$$

где $z = e^{pT_0}$.

Z-преобразование практически совпадает с дискретным преобразованием Лапласа и отличается только аргументом изображения. При такой замене трансцендентные функции от аргумента p преобразуются в рациональные функции от аргумента z. Благодаря этому упрощается анализ.

Рассмотрим z –изображения некоторых дискретных сигналов. Специальным входным сигналом для дискретных систем является *единичная импульсная функция*

$$\delta[n] = \begin{cases} 1 \quad при \quad n = 0, \\ 0 \quad при \quad n \neq 0. \end{cases} \tag{1.181}$$

Эта функция играет в дискретных системах такую же роль, как и δ-функция Дирака в непрерывных системах. Подставляя (1.181) в (1.180), получаем $X(z)$ =1.

Дискретный сигнал

$$x[n] = \begin{cases} 1 \ , \ n \geq 0, \\ 0 \ , \ n < 0 \end{cases} \tag{1.182}$$

называют *единичным скачком*. Подставляя (1.182) в (1.180), получаем бесконечный ряд

$$X(z) = 1 + z^{-1} + z^{-2} + \ldots \ . \tag{1.183}$$

Ряд (1.183) является геометрической прогрессией и сходится, если $|z^{-1}| < 1$. Суммируя прогрессию, получаем

$$X(z) = \frac{1}{1 - z^{-1}} \quad . \tag{1.184}$$

Аналогично находится z-преобразование простой экспоненциальной последовательности

$$x[n] = \begin{cases} a^n \quad , n \geq 0, \\ 0 \quad , n < 0, \end{cases} \tag{1.185}$$

где a – вещественное число. Подставив (1.185) в (1.180), получим

$$X(z) = \sum_{n=0}^{\infty} a^n z^{-n} = \sum_{n=0}^{\infty} \left(az^{-1}\right)^n = \frac{1}{1 - az^{-1}} \quad . \qquad (1.186)$$

Ряд (1.186) сходится, если $|az^{-1}| < 1$ или $|z^{-1}| < 1/a$.

Отметим, что комплексная плоскость $z = e^{pT_0} = x + jy$ является круговой. При движении точки *p*-плоскости вдоль оси $j\omega$, соответствующая точка z-плоскости опишет окружность единичного радиуса. Один полный оборот соответствует изменению частоты ω от 0 до $2\pi/T_0$. Левая *p*-полуплоскость отображается внутрь единичного круга. Правая *p*-полуплоскость преобразуется во всю z-плоскость, исключая единичный круг (рис.1.26).

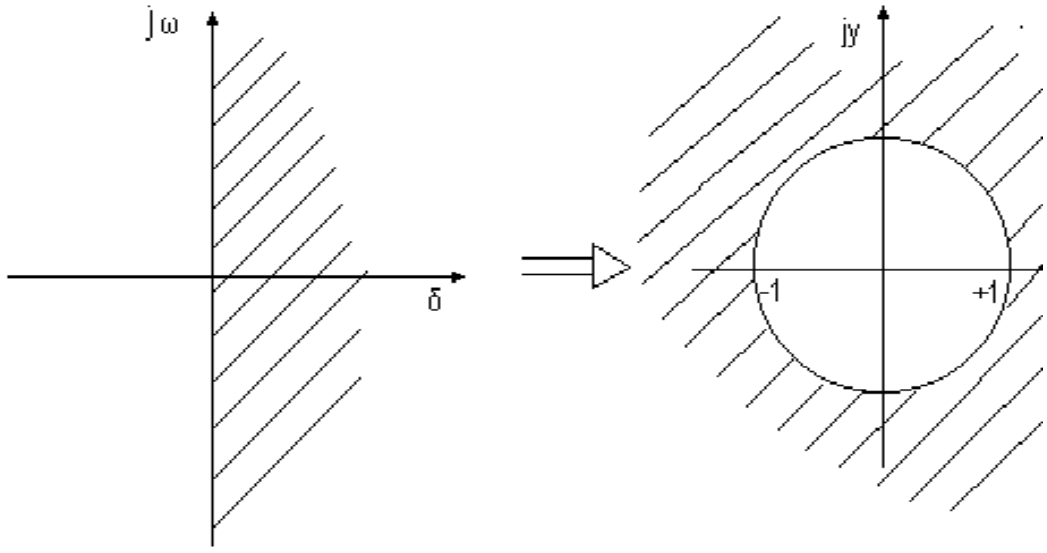


Рис.1.26. Соответствие *p*-плоскости и *z*-плоскости

Обратное z -преобразование определяется формулой

$$x[n] = Z^{-1}\{X(z)\} = \frac{1}{2\pi j} \oint X(z) z^{n-1} dz \quad , \qquad (1.187)$$

где обход замкнутого контура интегрирования в z-плоскости проводится против часовой стрелки. Контур интегрирования должен охватывать все полюсы функции $X(z)z^{n-1}$. Существует возможность вычисления $x[n]$ по более простой формуле:

$$x[n] = \frac{1}{h!}\left[\frac{d^{n} X(z^{1})}{dz^{n}}\right]_{z=0} . \qquad (1.188)$$

Рассмотрим без доказательств основные свойства z-преобразований.

1 Линейность. Пусть

$$x[n] = a_1 x_1[n] + a_2 x_2[n] . \qquad (1.189)$$

Тогда z-преобразование линейной комбинации нескольких дискретных сигналов представляет такую же линейную комбинацию их z-преобразований.

$$X(z) = a_1 X_1(z) + a_1 X_1(z) . \qquad (1.190)$$

2 **Z-преобразование смещенного сигнала.** Рассмотрим сигнал $x[n-m]$, запаздывающий на целое число тактов m. Тогда

$$Z\{x[n-m]\} = z^{-m} X(z) . \qquad (1.191)$$

Таким образом, задержка сигнала во временной области приводит к умножению его z-преобразования на множитель z^{-m}. Множитель z^{-1} соответствует задержке на один такт.

3 **Z-преобразование свертки.** Применительно к дискретным сигналам по аналогии с непрерывной сверткой вводится дискретная свертка последовательностей x[n] и h[n]

$$y[n] = \sum_{k=0}^{\infty} x[k]h[n-k] . \qquad (1.192)$$

Вычислим Z-преобразование свертки

$$Y(z) = \sum_{n=0}^{\infty}\left(\sum_{k=0}^{\infty} x[k]h[n-k]\right)z^{-n} = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty} x[k]z^{-k}h[n-k]z^{-(n-k)} =$$

$$= \sum_{k=0}^{\infty} x[k]z^{-k}\sum_{l=0}^{\infty} h[l]z^{-l} = X(z)H(z) . \qquad (1.193)$$

Таким образом, Z-преобразование свертки двух дискретных сигналов равно произведению их Z-преобразований.

1.5.7 Дискретная свертка и ее вычисление

Цифровая обработка сигналов выполняется над последовательностями конечной длины. Свертка двух конечных последовательностей $x[n]$ и $h[n]$, длиной N_1 и N_2, определяется соотношением

$$y[n] = \sum_{k=0}^{n} x[k]h[n-k] \tag{1.194}$$

или

$$y[n] = \sum_{k=0}^{n} h[k]x[n-k], \tag{1.195}$$

где n=0,1,2,….,N_1+N_2-2. Последовательность $y[n]$ также является конечной и имеет длину N_1+N_2-1. Выражения (1.194) и (1.195) определяют *линейную свертку.*

Если рассматриваются периодические последовательности x[n] и h[n] с периодами по N отсчетов, то вводят понятие *круговой свертки*, определяемой соотношениями

$$y[n] = \sum_{k=0}^{N-1} x[k]h[n-k] \tag{1.196}$$

или

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \quad . \tag{1.197}$$

Последовательность $y[n]$ также будет периодической с периодом в N отсчетов, поэтому достаточно ее вычислять на одном периоде.

Вычисление непосредственно по формулам (1.196) или (1.197) требует выполнения $N(N+1)$ операций умножения, что является ограничивающим фактором при реализации свертки. Уменьшить число умножений можно с помощью БПФ. Действительно, если для исходных периодических последовательностей $x[n]$ и $h[n]$ вычислены в соответствии с (1.159) их ДПФ $X[k]$ и $H[k]$, то, учитывая свойство(1.193), можно показать, что

$$Y[k] = X[k]H[k] \quad . \tag{1.198}$$

Выполнив обратное преобразование Фурье для последовательности $Y[k]$, получим

$$y[n] = \frac{1}{N}\sum_{k=0}^{N-1} Y[k]W_N^{-nk} \quad . \tag{1.199}$$

Последовательность y[n] есть искомая свертка. Применяя алгоритмы БПФ для реализации прямого и обратного ДПФ, можно значительно сократить количество операций при вычислении круговой свертки. Такой метод вычисления свертки эффективен для последовательностей, длина которых превышает 32, и называется *методом быстрой свертки*.

Линейная свертка также может быть вычислена с помощью БПФ. Для этого исходные последовательности $x[n]$ и $h[n]$ длиной N_1 и N_2 дополняют нулями до $N_1 + N_2 - 1$ отсчета. Тогда линейная свертка непериодических последовательностей $x[n]$ и $h[n]$ будет соответствовать круговой свертке

$$y[n] = \sum_{k=0}^{N_1+N_2-2} x[k]h[n-k] \quad , n = 0,1,....., N_1 + N_2 - 2 \qquad (1.200)$$

и может быть вычислена с использованием $(N_1 + N_2 - 1)$ – точечного БПФ. Имеются и другие методы быстрого вычисления свертки, использующие теоретико-числовые преобразования и модульную арифметику в кольце полиномов [4].

1.5.8 Линейные дискретные системы и цифровые фильтры

Важную роль в цифровой обработке сигналов играют линейные дискретные системы. Такие системы преобразуют входную последовательность $x[n]$ в выходную $y[n]$. Дискретная система считается заданной, если для любой допустимой входной последовательности $x[n]$ может быть найдена выходная последовательность. Среди всевозможных дискретных систем наибольший интерес представляют системы, которые могут быть заданы композицией из следующих трех элементов:

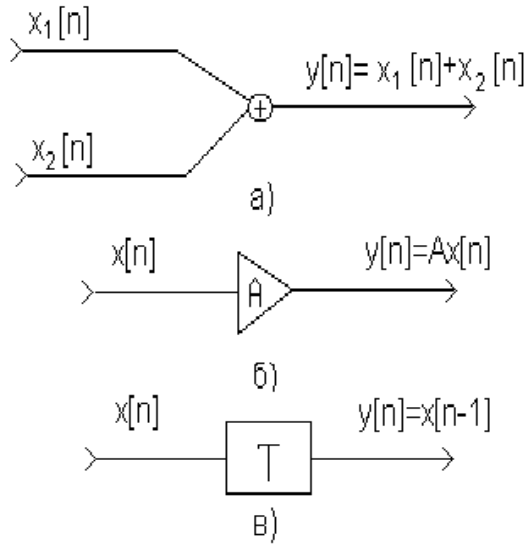


Рис. 1.27. Элементы ЛДС

а) сумматора последовательностей (рис. 1.27,а)

$$y[n] = x_1[n] + x_2[n] \; ; \qquad (1.201)$$

б) умножителя на постоянный коэффициент A (рис.1.27,б)

$$y[n] = Ax[n] \; ; \qquad (1.202)$$

в) задержки (рис.1.27,в)

$$y[n] = x[n-1] . \qquad (1.203)$$

Системы, состоящие из указанных элементов, называются *линейными дискретными системами* (ЛДС) или *линейными дискретными фильтрами* (ЛДФ). Если обозначить оператор преобразования ЛДС через L, то для линейной системы справедливы соотношения:

$$\begin{aligned} &y_1[n] = L\{x_1[n]\} \;; \quad y_2[n] = L\{x_2[n]\} , \\ &a_1 y_1[n] + a_2 y_2[n] = L\{a_1 x_1[n] + a_2 x_2[n]\} \quad , \end{aligned} \qquad (1.204)$$

где a_1, a_2 – константы.

Инвариантной к сдвигу называется такая система, для которой сдвиг входной последовательности на m тактов приводит к соответствующему сдвигу выходной последовательности, т.е.

$$y[n-m] = L\{x[n-m]\} . \qquad (1.205)$$

Покажем, что для инвариантной к сдвигу ЛДС входная и выходная последовательности связаны соотношением свертки. Реакция ЛДС на единичный импульс называется *импульсной характеристикой* $h[n]$, т.е.

$$h[n] = L\{\delta[n]\} . \qquad (1.206)$$

Единичный импульс $\delta[n]$, действующий только при n=0, вызывает на выходе системы отклик $h[n]$. Следовательно, реакция ЛДС на произвольный входной отсчет $x[k]$, действующий в точке $k \neq 0$, будет представлять импульсную характеристику $h[n]$, сдвинутую на k тактов и умноженную на значение $x[k]$, т.е. $x[k]\cdot h[n-k]$. Реакция ЛДС на всю последовательность отсчетов $x[k]$ равна сумме реакций на каждый из входных отсчетов [4,10,16]:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[k-n] . \qquad (1.207)$$

Импульсная характеристика полностью определяет свойства ЛДС при нулевых начальных условиях.

Большое значение имеют физически реализуемые и устойчивые ЛДС. ЛДС является *физически реализуемой*, если $h[n]$=0 при n<0. В этом случае значение отклика ЛДС при $n=n_0$ зависит только от входных отсчетов с номерами $n \le n_0$.

ЛДС является *устойчивой*, если ее импульсная характеристика удовлетворяет условию [4,16]:

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty \quad . \tag{1.208}$$

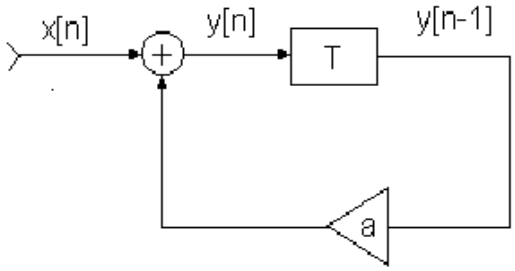


Рис. 1.28. ЛДС первого порядка

Для устойчивой ЛДС ограниченная по значениям входная последовательность $x[n]$ вызывает ограниченную выходную реакцию $y[n]$ при любых начальных условиях.

Рассмотрим инвариантную к сдвигу ЛДС, образованную из базовых элементов (рис.1.28). Уравнение функци-онирования рассматриваемой системы имеет вид:

$$y[n] = x[n] + ay[n-1] \,. \tag{1.209}$$

Данное уравнение является разностным уравнением первого порядка. Разностное уравнение позволяет рассчитать выходную последовательность по входной. Пусть $x[n]=\delta[n]$ и $y[n]=0$ при $n<0$. Тогда в соответствии с (1.209) получаем:

$$\begin{aligned} &y[0] = x[0] + ay[-1] = 1; \\ &y[1] = x[1] + ay[0] = a; \\ &y[2] = x[2] + ay[1] = a^2; \\ &\ldots\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots \\ &y[n] = x[n] + ay[n-1] = a^n. \end{aligned} \tag{1.210}$$

В общем случае инвариантная к сдвигу ЛДС, построенная на основе базовых элементов, функционирует на основе *разностного уравнения* общего вида [4,10,16]

$$y[n] = \sum_{k=0}^{N-1} b[k]x[n-k] - \sum_{k=1}^{M-1} a[k]y[n-k] \,, \tag{1.211}$$

где $a[k]$, $b[k]$ – постоянные коэффициенты.

Цифровое устройство, реализующее выражение (1.211), называется *цифровым фильтром* (ЦФ). В ЦФ все операции выполняются над отсчетами, представленными двоичными числами ограниченной разрядности. Очевидно, что конечная разрядность обусловливает погрешность ЦФ. Эта погрешность может быть уменьшена путем увеличения разрядности ЦФ. Ниже вводятся основные характеристики ЦФ в предположении, что разрядность ЦФ достаточно велика и погрешностями вычислений можно

пренебречь. Это позволяет для синтеза и исследования ЦФ использовать теорию ЛДС. Эффекты, связанные с конечной разрядностью представления отсчетов, рассматриваются в § 1.6.

Если известны коэффициенты $a[k]$ и $b[k]$, отсчеты входной последовательности $x[n]$ и начальные значения $y[-1]$, y[-2],…, y[-M], то, используя (1.211), можно вычислить выходной сигнал $y[n]$ при $n \geq 0$. Фильтр, функционирующий на основе (1.211), называется *рекурсивным фильтром* (РФ). Структурная схема РФ изображена на рис.1.29.

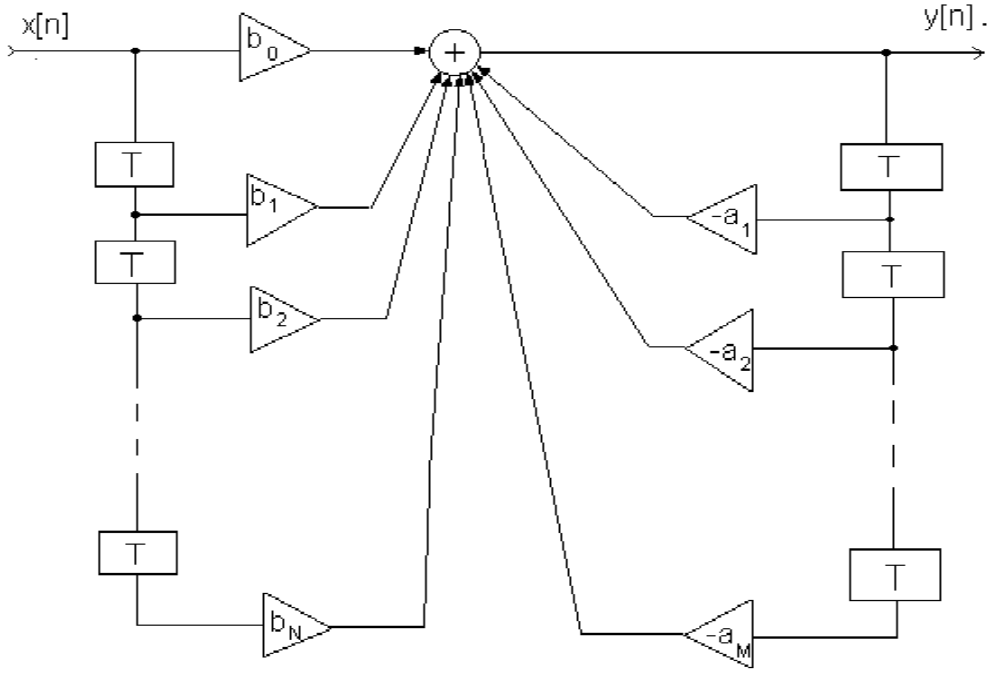


Рис. 1.29. Структурная схема рекурсивного цифрового фильтра

Если в уравнении (1.211) все коэффициенты $a[k]=0$, то

$$y[n] = \sum_{k=0}^{N} b[k]x[n-k]\,. \qquad (1.212)$$

Фильтр, функционирующий на основе уравнения (1.212), называется *нерекурсивным фильтром* (НРФ) и имеет конечную длительность импульсной характеристики. Такие фильтры также называют *фильтрами с конечной импульсной характеристикой* (КИХ-фильтрами). Фильтры, реализующие (1.211), характеризуются *бесконечной импульсной характеристикой* и называются БИХ-фильтрами. Из (1.212) и (1.207) следует, что для НРФ $b[k]=h[k]$, т.е. коэффициенты КИХ-фильтра совпадают с отсчетами его импульсной характеристики.

Важной характеристикой рассматриваемых фильтров является *передаточная функция* $H(z)$, которая представляет отношение z-образов выходной $Y(z)$ и входной последовательности $X(z)$ фильтра при нулевых начальных условиях:

$$H\ (z) = Y\ (z) / X\ (z)\ . \qquad (1.213)$$

Выполнив Z преобразование над левой и правой частями (1.211) и (1.212), получим передаточные функции для рекурсивного и нерекурсивного фильтров соответственно:

$$H\ (z) = \frac{\sum_{k=0}^{N-1} b[k]z^{-k}}{1 + \sum_{k=1}^{M-1} a[k]z^{-k}}\ , \qquad (1.214)$$

$$H\ (z) = \sum_{k=1}^{N-1} b[k]z^{-k}\ . \qquad (1.215)$$

Используя передаточные функции, можно получить комплексные частотные характеристики фильтров. Для этого выполняют подстановку $z=exp(j\omega T_0)$. Модуль комплексной частотной характеристики $A(\omega)=|H(exp(j\ \omega T_0))|$ называется *амплитудно-частотной характеристикой*. Аргумент комплексной частотной характеристики $\varphi(\omega)=arg[H(exp(j\ \omega T_0))]$ называется *фазо-частот-ной характеристикой* (ФЧХ). Так как частотные характеристики фильтров - функции аргумента $exp(j\ \omega T_0)$, то они являются периодическими по частоте с периодом, равным частоте дискретизации $\omega_д=2\pi/T_0$. Для фильтров, характеризуемых действительными коэффициентами, АЧХ является четной функцией частоты ω, а ФЧХ – нечетной.

С учетом периодичности требования к частотным характеристикам фильтров задают на интервале частот $[0,\ \pi/T_0]$. При этом для сопоставления фильтров используют нормирование. Существует два способа нормирования [4]. В первом случае вводится нормированная частота $w = \omega T_0$. Тогда при $\omega_д = 2\pi / T_0$, $w = 2\pi$ и требования к фильтрам задают на интервале $[0,\ \pi]$. Во втором случае используют нормировку по частоте $\omega T_0/2\pi$. При $\omega = \omega_д$ нормированное значение равно 1. В этом случае требования к фильтрам задают на интервале $[0;\ 0{,}5]$.

Импульсная $h[n]$ и частотная характеристики $H(exp(j\ \omega T_0))$ связаны между собой парой ДНПФ (1.156).

Пример 1.11 Определим АЧХ РФ первого порядка (рис. 1.28). Выполнив z-преобразование (1.209), получим $Y(z)=X(z)+a\cdot Z^{-1}Y(z)$. Отсюда

$$H(z) = \frac{Y(z)}{H(z)} = \frac{1}{1 + az^{-1}}.$$

Подставив $Z=exp(j\omega T_0)$, получим

$$H(e^{j\omega T_0}) = \frac{1}{1 + ae^{-j\omega T_0}}.$$

Следовательно, АЧХ равна

$$\left|H(e^{j\omega T_0})\right| = \frac{1}{\sqrt{(1 + a\cos\omega T_0)^2 + (a\sin\omega T_0)^2}}.$$

Существует несколько различных форм реализации РФ фильтров. Выделяют четыре основные формы: прямую, каноническую, последовательную и параллельную.

Прямая форма реализации РФ показана на рис. 1.29 и соответствует непосредственному использованию выражения (1.211).

Каноническая форма (случай, $N=M$) получается из (1.214). Представим $H(z)$ в виде

$$H(z) = \frac{1}{1 + \sum_{k=1}^{M-1} a[k]z^{-k}} \sum_{k=0}^{N-1} b[k]z^{-k} = H_1(z)H_2(z), \qquad (1.216)$$

где $H_1(z)$, $H_2(z)$ – передаточные функции соответственно РФ и НРФ, включенных последовательно. Передаточным функциям $H_1(z)$ и $H_2(z)$ соответствуют разностные уравнения

$$u[n] = x[n] - \sum_{k=1}^{M-1} a[k]u[n-k], \qquad (1.217)$$

$$y[n] = \sum_{k=0}^{N-1} b[k]u[n-k], \qquad (1.218)$$

которые можно реализовать в соответствии с рис.1.30.

Расположенная слева от элементов задержки часть схемы (рис.1.30) соответствует уравнению (1.217). Реализация уравнения (1.218) основывается на том, что отсчеты $u[n],u[n-1],\ldots,u[n-N]$ вспомогательного сигнала уже получены и хранятся на выходах элементов задержки. Поэтому для реализации второго уравнения не требуются дополнительные элементы задержки. Введение вспомогательной последовательности $u[n]$ позволяет уменьшить число элементов задержки в два раза по сравнению с прямой формой реализации РФ. Существуют и другие виды канонических форм реализации, обладающие указанным свойством.

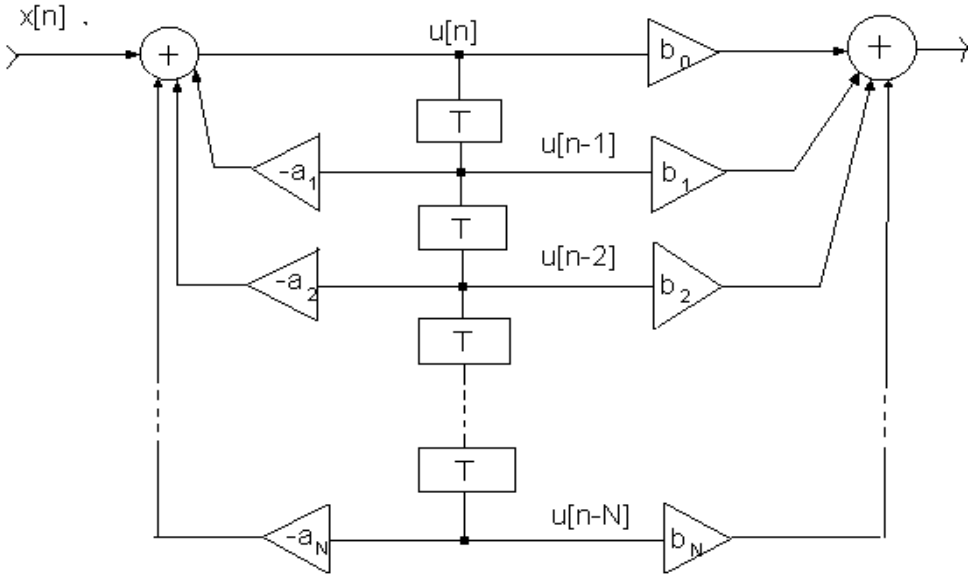


Рис. 1.30. Каноническая форма реализации РФ (1)

В частности, при реализации ЦФ с использованием ЦПОС часто используют каноническую форму, изображенную на рис.1.31.

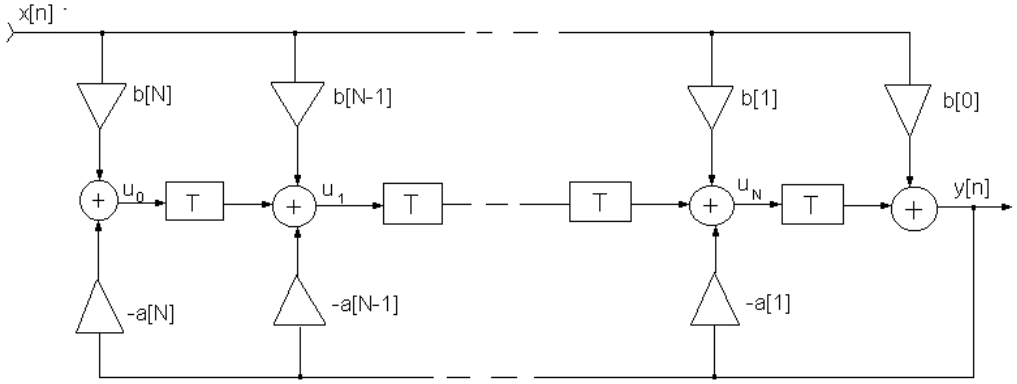


Рис.1.31. Каноническая форма реализации РФ (2)

Данная форма реализации РФ удобна с вычислительной точки зрения. Для каждого значения n выходная последовательность вычисляется с помощью выражения

$$y[n] = b[0]x[n] + u_N \quad . \tag{1.219}$$

В свою очередь, значение сигнала u_n, действующего на выходе n-го сумматора, равно

$$u_n = b[1]x[n-1] - a[1]y[n-1] + u_{n-1}, \tag{1.220}$$

где

$$u_{n-1} = b[2]x[n-2] - a[2]y[n-2] + u_{n-2} \tag{1.221}$$

и т.д. В итоге вычисления по приведенным формулам соответствуют выражению (1.211).

Как правило, реализация цифровых РФ высокого порядка в прямой или канонической форме из-за ошибок вычислений, обусловленных конечной разрядностью, нецелесообразна. В этом случае предпочтительнее реализовывать фильтры с использованием простых звеньев второго порядка

$$H(z) = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2}}{1 + a[1]z^{-1} + a[2]z^{-2}}. \qquad (1.222)$$

Такие звенья называют *биквадратными блоками*. *Последовательная форма* реализации ЦФ (рис.1.32) представляет собой последовательное соединение однотипных звеньев, соответствующих (1.222). В этом случае H(z) представляется в виде произведения

$$H(z) = \prod_{i=1}^{L} H_i(z), \qquad (1.223)$$

где $H_i(z)$ соответствует (1.222). При этом каждый биквадратный блок реализуется в канонической форме.

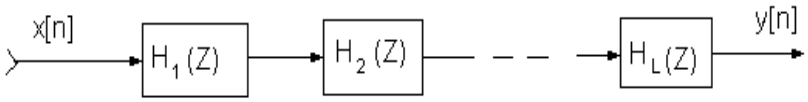


Рис. 1.32. Последовательная форма реализации ЦФ

Параллельная форма реализации (рис.1.33) соответствует представлению передаточной функции $H(z)$ в виде суммы

$$H(z) = \sum_{i=1}^{L} H_i(z), \qquad (1.224)$$

где $H_i(z)$ соответствует (1.222) при $b[2]=0$.

Нерекурсивные фильтры также могут быть реализованы в различных формах. Прямая форма (рис.1.34) соответствует реализации фильтра в соответствии с (1.212). НРФ также могут быть реализованы в эквивалентной форме (рис.1.35), соответствующей структуре , показанной на рис. 1.31.

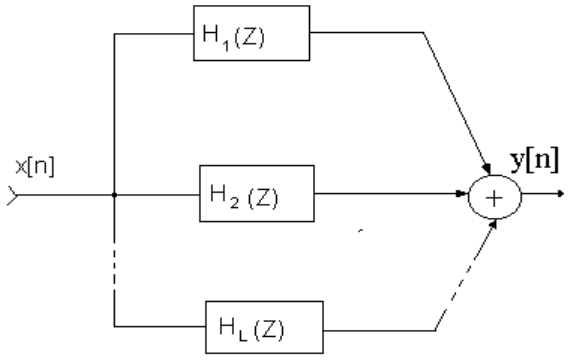


Рис.1.33. Параллельная форма реализации ЦФ

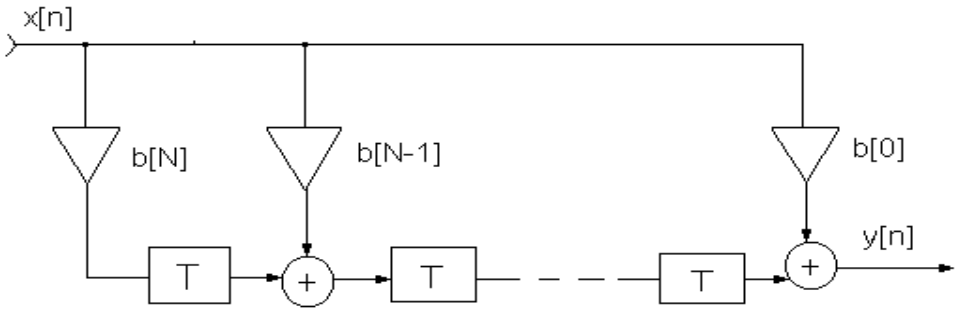


Рис. 1.34 Прямая форма реализации НРФ

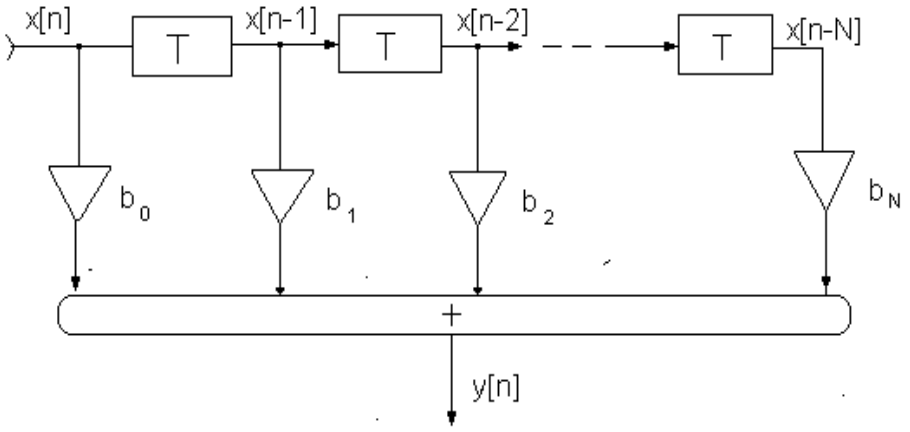


Рис. 1.35. Эквивалентная форма реализации НРФ

Нерекурсивный фильтр всегда устойчив. Это следует из (1.208). Действительно,

$$\sum_{k=0}^{N} \left| b\,[\,k\,] \right| < \infty \quad . \qquad (1.225)$$

Для определения устойчивости РФ рассмотрим частный случай (1.211), когда $x[n]=0$ для всех n. Тогда

$$y[n]=-\sum_{k=1}^{M}a[k]y[n-k]. \qquad (1.226)$$

Разностное уравнение (1.226) называется однородным и определяет собственное поведение РФ при ненулевых начальных условиях. Общее решение уравнения (1.226) имеет вид

$$y[n]=\sum_{k=1}^{M}c[k]z_k^n, \qquad (1.227)$$

где $c[1]$, $c[2]$,….,$c[k]$ – числа, определяемые начальными условиями; Z_1, Z_2,….Z_k – полюсы передаточной функции $H(z)$ (1.214). Предполагается, что все полюсы простые. Выходная последовательность $y[n]$, определяемая с помощью (1.226), будет ограничена по значениям, если выполняется условие

$$|z_k|<1, \quad k=1,2,....M, \qquad (1.228)$$

т.е. полюсы передаточной функции должны находиться внутри единичной окружности на комплексной z-плоскости. Рекурсивный фильтр устойчив, если устойчиво решение соответствующего однородного разностного уравнения. Следовательно, критерий (1.228) является также критерием устойчивости РФ.

Рассмотренные критерии устойчивости (1.208) и (1.228), строго говоря, относятся лишь к ЛДС, в которых отсутствует квантование коэффициентов. Реальные ЦФ из-за ограниченной разрядности вычислений требуют дополнительных исследований устойчивости.

1.5.9 Системы с повышением и понижением частоты дискретизации

При решении задач ЦОС иногда возникает необходимость в изменении частоты дискретизации. Это сопряжено с повышением эффективности обработки сигналов и обеспечивает возможность обработки с минимально допустимыми частотами дискретизации. Основными элементами систем, использующих различные частоты дискретизации, являются дециматоры и интерполяторы [3,4,21].

Дециматор – это элемент, понижающий частоту дискретизации входного сигнала. Уравнение функционирования дециматора имеет вид

$$y_d[n]=x[Mn], \quad M-\text{целое}. \qquad (1.229)$$

Значение выходной последовательности дециматора в точке *n* равно значению входной последовательности в точке *Mn*, т.е. на выход дециматора поступают только входные отсчеты с номерами, кратными *M*.

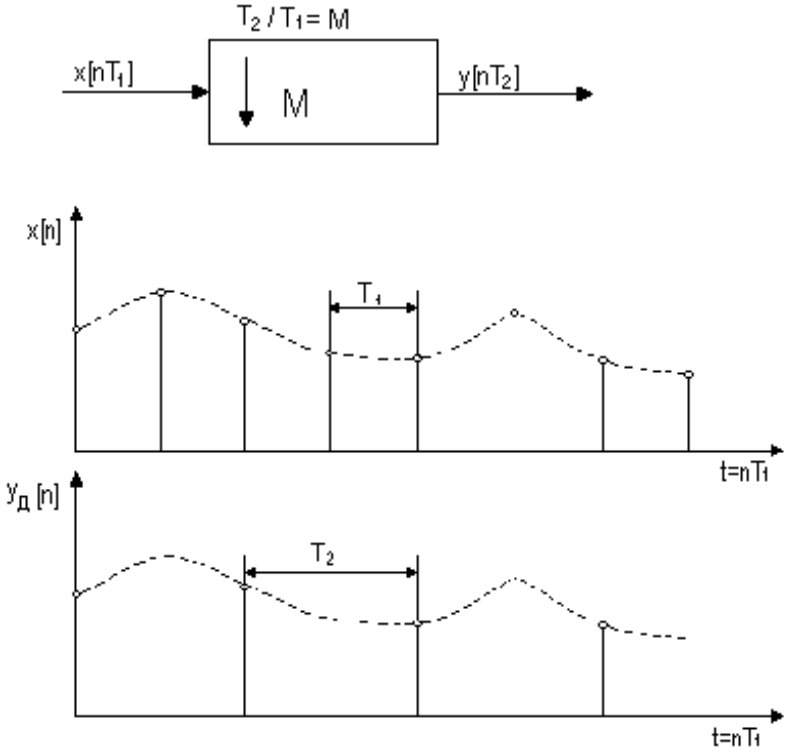


Рис. 1.36. Дециматор и пример децимации

Если входные отсчеты дециматора следуют с периодом T_1, то выходные отсчеты - с периодом $T_2=T_1 \cdot M$ (рис. 1.36). Операция, выполняемая дециматором, называется *прореживанием*.

Рассмотрим соотношение спектров сигналов $x[n]$ и $y_d[n]$. Спектр сигнала $x[n]$ периодичен с частотой дискретизации $\omega_1=2\pi/T_1$. Спектр сигнала $y_d[n]$ является периодическим с частотой дискретизации $\omega_2=2\pi/T_2=2\pi/(T_1 \cdot M)$, $\omega_2<\omega_1$.

Из-за уменьшения частоты дискретизации при децимации возможно наложение повторяющихся спектров, что приводит к потере информации. Рис. 1.37 иллюстрирует эффект наложения для случая M=2. Для исключения наложения необходимо ограничивать полосу частот входного сигнала дециматора значением $|\omega|< \omega_2/2= \pi/T_2$. Для этого перед дециматором устанавливают фильтр нижних частот, идеализированная АЧХ которого должна удовлетворять требованиям:

$$| H (e^{j\omega}) | = \begin{cases} 1, & |\omega| < \pi / T_2 \\ 0, & |\omega| \geq \pi / T_2 \end{cases} . \qquad (1.230)$$

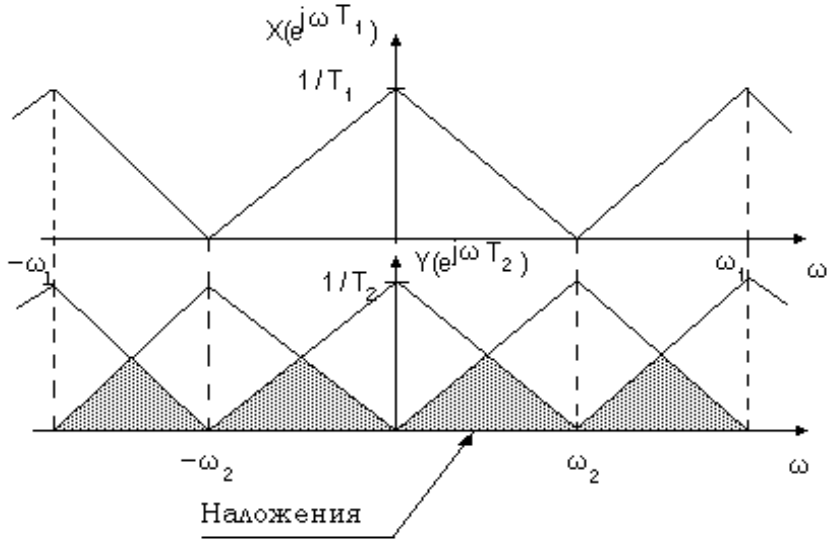


Рис. 1.37. Эффект наложения при децимации, M=2

Последовательное соединение ФНЧ и дециматора образует *простейшую нисходящую дискретную систему* (ПНДС) (рис. 1.38) [4]. В ПНДС понижение частоты дискретизации производится однократно.

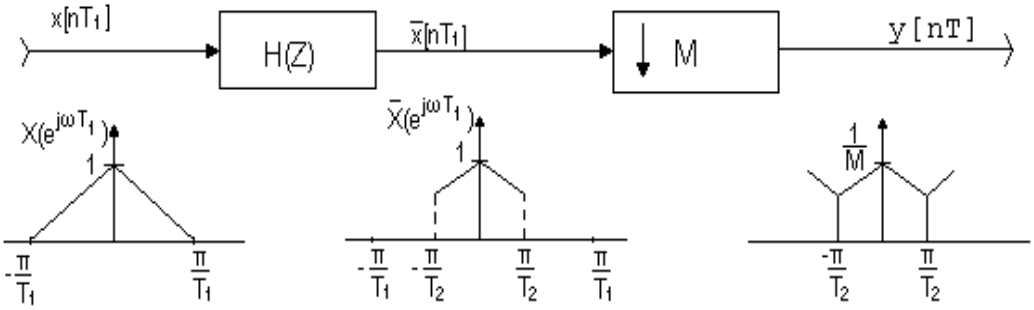


Рис.1.38. Простейшая нисходящая дискретная система

ФНЧ, применяемый в ПНДС, должен иметь малую ширину переходной полосы и большое подавление в полосе задерживания. Эти требования проще реализуются при использовании рекурсивной схемы построения фильтра. Однако в РФ для вычисления очередного выходного отсчета необходимо знать значения предыдущих отсчетов. Поэтому такой фильтр должен работать на частоте дискретизации входного сигнала. В случае применения НРФ нет необходимости в вычислении всех отсчетов последовательности $\bar{x}[n]$ (рис. 1.38), так как для вычисления очередного выходного отсчета не требуются значения предыдущих выходных

отсчетов. Поэтому НРФ, используемый в схеме ПНДС, работает на пониженной частоте дискретизации, что уменьшает количество операций, необходимых для вычисления $\bar{x}[n]$. Кроме того НРФ, обладающий линейной фазовой характеристикой, обеспечивает сохранение формы сигнала.

Интерполятор – это элемент, повышающий частоту дискретизации входного сигнала. Уравнение функционирования интерполятора имеет вид

$$y_I[n] = \begin{cases} x\left[\dfrac{n}{L}\right], \text{если } n \text{ кратно } L \\ 0 \quad , \text{в остальных случаях}. \end{cases} \tag{1.231}$$

Выходная последовательность интерполятора $y_I[n]$ получается в результате ввода между соседними отсчетами последовательности $x[n]$ дополнительных L-1 отсчетов с нулевыми значениями. На рис.1.39 показан интерполятор и последовательности $x[n]$ и $y_I[n]$ при L=2.

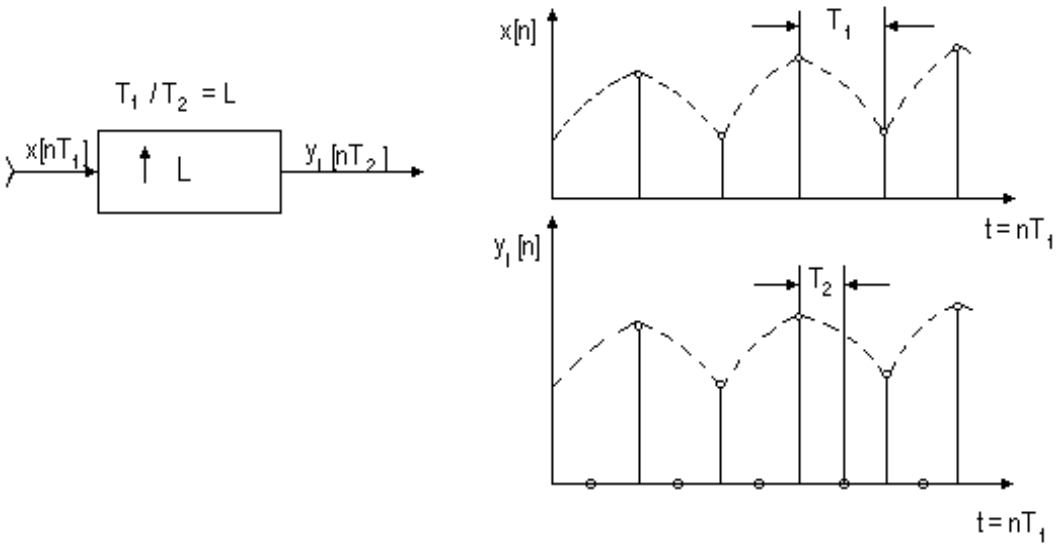


Рис.1.39. Интерполятор и пример интерполяции

Определим спектр выходного сигнала $y_I[n]$. В соответствии с дискретно-непрерывным преобразованием Фурье спектр выходного сигнала интерполятора равен

$$Y\left(e^{j\omega T_2}\right) = \sum_{n=0}^{\infty} y[nT_2]e^{-j\omega nT_2} . \tag{1.232}$$

Последовательность $y[nT_2]$ отлична от нуля только для значений n, кратных L. Поэтому, заменив n на nL, получим

$$Y\left(e^{j\omega T_2}\right)=\sum_{n=0}^{\infty} y[nLT_2]e^{j\omega nLT_2}=\sum_{n=0}^{\infty} x[nT_1]e^{j\omega nT_1}=X\left(e^{j\omega T_1}\right), \quad (1.233)$$

т.е. спектры входного и выходного сигнала интерполятора совпадают (рис. 1.40).

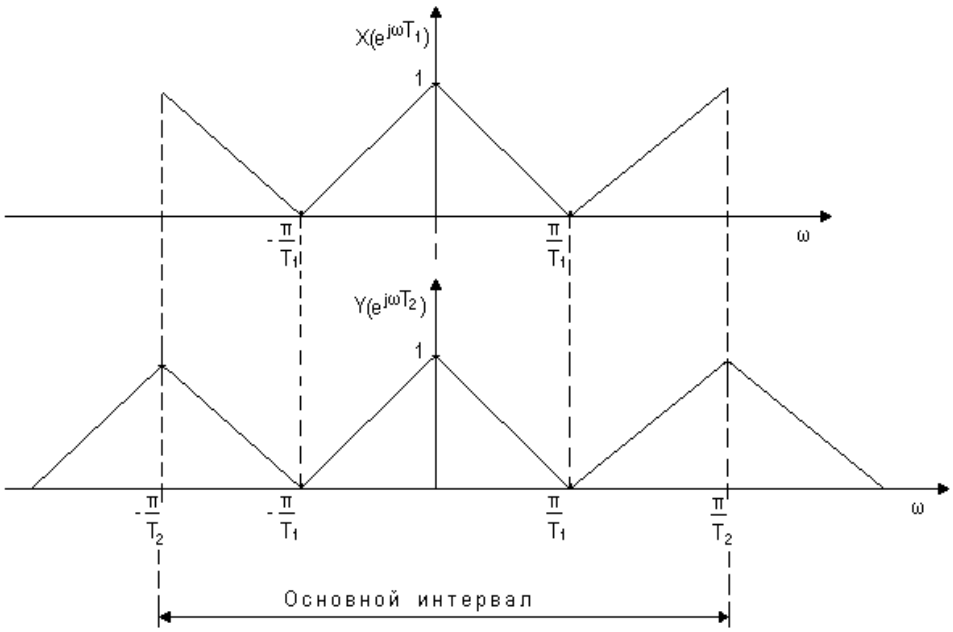


Рис. 1.40. Преобразование спектров при интерполяции

Спектр сигнала $y_1[n]$, отсчеты которого следуют с частотой $2\pi/T_2$, периодичен с частотой дискретизации входного сигнала $2\pi/T_1$. Если предположить, что некоторый сигнал $\overline{y}[nT_2]$ получается путем непосредственной дискретизации исходного аналогового сигнала с частотой дискретизации $2\pi/T_2$, то он будет иметь спектр, показанный на рис. 1.41.

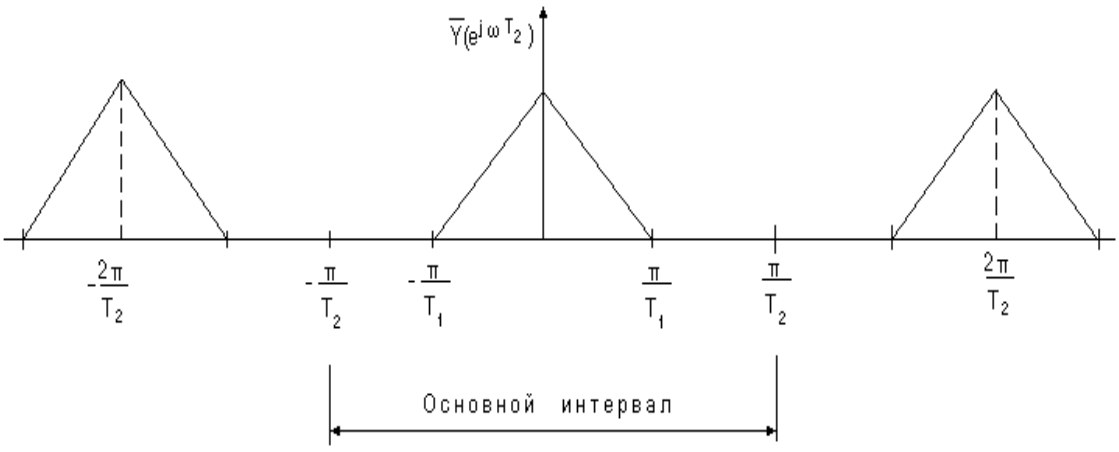


Рис. 1.41. Спектр сигнала $\overline{y}[nT_2]$

Из сравнения спектров сигналов $y[nT_2]$ и $\bar{y}[nT_2]$, следует, что сигнал на выходе интерполятора имеет в основном интервале “лишние” спектральные составляющие. Требуемый сигнал $\bar{y}[nT_2]$ может быть получен из $y[nT_2]$ путем исключения их из основного интервала. Для этого на выходе интерполятора устанавливают ФНЧ. Идеализированная АЧХ такого фильтра должна удовлетворять требованиям:

$$\left|H\left(e^{j\omega}\right)\right| = \begin{cases} L, & |\omega| < \pi / T_1, \\ 0, & \dfrac{\pi}{T_1} \le |\omega| \le \dfrac{\pi}{T_2}. \end{cases} \qquad (1.234)$$

Отличие значения АЧХ фильтра от единицы в полосе частот $|\omega| < \pi / T_1$ объясняется тем, что спектр сигнала $y[nT_2]$ имеет масштабный множитель $1/T_1$, а спектр сигнала $\bar{y}[nT_2]$ должен иметь масштабный множитель $1/T_2$ (см. рис.1.42).

Последовательное соединение интерполятора и ФНЧ образует *простейшую восходящую дискретную систему* (ПВДС, рис. 1.42) [4]. В ПВДС повышение частоты дискретизации производится однократно.

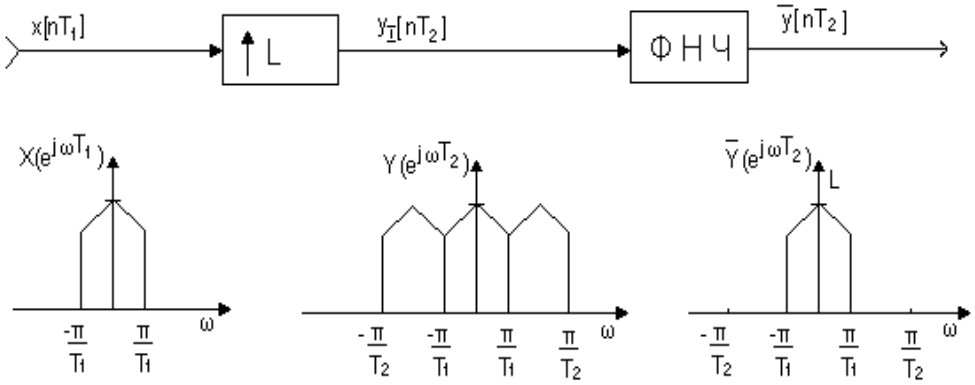


Рис. 1.42. Простейшая восходящая дискретная система

В ПВДС преимущественное применение находят нерекурсивные фильтры, так как они, благодаря линейности ФЧХ, позволяют сохранить форму сигнала. Кроме этого, НРФ, применяемые в ПВДС, требует меньшего количества операций умножения, поскольку во входном сигнале $y_I[nT_2]$ только каждый L-й отсчет отличен от нуля.

Последовательное соединение ПВДС и ПНДС позволяет построить систему, выполняющую преобразование частоты дискретизации в соотношении, выражаемом нецелым числом M/L [3]. На рис. 1.43 изображена структурная схема системы, выполняющей дробное преобразо-

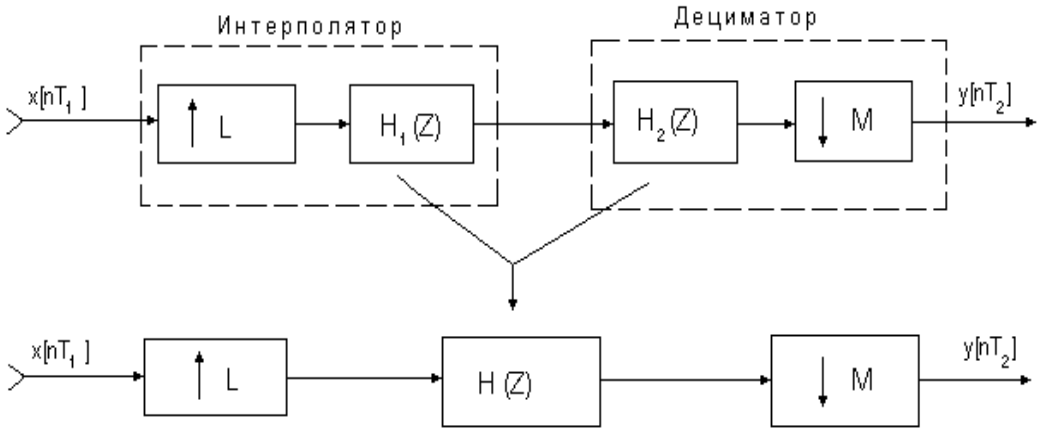


Рис. 1.43. Дробное преобразование частоты

вание частоты дискретизации. Здесь $T_2 = (M / L) \cdot T_1$. Если $M<L$, то частота дискретизации последовательности $y[nT_2]$ выше, чем у последовательности $x[nT_1]$ и полоса пропускания идеального ФНЧ определяется граничным значением $|\omega| = \pi / T_1$. Если $M>L$, то частота дискретизации последовательности $y[nT_2]$ ниже, чем у $x[nT_1]$ и идеальный ФНЧ должен иметь граничную частоту пропускания $|\omega| = \pi / T_2$.

Системы с дробным преобразованием частоты дискретизации находят широкое применение в цифровой обработке аудиосигналов.

1.6 Эффекты квантования в цифровой обработке сигналов

1.6.1 Формы представления чисел

В предыдущих разделах алгоритмы цифровой обработки сигналов рассматривались в предположении о неограниченной длине разрядной сетки вычислителя. Однако на практике число разрядов всегда ограничено, что приводит к появлению ошибок вычислений. При определенных условиях эти ошибки можно не учитывать, например, когда шаг квантования мал по сравнению с уровнем полезного сигнала. В общем случае ошибки вычислений могут накапливаться до значительной величины. Оценка результирующей ошибки вычислений представляет большой практический интерес. Результирующая ошибка вычислений зависит от особенностей алгоритма обработки, а также от формы представления чисел в вычислителе.

Обычно применяются две формы представления чисел: с фиксированной и с плавающей запятой (точкой) [4,10,16]. При использовании формы с *фиксированной запятой* считается, что положение запятой, от-

деляющей дробную часть числа от целой части, фиксировано и неизменно в процессе вычислений. В вычислителях, использующих эту форму представления чисел, возможны несколько вариантов представлений, отличающихся положением запятой (справа от младшей значащей цифры или слева от старшей значащей цифры) и способом представления отрицательных чисел.

Процессоры цифровой обработки сигналов, оперирующие числами с фиксированной запятой, обычно используют 16-разрядные двоичные кодовые слова.

Когда положение запятой фиксируется справа от младшей значащей цифры, то каждое число представляется в виде положительных степеней числа 2:

$$X = \sum_{i=0}^{B-1} a_i 2^i , \qquad (1.235)$$

где B – разрядность кодового слова; a_i - двоичная цифра 0 или 1.

Пример 1.12

Представим десятичное число $243_{(10)}$ в двоичной форме с фиксированной запятой

$$243_{(10)} = 1\cdot 2^7 + 1\cdot 2^6 + 1\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 1111\,0011_{(2)}.$$

Числа, представленные в соответствии с (1.235), являются положительными и при B=16 лежат в диапазоне от 0 до 2^{16}-1=65536.

Для представления отрицательных чисел используют: абсолютное значение со знаком (прямой код со знаком), обратный код или дополнительный код. В каждом из этих случаев старший разряд числа кодирует знак (обычно 0 – положительный знак, 1 – отрицательный).

В табл. 1.4. приведены примеры кодов 3-х разрядных двоичных чисел.

Таблица 1.4 - Примеры 3-х разрядных кодов

Десятичное число	Прямой код	Обратный код	Дополнительный код
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
-0	100	111	-
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4	-	-	100

Положительные числа для каждого из трех кодов совпадают, т.е. представляются в виде прямого кода со знаком. Отрицательные числа представляются по-разному. В прямом коде со знаком старший разряд предназначается для знака, а остальные разряды соответствуют абсолютному значению числа. Отрицательные числа в обратном коде образуются путем инверсии цифр, соответствующих двоичному представлению модуля числа. Отрицательные числа, представляемые в дополнительном коде, вычисляются с помощью выражения

$$X_{доп} = 2^{B} - \sum_{i=0}^{B-1} a_i 2^{i} . \qquad (1.236)$$

Пример 1.13

Определим дополнительный код числа $-625_{(10)}$ для случая B=16. Выполним вычисления в двоичной системе в соответствии с выражением (1.236). В двоичной системе число $625_{(10)}$ запишется в виде 1001110001. Тогда

$$X_{доп} = \begin{array}{r} 1\ 0000\ \ 0000\ \ 0000\ \ 0000 \\ -\ \ 0000\ \ 0010\ \ 0111\ \ 0001 \\ \hline 1111\ \ 1101\ \ 1000\ \ 1111 \end{array} .$$

Единица в старшем разряде $X_{доп}$ кодирует знак числа. Из примера видно, что незначащие позиции числа в дополнительном коде заполняются битом знака (это называется знакорасширением), т.е. единицей. Для нахождения дополнительного кода отрицательного числа при выполнении действий в двоичной системе счисления часто используют более простой способ. Сначала находят обратный код модуля двоичного числа, а затем увеличивают его на единицу, т.е.

$$X_{доп} = X_{обр} + 1 . \qquad (1.237)$$

Пример 1.14

Определим дополнительный код числа $-625_{(10)}$ при B=16.

Двоичный код: 0000 0010 0111 0001.

Обратный код: 1111 1101 1000 1110.

Добавление 1: 1111 1101 1000 1111= $X_{доп}$.

Можно показать, что представление чисел в дополнительном коде является циклическим (рис. 1.44), т.е. если начать с 0 и увеличивать каждое значение на 1, то получатся все положительные числа от 0 до 2^{B}-1. Прибавив еще единицу, получим отрицательное число, поскольку знаковый разряд станет равным 1, а значение числа будет соответствовать 2^{B}. Продолжая прибавлять 1, получим все отрицательные числа, а затем получим число 0. Отсюда следует важное свойство такой системы записи

чисел. Если окончательный результат сложения нескольких чисел (положительных или отрицательных) можно представить имеющимся количеством двоичных разрядов, то этот результат будет являться точным и не зависящим от порядка выполнения сложений, даже если в процессе выполнения операций имело место переполнение.

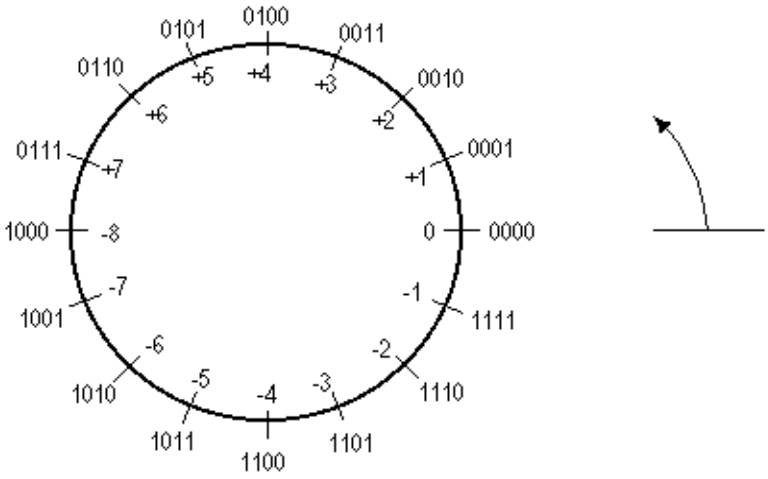


Рис.1.44. Циклическое расположение целых чисел в дополнительном коде

Пример 1.15

Рассмотрим сложение 4-х разрядных представлений чисел - 4 - 6 + 7 = - 3. Здесь - 4 - 6 = - 10 дает переполнение. Однако конечный результат будет правильным:

$$\begin{array}{l} 1100 \quad -4 \\ 1011 \quad -6 \\ \hline 0111 \\ 1101 \end{array} = \begin{array}{l} +7 \\ -3 \end{array}.$$

Рассмотренные кодовые представления целых чисел находят широкое применение в системах цифровой обработки сигналов. Прямой код используют в аналого-цифровых преобразователях и умножителях. Применение прямого кода позволяет упростить множительные устройства. Обратный и прямой коды имеют недостаток, состоящий в том, что числа +0 и –0 представляются по-разному (см. табл. 1.4). Поэтому в системах цифровой обработки сигналов чаще всего применяют дополнительный код. Использование дополнительного кода для представления в памяти вычислительного устройства отрицательных значений позволяет все арифметические операции свести к серии сложений и сдвигов. Например, операция вычитания двух чисел X–Y может быть заменена сложением: $X+Y_{доп}$, где $Y_{доп}$ - дополнительный код числа Y.

Пример 1.16

Найдем разность X – Y, где $X=71_{(10)}$ и $Y=63_{(10)}$.

$$\begin{array}{r} X = 0000\ 0000\ 0100\ 0111 \\ +\,Y_{доп} = 1111\ 1111\ 1100\ 0001 \\ \hline X + Y_{доп} = 0000\ 0000\ 0000\ 1000 = 8_{(10)} \end{array}.$$

При суммировании и вычитании двух произвольных чисел может возникнуть переполнение разрядной сетки, что приводит к неправильному результату, который без соответствующей коррекции не может быть использован в дальнейших вычислениях. Процессоры обработки сигналов имеют специальный бит в регистре состояний, который фиксирует случай переполнения.

В системах ЦОС обычно полагают, что значения обрабатываемых сигналов представляются дробными числами в диапазоне от –1.0 до 1.0. В этом случае условно считают, что запятая (точка) расположена справа от знакового бита (рис. 1.45), т.е. предполагается, что все числа масштабированы с коэффициентом $2^{(B-1)}$. Тогда наибольшее число, представимое в указанном формате, становится равным $(2^{(B-1)}-1)/2^{(B-1)} = 1-2^{-(B-1)}$, т.е. меньше единицы. Дробные числа, соответствующие рис. 1.45, лежат в диапазоне от –1 до $1 - 2^{-15}$.

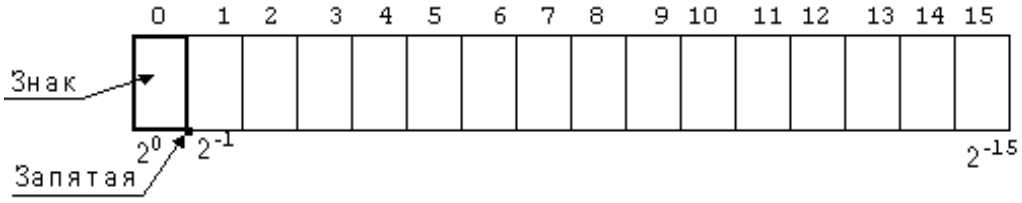


Рис.1.45. Формат представления дробных чисел

Дробные отрицательные числа хранятся в дополнительном коде. Дополнительный код числа в этом случае определяется путем вычитания модуля исходного числа из числа 2. Так как модуль исходного числа меньше 1, то в результате получается число, первый бит которого всегда равен 1.

Пример 1.17

Представим число –0,625. Дополнение числа 0,625 до числа 2 равно 2 – 0,625 = 1,375 или в двоичном представлении в соответствии с рис. 1.45

$$-\,0{,}625_{(10)} = 1{,}011\ 0000\ 0000\ 0000_{(2)}.$$

Запятая в двоичной форме записи числа является условной и отделяет знак. Для перевода полученной двоичной записи дробного числа в десятичную систему счисления используют формулу

$$N = -S + \sum_{i=1}^{B-1} a_i 2^i \ , \qquad (1.238)$$

где S – значение знакового бита числа; a_i – значение битов дробной части.

Пример 1.18

Найдем десятичный эквивалент числа 1,011 0000 0000 $0000_{(2)}$. Подставляя значения битов в формулу (1.238), получаем

$$N = -1 + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = -1 + 0,25 + 0,125 = -0,625 .$$

Применение дробных чисел с фиксированной запятой позволяет избежать переполнения при умножении. Основные проблемы применения такого представления связаны с ограниченным динамическим диапазоном. Поэтому следует особое внимание уделять масштабированию переменных.

Можно показать, что представление дробных чисел также является циклическим (рис. 1.46).

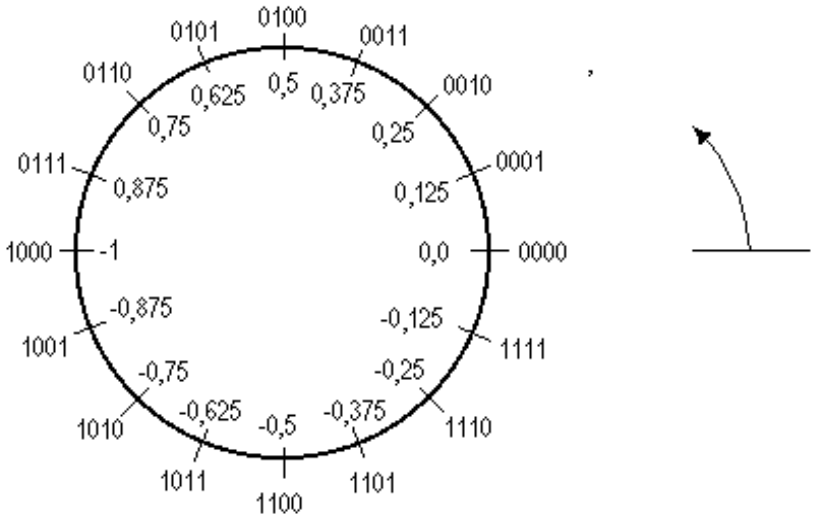


Рис. 1.46. Циклическое представление дробных чисел

Сравнивая рис. 1.45 и рис. 1.46, приходим к выводу: одни и те же кодовые комбинации представляют различные числа – целые и дробные. Таким образом, значение кодовой комбинации определяется способом ее интерпретации.

Наряду с представлением чисел в форме с фиксированной запятой, в ЦПОС широко используется *форма с плавающей запятой*. В этом случае число представляется в виде комбинации мантиссы и порядка

$$X = mP^n , \qquad (1.239)$$

где m-мантисса числа; n – порядок числа; P- основание системы счисления. Например $103,45 \cdot 10^0 = 10,345 \cdot 10^1 = 1,0345 \cdot 10^2$.

При использовании (1.239) в памяти вычислителя запоминают мантиссу и порядок. Мантисса называется *нормализованной*, если

$1/P \le m < 1.0$. В этом случае в старшем числовом разряде мантиссы стоит цифра, отличная от нуля. Например, $0{,}10345 \cdot 10^3$. Мантисса обычно представляется в памяти в виде дробного числа со знаком, а порядок – в виде целого числа без знака.

На рис. 1.47 показан 32-разрядный формат представления чисел с плавающей запятой, соответствующий стандарту IEEE/ANSI 754 [26,31].

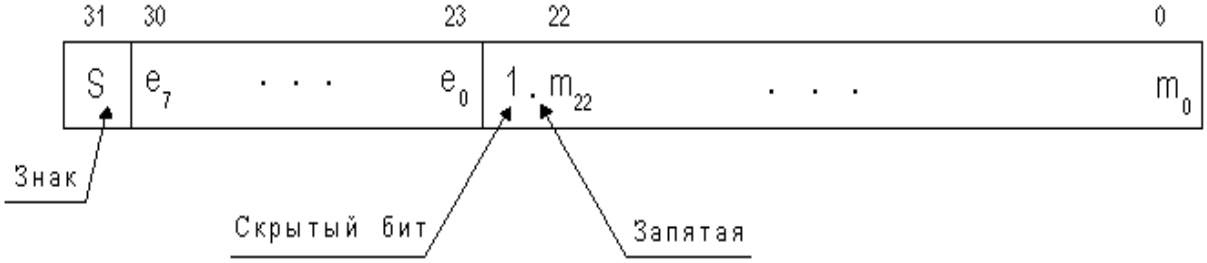


Рис. 1.47. 32-разрядный формат представления чисел с плавающей запятой

Число в этом формате включает знаковый бит *S*, 23-битовую мантиссу $m_{22}, \ldots, m_0$ и 8-битовую экспоненту $e_7, \ldots, e_0$. Мантисса дополнительно включает скрытый единичный бит, который на рис. 1.47 показан условно. Данный бит расширяет точность представления мантиссы с 23-разрядов до 24-разрядов. Соответственно в этом случае значения мантиссы лежат в диапазоне от 1,0 до 2,0. Значения экспоненты находятся в диапазоне от 1 до 254. Экспонента *e* соответствует смещенному значению порядка *n,* т.е. $e = n + 127$. Смещение необходимо для того, чтобы не хранить знак порядка.

Пример 1.19

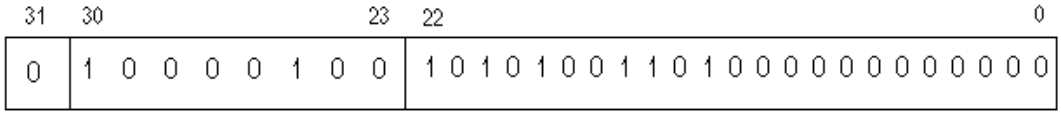
Представим число $(86{,}6249712)_{10}$ в 32- разрядном формате с плавающей запятой

$86{,}6249712_{(10)} = +110101{,}001101_{(2)} = +(1,)10101001101 \cdot 2^5$

Мантисса: 101010011010 0000 0000 000 (23 бита)

Знак: 0 '+'

Экспонента: 10000100 (+127+5)



Диапазон представления чисел с плавающей запятой намного шире, чем диапазон представления чисел с фиксированной запятой. Поэтому представление значений в формате с плавающей запятой увеличивает точность вычислений. Однако реализация арифметики с плавающей запятой сложнее. Действительно, чтобы сложить два числа в форме с плавающей запятой, необходимо сначала выровнять порядки (нормализовать числа), а затем выполнить сложение. Для умножения двух чисел в

форме с плавающей запятой требуется перемножить мантиссы и сложить порядки.

Важную роль при выборе формы представления чисел играет динамический диапазон, который определяется как отношение наибольшего числа соответствующего формата к наименьшему числу. Сравним динамические диапазоны 32-разрядного представления дробных чисел с фиксированной и с плавающей запятой. Для формата с фиксированной запятой минимальное значение равно 2^{-31}, а максимальное можно записать в виде $1 - 2^{-31}$. Динамический диапазон равен $2^{31} \approx 2{,}15 \cdot 10^{9}$ или в логарифмическом масштабе -187 дБ. Для формата с плавающей запятой (24-разрядная мантисса и 8-разрядная экспонента) минимальное и максимальное значения примерно равны $5{,}88 \cdot 10^{-39}$ и $3{,}4 \cdot 10^{38}$. Соответственно динамический диапазон будет равен $5{,}79 \cdot 10^{76}$ или 1535 дБ. Таким образом, числа с плавающей запятой обеспечивают более широкий динамический диапазон при одной и той же разрядности кодового слова.

Различные приложения цифровой обработки сигналов требуют разных значений динамического диапазона. Для большинства телекоммуникационных приложений достаточным является диапазон в 50 дБ. Для высококачественной обработки аудиосигналов требуется динамический диапазон в 90 дБ.

1.6.2 Квантование сигналов

При реализации алгоритмов цифровой обработки числовые значения переменных из-за ограниченности разрядной сетки вычислителя представляются конечным числом разрядов B. Данная операция соответствует замене непрерывных значений дискретными и называется квантованием. Операция квантования $Q\{\cdot\}$ является нелинейной, т.е.

$$Q\{x + y\} \neq Q\{x\} + Q\{y\} \qquad (1.240)$$

и вносит ошибку квантования

$$e_{кв} = Q\{x\} - x \ . \qquad (1.241)$$

Эта ошибка зависит от числа используемых двоичных разрядов, формы представления чисел и способа квантования. Квантование выполняют с использованием округления или усечения.

При *округлении* исходное m-разрядное число заменяется ближайшим B-разрядным числом($m>B$). Максимальная ошибка округления равна половине шага квантования. Шаг квантования q соответствует весовому коэффициенту младшего разряда числа, т.е. $q=2^{-B}$. Ошибка округ-

ления $e_0 = Q_0\{x\} - x$ удовлетворяет неравенству $-q/2 \le e_0 < q/2$ и не зависит от формы представления чисел. Характеристика нелинейности, соответствующая операции округления, показана на рис. 1.48,а.

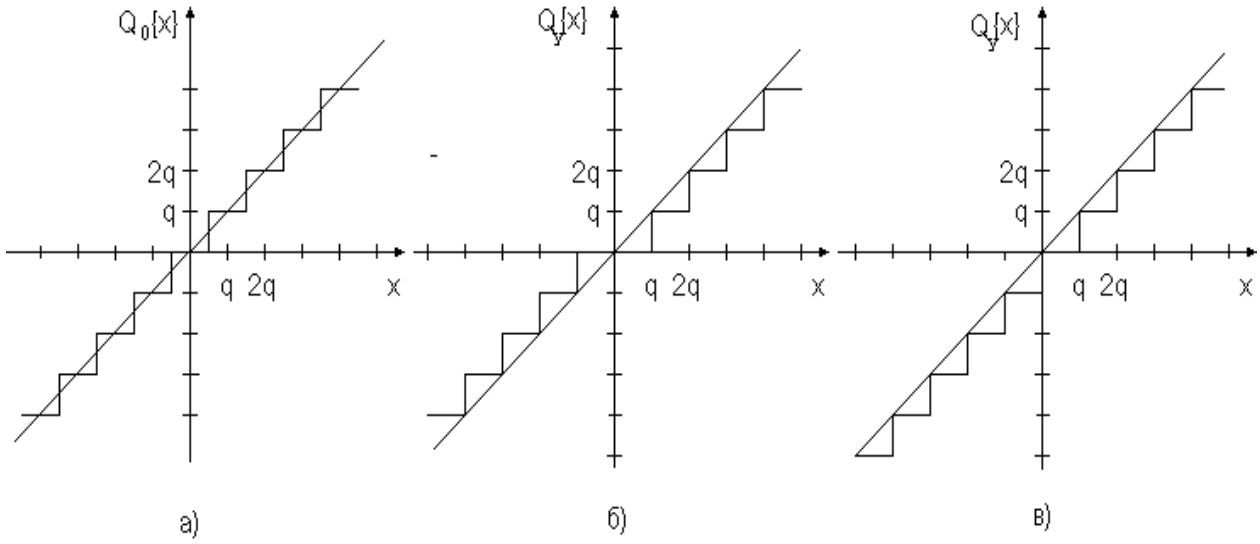


Рис. 1.48 . Характеристики нелинейности квантователей

При *усечении* исходное *m*-разрядное число заменяется *B*-разрядным числом путем отбрасывания младших *m-B* разрядов. Максимальная ошибка усечения меньше или равна шагу квантования *q*. Значение ошибки усечения зависит от формы представления отрицательных чисел. Если используется представление в прямом коде со знаком, то ошибка усечения $e_y = Q_y\{x\} - x$ удовлетворяет неравенствам

$$-q \le e_y < 0, \text{ если } x>0,$$
$$0 \le e_y < q, \text{ если } x<0, \qquad (1.242)$$

т.е. знак ошибки всегда противоположен знаку числа. Соответствующая характеристика нелинейности квантователя показана на рис. 1.48,б. Если используется представление в дополнительном коде, то ошибка усечения всегда отрицательна и удовлетворяет неравенству (рис. 1.48,в)

$$-q \le e_y < 0. \qquad (1.243)$$

В том случае, когда выбрано представление с плавающей запятой, квантование выполняется только для мантиссы числа. Однако абсолютное значение ошибки квантования будет зависеть также и от порядка числа. Поэтому при квантовании чисел, представленных в форме с плавающей запятой, рассматривают относительную ошибку квантования

$$\varepsilon = \frac{Q\{x\} - x}{x} \quad . \tag{1.244}$$

Если при квантовании мантиссы используется округление, то абсолютная ошибка округления удовлетворяет неравенству

$$-\frac{q}{2} \cdot 2^n \le Q\{x\} - x < \frac{q}{2} \cdot 2^n \,, \tag{1.245}$$

где n – порядок числа. Тогда, учитывая неравенство $2^{n-1} < x < 2^n$, получаем границы относительной ошибки квантования [16]

$$-q \le \varepsilon < q \,. \tag{1.246}$$

Аналоговый сигнал *x(t)*, подлежащий цифровой обработке, должен быть преобразован в цифровую форму. Такое преобразование выполняется аналого-цифровым преобразователем (АЦП) в два этапа. На первом этапе формируется дискретная последовательность $x[n] = x(t)\big|_{t=nT_0}$. На втором этапе значение каждого отсчета x[n] квантуется и представляется числом с конечным количеством двоичных разрядов. Квантование отсчетов $x[n]$ обычно выполняется устройством, имеющим одну из характеристик нелинейности, изображенных на рис. 1.48. В этом случае ошибка квантования представляется последовательностью

$$e[n] = Q\{x[n]\} - x[n] \,, \tag{1.247}$$

значения которой находятся в диапазоне $-q/2 \le e[n] < q/2$ при округлении и $-q \le e[n] < 0$ или $0 \le e[n] < q$ при усечении. Цифровые значения отсчетов $Q\{x[n]\}$ в дальнейшем подлежат обработке. Из (1.247) следует, что

$$Q\{x[n]\} = x[n] + e[n] \,. \tag{1.248}$$

В соответствии с (1.248) цифровые значения $Q\{x[n]\}$ представляются в виде суммы отсчетов неограниченной точности $x[n]$ и шума квантования $e[n]$. Относительно шума квантования обычно делают следующие допущения [4,16]: $e[n]$ является стационарным процессом; последовательность $e[n]$ не коррелирована с последовательностью $x[n]$; $e[n]$ представляет собой белый шум; закон распределения $e[n]$ является равномерным. Необходимо отметить, что указанные допущения не всегда верны. Например, если квантованию подвергается постоянный сигнал, то

все ошибки $e[n]$ будут одинаковы. Тем не менее, для большинства сигналов, встречающихся на практике, они применимы [16]. На рис. 1.49. изображены функции плотности вероятности ошибок квантования для различных способов квантования: а) для округления; б) для усечения и представления отрицательных чисел в прямом коде со знаком; в) для усечения и представления отрицательных чисел в дополнительном коде.

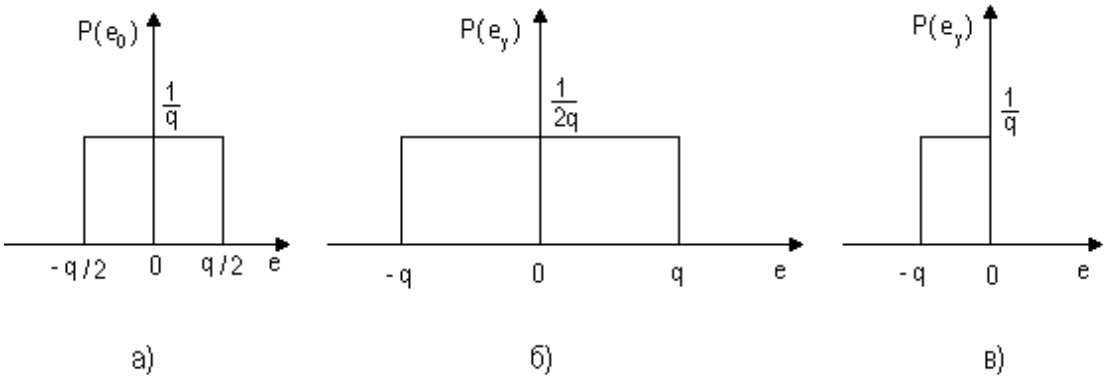


Рис. 1.49. Функции плотности вероятности ошибок квантования

Для приведенных распределений легко можно вычислить среднее значение и дисперсию ошибки квантования. Так, при округлении m_e=0 и $D_e = q^2/12$.

При рассмотрении ошибок квантования АЦП часто используют *отношение сигнал-шум (ОСШ)*

$$ОСШ = 10 \log \frac{D_x}{D_e} \text{ дБ} . \qquad (1.249)$$

Подставляя в (1.249) значение D_e и учитывая что $q=2^{-B}$, получаем

$$ОСШ = 6B + 10 \log 12 D_x . \qquad (1.250)$$

Из (1.250) следует, что добавление одного разряда АЦП (увеличение B на единицу) улучшает отношение сигнал-шум на 6 дБ.

1.6.3 Квантование и масштабирование сигналов в цифровых фильтрах

При реализации алгоритмов цифровой фильтрации выполняются две основные операции: сложение и умножение. При сложении двух B–разрядных чисел с фиксированной запятой максимальная разрядность суммы будет равна B+1. При умножении двух B–разрядных чисел произведение будет иметь разрядность 2B–1. На рис. 1.50 изображен нерекурсивный фильтр, в котором операции сложения и умножения выполняются с указанной точностью.

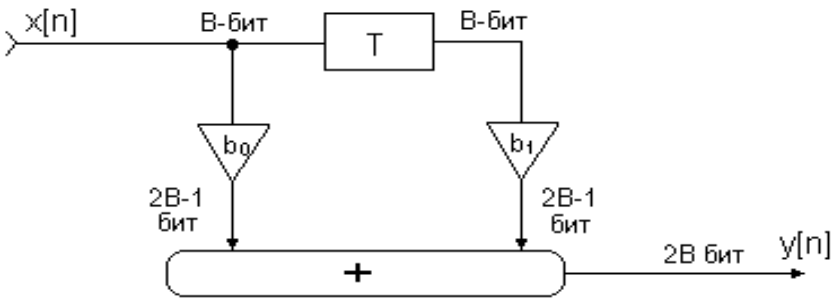


Рис.1.50. Разрядность в различных точках нерекурсивного фильтра.

На практике из–за конечной разрядности регистров цифрового фильтра результаты арифметических операций представляются *B*-разрядными кодовыми словами. При этом выполняются операции квантования (Q), ограничения (P) и масштабирования.

Операция квантования Q имеет место на выходе умножителей. При умножении двух дробных чисел произведение всегда будет меньше единицы. Представление произведения *B*-разрядным кодовым словом вместо требуемых 2*B*-1 разрядов соответствует квантованию, которое выполняется либо с помощью округления, либо усечения.

Операция ограничения имеет место на выходе сумматора. При сложении в сумматоре двух *B*–разрядных чисел их сумма может превысить максимально допустимое значение для *B* разрядов. В этом случае происходит переполнение разрядной сетки. Если числа представлены в дополнительном или обратном кодах, то переполнения в процессе многократного суммирования допустимы, если результирующая сумма не превышает 1,0. Это объясняется цикличностью представлений чисел в дополнительном и обратном кодах (рис. 1.46). Однако на циклической диаграмме наибольшее положительное число $1\text{-}2^{-B}$ и наименьшее отрицательное число –1,0 находятся рядом. Из-за этого характер переполнения сумматоров имеет пилообразный вид, показанный на рис.1.51.

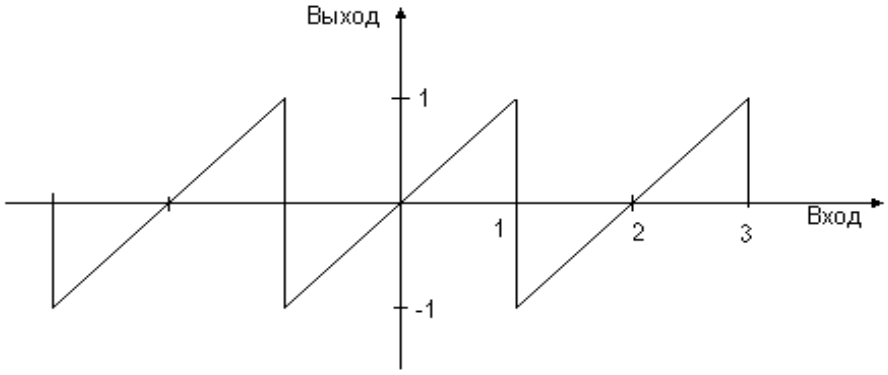


Рис. 1.51. Пилообразный характер переполнения сумматора

Переполнение происходит тогда, когда сумма входных значений сумматора выходит за пределы интервала (-1,0; 1,0). Переполнения соз-

дают колебания в фильтре и превращают его в нелинейное устройство, поэтому они крайне нежелательны и их необходимо устранять. Одним из простых способов борьбы с указанными колебаниями является использование сумматора, который имеет характеристику с насыщением (рис. 1.52)

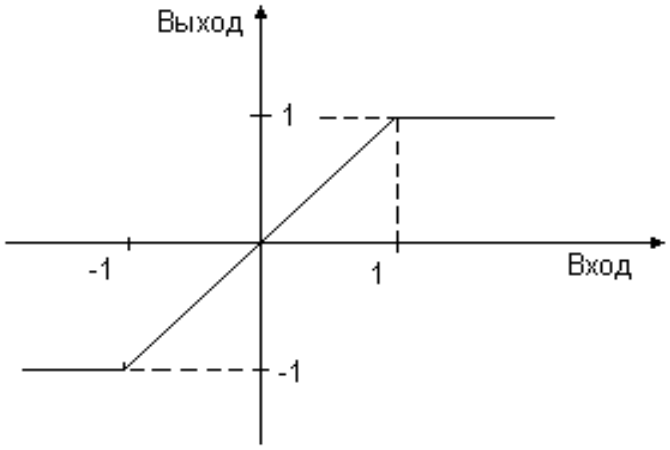


Рис. 1.52 Характеристика сумматора с насыщением

В этом случае значения на выходе сумматора ограничиваются на уровне либо 1,0, либо –1,0. Ограничение выходных значений сумматора широко применяется в цифровых фильтрах для предотвращения колебаний.

С учетом изложенных особенностей операций умножения и суммирования в схему цифрового фильтра, изображенную на рис. 1.50, необходимо внести нелинейные операции квантования Q и ограничения P (рис. 1.53).

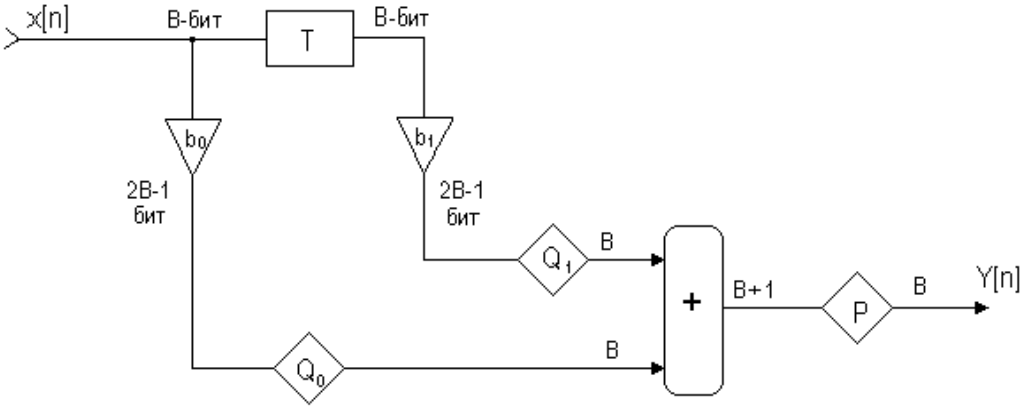


Рис.1.53 Квантование до суммирования

Так как сумматор на рис. 1.53 является линейным устройством, то операцию квантования можно учесть после выполнения суммирования (рис.1.54).

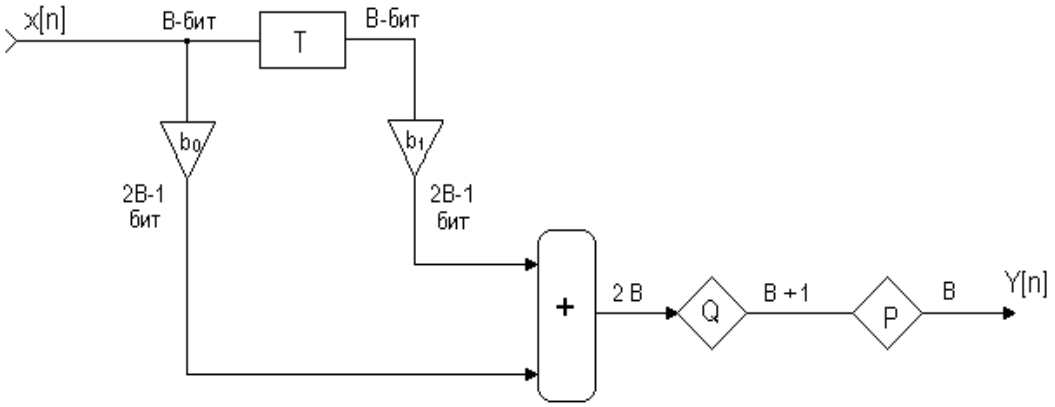


Рис. 1.54. Квантование после суммирования

Пример учета операций квантования и ограничения в структурной схеме рекурсивного фильтра показан на рис. 1.55.

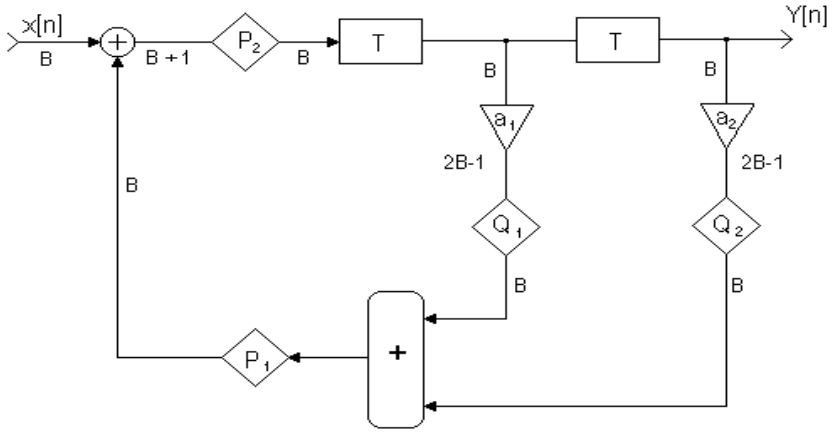


Рис 1.55. Квантование и ограничение в рекурсивном фильтре

Таким образом, в схемах реальных цифровых фильтров кроме линейных операций могут выполняться и нелинейные. Иногда это может приводить к возникновению в цифровых фильтрах *предельных циклов*[10,16]. Предельные циклы наблюдаются на выходе рекурсивных фильтров, когда входной сигнал имеет амплитуду, близкую к нулю.

Рассмотрим рекурсивный фильтр первого порядка

$$y[n] = x[n] - \alpha y[n-1]\ , \quad \alpha = 0{,}91\ , \qquad (1.251)$$

в котором выполняется квантование произведения $\alpha \cdot y[n-1]$ (рис. 1.56). Определим значения выходного сигнала фильтра $y[n]$ при нулевом сигнале на входе $x[n]=0$ и начальном условии $y[-1]=7q$, где q - шаг квантования. Результаты вычислений приведены в таблице 1.5. Здесь произведение $\alpha \cdot y[n-1]$ округляется до целого числа уровней квантования.

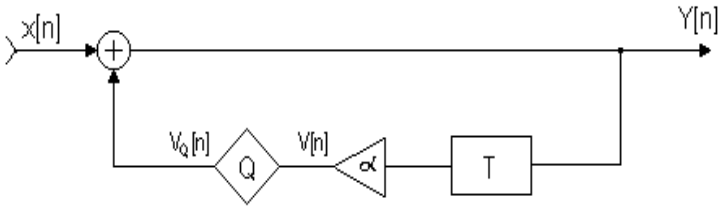


Рис 1.56. Рекурсивный фильтр первого порядка

Таблица 1.5 - Возникновение предельных циклов

n	$v[n] = \alpha \cdot y[n-1]$	$y[n] = v_Q[n]$
-1	–	+7q
0	$-0{,}91 \cdot 7q = -6{,}37q$	-6q
1	$-0{,}91 \cdot -6q = +5{,}46q$	+5q
2	$-0{,}91 \cdot 5q = -4{,}55q$	-5q
3	$-0{,}91 \cdot -5q = +4{,}55q$	+5q
4	$-0{,}91 \cdot 5q = -4{,}55q$	-5q
	и т.д.	

Из приведенной таблицы видно, что через определенное число тактов в цифровом фильтре возникают колебания. Амплитуда колебаний равна 5q, а частота колебаний равна половине частоты дискретизации. Предельные циклы весьма нежелательны, так как на выходе фильтра должны быть нулевые значения при отсутствии входного сигнала.

Одним из эффективных приемов исключения переполнений и уменьшения ошибок квантования в системах, использующих арифметику с фиксированной запятой, является масштабирование. Масштабирование выполняется умножением чисел на константу так , чтобы их значения по модулю не превышали единицы. Рассмотрим цифровой фильтр (рис. 1.57) с передаточной функцией

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad . \tag{1.252}$$

Передаточной функции (1.252) соответствует разностное уравнение

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2] \, . \tag{1.253}$$

При реализации данного уравнения в системах ЦОС, использующих арифметику с фиксированной запятой, его масштабируют [27]:

$$y_S[n] = \frac{b_0}{S_1} x_{S_2}[n] + \frac{b_1}{S_1} x_{S_2}[n-1] + \frac{b_2}{S_1} x_{S_2}[n-2] - a_1 y_S[n-1] - a_2 y_S[n-2], \tag{1.254}$$

где $y_S[n], x_{S_2}[n]$ - масштабированные последовательности $y[n]$ и $x[n]$. При этом $y_S[n] = y[n]/S$, а $x_{S_2}[n] = x[n]/S_2$ и $S = S_1 \cdot S_2$. Здесь $x[n]$ подвергается масштабированию в отношении S_2 до фильтрации, а коэффициенты числителя масштабируются в отношении S_1. Часто масштабные множители представляются в виде степени числа 2, тогда масштабирование сводится лишь к сдвигу значений. При выборе масштабных множителей руководствуются некоторой мерой, определяющей общие требования к масштабированию. Обычно для этого используют L_1 или L_2 - норму, которая накладывает ограничения на импульсную характеристику $h_k[n]$ от входа фильтра до выхода k-го сумматора

$$\left\| h_k \right\|_M = \left[\sum_{n=-\infty}^{\infty} \left| h_k[n] \right|^M \right]^{1/M}, \quad M=1, 2. \qquad (1.255)$$

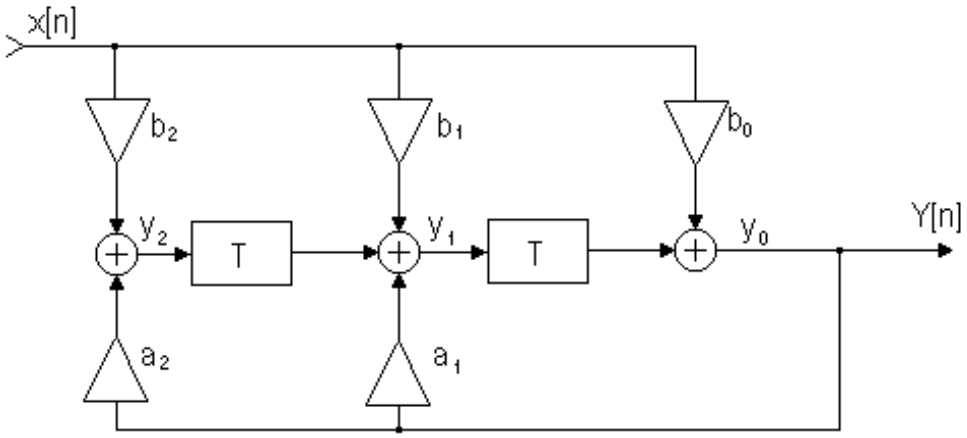


Рис. 1.57. Рекурсивный фильтр второго порядка

Иногда используют норму Чебышева, которая накладывает ограничения на амплитудно-частотную характеристику $H_k(\omega)$ от входа фильтра до выхода k-го сумматора

$$\left\| H_k(\omega) \right\| = \max_{\omega} \left| H_k(\omega) \right|. \qquad (1.256)$$

Для фильтра, изображенного на рис. 1.57, указанные АЧХ соответствуют передаточным функциям

$$H(z) = \frac{Y(z)}{X(z)}, \quad H_1(z) = \frac{Y_1(z)}{X(z)}, \quad H_2(z) = \frac{Y_2(z)}{X(z)}. \qquad (1.257)$$

Вычисленные максимальные значения меры (1.255) или (1.256) используются в качестве масштабного множителя. При вычислении значений меры (1.255) требуется вычислять импульсные характеристики $h_k[n]$,

которые являются реакциями на входной дельта–импульс $\delta[n]$, фиксируемыми на выходе сумматоров (рис. 1.57). Данные вычисления для L_1-меры можно выполнить с помощью следующего алгоритма:

$$
\begin{aligned}
&for\ n = 0\ to\ Nkon\\
&\quad y0 = y1 + b0 \cdot x[n]\\
&\quad y1 = b1 \cdot x[n] + a1 \cdot y0 + y2\\
&\quad y2 = b2 \cdot x[n] + a2 \cdot y0\\
&\quad h = h + |y|\\
&\quad h1 = h1 + |y1|\\
&\quad h2 = h2 + |y2|\\
&next\ n
\end{aligned}
$$

Здесь для каждого такта n вычисляются реакции $y0$, $y1$, $y2$ на выходе сумматоров на входной дельта–импульс в соответствии со структурной схемой фильтра (рис. 1.57). Вычисленные значения каждой из реакций суммируются по модулю и формируются значения *h, h1, h2*. Длина последовательностей ограничивается значением *Nkon*, выбираемым экспериментально.

Вычисление значений меры Чебышева можно выполнить с помощью следующего алгоритма:

```
        /* x[n] = cos( nwT )
        /* w = w0 : Начальное   значение   частоты
        /* dw = dw0 : Шаг   по   частоте
for  k = 0  to  Kkon        /* k – индекс   по   частоте
    w = w + dw
  for  n = 0  to  Nkon        /* n – индекс   по   времени
    x[n] = cos( nwT )
    /*   реакции
      y0 = y1 + b0 · x[n]
      y1 = b1 · x[n] + a1 · y0 + y2
      y2 = b2 · x[n] + a2 · y0
    /*   поиск   максимума
      if  (| y0 |> Hmax ) then  Hmax  =| y0 |
      if  (| y1 |> H1max ) then  H1max  =| y1 |
      if  (| y2 |> H2max ) then  H2max  =| y2 |
  next  n
next  k
```

Пример. 1.20

Пусть b_0=2,0585, b_1=0,4262, b_2= -1,6324, a_1= -0,8524, a_2=0,7047. В результате вычислений получаем:

$$\|h\|_1=16,5 \qquad \|h_1\|_1=14,4 \qquad \|h_2\|_1=13,3 \quad \text{и}$$
$$\|H\|=12,3 \qquad \|H_1\|=10,5 \qquad \|H_2\|=10,3.$$

Выбрав из полученных значений максимальное и округлив его до ближайшего числа, являющегося степенью двух, получим масштабирующий коэффициент S=16. Если предположить, что входная последовательность $x[n]$ формируется с помощью 12-разрядного АЦП, а фильтр использует 16-разрядную арифметику с фиксированной запятой, то при размещении 12-разрядного слова в младших разрядах 16-разрядного слова, получим масштабирующий коэффициент $S_2=8=2^3$. Так как $S = S_1 \cdot S_2$, то S_1=2. Коэффициенты фильтра после масштабирования будут равны:

$$b_0=1,0293\ ,\ b_1=0,2131\ ,\ b_2=-0,8162,\ a_1=-0,8524\ ,\ a_2=0,7047.$$

Коэффициент b_0 не может быть представлен в форме дробного числа с фиксированной запятой. Поэтому он дополнительно масштабируется $b_0^{'} = b_0/2 = 0,5147$. Для того чтобы значение произведения $b'_0 \cdot x[n]$ соответствовало $b_0 \cdot x[n]$, требуется дополнительная коррекция. После вычисления $b_0^{'} \cdot x[n]$ необходимо сдвинуть полученное произведение влево на один разряд.

1.6.4 Масштабирование сигналов при вычислении БПФ

Покажем, что при вычислении БПФ в процессорах с фиксированной запятой также возможны переполнения. Из теоремы Парсеваля [1] следует, что

$$\sum_{n=0}^{N-1} x^2[n] = 1/N \cdot \sum_{k=0}^{N-1} |X[k]|^2 \ . \qquad (1.258)$$

Перепишем (1.258) в виде

$$N \cdot [1/N \cdot \sum_{n=0}^{N-1} x^2[n]] = [1/N \cdot \sum_{k=0}^{N-1} |X[k]|^2], \qquad (1.259)$$

т.е. средний квадрат спектральных составляющих в N раз больше, чем средний квадрат отсчетов входного сигнала $x[n]$. Поэтому при вычислении БПФ в процессорах с фиксированной запятой могут происходить переполнения.

Для того чтобы выяснить как могут возникать переполнения, рассмотрим базовую операцию БПФ – “бабочка”, выполняемую на *m*-ом этапе БПФ (рис1.58.):

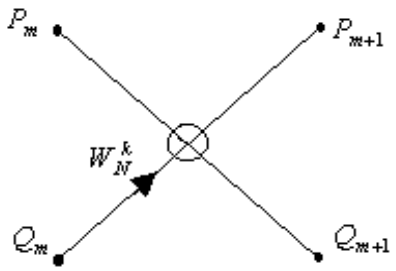


Рис.1.58. Бабочка

$$P_{m+1} = P_m + W_N^K Q_m ,$$
$$Q_{m+1} = P_m - W_N^K Q_m . \quad (1.260)$$

С целью упрощения выражения (1.260) индекс k при переменных P и Q не показан. Все переменные, входящие в выражение, являются комплексными.

Введем следующие обозначения для действительных и мнимых частей переменных выражения (1.260)

$$W_N^k = \cos(2\pi k / N) - j\sin(2\pi k / N) = \cos(x) - j\sin(x) , \quad (1.261)$$
$$P_m = PR + jPI , \; Q_m = QR + jQI , \quad (1.262)$$

где $x = 2\pi k / N$; *PR* и *QR* обозначают соответственно действительные части переменных P и Q ; *PI* и *QI* –мнимые части переменных P и Q.

Подставив (1.261) и (1.262) в (1.260), получим

$$P_{M+1} = \{PR + [QR\cos(x) + QI\sin(x)]\} + j\{PI + [QI\cos(x) - QR\sin(x)]\}, \quad (1.263)$$
$$Q_{M+1} = \{PR - [QR\cos(x) + QI\sin(x)]\} + j\{PI - [QI\cos(x) - QR\sin(x)]\}. \quad (1.264)$$

Если при вычислении “бабочки” в ЦПОС с фиксированной точкой значения входных переменных *PR,PI,QR,QI* представляются в форме дробных чисел, лежащих в диапазоне от –1 до +1, то действительные и мнимые части результирующих переменных в (1.263) и (1.264) могут иметь максимальные значения, выходящие за указанный диапазон. Например, максимальное значение действительной части переменной P_{m+1} равно:

$$\{PR + [QR \cdot \cos(x) + QI \cdot \sin(x)]\} = 1 + 1 \cdot \cos(\pi / 4) + 1 \cdot \sin(\pi / 4) \approx 2{,}4142.$$

Аналогичные максимальные значения могут иметь и другие составляющие в выражениях (1.263) и (1.264).

Чтобы исключить указанные переполнения, на каждом этапе БПФ необходимо выполнять масштабирование с коэффициентом 2. Так как при вычислении БПФ реализуется *M*-этапов, то выходные значения спектральных коэффициентов будут уменьшены в $2^M = N$ раз , где *N*- длина последовательности. Это соответствует общему требованию к масшта-

бированию, определяемому выражением (1.259). Конкретный пример реализации масштабирования сигналов при вычислении БПФ будет рассмотрен при реализации алгоритмов БПФ для ЦПОС с фиксированной запятой TMS320C2x.

1.7 Многомерные дискретные сигналы и системы

1.7.1 Двумерные дискретные сигналы

Многомерный сигнал описывается функцией M независимых переменных ($M \geq 2$). В настоящем разделе речь пойдет о двумерных сигналах, которые получили широкое распространение на практике. Дальнейшее повышение размерности сигналов в большинстве случаев может быть выполнено формальным расширением приведенных ниже формул.

Двумерный дискретный сигнал – это функция $x[n_1,n_2]$, определенная на множестве упорядоченных пар целых чисел n_1 и n_2, $-\infty<n_1,n_2<\infty$. Такие сигналы часто называют двумерными последовательностями или массивами.

Двумерный дискретный сигнал может представлять собой дискретизированный непрерывный двумерный сигнал $x(t_1,t_2)$, т.е.

$$x[n_1, n_2] = x[n_1T_1, n_2T_2] = x(t_1, t_2)\Big|_{t1=n1T2,t2=n2T2}. \quad (1.265)$$

Значения отсчетов в точке $[n_1T_1,n_2T_2]$ могут быть вещественными или комплексными. Интервалы дискретизации T_1 и T_2 могут отображаться пространственными или временными переменными.

Существует несколько способов дискретизации двумерных сигналов [7]. Рассмотрим наиболее простой способ дискретизации по прямоугольному растру. В этом случае последовательность $x[n_1,n_2]$ получается из непрерывного двумерного сигнала $x(t_1,t_2)$ путем взятия отсчетов в узлах прямоугольной сетки, отстоящих друг от друга соответственно на интервалы дискретизации T_1 и T_2.

Рассмотренная ранее теорема дискретизации одномерных сигналов может быть обобщена на случай двумерных сигналов. Функция двух переменных $x(t_1,t_2)$, для которой двумерное преобразование Фурье равно нулю при $\omega_1>\omega_{1B}$ и $\omega_2>\omega_{2B}$, однозначно определяется своими значениями в равномерно отстоящих точках плоскости (t_1,t_2), если интервалы дискретизации удовлетворяют условиям $T_1 \leq \pi/\omega_{1B}$, $T_2 \leq \pi/\omega_{2B}$.

Аналогично одномерному случаю спектр дискретного сигнала $x[n_1,n_2]$ будет периодической функцией частот ω_1 и ω_2. Повторяющиеся копии спектров не будут накладываться, если соблюдаются условия теоремы дискретизации.

Рассмотрим некоторые двумерные последовательности, которые часто используются на практике.

Единичный двумерный импульс определяется следующим образом:

$$\delta[n_1, n_2] = \delta[n_1] \cdot \delta[n_2] = \begin{cases} 1, & n_1 = n_2 = 0, \\ 0, & \textit{в остальных случаях}. \end{cases} \quad (1.266)$$

Условное графическое представление двумерного импульса показано на рис. 1.59.

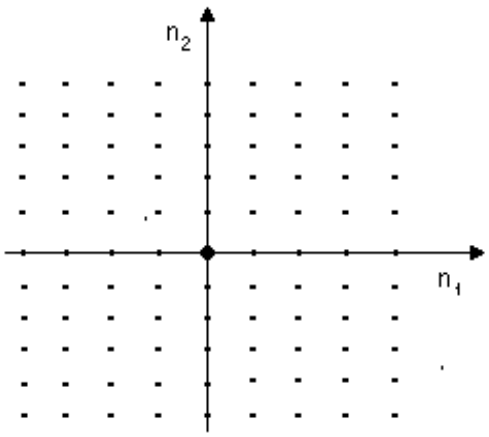


Рис.1.59. Двумерный импульс $\delta[n_1,n_2]$

Единичный скачок

$$u[n_1, n_2] = u[n_1] \cdot u[n_2] = \begin{cases} 1, & \textit{при } n_1, n_2 \geq 0 \\ 0, & \textit{в остальных случаях}. \end{cases} \quad (1.267)$$

Единичный скачок отличен от нуля в одном квадранте плоскости (n_1,n_2).

Экспоненциальная последовательность определяется следующим образом

$$x[n_1, n_2] = a^{n1} \cdot b^{n2}, \quad -\infty < n_1, n_2 < \infty, \quad (1.268)$$

где *a, b* – комплексные числа. В частном случае, когда $a = e^{j\omega_1}$ и $b = e^{j\omega_2}$, экспоненциальная последовательность представляет *комплексную синусоиду.*

Любую двумерную последовательность, которую можно представить в виде

$$x[n_1, n_2] = x[n_1] \cdot x[n_1], \quad (1.269)$$

называют *разделимой.* Последовательности (1.256)…(1.268) являются разделимыми.

Диапазон значений индексов n_1 и n_2, для которого двумерная последовательность отлична от нуля, называют опорной областью. Обычно на практике рассматривают последовательности конечной протяженности, которые отличны от нуля внутри прямоугольника $0 \le n_1 < N_1 - 1, \quad 0 \le n_2 < N_2 - 1$ или круга радиусом $R \ge \sqrt{n_1^2 + n_2^2}$.

Важным классом двумерных сигналов являются периодические двумерные последовательности. Периодический двумерный сигнал должен повторяться сразу в двух направлениях. *Периодом последовательности x* называется область плоскости (n_1,n_2), содержащая $N_1 \cdot N_2$ отсчетов, если значения отсчетов, входящих в эту область, периодически повторяются. Наиболее простой формой периода на плоскости (n_1,n_2) является прямоугольник. Однако период может иметь и другие формы [7].

1.7.2 Двумерные дискретные системы

Двумерная дискретная система выполняет преобразование входной последовательности $x[n_1,n_2]$ в выходную последовательность $y[n_1,n_2]$. Обозначим оператор преобразования системы через $L\{.\}$.

Двумерная система *линейна*, если соблюдается принцип суперпозиции, т.е.

$$L\{a_1 \cdot x_1[n_1,n_2] + a_2 \cdot x_2[n_1,n_2]\} = a_1 \cdot L\{x_1[n_1,n_2]\} + a_2 \cdot L\{x_2[n_1,n_2]\}. \quad (1.270)$$

Она является *инвариантной к сдвигу,* если

$$y[n_1 - m_1, n_2 - m_2] = L\{x[n_1 - m_1, n_2 - m_2]\}. \quad (1.271)$$

Входная $x[n_1,n_2]$ и выходная $y[n_1,n_2]$ последовательности инвариантной двумерной линейной дискретной системы (ИДЛДС) связаны уравнением свертки

$$y[n_1,n_2] = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} x[m_1,m_2] \cdot h[n_1 - m_1, n_2 - m_2], \quad (1.272)$$

где $h[n_1,n_2]$ – *импульсная характеристика* двумерной системы, являющаяся откликом ИДЛДС на единичный импульс (1.266). Выражение (1.272) путем замены переменных можно переписать в симметричной форме:

$$y[n_1,n_2] = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} h[m_1,m_2] x[n_1 - m_1, n_2 - m_2]. \quad (1.273)$$

Операцию двумерной свертки иногда обозначают двойной звездочкой **. Тогда выражения (1.272) и (1.273) можно записать в виде:

$$y[n_1, n_2] = h[n_1, n_2] * * x[n_1, n_2] \ . \qquad (1.274)$$

Разделимой называется ИДЛДС, импульсный отклик которой является разделимой последовательностью

$$h[n_1, n_2] = h[n_1] h[n_2] \ . \qquad (1.275)$$

Большой интерес представляют устойчивые двумерные дискретные системы. ИДЛДС *устойчива*, если ее выходная последовательность остается ограниченной для любой ограниченной входной последовательности. Необходимым и достаточным условием устойчивости ИДЛДС является ограниченность суммы

$$\sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} \left| h[n_1, n_2] \right| < \infty \ . \qquad (1.276)$$

Несмотря на сходство соотношений (1.276) и (1.208), исследование устойчивости двумерных систем значительно сложнее, чем одномерных.

Частотные характеристики ИДЛДС можно получить, рассматривая ИДЛДС с импульсной характеристикой $h[n_1,n_2]$ и входным сигналом, представляющим комплексную синусоиду вида

$$x[n_1, n_2] = \exp(\ j\omega_1 n_1 + \ j\omega_2 n_2) \ , \qquad (1.277)$$

где ω_1, ω_2 – соответственно горизонтальная и вертикальная пространственные частоты. Выходной сигнал ИДЛДС можно выразить через свертку (1.273) как

$$y[n_1, n_2] = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} h[m_1, m_2] \cdot e^{j\omega_1 (n_1 - m_1)} e^{j\omega_2 (n_2 - m_2)} =$$
$$= e^{j\omega_1 n_1 + j\omega_2 n_2} \cdot H\left(e^{j\omega_1}, e^{j\omega_2}\right) \ , \qquad (1.278)$$

где

$$H\left(e^{j\omega_1}, e^{j\omega_2}\right) = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} h[m_1, m_2] \cdot e^{-j\omega_1 m_1 - j\omega_2 m_2} \ . \qquad (1.279)$$

частотная характеристика ИДЛДС. Таким образом, выходной сигнал ИДЛДС представляет собой комплексную синусоиду, умноженную на частотную характеристику $H(e^{j\omega_1}, e^{j\omega_2})$. Частотная характеристика яв-

ляется периодической функцией двух переменных ω_1, ω_2 . По каждой из указанных переменных период равен 2π. Выражение (1.279) представляет разложение непрерывной двумерной функции $H(e^{j\omega_1}, e^{j\omega_2})$ в двумерный ряд Фурье. Возможно и обратное преобразование, т.е. импульсная характеристика $h[n_1,n_2]$ может быть найдена по частотной характеристике $H(e^{j\omega_1}, e^{j\omega_2})$ с использованием выражения для определения коэффициентов ряда Фурье

$$h[n_1, n_2] = \frac{1}{4\pi^2} \int_{-\pi}^{\pi}\int_{\pi}^{\pi} H\left(e^{j\omega_1}, e^{j\omega_2}\right) \cdot e^{j\omega_1 n_1 + j\omega_2 n_2} d\omega_1 d\omega_2 . \quad (1.280)$$

На рис. 3.8 показан пример импульсной и частотной характеристики двумерного фильтра нижних частот.

Аналогично одномерным системам выходной и входной сигналы ИДЛДС связаны *разностным уравнением*

$$\sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} b[m_1, m_2]x[n_1 - m_1, n_2 - m_2] = \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} a[m_1, m_2]y[n_1 - m_1, n_1 - m_2], \quad (1.281)$$

где $a[m_1,m_2]$, $b[m_1,m_2]$ – матрицы коэффициентов, которые определяют передаточную функцию двумерного рекурсивного фильтра.

Передаточная функция ИДЛДС определяется выражением

$$H\left(z_1, z_2\right) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} h[n_1, n_2] \cdot z_1^{-n_1} z_2^{-n_2} . \quad (1.282)$$

Формула (1.282) представляет двумерное Z-преобразование функции $h[n_1,n_2]$, т.е.

$$Z\left\{h[n_1, n_2]\right\} = H\left(z_1, z_2\right). \quad (1.283)$$

Выполнив двумерное Z-преобразование выражения (1.281), получим

$$H\left(z_1, z_2\right) = \frac{B\left(z_1, z_2\right)}{A\left(z_1, z_2\right)}, \quad (1.284)$$

где $A(z_1, z_2)$, $B(z_1, z_2)$ – z-преобразования матриц коэффициентов $a[m_1,m_2]$ и $b[m_1,m_2]$. Если все коэффициенты $a[m_1,m_2]$=0 за исключением $a[0,0]$=1, то из (1.281) следует

$$y[n_1, n_2] = \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} b[m_1, m_2] \cdot x[n_1 - m_1, n_2 - m_2] . \quad (1.285)$$

Уравнение (1.285) определяет нерекурсивный двумерный фильтр.

В общем случае пределы суммирования в (1.281) и (1.285) определяются выбранными опорными областями. Заданная в (1.281) опорная область, соответствующая положительным значениям индексов m_1, m_2, называется *каузальной опорной областью*.

Ранее рассматривались физически реализуемые одномерные ЛДС, для которых выполнялось условие $h[n] < 0$ при $n < 0$. Это было физически обосновано, так как координата n соответствовала времени. Для двумерных систем, где обе координаты n_1 и n_2 могут являться пространственными переменными, на значение $h[n_1, n_2]$ при $n_1, n_2 < 0$ может не накладываться ограничение $h[n_1, n_2] = 0$. В этом случае получают опорные области, отличные от каузальной.

Для ИДЛДС, определенной в каузальной опорной области, условие устойчивости записывается следующим образом:

$$\sum_{n_1=0}^{\infty}\sum_{n_2=0}^{\infty}\left|h\left[n_1, n_2\right]\right| < \infty \quad . \tag{1.286}$$

Если рассматривается нерекурсивный фильтр (1.285), то условие (1.286) выполняется всегда.

Для рекурсивных двумерных фильтров можно сформулировать условия устойчивости, сходные с условиями устойчивости одномерных фильтров. Впервые эти условия были сформулированы Шэнксом в виде теоремы [7].

Пусть передаточная функция $H(z_1, z_2) = 1/B(z_1, z_2)$ соответствует рекурсивному фильтру, определенному в каузальной области. Такой фильтр устойчив тогда и только тогда, когда $B(z_1, z_2) \neq 0$ для любой точки (z_1, z_2), такой что $|z_1| \geq 1$ или $|z_2| \geq 1$. Однако практическое применение этой теоремы затруднено. Поэтому в общем случае исследование устойчивости двумерного рекурсивного фильтра является достаточно сложной задачей. Ряд тестов проверки устойчивости приведен в [7].

1.7.3 Дискретно-непрерывное преобразование Фурье двумерных сигналов

Введенное ранее выражение (1.280) для определения импульсной характеристики $h[n_1,n_2]$ можно применить для представления произвольной периодической последовательности $x[n_1, n_2]$:

$$x[n_1,n_2]=\frac{1}{4\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}X\left(e^{j\omega_1},e^{j\omega_2}\right)\cdot e^{j\omega_1 n_1+j\omega_2 n_2}d\omega_1 d\omega_2 . \quad (1.287)$$

Здесь комплексная функция $X\left(e^{j\omega_1},e^{j\omega_2}\right)$ определяет *двумерное дискретно-непрерывное преобразование Фурье* (ДДНПФ)

$$X\left(e^{j\omega_1},e^{j\omega_2}\right)=\sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty}x[n_1,n_2]\cdot e^{-j\omega_1 n_1-j\omega_2 n_2} . \quad (1.288)$$

Для двумерного преобразования Фурье характерны следующие основные свойства, которые во многом аналогичны свойствам одномерного преобразования.

Линейность:

$$a_1x_1[n_1,n_2]+a_1x_2[n_1,n_2]\leftrightarrow a_1X_1\left(e^{j\omega_1},e^{j\omega_2}\right)+a_2X_2\left(e^{j\omega_1},e^{j\omega_2}\right). \quad (1.289)$$

Пространственный сдвиг:

$$x[n_1-m_1,n_2-m_2]\leftrightarrow e^{-j\omega_1 m_1-j\omega_2 m_2}\cdot X\left(e^{j\omega_1},e^{j\omega_2}\right). \quad (1.290)$$

Задержка последовательности $x[n_1,n_2]$ на m_1 и m_2 отсчетов по каждому из измерений соответствует умножению ее преобразования Фурье на множитель $e^{-j\omega_1 m_1-j\omega_2 m_2}$.

Модуляция:

$$x[n_1,n_2]\cdot e^{j\lambda_1 n_1+j\lambda_2 n_2}\leftrightarrow X\left(e^{j(\omega_1-\lambda_1)},e^{j(\omega_1-\lambda_2)}\right). \quad (1.291)$$

Умножение последовательности $x[n_1,n_2]$ на комплексную экспоненту соответствует сдвигу преобразования Фурье.

Свертка:

$$h[n_1,n_2]**x[n_1,n_2]\leftrightarrow H\left(e^{j\omega_1},e^{j\omega_2}\right)\cdot X\left(e^{j\omega_1},e^{j\omega_2}\right). \quad (1.292)$$

Свертке последовательностей $h[n_1, n_2]$ и $x[n_1,n_2]$ соответствует умножение их преобразований Фурье.

Транспонирование:

$$x[n_2,n_1]\leftrightarrow X\left(e^{j\omega_2},e^{j\omega_1}\right). \quad (1.293)$$

Для вещественной последовательности $x[n_1,n_2]$ справедливы следующие соотношения:

$$X\left(e^{j\omega_1}, e^{j\omega_2}\right) = X^*\left(e^{-j\omega_1}, e^{-j\omega_2}\right), \tag{1.294}$$

$$\mathrm{Re}\left[X\left(e^{j\omega_1}, e^{j\omega_2}\right)\right] = \mathrm{Re}\left[X\left(e^{-j\omega_1}, e^{-j\omega_2}\right)\right], \tag{1.295}$$

$$\mathrm{Im}\left[X\left(e^{j\omega_1}, e^{j\omega_2}\right)\right] = -\mathrm{Im}\left[X\left(e^{-j\omega_1}, e^{-j\omega_2}\right)\right]. \tag{1.296}$$

Вещественная часть преобразования Фурье обладает четной симметрией, мнимая часть – нечетной.

1.7.4 Двумерное дискретное преобразование Фурье

Рассмотрим двумерную периодическую последовательность $x[n_1,n_2]$ с периодом N_1 в одном измерении и N_2 – в другом, т.е.

$$x[n_1, n_2] = x[n_1 + N_1, n_2 + N_2] \ . \tag{1.297}$$

Для последовательности, полностью определенной в опорной области $0 \le n_1 \le N_1 - 1$ и $0 \le n_2 \le N_2 - 1$, двумерный дискретный ряд Фурье можно записать следующим образом:

$$x[n_1, n_2] = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X[k_1, k_2] \cdot e^{j\frac{2\pi}{N_1} n_1 k_1 + j\frac{2\pi}{N_2} n_2 k_2}, \tag{1.298}$$

где коэффициенты ряда $X[k_1,k_2]$ определяются с помощью выражения

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] \cdot e^{-j\frac{2\pi}{N_1} n_1 k_1 - j\frac{2\pi}{N_2} n_2 k_2}. \tag{1.299}$$

Соотношение (1.299) называют *двумерным дискретным преобразованием Фурье* (ДДПФ), а (1.298) – *двумерным обратным преобразованием Фурье* (ДОДПФ).

Двумерная последовательность $X[k_1,k_2]$ является периодической с периодами N_1, N_2 в соответствующих измерениях. Для вычисления всех значений последовательности $X[k_1,k_2]$ требуется выполнить $(N_1N_2)^2$ комплексных операций сложения и умножения. С целью сокращения количества операций можно применить одномерные алгоритмы БПФ.

Перепишем (1.299) в виде

$$X[k_1,k_2]=\sum_{n_1=0}^{N_1-1} e^{-j\frac{2\pi}{N_1}n_1k_1}\left[\sum_{n_2=0}^{N_2-1} x[n_1,n_2]\cdot e^{-j\frac{2\pi}{N_2}n_2k_2}\right]. \quad (1.300)$$

Сумма в квадратных скобках представляет одномерное дискретное преобразование Фурье одного столбца массива $x[n_1,n_2]$. Обозначим сумму в квадратных скобках через $X_p[n_1,k_2]$, тогда

$$X[k_1,k_2]=\sum_{n_1=0}^{N_1-1} X_p[n_1,k_2]\cdot e^{-j\frac{2\pi}{N_1}n_1k_1}. \quad (1.301)$$

Соотношение (1.301) также представляет одномерное дискретное преобразование Фурье, но в другом измерении. Таким образом, для определения значений $X[k_1,k_2]$ необходимо вычислить N_1 одномерных ДПФ по столбцам массива исходных данных, а затем N_2 одномерных ДПФ по строкам промежуточного массива $X_p[n_1,k_2]$. Применив для вычисления каждого одномерного ДПФ алгоритм БПФ, можно сократить число операций до $N_1N_2\log_2 N_1N_2$.

Большинство свойств одномерного дискретного преобразования Фурье легко обобщаются на случай ДДПФ и соответствуют свойствам двумерного дискретно-непрерывного преобразования Фурье. При этом необходимо учитывать периодичность $x[n_1,n_2]$. В частности, для ДДПФ справедлива теорема о циклической свертке. Если $X[k_1,k_2]$ и $H[k_1,k_2]$ являются ДДПФ последовательностей $x[n_1, n_2]$ и $h[n_1, n_2]$ и

$$0 \le k_1, n_1 \le N_1 - 1$$
$$0 \le k_2, n_2 \le N_2 - 1 \quad ,$$

то

$$Y[k_1,k_2] = X[k_1,k_2]\cdot H[k_1,k_2] \quad (1.302)$$

соответствует циклической двумерной свертке, задаваемой формулой

$$y[n_1,n_2]=\sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} x[m_1,m_2]\cdot h[n_1-m_1,n_2-m_2]. \quad (1.303)$$

Здесь последовательность $y[n_1,n_2]$ будет периодической с периодами N_1 и N_2. Отсюда следует, что двумерная циклическая свертка, аналогично одномерному случаю, может быть вычислена с использованием (1.302) по методу быстрой свертки на основе алгоритма БПФ.

Линейную свертку можно получить на основе (1.302), дополнив массивы $x[n_1,n_2]$ и $h[n_1,n_2]$ необходимым числом нулей.

Глава 2
ЦИФРОВОЙ СПЕКТРАЛЬНЫЙ АНАЛИЗ

2.1 Назначение спектрального анализа

Спектральный анализ - это метод обработки сигналов, который позволяет выявить частотный состав сигнала. Поскольку анализируемые сигналы во многих случаях имеют случайный характер, то важную роль в спектральном анализе играют методы математической статистики. Частотный состав сигналов определяют путем вычисления оценок спектральной плотности мощности (СПМ). Задачами вычисления СПМ являются *обнаружение* гармонических составляющих в анализируемом сигнале и *оценивание* их параметров. Для решения указанных задач требуется соответственно высокая разрешающая способность по частоте и высокая статистическая точность оценивания параметров. Эти два требования противоречивы. Аргументы в пользу выбора высокого разрешения или высокой точности оценки СПМ зависят от того, что интересует исследователя: устойчивые оценки в пределах всего диапазона частот или высокая степень обнаруживаемости периодических составляющих.

Пусть последовательность $x[n]$ представляет собой дискретный случайный процесс. Обозначим через E действительное значение некоторого параметра случайного процесса $x[n]$. В результате обработки случайного процесса вычисляется не действительное значение E параметра, а его оценка, обозначаемая $\hat{E}$. Оценки являются случайными величинами и характеризуются следующими свойствами.

Смещением оценки называют математическое ожидание отклонения оценки $\hat{E}$ от действительного значения E [2]:

$$b\{\hat{E}\} = M\{\hat{E} - E\}. \qquad (2.1)$$

Оценка $\hat{E}$ является смещенной, если $b\{\hat{E}\} \neq 0$.

Квадрат *среднеквадратической ошибки* оценивания определяется с помощью формулы

$$\overline{\varepsilon^2} = M\{(\hat{E} - E)^2\} = D[\hat{E}] + b^2[E] \ . \qquad (2.2)$$

Оценка, которая сходится по вероятности к оцениваемой характеристике при бесконечном увеличении длины N анализируемой последовательности, называется *состоятельной*, т.е.

$$p(|\hat{E} - E)| \to 1 \qquad при \quad N \to \infty \ , \qquad (2.3)$$

где p – вероятность того, что модуль отклонения оценки от действительного значения оцениваемого параметра будет меньше сколь угодно малого положительного числа ε_0. Для выполнения этого требования достаточно, чтобы среднеквадратическая ошибка стремилась к нулю при $N \to \infty$. В соответствии с (2.2) оценка будет состоятельной, если ее дисперсия стремится к нулю при $N \to \infty$ и она является несмещенной. При оценивании параметров случайных последовательностей желательно использовать состоятельные оценки.

Все методы цифрового спектрального анализа можно разделить на две группы [11]: *классические методы*, базирующиеся на использовании преобразований Фурье, и *методы параметрического моделирования*, в которых выбирается некоторая линейная модель формирующего фильтра и оцениваются его параметры. К первой группе относят корреляционный и периодограммные методы. Ко второй группе относят методы оценивания СПМ на основе авторегрессии скользящего среднего и др. Рассмотрим некоторые из указанных методов спектрального анализа.

2.2 Периодограммный метод

Периодограммный метод обеспечивает вычисление оценки СПМ непосредственно по числовой последовательности $x[nT_0]$, формируемой путем дискретизации стационарного эргодического случайного процесса $x(t)$. Периодограммная оценка СПМ равна [11]:

$$\widetilde{S}_x(\omega) = \frac{T_0}{N}\left|\sum_{n=0}^{N-1} x[nT_0]e^{-j\omega nT_0}\right|^2 . \qquad (2.4)$$

Выражение (2.4) соответствует рассмотренной ранее возможности вычисления СПМ с помощью преобразования Фурье непосредственно по реализации исходного сигнала (1.141).

Вычисленная с помощью (2.4) оценка СПМ является несостоятельной, т.е. с увеличением N она не улучшается. Для получения состоятельной оценки ее необходимо сглаживать. Кроме этого, при выполнении преобразования Фурье последовательности $x[nT_0]$ конечной длины N происходит «размывание» спектра, которое также оказывает влияние на состоятельность оценки СПМ.

Ограничение последовательности $x[nT_0]$ конечным числом значений равносильно умножению исходной бесконечной последовательности $x_0[nT_0]$ на другую последовательность

$$w[nT_0] = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & \textit{в остальных случаях}, \end{cases} \tag{2.5}$$

которую называют прямоугольным окном. Тогда можно записать:

$$x[nT_0] = x_0[nT_0]w[nT_0]. \tag{2.6}$$

Преобразование Фурье последовательности $x[nT_0]$ равно свертке преобразований Фурье последовательности $x_0[nT_0]$ и прямоугольного окна $w[nT_0]$:

$$X(\omega) = X_0(\omega) * W(\omega), \tag{2.7}$$

где

$$W(\omega) = T_0 e^{-j\omega T_0 \frac{(N-1)}{2}} \frac{\sin(N\omega T_0/2)}{\sin(\omega T_0/2)}. \tag{2.8}$$

Функция $W(\omega)$, называемая ядром Дирихле (рис.2.2,а), искажает преобразование Фурье $X_0(\omega)$. Поэтому $X(\omega)$ является искаженной копией преобразования Фурье $X_0(\omega)$ бесконечной последовательности $x_0[nT_0]$. Влияние прямоугольного окна на определение спектра дискретной косинусоиды с частотой ω_0 показано на рис. 2.1.

Из рис.2.1 видно, что острые спектральные пики последовательности $x_0[nT_0]$ расширились за счет их свертки со спектром окна $w[n]$, а также появились дополнительные составляющие, т.е. происходит утечка энергии спектральных пиков в боковые лепестки [2,11]. Боковые лепестки изменяют (искажают) амплитуду соседних спектральных пиков, что приводит к смещению оценки и к маскировке слабых спектральных составляющих.

С целью устранения указанного явления необходимо применять меры по локализации энергии у спектральных пиков. Для этого $x_0[nT_0]$ умножают на соответствующим образом подобранное окно $w[n]$, которое

позволяет снизить уровень боковых лепестков по сравнению со случаем прямоугольного окна. Однако это приводит к расширению основного лепестка, что ухудшает разрешение спектральных составляющих. Поэтому должен существовать компромисс между шириной основного лепестка и уровнем боковых лепестков.

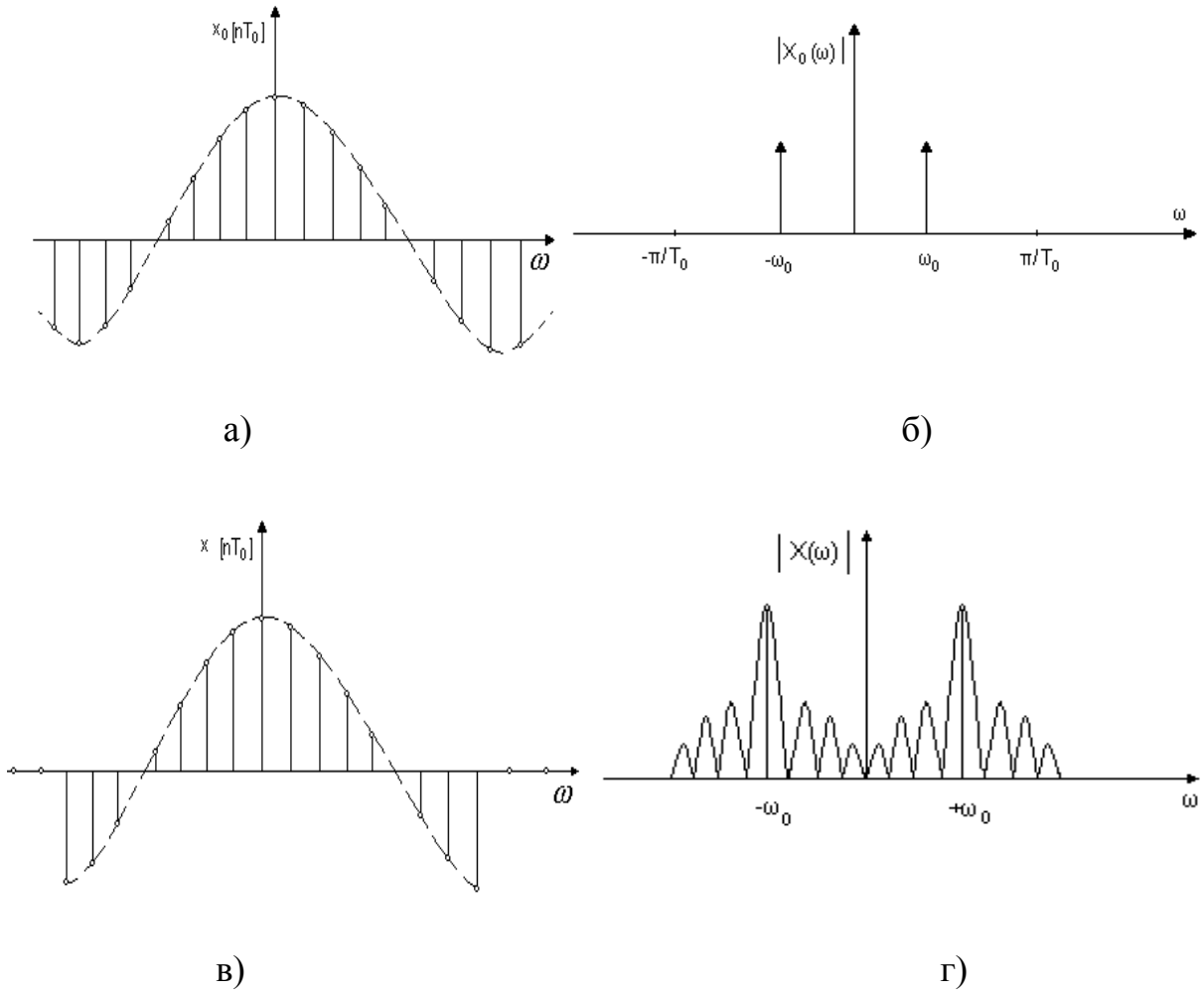


Рис. 2.1. Размывание спектра: а) исходная бесконечная последовательность; б) модуль преобразования Фурье от $x_0[nT_0]$; в) последовательность $x_0[nT_0]$, умноженная на прямоугольное окно; г) модуль преобразования Фурье последовательности $x[nT_0]$

Ниже приведены определения часто используемых оконных функций $w[n]$, где $-N/2 \le n \le N/2-1$ [2,4,6,11].

Треугольное окно (рис.2.2,б)

$$w[n] = 1 - \frac{|n|}{N} \quad . \tag{2.9}$$

Окно Ханна (косинус квадрат) (рис.2.2,в)

$$w[n] = \cos^2 \frac{\pi n}{N} = \frac{1}{2}\left(1 - \cos\frac{2\pi n}{N}\right). \qquad (2.10)$$

Окно Хемминга (рис.2.2,г)

$$w[n] = \alpha - (1-\alpha)\cdot\cos\left(\frac{2\pi n}{N}\right), \quad \text{обычно } \alpha=0{,}54. \qquad (2.11)$$

Окно Блэкмана (рис.2.2,д)

$$w[n] = 0{,}42 - 0{,}50Cos\left(\frac{2\pi n}{N}\right) + 0{,}08\cos\left(\frac{4\pi n}{N}\right). \qquad (2.12)$$

Окно Гаусса (рис.2.2,е)

$$w[n] = \exp\left[-\frac{1}{2}\left(\frac{\alpha n}{N/2}\right)^2\right], \quad \alpha=2{,}5. \qquad (2.13)$$

Окно Кайзера – Бесселя (рис.2.2,ж)

$$w[n] = \frac{I_0\left[\pi\alpha\sqrt{1-\left(\frac{n}{N/2}\right)^2}\right]}{I_0(\pi\alpha)}, \qquad (2.14)$$

где $I_0(x)$ – модифицированная функция Бесселя первого рода нулевого порядка; α - константа, определяющая компромисс между уровнем боковых лепестков и шириной основного лепестка (на рис.2.2,ж α=3,0). Модифицированная функция Бесселя может быть найдена с помощью ряда:

$$I_0(x) = 1 + \sum_{k=1}^{\infty}\left[\frac{(x/2)^k}{k!}\right]^2. \qquad (2.15)$$

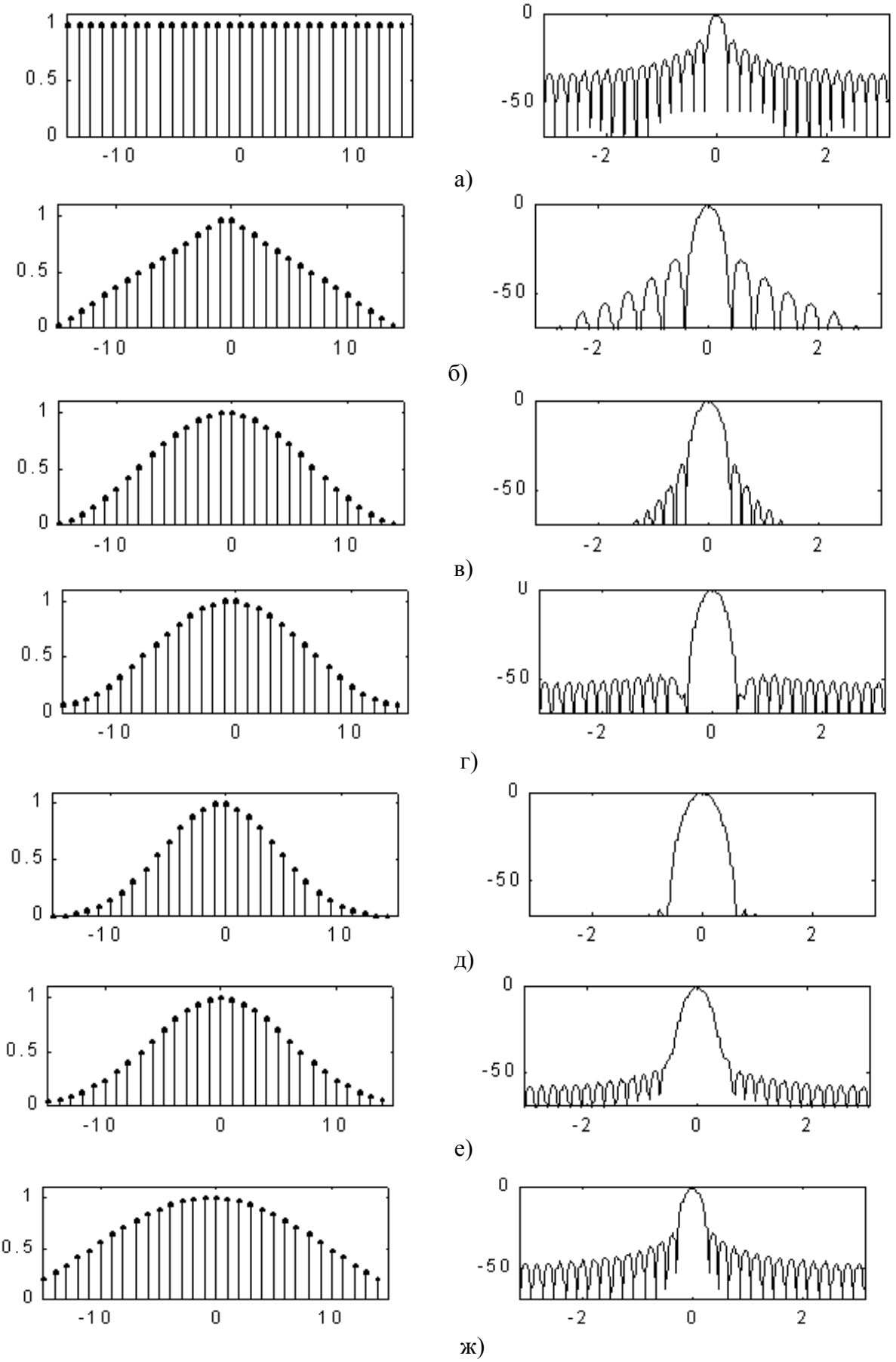


Рис.2.2 Оконные функции и логарифм модуля их ДНПФ

При выборе оконных функций используются следующие параметры: ширина основного лепестка, максимальный уровень боковых лепестков, скорость спадания уровня боковых лепестков. В табл. 2.1 приведены основные параметры окон, применяемых в спектральном анализе [6,11].

Таблица 2.1 - Параметры окон

Окно	Максимальный уровень боковых лепестков, дБ	Скорость спадания, дБ/октава	Ширина основного лепестка
Прямоугольное	-13,3	-6	0,89
Треугольное	-26,5	-12	1,28
Окно Ханна	-31,5	-18	1,44
Окно Хемминга	-43	-6	1,30
Окно Блэкмана	-58,2	-18	1,64
Окно Гаусса	-42	-6	1,33

Здесь ширина основного лепестка определена на уровне 3 дБ ниже его максимума и измерена в единицах разрешения преобразования Фурье, т.е. $2\pi/N$, где N – длина окна.

Для повышения состоятельности оценки (2.4) выполняют её сглаживание. Имеется несколько методов сглаживания: Даньелла, Бартлетта, Уэлча [11].

Метод Даньелла основан на осреднении значений СПМ в пределах смежных спектральных частот.

В соответствии с *методом Бартлетта* состоятельность оценки СПМ повышают усреднением оценок СПМ коротких реализаций, полученных из одной реализации длиной N отсчетов. Пусть дана реализация длиной N отсчетов. Она разбивается на n_s неперекрывающихся сегментов, длиной $N_s=N/n_s$ отсчетов. Для каждого сегмента по формуле (2.4) вычисляется выборочная оценка СПМ. Сглаженная оценка СПМ получается путем усреднения по всем n_s сегментам

$$\hat{S}_x(\omega) = \frac{1}{n_s}\sum_{i=1}^{n_s} \widetilde{S}_x(\omega) \quad . \tag{2.16}$$

Если последовательность $x[n]$ представляет нормальный стационарный эргодический процесс, то сглаженная оценка имеет дисперсию обратно пропорциональную числу сегментов n_s.

Спектральное разрешение оценки задается приближенным равенством

$$B \approx \Delta f = 1/(N_s T_0) . \qquad (2.17)$$

В *методе Уэлча* подход Бартлетта применяется к перекрывающимся сегментам исходной последовательности $x[n]$, и каждый сегмент взвешивается с помощью оконной функции для уменьшения смещения оценок из-за эффекта «просачивания» энергии в боковые лепестки. Цель перекрытия сегментов – увеличить число усредняемых участков при фиксированной длине последовательности и тем самым повысить точность оценок СПМ. Метод Уэлча – один из самых распространенных периодограммных методов [11].

Обозначим через ΔN величину сдвига между сегментами, которая должна удовлетворять условию $\Delta N \approx \tau_{\max}/T_0$, где $\tau_{\max}$ - максимальное время корреляции анализируемого процесса. При выполнении этого условия получим $P = \mathrm{int}[(N - N_s)/\Delta N + 1]$ слабо коррелированных сегментов. Отсчеты каждого сегмента взвешиваются окном $w[n]$:

$$x_p[n] = w[n] \cdot x[n + p\Delta N], \quad 0 \le n \le N_s - 1, \quad 0 \le p \le P - 1 . \qquad (2.18)$$

Выборочное значение СПМ сегмента p оценивается по формуле

$$\widetilde{S}_x{}^{(p)}(\omega) = \frac{1}{uN_s T_0}\left|X^{(p)}(\omega)\right|^2 , \qquad (2.19)$$

где

$$X^{(p)}(\omega) = T_0 \sum_{n=0}^{N_s - 1} x_p[n]\, e^{-j\omega nT_0} , \qquad (2.20)$$

$$u = T_0 \sum_{n=0}^{N_s - 1} w^2[n] . \qquad (2.21)$$

Сглаженная оценка периодограммы Уэлча вычисляется по формуле

$$\hat{S}_x(\omega) = \frac{1}{P}\sum_{p=0}^{P-1} S_x{}^{(p)}(\omega) . \qquad (2.22)$$

Введение перекрытия сегментов в методе Уэлча позволяет уменьшить изменчивость оценки СПМ. Так же как и в методе Бартлетта, дисперсия оценки СПМ по методу Уэлча обратно пропорциональна числу сегментов. Но благодаря большему числу сегментов, значение дисперсии будет меньше.

2.3 Спектральный анализ, основанный на параметрическом моделировании

В соответствии с методами параметрического линейного моделирования [4,11] случайная последовательность $x[n]$ может быть достаточно хорошо представлена (аппроксимирована) другой последовательностью $s[n]$

$$s[n] = -\sum_{k=1}^{P} a[k]s[n-k] + \sum_{k=0}^{N} b[k]u[n-k], \quad b[0]=1. \qquad (2.23)$$

Уравнение (2.23) соответствует уравнению линейного дискретного фильтра. Поэтому $s[n]$ обычно интерпретируют как процесс на выходе фильтра, на вход которого подается ненаблюдаемая последовательность $u[n]$. Обычно полагают, что $u[n]$ является белым шумом с нулевым средним значением и дисперсией D_u. Выражение (2.23) называют моделью *авторегрессии со скользящим средним* (АРСС – моделью). Здесь параметры $a[k]$ характеризуют авторегрессионную (АР) часть модели, а параметры $b[k]$ – соответствуют скользящему среднему (СС).

Задача оценивания СПМ с помощью (2.23) состоит в определении параметров модели $a[k]$ и $b[k]$ таким образом, чтобы ошибка

$$e[n] = \left|x[n] - s[n]\right| \qquad (2.24)$$

была мала. Тогда СПМ последовательности $x[n]$ можно вычислить по формуле

$$S_x(\omega) = D_u \left[\frac{B(\omega)}{A(\omega)}\right]^2, \qquad (2.25)$$

где $B(\omega) = \left|B(z)\right|$, $A(\omega) = \left|A(z)\right|$ при $z = \exp(j\omega T_0)$.

Если в уравнении (2.23) положить все АР-коэффициенты равными нулю, то получим модель

$$s[n] = \sum_{k=0}^{N} b[k]u[n-k], \qquad (2.26)$$

которую называют *моделью скользящего среднего* (СС-модель).

Полагая, что в уравнении (2.23) равны нулю все коэффициенты $b[k]$ (за исключением $b[0]=1$), получим *авторегрессионную модель* (АР-модель)

$$s[n] = -\sum_{k=1}^{P} a[k]s[n-k] + u[n] \ . \qquad (2.27)$$

СС-модель и АР-модель могут быть использованы для вычисления СПМ последовательности $x[n]$, если в (2.25) соответственно положить $A(\omega)$=1 или $B(\omega)$=1. Все три модели могут применяться в задачах спектрального анализа, но схемы вычислений с использованием АР-моделей являются наиболее простыми. Поэтому ниже рассмотрим определение только параметров АР-модели.

Умножим обе части уравнения (2.27) на $s[n-m]$ и определим математическое ожидание обеих частей. При этом получим

$$M\{s[n]s[n-m]\} = -\sum_{k=1}^{P} a[k]M\{s[n-m]s[n-k]\} + M\{u[n]s[n-m]\}. \qquad (2.28)$$

Поскольку

$$M\{s[n]s[n-m]\} = r_s[m] \qquad (2.29)$$

и

$$M\{s[n-k]s[n-m]\} = r_s[m-k], \qquad (2.30)$$

где $r_s[m]$ – автокорреляционная функция последовательности $s[n]$, то

$$r_s[m] = -\sum_{k=1}^{P} a[k]r_s[m-k] + r_{us}[m], \qquad (2.31)$$

где $r_{us}[m]$ – взаимная корреляционная функция между входом и выходом АР-модели (2.23). Поскольку $u[n]$ – белый шум, то

$$r_{us}[m] = \begin{cases} 0, & m > 0 \\ D_u, & m = 0 \end{cases} \ . \qquad (2.32)$$

Отсюда получаем уравнение, связывающее автокорреляционную последовательность с параметрами АР-модели:

$$r_s[m] = \begin{cases} -\sum_{k=1}^{P} a[k]r_s[m-k] \ , & m > 0, \\ -\sum_{k=1}^{P} a[k]r_s[-k] + D_u \ , & m = 0. \end{cases} \qquad (2.33)$$

Записав (2.33) для различных значений индексов m ($0 \le m \le P$), получим систему линейных алгебраических уравнений, которую можно представить в матричной форме:

$$\begin{bmatrix} r_s[0] & r_s[-1] & \cdots & r_s[-p] \\ r_s[1] & r_s[0] & \cdots & r_s[-p+1] \\ \vdots & \vdots & & \vdots \\ r_s[p] & r_s[p-1] & \cdots & r_s[0] \end{bmatrix} \begin{bmatrix} 1 \\ a[1] \\ \vdots \\ a[p] \end{bmatrix} = \begin{bmatrix} D_u \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \qquad (2.34)$$

Уравнения, образующие систему (2.34), называются *нормальными уравнениями Юла – Уолкера* для АР-модели [4,11]. Если известна автокорреляционная последовательность $r_s[m]$, то АР-параметры могут быть найдены из (2.34).

Матрица, входящая в (2.34), является симметричной и тёплицевой (матрица называется тёплицевой, если все ее элементы, расположенные на каждой диагонали, равны). Это позволяет использовать для решения системы (2.34) эффективный алгоритм, называемый алгоритмом Левинсона [11]. Отметим, что решение системы (2.34) требует предварительного вычисления автокорреляционной последовательности $r_s[m]$.

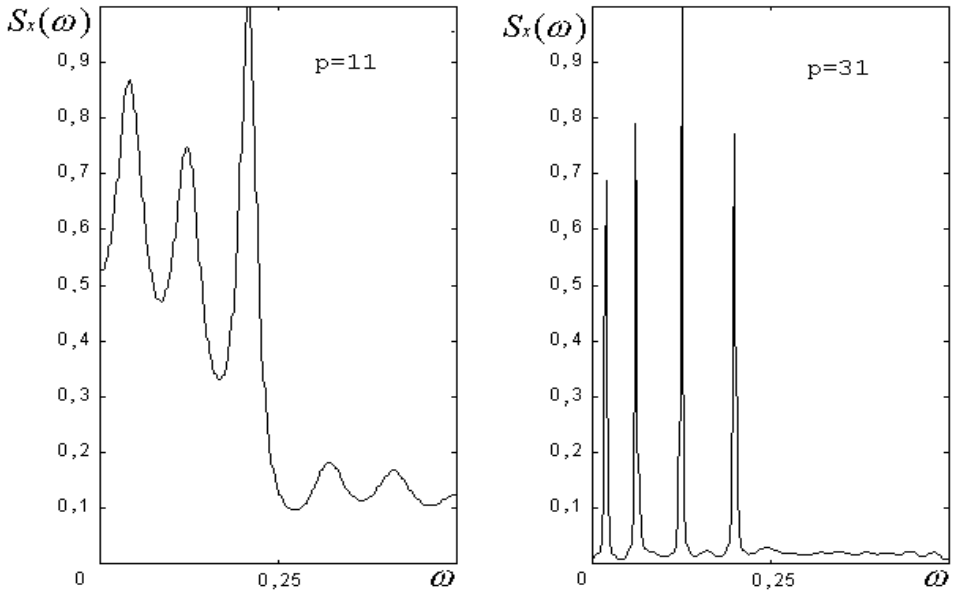


Рис.2.3. СПМ сигнала, состоящего из 4-х спектральных составляющих

Одна из причин применения параметрических моделей – возможность построения оценок СПМ в виде функциональных зависимостей, а также возможность получения более гладких оценок. Однако этими оценками необходимо пользоваться весьма осторожно, так как их свойства в значительной степени определяются адекватностью выбранной

параметрической модели (2.27) анализируемому процессу. В качестве примера на рис.2.3 изображены графики СПМ сигнала, содержащего четыре спектральные составляющие. Вычисление СПМ выполнялось на основе решения системы (2.34).

Из рис.2.3 следует, что в случае p=11 АР-модель является неадекватной исследуемому сигналу, так как по графику СПМ нельзя правильно восстановить структуру сигнала. Все четыре составляющие обнаруживаются, если p=31. Правильный выбор порядка модели требует дополнительной информации о характере исследуемого сигнала. В дальнейшем рассмотрим применение параметрических моделей при анализе речи.

Глава 3
СИНТЕЗ ЦИФРОВЫХ ФИЛЬТРОВ

3.1 Синтез нерекурсивных цифровых фильтров

3.1.1 Основные виды и свойства нерекурсивных фильтров

Синтез нерекурсивных фильтров (НРФ) состоит в определении импульсной характеристики фильтра $h[k]$, обеспечивающей выполнение требований, предъявляемых к фильтру. Обычно эти требования задаются в виде желаемых частотных характеристик. Импульсная характеристика $h[k]$ НРФ является конечной и определяется для значений k, лежащих в диапазоне $[0, N-1]$. Синтез НРФ предусматривает выполнение следующих основных этапов:

1) определение числа коэффициентов фильтра N;

2) выбор метода синтеза, минимизирующего отклонение между желаемыми и реальными характеристиками фильтров;

3) вычисление коэффициентов НРФ в соответствии с выбранным методом. Напомним, что для НРФ коэффициенты фильтра $b[k]$ равны отсчетам импульсной характеристики $h[k]$;

4) анализ результатов синтеза и принятие решений относительно свойств синтезированного фильтра. Если результаты синтеза не удовлетворяют заданным требованиям, то выполняют повторный синтез при других значениях N или выбирают другой метод синтеза.

Основным преимуществом НРФ по сравнению с РФ является возможность получения линейной ФЧХ. Поэтому выбор между рекурсивными и нерекурсивными фильтрами выполняют, ориентируясь на указанное свойство НРФ. Существует четыре вида НРФ с линейной ФЧХ [4,10,16]:

1) N - нечетное и коэффициенты симметричны $b[k] = b[N-k-1]$;

2) N - четное и коэффициенты симметричны $b[k] = b[N-k-1]$;

3) N - нечетное и коэффициенты антисимметричны $b[k] = -b[N-k-1]$;

4) N - четное и коэффициенты антисимметричны $b[k] = -b[N-k-1]$.

Фильтры с четными значениями N вносят в выходной сигнал задержку, равную половине периода дискретизации. Поэтому ниже будут

рассмотрены примеры синтеза НРФ первого и третьего видов с нечетными значениями N . Приведенные результаты легко распространяются на случай НРФ с четными N .

Определим частотную характеристику НРФ первого вида и покажем, что он обладает линейной ФЧХ. Рассмотрим сначала некаузальный НРФ с симметричной импульсной характеристикой

$$h_{нк}[-l] = h_{нк}[l] \ , \ l = 0,1,...,(N-1)/2, \ N \text{ - нечетное .} \qquad (3.1)$$

Частотная характеристика такого фильтра запишется в виде

$$H(e^{j\omega}) = h_{нк}[0] + \sum_{l=1}^{(N-1)/2} (h_{нк}[l]e^{-j\omega l} + h_{нк}[l]e^{j\omega l}) =$$

$$= h_{нк}[0] + \sum_{l=1}^{(N-1)/2} 2h_{нк}[l]\cos(l\omega) = \sum_{l=0}^{(N-1)/2} c[l]\cos(l\omega) \ , \qquad (3.2)$$

где $c[0] = h_{нк}[0]$; $c[l] = 2h_{нк}[l]$. Таким образом, частотная характеристика некаузального НРФ является действительной функцией и характеризуется нулевым сдвигом фазы.

Импульсная характеристика каузального НРФ первого вида может быть получена из (3.1) путем задержки на $(N-1)/2$ тактов. В частотной области эта задержка соответствует сдвигу фазы, определяемому множителем $e^{-j\omega(N-1)/2}$. Поэтому частотная характеристика НРФ первого вида запишется в виде

$$H_1(e^{j\omega}) = e^{-j\omega(N-1)/2} \sum_{l=0}^{(N-1)/2} c[l]\cos(l\omega) \ . \qquad (3.3)$$

Множитель $e^{-j\omega(N-1)/2}$ обеспечивает линейную зависимость фазы от частоты.

Для фильтров третьего вида с антисимметричной импульсной характеристикой выражение, определяющее частотную характеристику, аналогично (3.3) , но вместо $\cos(l\omega)$ необходимо подставить $\sin(l\omega)$. Фильтры с антисимметричными коэффициентами используются для построения преобразователей Гильберта. Для фильтров второго и четвертого видов в выражение для частотной характеристики необходимо дополнительно внести задержку на половину такта и вместо функций $\cos(l\omega)$ или $\sin(l\omega)$ использовать соответственно функции $\cos(l+0{,}5)\omega$ или $\sin(l+0{,}5)\omega$.

3.1.2 Синтез НРФ с использованием окон

Поскольку коэффициенты $b[k]$ НРФ равны отсчетам его импульсной характеристики $h[k]$ и частотная характеристика любого цифрового фильтра является периодической функцией, то коэффициенты НРФ можно вычислять разложением в ряд Фурье заданной (желаемой) частотной характеристики [4,21].

Рассмотрим сначала применение указанной возможности для синтеза нерекурсивных ФНЧ, а затем распространим полученные результаты на синтез нерекурсивных ФВЧ и нерекурсивных ПФ.

Заданная частотная характеристика идеального цифрового ФНЧ определяется соотношением

$$H_з(\omega) = \begin{cases} 1, & |\omega| < \omega_c, \\ 0, & \omega_c < |\omega| < \pi. \end{cases} \qquad (3.4)$$

Используя разложение в ряд Фурье частотной характеристики (3.4) , определим импульсную характеристику НРФ

$$h_з[k] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_з(\omega) e^{j\omega k} d\omega \quad . \qquad (3.5)$$

Учитывая , что $H_з(\omega)$ является четной функцией частоты перепишем (3.5) в виде

$$h_з[k] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_з(\omega)\cos(\omega k) d\omega \quad . \qquad (3.6)$$

Для ФНЧ после подстановки (3.4) в (3.6) и выполнения интегрирования получаем

$$h_з[k] = \sin(\omega_c k)/(\pi k) \quad . \qquad (3.7)$$

Импульсная характеристика (3.7) определена при любых целых значениях k и является бесконечной. Физически реализуемый НРФ обладает конечной импульсной характеристикой. Поэтому необходимо ограничить значения k. Пусть $|k| \le K$, где $K = (N-1)/2$ для фильтров с нечетными значениями N и $K = N/2$ для фильтров с четными значениями N. Тогда конечная импульсная характеристика, соответствующая (3.7), запишется в виде

$$h_к[k] = \begin{cases} h_з[k], & -K \le k \le K, \\ 0, & при \quad других \quad k. \end{cases} \qquad (3.8)$$

Импульсная характеристика (3.8) является некаузальной, так как имеет ненулевые значения при $k < 0$. Чтобы получить каузальную характеристику необходимо задержать $h_к[k]$ на K тактов, т.е.

$$h[k] = h_к[k - K] \ , \qquad 0 \le k \le N - 1 \ . \tag{3.9}$$

Синтезированный таким образом НРФ будет иметь значительные пульсации АЧХ в районе частоты среза (рис.3.1). Указанные пульсации представляют эффект Гиббса, проявляющийся в вблизи точек разрыва заданной частотной характеристики. Эффект Гиббса обусловлен ограничением длительности бесконечной импульсной характеристики $h_з[k]$.

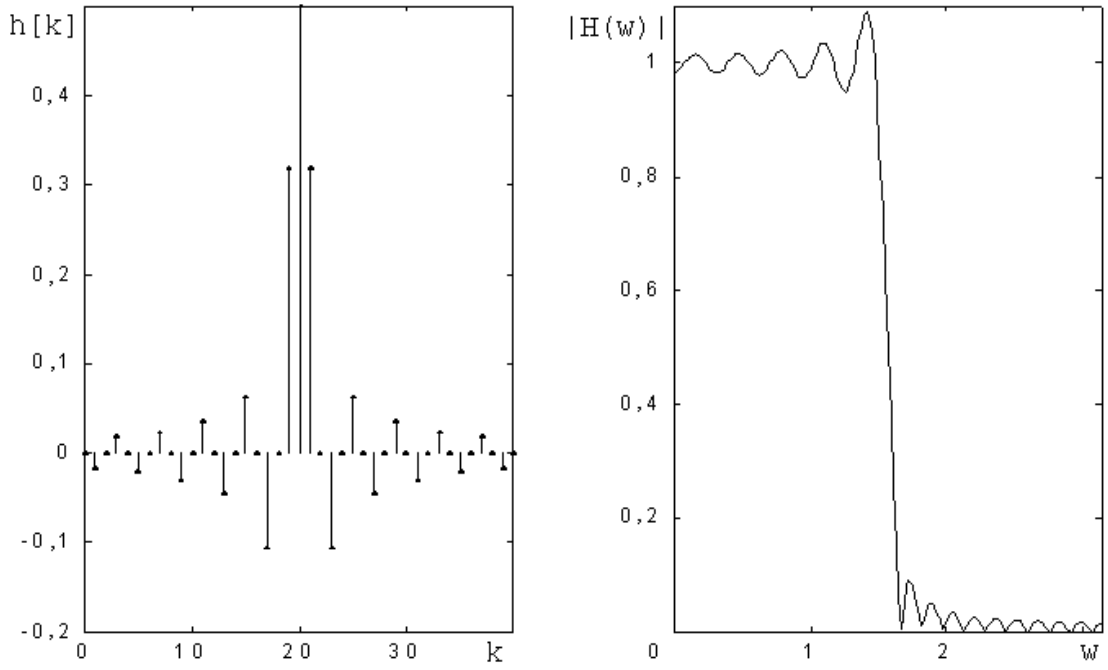


Рис.3.1. Импульсная характеристика (3.9) и АЧХ ФНЧ при *K*=20

Конечная импульсная характеристика $h[k]$, получаемая из $h_з[k]$, может быть также представлена в виде

$$h[k] = h_з[k - K]w[k] \ , \tag{3.10}$$

где $w[k]$ - прямоугольное весовое окно длиной N отсчетов. Умножение заданной импульсной характеристики на прямоугольное окно приводит к искажению частотной характеристики синтезируемого фильтра. Частотная характеристика, соответствующая (3.10), будет равна свертке заданной частотной характеристики с частотной характеристикой прямоугольного окна, описываемой выражением (2.8). Таким образом, эффект Гиббса связан с пульсирующим поведением частотной характеристики прямоугольного весового окна.

Для уменьшения отрицательного влияния эффекта Гиббса применяют весовые окна, отличные от прямоугольного. Здесь имеет место аналогия со спектральным анализом, когда для уменьшения утечки энергии

в боковые лепестки применяют оконные функции (2.9)-(2.14). С целью обеспечения линейности ФЧХ указанные оконные функции должны быть симметричными $w[k] = w[N-k-1]$ и определены для значений k, лежащих в диапазоне $0 \le k \le N-1$.

Пример 3.1

Определим коэффициенты нерекурсивного ФНЧ 41-го порядка с частотой среза $\omega_c = 0{,}5\pi$. Для уменьшения влияния эффекта Гиббса будем использовать окно Хэмминга. Используя (3.10) и (3.7) , получаем выражение для импульсной характеристики требуемого нерекурсивного ФНЧ в виде

$$h[k] = h_з[k-K] \cdot w[k] = \frac{\sin\omega_c(k-K)}{\pi \cdot (k-K)} \cdot [0{,}54 - 0{,}46\cos(\frac{2\pi k}{N-1})],$$

где $k = 0,1,..., N-1$ и $K = (N-1)/2$. После подстановки конкретных значений N и K получим выражение, определяющее коэффициенты фильтра

$$b[k] = h[k] = \frac{\sin\omega_c(k-20)}{\pi(k-20)}[0{,}54 - 0{,}46\cos(\frac{2\pi k}{40})]$$

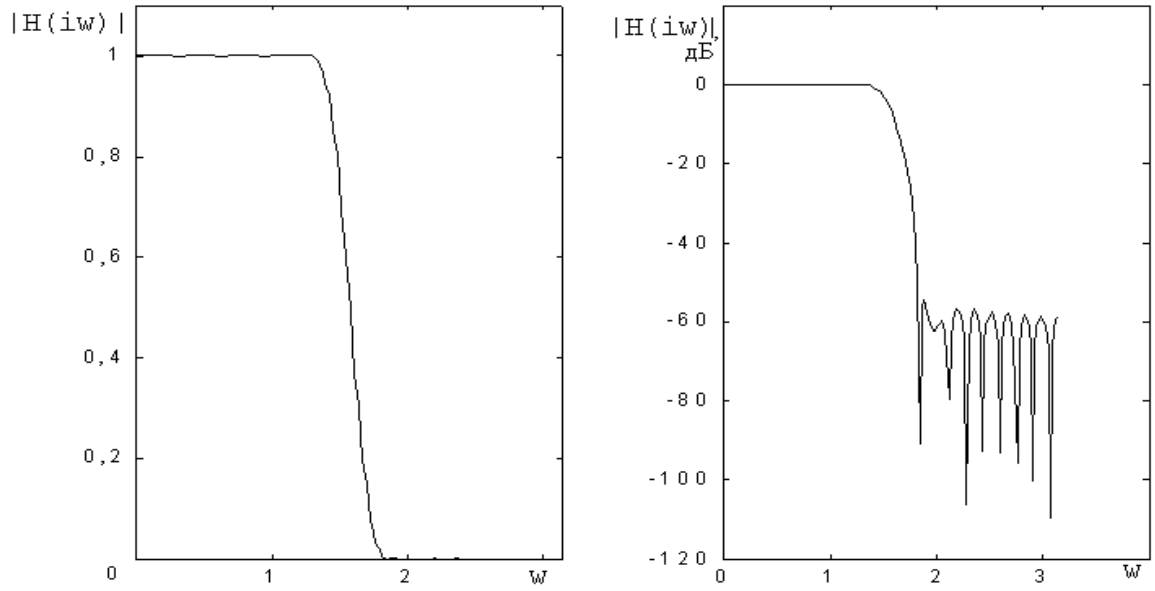


Рис.3.2. АЧХ ФНЧ, синтезированного с помощью окна Хемминга

АЧХ данного фильтра, вычисленная с помощью формулы (3.3), изображена на рис.3.2. На этом же рисунке изображена АЧХ фильтра в логарифмическом масштабе. Если выбрать граничную частоту пропускания $\omega_1 = 1{,}27$ рад. и граничную частоту задержания $\omega_2 = 1{,}87$ рад. (симметрично относительно частоты среза), то неравномерность в полосе пропускания R и ослабление в полосе задерживания A будут соответственно равны: 0,016дБ и –56,24 дБ.

Из сравнения рис.3.1 и рис.3.2 видно, что применение окна Хемминга позволило значительно уменьшить уровень пульсаций.

Синтез НРФ верхних частот или полосовых НРФ выполняется аналогично. Частотная характеристика идеального ФВЧ описывается соотношением

$$H_з^{ФВЧ}(\omega)=\begin{cases}0, & |\omega|<\omega_c,\\ 1, & \omega_c<|\omega|<\pi.\end{cases} \qquad (3.11)$$

Данная характеристика является обратной по отношению к частотной характеристике идеального ФНЧ $H_з^{ФНЧ}(\omega)$ и может быть представлена в виде

$$H_з^{ФВЧ}(\omega)=1-H_з^{ФНЧ}(\omega). \qquad (3.12)$$

Выражению (3.12) соответствует следующее соотношение импульсных характеристик ФВЧ и ФНЧ

$$h_з^{ФВЧ}[k]=\delta[k]-h_з^{ФНЧ}[k], \qquad (3.13)$$

где $\delta[k]$ - единичная импульсная функция. Подставив в (3.13) выражение (3.7), получим

$$h_з^{ФВЧ}[k]=\delta[k]-\sin(\omega_c k)/(\pi k). \qquad (3.14)$$

Частотная характеристика идеального полосно-пропускающего фильтра может быть представлена в виде разности частотных характеристик двух идеальных ФНЧ с различными частотами среза ω_1 и ω_2 ($\omega_1>\omega_2$)

$$H_з^{ПФ}(\omega)=H_з^{ФНЧ1}(\omega)-H_з^{ФНЧ2}(\omega). \qquad (3.15)$$

Выполнив разложение в ряд Фурье частотной характеристики (3.15), получим выражение, определяющее импульсную характеристику полосового фильтра

$$h_з^{ПФ}[k]=h_з^{ФНЧ\,1}[k]-h_з^{ФНЧ\,2}[k]. \qquad (3.16)$$

С учетом (3.7) выражение (3.16) можно переписать в виде

$$h_з^{ПФ}[k]=\sin(\omega_1 k)/(\pi k)-\sin(\omega_2 k)/(\pi k). \qquad (3.17)$$

Для уменьшения влияния эффекта Гиббса при синтезе ФВЧ или ПФ требуется также выполнять умножение бесконечных импульсных характеристик (3.14) или (3.17) на конечное весовое окно (2.9)-(2.14).

Рассмотренный метод синтеза НРФ характеризуется следующими достоинствами: не требует сложных вычислений; точность аппроксимации повышается простым увеличением порядка фильтра; позволяет получать аналитические выражения для определения значений коэффициентов фильтров.

Основной недостаток метода состоит в сложности определения граничных частот и в неравномерном распределении ошибок аппроксимации по частотному диапазону.

3.1.3 Метод частотной выборки

Пусть $H_з(\omega)$ является заданной (желаемой) частотной характеристикой НРФ. Обозначим через A_k ее отсчеты (выборки) для различных частот $\omega_k = 2\pi k / N$, принадлежащих интервалу $[0,2\pi]$, т.е.

$$A_k = H_з(\omega_k), \quad \text{k=0,1,…,N-1.} \tag{3.18}$$

По отсчетам частотной характеристики, используя ОДПФ, можно определить импульсную характеристику синтезируемого НРФ (соответственно и коэффициенты $b[k]$)

$$h[n] = \frac{1}{N}\sum_{k=0}^{N-1} A_k \cdot e^{j2\pi kn/N}, \text{ n=0,1,…,N-1} \quad . \tag{3.19}$$

Нерекурсивный фильтр с импульсной характеристикой (3.19) будет иметь частотную характеристику, определяемую выражением

$$H[\omega] = \sum_{n=0}^{N-1} h[n] \cdot e^{-j\omega n} \quad . \tag{3.20}$$

Исследуем свойства (3.20). Подставив (3.19) в (3.20), получим

$$H[\omega] = \sum_{n=0}^{N-1}\left[\frac{1}{N}\sum_{k=0}^{N-1} A_k \cdot e^{j2\pi kn/N}\right] \cdot e^{-j\omega n} \quad . \tag{3.21}$$

Поменяем местами знаки сумм в выражении (3.21)

$$H[\omega] = \frac{1}{N}\sum_{k=0}^{N-1} A_k \sum_{n=0}^{N-1} e^{-j(\omega-\omega_k)n}, \tag{3.22}$$

где $\omega_k = 2\pi k / N$. Внутреннюю сумму в выражении (3.22) можно переписать в виде

$$\sum_{n=0}^{N-1} e^{-j(\omega-\omega_k)n} = e^{-j(N-1)(\omega-\omega_k)/2} \cdot \frac{\sin[(N/2)(\omega-\omega_k)]}{\sin[1/2(\omega-\omega_k)]} \quad . \qquad (3.23)$$

При $\omega = \omega_k$ значение выражения (3.23), являющегося ядром Дирихле, равно N и $H(\omega_k) = A_k$. Следовательно, частотная характеристика синтезируемого НРФ, коэффициенты которого вычисляются в соответствии с (3.19), в точках взятия частотных выборок будет иметь значения, совпадающие со значениями желаемой частотной характеристики, т.е. $H(\omega_k) = H_з(\omega_k)$. Однако между точками взятия выборок реальная частотная характеристика НРФ будет отклоняться от $H_з(\omega_k)$ (рис.3.3). АЧХ НРФ, синтезированного таким образом, будет подобна АЧХ фильтра, построенного с помощью прямоугольного окна методом разложения в ряд Фурье (рис.3.1) .

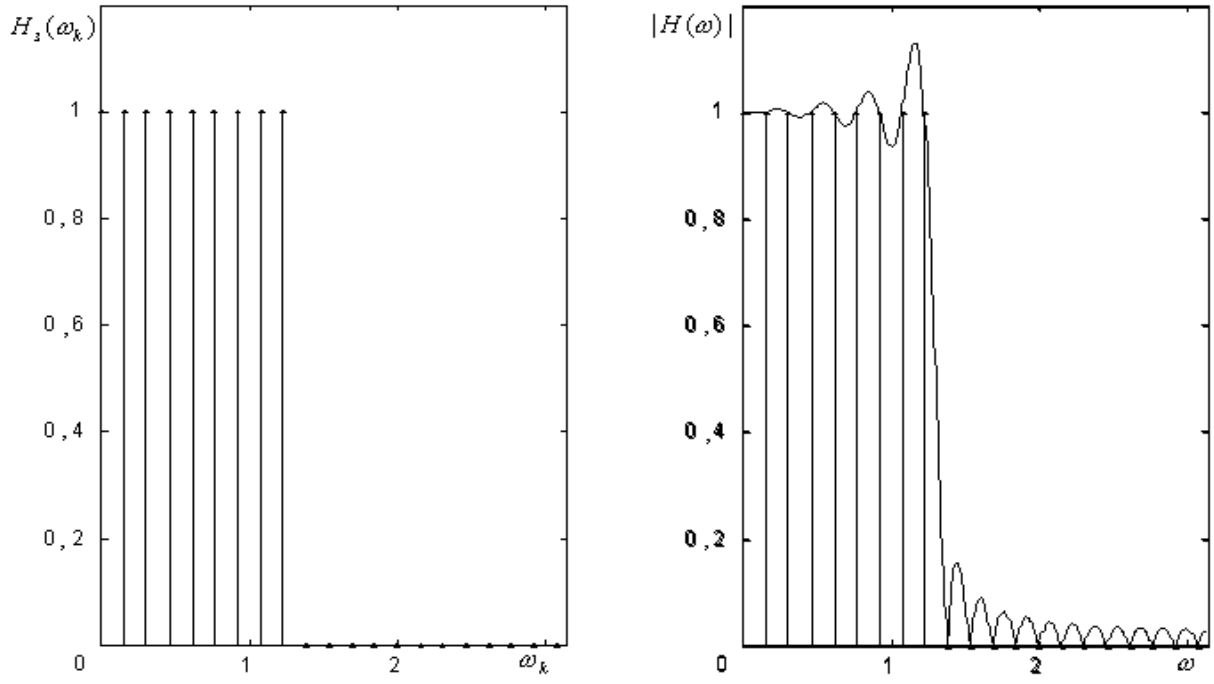


Рис.3.3. Заданная и реальная АЧХ НРФ

Пульсации АЧХ здесь также обусловлены тем, что заданная частотная характеристика имеет разрыв на частоте среза. Улучшить поведение АЧХ НРФ можно, если задать переходную полосу для $H_з(\omega_k)$, состоящую из нескольких дополнительных частотных выборок. Это позволяет перераспределить энергию пульсаций и уменьшить их уровень в полосе пропускания и полосе подавления. На рис. 3.4 показана АЧХ НРФ после добавления одной частотной выборки со значением 0,5 в переходной полосе.

Уровень пульсаций в полосах пропускания и подавления стал значительно ниже по сравнению с рис.3.3.

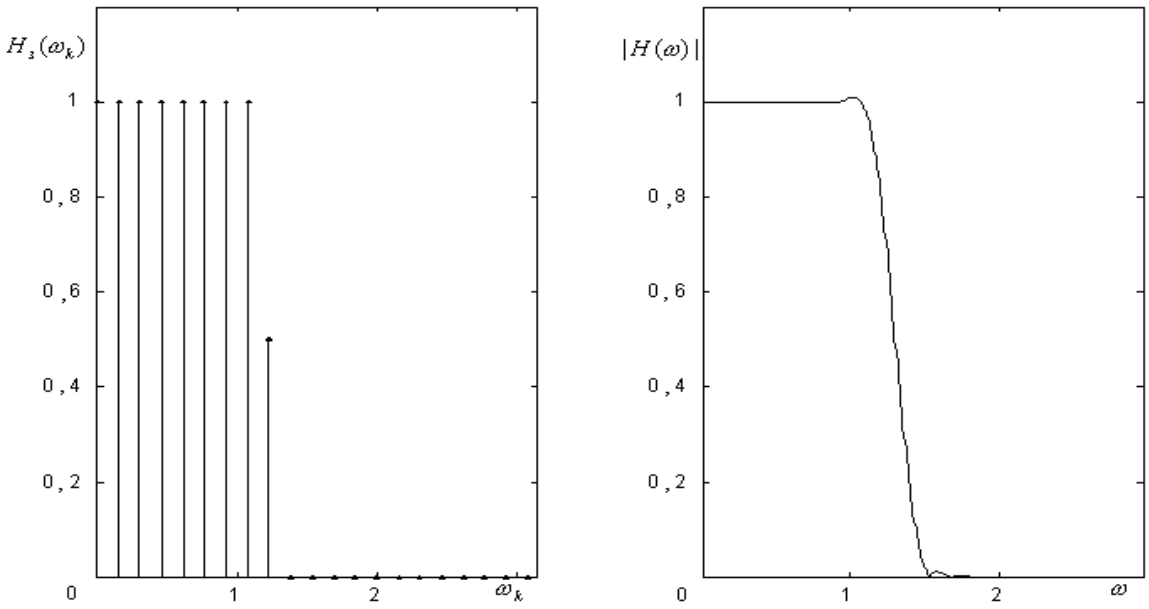


Рис. 3.4. АЧХ НРФ с добавленной выборкой в переходной полосе

Часто желаемую частотную характеристику задают в аналитической форме с помощью функций, дифференцируемых в переходной полосе. Например,

$$H_з(\omega) = \begin{cases} 1, & 0 \le \omega \le \omega_1, \\ 0{,}5 + 0{,}5\cos\left[\dfrac{\pi(\omega - \omega_k)}{\omega_2 - \omega_1}\right], & \omega_1 < \omega < \omega_2, \\ 0, & \omega_2 \le \omega \le \pi. \end{cases} \qquad (3.24)$$

В этом случае поведение частотной характеристики в переходной полосе определяется с помощью функции, представляющей приподнятый косинус. Данная функция имеет непрерывную производную, что обеспечивает получение гладких АЧХ НРФ.

Необходимо помнить, что коэффициенты фильтра, вычисленные с помощью (3.19), соответствуют некаузальной импульсной характеристике. Поэтому для физической реализации фильтра необходимо обеспечить соответствующую задержку импульсной характеристики, например на (*N*-1)/2 тактов, если *N*-нечетное.

Благодаря возможности оптимизации значений дополнительных отсчетов, вводимых в переходной полосе, метод частотной выборки позволяет получать фильтры с очень хорошими характеристиками. При этом поиск оптимальных значений выполняют с помощью ЭВМ на основе программ, использующих методы линейного программирования [4,10,16].

3.1.4. Метод наилучшей равномерной аппроксимации

Метод наилучшей равномерной аппроксимации основан на минимизации максимального отклонения частотной характеристики синтезируемого НРФ от заданной. Частотная характеристика синтезируемого фильтра представляется в виде функции двух аргументов $H(\omega,\vec{c})$, где ω-нормированная частота, а $\vec{c}$ - вектор параметров фильтра, связанных с его коэффициентами. Например,

$$H\ (\omega\ ,\vec{c}\ )\ =\ \sum_{l=0}^{L}\ c[\,l\,]\varphi_{\,l}\,(\omega\ )\ , \qquad (3.25)$$

где $\varphi_l(\omega)=\cos l\omega$ или $\varphi_l(\omega)=\sin l\omega$. В этом случае (3.25) соответствует частотной характеристике НРФ с симметричными или антисимметричными коэффициентами (см. 3.3). Задача синтеза состоит выборе вектора $\vec{c}$, минимизирующего выражение

$$\max\{\ W\ (\omega)\ |\ H_{\,з}\,(\omega)-H\,(\omega,\vec{c})\ |\}\quad\rightarrow\quad\min\ , \qquad (3.26)$$

где $W\ (\omega)$ - весовая функция, используемая для выделения отдельных диапазонов частот в ходе процесса оптимизации. Чем меньше должно быть отклонение $H\ (\omega,\vec{c})$ от заданной частотной характеристики $H_з(\omega)$ при определенной частоте $\omega=\omega_l$, тем больше должно быть значение $W(\omega_l)$. Обычно для задания весовой функции $W(\omega)$ весь интервал частот разбивают на поддиапазоны и в каждом из них задают допустимую ошибку аппроксимации ε_i

$$|\ H_{\ з}\,(\omega\ )\ -\ H\ (\omega\ ,\vec{c}\ )\ |\leq\ \varepsilon_{\ i}\ . \qquad (3.27)$$

Тогда для i-го поддиапазона $W(\omega)=a/\varepsilon_i$, где a- произвольная константа , общая для всех поддиапазонов.

Поиск вектора параметров фильтра, минимизирующего (3.26) выполняется численно с помощью алгоритма Ремеза. Суть алгоритма Ремеза состоит в последовательной модификации вектора параметров $\vec{c}$ с помощью ЭВМ. Коэффициенты вектора модифицируются до тех пор, пока не будет получено приближение с заданной точностью. При этом поиск вектора параметров выполняется с использованием обобщенной теоремы Чебышева, определяющей условия получения функции $H\ (\omega,\vec{c})$ наилучшего равномерного приближения. Функция $H\ (\omega,\vec{c})$ будет являться

функцией наилучшего равномерного приближения к функции $H_з(\omega)$ с весом $W(\omega)$, если текущая ошибка аппроксимации $\varepsilon(\omega,\vec{c})$ принимает равные друг другу наибольшие абсолютные значения , чередующиеся по знаку на L+2 частотах , т.е. $\omega_1,\omega_2,...,\omega_{L+2}$. Данные частоты называют частотами альтернанса.

Метод наилучшего равномерного приближения позволяет синтезировать НРФ заданного порядка N , для которого ошибка аппроксимации в полосах пропускания и подавления будет минимальна по сравнению с методами , рассмотренными выше.

Описание программ, выполняющих синтез НРФ в соответствии с указанным методом приведены в [4,10,16] .

Пример 3.2

Синтезируем методом наилучшей равномерной аппроксимации полосовой НРФ со следующей частотной характеристикой

$$H_з(\omega)=\begin{cases}0, & 0\le\omega\le 0{,}1\pi,\\ 1, & 0{,}2\pi\le\omega\le 0{,}4\pi,\\ 0, & 0{,}5\pi\le\omega\le\pi.\end{cases} \qquad (3.28)$$

При этом отклонения реальной АЧХ фильтра от (3.28) не должны превышать в полосе пропускания 0,01 и в полосах подавления 0,0005.

Для синтеза воспользуемся пакетом программ Matlab 5.2, в состав которого входит большой набор функций, реализующих различные методы синтеза цифровых фильтров. Рассматриваемый метод реализуется функцией remez(N,F0,A0,W), которая возвращает в качестве результатов синтеза вектор коэффициентов фильтра. Параметры функции: N-порядок фильтра; F0- вектор ,определяющий частотные полосы фильтра; A0- вектор, определяющий значения заданной частотной характеристики в частотных полосах; W-весовая функция. Для определения порядка фильтра можно воспользоваться функцией remezord(F,A,DEV,FS), где DEV допустимые отклонения реальной АЧХ в заданных полосах; FS- частота дискретизации (нормированное значение FS в пакете Matlab равно 2). Функция remezord в качестве результата возвращает вектор [N,F0,A0,W], элементы которого используются как входные параметры функции remez. Ниже приведен текст программы, выполняющей необходимый синтез фильтра:

```
%синтез НРФ методом наилучшей равномерной аппроксимации
f=[0.1 0.2 0.4 0.5];      %задание вектора частот
a=[0 1 0];                %задание АЧХ в полосах
dev=[0.0005 0.01 0.0005]; %задание отклонений в полосах
[n,f0,a0,w]=remezord(f,a,dev,2); %вычисление порядка
b=remez(n,f0,a0,w);        % определение коэф.фильтра
  %отображение результатов
subplot(1,2,1)
  k=1:1:max(size(b));
  stem(k,b,'.');
  [h ww]=freqz(b,1,200);
```

```
subplot(1,2,2)
plot(ww,20*log(abs(h))/2.3026)
```

Результаты вычислений представлены на рис.3.5. Порядок фильтра, вычисленный с помощью функции remezord , равен 54. Из рисунка видно, что метод обеспечивает равноволновые пульсации.

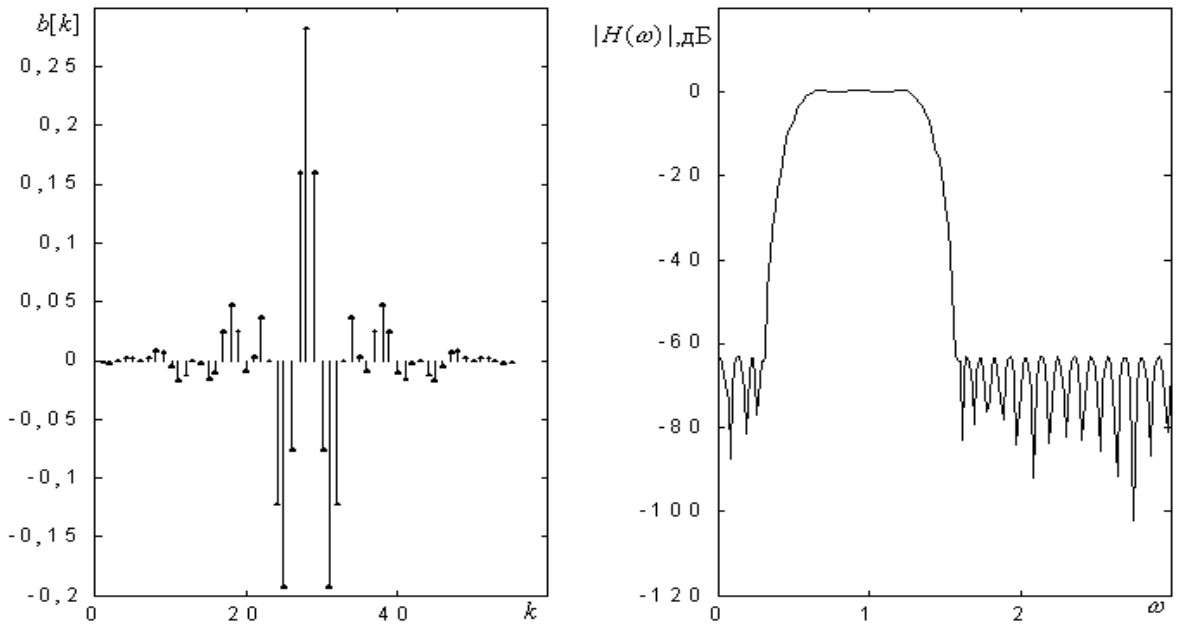


Рис.3.5. Импульсная характеристика и АЧХ полосового НРФ

3.2 Синтез рекурсивных цифровых фильтров

Решение задачи синтеза рекурсивных фильтров (РФ) сводится к нахождению коэффициентов $b[k]$ и $a[k]$ (1.211). Известны прямые и косвенные методы синтеза рекурсивных фильтров [4,10]. Прямые методы основаны на непосредственном определении параметров цифровых РФ по заданным временным или частотным характеристикам. Косвенные методы синтеза РФ основаны на дискретизации аналогового фильтра , удовлетворяющего заданным требованиям. При этом предполагается выполнение двух этапов. На первом этапе выбирают подходящий аналоговый фильтр-прототип, удовлетворяющий условиям обработки сигнала. Основные типы аналоговых фильтров были рассмотрены в § 1.3.3. На втором этапе осуществляется переход от аналогового фильтра-прототипа к цифровому фильтру. Наиболее распространенными методами дискретизации аналоговых фильтров являются следующие методы: инвариантности импульсной характеристики; согласованного z-преобразования; билинейного преобразования; отображения дифференциалов [4]. В настоящем параграфе рассмотрим один из наиболее простых и удобных косвенных методов синтеза РФ – метод билинейного преобразования.

В соответствии с методом билинейного преобразования непрерывная комплексная переменная p передаточной функции аналогового фильтра-прототипа $H(p)$ заменяется на комплексную переменную $z = e^{j\omega}$, являющуюся аргументом передаточной функции цифрового фильтра $H(z)$. Этим обеспечивается переход от аналогового фильтра – прототипа к цифровому фильтру. Указанная замена выполняется с помощью различных формул для фильтров разных типов.

Для фильтров нижних частот замену переменной p выполняют в соответствии с формулой

$$p = \gamma \frac{1 - z^{-1}}{1 + z^{-1}} \ . \qquad (3.29)$$

Использование (3.29) обеспечивает однозначное преобразование $H(p)$ в $H(z)$:

$$H(z) = H(p)\,|_{p=\gamma(1-z^{-1})/(1+z^{-1})} \ . \qquad (3.30)$$

В ходе преобразования (3.30) выполняется нелинейная трансформация частот. Соотношение между частотами *p*-плоскости (аналоговыми частотами Ω) и частотами z-плоскости (цифровыми частотами ω) определяется выражением

$$\Omega = \gamma \cdot tg(\pi\omega) \ . \qquad (3.31)$$

Обычно аналоговый фильтр-прототип выбирают по справочнику, содержащему характеристики фильтров с нормированной аналоговой частотой среза, равной единице. Тогда коэффициент преобразования γ в (3.29) можно определить из (3.31) с помощью выражения

$$\gamma = ctg\ \pi\omega_1 \ , \qquad (3.32)$$

где ω_1- нормированная (по отношению к частоте дискретизации) граничная частота пропускания синтезируемого цифрового рекурсивного ФНЧ. При синтезе цифровых фильтров верхних частот и полосовых фильтров используют формулы, приведенные в табл. 3.1 [4] . В табл.3.1 частоты ω_1 и ω_2, встречающиеся в формулах для полосового и режекторного фильтров , обозначают соответсвенно первую и вторую граничные частоты полосы пропускания.

Применение метода билинейного преобразования рассмотрим на примерах.

Таблица 3.1 - Формулы преобразований

Цифровой фильтр	Формула замены	Параметры	Связь аналоговых и цифровых частот
Верхних частот	$p \to \gamma \dfrac{1+z^{-1}}{1-z^{-1}}$	$\gamma = tg\pi\omega_1$	$\Omega = \gamma \cdot ctg\pi\omega$
Полосовой	$p \to \gamma \dfrac{1-2\alpha z^{1}+z^{-2}}{1-z^{-2}}$	$\gamma = ctg\pi(\omega_2-\omega_1)$ $\alpha = \dfrac{\cos\pi(\omega_2+\omega_1)}{\cos\pi(\omega_2-\omega_1)}$	$\Omega = \gamma \dfrac{\alpha-\cos 2\pi\omega}{\sin 2\pi\omega}$
Режекторный	$p \to \gamma \dfrac{1-z^{-2}}{1-2\alpha z^{1}+z^{-2}}$	$\gamma = tg\pi(\omega_2-\omega_1)$ $\alpha = \dfrac{\cos\pi(\omega_2+\omega_1)}{\cos\pi(\omega_2-\omega_1)}$	$\Omega = \gamma \dfrac{\sin 2\pi\omega}{\alpha-\cos 2\pi\omega}$

Пример 3.3

Определим коэффициенты цифрового рекурсивного полосового фильтра с монотонно убывающей АЧХ со следующими граничными частотами пропускания $f_{ГП1} = 1200$ Гц, $f_{ГП2} = 1800$ Гц и граничными частотами задерживания $f_{ГЗ1} = 500$ Гц, $f_{ГЗ2} = 2500$ Гц. Частота дискретизации $f_Д = 8$ КГц. Максимальное затухание (неравномерность) в полосе пропускания не более $R = 3$ дБ . Максимальное затухание в полосе задерживания не менее $A = 40$ дБ.

1 Определим нормированные граничные частоты по формуле $\omega = f / f_Д$, тогда

$\omega_{ГП1} = 1200 / 8000 = 0{,}15$; $\omega_{ГП2} = 1800/8000 = 0{,}225$;

$\omega_{ГЗ1} = 500/8000 = 0{,}0625$; $\omega_{ГЗ2} = 2500/8000 = 0{,}3125$.

2 По формулам для полосового фильтра (табл. 3.1) вычислим параметры γ и α :

$$\gamma = ctg\pi(0{,}225-0{,}15) = 4{,}165301;$$

$$\alpha = [\cos\pi(0{,}225+0{,}15)]/[\cos\pi(0{,}225-0{,}15)] = 0{,}393557 .$$

3 Находим по формулам , приведенным в табл. 3.1 , аналоговые граничные частоты задерживания:

$$\Omega_{ГЗ1} = \gamma \frac{\alpha-\cos 2\pi\omega_{ГЗ1}}{\sin 2\pi\omega_{ГЗ1}} = -5{,}772267 ;$$

$$\Omega_{ГЗ2} = \gamma \frac{\alpha-\cos 2\pi\omega_{ГЗ2}}{\sin 2\pi\omega_{ГЗ2}} = 3{,}499673 .$$

4 Так как АЧХ синтезируемого фильтра должна быть монотонно убывающей , то выберем в качестве аналогового фильтра – прототипа ФНЧ Баттерворта . По формуле (1.109) вычислим порядок ФНЧ. При этом вместо значения ω_2 в формуле (1.109) будем использовать минимальную по модулю аналоговую граничную частоту задерживания :

$$n \geq \frac{\log\sqrt{(10^{0{,}1\cdot 40}-1)/(10^{0{,}1\cdot 3}-1)}}{\log(3{,}499673)} = 3{,}678.$$

Выбираем порядок фильтра – прототипа n=4 .

5 По табл. 1.1 определяем передаточную функцию нормированного аналогового фильтра Баттерворта четвертого порядка [13,19]

$$H(p) = 1/[(1 + 0{,}765p + p^2) \cdot (1 + 1{,}848p + p^2)] .$$

6 Находим передаточную функцию цифрового полосового фильтра , осуществляя замену переменной p в соответствии с табл. 3.1 . Получаем $H_{П}(z) = H_1(z) \cdot H_2(z)$, где

$$H_1(z) = \frac{0{,}046433 \cdot (1 - z^{-2})^2}{1 - 1{,}384672z^{-1} + 2{,}017464z^{-2} - 1{,}151752z^{-3} + 0{,}704408z^{-4}} ,$$

$$H_1(z) = \frac{0{,}03839 \cdot (1 - z^{-2})^2}{1 - 1{,}281183z^{-1} + 1{,}668067z^{-2} - 0{,}815967z^{-3} + 0{,}408960z^{-4}} .$$

Таким образом , цифровой полосовой фильтр реализуется в виде последовательного соединения двух фильтров с передаточными функциями $H_1(z)$ и $H_2(z)$, которые могут быть представлены в общем виде

$$H_i(z) = \frac{b[0]_i + b[2]_i z^{-2} + b[4]_i z^{-4}}{1 + a[1]_i z^{-1} + a[2]_i z^{-2} + a[3]_i z^{-3} + a[4]_i z^{-4}} ,$$

где коэффициенты при переменных z соответствуют коэффициентам разностного уравнения (1.211) . АЧХ синтезированного полосового фильтра изображена на рис.3.6 .

Числовые значения АЧХ соответствуют заданным требованиям. В частности , затухание в полосе пропускания не превышает 3 дБ, затухание на частотах ниже 500 Гц составляет 60 дБ и более, затухание на частотах выше 2500 Гц составляет более 43 дБ.

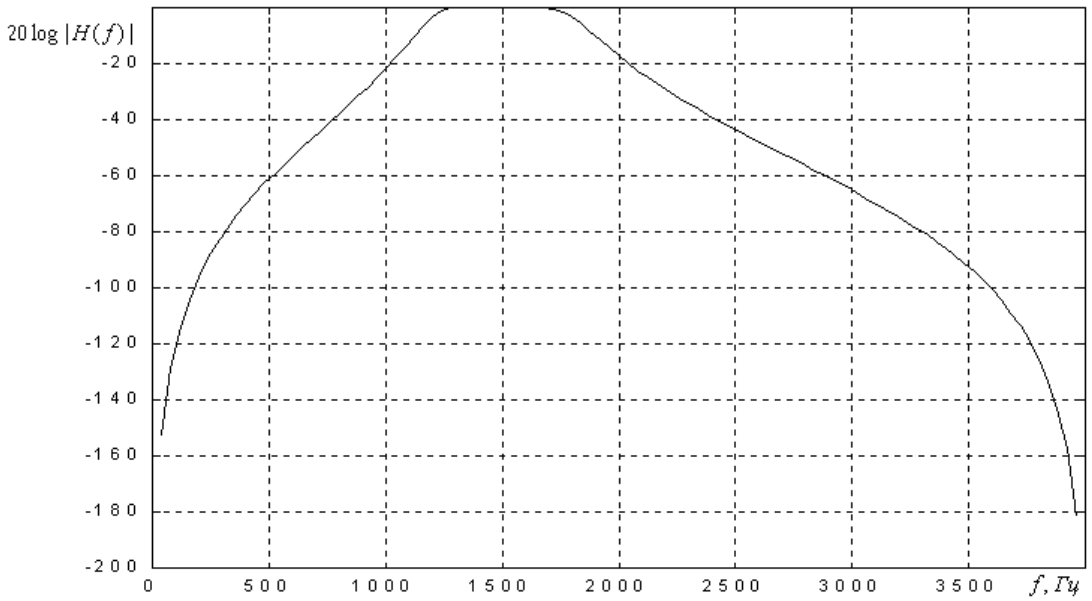


Рис. 3.6. АЧХ полосового фильтра

Пример 3.4

Определим коэффициенты цифрового рекурсивного ФНЧ с равноволновыми пульсациями АЧХ в полосе пропускания , удовлетворяющей следующим требованиям:

R=0,1 дБ; A=30 дБ; $f_Д = 12$ кГц; $f_{ГП1} = 1000$ Гц; $f_{ГЗ1} = 4000$ Гц.

1 Определим нормированные граничные частоты по формуле $\omega = f / f_Д$, тогда

$$\omega_{ГП1} = 1000 / 12000 = 0,0833 \; ; \; \omega_{ГЗ1} = 4000/12000 = 0,3333 .$$

2 Вычислим коэффициент γ :

$$\gamma = ctg\,(\pi\omega_{ГП1}) = ctg\,(\pi \cdot 0,0833) = 3,7321 .$$

3 Определим аналоговую граничную частоту задерживания:.

$$\Omega_{ГЗ1} = \gamma \cdot tg(\pi\omega_{ГЗ1}) = 3,7321 \cdot tg(\pi \cdot 0,3333) = 6,4641 .$$

4 Поскольку требуется выполнить синтез ФНЧ с равноволновой АЧХ в полосе пропускания, то выбираем в качестве фильтра-прототипа аналоговый фильтр Чебышева. По формуле (1.115) определяем требуемый порядок аналогового фильтра-прототипа

$$n \geq \frac{Arch\sqrt{(10^{0,1\cdot 30}-1)/(10^{0,1\cdot 0,1}-1)}}{Arch(6,4641)} = 2,36.$$

Принимаем n=3.

По табл.7.3 [13], используя значение порядка n=3 и заданное значение неравномерности R=0,1 дБ, выбираем полином знаменателя передаточной функции (1.110) аналогового ФНЧ Чебышева:

$$N(p) = (0,969 + p)(1,690 + 0,969\,p + p^2) .$$

Передаточная функция аналогового фильтра-протипа будет равна $H(p) = K / N(p)$, где K- нормирующий коэффициент , значение которого определяется из условия $|H(0)| = 1$ (K=1,6376).

5 Находим передаточную функцию цифрового рекурсивного ФНЧ , осуществляя замену переменной p в соответствии с (3.29) . Получаем

$$H(z) = 0,0111 \cdot \frac{1 + z^{-1}}{1 - 0,5878\,z^{-1}} \cdot \frac{1 + 2z^{-1} + z^{-2}}{1 - 1,2725\,z^{-1} + 0,6240\,z^{-2}} .$$

Требуемый цифровой ФНЧ может быть реализован в виде двух звеньев первого и второго порядков , соединенных последовательно.

АЧХ синтезированного фильтра изображена на рис.3.7. Неравномерность АЧХ в полосе пропускания не превышает 0,1 дБ. Уровень затухания на частоте 4000 Гц более 31дБ.

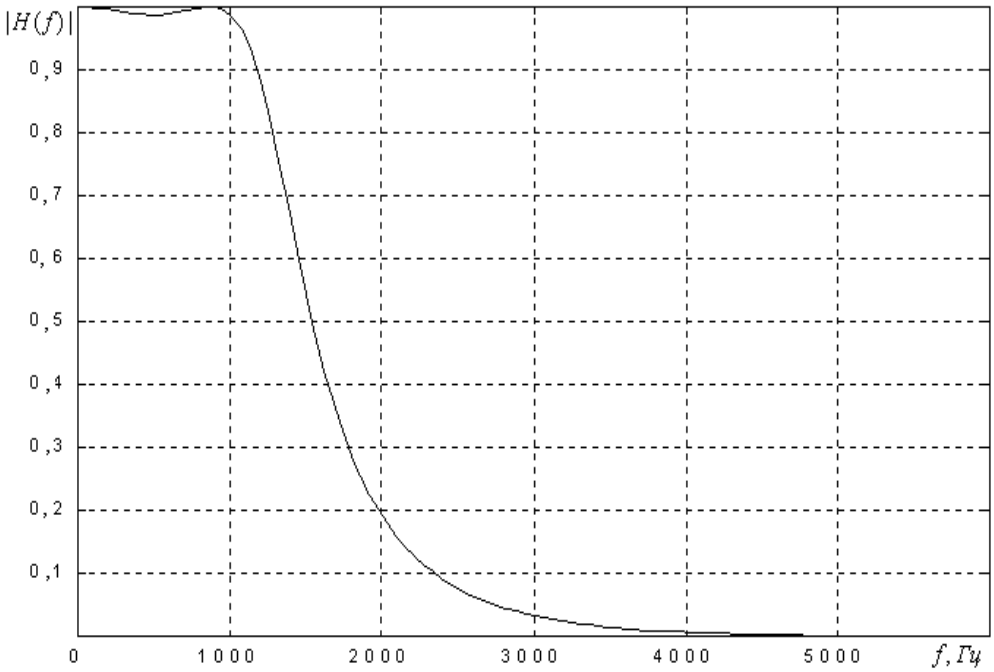


Рис.3.7. АЧХ цифрового ФНЧ Чебышева

3.3 Синтез двумерных цифровых фильтров

3.3.1 Общие сведения о методах синтеза двумерных ЦФ

При цифровой фильтрации двумерных сигналов в основном применяют фильтры с конечной импульсной характеристикой . Обусловлено это двумя обстоятельствами . Во-первых, двумерные КИХ-фильтры имеют ряд преимуществ по сравнению с двумерными БИХ-фильтрами. Во – вторых, методики синтеза двумерных БИХ-фильтров достаточно сложны из-за трудностей обеспечения устойчивости синтезируемых фильтров. Поэтому ниже будут рассматриваться только методы синтеза двумерных КИХ-фильтров.

Одно из важных преимуществ КИХ-фильтров перед БИХ-фильтрами состоит в возможности синтеза фильтров с нулевым фазовым сдвигом. Это свойство очень важно для многих приложений двумерной обработки сигналов. Например, при обработке изображений фильтры с ненулевой фазой могут привести к разрушению линий и границ на изображении [7].

Двумерный КИХ-фильтр будет обеспечивать нулевой сдвиг фазы, если его частотная характеристика является действительной функцией, т.е.

$$H(\omega_1,\omega_2) = H^*(\omega_1,\omega_2) \ , \qquad (3.33)$$

или если импульсная характеристика будет симметрична относительно начала координат

$$h[n_1, n_2] = h^*[-n_1, -n_2]. \qquad (3.34)$$

Существует несколько методов расчета двумерных КИХ фильтров [7,16]: метод оконных функций; метод частотной выборки; метод частотных преобразований. Приведем краткую характеристику каждого из указанных методов и продемонстрируем особенности синтезируемых с их помощью двумерных фильтров.

3.3.2 Метод оконных функций

Метод оконных функций, используемый для синтеза двумерных фильтров, имеет много общего с аналогичным методом синтеза одномерных фильтров. В соответствии с данным методом заданная (желаемая) двумерная частотная характеристика фильтра $H_з(\omega_1, \omega_2)$ представляется в виде ряда Фурье

$$H_з(\omega_1, \omega_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} h_з[n_1, n_2] e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}, \qquad (3.35)$$

где коэффициенты $h_з[n_1, n_2]$ ряда определяются с помощью выражения

$$h_з[n_1, n_2] = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} H(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2 . \qquad (3.36)$$

Здесь $h_з[n_1, n_2]$- бесконечная импульсная характеристика двумерного фильтра, соответствующего заданной частотной характеристике $H_з(\omega_1, \omega_2)$. Для реализации двумерного КИХ-фильтра необходимо ограничить пределы суммирования в (3.35). Аналогично одномерному случаю это приводит к ухудшению сходимости усеченного ряда (3.35) к заданной частотной характеристике в точках ее разрыва. Для улучшения сходимости усеченного ряда коэффициенты $h_з[n_1, n_2]$ умножают на конечную двумерную оконную функцию $w[n_1, n_2]$, т.е. в качестве коэффициентов фильтра используют значения

$$h\,[n_1, n_2] = h_з[n_1, n_2] \cdot w[n_1, n_2]. \qquad (3.37)$$

Чтобы получить фильтр с нулевым фазовым сдвигом, оконная функция должна удовлетворять условию:

$$w[n_1, n_2] = w^*[-n_1, -n_2]. \tag{3.38}$$

Для синтеза двумерных КИХ - фильтров применяют двумерные оконные весовые функции , определяемые на прямоугольной или круговой опорной области. В случае *прямоугольной опорной области* двумерное весовое окно формируется как прямое произведение двух одномерных окон:

$$w[n_1, n_2] = w_1[n_1] \cdot w_2[n_2]. \tag{3.39}$$

В случае *круговой опорной области* весовое окно формируется путем вращения одномерного весового окна вокруг оси симметрии с помощью выражения

$$w[n_1, n_2] = w(\sqrt{n^2{}_1 + n^2{}_2}), \tag{3.40}$$

где $w(\cdot)$ - функция одномерного непрерывного окна. С помощью (3.40) можно получить двумерные дискретные окна Хемминга, Ханна, Кайзера и др., обладающие круговой симметрией. Круговая симметрия желательна при обработке изображений, когда любое направление в плоскости изображения является равноценным.

Пример 3.5

Синтезируем двумерный КИХ-фильтр нижних частот , определяемый частотным откликом , заданным на прямоугольной опорной области

$$H_3(\omega_1, \omega_2) = \begin{cases} 1, & |\omega_1| \le \omega_c, |\omega_2| \le \omega_c, \\ 0, & в \ \ остальных \ \ случаях \end{cases}, \tag{3.41}$$

где ω_c - нормированная частота среза ($\omega_c \le \pi$). Из (3.36) следует, что идеальный импульсный отклик представляет разделяемую последовательность и равен

$$h_3[n_1, n_2] = \frac{1}{4\pi^2} \int_{-\omega_c}^{\omega_c} \int_{-\omega_c}^{\omega_c} e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2 = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega_1 n_1} d\omega_1 \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega_2 n_2} d\omega_2 =$$

$$= \frac{\sin \omega_c n_1}{\pi n_{1.}} \cdot \frac{\sin \omega_c n_2}{\pi n_{2.}}. \tag{3.42}$$

Для усечения ряда Фурье применим двумерное прямоугольное окно

$$w[n_1,n_2]=\begin{cases}1, & n_1 \le (P-1)/2, \qquad n_2 \le (P-1)/2,\\ 0, & в \quad остальныхслучаях \end{cases}, \tag{3.43}$$

где *P*- нечетное число, определяющее порядок фильтра. На рис. 3.8 представлены импульсная (ИХ) $h\,[n_1,n_2]=h_3[n_1,n_2]\cdot w[n_1,n_2]$ и частотная характеристики (ЧХ) синтезированного двумерного ФНЧ ($P=11,\ \omega_c=0{,}5\pi$) . ЧХ фильтра имеет значительные пульсации вблизи частот , где заданная частотная характеристика имеет разрыв.

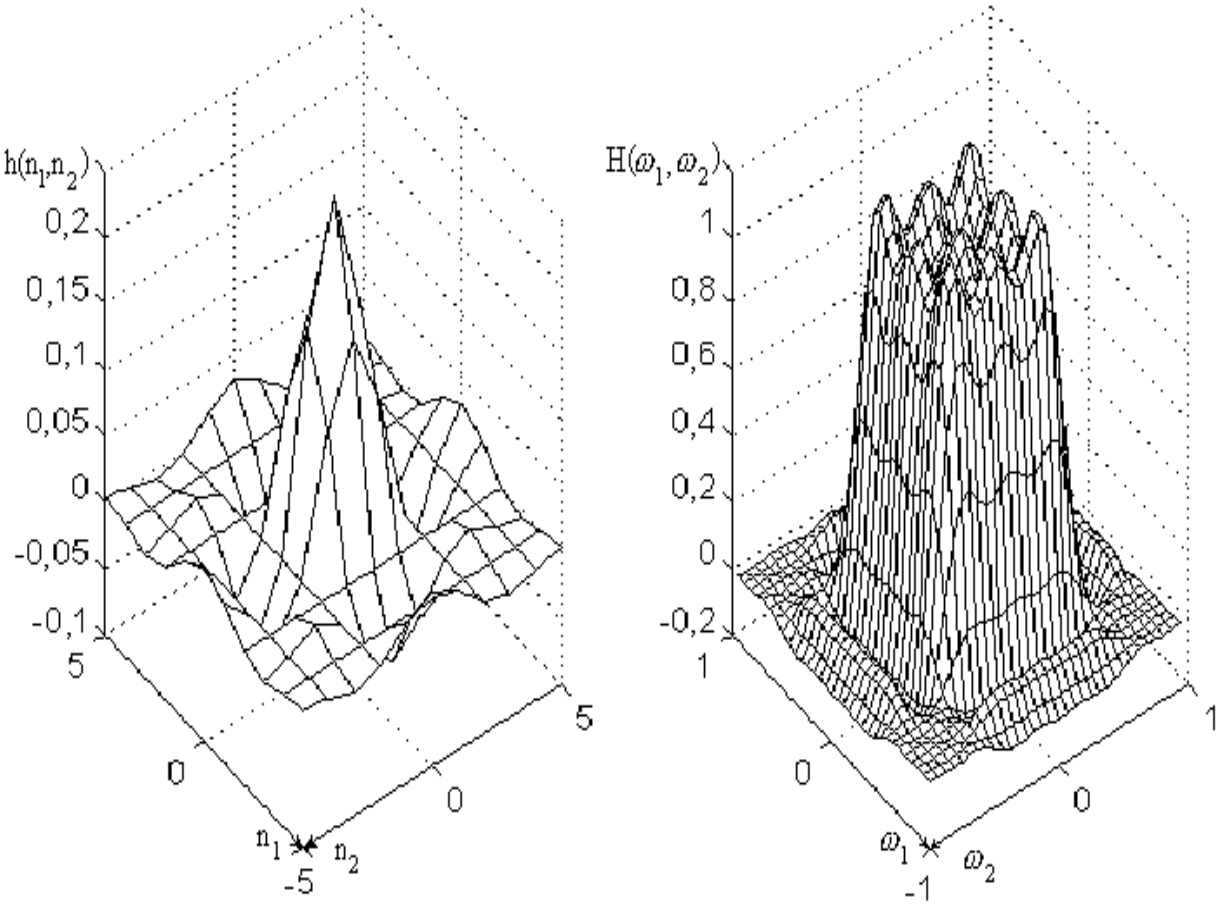


Рис.3.8. ИХ и ЧХ ФНЧ, синтезированного с помощью прямоугольного окна

Улучшить поведение ЧХ можно, если применить весовую функцию, отличную от прямоугольной. На рис.3.9 показана импульсная характеристика и ЧХ двумерного ФНЧ, синтезированного с применением окна Ханна. Двумерная оконная функция Ханна формировалась с помощью соответствующей одномерной функции на основе выражений (3.39) и (2.10) . Импульсная характеристика фильтра вычислялась умножением (3.42) на двумерную оконную функцию Ханна. Порядок фильтра и частота среза оставались прежними.

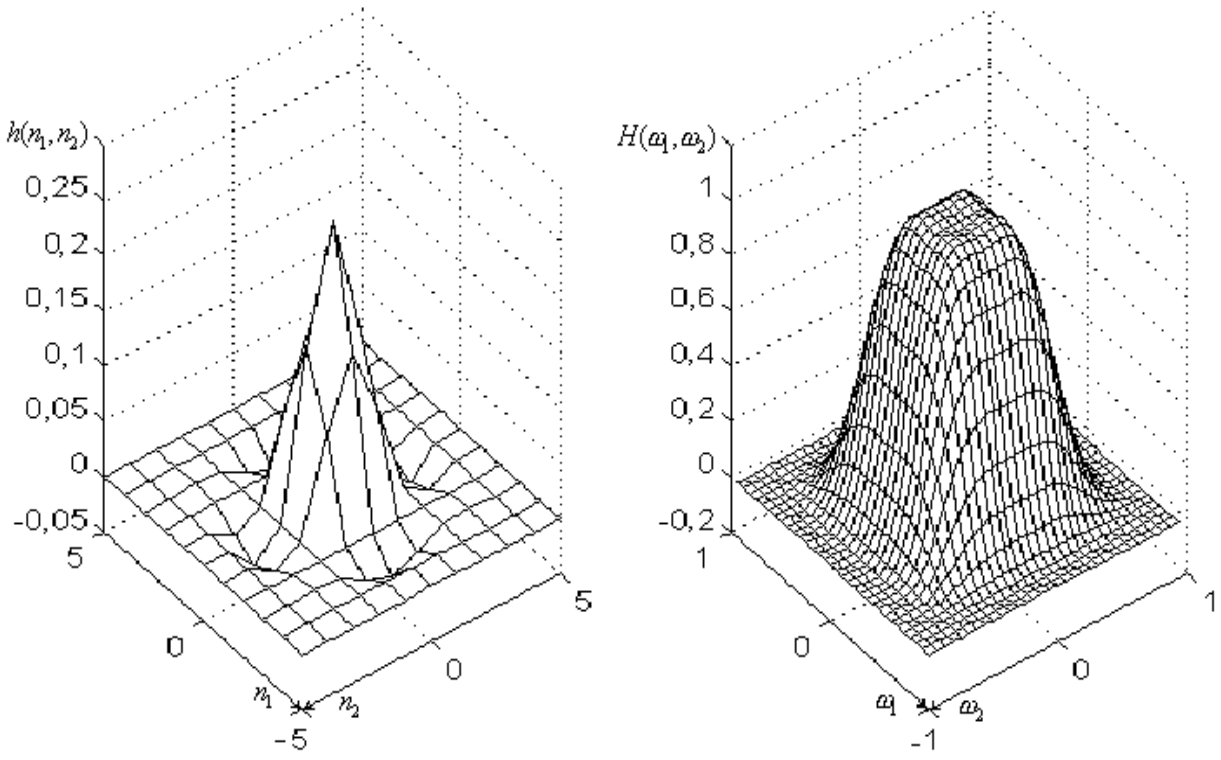


Рис.3.9. ИХ и ЧХ ФНЧ, синтезированного с использованием окна Ханна

Пример 3.6

Синтезируем двумерный КИХ-фильтр нижних частот , заданный частотной характеристикой с круговой симметрией

$$H_3(\omega_1,\omega_2)=\begin{cases}1, & \omega^2{}_1+\omega^2{}_2\le\omega^2{}_c,\\ 0, & в\ \ остальных случаях\ .\end{cases} \tag{3.44}$$

Импульсная характеристика фильтра определяется с помощью выражения (3.36), но интегрирование здесь выполняется не по прямоугольной области, а по круговой. Это требует замены переменных ω_1 и ω_2 на переменные в полярных координатах, что усложняет решение задачи по сравнению с предыдущим примером. После интегрирования получим [7]:

$$h_3[n_1,n_2]=\frac{\omega_c}{\pi}\cdot\frac{J_1(\omega_c\sqrt{n^2{}_1+n^2{}_2})}{\sqrt{n^2{}_1+n^2{}_2}}, \tag{3.45}$$

где $J_1(x)$ - функция Бесселя первого рода первого порядка. Импульсная характеристика (3.45) является бесконечной и требует ограничения. Ограничим ее с помощью кругового двумерного окна

$$w\ [n_1, n_2] = \begin{cases} 1, & n^2{}_1 + n^2{}_2 \le r^2\ , \\ 0, & в\ \ остальных\ \ случаях\ , \end{cases} \qquad (3.46)$$

где $r = (P-1)/2$ - радиус окна. На рис.3.10 изображены импульсная и частотные характеристики синтезированного фильтра (*P=11*, $\omega_c = 0.5\pi$)

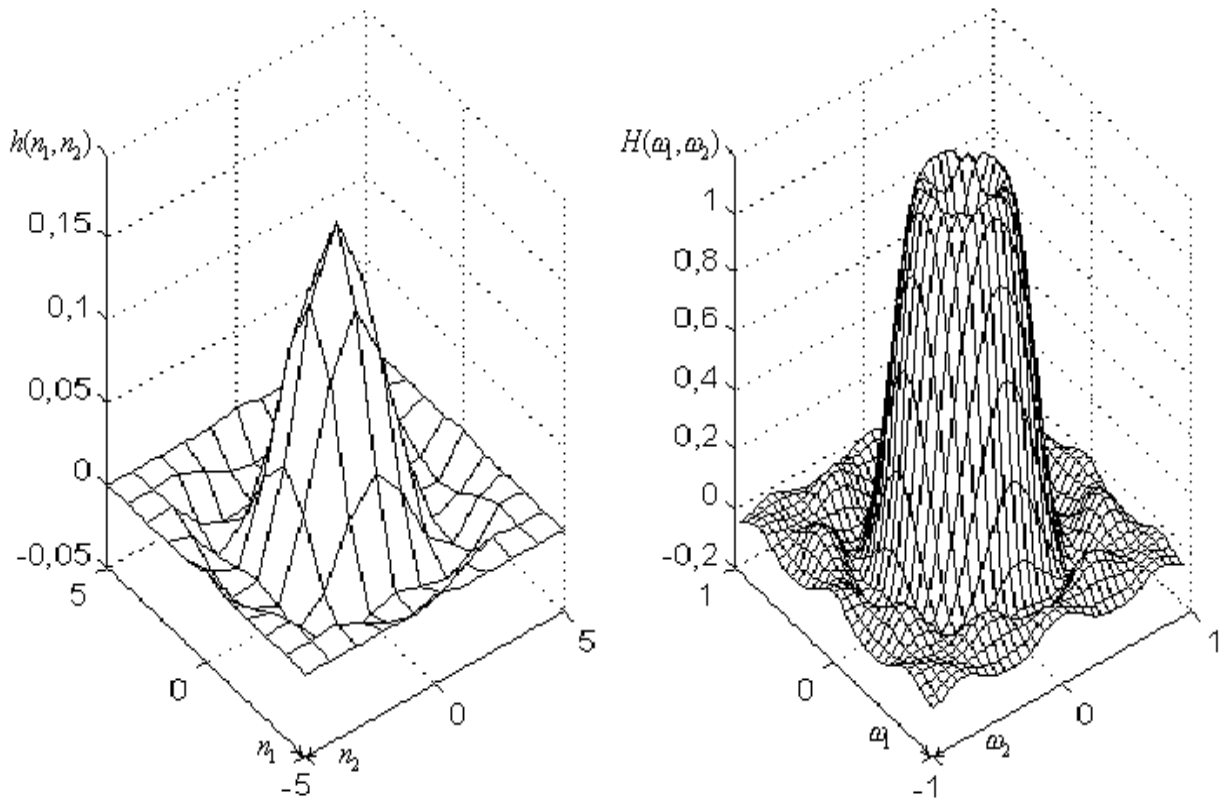


Рис.3.10. ИХ и ЧХ ФНЧ с круговой симметрией

Как видно из рис. 3.10 частотная характеристика синтезированного ФНЧ имеет необходимую круговую симметрию. Из-за применения кругового окна (3.46), близкого по своим свойствам к прямоугольному окну (3.43), синтезированная ЧХ имеет значительные пульсации вблизи частоты среза. Для уменьшения пульсаций следует применять иные весовые окна.

3.3.3 Метод частотной выборки

Данный метод использует для синтеза двумерных КИХ-фильтров прямое и обратное двумерные дискретные преобразования Фурье (ДДПФ). В соответствии с (1.260) импульсная характеристика ЦФ может быть вычислена с помощью ОДДПФ по заданной (желаемой) частотной характеристике $H[k_1, k_2]$:

$$h[n_1, n_2] = \frac{1}{N_1 \cdot N_2} \cdot \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} H[k_1, k_2] e^{j\frac{2\pi}{N_1} n_1 k_1 + j\frac{2\pi}{N_2} n_2 k_2} . \quad (3.47)$$

Если подставить (3.47) в (1.244), то можно получить выражение [16]

$$H(\omega_1, \omega_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} H[k_1, k_2] \cdot A(k_1, k_2, \omega_1, \omega_2) , \quad (3.48)$$

определяющее частотный отклик фильтра в любой точке (ω_1, ω_2). В выражении (3.48) член $A(k_1, k_2, \omega_1, \omega_2)$ представляет интерполирующую функцию, которая определяет поведение частотной характеристики между точками с частотами $\omega_{k1} = k_1 2\pi / N_1$, $\omega_{k2} = k_2 2\pi / N_2$. В точках $\omega_1 = \omega_{k1}$, $\omega_2 = \omega_{k2}$ частотная характеристика $H(\omega_1, \omega_2)$ синтезированного фильтра совпадает с желаемой частотной характеристикой, заданной своими отсчетами $H[k_1, k_2]$.

В качестве примера на рис.3.11 показаны результаты синтеза двумерного КИХ -фильтра нижних частот с частотной характеристикой , определяемой выражением (3.41). Из рис.3.11 видно, что реальная ЧХ имеет значительные пульсации.

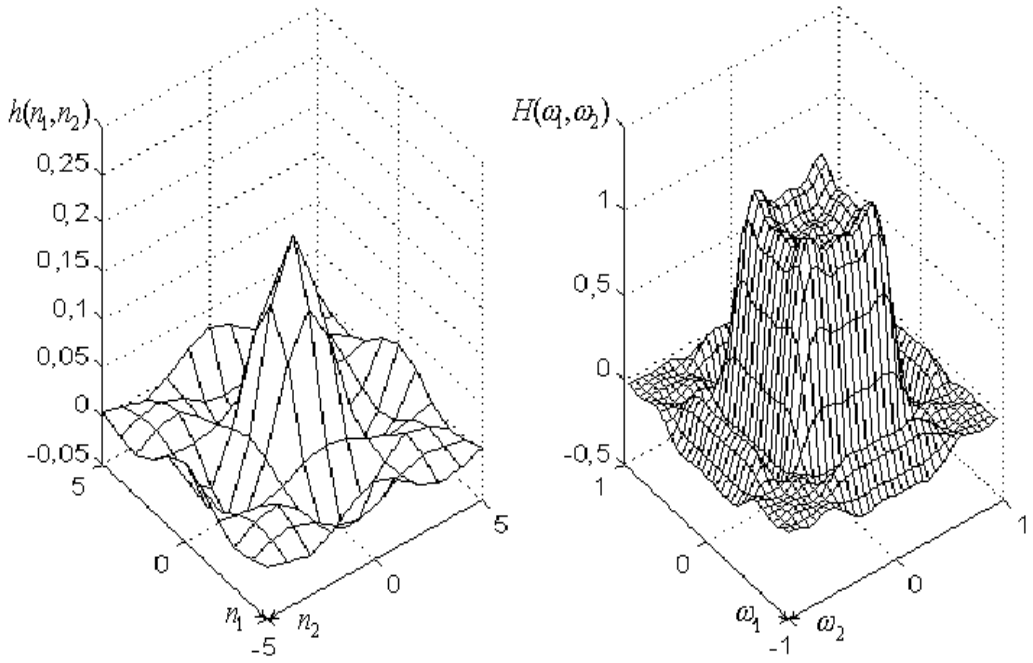


Рис. 3.11. ИХ и ЧХ ФНЧ, синтезированного методом частотной выборки

Аналогично одномерному случаю для уменьшения пульсаций ЧХ можно вводить дополнительные отсчеты в переходную полосу. Введе-

ние дополнительных отсчетов позволяет перераспределить энергию пульсаций, сосредоточив ее в переходной полосе. На рис.3.12 изображены заданная и реальная ЧХ ФНЧ, синтезированного методом частотной выборки. Заданная ЧХ содержит дополнительные отсчеты в переходной полосе. Амплитуда дополнительных отсчетов равна 0.75. Видно, что по сравнению с рис.3.11 уровень пульсаций в полосе пропускания стал значительно ниже.

Метод частотной выборки позволяет выбирать значения отсчетов в переходной полосе в соответствии с некоторым критерием оптимизации, что обеспечивает возможность синтеза оптимальных двумерных фильтров [7].

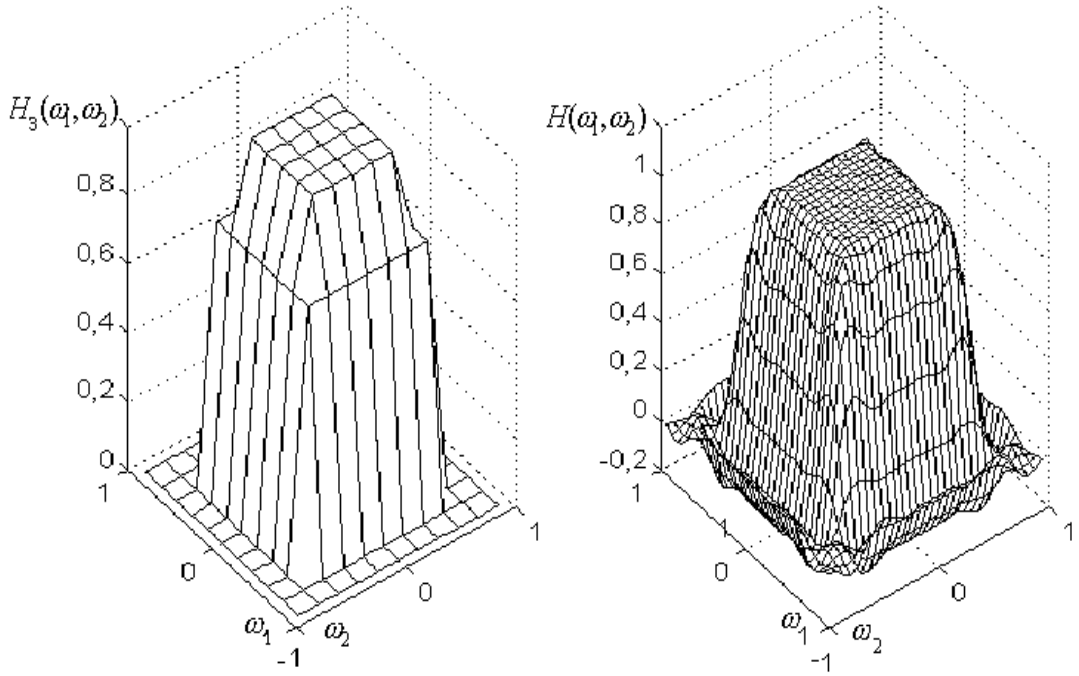


Рис.3.12. Заданная и реальные ЧХ ФНЧ с дополнительными отсчетами

3.3.4 Метод частотных преобразований

Прямой синтез двумерных оптимальных КИХ-фильтров затруднен из-за большой размерности задачи и невозможности распространения теоремы Чебышева на двумерный случай. Поэтому был разработан метод частотных преобразований, позволяющий выполнять преобразование оптимальных одномерных КИХ-фильтров в квазиоптимальные двумерные фильтры.

Метод частотных преобразований обеспечивает сохранение основных свойств одномерных фильтров в процессе преобразований. В частности, сохраняется ширина полос пропускания и заданные уровни пуль-

саций в полосах. Теоретическое обоснование метода приведено в [16]. Возможности метода рассмотрим на примере.

Пример 3.7

Выполним синтез двумерного КИХ-фильтра нижних частот, желаемая частотная характеристика которого определяется выражением

$$H_3(\omega_1,\omega_2)=\begin{cases}1, & \omega^2{}_1+\omega^2{}_2\le\omega^2{}_{c1},\\ 0, & \omega^2{}_1+\omega^2{}_2\ge\omega^2{}_{c2},\end{cases}$$

где ω_{c1},ω_{c2} - частоты, определяющие переходную область ($\omega_{c1}=0{,}45\pi,\omega_{c2}=0{,}65\pi$). Пусть также требуется, чтобы реальная частотная характеристика фильтра имела равноволновые пульсации и уровень пульсаций в области пропускания и подавления был соответсвенно не более 0,05.

Синтезируем сначала одномерный фильтр Чебышева с заданными характеристиками. Для этого воспользуемся модифицированной программой, приведенной в примере 3.2. Получим вектор коэффициентов *b* одномерного ФНЧ, удовлетворяющего заданным требованиям. После чего синтезируем двумерный КИХ-фильтр с помощью функции ftrans2 пакета Matlab 5.2 . Данная функция реализует метод частотных преобразований и вызывается следующим образом:

```
h=ftrans2(b).
```

В качестве результата функция возвращает матрицу коэффициентов *h* двумерного фильтра.

На рис. 3.13 показана АЧХ синтезированного одномерного КИХ фильтра и ЧХ соответствующего двумерного фильтра. Частоты нормированы относительно π .

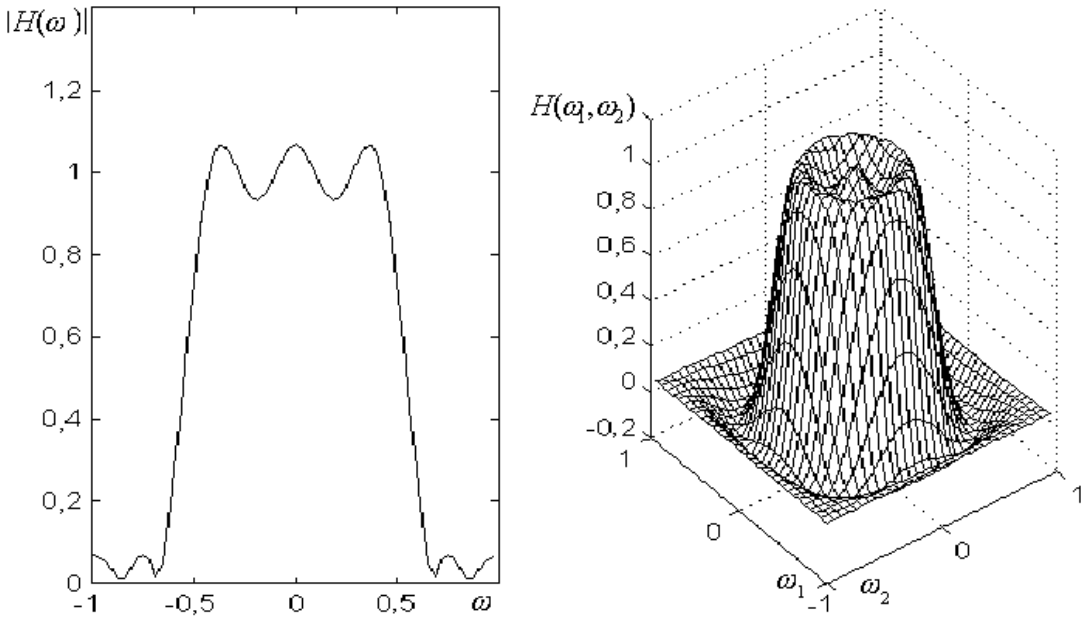


Рис.3.13. АЧХ одномерного ФНЧ и ЧХ соответствующего двумерного ФНЧ

Глава 4
ОБРАБОТКА РЕЧЕВЫХ И АУДИОСИГНАЛОВ

4.1 Основные свойства речевых сигналов

Голосовой аппарат человека представляет собой акустическую систему, состоящую из ротового и носового каналов, возбуждаемую квазипериодическими импульсными колебаниями голосовых связок и турбулентным шумом. Турбулентный шум образуется путем проталкивания воздуха через сужения в определенных областях голосового тракта. Голосовой аппарат, возбуждаемый указанными источниками, действует как линейный фильтр с изменяющимися во времени параметрами, на выходе которого формируется речевой сигнал. На коротких интервалах времени речевой сигнал можно аппроксимировать сверткой возбуждающего сигнала с импульсной характеристикой голосового тракта. На рис.4.1 изображена упрощенная модель формирования речевого сигнала [15,17]. В соответствии с этой моделью *вокализованные* (звонкие) звуки формируются с помощью генератора импульсной последовательности, а *фрикативные* (шумовые) – с помощью генератора случайных чисел.

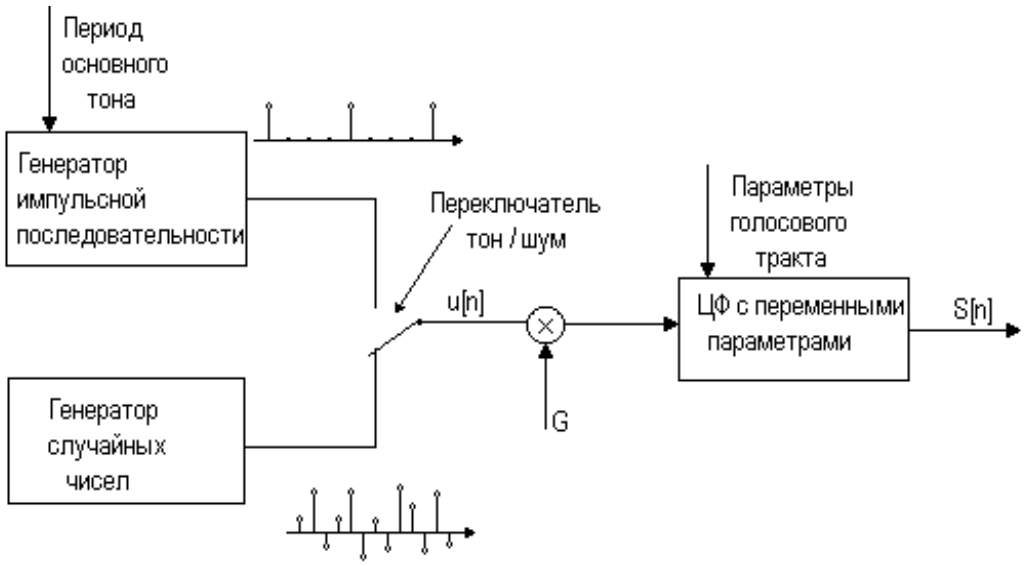


Рис.4.1. Цифровая модель формирования речевого сигнала

Период следования импульсов на выходе генератора импульсной последовательности соответствует *основному периоду* возбуждения голосовыми связками. Генератор случайных чисел формирует шумовой сигнал с равномерной спектральной плотностью. Цифровой фильтр (ЦФ) с переменными параметрами аппроксимирует передаточные свойства голосового тракта. На временном интервале порядка 5-20 мс форма голосового тракта не меняется, поэтому характеристики ЦФ на данном интервале остаются постоянными. Амплитуда входного сигнала u[n] цифрового фильтра определяется коэффициентом усиления G.

Вокализованные звуки представляют собой квазипериодические сигналы (рис. 4.5,а), гармоническая структура которых хорошо видна на графике кратковременного спектра. Фрикативные звуки имеют случайный характер (рис. 4.5,б) и занимают более широкий частотный диапазон. Энергия вокализованных звуков речи намного больше, чем энергия фрикативных звуков. Структура кратковременного спектра вокализованных участков речи (рис. 4.5,а) характеризуется наличием медленно меняющейся и быстро меняющейся составляющих. Быстро меняющаяся или пульсирующая составляющая обусловлена квазипериодическими колебаниями голосовых связок. Медленно меняющаяся составляющая связана с собственными (резонансными) частотами голосового тракта - формантами. В среднем насчитывается 3-5 формант. Первые три форманты оказывают существенное влияние на синтез и восприятие вокализованных участков речи. Их частоты находятся ниже 3 кГц. Форманты с более высокими частотами оказывают влияние на синтез и представление фрикативных звуков.

Рассмотренная цифровая модель формирования речевого сигнала характеризуется следующими параметрами: наличием классификатора вокализованных и невокализованных звуков (переключатель тон/шум), периодом основного тона, коэффициентом усиления G, параметрами (коэффициентами) ЦФ.

На рассмотренной модели базируются многочисленные способы представления речевых сигналов: от простейшей периодической дискретизации речевого сигнала до оценок параметров модели, представленной на рис.4.1. Выбор того или иного способа представления речевого сигнала определяется решаемой задачей, которые разделяются на три класса. К первому классу относят задачи, связанные с анализом речи. *Анализ речи* является неотъемлемой частью систем распознавания речевых сигналов, а также систем идентификации дикторов по голосу. Ко второму классу относят задачи, связанные с *синтезом речи* по тексту. Задачи такого типа возникают в многочисленных информационно-справочных системах. В задачах, относящихся к третьему классу, выполняется как анализ речевого сигнала, так и его синтез. Примерами могут служить разнообразные системы сжатия речевых сигналов с целью передачи речи

по компьютерным сетям или по традиционным линиям связи. Одним из перспективных направлений применения обработки речевых сигналов являются системы распознавания речи в сети Internet. В этом случае пользователь сети, используя телефон, может соединиться с программой распознавания речи, находящейся на сервере и транслирующей диалог в команды Web-сервера. Это позволяет получить доступ к распределенным информационным ресурсам сети по телефону. Данная технология, использующая методы цифровой обработки сигналов, базируется на использовании специального языка программирования Web-серверов VoxML (Voice Markup Language).

В дальнейшем рассмотрим основные способы цифрового представления и обработки речевых сигналов, применяемые как в задачах анализа речевых сигналов, так и в задачах синтеза.

4.2 Дискретизация и квантование речевых сигналов

Простейшее цифровое представление речи заключается в непосредственной дискретизации непрерывного речевого сигнала в соответствии с теоремой Котельникова. Такое представление речевого сигнала соответствует *импульсно-кодовой модуляции* (ИКМ). Выбор частоты дискретизации зависит от конкретных условий решаемой задачи. Фрикативные звуки речи занимают сравнительно широкую полосу частот (примерно до 10 кГц). Вокализованные звуки, значительно влияющие на разборчивость речи, занимают полосу частот до 3 кГц. Таким образом, обычно используемая частота дискретизации выбирается в пределах от 6 до 20 кГц.

Следует отметить, что количество операций, выполняемое при обработке речи, находится в прямой зависимости от частоты дискретизации. Поэтому в соответствии с условиями решаемой задачи необходимо по возможности снижать частоту дискретизации. Для этого перед дискретизацией речевой сигнал предварительно обрабатывают с помощью аналогового ФНЧ, устраняя нежелательные высокочастотные составляющие.

Выбор числа двоичных единиц B для кодирования одного отсчета речевого сигнала определяется задачей обработки. Объективной характеристикой точности представления сигнала посредством ИКМ является *шум квантования*

$$e[n] = x[n] - Q\{x[n]\}, \qquad (4.1)$$

где $x[n]$ – исходная речевая последовательность; $Q\{x[n]\}$ – оператор квантования.

Можно показать, что при равномерном квантовании *отношение сигнал шум (ОСШ)*, выраженное в децибелах, будет равно [17]

$$ОСШ = 6 \cdot B - 7{,}2 \,. \qquad (4.2)$$

Например, если B=11, то $ОСШ \approx 60$ дБ, что служит мерой качества хорошего телефонного канала связи. Добавление одного двоичного разряда для представления отсчета речевого сигнала увеличивает *ОСШ* на 6 дБ. Таким образом, для представления речевых сигналов посредством ИКМ требуется скорость передачи в пределах от 66 000 до 220 000 бит/с. Это необходимо учитывать при компьютерном хранении и обработке речи. Например, речевой сигнал длительностью 15 с будет занимать объём памяти примерно в 1 Мбайт. Для снижения скорости передачи речевых сигналов и, соответственно, уменьшения объемов требуемой памяти сокращают число двоичных единиц, выделяемых на один отсчет. Ключом к решению задачи является учет того обстоятельства, что для вокализованных участков речи, имеющих большую амплитуду, можно использовать большой шаг квантования, а для невокализованных – мелкий, т.е. квантование должно выполняться с неравномерным шагом. Это стабилизирует отношение сигнал/шум и делает его не зависящим от уровня сигнала.

Чтобы относительная ошибка квантования оставалась постоянной при изменении амплитуды речевого сигнала, уровни квантования должны быть распределены по логарифмическому закону. Вместо распределения уровней квантования по логарифмическому закону можно выполнять квантование логарифма речевого сигнала. В этом случае перед квантованием речевой сигнал обрабатывают в *компрессоре*, а при восстановлении исходного речевого сигнала используют *экспандер*. Совокупность этих двух устройств называют *компандером*. Одной из часто используемых характеристик компрессии является функция:

$$y = \left[\frac{\operatorname{sgn}(x)}{\ln(1+\mu)} \cdot \ln\left(1 + \mu \cdot \frac{|x|}{x_{\max}} \right) \right] x_{\max} \,. \qquad (4.3)$$

Функция компрессии (4.3) называется *μ-законом*. Экспандер реализует соответствующую обратную функцию

$$X = \frac{Y_{\max} \cdot \operatorname{sgn}(y)}{\mu} \cdot \left[(1+\mu)^{|y|} - 1 \right] \,. \qquad (4.4)$$

При использовании компрессора, функционирующего на основе (4.3), для обеспечения $ОСШ \approx 60$ дБ достаточно 7 бит на один отсчет ре-

чевого сигнала, т.е. скорость передачи может быть снижена по сравнению с равномерным квантованием в 1,57 раз. Отметим, что формулы (4.3) и (4.4) требуют, чтобы все отсчеты $x[n]$ находились в интервале $[-x_{max},x_{max}]$. Любой отсчет, не принадлежащий указанному интервалу, полагается равным $\pm x_{max}$. Значения константы μ обычно равны 30; 100; 254. Кроме μ- закона, часто используют A - *закон компандирования* [14].

Другой подход к снижению скорости передачи основан на учете избыточности речевого сигнала. Соседние отсчеты речевого сигнала, дискретизированного в соответствии с теоремой Котельникова, имеют сравнительно высокую корреляцию. Это позволяет по предыдущим отсчетам предсказать текущее значение речевого сигнала. Предположим, что $\tilde{x}[n]$ является предсказанным значением речевого сигнала $x[n]$. Если это предсказание является достаточно точным, то *ошибка предсказания*

$$d[n] = x[n] - \tilde{x}[n] \tag{4.5}$$

должна иметь небольшую величину, и, следовательно, дисперсия ошибки квантования разностного сигнала будет меньше, чем дисперсия ошибки квантования отсчетов речи $x[n]$. Таким образом, квантователь с заданным количеством уровней обеспечит меньшую погрешность при квантовании разностного сигнала, чем при квантовании исходного сигнала. Поэтому для представления разностного сигнала требуется меньшее число двоичных разрядов.

Квантователь, построенный на использовании указанного подхода, называется *дифференциальным (разностным) импульсно-кодовым модулятором* (ДИКМ). Схема его показана на рис.4.2. Здесь $\hat{x}[n]$ представляет восстановленный сигнал, образуемый путем добавления квантованного разностного сигнала $\hat{d}[n]$ к $\tilde{x}[n]$.

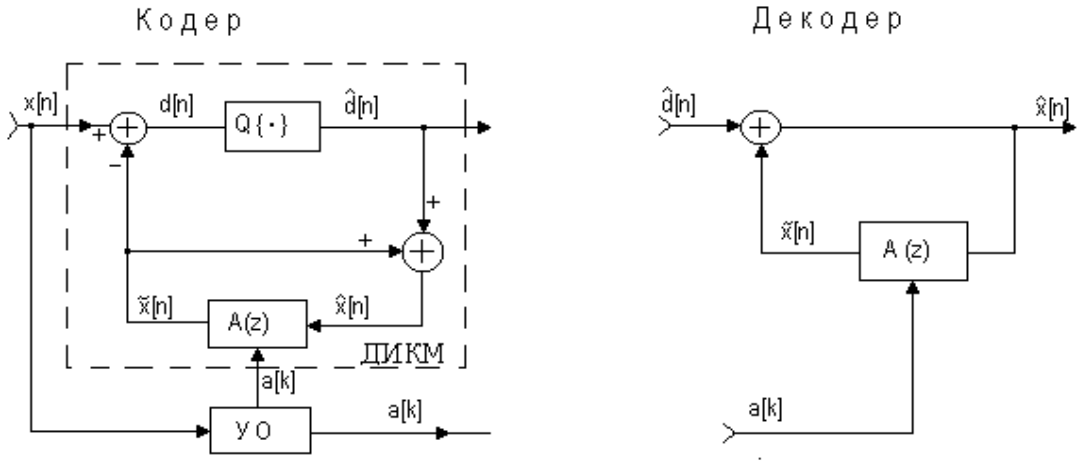


Рис. 4.2. Общая схема АДИКМ

Покажем, что восстановленный сигнал $\hat{x}[n]$ будет отличаться от $x[n]$ на величину шума квантования разностного сигнала

$$e[n] = d[n] - \hat{d}[n]. \tag{4.6}$$

Заметим, что

$$\hat{d}[n] = \hat{x}[n] - \tilde{x}[n]. \tag{4.7}$$

Тогда, подставляя (4.7) и (4.5) в (4.6), получим искомый результат

$$e[n] = x[n] - \hat{x}[n]. \tag{4.8}$$

Для предсказания значений речевого сигнала в схеме используется нерекурсивный ЦФ с передаточной функцией *A(z),* реализующий уравнение

$$\tilde{x}[n] = \sum_{k=1}^{P} a[k]\hat{x}[n-k]. \tag{4.9}$$

В простейшем случае для предсказания используется фильтр первого порядка.

В том случае, когда для кодирования разностного сигнала используется один бит, рассматриваемый модулятор называют *дельта-модулятором*. Обычно дельта модулятор функционирует на более высоких частотах дискретизации, чем ДИКМ.

Для уменьшения ошибки квантования в ДИКМ может применяться адаптивное изменение шага квантования и коэффициентов предсказывающего фильтра. Такие модуляторы называются *адаптивными дифференциальными импульсно-кодовыми модуляторами* (АДИКМ). Коэффициенты предсказывающего фильтра вычисляются в устройстве оценивания (УО) так, чтобы минимизировать дисперсию ошибки предсказания (4.5). Оценивание коэффициентов предсказывающего фильтра возможно как по входной речевой последовательности $x[n]$, так и по восстановленной последовательности $\hat{x}[n]$. В первом случае, коэффициенты предсказывающего фильтра должны кодироваться и передаваться в декодер, так как последовательность $x[n]$ отсутствует в декодере. Во втором случае такой необходимости нет. Коэффициенты предсказателя могут быть вычислены в декодере по восстановленной последовательности $\hat{x}[n]$. Основные принципы вычисления коэффициентов линейного предсказания будут рассмотрены ниже.

АДИКМ позволяет снизить скорость передачи до 16 Кбит/с. Благодаря этому АДИКМ широко применяется для представления речевых

сигналов в компьютерных системах. Детально вопросы реализации алгоритмов АДИКМ оговорены рекомендацией МККТТ G.726 [28]. В соответствии с этой рекомендацией на вход АДИКМ поступает стандартный ИКМ-сигнал с частотой дискретизации 8 кГц. Для представления квантованных разностных значений используется 5-, 4-, 3- и 2-разрядные двоичные коды. Это обеспечивает соответственно получение скорости передачи со значениями: 40, 32, 24, 16 Кбит/с.

4.3. Анализ речевых сигналов во временной области

При решении многих задач обработки речи интерес представляют временные характеристики речевых сигналов. Поскольку речь является нестационарным процессом, то ее принято анализировать на коротких участках (10 – 30 мс), где спектрально-корреляционные характеристики остаются примерно постоянными [14].

Одним из важных параметров речевого сигнала является его энергия

$$E = \sum_{n=0}^{N-1} x^2[n] . \qquad (4.10)$$

Энергия может служить хорошей мерой отличия вокализованных и невокализованных участков речи. Энергия невокализованных участков речи намного меньше, чем вокализованных.

Иной способ обнаружения вокализованных и невокализованных участков речи основан на измерении среднего числа переходов через ноль речевого сигнала. Это измерение является грубой оценкой частотного состава речевого сигнала. Известно, что энергия вокализованных звуков концентрируется в диапазоне ниже 3 кГц, тогда как энергия фрикативных звуков сосредоточена, в основном, на частотах выше 3 кГц. Поэтому, если среднее число переходов через ноль велико, то это свидетельствует о невокализованном характере речи, и наоборот.

Важной задачей анализа речевых сигналов во временной области является оценивание периода основного тона. Период основного тона может быть определен как временной интервал между соответствующими пиками вокализованного участка речевого сигнала. Однако главная трудность здесь состоит в том, что даже на коротких интервалах времени речевой сигнал не имеет строгой периодической структуры.

Иной способ определения периода основного тона во временной области основан на вычислении функции кратковременной автокорреляции

$$r[m] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x[n+m], \qquad 0 \le m < M_0 - 1, \qquad (4.11)$$

где M_0 – максимальная задержка сигнала.

При выявлении периода основного тона P по автокорреляционной функции необходимо учитывать условие $M_0 > P$. Для вокализованных участков речи на графике $r[m]$ прослеживаются пики с интервалом, равным периоду основного тона.

С целью обострения пиков на графике $r[m]$ сигнал $x[n]$ клиппируют [17]. Суть этой нелинейной операции показана на рис.4.3,а. На рис.4.3,б представлена автокорреляционная функция клиппированного сигнала.

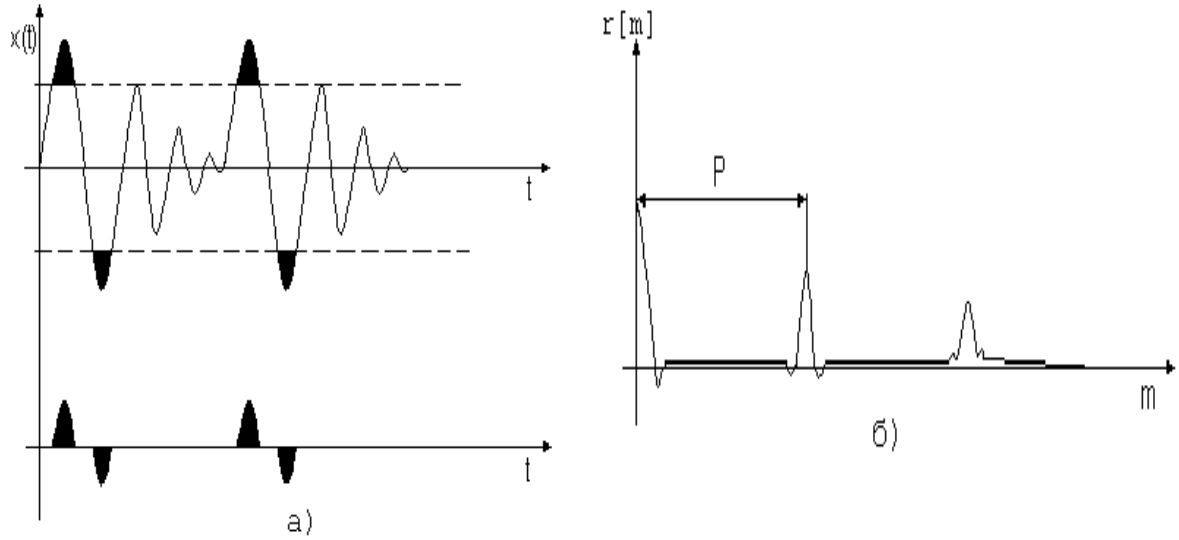


Рис.4.3. Операция клиппирования

Недостатком рассматриваемого метода определения периода основного тона является то, что для вычисления автокорреляционной функции требуется выполнить большое число арифметических операций.

4.4 Анализ речевых сигналов в частотной области

Одним из основных способов обработки речи в частотной области является кратковременный спектральный анализ. На выполнении кратковременного спектрального анализа основана работа многих систем распознавания речи, спектрографов, вокодеров [15].

Кратковременный спектральный анализ может быть реализован с использованием гребенки полосовых фильтров (рис. 4.4) или с помощью дискретного преобразования Фурье. Полосы пропускания фильтров выбираются так, чтобы перекрыть весь частотный диапазон речи. Средние

значения модулей выходных сигналов фильтров будут представлять значения спектральных коэффициентов в полосах.

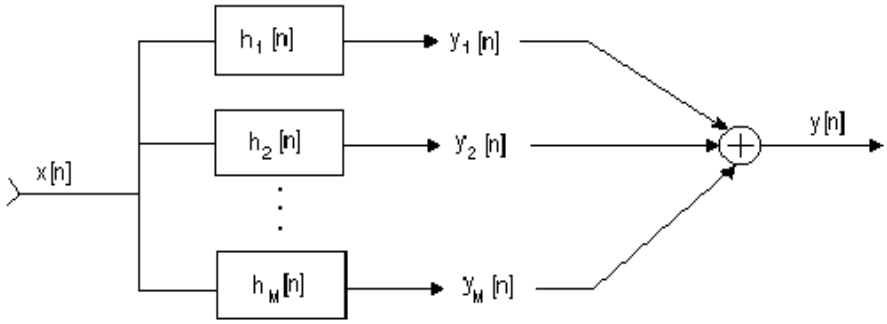


Рис. 4.4. Гребенка фильтров

Иногда частотный диапазон разбивают на неравные полосы с учетом особенностей слухового восприятия человека. Экспериментально установлено, что во внутреннем ухе человека высота тона (частота) звукового сигнала преобразуется в механические колебания определенных участков базилярной мембраны. При этом линейным приращениям координаты вдоль тела мембраны соответствуют логарифмические приращения частоты звука, т.е. частота звука, воспринимаемого человеком, нелинейно зависит от действительной физической частоты. Это приводит к неодинаковой разрешающей способности по частоте и к восприятию звуков в соответствии с *механизмом критических частотных полос*. Сложный звук постоянной громкости, состоящий из нескольких тонов, лежащих в пределах критической полосы, воспринимается человеком с таким же субъективным ощущением, как одно-тональный звук, соответствующий центральной частоте критической полосы. Ширина критических полос для области частот до 500 Гц составляет примерно 100 Гц. Выше 500 Гц ширина критических полос увеличивается примерно на 20% по сравнению с шириной предыдущей полосы. Ширина критических полос аппроксимируется зависимостью [36]

$$\Delta f = 25 + 75[1 + 1{,}4(f/1000)^2]^{0{,}69} \qquad [Гц].$$

Для характеристики субъективных частот, воспринимаемых человеком, предложено несколько шкал: *барк-шкала, мел-шкала*. Функция

$$b = 13\arctan(0{,}00076\, f) + 3{,}5\arctan[(f/7500)^2] \qquad [Барк]$$

используется для перевода частот, заданных в герцах, в барки.

Гребенка (банк) фильтров с неравными полосами пропускания, заданными в герцах и соответствующими критическим полосам слуха

(табл. 4.1), будет иметь равномерное распределение центральных частот и равные значения полос пропускания, измеренных в барках. Таким образом, применение барк-шкалы соответствует равномерному разбиению оси субъективных частот. Мел-шкала вводится с аналогичной целью и имеет незначительные отличия от барк-шкалы [36].

Таблица 4.1 - Критические полосы слухового восприятия

b, Барк	Центральная частота, Гц	Полоса, Гц
0	50	0-100
1	150	100-200
2	250	200-300
3	350	300-400
4	450	400-510
5	570	510-630
6	700	630-770
7	840	770-920
8	1000	920-1080
9	1170	1080-1270
10	1370	1270-1480
11	1600	1480-1720
12	1850	1720-2000
13	2150	2000-2320
14	2500	2320-2700
15	2900	2700-3150
16	3400	3150-3700
17	4000	3700-4400
18	4800	4400-5300
19	5800	5300-6400
20	7000	6400-7700
21	8500	7700-9500
22	10500	9500-12000
23	13500	12000-15500
24	19500	15500-

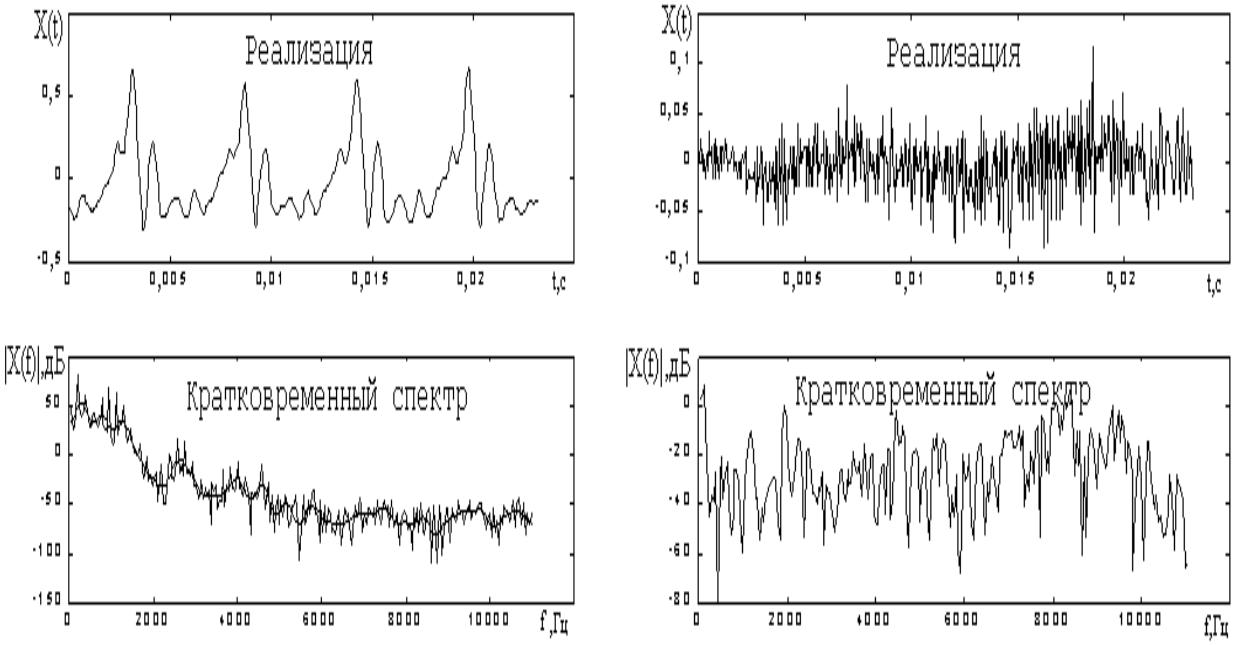
Кратковременный спектральный анализ речи может быть также выполнен на основе ДПФ. *Кратковременное дискретное преобразование Фурье* определяется следующим образом

$$X_l(\omega) = T_0 \sum_{n=0}^{N-1} x_l[n] e^{-j\omega n T_0}, \qquad (4.12)$$

где $x_l[n]$ представляет отрезок речи, взвешенный окном $w[n]$, длиной N отсчетов:

$$x_l[n] = w[n]x[n + lN], \qquad l = 0, 1, 2, \ .....\tag{4.13}$$

На рис.4.5 показаны кратковременные спектры Фурье и соответствующие реализации вокализованного и фрикативного участков речи. Спектры представлены в логарифмическом масштабе. При построении спектров использовалось окно Хемминга длительностью примерно 23 мс. Заметим, что кратковременный спектр вокализованной речи имеет высокое разрешение и характеризуется пульсациями, обусловленными квази-периодическими импульсами основного тона. Поэтому по кратковременному спектру вокализованной речи можно оценить частоту основного тона.



а) б)

Рис. 4.5. Вокализованный (а) и фрикативный (б) речевые сигналы и их спектры

Один из алгоритмов определения частоты основного тона основан на вычислении произведения [17]

$$F_l(\omega) = \prod_{k=1}^{K} \left|X_l(\omega k)\right|^2 \ .\tag{4.14}$$

Значения, вычисляемые с помощью (4.14), могут быть весьма большими. Для уменьшения значений вычисляют логарифм от (4.14). Зависимость $F_l(\omega)$ представляет произведение функций $|X_l(\omega k)|^2$, сжатых по частоте. В вокализованной речи сжатие частоты в *k* раз должно привести к совпадению гармоник основного тона. Благодаря этому в спектре $F_l(\omega)$ появляется максимум на частоте основного тона. Невокализованная речь характеризуется существенно меньшими значениями $F_l(\omega)$ и она не имеет максимума в спектре $F_l(\omega)$ на частоте основного тона. Данный способ определения частоты основного тона устойчив к шумам, поскольку шумовые компоненты в спектре $F_l(\omega)$ не регулярны. На рис.4.6 показаны графики $F_l(f)$ для вокализованной и невокализованной речи, полученные при *K*=5. По расположению пика на графике спектра $F_l(f)$ вокализованной речи легко определяется частота основного тона.

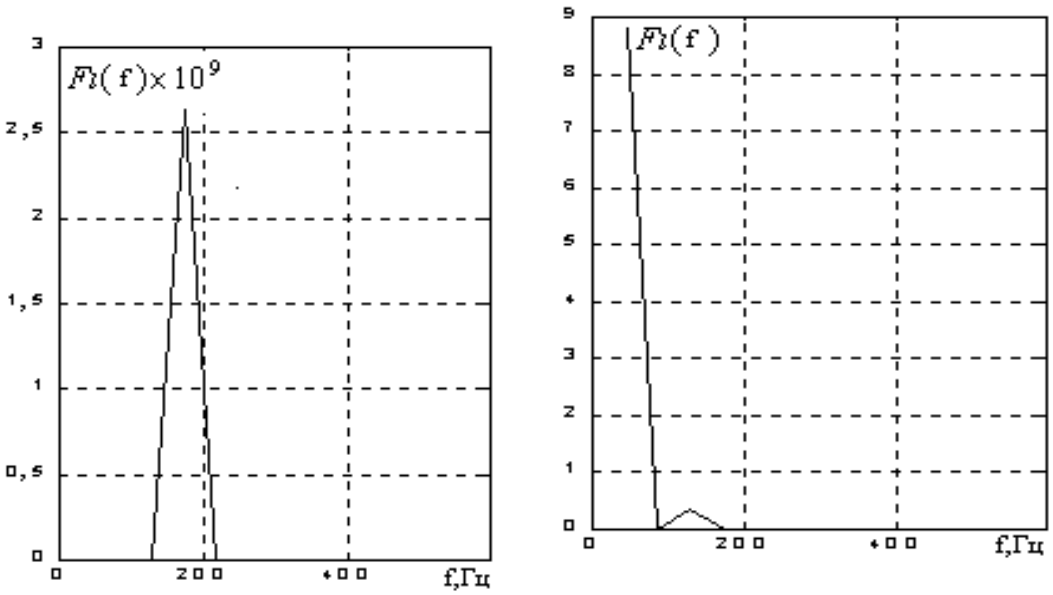


Рис.4.6. Спектр произведения гармоник вокализованной и невокализованной речи

4.5 Гомоморфная обработка речи

В соответствии с рис.4.1 речевой сигнал является сверткой функции возбуждения (случайного шума либо квазипериодической последовательности импульсов) и импульсной характеристики голосового тракта. Гомоморфный анализ речи позволяет разделить эти компоненты. Поэтому, используя гомоморфный анализ, можно определить период основного тона и частотные свойства голосового тракта. Общая схема гомоморфной обработки приведена на рис.4.7.

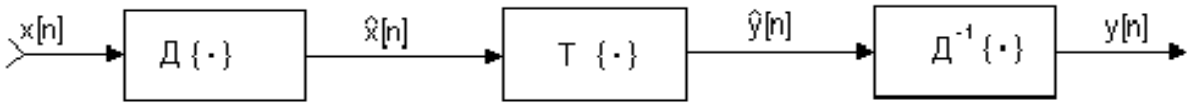


Рис. 4.7. Общая схема гомоморфной обработки

В соответствии с этой схемой сначала выполняется нелинейное преобразование Д{ } сигнала $x[n]$, которое определяется соотношением

$$\hat{X}(z) = \log X(z). \tag{4.15}$$

Затем выполняется оператор Т{ }, который соответствует линейной инвариантной системе. В конце реализуется преобразование $Д^{-1}\{ \}$, являющееся обратным по отношению к преобразованию Д{ }.

Пусть сигнал $x[n]$ является сверткой двух последовательностей $x_1[n]$ и $x_2[n]$. Тогда

$$X(z) = X_1(z) \cdot X_2(z). \tag{4.16}$$

Подставив (4.16) в (4.15), получим

$$\hat{X}(z) = \log X_1(z) + \log X_2(z) = \hat{X}_1(z) + \hat{X}_2(z). \tag{4.17}$$

Линейная инвариантная система пропускает на выход только одну из компонент $X_1(z)$ или $X_2(z)$. Соответственно обратное преобразование $Д^{-1}\{ \}$ дает $y[n]=x_1[n]$ или $y[n]=x_2[n]$. Следовательно, гомоморфная обработка разделяет входные компоненты $x_1[n]$ и $x_2[n]$, содержащиеся во входном сигнале.

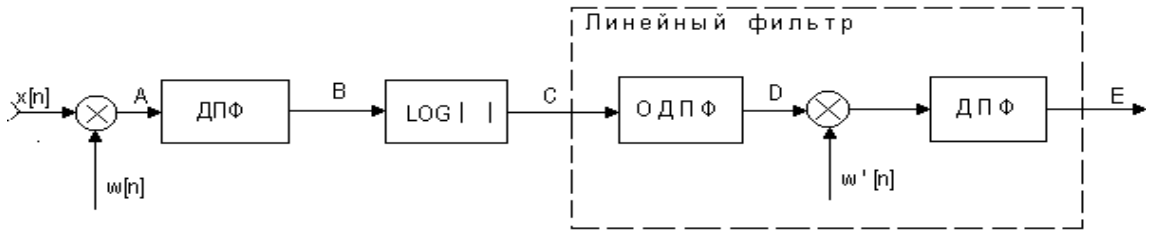


Рис.4.8. Гомоморфная система анализа речи

Гомоморфная система анализа речи показана на рис.4.8. Здесь на первом этапе вычисляется логарифм модуля кратковременного преобразования Фурье. Если предположить, что сигнал в точке А является сверткой функции возбуждения и импульсной характеристики голосового тракта, то в точке С мы получим сумму логарифмов спектра функции возбуждения и импульсной характеристики голосового тракта. Сигнал в

точке D, полученный с помощью обратного дискретного преобразования Фурье, называется *кепстром*. Кепстр в точке D равен сумме кепстров функции возбуждения и импульсной характеристики голосового тракта. Покажем это.

В общем случае, комплексный кепстр сигнала $x[n]$ определяется как обратное преобразование Фурье от логарифма комплексного спектра $X(\omega)$. Для модуля кратковременного спектра $|X(\omega)|$, который является четной и периодической функцией, можно использовать разложение в ряд Фурье

$$\log|X(\omega)| = \sum_{n=-\infty}^{\infty} c[n] \cdot e^{-jn\omega T_0}, \qquad (4.18)$$

где $c[n]$ – *кепстральные коэффициенты* и

$$c[n] = \frac{T_0}{2\pi} \int_{-\pi/T_0}^{\pi/T_0} \log|X(\omega)| \cdot e^{jn\omega T_0} d\omega . \qquad (4.19)$$

Если речевой сигнал является сверткой функции возбуждения $u[n]$ и импульсной характеристики $h[n]$ голосового тракта

$$x[n] = u[n] * h[n], \qquad (4.20)$$

то модуль спектра $|X(\omega)|$ равен произведению модулей $|U(\omega)|$ и $|H(\omega)|$:

$$|X(\omega)| = |U(\omega)| \cdot |H(\omega)| . \qquad (4.21)$$

Взяв логарифм от обеих частей (4.21), получим

$$\log|X(\omega)| = \log|U(\omega)| + \log|H(\omega)| . \qquad (4.22)$$

Так как обратное преобразование Фурье является линейной операцией, то из(4.22) следует, что

$$c_x[n] = c_u[n] + c_h[n], \qquad (4.23)$$

где $c_u[n]$ и $c_h[n]$ - кепстры последовательностей $u[n]$ и $h[n]$.

В схеме обработке речи, изображенной на рис.4.8 , вместо дискретно-непрерывного преобразования Фурье по непрерывной перемененной ω используется дискретное преобразование Фурье, определяемое на

фиксированных частотах. Рис.4.9 иллюстрирует указанные преобразования для вокализованной речи.

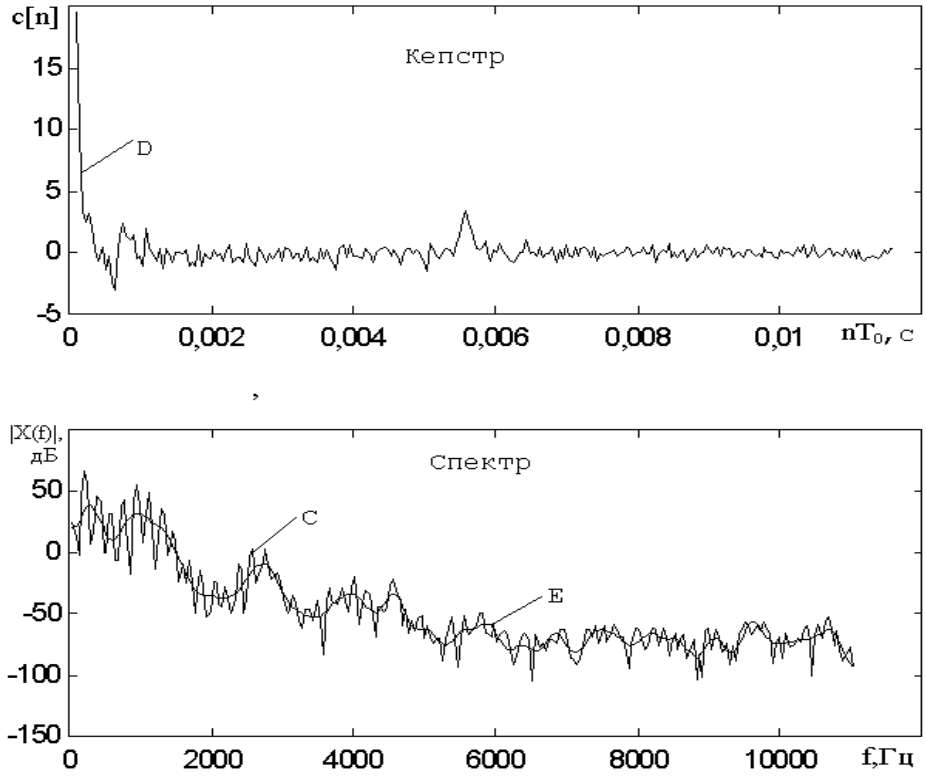


Рис. 4.9. Гомоморфный анализ вокализованной речи

Пульсирующая кривая С (рис.4.9) соответствует логарифму модуля кратковременного спектра. Она содержит медленно меняющуюся составляющую, соответствующую амплитудно-частотной характеристике голосового тракта, и быстро меняющуюся составляющую, обусловленную периодической функцией возбуждения. Выполнив ОДПФ логарифма спектра, получим кепстр, который является функцией времени. Медленно меняющаяся составляющая спектра соответствует области малых времен кепстра, а быстро меняющаяся периодическая составляющая спектра соответствует удаленному пику в кепстре (рис.4.9), возникающему через интервал времени, равный периоду основного тона. Для невокализованной речи указанный пик будет отсутствовать. Поэтому кепстр может использоваться для определения периода основного тона и характера речи (вокализованная или невокализованная).

Амплитудно-частотная характеристика голосового тракта получается низкочастотной фильтрацией сигнала, действующего в точке С (рис.4.8). В результате этого будут подавлены быстро меняющиеся элементы

на кривой С (рис. 4.9). Фильтрация может быть выполнена по методу быстрой свертки. Для этого вычисляется ОДПФ сигнала С, полученный кепстр умножается на подходящую функцию окна, пропускающую лишь область малых времен кепстра, и затем выполняется ДПФ. В результате такой обработки получим сглаженный спектр (рис. 4.9, кривая Е). Резонансные пики на кривой Е позволяют определить формантные частоты. Используя оценки формантных частот голосового тракта и период основного тона, можно синтезировать речь на основе модели, изображенной на рис.4.1.

4.6 Анализ речи на основе линейного предсказания

Анализ речи на основе линейного предсказания базируется на использовании модели речевого сигнала, представленной на рис.4.1. Основная задача метода состоит в том, чтобы по наблюдениям последовательности отсчетов речевого сигнала $s[n]$ определить коэффициенты $a[k]$ цифрового фильтра указанной модели [14,15,17].

Найденные значения коэффициентов, которые называют коэффициентами *линейного предиктивного кодирования* (ЛПК), могут применяться при определении частоты основного тона, при кодировании речи в соответствии с АДИКМ, в задачах распознавания и синтеза речи.

Главное допущение метода линейного предсказания состоит в том, что речевой отсчет на выходе голосового тракта $s[n]$ может быть предсказан по линейной комбинации своих предыдущих значений и значению сигнала $u[n]$

$$s[n] = -\sum_{k=1}^{P} a[k] \cdot s[n-k] + G \cdot u[n], \tag{4.24}$$

где G – коэффициент усиления; P – порядок линейного предсказателя. В этом случае *передаточная функция предсказателя* соответствует передаточной функции рекурсивного фильтра

$$H(z) = \frac{S(z)}{U(z)} = \frac{G}{1 + \sum_{k=1}^{P} a[k] \cdot z^{-k}}. \tag{4.25}$$

Определение коэффициентов линейного предсказания речи имеет прямое отношение к спектральному анализу, основанному на использовании АР-модели. Вместе с тем, использование модели, приведенной на рис.4.1, вносит некоторую специфику. Поэтому рассмотрим оценивание ЛПК речи подробнее.

Так как отсчеты возбуждающей последовательности $u[n]$ неизвестны, то последовательность $s[n]$ может быть предсказана только по своим предыдущим значениям

$$\tilde{s}[n] = -\sum_{k=1}^{P} a[k] \cdot s[n-k] . \qquad (4.26)$$

Ошибка предсказания в этом случае будет равна

$$d[n] = s[n] - \tilde{s}[n] = s[n] + \sum_{k=1}^{P} a[k] s[n-k] . \qquad (4.27)$$

Определим коэффициенты $a[k]$ таким образом, чтобы сумма квадратов ошибок предсказания была минимальна

$$E = \sum_{n} d^2[n] = \sum_{n} \left[s[n] + \sum_{k=1}^{P} a[k] \cdot s[n-k] \right]^2 . \qquad (4.28)$$

Для минимизации (4.28) найдем частные производные (4.28) по $a[k]$ и приравняем их к нулю

$$\frac{\partial E}{\partial a[k]} = 0 , \qquad 1 \le k \le P . \qquad (4.29)$$

В результате получим систему уравнений

$$\sum_{k=1}^{P} \hat{a}[k] \sum_{n} s[n-k] \cdot s[n-m] = -\sum_{n} s[n] s[n-m] , \qquad 1 \le m \le P , \qquad (4.30)$$

где $\hat{a}[k]$ - оценки коэффициентов $a[k]$.

В общем случае суммирование в (4.30) должно выполняться по всем значениям n. Однако на практике суммирование по n в уравнении (4.30) выполняют для ограниченного числа отсчетов $s[n]$, чтобы соблюдалось условие стационарности $s[n]$. Для этого ограничивают последовательность $s[n]$ с помощью окна $w[n]$

$$s'[n] = \begin{cases} s[n] \cdot w[n] & , \ 0 \le n \le N-1, \\ 0 & \text{для других } n \end{cases} . \qquad (4.31)$$

Тогда систему уравнений (4.30) можно переписать в виде

$$r[m] = -\sum_{k=1}^{P} \hat{a}[k] r[m-k] \quad , \qquad 1 \le m \le P , \qquad (4.32)$$

где

$$r[m] = \sum_{n=-\infty}^{\infty} s'[n]s'[n+m] \tag{4.33}$$

автокорреляционная функция ограниченной последовательности $s'[n]$.

Так как автокорреляционная функция является четной, т.е. $r[m]=r[-m]$, то (4.32) можно записать в матричной форме

$$\begin{bmatrix} r[0] & r[1] & r[2] & \cdots & r[P-1] \\ r[1] & r[0] & r[1] & \cdots & r[P-2] \\ \vdots & \vdots & & & \\ r[P-1] & r[P-2] & & \cdots & r[0] \end{bmatrix} \begin{bmatrix} \hat{a}[1] \\ \hat{a}[2] \\ \vdots \\ \hat{a}[P] \end{bmatrix} = - \begin{bmatrix} r[1] \\ r[2] \\ \vdots \\ r[P] \end{bmatrix}. \tag{4.34}$$

Матричное уравнение (4.34) имеет структуру аналогичную уравнению (2.35) и может быть решено с помощью рекурсивного алгоритма Левинсона – Дарбина. В соответствии с этим алгоритмом решение для предсказателя m-го порядка получается на основе решения для предсказателя m-1 порядка. Формально алгоритм определяется следующими соотношениями [4,14,15,17]:

$$E^{(0)} = r[0], \tag{4.35}$$

$$k_m = -\frac{r[m] + \sum_{j=1}^{m-1} \alpha_j{}^{(m-1)} \cdot r[m-j]}{E^{(m-1)}}, \tag{4.36}$$

$$\alpha_m{}^{(m)} = k_m, \tag{4.37}$$

$$\alpha_j{}^{(m)} = \alpha_j{}^{(m-1)} + k_m \alpha_{m-j}{}^{(m-1)}, \quad 1 \le j \le m-1 \tag{4.38}$$

$$E^{(m)} = (1-k_m{}^2)E^{(m-1)}. \tag{4.39}$$

Уравнения (4.35 – 4.39) решаются рекурсивно для m=1, 2, …,P. Отметим, что для m=1 параметр $\alpha_1{}^{(1)} = k_1 = r[1]/r[0]$ и $E^{(1)} = (1-k_1{}^2)r[0]$. Для конечного решения порядка P коэффициенты линейного предсказания будут равны

$$\hat{a}[j] = \alpha_j{}^{(P)}, \quad 1 \le j \le P. \tag{4.40}$$

Коэффициенты k_m называются *коэффициентами отражения*. $E^{(m)}$ представляет сумму квадратов ошибки предсказания для предсказателя m-го порядка. Автокорреляционную функцию последовательности $s'[n]$ оценивают на основе соотношения:

$$r[m] = \frac{1}{N-m} \sum_{n=0}^{N-1-m} s'[n] \cdot s'[n+m] . \qquad (4.41)$$

Линейное предсказание речи можно использовать для определения частотной характеристики голосового тракта

$$H(e^{j\omega T_0}) = \frac{G}{1 + \sum_{k=1}^{P} \hat{a}[k] e^{-jk\omega T_0}} . \qquad (4.42)$$

Данная характеристика соответствует медленно меняющейся составляющей кратковременного спектра речевого сигнала. На рис. 4.10 представлен кратковременный спектр речевого сигнала и АЧХ голосового тракта, вычисленная с помощью (4.42). Порядок фильтра P=28. На графике АЧХ хорошо представлены форманты.

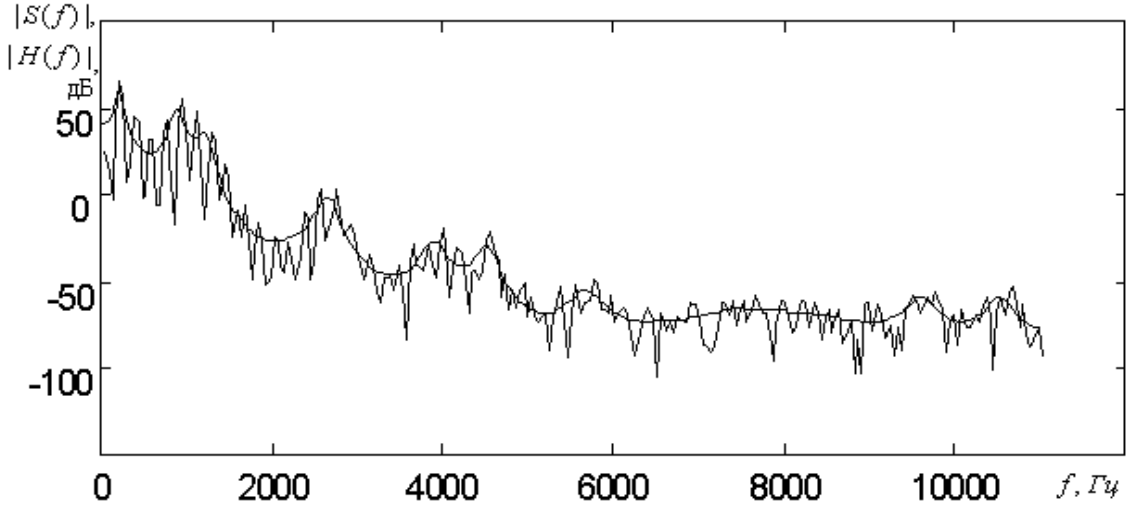


Рис. 4.10. Кратковременный спектр и АЧХ голосового тракта

Недостатком рассмотренного метода определения ЛПК является необходимость вычисления матрицы автокорреляций. Кроме этого, если вычисленные значения ЛПК применяются при синтезе речи в соответствии со схемой, показанной на рис.4.1, то возникают вопросы обеспечения устойчивости цифрового рекурсивного фильтра высокого порядка.

В настоящее время развит класс методов, которые оценивают ЛПК непосредственно по отсчетам последовательности s[n] и которые лучше приспособлены для решения задач синтеза речи. Эти методы базируются на использовании лестничного фильтра [17].

Рассмотрим алгоритм Левинсона–Дарбина. Параметры $\alpha_k^{(m)}$ представляют коэффициенты предсказывающего фильтра *m*-го порядка. Определим передаточную функцию

$$A^{(m)}(z) = 1 + \sum_{k=1}^{m} \alpha_k{}^{(m)} z^{-k} \quad . \tag{4.43}$$

Эта передаточная функция соответствует *инверсному фильтру* и является обратной по отношению к передаточной функции предсказателя (4.25). В соответствии с (4.27) на вход инверсного фильтра поступает речевой сигнал $s[n]$, а на выходе формируется ошибка предсказания. Ошибка предсказания для предсказателя m-го порядка будет равна

$$d^{m}[n] = s[n] + \sum_{k=1}^{m} \alpha_k{}^{(m)} s[n-k] \quad . \tag{4.44}$$

Найдем z-преобразование (4.44). Тогда

$$D^{(m)}(z) = A^{m}(z) S(z) \quad . \tag{4.45}$$

Подставив (4.38) в (4.43), получим

$$A^{(m)}(z) = 1 + \sum_{k=1}^{m} \left(\alpha_k{}^{(m-1)} + k_m \alpha_{m-k}{}^{(m-1)} \right) \cdot z^{-k} \quad . \tag{4.46}$$

Отсюда получаем рекурсивное выражение для вычисления $A^{(m)}(z)$

$$A^{(m)}(z) = A^{(m-1)}(z) + k_m z^{-m} A^{(m-1)}(z^{-1}) \quad . \tag{4.47}$$

Подставляя (4.47) в (4.45), получаем выражение для ошибки предсказания

$$D^{(m)}(z) = A^{(m-1)}(z) S(z) + k_m \cdot z^{-m} A^{(m-1)}(z^{-1}) S(z) \quad . \tag{4.48}$$

Первый член в (4.48) соответствует ошибке предсказания для предсказателя (m-1)-го порядка. Для второго члена в (4.48) без k_m введем обозначение

$$B^{(m)}(z) = z^{-m} A^{(m)}(z^{-1}) S(z) \quad . \tag{4.49}$$

Выражению (4.49) соответствует разностное уравнение

$$b^{(m)}[n] = s[n-m] + \sum_{k=1}^{m} \alpha_k{}^{(m)} s[n+k-m] \quad . \tag{4.50}$$

Данное уравнение соответствует *обратному предсказанию*, т.е. оно позволяет предсказать отсчет $s[n\text{-}m]$ по предстоящим отсчетам $s[(n\text{-}m)+k]$ (рис.4.11).

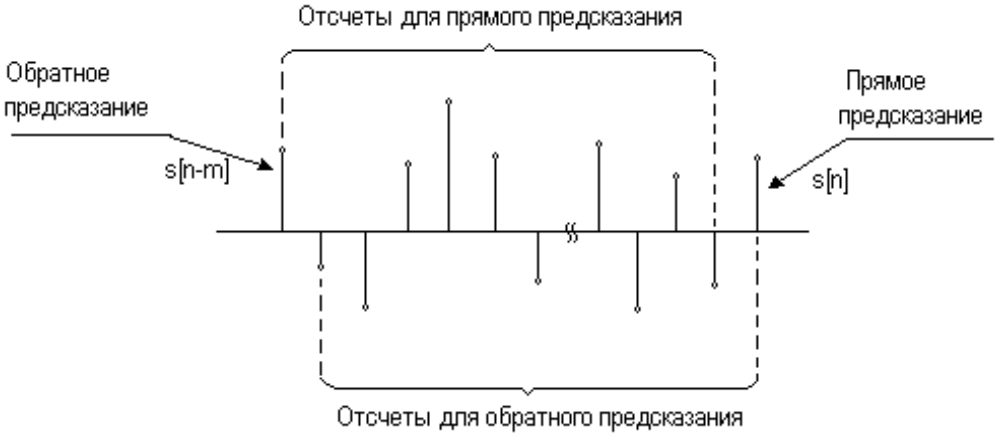


Рис. 4.11. Прямое и обратное предсказание

Сравнивая (4.50) и (4.44), отмечаем, что $b^m[n]$ соответствует ошибке обратного предсказания. Таким образом, ошибка прямого предсказания (4.48) может быть представлена в виде

$$d^{(m)}[n] = d^{(m-1)}[n] + k_m b^{(m-1)}[n-1]. \qquad (4.51)$$

Выполнив аналогичные преобразования для (4.50), получим симметричное выражение для ошибки обратного предсказания

$$b^{(m)}[n] = b^{(m-1)}[n-1] + k_m \alpha^{(m-1)}[n]. \qquad (4.52)$$

Уравнения (4.51) и (4.52) являются рекуррентными и определяют ошибки предсказаний для предсказателя m-го порядка через ошибки предсказания для предсказателя (m-1)-го порядка. При этом для предсказателя нулевого порядка

$$d^0[n] = b^0[n] = s[n] \quad . \qquad (4.53)$$

Уравнения (4.51) и (4.52) соответствуют *лестничному фильтру* и могут быть представлены в виде структурной схемы, изображенной на рис.4.12.

Коэффициенты отражения k_m могут вычисляться в соответствии с уравнениями (4.35 – 4.39). Однако имеется и иная возможность. В [11,36] показано, что коэффициенты отражений можно вычислять через ошибки предсказания $d^m[n]$ и $b^m[n]$ в соответствии с соотношением

$$k_m = \frac{2\sum_{n=0}^{N-1} d^{(m-1)}[n]\, b^{(m-1)}[n-1]}{\sum_{n=0}^{N-1}\left(d^{(m-1)}[n]\right)^2 + \sum_{n=0}^{N-1}\left(b^{(m-1)}[n-1]\right)^2} \quad . \qquad (4.54)$$

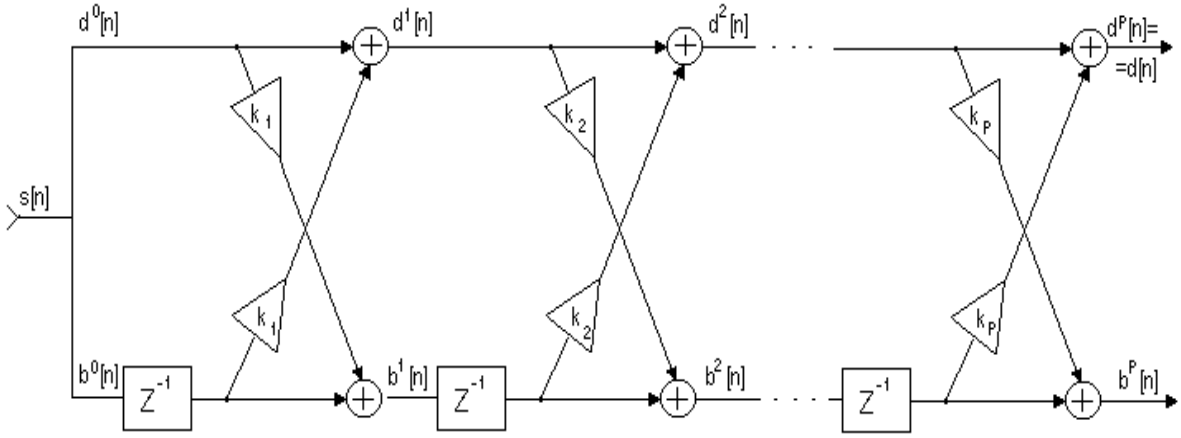


Рис.4.12. Структурная схема лестничного фильтра

Выражение (4.54) является нормированной корреляционной зависимостью и показывает степень корреляции между ошибкой прямого предсказания и ошибкой обратного предсказания. Поэтому коэффициенты k_m иногда называют коэффициентами частной корреляции. Уравнение (4.54) может использоваться вместо уравнения (4.36) при оценивании коэффициентов линейного предсказания.

Оценки коэффициентов k_m лестничного фильтра, вычисленные с помощью (4.54), будут находиться в диапазоне $-1 \le k_m \le 1$. Достоинством лестничного фильтра является низкая чувствительность к шумам округления. Соответственно коэффициенты лестничного фильтра допускают более грубое квантование. Благодаря этому гарантируется получение устойчивого фильтра.

Важное отличие оценивания ЛПК с использованием лестничного фильтра от автокорреляционного подхода состоит в том, что коэффициенты k_m оцениваются непосредственно по речевому сигналу без вычисления автокорреляционной функции.

Обратим внимание на то, что лестничный фильтр, изображенный на рис.4.12, в соответствии с (4.45) имеет передаточную функцию *A(z)*. При этом *A(z)* является обратной по отношению к передаточной функции предсказывающего фильтра (4.25). Поэтому, при подаче на вход лестничного фильтра речевого сигнала $s[n]$, на выходе формируется сигнал $u[n]$, т.е. функция возбуждения. Иными словами, ошибка предсказания $d[n]$ соответствует функции возбуждения.

4.7 Сжатие речевых сигналов

Наиболее важными областями применения рассмотренных способов обработки речевых сигналов являются системы низкоскоростной пере-

дачи речи и компьютерные мультимедийные системы. Непосредственное кодирование речи по методу ИКМ требует значительных объемов памяти для хранения речевых сигналов. Поэтому при обработке речи в компьютерных системах не обойтись без сжатия речевых сигналов.

Многие методы сжатия речевых сигналов основаны на линейном предсказании речи. В частности, линейное предсказание используется при сжатии речи по методу АДИКМ. Стандарт G.726, определяющий алгоритмы АДИКМ, устанавливает для данного типа сжатия речевых сигналов нижнюю скорость передачи 16 Кбит/с.

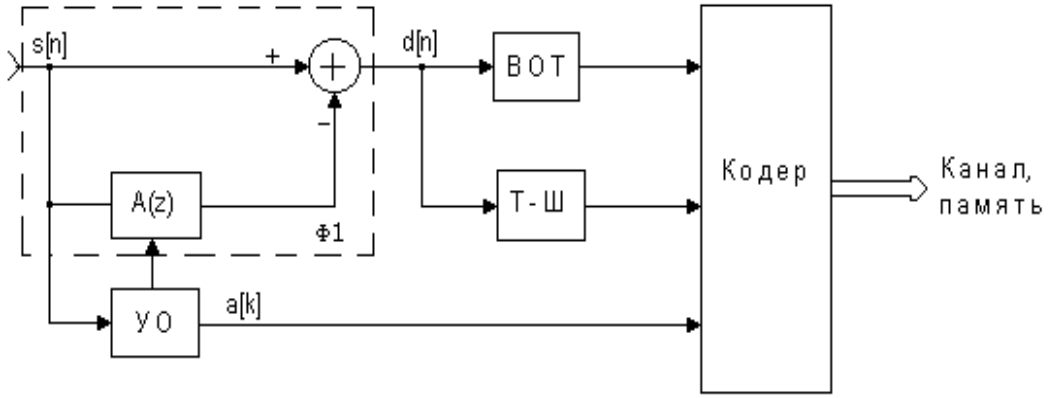
Дальнейшее снижение скорости передачи возможно при использовании схем анализ-синтез речи, учитывающих особенности цифровой модели формирования речи (рис.4.1). Применяют два варианта таких схем – без обратной связи и с обратной связью [36].

На рис.4.13,а приведена схема сжатия речи без обратной связи, основанная на анализе по методу линейного предсказания и синтезе речевого сигнала в соответствии с моделью, представленной на рис.4.1. Здесь речевой сигнал $s[n]$ разбивается на сегменты длительностью 20-30 мс. На каждом из сегментов с помощью устройства оценивания (УО) определяются коэффициенты линейного инверсного фильтра-анализа Ф1 10-го порядка. Кроме этого, на этапе сжатия с помощью выделителя основного тона (ВОТ) и анализатора тон-шум (Т-Ш) определяются соответствующие параметры функции возбуждения. В кодере выполняется кодирование коэффициентов фильтра и параметров функции возбуждения, которые затем передаются по каналу связи или сохраняются в памяти.

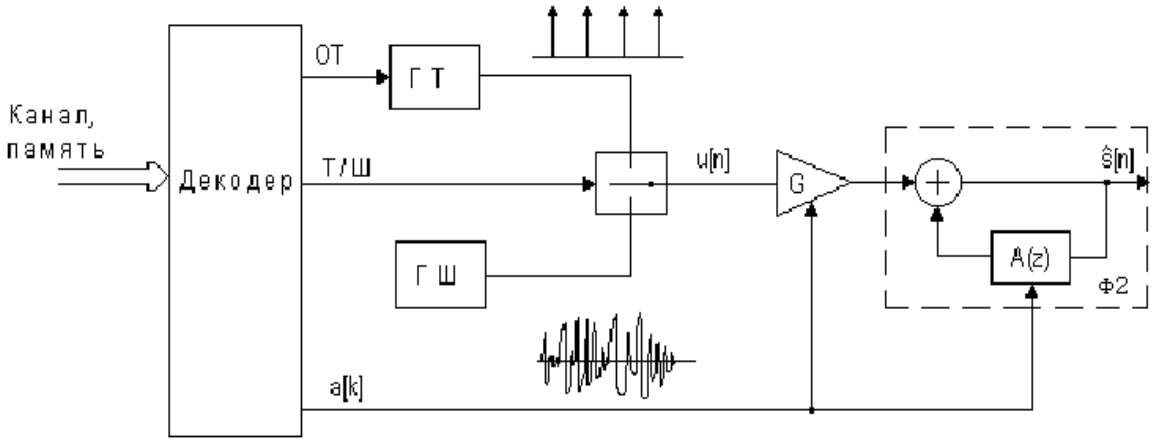
В восстанавливающем устройстве (рис.4.13,б) сначала происходит декодирование коэффициентов фильтра и параметров функции возбуждения, а затем выполняется синтез речевого сигнала $\hat{s}[n]$ в соответствии с моделью, изображенной на рис. 4.1. Для этого в зависимости от значения признака тон-шум (ТШ) на вход фильтра-синтеза Ф2 подается сигнал либо с выхода генератора тона (ГТ), либо с выхода генератора шума (ГШ). В технике связи устройство, выполняющее сжатие и восстановление речевых сигналов по приведенной схеме, называют *вокодером.* Для кодирования периода основного тона используют 6 бит, для коэффициента усиления - 5 бит, для признака тон/шум – 1 бит, для коэффициента усиления - 5 бит, для коэффициентов линейного предсказания - 8-10 бит. С учетом того, что для каждого сегмента речи оценивается десять коэффициентов предсказания, получим 97-117 бит на один сегмент. Скорость передачи при длительности сегмента 30 мс составит примерно 3000 бит/с.

В схеме, изображенной на рис. 4.13,б, параметры возбуждения (частота основного тона, признак тон/шум, форма сигнала возбуждения) формируются без учета их влияния на качество синтезированной речи.

Поэтому восстановленная речь воспринимается как механическая и не обеспечивает узнаваемости голоса.



а)



б)

Рис. 4.13. Сжатие речевых сигналов в схеме без обратных связей

Для повышения натуральности речи используется схема анализа-синтеза с обратной связью (рис.4.14). В этой схеме возбуждающая последовательность формируется путем минимизации ошибки восстановления речевого сигнала, т.е. разности между исходным речевым сигналом $s[n]$ и восстановленным сигналом $\hat{s}[n]$. Восстановленный речевой сигнал формируется с помощью фильтров Ф1 и Ф2, на вход которых подается сигнал с выхода генератора функции возбуждения (ФВ). Фильтр Ф1 учитывает квазипериодические свойства вокализованных участков речи, а фильтр Ф2 моделирует формантную структуру речи. Инверсный фильтр, соответствующий фильтру Ф1, является фильтром долговременного предсказания, а инверсный фильтр, соответствующий фильтру Ф2, называется фильтром кратковременного предсказания.

Фильтр долговременного предсказания описывается передаточной функцией

$$P_L(z) = 1 - A_L(z), \tag{4.55}$$

где $A_L(z) = az^{-\tau}$ и τ - задержка, соответствующая периоду основного тона, равная 20-150 интервалам дискретизации. Если на вход фильтра долговременного предсказания подать сигнал ошибки кратковременного предсказания $d_K[n]$, то в соответствии с (4.55) ошибка долговременного предсказания $d_Д[n]$ будет равна

$$d_Д[n] = d_K[n] - ad_K[n-\tau] \ . \tag{4.56}$$

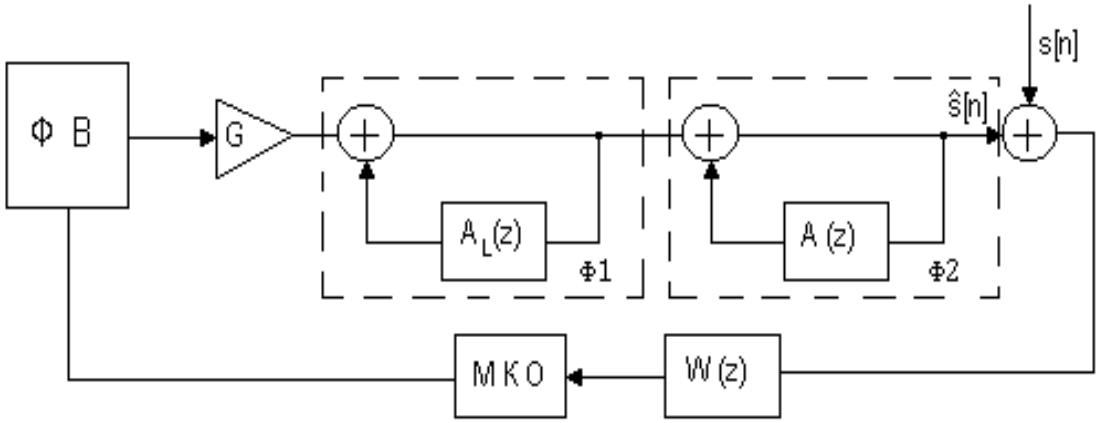


Рис. 4.14. Сжатие речевых сигналов в схеме анализ-синтез

Данная ошибка по своим свойствам близка к белому шуму с нормальным законом распределения. Это упрощает формирование сигнала возбуждения, так как при синтезе последовательности $\hat{s}[n]$ ошибка долговременного предсказания выступает в роли сигнала возбуждения.

Фильтр с передаточной функцией W(z) (рис. 4.14) позволяет учесть особенности слухового восприятия человека. Для человека шум наименее заметен в частотных полосах сигнала с большими значениями спектральной плотности. Этот эффект называют маскировкой (см. §4.8). Фильтр W(z) учитывает эффект маскировки и придает ошибке восстановления различный вес в разных частотных диапазонах. Вес выбирается так, чтобы ошибка восстановления маскировалась в полосах речевого сигнала с высокой энергией.

Принцип работы схемы, изображенной на рис.4.14, состоит в выборе функции возбуждения (ФВ), минимизирующей квадрат ошибки (МКО) восстановления.

Существует несколько различных способов формирования функции возбуждения: многоимпульсное, регулярно-импульсное и векторное (кодовое) возбуждение [36]. Соответствующие алгоритмы представляют многоимпульсное (MLPC), регулярно-импульсное (RPE-LPC) и *линейное предсказание с кодовым возбуждением* (code excited linear prediction -

CELP). MLPC использует функцию возбуждения, состоящую из множества нерегулярных импульсов, положение и амплитуда которых выбирается так, чтобы минимизировать ошибку восстановления. Алгоритм RPE-LPC является разновидностью MLPC, когда импульсы имеют регулярную расстановку. В этом случае оптимизируется амплитуда и относительное положение всей последовательности импульсов в пределах сегмента речи. CELP представляет способ, который основывается на векторном квантовании. В соответствии с этим способом из кодовой книги возбуждающих последовательностей выбирается квазислучайный вектор, который минимизирует квадрат ошибки восстановления. Кодовая книга используется как на этапе сжатия речевого сигнала, так и на этапе его восстановления. Для восстановления сегмента речевого сигнала необходимо знать номер соответствующего вектора возбуждения в кодовой книге, параметры фильтров $A_L(z)$ и $A(z)$, коэффициент усиления G. Восстановление речевого сигнала по указанным параметрам выполняется в декодере только с помощью элементов, входящих в верхнюю часть схемы, изображенной на рис.4.14.

В настоящее время применяется несколько стандартов, основывающихся на рассмотренной схеме сжатия:

1) RPE-LPC со скоростью передачи 13 Кбит/с используется в качестве стандарта мобильной связи в Европейских странах;

2) CELP со скоростью передачи 4,8 Кбит/с. Одобрен в США федеральным стандартом FS-1016. Используется в системах скрытой телефонной связи;

3) VCELP со скоростью передачи 7,95 Кбит/с (vector sum excited linear prediction). Используется в цифровых сотовых системах в Северной Америке. VCELP со скоростью передачи 6,7 Кбит/с принят в качестве стандарта в сотовых сетях Японии;

4) LD-CELP (low-delay CELP) одобрен стандартом МККТТ G.728. В данном стандарте достигается небольшая задержка примерно 0,625 мс (обычно методы CELP имеют задержку 40-60 мс), используются короткие векторы возбуждения и не применяется фильтр долговременного предсказания с передаточной функцией $A_L(z)$.

Необходимо отметить, что рассмотренные методы сжатия речи, использующие линейное предсказание с кодовым возбуждением, хорошо приспособлены для работы с речевыми сигналами в среде без шумов. В случае шумового воздействия на речевые сигналы синтезированная речь имеет плохое качество. Поэтому в настоящее время разрабатывается ряд методов линейного предсказания с кодовым возбуждением для использования в шумовой обстановке (ACELP, CS-CELP).

4.8 Сжатие аудиосигналов

4.8.1 Психоакустическая модель восприятия звука

По сравнению с речевыми сигналами аудиосигналы характеризуются более широким частотным диапазоном (10-22000Гц), большим динамическим диапазоном, большей изменчивостью спектральных свойств, многоканальностью (стерео звук). Частота дискретизации аудиосигналов обычно равна 44,1 кГц. Данная частота используется при обработке звука в цифровых магнитофонах, при записи звука на компакт диски. Каждый отсчет звукового сигнала представляется 16-ти разрядным двоичным кодом. В случае стерео звука это создает цифровой поток со скоростью передачи $2 \times 44{,}1 \times 16 \times 1000 = 1{,}41$ Мбит/с. На практике из-за наличия дополнительной служебной информации скорость передачи оказывается существенно выше. Так, при считывании отсчетов звукового сигнала с компакт диска формируется цифровой поток со скоростью 4,32 Мбит/с. Ограниченная пропускная способность каналов связи, ограниченная емкость запоминающих устройств требуют сжатия аудиосигналов.

Сжатие аудиосигналов основано на учете особенностей слуха человека. Ухо человека воспринимает звуковой сигнал в изолированном от внешних шумов помещении, если звуковое давление превышает некоторый порог, называемый *абсолютным порогом слышимости* (АПС). Зависимость АПС от частоты аппроксимируется нелинейной функцией (рис. 4.15) [36]:

$$T(f) = 3{,}64(f/1000)^{-0,8} - 6{,}5e^{-0,6(f/1000-3,3)^2} + 10^{-3}(f/1000)^4 \text{ дБ}. \quad (4.57)$$

Как отмечалось выше, внутренне ухо человека выполняет кратковременный спектральный анализ на основе механизма критических частотных полос (табл. 4.1). Если в критической частотной полосе находится несколько спектральных составляющих, то наблюдается явление частотного маскирования. Суть его состоит в том, что спектральная составляющая (тон) с низким уровнем может не прослушиваться, если в этой же критической полосе имеется спектральная составляющая звука с более высоким уровнем (рис.4.15). Спектральная составляющая с высоким уровнем повышает порог слышимости и маскирует присутствие составляющих с низким уровнем. Повышенный порог слышимости называется порогом маскирования. Все спектральные составляющие, уровень которых ниже порога маскирования, не прослушиваются.

Зависимость порога маскирования от частоты определяется уровнем и частотой маскирующей составляющей, а также уровнями и частотами маскируемых спектральных составляющих. Порог маскирования имеет максимальное значение на частоте маскирующей спектральной составляющей и снижается при уменьшении или увеличении частоты. При

уменьшении частоты относительно частоты маскирующей составляющей порог маскирования снижается быстрее, чем при её увеличении, т.е. составляющие, частоты которых лежат выше частоты маскирующей частотной составляющей, маскируются в большей степени.

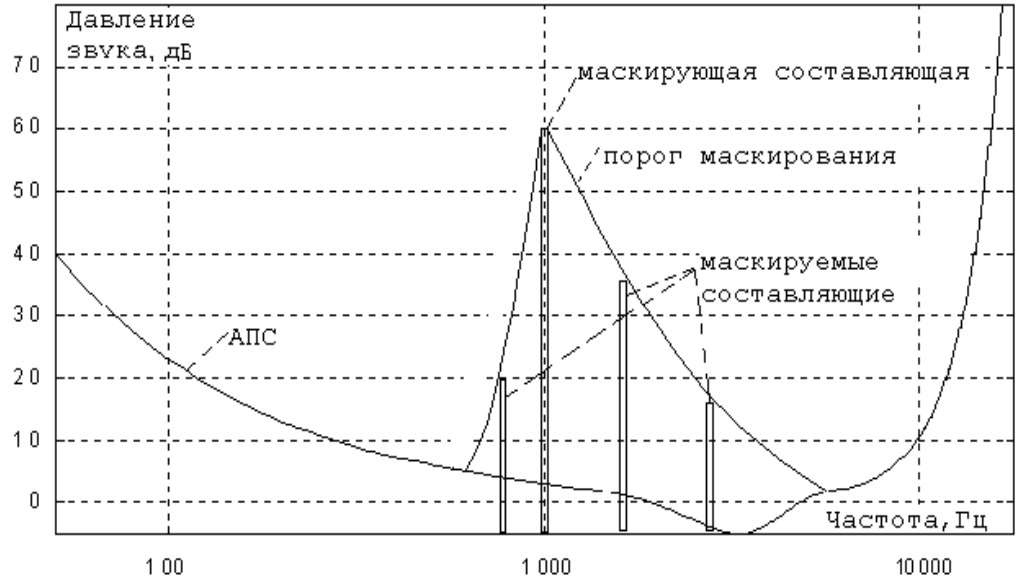


Рис.4.15. Абсолютный порог слышимости и порог маскирования

Качественный характер изменения порога маскирования показан на рис. 4.16. Разность между уровнем маскирующей составляющей и порогом маскирования называется *отношением сигнал-маска* (ОСМ). ОСМ имеет максимальное значение для левой границы критической полосы (точка А на рис. 4.16). На рис. 4.16 также показан уровень шума *m*-разрядного квантователя. Если отношение сигнал-шум (ОСШ) больше, чем ОСМ, то шумы квантования будут маскироваться основной спектральной составляющей. Разность ОШМ(*m*)=ОСМ-ОСШ(*m*) называется *отношением шум-маска* (ОШМ). Шумы квантования не будут прослушиваться, если ОШМ будет иметь отрицательные значения. Учет этого обстоятельства позволяет уменьшить количество двоичных разрядов, отводимых для представления спектральных составляющих.

На рис. 4.15 показана маскирующая составляющая только в одной критической полосе. На практике наблюдается присутствие маскирующих спектральных составляющих во многих полосах. Такие составляющие оказывают взаимное влияние на пороги маскирования в соседних полосах. В результате этого образуется суммарный порог маскирования. Частотные составляющие, уровень которых ниже суммарного порога маскирования, не воспринимаются на слух и, соответственно, могут быть исключены из спектра сигнала при его записи и хранении в запоминающих устройствах или при передаче по каналам связи. Это позволяет снизить требования к емкости запоминающих устройств или пропускной способности каналов связи.

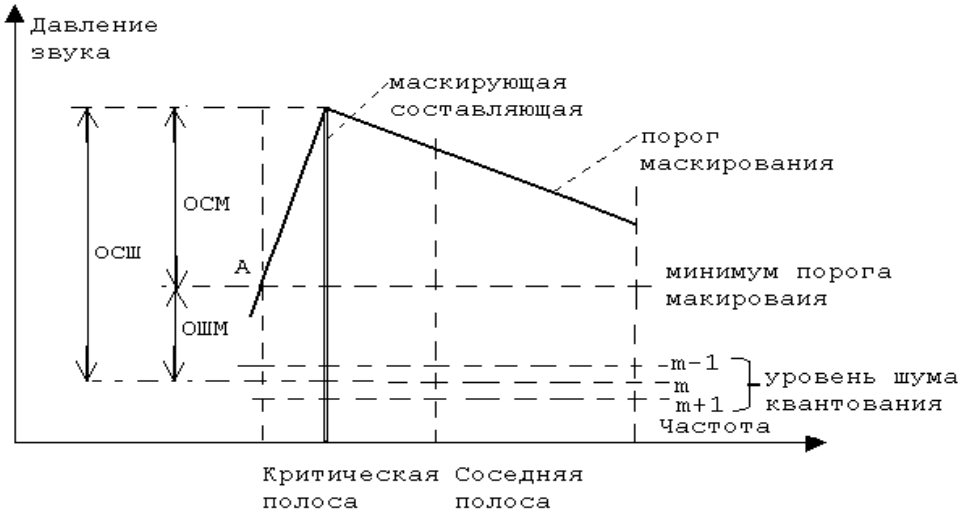


Рис.4.16. Отношение сигнал-маска

Помимо частотного маскирования, описанного выше, важную роль в восприятии звука играет временное маскирование. Временное маскирование наблюдается в том случае, когда два звука появляются через короткий интервал времени. Тогда звук, характеризующийся большим уровнем звукового давления, может частично или полностью маскировать присутствие звука с меньшим звуковым давлением, даже в том случае, если маскируемый звук предшествует маскирующему. Это связано с переходными процессами во внутреннем ухе, которые формируются маскирующим звуком.

Продолжительность эффекта временного маскирования до восприятия маскирующего звука равна примерно 5 мс. Продолжительность маскирующего эффекта после окончания восприятия маскирующего звука равна 50-300 мс. Временное маскирование используется для подавления предварительного эхо, которое наблюдается при восстановлении сжатых звуковых сигналов. Предварительное эхо прослушивается до появления восстановленного звукового сигнала и обусловлено шумами квантования, распространяющими свое действие на весь блок данных, подвергшийся сжатию, в том числе и на ту его часть, которая предшествовала звуку.

Сжатие аудиосигналов с учетом рассмотренных особенностей восприятия звуков человеком называют *перцептивным кодированием*, т.е. кодированием, основанным на восприятии.

4.8.2 Перцептивное кодирование аудиосигналов

Алгоритмы перцептивного кодирования обеспечивают снижение корреляционной зависимости между отсчетами аудиосигнала, а также исключение элементов сигнала, не воспринимаемых человеком на слух. Достигается это вычислением слабо коррелированных спектральных составляющих аудиосигналов и адаптивным квантованием только тех составляющих, уровень которых выше порога маскирования.

Применяются два основных подхода для построения перцептивных кодеров: кодирование с использованием гребенки полосно-пропускающих фильтров и кодирование с применением ортогональных преобразований [33]. Оба подхода основаны на выполнении кратковременного спектрального анализа входных аудиосигналов.

В первом случае входной аудиосигнал поступает на блок фильтров анализа, состоящий из M полосовых фильтров, каждый из которых пропускает соответствующую частотную полосу аудиосигнала. Полосы пропускания фильтров-анализа частично перекрываются и выбираются так, чтобы сумма выходных полосовых сигналов соответствовала исходному аудиосигналу. Выходной сигнал каждого из фильтров децимируется с коэффициентом M и подвергается адаптивному квантованию. Алгоритмы адаптивного квантования учитывают свойства слуховой системы человека. В декодере выполняются обратные процессы: полосовые сигналы сначала интерполируются с коэффициентом M, а затем восстанавливаются с помощью блока полосовых фильтров-синтеза. Из-за перекрытия полос пропускания фильтров в процессе децимации могут возникать искажения. Данные искажения обусловлены шумами наложения. Шумы наложения могут быть подавлены, если в качестве полосовых фильтров использовать квадратурные зеркальные фильтры [3,33]. Однако последующее квантование выходных сигналов фильтров не позволяет исключить их полностью. Это снижает динамический диапазон восстановленного сигнала.

Во втором случае на основе ортогональных преобразований вычисляются слабо коррелированные спектральные коэффициенты, которые затем подвергаются адаптивному квантованию. Обычно в качестве ортогонального преобразования применяется дискретное преобразование Фурье или дискретное косинусное преобразование (ДКП), каждое из которых может вычисляться на основе алгоритма БПФ. В декодере реализуются обратные процессы. При этом конечная длина преобразований приводит к появлению краевых эффектов в восстановленном аудиосигнале. Краевые эффекты проявляются в точках сопряжения блоков отсчетов, полученных с помощью обратных ортогональных преобразований. Для снижения уровня краевых эффектов применяется модифицированное дискретное косинусное преобразование (МДКП). МДКП основано на 50%-ом перекрытии блоков отсчетов входного аудиосигнала при выполнении спектрального анализа.

Часто спектральный анализ и последующий синтез аудиосигналов выполняется с помощью гибридного банка фильтров, который использует как линейную фильтрацию, так и ортогональные преобразования. Основным преимуществом такого подхода является возможность выполнения спектрального анализа с большей разрешающей способностью, что позволяет полнее учесть особенности слуховой системы человека. Высо-

кая разрешающая способность спектрального анализа аудиосигналов может быть получена с помощью гребенки фильтров, выходные сигналы которых подвергаются дальнейшему анализу с помощью МДКП. Такой подход используется при сжатии аудиосигналов в соответствии со стандартами, предложенными экспертной группой MPEG (Moving Picture Expert Group).

Одним из отрицательных эффектов, который проявляется при использовании рассмотренных подходов к сжатию аудиосигналов, является предварительное эхо. Предварительное эхо прослушивается, когда в пределах одного сегмента аудиосигнала, подвергающегося ортогональному преобразованию, имеется участок молчания и участок со значительным уровнем звука (рис. 4.17,а). Наличие участка с высоким уровнем звука приводит к росту шага квантования и, соответственно, росту ошибки квантования.

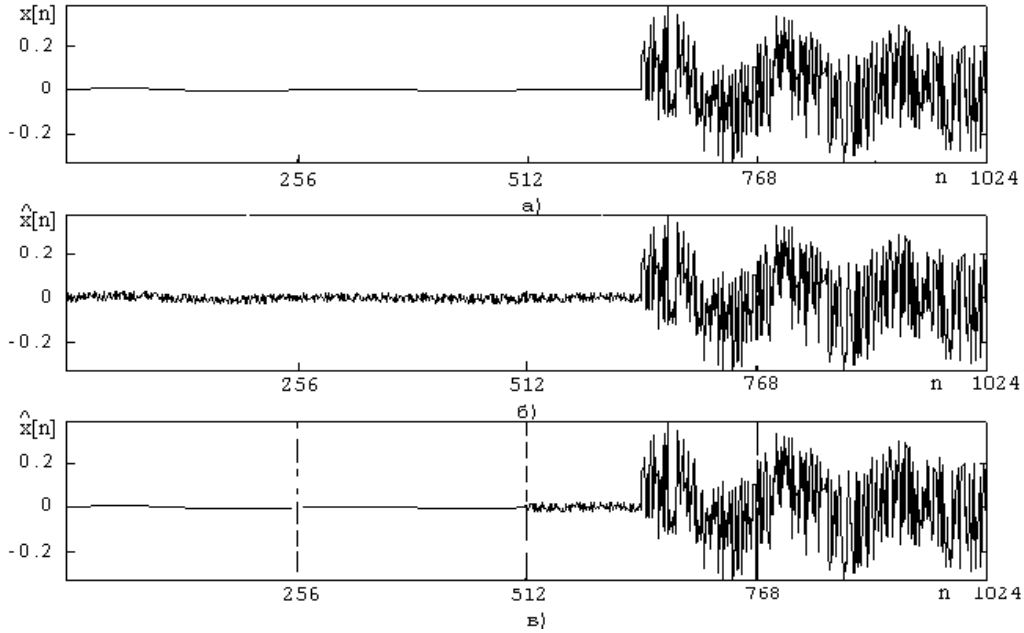


Рис.4.17. Предварительное эхо и переключение окон

При выполнении обратного ортогонального преобразования полученная ошибка квантования распространяет свое действие на весь сегмент (рис. 4.17,б N=1024). Предварительное эхо может быть частично подавлено за счет явления временного маскирования, если продолжительность эхо невелика. Продолжительность эхо можно сократить, если выполнять ортогональные преобразования коротких сегментов аудиосигналов (рис.4.17,в N=256). Однако преобразование аудиосигналов на коротких сегментах увеличивает общее число анализируемых сегментов и, следовательно, приводит к росту скорости передачи. Решение проблемы состоит в динамическом изменении (переключении) длины сегментов. Типовое значение длины сегмента лежит в диапазоне от N=64 до N=1024.Указанное переключение возможно на основе анализа стационарности участка аудиосиг-

нала. Короткие сегменты используются на нестационарных участках аудиосигнала. На стационарных участках выполняется переключение на длинные сегменты.

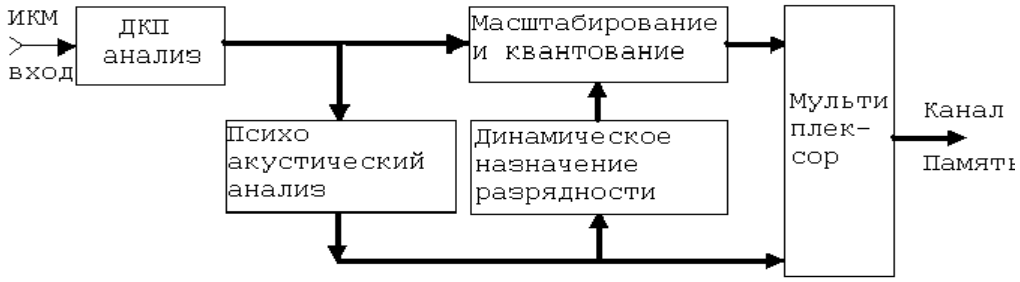


Рис.4.18. Перцептивный кодер на основе ДКП

На рис. 4.18 изображена упрощенная схема перцептивного кодера аудиосигналов, основанного на применении ДКП. В рассматриваемой схеме входной сигнал разбивается на сегменты, длиной 512 отсчетов. Каждый сегмент отображается в частотную область с помощью ДКП. Затем выполняется масштабирование и адаптивное квантование коэффициентов ДКП на основе психоакустического анализа ДКП-спектра. Спектральные составляющие, уровни которых ниже порога маскирования (рис.414), исключаются из дальнейших преобразований. Квантование спектральных составляющих, с уровнем выше порога маскирования, осуществляется в соответствии со значением ОСМ. Чем больше значение ОСМ в соответствующей полосе, тем большее число двоичных разрядов отводится для представления спектральных составляющих данной частотной полосы. Эта операция выполняется в блоке динамического назначения разрядности. Кодированные значения спектральных составляющих и информация психоакустического анализа мультиплексируются и передаются в канал связи или запоминаются в устройстве хранения информации. В декодере выполняются обратные операции.

4.8.3 Сжатие аудиосигналов в соответствии со стандартами ISO/MPEG

Экспертная группа MPEG совместно с международной организацией стандартов ISO разработала серию стандартов для сжатия звука и изображений, обозначаемых MPEG-1, MPEG-2, MPEG-4.

Стандарт MPEG-1 поддерживает передачу цифрового видео со скоростью 1,2 Мбит/с (с качеством аналогового видеомагнитофона) и цифрового звука со скоростями 32-192 Кбит/с (моно режим) и 64-384 Кбит/с (стерео режим с качеством компакт-диска).

Стандарт MPEG-2 (IS 13818) в своей видео части поддерживает передачу высококачественного видео (включая телевидение высокой четкости) со скоростью от 3 до 15 Мбит/с, а в звуковой части – низкоскоро-

стное кодирование многоканального звука со скоростью 64 Кбит/с и ниже. MPEG-2 поддерживает от 2-х до 5-ти широкополосных звуковых каналов.

Стандарт MPEG-4 предназначен для применения в области мультимедиа систем. MPEG-4 предусматривает широкий набор средств, позволяющих выполнять передачу звуковых и речевых сигналов с низкими скоростями от 2 до 64 Кбит/с. В звуковую часть стандарта включены следующие возможности: параметрическое низкоскоростное кодирование речи (2-10 Кбит/с); среднескоростное кодирование речи на основе схем анализ-синтез (6-16 Кбит/с); кодирование аудио и речевых сигналов с использованием гибридных банков фильтров и ортогональных преобразований (ниже 64 Кбит/с). Стандарт MPEG-4 обеспечивает высокую степень сжатия и охватывает широкий диапазон звуковых сигналов от HF стерео звука до кодирования речи, включая синтез речи. На основе этого стандарта обеспечивается сопряжение звуковых каналов различного качества. Стандарт предусматривает дополнительные функции, такие как временное масштабирование, управление частотой основного тона, доступ к базам данных.

Стандартом MPEG-1 предусматривается сжатие аудиосигналов с помощью систем трех уровней сложности. Системы сжатия верхних уровней сложности включают возможности систем нижних уровней. Структурная схема системы сжатия аудиосигналов, соответствующая первому и второму уровню сложности, показана на рис. 4.19. Система может функционировать с частотами дискретизации 48 кГц, 44,1 кГц, 32 кГц.

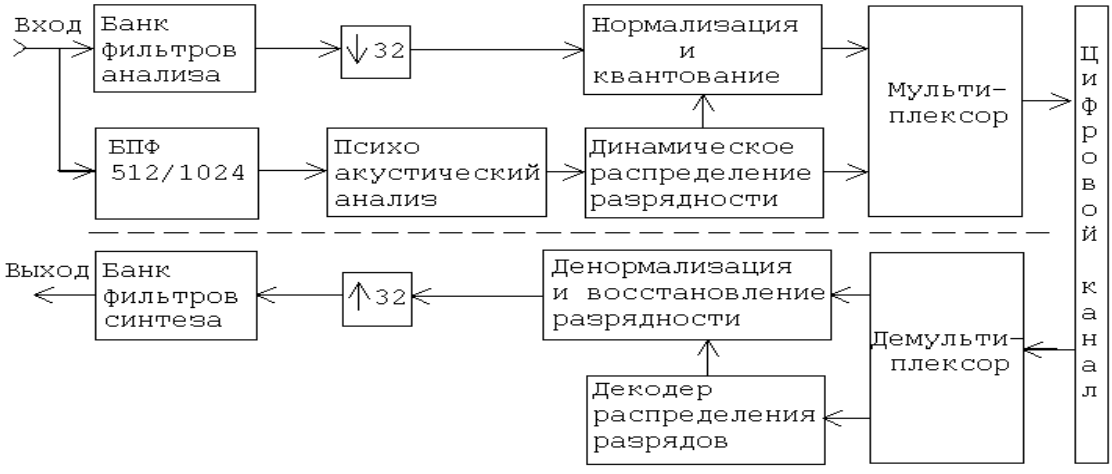


Рис. 4.19. Кодер и декодер аудиосигналов MPEG-1 (уровни I и II)

Входной аудиосигнал с помощью банка анализирующих фильтров разделяется на 32 полосовых сигнала. Фильтры обладают полифазной структурой [3] и их импульсная характеристика задается 512 коэффициентами. При частоте дискретизации 48 кГц полоса пропускания каждого фильтра равна 24000/32=750 Гц. Спектральный анализ, выполняемый с

помощью такого набора фильтров, характеризуется недостаточным частотным разрешением в области нижних частот, так как полоса пропускания одного фильтра перекрывает несколько критических частотных полос (табл.4.1). Выходные сигналы фильтров децимируются с коэффициентом 32, т.е. частота дискретизации понижается до критического значения, равного 1500 Гц. Импульсная характеристика $h_k[n]$ фильтра, пропускающего сигналы k -ой частотной полосы, получается путем умножения импульсной характеристики $h[n]$ ФНЧ, выступающего в качестве фильтра-прототипа, на модулирующую функцию, которая обеспечивает требуемое смещение полосы пропускания ФНЧ:

$$h_k[n] = h[n]\cos[(2k-1)/(2M) + \varphi[k]], \qquad (4.58)$$

где M=32; k=1,2,…,32; n=1,2,…,512.

АЧХ низкочастотного фильтра прототипа характеризуется ослаблением 3 дБ в полосе пропускания и 96 дБ в полосе подавления. Переходные полосы фильтров существенно перекрываются. В результате этого возможно появление шумов наложения [33], которые исключаются соответствующим выбором фазового сдвига $\varphi[k]$.

Выходные сигналы фильтров разбиваются на блоки по 12 отсчетов (уровень I) и 36 отсчетов (уровень II) в каждой полосе. Над каждым блоком выполняются операции нормализации и квантования. Нормализация выполняется таким образом, чтобы максимальный отсчет в каждом блоке имел единичное значение. В пределах блока сохраняется одно и то же распределение уровней квантования по частотным полосам.

Для целей психоакустического анализа и определения суммарного порога маскирования используется БПФ на 512 точек (уровень I) и на 1024 точки (уровень II). В каждом из указанных случаев входной сигнал взвешивается с помощью окна Ханна. В блоке психоакустического анализа для каждой маскирующей составляющей вычисляется индивидуальный порог маскирования. Суммарный порог маскирования получается путем сложения индивидуальных порогов и абсолютного порога слышимости. Отношение сигнал-маска определяется как разность между уровнем маскирующей составляющей в полосе и минимальным значением суммарного порога маскирования в этой же полосе.

Число уровней квантования для каждой полосы определяется в блоке динамического распределения разрядности, в котором используется итерационный алгоритм минимизирующий отношение шум-маска в каждой частотной полосе. Это обеспечивает получение минимально допустимой разрядности представления полосовых сигналов.

В кодере, относящемся к уровню I, квантованию подвергается блок из 12 отсчетов в каждой частотной полосе. При частоте дискретизации 48 кГц это соответствует 8мс аудиосигнала (384 отсчета). В кодере, относящемся к уровню II, квантованию подвергаются одновременно три блока

по 12 отсчетов (т.е. 36 отсчетов), что соответствует 24 мс звукового сигнала. При этом для всех трех блоков используется одно и тоже распределение разрядности квантования по частотным полосам. Однако коэффициенты нормализации для каждого из блоков вычисляются индивидуально. В зависимости от изменчивости коэффициентов нормализации в канал могут передаваться 1,2 или 3 значения коэффициентов нормализации. Квантованные значения указанных блоков совместно с информацией о распределении разрядов по полосам передаются в цифровой канал. Когда уровни спектральных составляющих в частотной полосе оказываются меньше суммарного порога маскирования в данной полосе, отсчеты соответствующего полосового сигнала не передаются, т.е. для квантования сигнала данной полосы выделяется 0 бит.

В декодере выполняется восстановление выходных сигналов полосно-пропускающих фильтров и синтез исходного аудиосигнала. Для этого, сначала выполняется приведение квантованных значений отсчетов в каждой полосе к исходной разрядности и их масштабирование. Выполняется это на основе имеющейся информации о распределении разрядности квантования по частотным полосам. Если в какой –либо частотной полосе разряды квантования не распределялись, то соответствующие отсчеты считаются нулевыми. Когда кодер и декодер реализуются в виде одного устройства, банк фильтров может быть общим для кодера и декодера.

В кодере и декодере звуковых сигналов, относящемся к уровню III, с целью повышения разрешающей способности по частоте и более полного учета особенностей слухового восприятия человека применяется гибридный банк фильтров. В этом случае предусматривается выполнение МДКП и адаптивное переключение длины анализируемых сегментов аудиосигнала для подавления предварительного эхо.

Высокое частотное разрешение достигается выполнением дополнительного спектрального анализа выходных сигналов полосовых фильтров. Такой анализ выполняется с помощью 6-ти или 18-ти точечного МДКП. Так как МДКП использует 50%-ое перекрытие анализируемых сегментов, то максимальное число спектральных компонент по всем полосам равно 32x18=576. Следовательно, разрешающая способность спектрального анализа будет равна 24000/576=41,67 Гц. Короткое МДКП (6 точек) применяется в случае необходимости подавления предварительного эхо.

Результаты МДКП подвергаются неравномерному квантованию и кодированию Хаффмана. Чтобы шумы квантования были ниже суммарного порога маскирования, распределение разрядности по полосам выполняется на основе итерационной схемы анализ-синтез.

Рассмотренная схема сжатия звуковых сигналов позволяет снизить скорость передачи аудиосигнала до 14 Кбит/с.

Глава 5
ОБРАБОТКА ИЗОБРАЖЕНИЙ

5.1 Устройство компьютерных систем обработки изображений

Под обработкой изображений понимают совокупность компьютерных методов обработки цифровых сигналов изображений. Типовая схема обработки изображений показана на рис.5.1. В общем случае система может обрабатывать сигналы нескольких каналов, выбор которых осуществляется видеомультиплексором. Для цветных изображений используются три цветовых компоненты: красная *R (Read),* зеленая *G (Green)* и синяя *B (Blue)* . Вместо трех цветовых составляющих могут быть использованы три чернобелых камеры или другие источники.

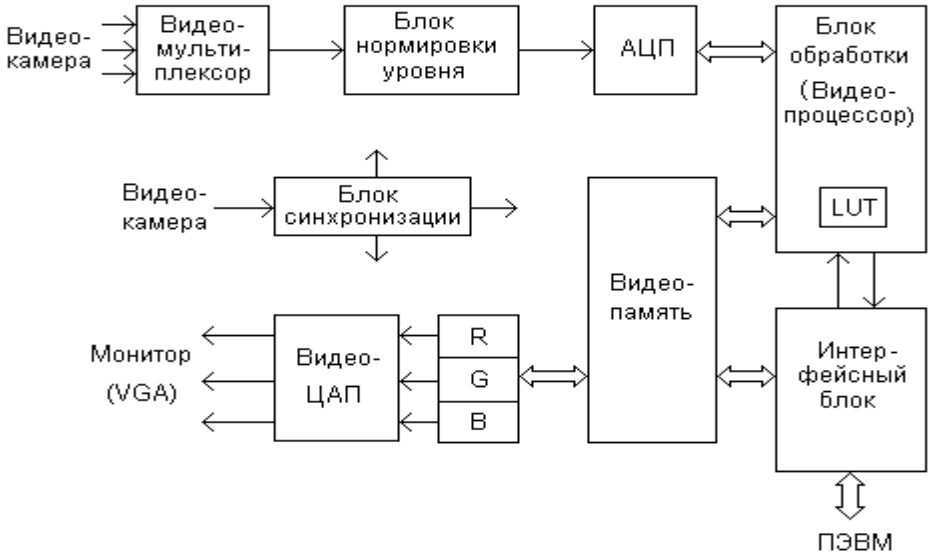


Рис. 5.1. Типовая схема устройства обработки изображений

С целью согласования уровней видеосигналов со входом последующего аналого-цифрового преобразователя применяется блок нормировки уровня, представляющий собой программно управляемый усилитель. В качестве стандартных источников видеоинформации используются аналоговые или дискретные камеры. Последние построены на базе матрицы светочувствительных элементов на основе приборов с зарядовой связью (ПЗС) и в зарубежной литературе называются CCD-камерами.

На рис.5.2. приведены параметры и вид видеосигнала одной строки телевизионного черно-белого изображения, соответствующие европейским нормам, регламентируемым международным консультативным комитетом по радио *(CCIR)*. Кроме информации о распределении яркости вдоль строки в сигнале присутствуют импульсы строчной и кадровой синхронизации, которые определяют начало строки и кадра текущего изображения. Эти импульсы могут быть выделены схемой синхронизации видеокарты и использованы для формирования импульсов внутренней синхронизации. Основные параметры телевизионных сигналов в соответствии с европейскими *(CCIR)* и американскими (*RS*-170) нормами приведены в табл.5.1.

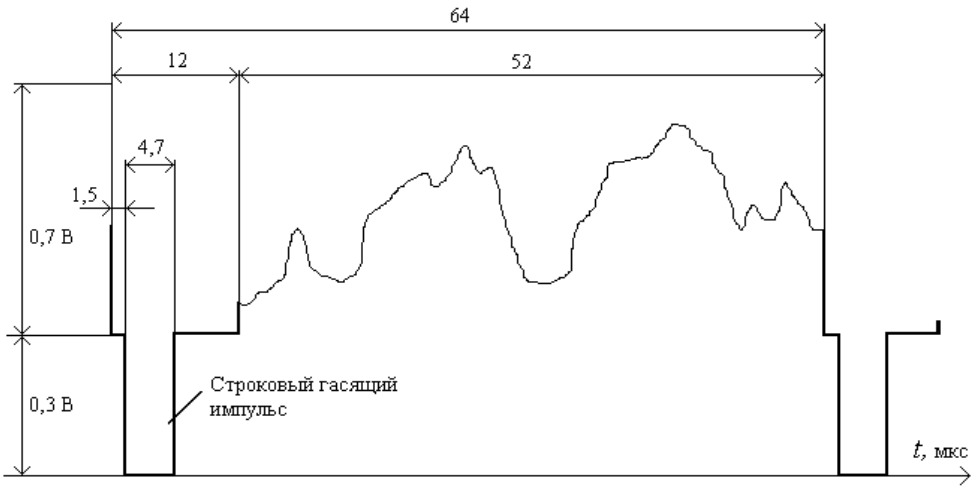


Рис.5.2. Параметры телевизионного черно-белого изображения

Видеосигнал преобразовывается в цифровую форму аналого-цифровым преобразователем (АЦП), в котором происходит дискретизация и квантование видеосигналов. Стандартом установлена частота дискретизации 10 МГц и 256 градаций уровней квантования (8-битовое слово). При этом изображение представляет собой матрицу размером 512×512 элементов изображения (пикселов). Новое поколение систем обработки изображений является более гибким. В них количество строк развертки и частота дискретизации могут быть заданы программно, что позволяет оцифровывать нестандартные видеосигналы, поступающие, например, с электронных микроскопов, ультразвуковых и инфракрасных датчиков, а также обрабатывать видеосигналы европейского *(CCIR)* и американского (*RS*-170) стандартов. Так например, в видеокарте *VISTA*, работающей в стандарте CCIR может быть установлено количество строк 592 при числе элементов в строке 578. Частота дискретизации при этом составляет 14,1875 МГц. При работе в стандарте *RS*-170 эти параметры соответственно равны 756, 486 и 14,3182 МГц.

При цифровой обработке необходимо учитывать следующие особенности оцифровки телевизионных изображений:

1) из 576 видимых строк изображения европейского стандарта *CCIR* большинство видеокарт оцифровывают только 512. Причем стандартом не установлено, какие из 576 строк используются, в связи с чем в видеокартах различных производителей выделяются не одни и те же группы строк;

2) пикселы имеют не квадратную, а прямоугольную форму, что оказывает влияние на процесс съема и фильтрации данных;

3) в системах с *CCD*-камерами тактовая частота дискретизации не соответствует тактовой частоте считывания с чувствительных элементов камеры. Поэтому элементы изображения не соответствуют чувствительным элементам, что приводит к появлению вертикальных мешающих полос на экране монитора. В связи с эти необходимо предусматривать внешнюю синхронизацию генератора строчной развертки синхроимпульсами *CCD*-камеры;

4) обычные видеокамеры работают в режиме чересстрочной развертки, при которой полное изображение представляется в виде двух полукадров, состоящих из нечетных и четных строк соответственно. Оба полукадра считываются с камеры последовательно с интервалом 20 мс. В связи с этим оба полукадра имеют разную освещенность, что приводит к мерцанию неподвижных изображений. Поэтому при обработке следует использовать только один полукадр, либо обеспечить одинаковую освещенность четного и нечетного полукадров.

Таблица 5.1 – Параметры телевизионных сигналов

Параметры	CCIR	RS-170
Частота строк fc, кГц	15,625	15,7343
Время строчной развертки tc, мкс	64	63,5556
Нерабочее время t, мкс	11,8414	10,7556
Частота кадров fк, Гц	50	59,9401
Время кадровой развертки tк, мс	20	16,6833
Нерабочее время t, мс	1,5360	1,2711
Количество строк всего	625	525
Количество активных строк	576	485

Оцифрованное изображение до занесения его в видеопамять подвергается, как правило, предварительной обработке, в ходе которой осуществляются гомогенные операции над пикселами. Для этой цели используется одна или несколько (8 - 16) таблиц перекодировки, так называемых *Look-Up-Table (LUT).*

Видеопамять является одним из важнейших блоков системы обработки изображений, в которой хранятся один или несколько кадров изображения. Каждая ячейка памяти служит для хранения 8 или 16 битов. Большинство систем позволяют запоминать от 2-х до 16 кадров, в связи с чем емкость видеопамяти должна быть в пределах от 0,5 до 16 Мбайт. Видеопамять имеет сложную внутреннюю структуру, обеспечивающую одновременную запись и отображение изображения. Использование двухпортового интерфейса *(dual-ported memory)* обеспечивает одновременный доступ к памяти как со стороны видеопроцессора, так и со стороны процессора ПЭВМ. Гибкая система адресации памяти позволяет автоматически осуществлять инкремент/декремент адресов строк и столбцов после каждого цикла доступа к ней, а также оперативно получать или модернизировать адреса. В ряде видеокарт предусмотрена возможность сегментации памяти, при которой осуществляется доступ к строке или некоторому фрагменту изображения. Последнее является очень важным для повышения быстродействия выполнения операций дискретно-косинусного преобразования при сжатии изображений.

В вычислительном блоке (видеопроцессоре) производятся все операции над изображениями или отдельными его фрагментами. Типичная последовательность операций видеопроцессора сводится к чтению изображения из одной или нескольких страниц видеопамяти, выполнения операций над изображением и записи результатов обработки в другую страницу памяти. Основными операциями, выполняемыми в вычислительном блоке, являются:

- изменение яркости и контрастности;
- усреднение;
- пороговая обработка;
- выделение контуров;
- различная фильтрация;
- преобразование координат;
- сжатие и восстановление изображений и др.

Главной особенностью видеопроцессора является высокая скорость обработки, которая достигается за счет реализации ряда операций аппаратным способом, выполнения нескольких операций одновременно, использования таблиц, применения специализированных процессоров.

В выходном блоке осуществляется преобразование выбранных из видеопамяти цифровых данных в видеосигнал и выдачи его на монитор или видеорекордер. Как правило, блок состоит из трех идентичных каналов, в которых производится обработка красной *R*, зеленой *G* и синей *B* составляющих цветного видеоизображения. Значения отсчетов каждой из цветовых компонент могут быть перекодированы с помощью соответствующих *LUT*-таблиц. Использование таблиц позволяет модифицировать выходное

изображение не изменяя исходного изображения, хранящегося в видеопамяти. В трехканальном цифроаналоговом преобразователе (ЦАП) видеоданные преобразовываются в аналоговую форму. При этом, как правило, в зеленую компоненту дополнительно добавляются синхронизирующие сигналы.

Блок сопряжения с ПЭВМ позволяет осуществлять отладку программного обеспечения системы видеобработки, оперативно изменять параметры (*LUT*-таблицы) и режимы работы.

5.2 Операции над изображениями

Изображение в компьютере представляется в виде матрицы элементов изображения размером $M \times N$, где M- количество строк, а N-количество элементов в строке (Рис.5.3). Элемент изображения (пиксел) f с координатами (x,y) имеет свое значение интенсивности (яркости) $u=f(x,y)$. Все возможные значения изображения $f(x,y)$ могут принимать G значений уровня, $G \geq 2$. Нулевому уровню G соответствует черный цвет, а уровню G-1 – белый. На практике чаще всего используется 256 уровней яркости G, т.к. для отображения любого допустимого значения уровня достаточно 8 бит. При $G > 2$ изображение называется полутоновым, а при $G = 2$ – бинарным.

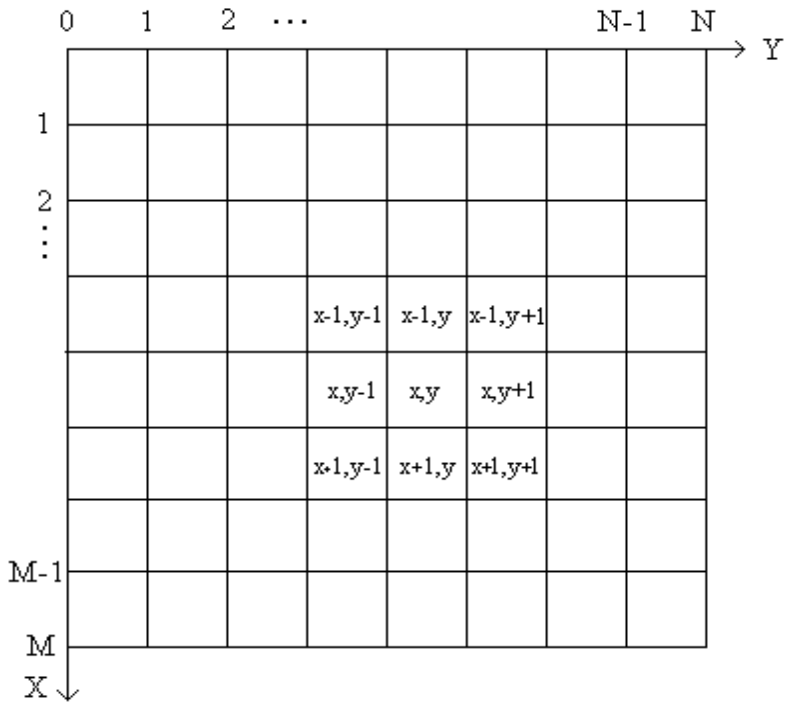


Рис.5.3. Матрица элементов изображения

Несколько бинарных или полутоновых изображений могут обрабатываться как многоканальное изображение. Такие изображения имеют место при обработке видеоданных в различных спектральных диапазо-

нах или в системах цветного изображения. При совместной обработке изображений $f_1, f_2, \ldots, f_n$ значения функции изображения определяется как некоторый вектор

$$\boldsymbol{f}(x,y) = (f_1(x,y), f_2(x,y), \ldots, f_c(x,y)),$$

где $f_i(x,y)$ – значение яркости в i - м канале, для $i = 1,2, \ldots, c$; c-количество каналов. Рассмотрим основные характеристики цифровых изображений. Среднее значение интенсивности изображения, вычисляемое по формуле

$$m_f = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) = \sum_{i=0}^{255} P_i u_i,$$

где u_i-i-й уровень серого; P_i-вероятность появления i–го уровня. Среднее значение в определенной степени свидетельствует о том, является ли изображение более светлым или более темным. Однако по нему нельзя судить о контрастности, которая определяется разностью между максимальным и минимальным уровнями пикселов всего изображения.

Для характеристики контрастности изображения используют среднее значение квадрата отклонения величины текущего пиксела от среднего значения интенсивности

$$q_f = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f(x,y) - m_f]^2. \qquad (5.1)$$

С целью упрощения вычисления среднего квадрата отклонения обычно пользуются формулой

$$q_f = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f^2(x,y) - m_f.$$

В случае многоканальных изображений для характеристики средней интенсивности и среднего квадрата отклонения применяют соответственно векторы

$$\boldsymbol{m_f} = (m_{f0}, m_{f1}, \ldots . m_{fC-1})$$

и

$$\boldsymbol{q_f} = (q_{f0}, q_{f1}, \ldots, q_{fC-1}).$$

Для характеристики распределения уровней полутонового изображения применяется гистограмма относительных частот появления определенных уровней в изображении. Относительная частота интенсивности g–го уровня пиксела определяется как

$$P_f(g) = n_g / (MN); \text{ при } g = 0,1, \ldots, G\text{-}1,$$

где n_g - количество элементов изображения с уровнем g. Очевидно, что $0 \le P_f(g) \le 1$.

Статистика распределения полутонов и, следовательно, гистограммы являются основой для так называемых точечных операций, которые могут изменять это распределение. В зависимости от вида гистограммы различают (Рис.5.4,а) - унимодальные (одногорбовые), (Рис.5.4,б) - бимодальные (двугорбовые) и мультимодальные распределения (Рис.5.4,в). Вид распределения играет существенную роль в задачах изменения контрастности и сегментации изображений.

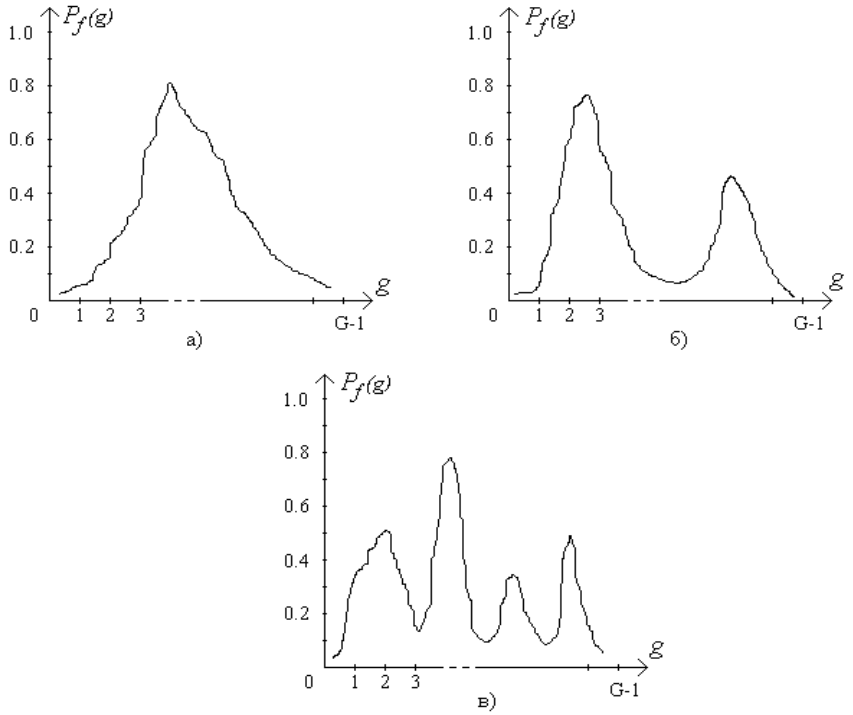


Рис.5.4. Гистограммы распределения уровней элементов изображения: а) унимодальное; б) бимодальное; в) мультимодальное

Для оценки среднего информационного содержания изображения используется его энтропия

$$H = -\sum_{g=0}^{G-1} \left[P_f(g) \log_2 P_f(g) \right]$$

Если в состав некоторого фрагмента изображения входят элементы с одинаковым уровнем, то его называют *гомогенным*. Для гомогенного изображения справедливо соотношение

$P_f(g) = 1$ при $g = g_k$;
$P_f(g) = 0$ в противном случае.

При выполнении многих операций над изображениями часто используются точки цифрового изображения, окружающие данный пиксел.

Множество точек, соседних по отношению к заданной, называются *окрестностью NS*i. Здесь буквой i обозначается количество точек окрестности. Так например, 4-точечная окрестность *NS*4 (Рис.5.5,а) определяется выражением

$$NS4(x,y) = \{(x-1,y),(x+1,y),(x,y+1),(x,y-1)\}, \qquad (5.2)$$

а 8-точечная окрестность (Рис.5.5,б) NS8-выражением

$$NS8(x,y) = \{(x-1,y+1),(x-1,y)(x-1,y-1),(x,y+1),$$
$$(x,y-1),(x+1,y+1),(x+1,y),(x+1,y-1)\}.$$

	x,y-1	
x-1,y	x,y	x+1,y
	x,y+1	

а)

x-1,y-1	x,y-1	x+1,y-1
x-1,y	x,y	x+1,y
x-1,y+1	x,y+1	x+1,y+1

б)

Рис.5.5. Иллюстрация видов окрестностей опорной точки:
а) 4-точечная окрестность; б) 8-точечная окрестность

Приведенные выше определения окрестностей не включают центральной точки *(x,y)*. В ряде случаев целесообразно включать в рассмотрение и ее. Тогда имеем

$$NS5(x,y) = NS4(x,y) \cup \{(x,y)\}$$

и

$$NS9(x,y) = NS8(x,y) \cup \{(x,y)\}.$$

Аналогично можно определить и другие окрестности.

Множество всех входящих в окрестность пикселов вместе с центральной точкой представляют собой фрагмент изображения, который называют *окном*. Функция окна *h* задана в пределах его размеров $m \times n$, где *m*- число строк, а *n*-число столбцов окна, при этом $m<M$ и $n<N$.

$$h = \{(i,j):\ 1 \le i \le m\ \wedge\ 1 \le j \le n\}. \qquad (5.3)$$

Форма окна может быть произвольной, хотя на практике в большинстве случаев используется квадратная или прямоугольная форма. Для определения местоположения окна задаются координаты опорной точки *(I,J),* выбранной внутри окна. В случае прямоугольного окна в качестве опорной выбирают точку с координатами

$$I = Integer\left(\frac{m+1}{2}\right) \quad \text{и} \quad J = Integer\left(\frac{n+1}{2}\right). \tag{5.4}$$

Позиционирование окна *h* можно описать посредством движения опорной точки *(I,J).* Позиционирование окна *h* на относительную опорную точку *(r,s)* соответствует сдвигу окна на вектор (*r-I, s-J*),

Операцией над изображением называется процедура, в результате которой исходное изображение *f(x,y)* преобразуется в другое - *f'(x,y).* Аналитически она записывается в виде

$$f' = T(f), \tag{5.5}$$

где *T* - оператор преобразования исходного изображения. Операция над изображением называется локальной, если *f'(x,y)* зависит от значения *f(x,y)* в точке *(x,y)* множества *NS(x,y).* Если *NS(x,y)={(x,y)}* и *f'(x,y)=T(f(x,y)),* то *T* называется точечной операцией. Она является специальным случаем локальной.

Линейные локальные операции можно представить в виде

$$f'(x,y) = \sum_{(i,j)\in NS(x,y)} f(x,y)h(i,j), \tag{5.6}$$

где *h(i,j)*- локальный оператор (или оператор окна), который определен в окрестности *NS(x,y).* Поведение функции *h* в окне *NS* может быть задано пользователем путем указания ее значений в клетках окна. Выражение (5.6) представляет собой дискретную свертку *f* и *h.*

В области пространственных частот это выражение может быть записано как произведение спектров *f* и *h*, т.е. ***F*×*H*.** Поэтому локальный оператор можно рассматривать как фильтр пространственных частот.

На эффективность алгоритмических решений существенное влияние оказывает представление данных. Наиболее целесообразным является *иконическое* представление, при котором отсчеты полутонового или бинарного изображения, образующие двумерный массив, хранятся в памяти в векторной форме. Для изображения *f* размером *M*×*N* значение элемента изображения *f(x,y)* с координатами *(x,y)* заносится в память с индексом

$$(y-1)M + x, \tag{5.7}$$

где *x* – индекс столбца (слева направо), а *y* - индекс строки (сверху вниз). Строка изображения соответствует при этом обычно одной записи *(re-*

cord) файла. При $M=N$=512 и G=256 полутоновое изображение занимает в накопителе 256 кбайт. В некоторых изображениях формат видеоданных задается т.о., что первая запись резервируется для общей управляющей информации, а первая строка изображения записывается, начиная со второй записи.

Векторные (многоканальные) изображения могут быть однозначно представлены последовательностью скалярных изображений $f_1,f_2, \ldots , f_n$. Отдельные матрицы изображений $f_1,f_2, \ldots , f_n$ образуют в итоге одну трехмерную матрицу размером $M\times N\times n$, элементом которой является значение изображения $f_i(x,y)$, соответствующее координатам (x,y,i). Линейное позиционирование элементов трехмерной матрицы изображения можно осуществить несколькими способами, располагая пикселы в памяти: а) построчно; б) поканально; в) канально-строчным способом.

При построчном формате одна запись состоит из строки элементов изображения, принадлежащих одному скалярному изображению f_i. При этом в памяти последовательно располагаются n записей для отдельных строк векторного изображения. Элемент $f_i(x,y)$ изображения располагается на позиции

$$(y-1)\,nM + (i-1)M + x\ . \qquad (5.8)$$

В поканальном формате в накопитель последовательно заносятся значения элементов сначала первого канала f_1, затем второго f_2 и наконец c-го f_c. Элемент изображения $f_i(x,y)$ располагается на позиции

$$(y-1)\,MN + (y-1)M + x\ . \qquad (5.9)$$

В канально-строчном формате n–я запись состоит из M групп по n значений в каждой одноименных пикселов одной из строк векторного изображения f. При этом c канальных значений $f_1(x,y), f_2(x,y), \ldots , f_c(x,y)$ располагаются непосредственно друг за другом.

При обработке изображений из видеобуфера выбираются фрагменты изображения, которые заносятся в рабочую память, а после обработки они снова возвращаются в исходный буфер. Размер и форма фрагмента зависят от используемого алгоритма. Для обработки изображений целесообразно выделять в памяти две области данных. В первую записывается исходное изображение, а во вторую – результирующее (выходное).Такое решение позволяет производить быстрое сравнение изображений, а также применять параллельную обработку с помощью локальных операторов. Передача данных из буфера исходного изображения в рабочую память осуществляется обычно построчно.

Применяемые для операторов окна покрывают по высоте несколько строк. Поэтому при переходе от одной строки к следующей необходимо

обеспечить быстрое пополнение требуемых данных. В конце каждой строки необходимо кроме этого занести в буфер результирующего изображения обработанные значения. При ширине окна *n* строк и расположении опорной точки в середине окна необходимо вначале считать первые *n* строк в буферный накопитель *BUF(*1...*M, j), (j*=1, ... ,*n),* расположенный в рабочей памяти. При считывании видеоданных исходного изображения, попавших в окно размером *n*×*n*, значения пикселов записываются в одномерный массив, причем индекс *z* изменяется от 1 до *a*, где $a = n^2$.

На рис.5.6 показана последовательность нумерации ячеек окна, обычно принятая в литературе [29].

34	33	32	31	30	29	28
35	15	14	13	12	11	10
36	16	4	3	2	10	26
37	17	5	*a*	1	9	25
38	18	6	7	8	24	48
39	19	20	21	22	23	47
40	41	42	43	44	45	46

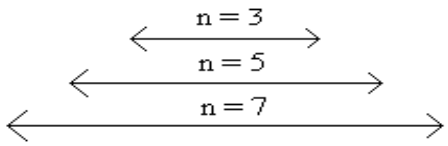


Рис.5.6. Индексация ячеек окна

5.3 Предварительная обработка информации

5.3.1 Линейная фильтрация изображений

Целью предварительной обработки видеоданных является уменьшение зашумленности информации, повышение ее контрастности, резкости, подчеркивание контуров. На этапе предварительной обработки осуществляется также фильтрации сигналов изображений.

Качество изображения во многом определяется его *контрастностью*, которая представляет собой разность между максимальным и минимальным уровнями полутонов в некоторой области изображения, например, в окне (*локальная контрастность*) или всего изображения (*глобальная контрастность*). Путем усиления контрастности можно улучшить различимость отдельных структур изображения или уменьшить искажения, вызванные влиянием условий получения изображений в оптико-электронных преобразователях.

Слабый контраст - наиболее распространенный дефект фотографических и телевизионных изображений, обусловленный ограниченностью диапазона воспроизведения яркостей, нередко сочетающийся с нелинейностью характеристики передачи уровней (градационной характеристикой). Одним из эффективных путей улучшения качества изображения является использование метода *контрастного растягивания*, при котором заданный диапазон изменения яркости входного изображения *f(x,y)* линейно растягивается на всю шкалу уровней полутонов [0 - (*G*-1)]. В результате этого расширяется диапазон изменения яркости выходного изображения *f'(x,y)* и тем самым изображение становится более контрастным. Наиболее распространенным и простейшим способом контрастного растягивания (градационной коррекции) является линейное преобразование

$$f'(x,y) = 0 \qquad \text{для } f(x,y) < u_1;$$

$$f'(x,y) = \frac{f(x,y) - u_1}{u_2 - u_1}(G-1) \qquad \text{для } u_1 \le f(x,y) \le u_2; \tag{5.10}$$

$$f'(x,y) = G - 1 \qquad \text{для } f(x,y) > u_2,$$

где u_1, u_2 - минимальное и максимальное значения яркости входного изображения. Результирующая кусочно-линейная градационная характеристика, построенная по (5.10), показана на рис.5.7,а.

Во многих случаях контрастность можно повысить, изменяя яркость каждого элемента изображения с помощью некоторого нелинейного оператора преобразования, который связывает яркость изображения на выходе с яркостью входного изображения. Математически операция преобразования представляется в виде степенной функции

$$f'(x,y) = (G-1)\left[\frac{f(x,y)}{G-1}\right]^r; \tag{5.11}$$

или S- образной функции преобразования

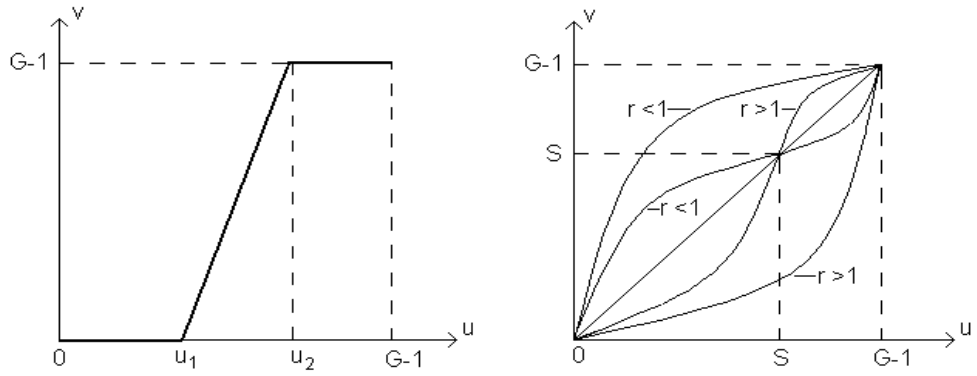
$$v_1 = (G-1)\frac{u^r}{s^{r-1}} \qquad \text{для } 0 \le u \le s;$$

$$v_2 = (G-1)\left[1 - \frac{(1-u)^r}{(1-s)^{r-1}}\right] \qquad \text{для } s \le u \le 1, \tag{5.12}$$

где $u = f(x,y) / (G-1)$ - нормированное значение функции входного изображения; $v=f'(x,y) / (G-1)$ - нормированное значение функции выходно-

го изображения; $s = S / (G\text{-}1)$ - нормированное значение точки перегиба характеристики изображения; r- показатель степенной функции.

Нелинейные характеристики преобразования показаны на рис.5.7,б. Изменяя показатель r, можно получить выпуклую ($r < 1$) или вогнутую ($r > 1$) характеристику преобразования. В первом случае при высоких уровнях яркости изображения в результате преобразования растягиваются уровни светлых элементов, а при низких уровнях - сжимаются темные элементы. Во втором случае наблюдается обратная картина. При S-образной характеристике при $r > 1$ происходит растягивание шкалы уровней серого около точки перегиба S и сжатие на обоих концах шкалы. В случае $r < 1$ происходит обратное действие.



а) б)

Рис.5.7. Характеристики преобразования изображения а) кусочно-линейная; б) нелинейные

Линейная фильтрация изображений имеет свои особенности по сравнению с фильтрацией радио- и аудиосигналов. Эти особенности связаны с двумерной природой самого изображения, которая приводит к тому, что вместо одномерных фильтров, предназначенных для фильтрации сигналов одной переменной, при фильтрации изображений применяют двумерные фильтры.

Двумерная фильтрация осуществляется с использованием оконных функций. Последовательность значений выходного изображения Q формируется путем дискретной свертки входного поля f с функцией окна $h(i,j)$, для задания которой используются локальные операторы. Фильтрация производится путем перемещения слева направо (или сверху вниз) окна (апертуры) на один пиксел. При каждом положении окна осуществляется перемножение весовых коэффициентов с соответствующими значениями исходного изображения, суммирование полученных произведений, нормирование результата суммирования и присвоение центральному элементу окна полученного значения. Эти однотипные операции многократно повторяются для каждого положения окна. Наиболее часто применяются окна размером 3×3 и 5×5. Функция окна определяется

множеством весовых значений *h(i,j),* задаваемых для каждого *(i,j)*-го элемента окна.

Пусть окно имеет размер m×n элементов, текущий элемент, которого задается координатами *(i,j)*

$i = 1,2, \ldots, m$ – текущая строка;
$j = 1,2, \ldots, n$ – текущий столбец.

Позиционирование окна на изображении определяется с помощью условного центра *(I,J)* окна (в системе координат окна). Условный центр определяется таким образом, что он при нечетных размерах окна совпадает с его центральным элементом:

$$I = \lceil (m+1)/2 \rceil \; ; \quad J = \lceil (n+1)/2 \rceil ,$$

где $\lceil x \rceil$ обозначает целую часть от числа x.

Текущее положение окна на исходном изображении f обозначим через *(r,q)*. Отклик фильтра присваивается той же точке *(r,q)* профильтрованного изображения *Q*. Массив пикселов выходного изображения *Q* формируется путем дискретной свертки входного изображения *f* и функции окна *h(i,j)*

$$Q(r,q) = \sum_{i=1}^{m} \sum_{j=1}^{n} f(r - I + i, q - J + j) h(i,j).$$

Формула справедлива лишь при условии, что функция окна не выходит за пределы исходного изображения, т.е. выполняются условия

$$I \le r \le N - m + i ,$$
$$J \le q \le M - n + j,$$

где *M, N* – размеры изображения.

Пример 5.1

Пусть имеется прямоугольное окно размером 3×3 вида

$$h_1 = 1/10 \begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{vmatrix} .$$

Тогда выходное значение фильтра определяется как

$$y = \frac{1}{10}(x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1} + x_{i,j-1} + 2x_{i,j} + x_{i,j+1} + x_{i+1,j-1} + x_{i+1,j+1}),$$

где $x_{i,j}$ – элементы входного (исходного) изображения, попавшие в окно фильтра.

При линейной фильтрации с помощью оконных функций различают два случая:

1) сумма коэффициентов *SK* оконной функции отлична от нуля. Импульсный отклик такого фильтра имеет низкочастотный характер и поэтому содержит постоянную составляющую, т.е. фоновую компоненту. Для получения этой компоненты значения коэффициентов следует делить на величину *SK*;
2) сумма коэффициентов *SK*=0. Фильтр имеет высокочастотный характер и служит для выделения верхних пространственных частот (повышения резкости изображений, выделения контуров и т.д.). Импульсный отклик содержит отрицательные составляющие, которые не могут быть представлены в результирующем изображении с положительными значениями полутонов. Поэтому при воспроизведении результирующего изображения необходимо сдвинуть уровень черного на величину *G*/2 и уменьшить динамический диапазон изменения яркости до *G*/2.

5.3.2 Подавление шумов и сглаживание

Изображение может искажаться помехами различного происхождения. Их влияние можно минимизировать, используя классические методы линейной локальной фильтрации [24]. Для подавления шума часто применяются фильтры, использующие следующие оконные операторы (Рис.5.7)

$$h_2 = 1/9 \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}; \qquad h_3 = 1/16 \begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

Рис.5.7. Шумоподавляющие операторы

Функция h_2 осуществляет усреднение по всем элементам, попавшим в окно. С помощью функции h_3, благодаря большим весам, подчеркиваются (выделяются) четырехсвязные элементы исходного изображения, т.е. горизонтальные и вертикальные линии. Если необходимо подчеркнуть диагональные линии, то следует выделить восьмисвязные элементы исходного изображения, не являющиеся четырехсвязными. Для этого целесообразно применять оконную функцию представленную на рис.5.8.

$$h_4 = 1/16 \begin{vmatrix} 2 & 1 & 2 \\ 1 & 4 & 1 \\ 2 & 1 & 2 \end{vmatrix}.$$

Рис.5.8. Шумоподавляющий оператор с выделением диагональных линий изображения

Нормирующий коэффициент оконных функций выбирается т.о., чтобы процедура подавления шума не вызывала смещение средней интенсивности обработанного изображения.

Существенным недостатком линейной фильтрации изображений является то, что наряду с уменьшением шумов одновременно происходит размывание контуров изображения. Это вызвано тем, что все элементы исходного изображения обрабатываются с одинаковым коэффициентом, т.е. линейные фильтры являются независимыми от структуры элементов и поэтому они не могут определить разницу между зашумленными и контурными элементами.

Чтобы уменьшить размывание изображения, целесообразно использовать метод селективного сглаживания. В ряде случаев, если яркость пикселов входного изображения распределена по нормальному закону, весьма эффективной при проведении предварительной обработки может оказаться сигма-фильтрация, при которой учитываются только те элементы входного изображения (находящиеся внутри окна), яркость которых находится в пределах $Ex \pm 2\,\sigma$. Здесь Ex – математическое ожидание, а σ - среднеквадратическое отклонение яркости пикселов изображения.

5.3.3 Нелинейная локальная фильтрация

Основное отличие нелинейной фильтрации от линейной заключается в том, что выходной сигнал формируется нелинейным образом от данных исходного изображения, а для рекурсивной фильтрации – нелинейным образом от отфильтрованного на предыдущих шагах изображения. В системах обработки изображения наиболее широкое применение находят два класса нелинейных фильтров. Фильтры, относящиеся к первому классу, используются для подавления шумов, а фильтры второго класса - для подчеркивания перепадов яркости. Весьма эффективным методом для подавления шумов является *медианная* фильтрация, в процессе которой сначала производится сортировка по величине пикселов, попавших в окно фильтра, а затем замена значения опорной точки *(x,y)* величиной яркости пиксела, расположенного в середине (на медиане) этого ряда. В связи с этим медианный фильтр не оказывает влияние на ступенчатые и линейно-изменяющиеся сигналы. При этом медианный фильтр подавляет импульсные выбросы исходного изображения, если длительность импульса составляет менее половины ширины окна. Так, например, если фрагмент изображения, попавший в окно фильтра, имеет вид (Рис.5.9,а), то после медианной фильтрации его опорный элемент принимает значение 39 вместо исходного 28 (Рис.5.9,б).

Для медианной фильтрации $\mathrm{med}\{f\}$ справедливо следующее правило вычисления, которое описывает суммирование константы со значением

последовательности и умножение последовательности на некоторый коэффициент c:

$$\text{med}\{c + f(k)\} = c + \text{med}\{f(k)\};$$

и

$$\text{med}\{cf(k)\} = \text{c med}\{f(k)\}.$$

51	19	47
42	28	161
33	21	39

После обработки $\Rightarrow$ med{f} = 39

51	19	47
42	39	161
33	21	39

а) б)

Рис.5.9. Иллюстрация медианной обработки фрагмента изображения

Однако не является справедливым, что медианное значение суммы двух последовательностей равно сумме их медианных значений. Поэтому медианный фильтр является нелинейным. Медианные фильтры нередко применяются итеративно, причем обработка повторяется до тех пор, пока на профильтрованном изображении не прекратятся изменения.

5.4 Подчеркивание перепадов яркости и границ

Сглаживающие фильтры подавляют верхние частотные компоненты изображения, что приводит не только к уменьшению влияния шума, но и к потере мелких объектов изображения. Во многих задачах распознавания образов необходимо выделять границы объектов, подчеркивать перепады яркостей, селектировать мелкие объекты. Для реализации таких задач используются так называемые дифференциальные операторы. В таких операторах сумма элементов оконной функции h равна нулю, что свидетельствует о том, что в зонах изображения с постоянной яркостью они дают нулевой отклик. Дифференциальные операторы используются для вычисления градиента функции. Операция взятия градиента преобразует исходное изображение в новое с подчеркнутыми границами. Классическая функция градиента имеет вид

$$\nabla f(x, y) = \frac{\partial f(x, y)}{\partial x}\cos\theta + \frac{\partial f(x, y)}{\partial y}\sin\theta.$$

Абсолютное значение функции градиента определяется как

$$S = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2},$$

а угол θ равен

$$\theta = arctg \left(\frac{\partial f / \partial y}{\partial f / \partial x} \right).$$

На практике применяют различные аппроксимации абсолютного значения градиента:

$$S = \sqrt{S_x^2 + S_y^2},$$

а для уменьшения затрат на вычисление градиента можно воспользоваться формулой

$$S = |S_x| + |S_y|$$

или

$$S = \max \left(|S_x|, |S_y| \right),$$

а направление границы определять следующим образом

$$\theta = arctg \ (S_x / S_y).$$

Здесь S_x и S_y - приращения функции изображения по направлениям x и y, которые вычисляются по формулам:

$$S_x = f(x+1,y) - f(x,y); \qquad S_y = f(x,y+1) - f(x,y) .$$

Соответствующие окна для определения первой производной в направлениях x и y изображены на рис.5.10.

$$h_x = \begin{vmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{vmatrix} \qquad h_y = \begin{vmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{vmatrix}$$

Рис.5.10. Дифференцирующие операторы по направлениям x и y

Существует много операторов, полученных из градиентных и служащих для обнаружения границ. Одним из простейших является оператор Робертса, который определяется следующим образом

$$f_p(x,y) = max[|f(x,y)-f(x+1,y)| , |f(x+1,y)- f(x,y+1)|] ,$$

или приближенно

$$f_p(x,y) = |f(x,y) - f(x+1, y+1)| + |f(x+1,y) - f(x,y+1)| .$$

Так, например, полагая, что после обработки фрагмента изображения (рис.5.11,а), который окружен нулями, с помощью оператора Робертса получим выходной фрагмент (рис.5.11,б).

0	2	2	2
0	2	2	2
0	2	2	2
0	2	2	2

а)

$\Rightarrow$

2	4	4	2
4	0	0	4
4	0	0	4
4	0	0	4

б)

Рис.5.11. Обнаружение границы посредством оператора Робертса

Кроме этого, широко применяются операторы Превита (Рис.5.12,а), Собела (Рис.5.12,б), ряд операторов Кирша (Рис.5.12,в).

$$h_x = \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{vmatrix}; \quad h_y = \begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix};$$

а)

$$h_x = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}; \quad h_y = \begin{vmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{vmatrix};$$

б)

$$h_x = \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{vmatrix}; \quad h_y = \begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix};$$

$$h_{/} = \begin{vmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{vmatrix}; \quad h_{\backslash} = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{vmatrix}.$$

в)

Рис.5.12. Градиентные операторы: а) Превита; б) Собела: в) Кирша

Рассмотренные выше дифференциальные операторы выделяют более сильно вертикальные или горизонтальные линии, т.е. они зависимы от направления.

Другим видом зависимых от направления операторов являются так называемые компасс-градиентные (Рис.5.13). Название географического направления говорит о направлении перепада, которое вызывает максимальный отклик фильтра.

север	северо-восток	восток	юго-восток
$h = \begin{vmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{vmatrix}$;	$h = \begin{vmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{vmatrix}$;	$h = \begin{vmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{vmatrix}$;	$h = \begin{vmatrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{vmatrix}$;

юг	юго-запад	запад	северо-запад
$h = \begin{vmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{vmatrix}$;	$h = \begin{vmatrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{vmatrix}$;	$h = \begin{vmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{vmatrix}$;	$h = \begin{vmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{vmatrix}$;

Рис.5.13. Компасс-градиентные операторы

Для обнаружения линий или для фокусировки изображений во многих случаях целесообразно использовать оператор Лапласа (лапласиан), который вычисляет вторую производную функции. Для непрерывных функций он представляется в виде

$$\nabla^2 = f(x,y) = \frac{\partial f^2(x,y)}{\partial x^2}\cos\theta + \frac{\partial f^2(x,y)}{\partial y^2}\sin\theta.$$

На практике лапласиан вычисляется по приближенной формуле

$$f_L(x,y) = 4f(x,y) - \{f(x,y+1) + f(x+1,y) + f(x-1,y) + f(x,y-1)\}. \qquad (5.13)$$

Оконная функция Лапласиана имеет вид, представленный на рис.5.14,а. Часто вместо нее используют также функции рис.5.14,б и рис.5.14,в.

$$h_{L1} = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix}; \quad h_{L2} = \begin{vmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{vmatrix}; \quad h_{L3} = \begin{vmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{vmatrix}.$$

а) б) в)

Рис.5.14. Оконные функции Лапласа

Оператор Лапласа обладает инвариантностью к повороту изображений, т.е. на одном и том же изображении он дает один и тот же результат независимо от ориентации этого изображения.

Лапласиан (5.13) можно представить в несколько ином виде, который позволяет более наглядно объяснить процесс улучшения резкости изображения:

$$f(x,y)-\nabla^2 f(x,y)=[f(x,y)+f(x+1,y)+f(x-1,y)+f(x,y+1)+f(x,y-1)]-5f(x,y). \quad (5.14)$$

Т.е. его эффект достигается за счет того, что сначала изображение усредняется, а затем из него вычитается исходное. В результате линии границ на изображении становятся более резкими. Если вычесть лапласиан из изображения, то получим

$$f(x,y) - \nabla^2 f(x,y) = 5f(x,y) -[f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] =$$

$$f(x,y) + 4\{f(x,y) -0{,}25[f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)]\} . \quad (5.15)$$

Следовательно, на исходном изображении границы усиливаются, т.к. к ним прибавляются высокочастотные компоненты. Поэтому вычитание лаплассиана из исходного изображения приводит к улучшению фокусировки последнего. Обычно такой же эффект дает вычитание из исходного изображения взвешенного среднего.

5.5 Сжатие полутоновых черно-белых и цветных изображений

5.5.1 Кодирование изображений с частичной потерей информации

Сжимаемые изображения предназначаются для восприятия их человеком, либо для обработки автоматическими устройствами. Если изображение кодируется для передачи зрителю, то уменьшить объем передаваемой информации можно, используя особенности восприятия зрительного анализатора [12]. Поскольку точность восприятия зрительного анализатора человека ограничена, то это позволяет считать некоторые искажения изображения незаметными или незначительными. Эта особенность дает возможность сжимать исходное изображение за счет потери части малозначительной информации, т. е., вносить определенные искажения. При декодировании, естественно, исключенная информация не может быть восстановлена и изображение воспроизводится с некоторой погрешностью. Различные методы кодирования вносят искажения разной степени. Поэтому при разработке системы компрессии изображений необходимо выбрать такой метод преобразования, который вносит наименее заметные искажения. В настоящее время большинство систем сжатия черно-белых и цветных неподвижных и подвижных изображений явля-

ются системами с потерей части информации. В то же время имеются области применения обработки изображений с использованием автоматических анализаторов, где потери какой-либо части изображений не допускается.

При проектировании и оценке эффективности методов сжатия изображений необходимо иметь достоверную количественную меру качества изображения. К сожалению, не существует аналитической объективно адекватной меры качества изображения для различных систем сжатия изображений. Поэтому для характеристики качества применяются шкалы *субъективной оценки* качества изображения, испытательные изображения, численные зависимости качества от искажений [8,12]. Любая «хорошая» мера качества изображения должна быть коррелирована с субъективными оценками качества изображения, представленными в виде шкалы. Существуют две шкалы субъективной оценки: *шкала качества* и *шкала ухудшения изображения*. Обычно используется пятибалльная система оценок. Каждая ступень качества шкалы характеризует качество рассматриваемого изображения с учетом некоторого множества испытательных изображений. По шкале ухудшения можно оценить степень искажения кодированного изображения по отношению к некоторому исходному изображению. В табл.5.2 приведены шкалы качества и ухудшения, принятые в технике передачи изображений.

Таблица 5.2 - Шкала качества и ухудшения изображения

Качество	Оценка в баллах	Ухудшение
Отличное	5	Незаметно
Хорошее	4	Заметно, но не мешает
Удовлетворительное	3	Слегка мешает
Плохое	2	Мешает
Очень плохое	1	Очень мешает

Процедура оценки качества изображения осуществляется методом экспертной оценки. До эксперимента экспертам предъявляется неискаженное испытательное изображение. В течение эксперимента периодически показывают неискаженное изображение, сменяющееся с оцениваемым или рядом с ним.

Числовые оценки качества изображений делятся на два класса: одномерные и двухмерные. *Одномерные* используются только для одного

изображения и измерительная шкала калибруется с помощью сравнения оцениваемого изображения с исходным. *Двухмерные* являются дифференциальными показателями качества изображения до обработки и после. Одномерные методы, как правило, основываются на пространственном спектре изображения. Типичные меры качества изображения включают общую или среднюю энергию сигнала на всех частотах. Другой класс одномерных числовых мер основан на использовании статистических характеристик изображения, в частности, математического ожидания и дисперсии отсчетов дискретного изображения или гистограммы полутонового изображения. Одномерная гистограмма показывает распределение в изображении пикселей с определенным уровнем яркости. Такую гистограмму можно рассматривать как аппроксимацию одномерной плотности распределения вероятностей изображения. По ее форме можно судить о контрастности изображения. Аналогично двумерная гистограмма - это аппроксимация двумерной плотности распределения вероятностей изображения. Ширина такой гистограммы в диагональном направлении указывает на пространственную корреляцию изображения.

Двухмерные меры качества изображения применяют наиболее часто при оценке качества сжатого изображения, поскольку они указывают на относительные искажения закодированного изображения по сравнению с исходным. Самой распространенной мерой является среднеквадратическая ошибка, представляющая собой разность между значениями соответствующих пикселей исходного и искаженного изображения. К сожалению, среднеквадратическая ошибка часто слабо коррелирована с субъективными оценками качества изображения [9].

5.5.2 Кодирование изображений методом ИКМ и ДИКМ

При *импульсно-кодовой модуляции* (ИКМ) сигнал изображения дискретизируется, каждый отсчет квантуется и кодируется двоичным кодом. В отличие от факсимильной передачи, где сигналы квантуются на два уровня: белый и черный и кодируются с точностью 1 бит на отсчет (нулем или единицей), количество уровней квантования полутоновых изображений устанавливается от 64 до 256, для кодирования которых затрачивается 6 ... 8 бит. Цветное изображение обычно представляет собой комбинацию трех цветов: красного *(R),* зеленого *(G)* и синего *(B)*. Каждый из этих цветов обрабатывается независимо от других и для кодирования каждого из них выделяется от 6 до 8 битов на отсчет. Кодирование методом ИКМ является своего рода эталоном кодирования. Поэтому степень снижения скорости передачи за счет сжатия исходного сообщения одним из предлагаемых методов обычно оценивается по отношению к ИКМ. Снижение числа уровней квантования при кодировании методом ИКМ не

целесообразно, так как это вызывает появление ложных контуров, которые возникают за счет скачкообразных изменений уровней на участках изображения с плавными изменениями яркости или цвета. В то же время энтропия полутонового изображения составляет 4 ... 6 бит на пиксел, что свидетельствует об избыточности кодирования сигналов изображения методом ИКМ.

Для уменьшения избыточности сигналов изображения применяется *дифференциальная импульсно - кодовая модуляция* (ДИКМ). При таком способе модуляции кодируется не полное значение отсчетов, представляющих уровень яркости элементов изображения (пикселей), а разность между текущим и предыдущим отсчетами сигнала. Еще в большей степени уменьшается избыточность в системах с ДИКМ, в которых кодированию подлежит разность между предыдущим отсчетом и предсказанным значением яркости последующего пикселя. Такой вид обработки видеосигналов является разновидностью кодирования с предсказанием, который наиболее широко используется в современных системах передачи изображений . При этой модуляции используется то обстоятельство, что непрерывно изменяющаяся яркость изображения *f(x,y)* может быть предсказана для различных *x*. Это используется для сжатия сообщения за счет передачи только той информации, которая требуется для коррекции предсказания. Поскольку предсказанное значение каждого отсчета вычисляется на основе только предшествующих значений, то оно имеется как в кодере, так и в декодере. Восстановление отсчетов в декодере производится путем суммирования ошибки предсказания с предсказанным значением этого отсчета. Структурная схема цифровой ДИКМ изображена на рис.5.15.

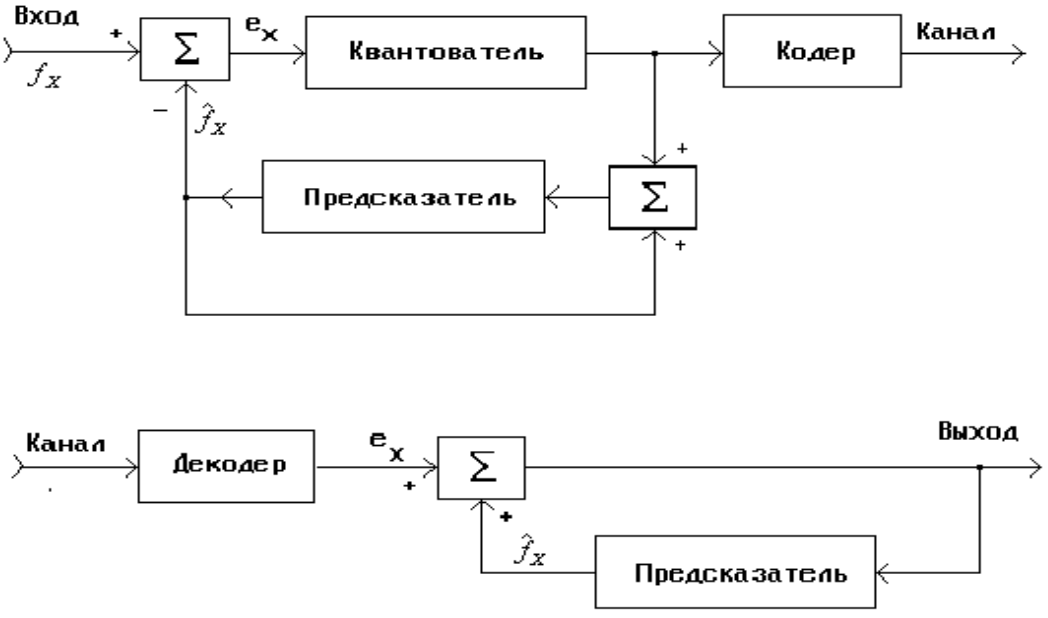


Рис.5.15. Структурная схема цифровой ДИКМ

Входным сигналом цифровой ДИКМ является ИКМ сигнал неподвижного изображения или последовательности телевизионных кадров. Отсчеты исходного сигнала изображения кодируются методом 8-разрядной равномерной ИКМ. Для каждого входного отсчета яркости f_X точки X предсказатель по N-1 предыдущим отсчетам определяет предсказанное значение $\hat{f}_X$.

Наибольшее распространение получили методы линейного предсказания. В этом случае предсказанное значение яркости X - го пиксела определяется по формуле

$$\hat{f}_X = \sum_{i=1}^{N-1} a_i f_{X-i} \ , \qquad (5.16)$$

где a_i - соответствующие весовые коэффициенты.

Разность между предсказанным и истинным значениями M-го отсчета

$$e_M = \hat{f}_M - f_M \qquad (5.17)$$

называют *ошибкой предсказания* , а соответствующий ей сигнал ошибки - разностным сигналом. Весовые коэффициенты a_i выбираются таким образом, чтобы минимизировать дисперсию ошибки предсказания. Ошибка предсказания квантуется неравномерным оптимальным квантователем, который обеспечивает либо минимальную среднеквадратическую ошибку квантования, либо согласуется со свойствами восприятия шумов квантования зрительным анализатором человека. Затем квантованные значения кодируются одним из способов, позволяющим дополнительно сжать передаваемое сообщение.

Кодирование с предсказанием базируется на использовании как статистических свойств сигнала изображения, так и свойств зрительного анализатора. В общем случае отсчеты значений яркости соседних в пространстве пикселов *коррелированы*. В телевизионном сигнале, кроме того, существует временная корреляция между элементами изображений в последовательных кадрах. Корреляция, или линейная статистическая зависимость, указывает на то, что при линейном предсказании отсчета на основе соседних пикселов ошибка предсказания будет иметь меньшую дисперсию, чем исходный отсчет. Сигнал ошибки имеет больший динамический диапазон, чем исходный: он может иметь не только положительные но и отрицательные значения. Однако распределение вероятностей для сигнала ошибки выгодно отличается от распределения вероятностей исходного сигнала. Условные распределения вероятностей, характеризующие связи между близкими элементами изображения, а вместе с тем и распределения вероятностей сигнала ошибки, весьма нерав-

номерны. Возникает резкий пик вокруг нуля в распределении для разностного сигнала, так как ошибка предсказания, благодаря сильным связям между соседними элементами, как правило, мала. Хотя и в разностном сигнале остаются корреляционные связи (например, контурные линии объектов), в целом эти связи значительно ослабевают, что и дает возможность утверждать о декорреляции изображений при замене исходного сигнала разностным, полученным при линейном предсказании.

В качестве критерия декорреляции обычно используют *минимум среднего квадрата ошибки предсказания*. Весовые коэффициенты a_i, минимизирующие средний квадрат ошибки предсказания, могут быть вычислены, если известна корреляционная функция изображения. Чтобы осуществить оптимизацию линейного предсказания для ансамбля изображений, нужно знать усредненную по ансамблю функцию корреляции. При этом телевизионные сообщения упрощенно полагают стационарным случайным процессом.

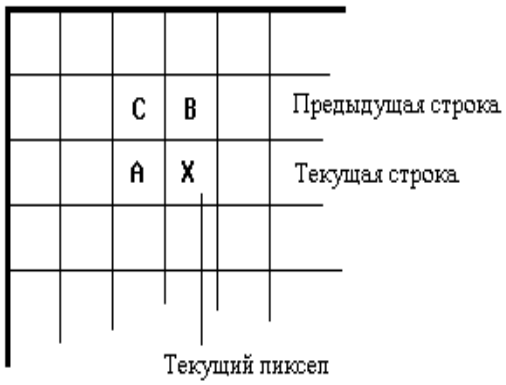


Рис.5.16. Схема размещения пикселов по *JPEG*

В зависимости от того, как выбираются элементы изображения, по которым ведется предсказание, все системы с ДИКМ делят на три группы: *одномерные, двухмерные* и *трехмерные*. В алгоритмах одномерного предсказания используется корреляция соседних элементов изображения вдоль строки развертки *(строчная корреляция)*. Такие алгоритмы часто применяют на практике, так как они просты в реализации и достаточно хорошо согласуются со статистическими характеристиками сигнала изображения.

Алгоритмы двухмерного предсказания предполагают учет значений яркости пикселов, расположенных в текущей и предыдущей строках. В этом случае уменьшается ошибка предсказания не только по координате *x* но и по *y,* что приводит к субъективному улучшению качества изображения. Алгоритмы одномерного и двухмерного кодирования относят к *внутрикадровому* кодированию, так как при этом учитываются элементы изображения только текущего кадра.

В алгоритмах третьей группы предсказание осуществляется сразу по трем осям, то есть, кроме пикселов, используемых в двухмерном предсказании, учитываются значения отсчетов этих пикселов в предыдущем кадре. Такое предсказание получило название *междукадрового* кодирования. Как правило, междукадровая избыточность достигает большой величины и кодирование с междукадровым предсказанием может быть весьма эффективным.

Однако трудность его применения связана с необходимостью хранения отсчетов предыдущих кадров, что требует огромных затрат памяти. На практике используют различные формулы расчета предсказанного значения яркости *X*-го пиксела $\hat{f}_X$. При сжатии неподвижных изображений в соответствии с рекомендациями международного стандарта *JPEG* [37] для прогнозирования отсчета текущего пиксела может быть использовано значение отсчета одного из соседних элементов изображения, расположенного в текущей или предыдущей строке, либо линейная комбинация отсчетов этих элементов. При учете одного пиксела яркость текущего элемента может быть определена по одной из следующих формул:

$$\hat{f}_X = f_A; \quad \hat{f}_X = f_B \quad \text{или} \quad \hat{f}_X = f_C \ .$$

Здесь буквенными индексами *A, B* и *C* обозначены пикселы, схема расположения которых показана на рис.5.16. Для более точного предсказания значения $\hat{f}_X$ стандартом рекомендуется использовать одну из формул :

$$\hat{f}_X = f_A + f_B - f_C \ ; \qquad (5.18)$$

$$\hat{f}_X = f_A + (f_B - f_C)/2 \ ; \qquad (5.19)$$

$$\hat{f}_X = f_B + (f_A - f_C)/2 \ ; \qquad (5.20)$$

$$f_X = (f_A + f_B)/2 \ . \qquad (5.21)$$

В подвижных (телевизионных) изображениях применяется *чересстрочная* развертка, при которой кадр изображения состоит из двух полей, строки которых перемежаются, то есть строки второго поля кадра (четные) располагаются между строками первого (нечетными). В таких системах рекомендуется определять предсказанное значение *X*-го пиксела по формулам [8]:

$$\hat{f}_X = f_A \ ; \qquad (5.22)$$

$$\hat{f}_X = (f_A + f_D)/2 \ ; \qquad (5.23)$$

$$\hat{f}_X = 3(f_A + f_C)/4 - f_B/2 \ ; \qquad (5.24)$$

$$\hat{f}_X = (f_E + f_F)/2 \ ; \qquad (5.25)$$

$$\hat{f}_X = f_A + (f_E + f_F)/2 - (f_C + f_H)/2 \ ; \qquad (5.26)$$

$$f_X = f_{X'} \ . \qquad (5.27)$$

Схема размещения пикселов для текущего и предыдущего кадров показана на рис.5.17. Штриховой линией обозначены перемежающиеся строки чересстрочной развертки.

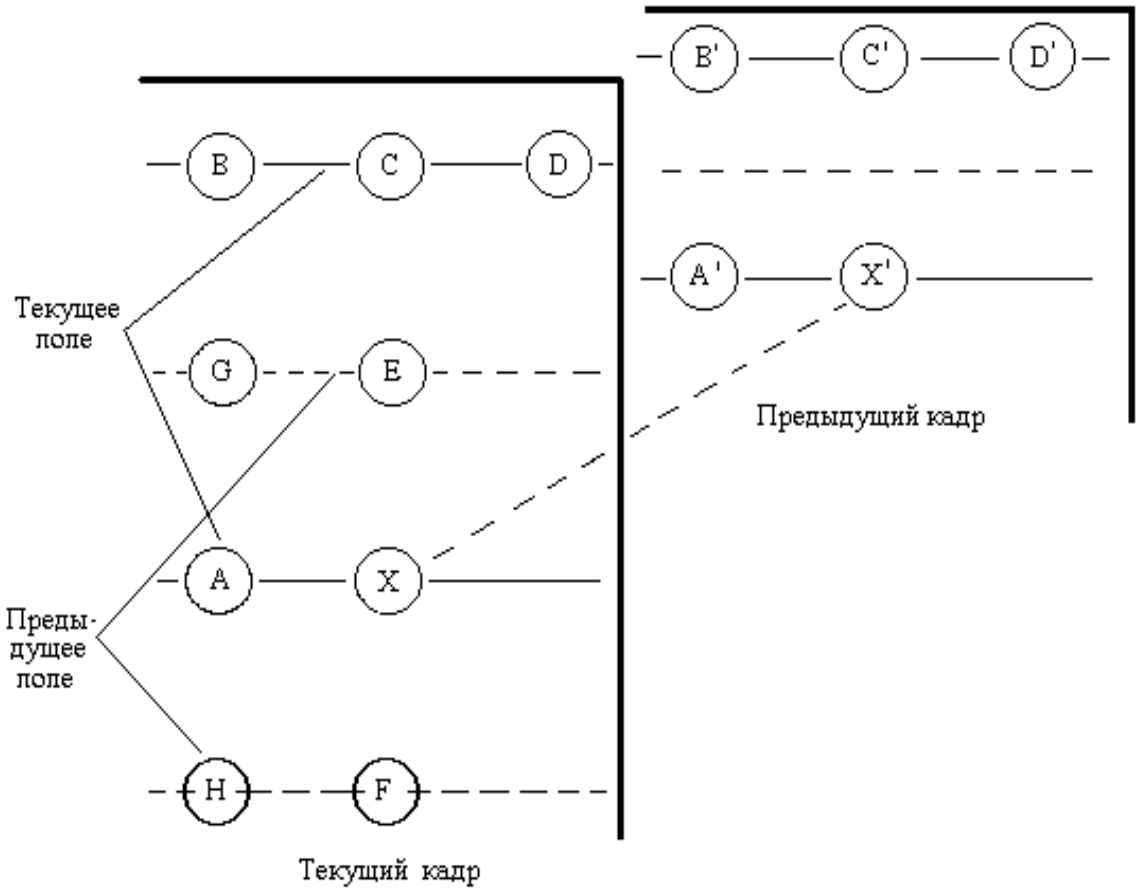


Рис.5.17. Схема размещения пикселов телевизионных изображений

Выбор формулы (5.18 - 5.27) для расчета $\hat{f}_X$ определяется характером изображения и его статистическими характеристиками, а также требуемым качеством восстанавливаемого изображения. Зачастую выбор той или иной формулы осуществляется экспериментальным путем. Во многих случаях целесообразно использовать различные формулы предсказания в зависимости от характеристик сжимаемых изображений.

5.5.3 Кодирование изображений с преобразованием

Кодированием с преобразованием называется процедура, при которой получаемый с помощью ИКМ видеосигнал до передачи подвергается некоторому обратимому преобразованию с последующим квантованием и кодированием. Целью преобразования является *превращение*

статистически зависимых элементов изображения *в независимые коэффициенты*. Примечательным свойством преобразования является то, что все «важные» коэффициенты сосредоточиваются в определенной зоне. Менее значимые коэффициенты преобразованного изображения отбрасываются. Тем самым уменьшается объем исходного изображения. Коэффициент сжатия зависит от числа сохраняемых коэффициентов и может достигать 10. Входной сигнал $S_и$, который представляет собой оцифрованные отсчеты элементов изображения, вначале разбивается на блоки (фрагменты) размером $M \times N \times K$. Здесь M и N- количество элементов изображения в строке и столбце соответственно, а K - количество кадров изображений. Для неподвижных изображений K=1. Структурная схема кодирования с преобразованием показана на рис.5.18.

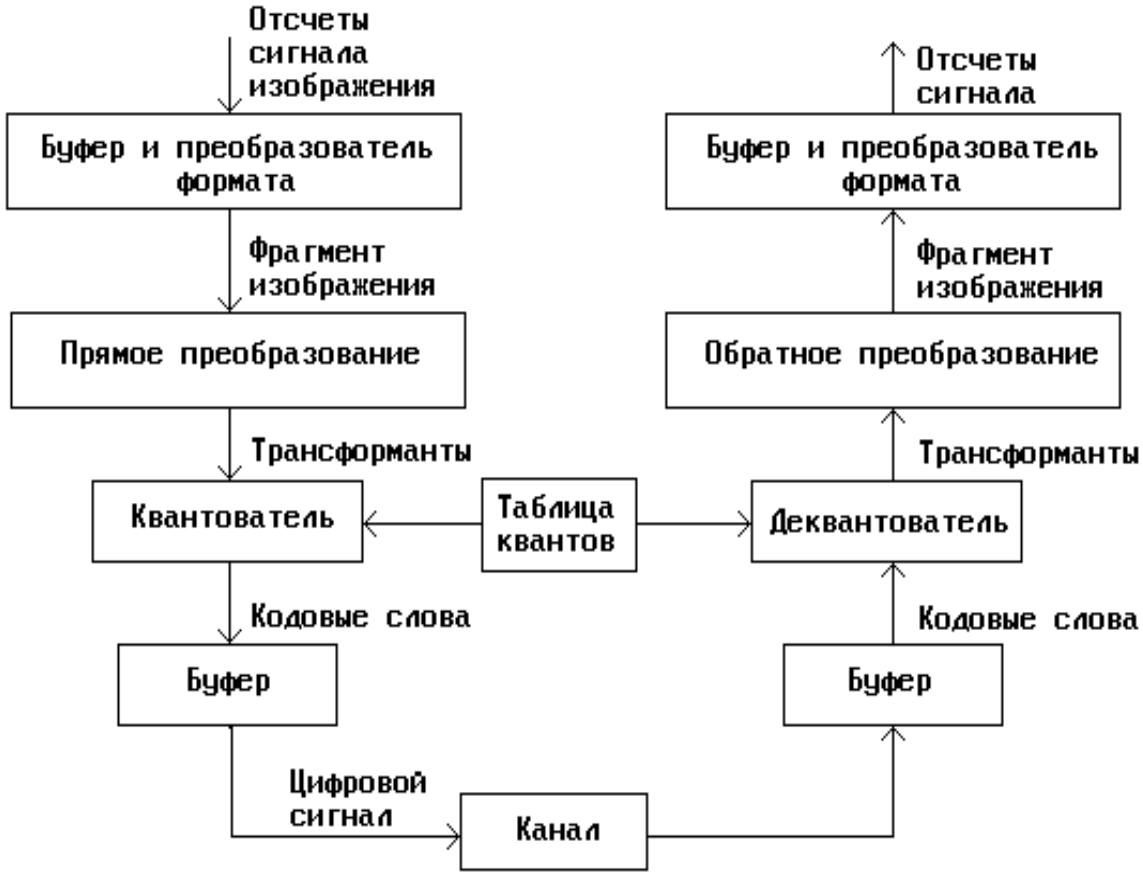


Рис.5.18. Структурная схема кодирования с преобразованием

Двумерному преобразованию подвергается все изображение или его области, называемые фрагментами. Типичный фрагмент изображения состоит из 8 × 8, 16 × 16 или 32 × 32 отсчетов, хотя для некоторых изображений он может быть увеличен до 256 × 256. Выбор такого размера фрагмента обусловлен тем, что интервал корреляции в изображениях не превышает 8 ... 32 отсчетов. В то же время, увеличение размера фраг-

мента резко увеличивает необходимый объем памяти, и затраты времени вычислителя на его обработку.

При сжатии изображений широко применяется преобразование Фурье, косинусное и синусное преобразования, преобразования Адамара, Хаара и Карунена-Лоэва [12]. В результате преобразования создается некоррелированный ряд чисел, называемый *трансформантами*. При этом число относительно больших трансформант невелико, в связи с чем трансформанты могут быть закодированы гораздо эффективнее, чем информация о самом изображении.

Полученные трансформанты разделяются для последующей обработки. Разделение базируется на зональном или пороговом принципе. При зональном разделении отбираются только те трансформанты, которые находятся в заранее определенной зоне (обычно в области нижних пространственных частот) и оказывают существенное влияние на субъективное качество изображения.

В случае порогового разделения отбираются трансформанты, превышающие некоторый пороговый уровень. Затем производится квантование и кодирование отобранных трансформант, а остальные приравниваются к нулю. За счет *отбрасывания* части трансформант осуществляется сжатие исходного изображения. Очевидно, что при этом происходит искажение передаваемого изображения. Поэтому очень важным является исключение только тех трансформант, которые мало сказываются на качестве воспроизводимого изображения. Квантование является отображением области непрерывных коэффициентов в область их целочисленных значений, которые далее преобразуются в кодовые слова. Следует заметить, что квантование является вторым источником искажений при сжатии изображений, так как это процесс необратимого преобразования аналогового источника в его квантованный эквивалент. В некоторых схемах сжатия изображений отбрасывание несущественных трансформант производится после процесса квантования.

Квантование трансформант осуществляется в два этапа:

1) трансформанты нормируются измеренной дисперсией, которая определяется путем оценки большого числа фрагментов;

2) нормированные трансформанты обрабатываются квантователем, оптимальным для данной модели сигнала; число битов, необходимых для представления квантованных трансформант, определяется исходя из предполагаемой дисперсии трансформант до квантования и допустимого уровня искажений; выходная целочисленная последовательность квантователя поступает в канал и затем на вход приемника, где происходит обратное преобразование последовательности проквантованных трансформант и восстановление исходного изображения.

Наилучшим способом преобразования сигналов изображения, обеспечивающим минимальную среднеквадратическую ошибку, является

преобразование Карунена-Лоэва (К-Л). На практике это преобразование не нашло применения в связи с тем, что для его реализации требуется знание статистических характеристик ансамбля обрабатываемых изображений. Кроме того, для этого преобразования нет алгоритма быстрого вычисления. Наиболее близко по эффективности к преобразованию К-Л приближается дискретное косинусное преобразование, имеющее быстрый алгоритм вычислений.

Дискретное косинусное преобразование (ДКП) является родственным дискретному преобразованию Фурье. Однако ДКП работает не с двумерным сигналом (яркость *f*, время *t*), а с трехмерным (координаты изображения *x, y* и яркость *f*). В соответствии с алгоритмом ДКП последовательность отсчетов яркости пикселей преобразуется из трехмерного пространства в идентичное представление в частотной области. Другими словами, посредством косинусного преобразования осуществляется преобразование *пространственной* информации в *частотную* (спектральную). Оси *x* и *y* представляют собой пространственные частоты сигнала преобразования в двух различных измерениях. При этом пространственные частоты изображения по координатам *x* и *y* определяются числом черных штрихов (разделенных белыми промежутками такой же ширины, как черный штрих) в изображении, укладывающиеся в отрезке длиной 1 мм, перпендикулярной к этим штрихам прямой, совпадающей с направлениями *x* и *y* соответственно. Отсюда следует, что пространственные частоты имеют размерность мм$^{-1}$.

Дискретное косинусное преобразование является обратимым, то есть, посредством обратного косинусного преобразования осуществляется перенос сигнала из *частотной* области в *пространственное* представление. Косинусное преобразование оперирует с квадратной матрицей отсчетов яркости элементов изображения *f(x,y)* размером $N \times N$ пикселов. Результатом преобразования является квадратная матрица $N \times N$ частотных коэффициентов *(трансформант) F(i, j)*. Формулы для прямого и обратного ДКП представлены соответственно выражениями:

$$F(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)i\pi}{2N} \cos\frac{(2y+1)j\pi}{2N}, \quad (5.28)$$

$$f(x,y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)F(i,j) \cos\frac{(2x+1)i\pi}{2N} \cos\frac{(2y+1)j\pi}{2N}. \quad (5.29)$$

Здесь *C(i)* и *C(j)* равны $1/\sqrt{2}$ для $i, j = 0$ и *C(i), C(j) = 1* при $i, j > 0$; *f(x,y)* – зна-чение яркости пиксела фрагмента изображения с координатами *x* и *y*.

На первый взгляд формулы представляются довольно громоздкими, однако вычисления по ним программируются с помощью несложных

процедур. Отдельные выражения этих формул могут быть заменены простыми табличными операциями. Например, произведение двух косинусов целесообразно вычислить заранее и сохранить результаты в памяти. Аналогично составляется таблица значений коэффициентов *C(i)* и *C(j)*.

5.5.4 Стандартная процедура кодирования изображений *JPEG*

Для обобщения опыта разработки и использования методов сжатия неподвижных полутоновых изображений и разработки международного стандарта МККТТ и МОС в 1991 году была создана организация, состоящая из группы экспертов, которая получила название *JPEG (Joint Photographic Expert Group)*. Разработанный комиссией стандарт алгоритма обработки изображений получил название *JPEG*, который определяет правила сжатия многоградационных как черно-белых, так и цветных изображений. Стандарт состоит из ряда частей, включающих как сжатие данных без потерь, так и с частичным искажением преобразуемой информации. Компрессия без потерь базируется на основе ДИКМ с предсказанием, адаптивных алгоритмов Хаффмена или арифметического кодирования. Компрессия изображений с потерями использует метод косинусного преобразования с последующим квантованием.

Вносимые искажения информации при ее компрессии в соответствии с алгоритмом *JPEG* не должны приводить к заметному ухудшению качества восстанавливаемого изображения, в частности, качество изображения по сравнению с оригиналом должно оцениваться на «отлично» или «хорошо». Кроме этого метод должен быть применим для любых многоградационных изображений, а также быть достаточно простым в реализации [34].

Структура компрессора и декомпрессора видеоинформации по стандарту *JPEG* показана на рис.5.19. Обратите внимание на то, что она мало отличается от схемы сжатия с преобразованием, показанной на рис.5.18.

Рассмотрим подробнее некоторые особенности процедуры обработки изображений *JPEG*. Кодируемое изображение разбивается на *блоки* размером 8 × 8 элементов (пикселов). Каждый блок представляет собой 64-точечный дискретный сигнал, состоящий из последовательности целых чисел в диапазоне $[0, 2^k - 1]$, которые затем преобразовываются в знаковые числа диапазона $[-2^{k-1}, 2^{k-1} - 1]$. Так, при 256 градациях яркости количество разрядов для кодирования отсчета изображения $k = 8$. Яркость пиксела путем масштабирования переносится из интервала 0 - 255 в диапазон от - 127 до 127. В блоке прямого дискретного косинусного преобразования *FDCT* осуществляется вычисление в соответствии с выражением (5.28) 64-х коэффициентов преобразования (трансформант),

состоящих из последовательности целых знаковых чисел в диапазоне [-2^{10}, 2^{10} - 1].

Выходной сигнал блока *FDCT* представляет собой 64-элементный массив, организованный в матрицу 8 × 8. Амплитуды трансформант однозначно определяются исходным блоком отсчетов видеосигнала и представляет собой коэффициенты при дискретных частотах. Коэффициент при нулевой частоте определяет амплитуду постоянной составляющей *(DC)*, а остальные коэффициенты - амплитуды переменных составляющих *(AC)*. В связи с тем, что элементы изображения во входном блоке изменяются слабо, то за счет косинусного преобразования удается сгруппировать трансформанты в области нижних пространственных частот. Следует еще раз подчеркнуть, что косинусное преобразование является обратимым и не приводит к сжатию сообщения. Оно осуществляет только подготовку данных к процедуре сжатия, которая осуществляется в квантователе.

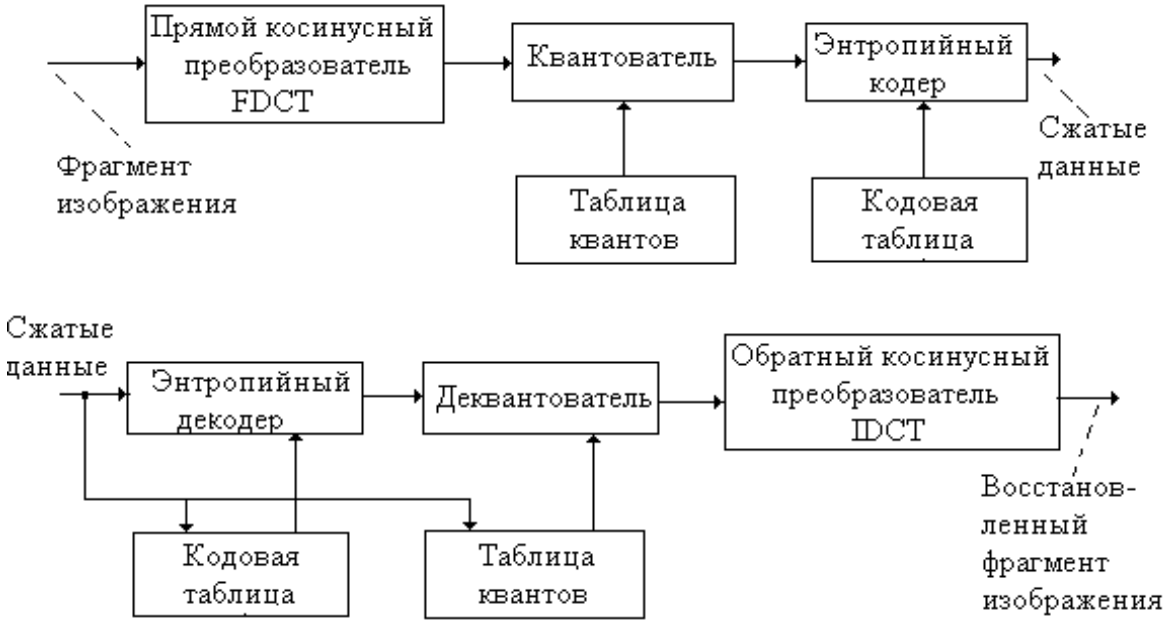


Рис. 5.19. Компрессор и декомпрессор JPEG

Целью квантования является компрессия изображения путем задания точности квантования не большей, чем это необходимо для получения желаемого качества воспроизведения изображения. При сжатии трансформант можно снижать их точность квантования, причем тем больше, чем дальше расположена трансформанта от постоянной составляющей *DC*, находящейся в матрице с индексами (0,0). Снижение точности отображения трансформант уменьшает количество требуемых для их представления битов. Элементы, которые расположены ближе к постоянной составляющей, кодируются большим числом битов, а более удаленные - меньшим.

В алгоритме *JPEG* операция квантования реализуется с помощью матрицы квантования. Для каждого элемента матрицы трансформант имеется соответствующее ему значение кванта *Q(i,j),* расположенное в матрице квантования. Квантование осуществляется делением каждой трансформанты *F(i,j)* на соответствующий ей квант *Q(i,j)* и выделением целой части числа

$$F_Q(i,j) = \lceil F(i,j) / Q(i,j) \rceil \ . \qquad (5.30)$$

Значение *Q(i, j)* находится в диапазоне от 1 до 255. Величина *Q(i,j)*=1 обеспечивает наибольшую точность. По мере удаления от верхнего левого угла матрицы значения квантов увеличиваются. Нетрудно заметить, что, начиная с некоторых значений, когда *Q(i,j)>F(i,j)* кантованное значение $F_Q(i,j)$ обращается в нуль, т. е. происходит невозвратимая потеря части информации. Выбор величины изменения квантов *Q(i,j)* от пиксела к пикселу определяет степень сжатия и качество воспроизведения информации. Несмотря на наличие стандартной таблицы квантов, *JPEG* предоставляет пользователям определенную свободу выбора элементов матрицы квантов в зависимости от желаемого качества воспроизведения. В [37] предложено определять значения квантов по формуле

$$Q[i, j] = 1 + (1 + i + j) \times \gamma \ , \qquad (5.40)$$

где i и j - индексы элементов матрицы квантов, при $i, j = 1, 2, \ldots, N$; γ-коэффициент качества воспроизведения изображения, задаваемый пользователем. Величину этого коэффициента рекомендуется выбирать в диапазоне от 1 до 25. Большие значения коэффициента качества также возможны, однако при этом качество воспроизводимого изображения резко ухудшается. В табл.5.3 представлена матрица квантов, рассчитанная по (5.40) при коэффициенте качества $\gamma = 2$.

Таблица 5.3 - Коэффициенты квантования *Q(i,j)*

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

При деквантовании производится операция умножения, т. е.

$$F'(i,j) = F'_{Q}(i,j) \times Q(i,j) . \qquad (5.41)$$

Величина $F'(i, j)$ является входной для обратного косинусного преобразования. В табл.5.4, в качестве примера, приведены значения трансформант на выходе косинусного преобразователя произвольного фрагмента изображения, а в табл.5.5 - значения на выходе квантизатора. В связи с тем, что многие трансформанты приобрели нулевое значение, объем передаваемой информации существенно уменьшится.

Таблица 5.4 - Значение трансформант перед квантованием

92	3	-9	-7	3	-1	0	2
-39	-58	12	17	-2	2	4	2
-84	62	1	-18	3	4	-5	5
-52	-36	-10	14	-10	4	-2	0
-86	-40	49	-7	17	-6	-2	5
-62	65	-12	-2	3	-8	-2	0
-17	14	-36	17	-11	3	3	-1
-54	32	-9	9	22	0	1	3

Таблица 5.5 - Значение трансформант после квантования

30	0	-1	0	0	0	0	0
-7	-8	1	1	0	0	0	0
-12	6	0	-1	0	0	0	0
-5	-3	0	0	0	0	0	0
-7	-3	3	0	0	0	0	0
-4	4	0	0	0	0	0	0
-1	0	-1	0	0	0	0	0
-3	1	0	0	0	0	0	0

Дальнейшим шагом *JPEG*-процедуры является кодирование квантованного изображения. Сначала разделяются трансформанты постоянной (*DC*) и переменной (*AC*) составляющих. Трансформанта постоянной составляющей является мерой среднего значения 63-х отсчетов изображения. Так как соседние блоки изображения обычно имеют сильную

корреляционную связь, то постоянная составляющая последующего блока в большинстве случаев мало отличается от *DC* - составляющей предыдущего блока. Она преобразуется из абсолютного значения в относительное, и затем кодируется приращение текущей *DC*-составляющей по отношению к предыдущей (ДИКМ).

Трансформанты переменных составляющих преобразуются в последовательность способом «Зигзаг» (рис.5.20). Последовательность трансформант AC_{01} ... AC_{77} можно сжать методом кодирования длин повторяющихся символов (так как в образованной последовательности имеется большое число нулей), либо хаффменовским или арифметическим кодированием. Для сжатия последовательности *AC*-трансформант фрагмента изображения рекомендуется использовать так называемое *энтропийное кодирование*. При этом способе амплитуды ненулевых *AC*-составляющих отображаются неравномерным кодом, не обладающим свойством разделимости кодовых комбинаций. Поэтому для их разделения перед каждой из комбинаций ставится индикатор, указывающий длину текущей кодовой комбинации. Длинные группы нулей , расположенные между ненулевыми трансформантами, сжимаются методом кодирования длин последовательности одинаковых символов. В соответствии с рекомендацией *JPEG* отрезки зигзаг-последовательности, состоящие из группы нулевых и одной ненулевой трансформант, кодируются двумя символами: *SYM1* и *SYM2*.

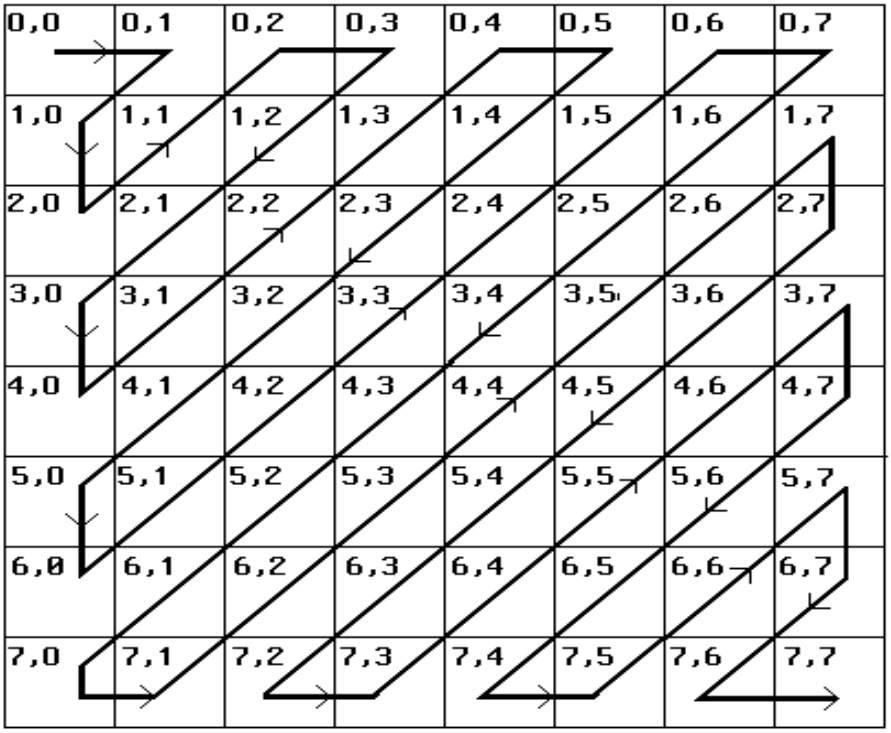


Рис.5.20. Схема считывания отсчетов фрагмента изображения

SYM1 представлен одним байтом, старший полубайт которого указывает длину ряда нулевых трансформант кодируемого отрезка, а младший - размер (количество битов) второго символа *SYM2*, отображающего амплитуду ненулевой трансформанты, завершающую отрезок последовательности нулевых трансформант. Очевидно, что полубайт может закодировать длину отрезка, состоящего из 1 ... 15 нулевых трансформант. На практике длина отрезка может быть больше 15. В этом случае длинная последовательность нулей представляется *SYM1* с кодом (15,0), отображающим группу, состоящую из 16 нулей. Таких символов при величине кодируемого фрагмента изображения 8 × 8 может быть до 3-х. Затем следует *SYM1* с кодом длины, дополняющей последовательность до действительного числа нулевых трансформант. Значение *SYM1* с кодом (0,0) используется для индикации конца кодирования текущего фрагмента отсчетов размером 8 × 8. Для кодирования амплитуды ненулевых трансформант используются целые двоичные знаковые числа, содержащие различное число битов (табл.5.6).

Таблица 5.6 – Кодирование амплитуды трансформант

Количество битов	Значение амплитуды
1	-1 , 1
2	от -3 до -2, от 2 до 3
3	от -7 до -4, от 4 до 7
4	от -15 до -8, от 8 до 15
5	от -31 до -16, от 16 до 31
6	от -63 до -32, от 32 до 63
7	от -127 до -64, от 64 до 127
8	от -255 до -128, от 128 до 255
9	от -511 до -256, от 256 до 511
10	от -1023 до -512, от 512 до 1023

Каждая группа битов кодирует симметричный диапазон амплитуд, состоящий из положительных и отрицательных значений. Старший бит чисел отображает знак, а остальные - значение амплитуды. Постоянная составляющая трансформант *DC* кодируется неравномерным кодом и представляется посредством двух символов. Первый символ *SYM1* указывает длину, а второй *SYM2* - амплитуду *DC*-составляющей. В связи с тем, что постоянные составляющие кодируются дифференциальным способом, диапазон

их представления увеличивается вдвое и изменяется от -2^{-11} до $2^{11} - 1$. Поэтому в табл.5.6 добавляется дополнительная строка, а *SYM1* принимает значение от 1 до 11. Такой неравномерный код по степени сжатия несколько уступает хаффменовскому или арифметическому кодам. Однако он значительно проще в реализации и является достаточно эффективным, когда большинство трансформант состоит из малых значений, что чаще всего наблюдается на практике.

В процессе исследований процедуры сжатия установлено, что кодовые комбинации *SYM1*, отображающие длину нулевых последовательностей и амплитуду трансформант, характеризуются большой неравномерностью вероятности появления. Поэтому *JPEG* рекомендует производить дополнительное сжатие информации путем хаффменовского кодирования символов *SYM1*.

Пример 5.2

Закодировать способом энтропийного кодирования проквантованые трансформанты блока, приведенного в табл.5.7.

Таблица 5.7 - Значение трансформант после квантования

86	-5	3	0	0	2	0	-1
12	-3	0	0	0	0	0	0
-1	17	0	0	0	-2	0	0
0	0	-1	0	0	0	0	0
0	21	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-7	0	0	0	0	0
0	0	0	0	0	0	0	0

На основе зигзагообразной развертки блока (рис.5.20) запишем последовательность значений трансформант:

86 -5 12 -1 -3 3 0 0 17 0 0 0 0 0 0 2 0 0 -1 21 0 0 0 0 0 0 0
0 -1 0 -2 0 0 0 0 0 0 -7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 .

Используя описанный выше способ кодирования и данные табл.5.6, получаем последовательность символов *SYM1* и *SYM2*:

(7)(86) (0,3)(-5) (0,4)(12) (0,1)(-1) (0,2)(-3) (0,2)(3) (2,5)(17)
(6,2)(2) (2,1)(-1) (0,5)(21) (8,1)(-1) (1,2)(-2) (6,3)(-7) (0,0) .

Рассмотренный *JPEG*-алгоритм относится к монохромным изображениям. Цветные изображения обычно состоят из трех компонент: красного, зеленого и синего цветов (*RGB* - формат сигнала). В этом случае *JPEG*-алгоритм рассматривает изображение так, как будто оно состоит из трех отдельных изображений. При *RGB*-изображении сначала сжимается красная, затем зеленая и в конце синяя компонента. Для каждой компоненты могут использоваться различные таблицы квантования и энтропийного кодирования, которые определяются статистическими характеристиками составляющих изображения. В процессе компрессии и декомпрессии осуществляется синхронное переключение таблиц в соответствии с обрабатываемой компонентой.

5.6 Обработка изображений в системе *MATLAB*

Прежде, чем приступить к разработке программ по обработке изображений для цифровых сигнальных процессоров, необходимо выбрать метод и алгоритм обработки, провести их всесторонние исследования и определить оптимальные параметры процедур. Затем эти процедуры должны быть реализованы в виде программных модулей, написанных на языке С или ассемблере конкретного сигнального процессора. С практической точки зрения представляется наиболее целесообразным проводить вышеуказанные исследования в области обработки изображений с использованием системы *MATLAB*, которая является удобным и достаточно простым инструментом исследования и, кроме того, в ее состав входит компилятор языка *MATLAB* на язык *C* и библиотека математических функций на *C++*. Таким образом, процедуры обработки изображений, написанные в системе *MATLAB*, без особых трудностей преобразуются в исполняемые модули для цифровых сигнальных процессоров семейств *TMS* и *ADSP*. Для исследования методов обработки изображений в системе *MATLAB* 5.2 разработан пакет *Image Processing Toolbox*, включающий широкий набор функций, который может быть дополнен путем написания пользователем собственных нетиповых функций.

Ниже рассматриваются преимущественно только функции, используемые для задач обработки, алгоритмы которых освещены в предыдущих параграфах. С полным списком функций и особенностями их использования можно ознакомиться в среде *MATLAB Command Window* с помощью команд *help images* и *help имя функции.*.

Базовым типом данных в *MATLAB* является прямоугольная матрица упорядоченного множества действительных или комплексных элементов изображения. Поэтому доступ к произвольному пикселу изображения осуществляется как к элементу матрицы.

MATLAB 5.2 поддерживает четыре типа изображений:

1) цветное индексированное;

2) полутоновое яркостное;
3) бинарное изображение;
4) цветное *RGB*.

Для индексированного цветного изображения требуется наличие двух матриц: таблицы цветности и матрицы индексов. Таблица цветности *(colormap)* представляет собой набор используемых в изображении цветов, а матрица индексов – набор номеров (индексов) от 0 до *n*-1 соответствующих цветов таблицы цветности. Размер таблицы цветности зависит от количества $N_ц$ используемых в изображении цветов и равен $N_ц \times 3$. Каждому индексу таблицы соответствуют значения трех *RGB* цветов. Для отображения каждого из этих цветов используются действительные числа с двойной точностью в диапазоне от 0,0 (уровень черного) до 1,0 (максимальная интенсивность).

В яркостном типе изображения используется только одна матрица, содержащая значения интенсивностей (уровней серого) каждого из пикселов. Значения интенсивности представляются реальными числами с двойной точностью в диапазоне от 0.0 до 1.0 , где уровень 0,0 соответствует черному а 1,0 – белому цвету.

В бинарном (черно-белом) изображении используется только два уровня: 0 - черный и 1 – белый цвет. В системе *MATLAB* эти уровни представляются числами с двойной точностью. Бинарные изображения могут быть также представлены логической (булевой) матрицей. В этом случае при обработке изображения применяются логические операции.

В цветном *RGB*-изображении используются три матрицы интенсивностей соответствующих цветов. Интенсивности представляются реальными числами с двойной точностью в диапазоне 0,0 – 1,0. Цвет и интенсивность пикселов определяется комбинацией интенсивностей соответствующих им цветовых компонент.

Если тип обрабатываемого изображения заранее не известен, то его можно определить с помощью функций *ISBW, ISGRAY, ISIND* или *ISRGB*. Эти функции проверяют на истинность соответственно бинарные черно-белые, полутоновые, индексированные или цветные *RGB* изображения. Общий синтаксис функции FLAG = ISGRAY (A) . Функция возвращает 1, если это изображение интенсивностей уровня серого, в противном случае возвращается 0.

При необходимости изображения некоторых типов могут быть конвертированы в другой тип, в частности, можно преобразовать индексированные изображения в полутоновые или цветные *RGB* (функции *IND2GRAY, IND2RGB*), либо произвести их обратное преобразование в индексированные (функции *GRAY2IND, RGB2IND*).

Например, чтобы прочитать из файла *floers.tif* *RGB*-изображение и преобразовать его в индексированный тип с матрицей интенсивностей X

и картой цветности (палитрой) map, содержащей 128 цветов следует включить в программу следующий фрагмент:

```
RGB=imread (‘flowers.tif’);
[X, map]= rgb2ind(RGB,128);
imshow (X, map) .
```

Инструментальные средства *MATLAB* позволяют обрабатывать изображения отдельными фрагментами (по терминологии *MATLAB* - блоками) с использованием оконных функций. Это позволяет снизить расход памяти компьютера и повысить скорость обработки. При этом нет необходимости программно задавать перемещения окна (блока) по изображению. Заданная функция автоматически выполняется для каждого фрагмента всего обрабатываемого изображения. Отдельные блоки поочередно покрывают матрицу изображения начиная с левого верхнего угла. Если блоки не укладываются в границах изображения, то недостающие пикселы автоматически заполняются нулями. Аналогичное заполнение нулями производится при обработке пикселов, расположенных на краях изображения.

Для блочной обработки в *Image Processing Toolbox* используется процедура BLKPROC, которая выделяет отдельный блок (фрагмент) изображения и передает его в функцию определенную пользователем. Затем BLKPROC собирает возвращенные функцией блоки для создания выходного изображения. Функция, используемая для обработки каждого блока, должна возвращать матрицу того же размера, что и блок. Функция BLKPROC описывается следующим образом:

```
B = BLKPROC (A, [m n], ‘fun’),
```

где A - матрица исходного, а B - выходного изображения; [m n] - размер фрагмента в пикселах; fun - функция обработки фрагмента. При использовании блоковой обработки изображения с помощью функции SIZ=BESTBLK([M N], K) можно предварительно найти оптимальный размер блока. Здесь M, N - размер исходного изображения, а K - предельные размеры искомого блока

Для чтения изображения из графического файла применяется функция IMREAD. Ее синтаксическое представление

```
A = IMREAD (FILENAME, FMT)  или
[X, MAP] = IMREAD (FILENAME, FMT).
```

В первом случае производится чтение изображения из файла. Если изображение полутоновое, то *A* представляет собой двумерный массив значений интенсивности элементов изображения. В случае трехцветного

(*RGB*) изображения *A* является трехмерным массивом размером *M×N×3*. *FMT* специфицирует формат представления изображения. Среди форматов представления графических изображений используются *.bmp, .jpeg, .pcx, .tif* и ряд других.

Функция [X, MAP] = IMREAD (FILENAME, FMT) читает индексированное изображение из файла в X , а карту цветов - в MAP.

Функция IMWRITE (A, FILENAME, FMT) - записывает изображение *A* в файл с именем FILENAME и форматом FMT , а

IMWRITE (X, MAP, FILENAME, FMT) - записывает в файл индексированное изображение *X* и связанную с ним карту цветов map.

Функция IMCROP (I, [XMIN YMIN WIDTH HEIGHT]) - вырезает фрагмент изображения заданного размера. Здесь XMIN, YMIN – задают начальные координаты прямоугольного фрагмента, а WIDTH, HEIGHT – количество колонок и строк фрагмента соответственно. Прямоугольник может быть также определен с помощью мыши.

Функция IMHIST(I,N) – отображает гистограмму распределения интенсивности пикселей изображения *I* с числом уровней интенсивности *N*. По умолчанию *N* принимается равным 256.

Функция EDGE - определяет границы интенсивности изображения. При этом в качестве параметров функции указывается метод определения границы, уровень порога и ряд других.

Функция HISTEQ улучшает контраст изображения за счет выравнивания гистограммы.

IMNOISE (I, TYPE, ...) добавляет к изображению интенсивностей I аддитивный гауссовский или мультипликативный шум типа TYPE . Для каждого типа шума дополнительно задаются его параметры.

Для фильтрации изображений используются функции FILTER2 и FSPECIAL(TYPE) . Во второй функции спецификация TYPE задает тип используемого линейного или нелинейного фильтра. В частности может быть задан один из следующих типов

‘gaussian’ - гауссовский низкочастотный фильтр ;
‘sobel’ - выделение границ на основе оператора Собела ;
‘previt’ - выделение границ на основе оператора Превита ;
‘laplacian’ - с использованием оператора Лапласа
‘average’ - усредняющий фильтр .

Кроме этого используются и другие функции фильтрации. Среди них MEDFILT2 - медианный фильтр; WIENER2 - адаптивный шумоподавляющий винеровский фильтр.

В составе *MATLAB* имеется широкий набор функций для различных преобразований изображений (Image transforms), в частности:

DCT2 и IDCT2 - вычисление двумерного прямого и обратного дискретно-косинусного преобразования соответственно;

FFT2 , FFTn, IFFT2 , IFFTn- прямое и обратное, двумерное и n-мерное быстрые преобразования Фурье;

RADON - преобразование Радона и ряд других.

Пример 5.3

В этом примере показано использование различных функций Images Processing для обработки изображений.

```
>> I=imread('forest.tif');  %Чтение изображения из
                            %файла
>> IC=imcrop(I,[50 100 40 60]);  %Вырез фрагмента с
                                 %указанием координат
                                 % его начала и размеров
>> imshow(I),figure(2),imshow(IC)  %отображение полного и
                                   % усеченного изображения
>> J=imnoise(I,'gaussian',0,0.05);  %воздействие на
                           %изображение гауссовской помехи
                           %с нулевым средним и нормированным
                           %среднеквадратическим отклонением
                           %равным 0.05

>> figure(3),imshow(J)       % изображение на экране
                            % зашумленного изображения
>> M=medfilt2(J);            % медианная фильтрация
>> W=wiener2(J,[3 3]);       % Винеровская фильтрация
>> figure(4), imshow(M),figure(5),imshow(W)
>> figure(6),imhist(I),figure(7),imhist(IC); %Получение
                                   % и отображение на экране
                                   % гистограмм полного и
                                   % усеченного изображений

>> BWs=edge(I,'sobel');     % выделение контуров с
>> BWp=edge(I,'prewitt');   % использованием операторов
                            % Собела и Превита

>> figure(1),imshow(PWs),figure(2),imshow(PWp)
                           %отображение гистограмм.
```

Обратите внимание на изменение качества изображения при вариации параметров помехи и затем при проведении различных видов фильтрации, а также на существенное отличие гистограмм полного изображения и его фрагмента.

Пример 5.4

Составить программу сжатия изображения с использованием дискретно-косинусного преобразования и функций блочной обработки.

```
>>  load trees          % загрузка входного изображения
```

```
>>  I=ind2gray(X,map); % преобразование
                       % индексированного
                       % изображения в полутоновое
>>  J=blkproc(I, [8 8], 'dct2'); % поблочное
                        % дискретно-косинусное
                       % преобразование с блоком 8 × 8
>> nz=find(abs(J)< 10); % нахождение индексов
                  %пикселов,интенсивность которых меньше
                  % 10
>> J(nz)= zeros(size(nz)); % замена этих пикселов
                           % нулевым значением
                           % (собственно сжатие)
>> K=blkproc(J,[8 8],'idct2'); % поблочное
                                %восстановление
                                % изображения
>> imshow(I), figure, imshow(K) % просмотр исходного
                       % и восстановленного изображений
```

Глава 6
ЦИФРОВЫЕ ПРОЦЕССОРЫ ОБРАБОТКИ СИГНАЛОВ (ЦПОС)

6.1 Общая характеристика ЦПОС

ЦПОС представляют собой программируемые микропроцессоры, предназначенные для реализации алгоритмов цифровой обработки сигналов. Спектр применений ЦПОС весьма широк. Например, ЦПОС применяют для реализации алгоритмов кодирования и декодирования речи, распознавания и синтеза речи, идентификации говорящего по голосу, профессиональной обработки звука, передачи данных, подавления шумов, сжатия и формирования изображений, формирования диаграмм направленности антенн, спектрального оценивания и др.

Основные серии ЦПОС, выпускаемые промышленностью, приведены в табл.6.1. ЦПОС различаются типом используемой арифметики, разрядностью и быстродействием. По типу арифметики все ЦПОС подразделяются на две группы: с фиксированной и с плавающей запятой. Рассмотрению особенностей ЦПОС с фиксированной и плавающей запятой посвящены соответственно главы 7 и 8. В настоящей главе рассматриваются общие архитектурные особенности ЦПОС.

Отличительными характеристиками ЦПОС, поддерживаемыми на аппаратном уровне, являются: высокоскоростное выполнение операции умножения с накоплением; многократный доступ к памяти; специальные режимы адресации; управление циклами; наличие на кристалле дополнительных устройств ввода-вывода цифровых и аналоговых сигналов.

Многие ЦПОС способны выполнять операцию *умножения с накоплением* за один командный цикл. Данную операцию обычно обозначают аббревиатурой MAC (Multiply Accumulate). Суть MAC операции выражается оператором:

$$R \leftarrow R + a \cdot x . \qquad (6.1)$$

В соответствии с оператором (6.1) сначала вычисляется произведение значений переменных a и x, а затем полученное произведение складывается со значением переменной R. Результат вновь запоминается в переменной R. К выполнению MAC-операции сводятся многие алгоритмы обработки сигналов: КИХ и БИХ фильтрация, вычисление корреляционных функций, БПФ, умножение векторов и матриц и др. Например, уравнение линейной фильтрации

$$y[n] = \sum_{i=1}^{N} b[i]x[n-i] - \sum_{i=1}^{M} a[i]y[n-i] \qquad (6.2)$$

предполагает вычисление суммы произведений. Основная операция БПФ -“бабочка” также соответствует сумме произведений:

$$U \leftarrow u + w \cdot v,$$
$$V \leftarrow v + (-w \cdot v), \qquad (6.3)$$

где w – поворачивающий множитель. MAC–операция реализуется в ЦПОС с помощью аппаратного умножителя и аккумулятора.

Таблица 6.1 - Основные серии ЦПОС

Серия	Тип арифметики	Длина слова, бит	Быстродействие, 10^6 команд/с	Производитель
ADSP- 21xx ADSP- 21xxx	Фиксиров. Плавающ.	16 32	33.3 40.0	Analog Devices
DSP 16xx DSP 32xxx	Фиксиров. Плавающ.	16 32	70.0 20.0	AT&T
DSP 5600x DSP 561xx DSP 563xx DSP 566xx DSP 568xx DSP 96002	Фиксиров. Фиксиров. Фиксиров. Фиксиров. Фиксиров. Плавающ.	24 16 24 16/24 24 32	40.0 30.0 80.0 60.0 20.0 20.0	Motorola
TMS 320 C1x TMS 320 C2x TMS 320 C2xx TMS 320 C3x TMS 320 C4x TMS 320 C5x TMS 320 C54x TMS 320 C62x TMS 320 C8x	Фиксиров. Фиксиров. Фиксиров. Плавающ. Плавающ. Фиксиров. Фиксиров. Фиксиров. Фиксиров.	16 16 16 32 32 16 16 16 8/16	8.8 12.5 40.0 25.0 30.0 50.0 66.0 1600.0 5x50.0	Texas Instruments
μPD 7701x	Фиксиров.	16	33.3	NEC

Многократный доступ к памяти позволяет ЦПОС обращаться к памяти несколько раз за время выполнения одной команды. Это дает возможность совмещать выборку очередной команды из памяти с выборкой необходимых операндов или с сохранением результатов выполнения предыдущей команды.

Для повышения эффективности реализации алгоритмов цифровой обработки сигналов ЦПОС используют специальные режимы адресации. Наиболее часто используется *косвенная регистровая адресация* с после-

дующим инкрементом адреса. Обычно такая адресация используется при циклической обработке данных , размещенных в смежных ячейках памяти. Кроме этого, некоторые ЦПОС аппаратно поддерживают режим *модульной (кольцевой) адресации*, что обеспечивает работу с кольцевой очередью данных (кольцевым буфером). Большинство процессоров поддерживают *бит-инверсную адресацию*, применяемую в алгоритмах БПФ.

Многие алгоритмы ЦПОС предполагают выполнение циклических действий. Для повышения эффективности *управления циклами* ЦПОС аппаратно поддерживают выполнение операторов циклов.

С целью обеспечения возможности ввода данных большинство ЦПОС имеют встроенные порты ввода-вывода, средства прямого доступа к памяти (ПДП). Ряд ЦПОС имеют встроенные аналоговые и цифроаналоговые преобразователи.

Указанные особенности ЦПОС рассматриваются ниже на примерах конкретных серий ЦПОС.

6.2 MAC – операция и другие операции обработки данных ЦПОС

Рассмотрим особенности реализации основных операций обработки данных в ЦПОС. Это позволит выяснить общие принципы построения арифметических устройств ЦПОС.

Пусть требуется реализовать уравнение нерекурсивной фильтрации

$$y[n] = h[0] \cdot x[n] + h[1] \cdot x[n-1] + h[2] \cdot x[n-2] + h[3] \cdot x[n-3]. \quad (6.4)$$

В этом случае необходимо четыре раза выполнить MAC-операцию (рис.6.1), которая в ЦПОС реализуется в соответствии с обобщенной схемой, изображенной на рис. 6.2 . Аргументы MAC-операции хранятся в X и Y регистрах. Загрузка регистров выполняется обычно с помощью команд пересылок данных. В X-регистрах хранятся значения сигнала $x[n]$, …, $x[n\text{-}3]$, а Y-регистрах хранятся значения коэффициентов фильтра $h[0]$,…, $h[3]$. В каждый момент времени вычисляется произведение значений X и Y регистров, на которые ссылаются регистры-указатели. Произведения накапливаются в аккумуляторе, входящем в состав устройства, реализующего MAC-операцию.

На рис. 6.3 - 6.6 детально показаны различные фазы вычислений выходного сигнала фильтра (6.4) в обобщенном ЦПОС. Здесь использованы следующие обозначения: MULT – умножитель; ADD-устройство сложения; ACC – аккумулятор (устройство хранения суммы значений). Фазе 0 соответствует запись текущего значения сигнала $x[n]$ в регистр X_0 и вывод результатов вычислений выходного сигнала $y[n\text{-}1]$, соответствующих

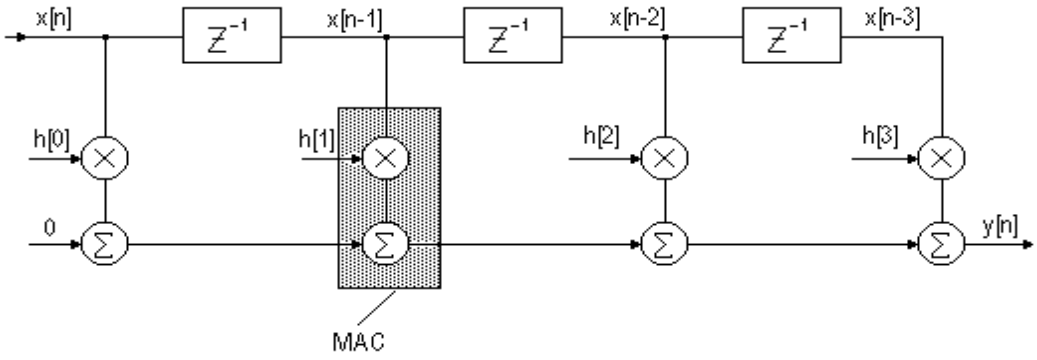


Рис.6.1. Структурная схема НФ фильтра

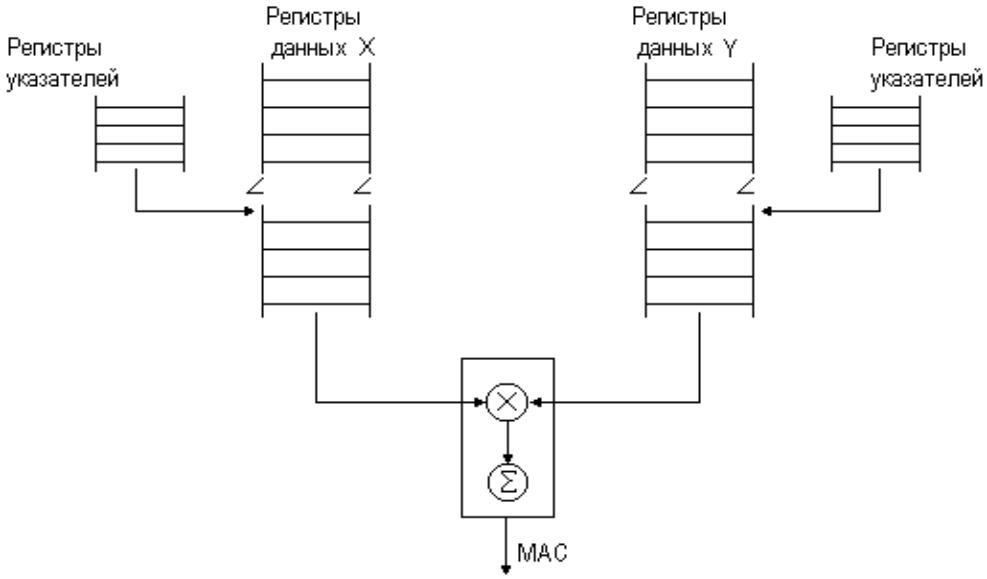


Рис.6.2. Реализация MAC-операции

предыдущему шагу (жирные линии на рис. 6.3). В фазе 1 обнуляется аккумулятор, значение $x[n]$ переписывается из регистра X_0 в X–регистр по указателю, указатель Y–регистров переводится из состояния, соответствующего указанию на $h[3]$, в состояние, соответствующее указанию на $h[0]$. В фазе 2 выполняется первая MAC-операция. В этом случае перемножаются значения по указателям Rx и Ry, т.е. $0+x[n]\cdot h[0]$. После выполнения операции указатели переводятся в новое состояние. Фаза 3 сопряжена с выполнением второй MAC-операции: $x[n]\cdot h[0]+x[n-1]\cdot h[1]$. В фазах 4 и 5 выполняется третья и четвертая MAC-операции. Значение, накопленное в аккумуляторе, будет равно $x[n]\cdot h[0]+x[n-1]\cdot h[1]+x[n-2]\ h[2]+x[n-3]\cdot h[3]$. Фаза 6 соответствует вводу нового значения входного сигнала $x[n+1]$ и выводу результата y[n]. Фаза 7 соответствует фазе 1, но сопряжена с подготовкой к вычислению следующего значения выходного сигнала $y[n+1]$.

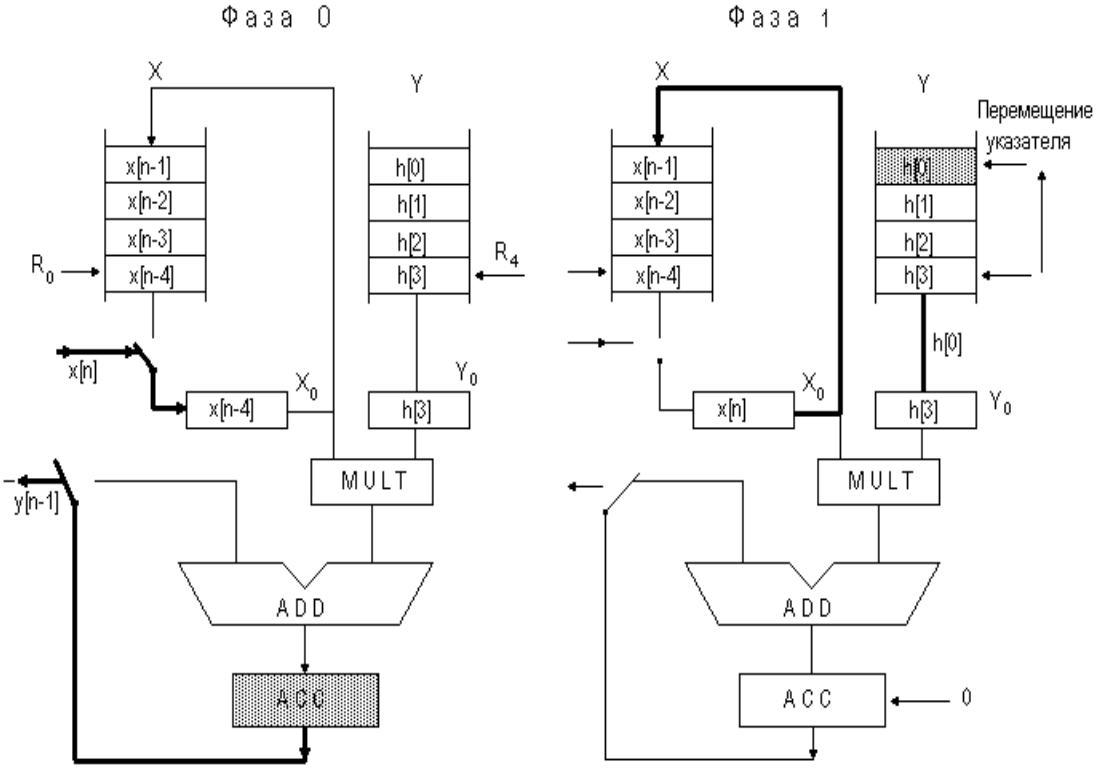


Рис.6.3. Схема выполнения MAC –операции (фазы 0 и 1)

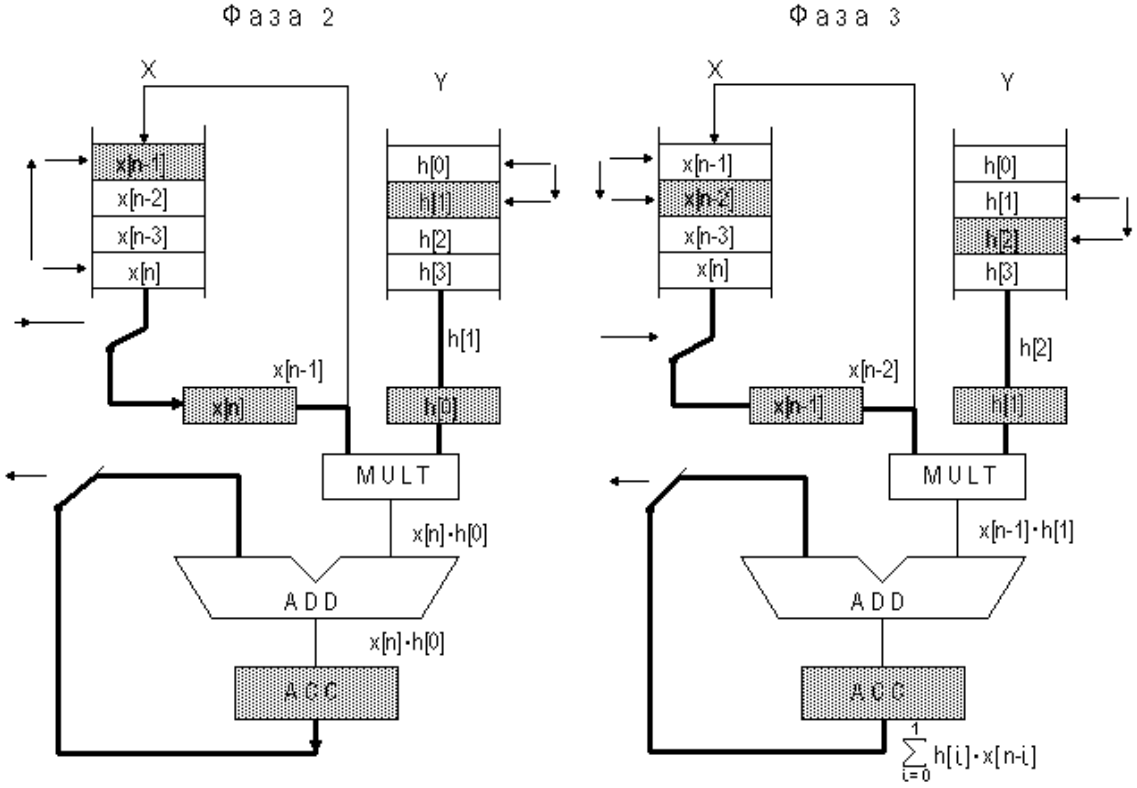


Рис.6.4. Схема выполнения MAC –операции (фазы 2 и 3)

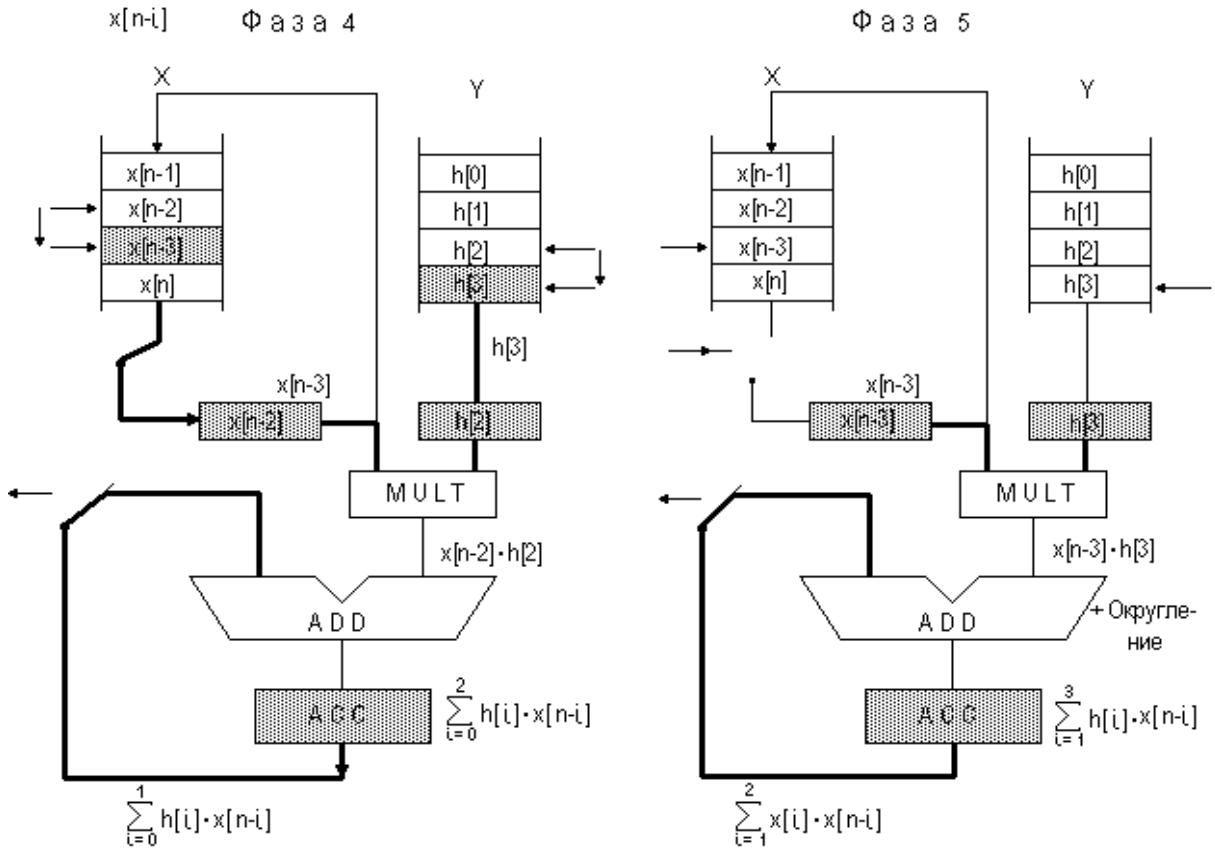


Рис.6.5. Схема выполнения MAC –операции (фазы 4 и 5)

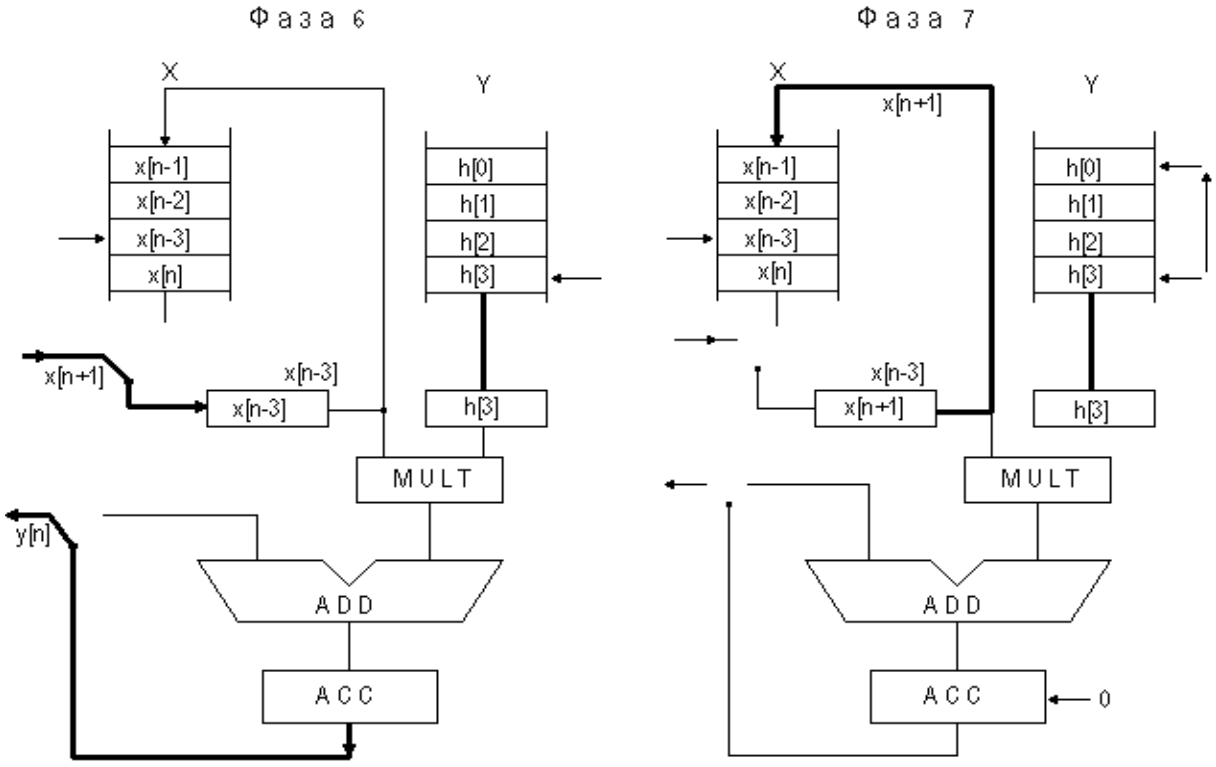


Рис.6.6. Схема выполнения MAC–операции (фазы 6 и 7)

Рассмотренные общие схемы выполнения MAC-операции характерны для многих ЦПОС. Однако детали реализации могут отличаться для процессоров с фиксированной и плавающей запятой.

В процессорах с фиксированной запятой произведение, формируемое на выходе умножителя MULT, имеет удвоенную длину (рис.6.7.). Например, если перемножаются два *n*-разрядных числа, то произведение будет 2*n*-разрядным числом. Чтобы можно было использовать полученное произведение в других операциях с *n*-разрядными аргументами, необходимо иметь возможность преобразования разрядности произведения к исходному значению. ЦПОС с фиксированной запятой обеспечивают представление выходных значений умножителя или аккумулятора в виде двух *n*-разрядных слов, которые хранятся в двух отдельно адресуемых регистрах. Это позволяет программисту выбирать либо старшие, либо младшие разряды результата в зависимости от выбранной формы представления чисел с фиксированной запятой.

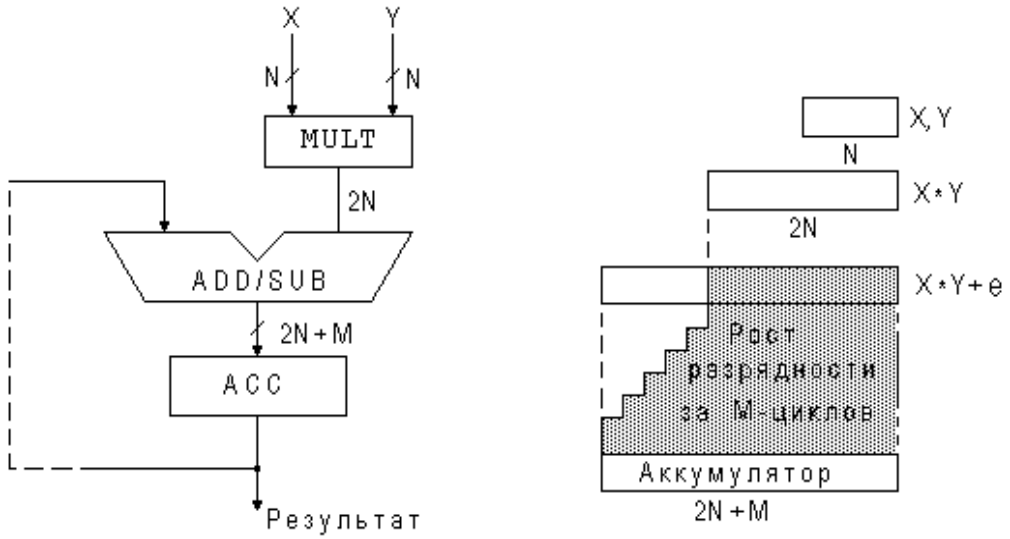


Рис.6.7. Изменение разрядности при выполнения MAC –операции

Содержимое аккумулятора обычно используются в качестве одного из операндов сумматора (рис.6.7). Кроме этого, в аккумуляторе накапливается конечный результат выполнения серии MAC-операций. Поэтому разрядность аккумулятора должна быть на M разрядов больше, чем разрядность умножителя. *Дополнительные разряды* позволяют избежать переполнения аккумулятора при накоплении суммы. Большинство ЦПОС имеют от 4 до 8 дополнительных разрядов. Например, AT&T DSP16xx имеет 4 дополнительных бита (36-разрядный аккумулятор и 32-разрядное произведение), ADSP-21xx фирмы Analog Devices имеет 8 дополнительных битов (40-разрядный аккумулятор и 32-разрядное произведение).

В процессорах, не имеющих дополнительных разрядов, промежуточные результаты должны масштабироваться. Обычно это выполняется

путем сдвига выходных значений умножителя вправо на несколько битов. Например, процессоры фирмы Texas Instruments TMS320C2x и TMS320C5x обеспечивают сдвиг произведения право на 6 разрядов. В общем случае такое масштабирование приводит к потере точности. Однако, если значение суммы, накопленной в аккумуляторе велико, то потери точности незначительны.

Наличие устройств сдвига и дополнительных битов аккумулятора не исключает возможности возникновения переполнений. Один из приемов борьбы с переполнениями основан на использовании рассмотренной ранее операции *ограничения максимальных значений сигнала*. Такая операция можетподдерживаться в ЦПОС с фиксированной запятой на аппаратном уровне. В этом случае специальные схемы ЦПОС обнаруживают ситуацию переполнения и ограничивают значения на уровне максимально возможных положительных или отрицательных значений.

Важными операциями, выполняемыми в ЦПОС, являются *усечение и округление*. Операция усечения сопряжена с простым “отбрасыванием” младших разрядов. Поэтому результат такой операции будет иметь постоянное смещение (рис.1.48). Округление основано на замене исходного значения большей разрядности ближайшим к нему значением меньшей разрядности. Большинство ЦПОС поддерживают указанный тип операции округления. Для реализации операции округления необходимо добавить к исходному значению значение, соответствующее половине наименее значащего бита, а затем выполнить операцию усечения. Некоторые ЦПОС имеют специальные команды для загрузки в аккумулятор соответствующей константы. Когда такой команды нет, загрузка аккумулятора выполняется с помощью обычной команды пересылки.

Рассмотренная операция округления является несимметричной. При округлении чисел, лежащих точно посредине между двумя допустимыми значениями округление выполняется в сторону большего значения. Это приводит к смещению результатов, что может вызвать проблемы при реализации БИХ-фильтров. Поэтому некоторые ЦПОС обеспечивают поддержку *операции симметричного округления*. В этом случае числа, лежащие точно посредине между двумя уровнями, могут округляться как в большую сторону, так и в меньшую. Направление округления зависит от значения разряда числа, который будет являться наименьшим значащим разрядом после округления. Если этот разряд (бит) имеет нулевое значение, то округление выполняется вниз, в противном случае - вверх. Симметричное округление поддерживается в ЦПОС ADSP-21xx и DSP5600x.

Для многих применений выбор способа округления регламентируется техническими стандартами (особенно в области телекоммуникаций).

АЛУ ЦПОС выполняет основные арифметические и логические операции за один цикл команды. Важным параметром АЛУ является разрядность операндов, участвующих в логических операциях. Некоторые ЦПОС позволяет выполнять логические операции над полной длиной слова, хранящегося в аккумуляторе, другие - только над словами, длина которых соответствует общей разрядности процессора. Например, ЦПОС DSP16xx выполняет логические операции над 36-разрядными значениями аккумулятора, в то время как ЦПОС DSP 5600x, имеющий 56-разрядный аккумулятор, оперирует 24-разрядными словами при выполнении логических операций. Если АЛУ не может выполнять логических операций над словами с разрядностью, соответствующей разрядности аккумулятора, то это значительно усложняет программирование. Некоторые ЦПОС имеют наряду с АЛУ, входящим в состав MAC-блока, дополнительное АЛУ. Например, ЦПОС ADSP–21xx и др.

Для ЦПОС с плавающей запятой схема обработки данных в целом соответствует рис.6.7. Однако имеются и некоторые отличия. В большинстве ЦПОС с плавающей запятой допускается также обработка значений в формате с фиксированной запятой. Например, такой возможностью обладают ЦПОС фирм Texas Instruments, Analog Devices, Motorola.

В ЦПОС с плавающей запятой умножитель воспринимает 32-разрядные входные значения, но в отличие от ЦПОС с фиксированной запятой, здесь не требуется увеличивать в два раза разрядность результата. Обычно разрядность мантиссы результата на 8-12 разрядов больше, чем у входных значений.

Для ЦПОС с плавающей запятой характерны переполнения и антипереполнения, которые называют исключительными ситуациями. Возникновение исключительной ситуации приводит к установке соответствующих битов в регистре состояний процессора, а также может приводить к вызову соответствующих подпрограмм обработки прерываний. В случае переполнения большинство ЦПОС с плавающей запятой устанавливают соответствующий флаг состояний и выполняют операцию ограничения уровня сигнала. В том случае, когда возникает антипереполнение (т.е. получено слишком малое значение), процессор обнуляет результат и устанавливает флаг антипереполнения в регистре состояний.

Анализируются также и другие исключительные ситуации, например, деление на ноль. Что касается операции округления, то большинство ЦПОС с плавающей точкой обеспечивают несимметричное округление. Исключение составляет ЦПОС DSP 96002, который поддерживает все виды операций округления в соответствии со стандартом IEEE 754: симметричное округление, округление в сторону $+\infty$, округление в сторону нуля, округление в сторону $-\infty$.

АЛУ ЦПОС с плавающей запятой позволяет выполнять сложения и вычитания, а также вычисление абсолютного значения, вычисление мак-

симума и минимума. Некоторые АЛУ ЦПОС с плавающей запятой обеспечивают дополнительные операции: определение начальных значений для рекурсивных алгоритмов; вычисление функций 1/x, $\sqrt{x}$, (ADSP 210xx, DSP 96002); одновременное вычисление суммы и разности двух значений и размещение результатов в двух разных регистрах (ADSP-210xx, DSP 96002); преобразования форматов чисел с фиксированной запятой в форму с плавающей запятой и обратно (TMS 320C3x, TMS 320C4x, DSP 32xx).

6.3 Организация памяти ЦПОС и режимы адресации

Рассмотренные выше схемы реализации MAC-операции образуют ядро ЦПОС. Для того чтобы эффективно выполнять MAC-операцию, требуется рациональное взаимодействие рассмотренных выше схем с памятью.

Обратимся к примеру нерекурсивного фильтра (рис.6.1). Вычисление выходной реакции фильтра сводится к многократному выполнению MAC-операции. При этом в ходе выполнения одной MAC-операции необходимо осуществить несколько обращений к памяти:

- выбрать MAC-команду из памяти;
- выполнить чтение аргумента X из памяти;
- выполнить чтение аргумента Y из памяти;
- осуществить перемещение значений сигнала вдоль линии задержки КИХ-фильтра. Следовательно, процессор за один цикл команды должен четыре раза обратиться к запоминающему устройству. Такое требование, предъявляемое к ЦПОС, называют *многократным доступом к памяти*.

Классическая архитектура процессора с одним банком памяти не позволяет реализовать многократный доступ. Поэтому ЦПОС строят на основе *Гарвардской архитектуры* (рис. 6.8), в соответствии с которой процессорное ядро взаимодействует с двумя банками памяти A и B.

Обращение к каждому из банков памяти выполняется с помощью двух независимых шин адреса и данных. В Гарвардской архитектуре один из банков памяти используется для хранения программ, а другой для хранения данных. Обычно используется модифицированная Гарвардская архитектура. В этом случае один из банков хранит как программы, так и данные, а другой - только данные. Гарвардская архитектура позволяет процессорному ядру за один цикл команды параллельно обращаться к памяти данных и памяти программ. Указанную архитектуру имеют многие ЦПОС. Некоторые ЦПОС используют три банка памяти с тремя независимыми парами шин адреса–данных. Наличие трех банков позволяет за один командный цикл выполнить три параллельных обращения к памяти: выбрать очередную команду, выбрать аргумент X,

выбрать аргумент У. Подобной архитектурой обладают процессоры фирмы Motorola DSP 5600x, DSP 96002.

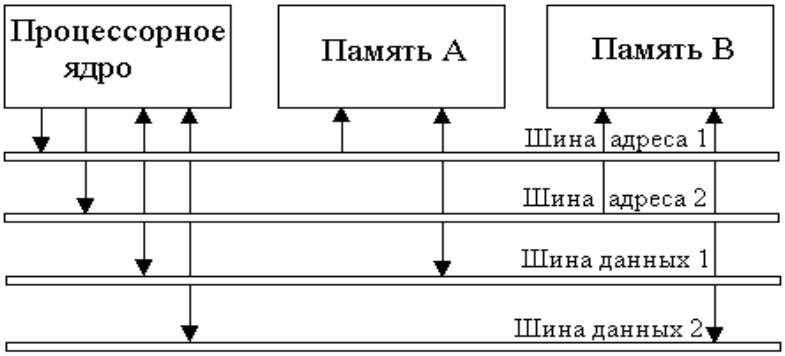


Рис. 6.8. Гарвардская архитектура

Как было отмечено выше, для реализации одной базовой МАС-операции в КИХ-фильтре требуется четыре обращения к памяти. Поэтому в ЦПОС, наряду с Гарвардской архитектурой, применяют и другие меры, чтобы обеспечить многократный доступ к памяти. В частности, в ЦПОС применяют быстродействующую память, которая имеет возможность двух последовательных доступов за один цикл команды. Два таких банка памяти могут обеспечить четыре обращения за командный цикл и соответственно реализовать требования к организации взаимодействия с памятью со стороны алгоритма КИХ-фильтрации.

Другой прием повышения эффективности взаимодействия с памятью состоит в применении *памяти с несколькими портами доступа*. Такая память имеет обычно две независимых пары шин адреса и данных и обеспечивает параллельный доступ по двум адресам.

Комбинация Гарвардской архитектуры с быстродействующей и многопортовой памятью позволяет реализовать требования к эффективной организации взаимодействия с памятью, выдвигаемые со стороны алгоритмов ЦОС. Рассмотренные возможности относятся к памяти, располагаемой на кристалле. Взаимодействие с внешней памятью обычно выполняется с помощью одной шины адреса и одной шины данных. Для этого внутренние шины ЦПОС мультиплексируются.

Некоторые ЦПОС имеют специальный *кэш команд,* который представляет собой память небольшого объема, входящую в состав ядра процессора и предназначенную для хранения команд. Предварительная загрузка команд в кэш позволяет освободить шины процессора для доступа к данным. Возможности кэширования команд значительно отличаются в разных семействах ЦПОС. Простейшие ЦПОС имеют кэш на одну команду. В более сложных ЦПОС емкость кэш-памяти составляет 1К-слов.

Для формирования адресов при обращении к памяти в ЦПОС используются *генераторы адреса*. Генераторы адреса могут формировать один или несколько адресов данных за один цикл команды, не используя

для этого ресурсы основного арифметического устройства, реализующего обработку данных. Это позволяет вычислять необходимые адреса данных параллельно с выполнением арифметических операций, что повышает производительность ЦПОС.

При выполнении любой команды процессор обращается к памяти программ или памяти данных. Способ определения адреса операнда в команде или адреса передачи управления называется *режимом адресации*. В ЦПОС используется широкий спектр режимов адресации: прямая, непосредственная , различные виды косвенной адресации.

В случае *прямой адресации* абсолютный адрес операнда содержится в кодовом слове команды. Например, команда ЦПОС ADSP-21xx

AX0=DM(0800)

загружает значение, расположенное по абсолютному адресу 0800 памяти данных, в регистр AX0.

Непосредственная адресация предполагает, что значение операнда содержится в самой команде. Примером может служить следующая команда ЦПОС ADSP 21xx

AX0=1357.

Здесь константа 1357 загружается в регистр AX0. Если значение константы велико, то она не может быть размещена в одном кодовом слове команды. В этом случае она помещается в следующем после команды слове. Это приводит к необходимости чтения из памяти программ двух слов, что увеличивает время выполнения программы.

Косвенная адресация основана на использовании регистров указателей, содержимое которых является реальным адресом размещения данных в памяти. В этом случае в команде содержится только ссылка на регистр указатель. Так как количество регистров, используемых для косвенной адресации, в ЦПОС не велико, то длина команды косвенной адресации соответствует одному слову. В ЦПОС используются следующие разновидности режимов косвенной адресации: косвенная регистровая адресация; косвенная регистровая адресация с инкрементированием; косвенная регистровая адресация с модульной арифметикой; бит-инверсная адресация. Поясним эти режимы адресации с использованием нотации языка Си.

Косвенной регистровой адресации соответствует следующий оператор присваивания языка Си

A=A+*R;

В соответствии с этим оператором значение, которое хранится по адресу памяти, содержащемуся в регистре R, складывается со значением аккумулятора A. Регистр R рассматривается как указатель, для этого используется обозначение *R.

Косвенной регистровой адресации с инкементированием и декрементированием соответствуют следующие записи на языке Си

```
A=A+(*R)++;
A=A+(*R)--;
```

Здесь также значение, которое хранится по адресу памяти, содержащемуся в регистре R, складывается с аккумулятором A. После извлечения значения из памяти содержимое регистра-указателя увеличивается (++) или уменьшается (--) на единицу. Некоторые ЦПОС позволяют увеличивать или уменьшать значения регистра-указателя не на единицу, а на другую величину, значение которой хранится в дополнительном регистре. Кроме этого, ряд процессоров позволяет изменять значение регистров-указателей до обращения к памяти.

Косвенная регистровая адресация с модульной арифметикой используется при организации в памяти кольцевых буферов. *Буфер* представляет последовательность из однотипных элементов данных, размещенных в смежных ячейках памяти. Если элементы данных обрабатываются в порядке их записи в память, то буфер соответствует структуре данных, называемой *очередью*. Для доступа к значениям, хранящимся в буфере используются указатели. Каждый раз после обращения к буферу значение указателя изменяется с некоторым шагом. При достижении конца буфера значение указателя изменяется на начальное. Указанный механизм можно описать с помощью условного оператора

```
if (pointer + step < BkEnd)
        pointer = pointer + step;
else    pointer = BkBegin;
```

Здесь переменная BkBegin — обозначает начальный адрес, а BkEnd — конечный адрес области памяти, в которой размещен буфер. Переменная pointer является указателем, а переменная step соответствует шагу, с которым изменяется значение указателя. Приведенная схема адресации используется, например, в ЦПОС TMS320C5x.

Возможен и другой механизм адресации кольцевого буфера, используемый в ЦПОС ADSP 21xx, ADSP 21xxx, TMS320C3x, TMS320C4x. В этих ЦПОС не задается конечный адрес области памяти, отведенной под буфер, а фиксируется длина буфера в специальном регистре. Значение, соответствующее длине буфера, называется *модулем*. При увеличении начального адреса на величину модуля происходит переход к начальному адресу, что и составляет суть косвенной адресации с модульной арифметикой.

Одним из особых способов адресации, используемых в ЦПОС, является *бит-инверсная адресация*. Бит-инверсная адресация позволяет выполнять переупорядочение входных или выходных данных в алгоритмах БПФ. Суть бит-инверсной адресации описана в § 1.5.4.

6.4 Архитектурные особенности основных семейств ЦПОС

6.4.1. ЦПОС фирмы Motorola

Перечень основных типов ЦПОС, выпускаемых фирмой Motorola приведен в табл.6.1. Рассмотрим структурные особенности данных процессоров на примерах семейств ЦПОС с фиксированной запятой DSP5600x и с плавающей запятой DSP96002.

DSP5600x. Упрощенная структурная схема ЦПОС DSP5600x приведена на рис. 6.9. ЦПОС этого семейства обладают следующими характеристиками:

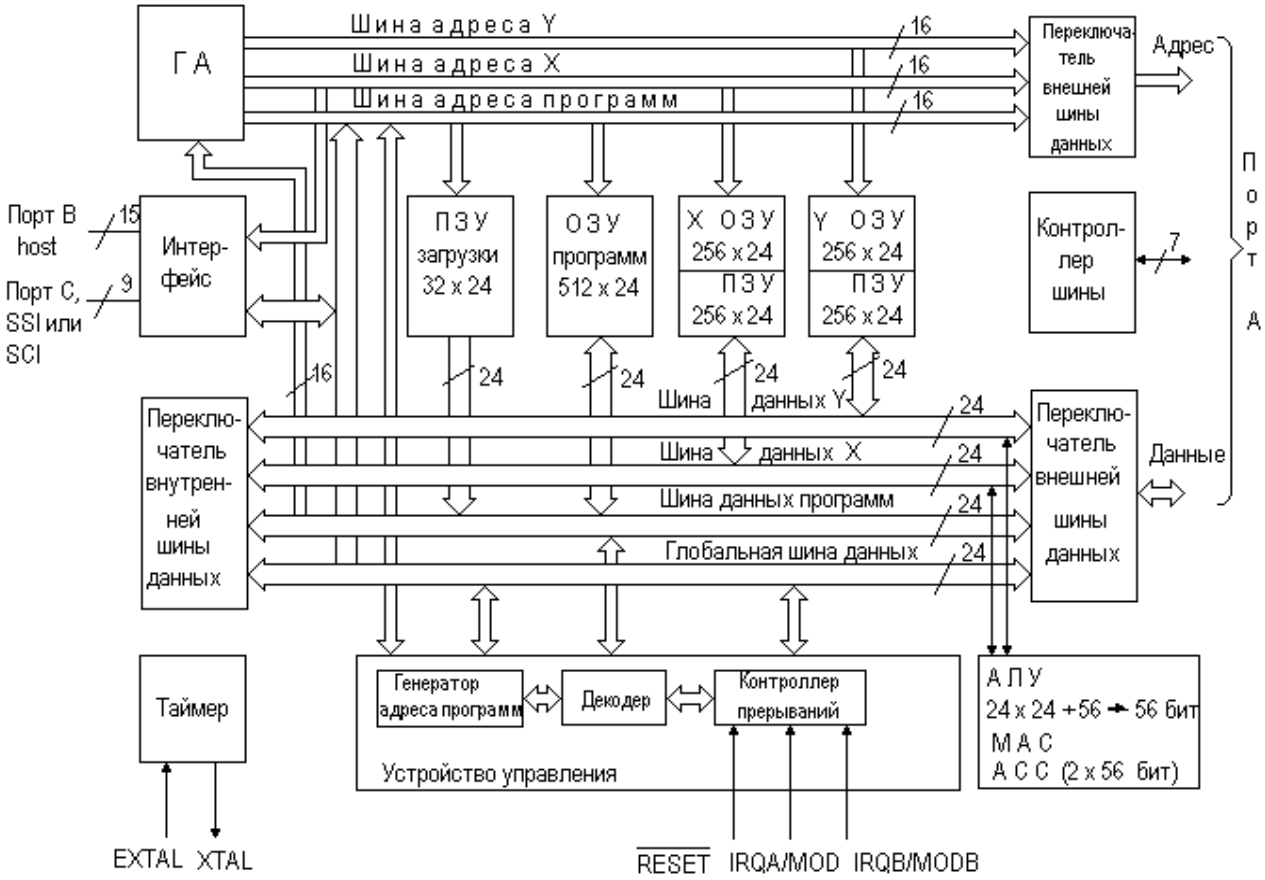


Рис. 6.9. Структурная схема ЦПОС DSP 5600x

- 24-разрядное представление данных;
- выполнение MAC-операции над 24-разрядными данными за один цикл команды ;
- наличие двух аккумуляторов;
- наличие двух блоков ОЗУ данных (X и Y) ;
- наличие двух блоков ПЗУ (X и Y), предназначенных для хранения таблиц функций (A- и μ -законы компандирования, sin(x));
- возможность вычисления произведения с удвоенной точностью представления данных (48x48 бит);

- аппаратная поддержка выполнения циклических алгоритмов;
- быстрая обработка прерываний (2 цикла);
- наличие ПЗУ загрузки программ из внешней памяти;
- поддержка целочисленной и дробной арифметики;
- аппаратная поддержка блочных операций с плавающей запятой.

Архитектура ЦПОС DSP5600x позволяет на аппаратном уровне поддерживать обработку простых структур данных, состоящих из двух элементов. Пара элементов данных, входящих в структуру, может рассматриваться как значение сигнала и коэффициент фильтра, как вещественная и мнимая часть комплексного числа и т.п.

ЦПОС DSP5600x обладают *"дублированной" гарвардской архитектурой*. В состав процессоров семейства входят (рис.6.9) по два блока ОЗУ данных, ПЗУ данных, генераторов адреса, аккумуляторов, устройств сдвига. Процессор может одновременно обращаться к ОЗУ программ и двум блокам ОЗУ данных (X и Y). В состав процессора включены три шины адреса: две однонаправленные 16-разрядные шины адреса блоков памяти X и Y; двунаправленная шина адреса памяти программ. Для передачи и обмена данными внутри процессора используются четыре двунаправленные шины данных: шина данных X; шина данных Y; шина данных памяти программ; глобальная шина данных. Все шины данных являются 24-разрядными.

В ЦПОС DSP 5600x отсутствует кэш команд, так как три пары шин адреса/данных позволяют осуществлять одновременную выборку очередной команды и пересылку двух операндов по шинам данных X и Y. Для взаимодействия с внешними устройствами внутренние шины данных и адресов процессора объединяются с помощью мультиплексоров (переключателей) в одну пару внешних шин адреса/данных (рис. 6.9) Внешняя шина адреса является 16-разрядной, а шина данных - 24-разрядной. Указанные шины совместно с управляющими сигналами образуют порт A, имеющий 47 выводов.

В состав АЛУ процессора (рис.6.10) входит два 56-разрядных аккумулятора с 8-ю дополнительными разрядами для исключения переполнений при циклическом выполнении MAC-операций. Каждый аккумулятор представлен тремя регистрами: A2, A1, A0 и B2, B1, B0. Аккумуляторы участвуют в MAC-операциях в соответствии с общими схемами, изображенными на рис. 6.3-6.6. При выполнении MAC-операции для каждого из аккумуляторов используется своя пара 24-разрядных регистров , содержащих значения X и Y операндов. На рис. 6.10 указанные пары регистров обозначены соответственно X0,Y0 и X1,Y1.

Если требуется сохранить 56-разрядное значение аккумулятора в 24-разрядных регистрах или памяти, то с помощью сдвигающего устройства выполняются необходимые операции сдвига или ограничения (рис.6.10).

Блок манипуляции битами, входящий в состав АЛУ, выполняет простые операции преобразования битов 24-разрядных слов.

Блоки памяти данных X и У имеют непересекающиеся адресные пространства, каждое из которых включает в себя как область адресов ОЗУ данных, так и область адресов ПЗУ данных

ЦПОС DSP 5600x имеет развитую систему дополнительных интерфейсов для взаимодействия с другими процессорами. В состав ЦПОС входят: последовательный коммуникационный интерфейс (SCI), поддерживающий режимы синхронного и асинхронного обмена 8-,12-,16- и 24-разрядными словами; синхронный последовательный интерфейс (SSI), предназначенный для дуплексной связи.

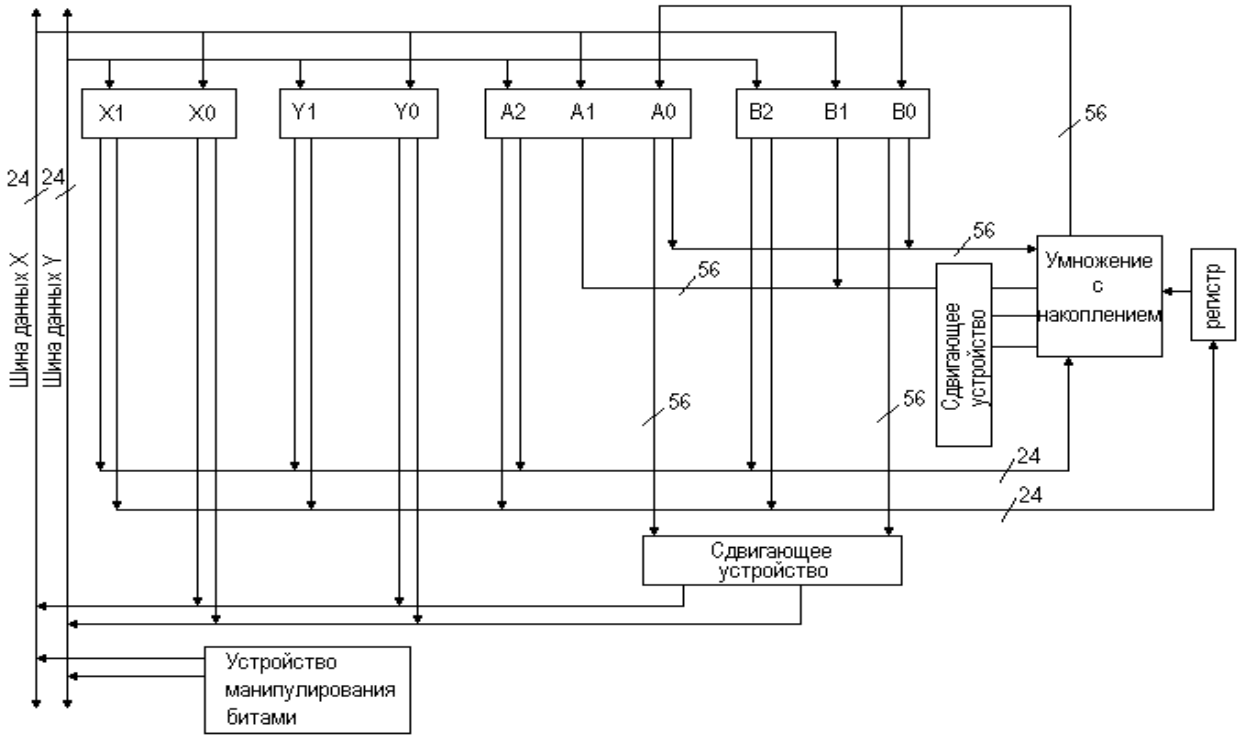


Рис.6.10. Структурная схема АЛУ DSP5600x

Процессор DSP 5600x имеет возможность обработки одного немаскируемого и двух маскируемых внешних прерываний.

Тактовая частота, на которой работает процессор, может выбираться в широких пределах от 12.2 кГц до 80 МГц. Такой широкий диапазон возможен благодаря синтезатору частот с фазовой автоподстройкой частоты, входящему в состав процессора.

Процессор поддерживает широкий спектр режимов адресации: прямую адресацию регистров, прямую адресацию памяти, непосредственную адресацию, косвенную регистровую адресацию, бит-инверсную адресацию.

Все команды процессора можно разделить на шесть групп:

- арифметических операций (AND, SUB,MUL,DIR,MAC,CMP,…);

- логических операций (AND, OR, LSR, OR,…);
- манипуляции битами (BSET,BTST,…);
- управления циклами (DO, ENDDO);
- адресации памяти (LUA,MOVE,…).

Команды процессора имеют простой формат:

<код операции> <операнд> <передача по X-шине> <передача по Y-шине>

Каждая команда параллельно с выполнением заданной операции над операндом может выполнять пересылку данных по шине данных X и по шине данных Y. Например, команда, записанная в общей форме

MAC X0,Y0,A X:R0+,X0 Y:R4+,Y0 ,

выполняет:

1) MAC-операцию : $A \leftarrow A + X0 \cdot Y0$;

2) пересылку данных по шине данных X из ячейки памяти, адресуемой регистром-указателем R0, в регистр X0 с последующим инкрементом адреса, находящегося в R0;

3) пересылку данных по шине данных Y из ячейки памяти, адресуемой регистром-указателем R4, в регистр Y0 с последующим инкрементом адреса, находящегося в R4.

Возможности процессора DSP 5600x по реализации основных алгоритмов ЦОС приведены в табл. 6.2.

Таблица 6.2 - Параметры программ для DSP5600x

Алгоритм	Длина программы, слов	Кол-во циклов
КИХ-фильтр n-го порядка (веществ. коэфф.)	7	N+7
КИХ-фильтр n-го порядка (компл. коэфф.)	12	4N+9
Биквадратное звено	7	7
N биквадратных звеньев	11	4N+8
БПФ по основанию 2	15	6N+10

ЦПОС DSP96002 является 32-х разрядным процессором с плавающей запятой, архитектура которого подобна архитектуре ЦПОС DSP5600x. Процессор 96002 имеет семь внутренних шин. В их состав входят шины памяти программ, шины блоков памяти X и Y, а также глобальная шина данных и шина данных ПДП. Контроллер ПДП при пересылке данных не использует другие ресурсы ЦПОС.

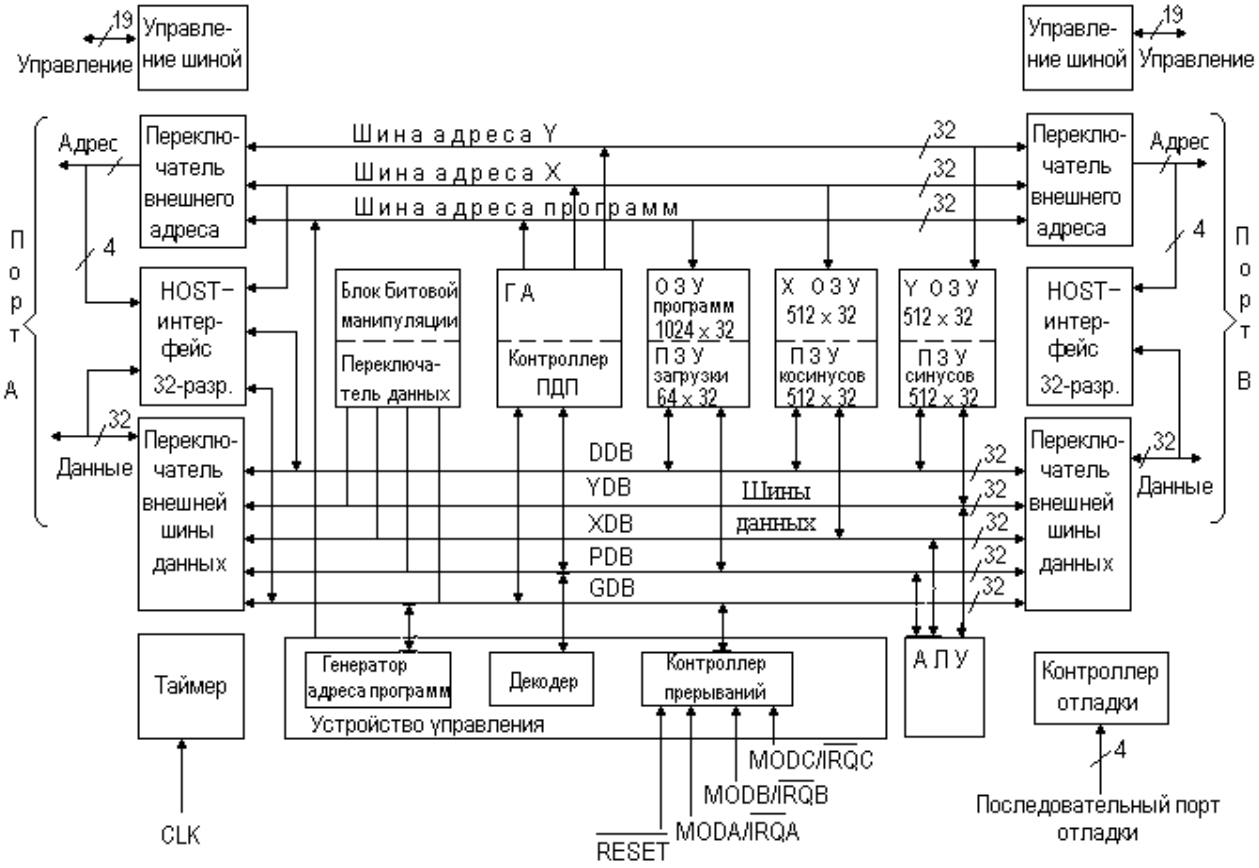


Рис. 6.11. Структурная схема процессора DSP 96002.

Аналогично DSP56000 процессор DSP96002 имеет блоки ПЗУ и ОЗУ, которые обеспечивают хранение значений коэффициентов и переменных, используемых при выполнении MAC –операции. Перед выполнением MAC-операции данные должны быть размещены в регистровом файле, включающем десять 96 разрядных регистров.

Вычислительная часть процессора представлена умножителем, сумматором (который может выполнять как сложение, так и вычитание при реализации операции БПФ), блоком логических операций, кольцевым сдвигающим устройством. Все эти устройства поддерживают целочисленную арифметику и арифметику с плавающей запятой. При этом для представления мантиссы отводится 32 разряда, а для представления порядка 11 разрядов. DSP96002 поддерживает форматы IEEE для представления чисел с плавающей точкой.

ЦПОС 96002 имеет два генератора адреса, которые могут функционировать параллельно. В состав каждого из генераторов входит четыре набора 32-разрядных регистров. Каждый набор состоит из трех регистров : указателя адреса, регистра смещения и регистра модификации адреса.

Доступ к этим регистрам осуществляется через глобальную шину данных. Процессор поддерживает прямую адресацию регистров, прямую адресацию памяти, различные виды косвенной регистровой адресации, непосредственную адресацию. Генераторы адреса также обеспечивают режимы модульной и бит-инверсной адресации.

Набор команд процессора включает команды: управления циклами; проверки значений битов двоичных слов; условного выполнения подпрограмм; преобразования целых чисел в формат с плавающей запятой и обратно ; формирования начальных значений при вычислении квадратного корня и генерации случайных чисел. Набор команд процессора не содержит команд условного выполнения инструкций.

6.4.2 ЦПОС фирмы Texas Intruments

Фирма Texas Intruments выпускает широкий спектр ЦПОС (табл.6.1) , которые в соответствии с терминологией Texas Intruments, называются поколениями. Поколения процессоров группируются по четырем направлениям:

- процессоры с фиксированной запятой: C1x, C2x, C2xx, C5x, C54x, C62x;
- процессоры с плавающей запятой: C3x, C4x, C67x;
- мультимедиа-видеопроцессоры : C8x;
- процессоры специализированных приложений: AV 7100, AV710.

Процессоры с фиксированной запятой C1x, C2x, C2xx, C5x совместимы на уровне исходных кодов. При этом множество команд процессора C1x является подмножеством команд процессора C2x и т.д.

В главе 7 детально рассматривается архитектура и программирование процессоров с фиксированной запятой на примере семейства процессоров TMS320C2x. Поэтому здесь приведем в качестве примера архитектурные особенности процессоров с плавающей запятой TMS320C3x и мультимедиа процессоров TMS320C8x.

ЦПОС TMS320C3x . Семейство процессоров C3x является первым поколением 32-разрядных процессоров фирмы Texas Intruments с плавающей запятой. Они предназначены для решения широкого спектра задач ЦОС в различных областях, включая цифровую обработку аудиосигналов, передачу данных, автоматическое управление и контроль.

Процессоры TMS320C3x характеризуются интеграцией фон-Неймановской архитектуры с 32 разрядным ядром ЦОС (рис.6.12), аппаратно поддерживающим обработку значений в формате с плавающей запятой. Отметим, что процессоры C3x не поддерживают формат данных с плавающей запятой, соответствующий стандарту IEEE.

Процессоры C3x состоят из трех подсистем: центрального процессорного блока; подсистем доступа к памяти и ввода-вывода. Подсистема

доступа к памяти предоставляет отдельные шины для доступа к памяти программ, памяти данных и выполнения ПДП. Контроллер ПДП имеет свою отдельную шину данных и работает параллельно с центральным арифметическим устройством (ЦАУ). Наличие отдельных шин для доступа к памяти программ и памяти данных позволяет осуществлять выборку очередной команды и двух значений данных. При этом процессор может выполнять передачу и прием данных из подсистемы ввода-вывода.

Процессор имеет два блока ОЗУ емкостью 1К-слов и ПЗУ емкостью 4К-слов. Внешняя шина адреса процессора является 24-разрядной, что позволяет адресовать память емкостью 16М-слов. Семейство процессоров С3х включает три основные модели: С30, С31,С32. В микросхемах С31 и С32 вместо ПЗУ данных применяется ПЗУ загрузки программ.

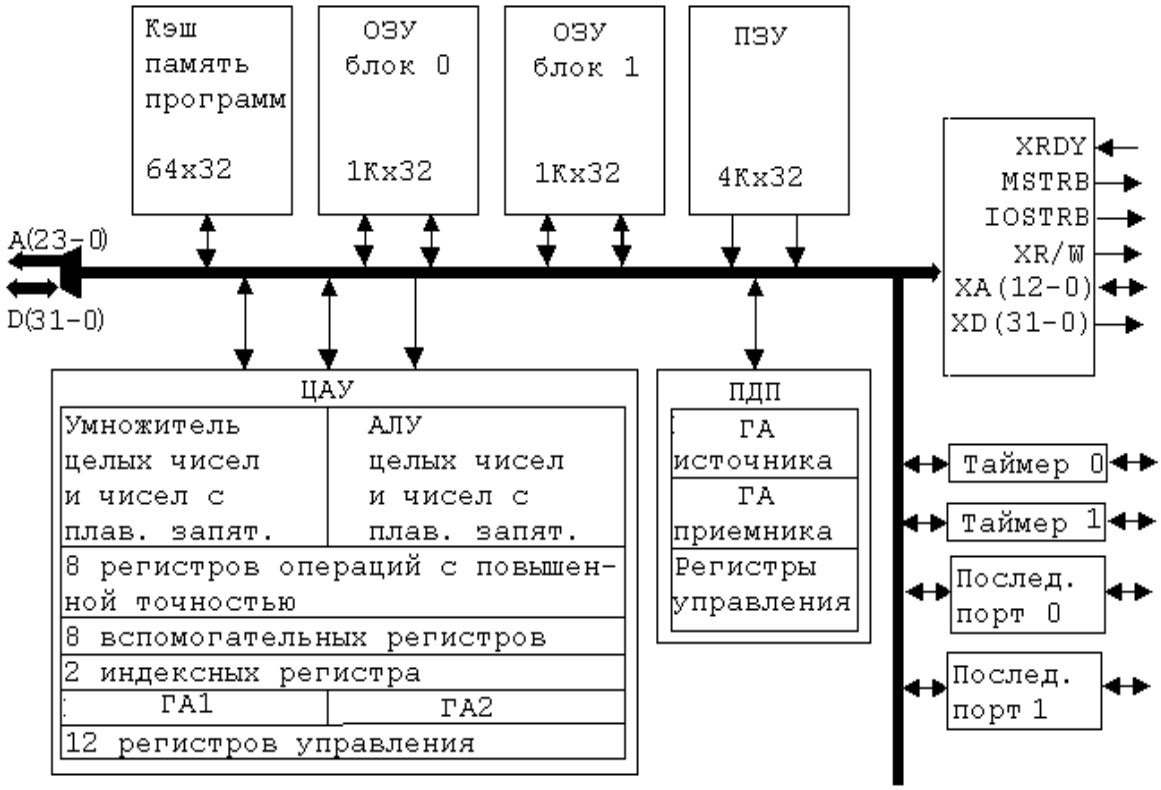


Рис.6.12. ЦПОС TMS320C30

ЦАУ содержит умножитель, АЛУ, устройства сдвига, 8 регистров для выполнения операций с повышенной точностью. Оно имеет свои собственные шины для пересылки данных в процессе вычислений. Эти шины позволяют пересылать данные между внутренними регистрами ЦАУ, умножителем, устройствами сдвига, АЛУ. ЦПОС семейства выполняют MAC-операцию за один цикл команды.

Процессоры TMS320C3x поддерживают прямую адресацию регистров, страничную адресацию памяти, косвенную регистровую адресацию, непосредственную, кольцевую адресацию и бит–инверсную адресацию.

ЦПОС TMS320C8x. Данный процессор ориентирован на применение в мультимедиа видео приложениях и состоит из четырех процессоров ЦОС и управляющего RISC процессора (УП). Процессоры ЦОС функционируют параллельно, обеспечивая общую производительность до 5x50 млн. операций в секунду (рис.6.13). Высокоскоростная шина соединяет процессоры ЦОС с шестнадцатью блоками ОЗУ емкостью 2К (11–4К блоков для TMS320C82). Любой из процессоров ЦОС может быть соединен с любым из блоков ОЗУ. Высокоскоростная шина (коммутатор) обеспечивает от 9 до 12 одновременных обращений к памяти в пределах цикла команды (по три обращения для каждого из процессоров ЦОС, два обращения для управляющего процессора и одно для контроллера передачи). Максимальная скорость передачи по шине достигает 4,2 Гбит/с.

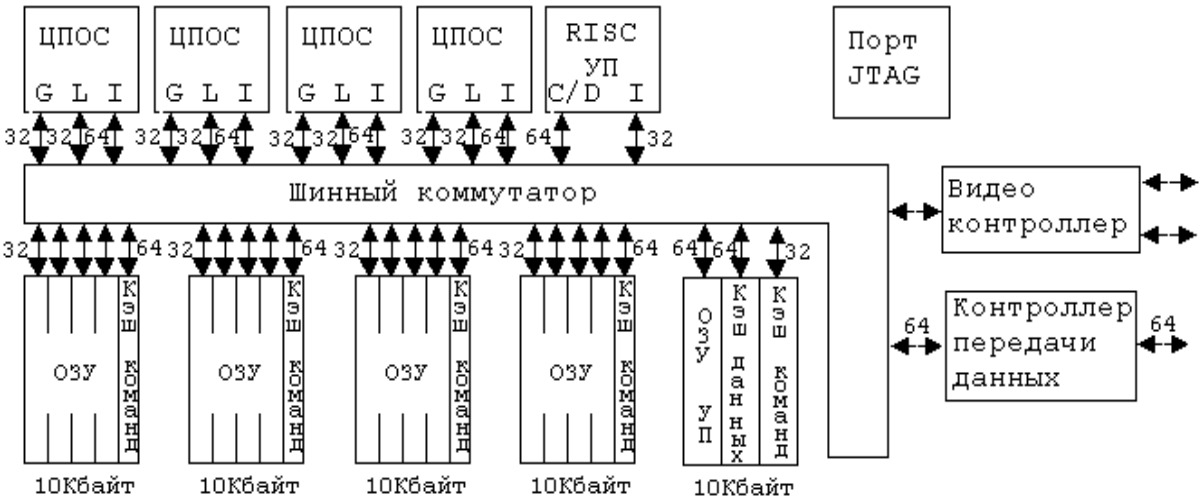


Рис.6.13. ЦПОС TMS320C8x

Процессоры ЦОС, входящие в состав семейства C8x, управляются 64-х разрядными командами. Каждый процессор ЦОС имеет 32-разрядное АЛУ, 16-разрядный умножитель, два независимых генератора адреса. АЛУ может быть разделено на два 16-разрядных или четыре 8-разрядных блока. Умножитель может вычислять произведение двух 16-разрядных слов или двух пар 8-разрядных слов. Дополнительно аппаратные средства поддерживают манипуляции битами и обработку пикселов изображений.

Управляющий процессор содержит 64-разрядное АЛУ с плавающей запятой и регистровый файл емкостью 31x32 бит. АЛУ состоит из умножителя, выполняющего операции с одинарной точностью, и сумматора, выполняющего операции с двойной точностью. АЛУ поддерживает век-

торную обработку на аппаратном уровне и обеспечивает накопление результатов векторной обработки в четырех аккумуляторах. УП функционирует в конвейерном режиме. Конвейер является трехэтапным.

В состав ЦПОС TMS320C8x входит интегрированный контроллер передачи данных для взаимодействия с различными видами внешней памяти. Видеоконтроллер минимизирует затраты на разработку видеоприложений.

6.4.3 ЦПОС фирмы Analog Devices

Фирма Analog Devices выпускает ЦПОС с фиксированной и с плавающей запятой (табл.6.1). Процессоры с плавающей запятой подробно рассматриваются в главе 8. Поэтому ниже рассмотрим только процессоры с фиксированной запятой.

Семейство процессоров ADSP-21xx представляет собой набор однокристальных микропроцессоров с общей базовой архитектурой, оптимизированной для цифровой обработки сигналов и для других приложений, требующих числовых вычислений с высокой скоростью. Все процессоры семейства используют ядро цифровой обработки сигналов, соответствующее СБИС ADSP 2100. Отличие процессоров семейства друг от друга определяется набором дополнительных устройств, расположенных на кристалле. К ним относятся: память, последовательные порты, интерфейсы прямого доступа к памяти (ПДП) и сопряжения с основным процессором (хост-интерфейс), таймер, аналоговый интерфейс [25].

Рассмотрим обобщенную структурную схему процессоров семейства ADSP-21xx (рис.6.14). Структурные схемы конкретных процессоров отличаются от приведенной набором дополнительных устройств. Каждый процессор семейства содержит три независимых вычислительных устройства: арифметическо-логическое устройство (АЛУ), умножитель с аккумулятором (УСА / MAC), и сдвигающее устройство (СУ).

Вычислительные устройства непосредственно оперируют 16-разрядными данными и могут поддерживать работу с данными расширенной точности. АЛУ обеспечивает выполнение типового набора арифметических и логических операций, а также обеспечивает реализацию примитивов операции деления. УСА позволяет выполнять операции умножения, умножения со сложением и вычитанием. СУ выполняет логические и арифметические сдвиги, нормализацию и денормализацию, операции над порядком чисел. СУ может быть использовано при выполнении операций над числами, представленными в различных форматах, в том числе и над числами с плавающей запятой. Все вычислительные устройства работают параллельно. При этом выходные значения любого из устройств могут, благодаря общей шине результатов (ШР), служить входными значениями для других устройств. Все три вычислительных

устройства имеют входные и выходные регистры для хранения операндов и результатов выполнения операций. Эти регистры доступны через шину данных памяти данных (ШД ПД).

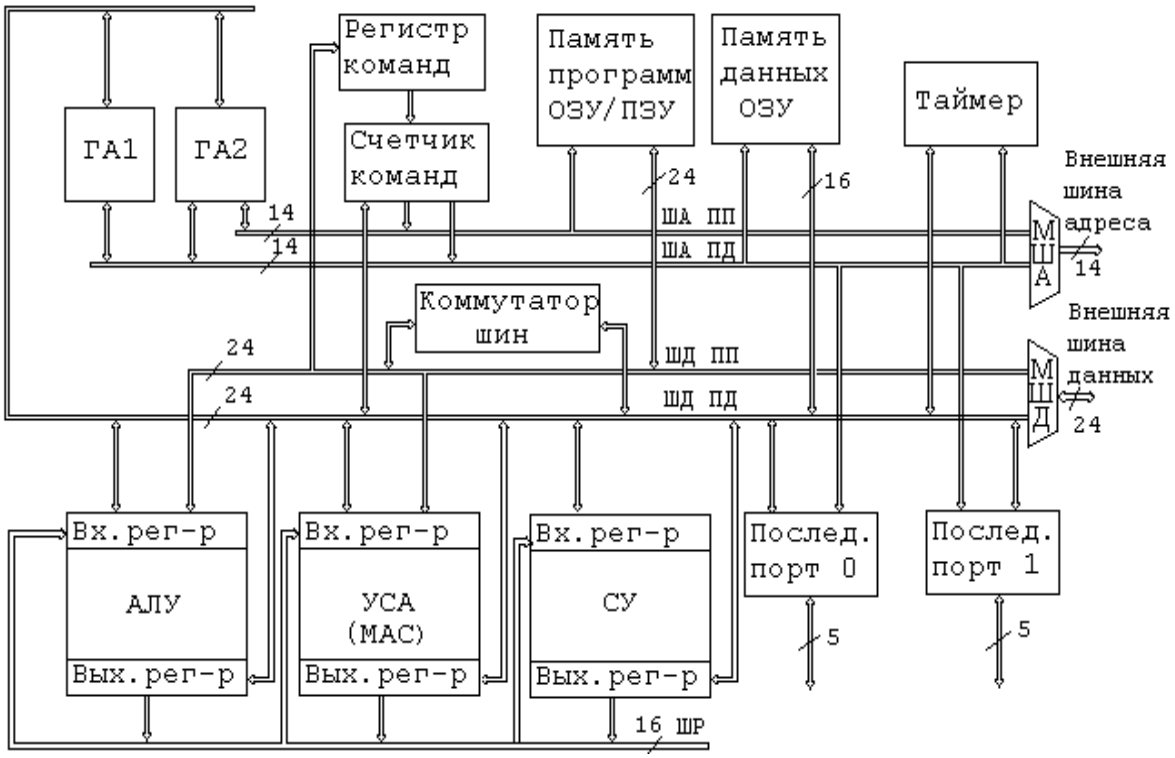


Рис.6.14. ЦПОС ADSP 21xx

Эффективное использование вычислительных устройств обеспечивается многофункциональным счетчиком команд и двумя генераторами адресов данных (ГА). Многофункциональный счетчик команд поддерживает условные переходы, вызовы и возвраты из подпрограмм и формирует адрес следующей команды. Он имеет внутренний счетчик циклов, стек циклов и может выполнять циклы без дополнительных команд перехода.

Генераторы адресов данных управляют указателями адресов. Каждый из генераторов управляет четырьмя указателями. После использования указателя для доступа к внешним данным, он может автоматически модифицироваться значением, хранящимся в заданном регистре генератора. При этом с каждым из указателей может быть связана длина области памяти для автоматической реализации кольцевых буферов. Наличие двух генераторов адресов позволяет процессору формировать одновременно два адреса для выборки операндов в одном цикле команды.

Эффективная пересылка данных в процессоре обеспечивается пятью внутренними шинами:

- шиной адреса памяти программ (ША ПП);
- шиной адреса памяти данных (ША ПД);

- шиной данных памяти программ (ШД ПП);
- шиной данных памяти данных (ШД ПД);
- шиной результатов (ШР).

Шины адресов предназначенны для передачи адресов памяти программ или памяти данных, а также дополнительных устройств. Шины данных обеспечивают передачу значений, извлекаемых из памяти по заданным адресам, между устройствами, расположенными на кристалле. Каждая пара указанных шин мультиплексируется в одну внешнюю шину адреса или данных с помощью соответствующего мультиплексора (МША-мультиплексор шины адреса; МШД-мультиплексор шины данных). Это обеспечивает взаимодействие процессора с внешней памятью. Ширина внешней шины адреса 14 разрядов, что обеспечивает доступ к 16К-словам внешней памяти. Внешняя шина данных является 24-разрядной. Это позволяет загружать через внешнюю шину данных как 16-разрядные данные, так и 24-разрядные команды.

Процессор поддерживает прямую и косвенную адресацию памяти данных. В случае прямой алресации адрес записывается непосредственнно в команде. При косвенной адресации используется указатель на адрес. Память программ адресуется только с помощью косвенной адресации. При этом в памяти программ могут храниться как команды, так и данные. Это позволяет процессору выбирать в одном цикле два операнда. Один выбирается из памяти данных, другой - из памяти программ.

Коммутатор шин позволяет пересылать данные с ШД ПП на ШД ПД и обратно.

Практически все процессоры семейства имеют по два последовательных порта (за исключением ADSP 2105) с двойной буферизацией и с синхронной передачей данных. Последовательные порты поддерживают работу со словами длиной от 3- до 16-бит и компандированием по А или μ -закону в соответствии с рекомендацией МКТТ G.711.

Программируемый таймер обеспечивает периодическую генерацию прерываний. Таймер содержит регистр – счетчик. Прерывание генерируется, когда значение в регистре – счетчике становится равным нулю.

Дополнительно в состав некоторых моделей процессоров может входить контроллер прямого доступа к памяти (ПДП), обработчик прерываний, хост-интерфейс, аналоговый интерфейс. ПДП имеет 16- разрядную шину адреса/данных и обеспечивает непосредственный доступ к памяти программ или памяти данных, расположенной на кристалле. Обработчик прерываниий реагирует на четыре внешних прерывания, которые имеют определенный приоритет. Каждое из прерываний может быть замаскированно и програмно настроено на срабатывание по фронту или уровню сигнала. Порт хост-интерфейса (HIP) – параллельный порт ввода-вывода, обеспечивающий взаимодействие с основным процессором (хост-процессором). Порт хост-интерфейса работает с 8- или 16-

разрядными шинами данных. Аналоговый интерфейс включает аналого-цифровые и цифро-аналоговые преобразователи и предназначен для организации взаимодействия с аналоговыми подсистемами.

На рис.6.15 приведена обобщенная схема однопроцессорной системы на базе ADSP 21xx, содержащая два последовательных устройства, байтовое ППЗУ загрузки и внешнее ОЗУ памяти программ и памяти данных [25].

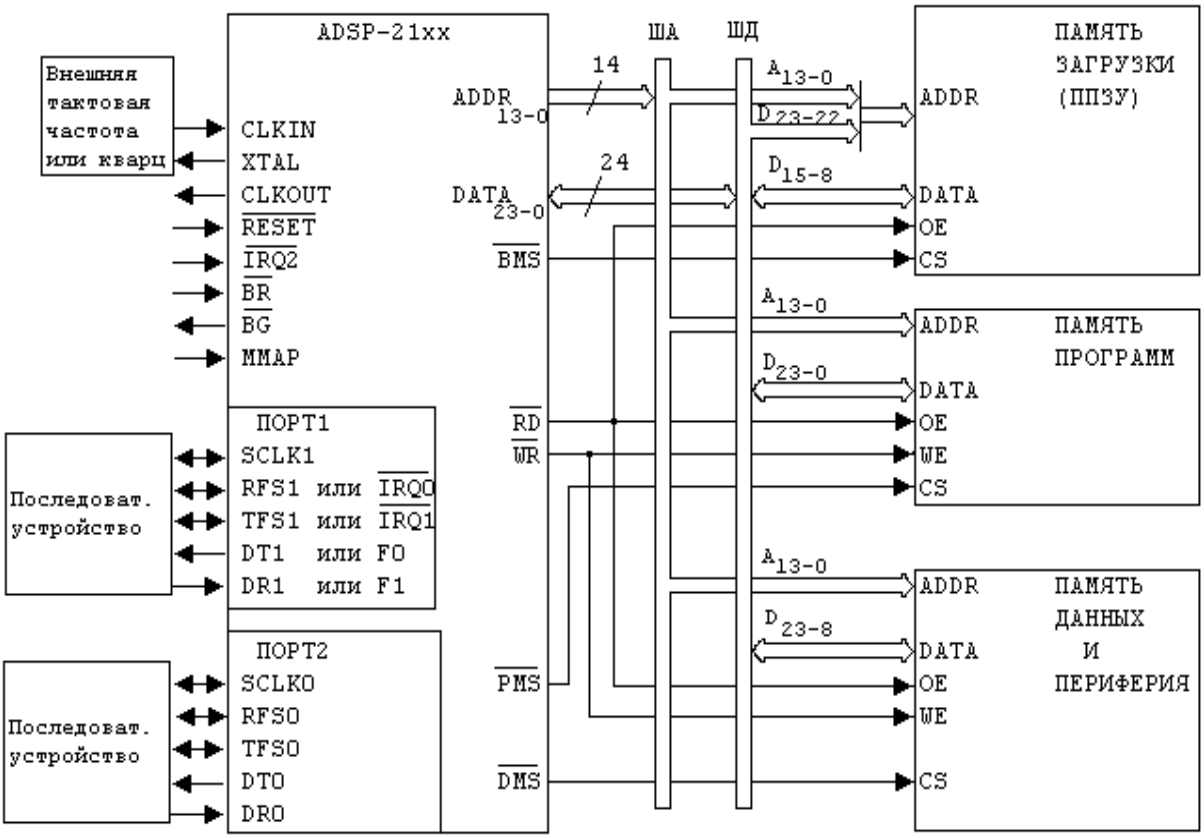


Рис.6.15. Структурная схема однопроцессорной системы на базе ADSP21xx

Синхронизация процессора выполняется ТТЛ-совместимыми синхроимпульсами, подаваемыми от генератора тактовых импульсов (ГТИ) на вход CLKIN. Вход XTAL используется в том случае, если выполняется синхронизация с помощью кварцевого резонатора, который подсоединяется между входами CLKIN и XTAL. Выходной сигнал CLKOUT формируется процессором для синхронизации внешних устройств. Период сигнала CLKOUT соответствует прцессорному циклу комманды.

Интерфейс с внешними устройствами и памятью поддерживается тремя шинами: управления, адреса (ША) и данных (ШД). ША позволяет непосредственно адресовать 16К-слов памяти. Для выбора соответствующего устройства памяти используются сигналы: BMS – выбор ППЗУ загрузки; PMS – выбор памяти программ; DMS – выбор памяти данных.

ППЗУ загрузки имеет 8-разрядную шину данных, через которую может быть побайтно загружена внутренняя память программ процессо-

ра. Процессоры семейства ADSP 21xx имеют различные возможности взаимодействия с ППЗУ загрузки, отличающиеся способом формирования адреса. На рис.6.15 адрес байта в ППЗУ формируется с помощью шины данных (разряды D23-22) и шины адреса (A13-0). Это позволяет адресовать до 16К восьмиразрядных слов. Данные возможности реализованы в ADSP-2101, ADSP-2103, ADSP-2111, ADSP-2115. Другие модели процессоров могут использовать дополнительные разряды шины данных для расширения адресного пространства.

6.5 Инструментальные средства разработки систем на основе ЦПОС

ЦПОС, по существу, есть не что иное как микропроцессор специального применения. Как и для любого другого микропроцессора, создание систем на основе ЦПОС невозможно без инструментальных средств поддержки разработок. Данные средства позволяют разрабатывать архитектуру аппаратных средств и осуществлять создание и отладку соответствующего программного обеспечения. Они применяются на всех этапах разработки - от исследований до изготовления и тестирования. Во многом успех создания систем на основе ЦПОС определяется наличием развитых инструментальных средств поддержки разработок. В настоящее время практически для всех выпускаемых ЦПОС имеются такие средства.

Инструментальные средства разработок систем на основе ЦПОС разделяются на программные и аппаратные [31]. Состав средств, входящих в каждую группу, различен для разных семейств ЦПОС. Типовой набор программных средств разработки включает следующие программы: ассемблер, редактор связей (линкер), имитатор (симулятор), программатор ППЗУ, ассемблерные библиотеки, оптимизирующий компилятор с языка Си, отладчики программ на языке Си, библиотеки программ на языке Си.

Ассемблер транслирует исходную программу на языке ассемблера в программу, подлежащую исполнению в объектном коде, т.е. коде непосредственно воспринимаемом ЦПОС. Обычно ассемблер генерирует перемещаемый объектный код, который затем размещается в памяти системы ЦОС с помощью редактора связей.

Редактор связей позволяет разрабатывать программу в виде автономных модулей, связав которые воедино, можно получить полный код программы. Редактор связей выполняет объединение ассемблированных объектных файлов и дополнительных объектных файлов, хранящихся в библиотеках, в единую исполняемую программу.

Ассемблерные библиотеки содержат типовые арифметические подпрограммы и подпрограммы ЦОС. Для создания библиотек используются специальные программы-библиотекари.

Имитатор программным способом моделирует ЦПОС на уровне системы команд, позволяя тем самым проверять и отлаживать программы без наличия аппаратных средств. Он использует объектный код, вырабатываемый ассемблером и редактором связей. Имитатор моделирует структуру памяти и возможности ввода-вывода в системе ЦОС. Для этого он использует специальный файл, определяющий архитектуру разрабатываемой системы. Обычно программа-имитатор имеет оконный интерфейс для взаимодействия с пользователем. В окнах имитатора отображаются содержимое внутренних регистров ЦПОС и состояние памяти. Имитатор представляет мощное средство отладки, допускающее пошаговое выполнение команд, имитацию прерываний, установку точек программного прерывания, отображение содержимого памяти.

Компилятор с языка Си переводит исходные тексты программ, написанные на стандартной (ANSI) версии языка Си либо в объектный код, либо в код на языке ассемблера. Компилятор обычно допускает включение в программу, написанную на языке Си, разделов на языке ассемблера. Можно также программировать отдельные функции непосредственно на языке ассемблера и затем обращаться к ним из программы на языке Си. Аналогично функции на языке Си можно вызывать из ассемблерных программ. Это позволяет программисту самостоятельно оптимизировать критические участки программы ЦОС. *Отладчик* программ, написанных на языке Си, обеспечивает отображение переменных и значений выражений, оценивание значений выражений, пошаговое выполнение инструкций, расстановку точек программных прерываний, отображение дерева вызова функций и др. *Библиотеки языка Си* содержат все стандартные функции и функции, применяемые при обработке сигналов. Например, функции цифровой фильтрации, быстрого преобразования Фурье, матричных операций, обработки прерываний.

Программатор ППЗУ транслирует код, предназначенный для выполнения процессором ЦОС, в один из нескольких форматов для программирования различных типов ППЗУ.

Аппаратные средства поддержки разработок базируются на использовании внутрисхемных эмуляторов. *Внутрисхемный эмулятор* позволяет выполнять отладку и оптимизацию приложения путем его исполнения на физической модели системы ЦОС. Такие эмуляторы состоят из специальных плат, построенных на основе соответствующего ЦПОС и программного обеспечения. Программное обеспечение загружается в ПЭВМ или рабочую станцию и управляет платой, которая представляет реконфигурируемую систему на основе выбранного ЦПОС. Такая система может выполнять загружаемое в нее приложение ЦОС. Внутри-

схемный эмулятор позволяет выполнять пошаговое выполнение программ с отображением содержимого регистров ЦПОС и памяти. Эмуляторы отличаются между собой максимальной скоростью работы процессора, способностью выполнять трассировку программ и отображать состояния выводов ЦПОС в реальном времени, способностью обрабатывать условные точки прерывания в реальном масштабе времени и др. Внутрисхемные эмуляторы могут быть также реализованы с использованием специальной системы контроля и отладки, встроенной в ЦПОС. Такая система основана на стандарте JTAG (Joint Test Action Group), который определяет принципы построения 4-х проводного последовательного интерфейса для тестирования СБИС. В настоящее время многие ЦПОС имеют порт JTAG, позволяющий получить доступ к внутренним ресурсам кристалла и тестировать систему на уровне процессора или уровне платы.

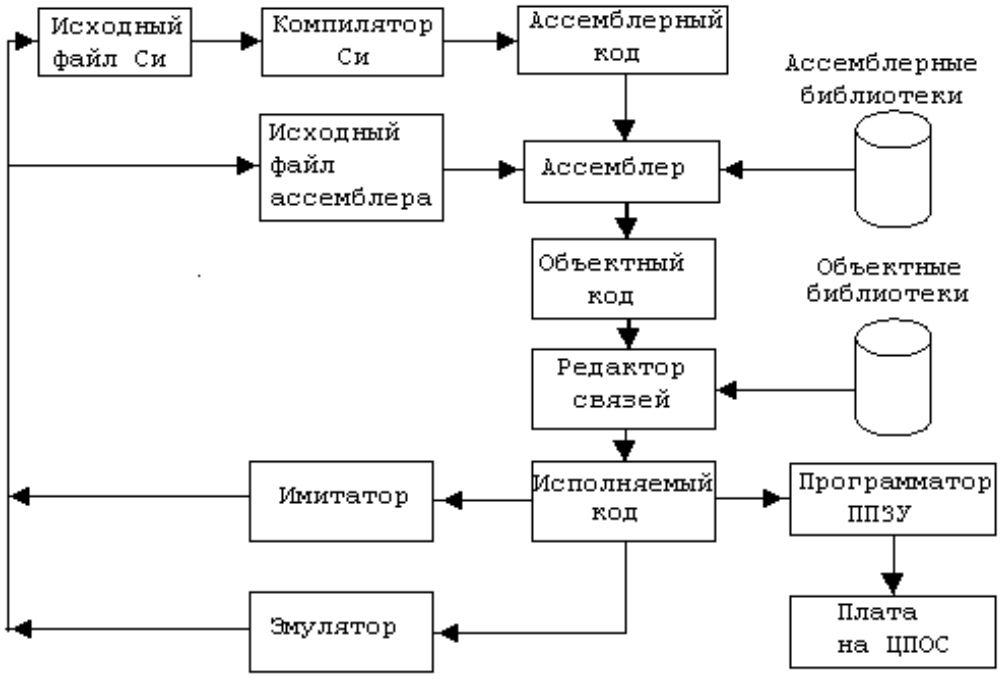


Рис.6.16. Общая схема разработки систем ЦОС

Общая схема разработки системы ЦОС с использованием рассмотренных инструментальных средств изображена на рис. 6.16. По сравнению с программированием цифровых вычислительных устройств общего применения программирование ЦПОС требует эффективной реализации алгоритмов, работающих в реальном масштабе времени. Традиционно программирование таких алгоритмов выполняется на языке ассемблера. При этом большое внимание уделяется оптимизации программ. Такой подход является достаточно трудоемким и дорогим. Процесс проектирования новых систем ЦОС становится проще, если имеется весь арсенал средств, указанных на рис.6.16. В частности, разработка систем ЦОС

значительно упрощается, если выполнять программирование на языке Си. Однако в ряде случаев это приводит к потере быстродействия программ. Поэтому на практике обычно используют комбинированную реализацию алгоритмов ЦОС: одна часть алгоритмов программируется на языке Си, а другая - на языке ассемблера. Программирование алгоритмов ЦОС на языке Си не имеет каких-либо существенных особенностей. Поэтому ниже, при описании ЦПОС с фиксированной и плавающей запятой, рассматривается программирование на языке ассемблера.

Глава 7
ЦПОС С ФИКСИРОВАННОЙ ЗАПЯТОЙ СЕМЕЙСТВА TMS320C2x

7.1 Назначение выводов процессора

Второе поколение процессоров серии TMS320 включает следующие основные СБИС: TMS320C25, TMS320C25-50, TMS320P25. Технические параметры СБИС приведены в табл.7.1. В дальнейшем будем рассматривать СБИС TMS320C25, являющуюся базовой. На рис.7.1 приведены обозначения выводов ЦПОС TMS320C2x.

Таблица 7.1 - Параметры ЦПОС TMS320C2x

Тип БИС	Частота, МГц	Цикл команды, нс	ОЗУ, слов	ПЗУ, слов	ПДП	Последоват. порт	Параллельн. порт	Таймер
TMS320C25	40	100	544	4К	Внеш.	1	16x16	1
TMS320C25-50	50	80	544	4К	Внеш.	1	16x16	1
TMS320P25	40	100	544		Внеш.	1	16x16	1

Системную шину адреса образуют 16 выводов $A0 - A15$, которые могут находиться в трех состояниях: единичном, нулевом и третьем состоянии, которое называют состоянием с высоким импедансом. Выводы шины адреса устанавливаются в высокоимпедансное состояние в режиме захвата шин (HOLD-режиме).

Выводы $D0 - D15$ образуют системную шину данных, которая является двунаправленной и устанавливается в высокоимпедансное состояние, когда нет ввода-вывода данных или когда действует сигнал сброса, а также в режиме захвата шин. Направление передачи сигналов по шине определяется сигналом $R/\overline{W}$.

$R/\overline{W}$ -сигнал чтения/записи. Он определяет направление передачи данных при взаимодействии процессора с внешними устройствами. Если на выводе $R/\overline{W}$ действует сигнал высокого уровня, то выполняется чтение, если низкого то - запись. В режиме захвата шин вывод находится в третьем состоянии.

$\overline{DS}, \overline{PS}, \overline{IS}$ - сигналы выбора пространства адресов: памяти данных, памяти программ, устройств ввода-вывода (соответственно). Обычно сигналы имеют высокий уровень, за исключением случаев обращения к внешней памяти или к устройствам ввода-вывода. В режиме захвата шин выводы переводятся в третье состояние.

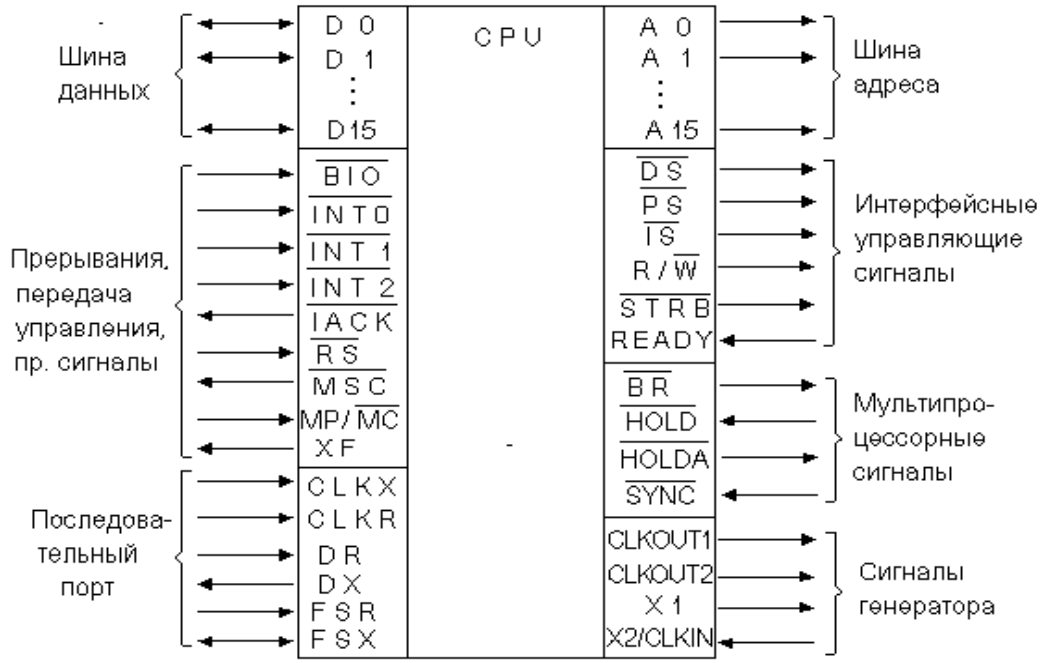


Рис. 7.1. Назначение выводов ЦПОС TMS320C25

READY - сигнал готовности ввода данных. Высокий уровень сигнала информирует ЦПОС о готовности внешнего устройства для обмена данными. Если сигнал *READY* будет иметь низкий уровень, то ЦПОС формирует один цикл ожидания и затем опять проверяет уровень сигнала *READY* . Сигнал *READY* используется также для информирования о готовности глобальной памяти к обмену (реакция на сигнал запроса шины BR).

$\overline{STRB}$ - стробирующий сигнал. Имеет низкий уровень во время чтения/записи данных. В режиме захвата вывод $\overline{STRB}$ устанавливается в высокоимпедансное состояние.

$\overline{INT0} - \overline{INT2}$ - входы внешних пользовательских прерываний, маскируемых с помощью регистра маски и управляемых битом режима прерываний.

$\overline{IACK}$ - выходной сигнал подтверждения прерывания. Сигнал должен анализироваться только при низком уровне на выходе CLKOUT1. Сигнал информирует о том, что прерывание принято и что произошла передача управления вектору прерывания установленному на шинах A0-A15.

$\overline{RS}$ - входной сигнал сброса. Сигнал вызывает прерывание процессора и передает управление команде, расположенной в нулевой ячейке памяти программ. Процессор начинает выполнение этой команды, когда $\overline{RS}$ будет иметь высокий уровень.

$\overline{BIO}$ - вход выбора ветви управления. Данный сигнал обрабатывается командой BIOZ. Если сигнал имеет низкий уровень, то процессор передает управление по адресу, указанному в команде BIOZ.

$\overline{MSC}$ - сигнал выполнения микросостояния. Сигнал является действующим только при низком уровне на выходе CLKOUT1, когда ЦПОС завершает операции с памятью, например, выборку очередной команды или чтение/запись данных. MSC может использоваться для формирования состояний ожидания сигнала READY при работе с медленно действующей памятью.

$MP/\overline{MC}$ сигнал выбора режима - микропроцессор/микрокомпьютер. Если сигнал имеет низкий уровень, то процессор находится в режиме микрокомпьютера и внутреннему ПЗУ отводится область младших 4К адресов. В режиме МР младшие адреса памяти программ отводятся под внешнюю память

XF - вывод внешнего флага. Сигнал XF является программно устанавливаемым. Используется для построения мультипроцессорных систем.

$\overline{HOLD}$ - вход запроса захвата шин для организации режима прямого доступа к памяти.

$\overline{HOLDA}$ - подтверждение запроса захвата шин. Этим выходным сигналом ЦПОС подтверждает запрос захвата шин. Процессор переводит шины данных, адреса и управления в высокоимпедансное состояние. Затем устройство, инициировавшее запрос, может управлять шинами ЦПОС.

$\overline{BR}$ - сигнал запроса шины. Сигнал формируется, когда требуется доступ к пространству внешней глобальной памяти данных в мультипроцессорных системах. Сигнал READY информирует ЦПОС, если глобальная память доступна.

$\overline{SYNC}$ - вход синхронизации, позволяющий выполнить синхронизацию двух или более процессоров TMS320C25.

$X1$ - вывод внутреннего генератора для подключения кварцевого резонатора. Если кварцевый резонатор не используется, то вывод должен оставаться свободным.

$X2/CLKIN$ - входной вывод внутреннего генератора для подключения кварцевого резонатора. Если резонатор не применяется, то вход используется для подачи тактовых импульсов.

CLKOUT 1 - основной выходной тактовый сигнал. Частота следования импульсов сигнала *CLKOUT* 1 равна CLKIN/4.

CLKOUT 2 - дополнительный выходной тактовый сигнал. Этот сигнал сдвинут относительно *CLKOUT* 1 на четверть фазы и имеет ту же частоту, что *CLKOUT* 1 .

Выводы ЦПОС CLKR, CLKX, FSR, FSX, DX, DR предназначены для работы с последовательным портом передачи и приема данных, встроенным в ЦПОС.

7.2 Структурная схема ЦПОС TMS320C25

Выполнение программ в ЦПОС сводится к циклическому повторению следующих действий: выборке команды из памяти; декодированию команды; исполнению команды.

В ЦПОС TMS320C2x с целью сокращения длительности командного цикла и повышения производительности используется *модифицированная гарвардская архитектура* и конвейерный режим работы, которые позволяют совмещать во времени указанные действия и обрабатывать одновременно несколько команд. Гарвардская архитектура предполагает раздельное хранение программ и данных, что позволяет совмещать декодирование и исполнение команды с вычислением адреса и выборкой следующей команды. В ЦПОС TMS320C2x возможен обмен данными между шинами памяти программ и памяти данных. Это позволяет, при необходимости, рассматривать память программ как память данных.

В архитектуре процессоров второго поколения используется трехэтапный конвейер. Процессор может одновременно обрабатывать три команды, находящиеся на разных этапах выполнения: предварительной выборки, декодирования, исполнения (рис. 7.2).

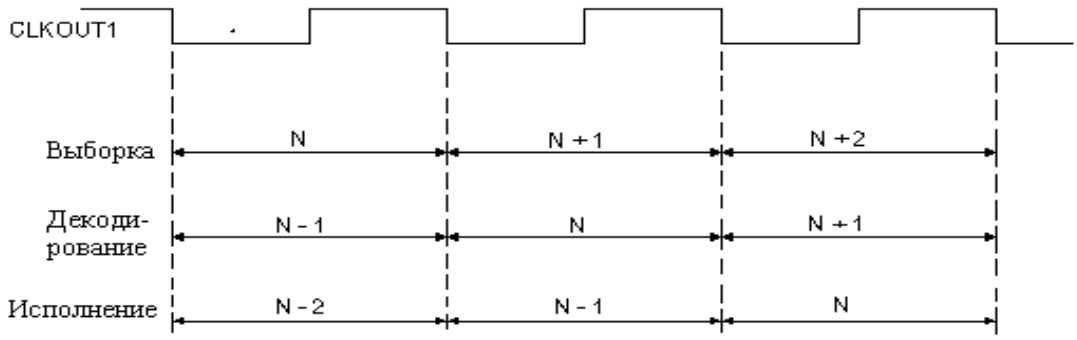


Рис. 7.2. Временные диаграммы конвейерного режима.

Например, когда производится выборка N-ой команды, предыдущая команда (N-1) находится на этапе декодирования, а команда N-2 - на эта-

пе исполнения. Конвейерный режим работы процессора остается для пользователя незаметным, за исключением случаев команд передачи управления, когда возникает необходимость перезагрузки конвейера.

Структурная схема ЦПОС TMS320C25 представлена на рис. 7.3. Процессор содержит: центральное арифметическо-логическое устройство (ЦАЛУ), дополнительное арифметическое устройство (ДАУ), счетчик команд (PC) и аппаратный стек на восемь 16-разрядных слов, ОЗУ емкостью 544 слова, состоящее из трех блоков B0, B1 и B2, ПЗУ емкостью 4К слов, 14 регистров различного назначения.

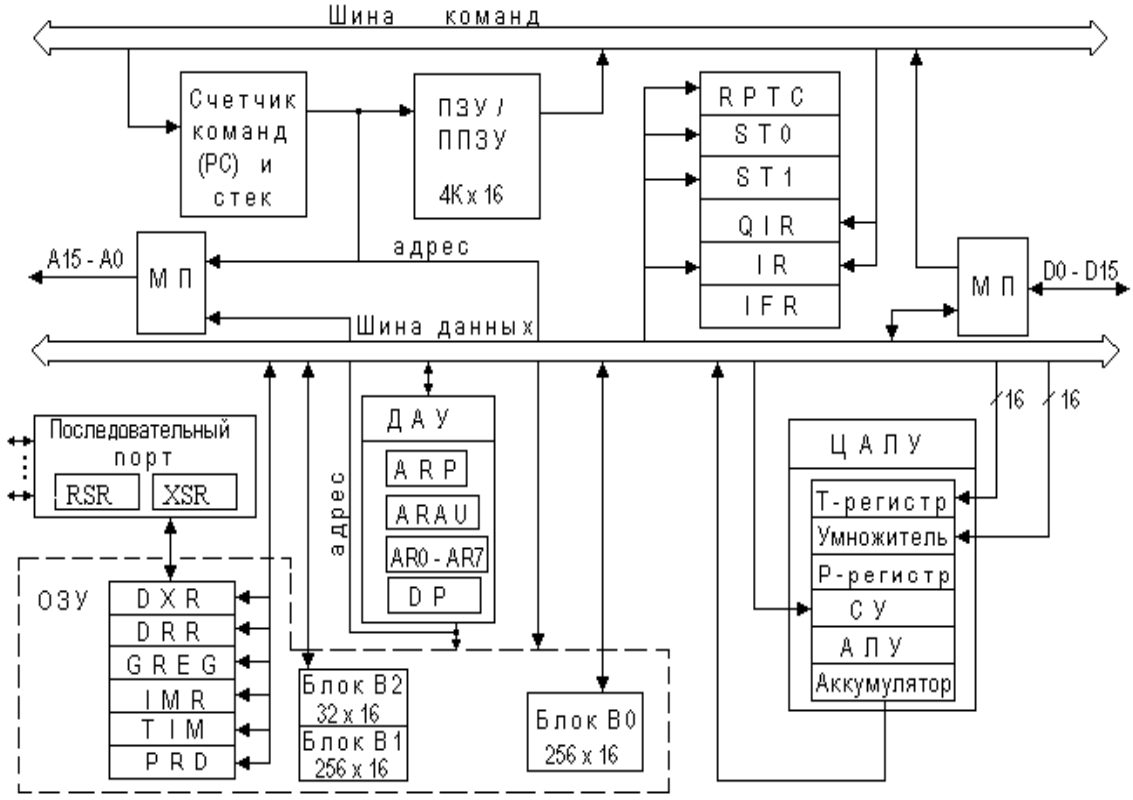


Рис.7.3. Структурная схема ЦПОС TMS320C25

Основные блоки процессора связаны с шиной данных и шиной команд. Шина команд обеспечивает передачу кодов команд и непосредственных операндов из памяти программ. Шина данных связывает ЦАЛУ и ДАУ с ОЗУ данных. Наличие двух шин позволяет считывать данные параллельно из памяти данных и памяти программ и передавать их в ЦАЛУ для выполнения MAC-операций за один цикл.

Благодаря наличию двух арифметических устройств возможна одновременная обработка данных в ЦАЛУ и вычисление адреса в ДАУ. Это обеспечивает выполнение большинства команд за один цикл.

При обращении к внешней памяти (адресное пространство 64К) программ или данных адресация или считывание данных выполняются с помощью одной внешней шины адреса (A15-A0) и одной внешней шины данных (D15-D0). Поэтому обращения к внешней памяти программ и

данных могут выполняться только последовательно. С целью повышения быстродействия процессора и возможности параллельного доступа к памяти программ и памяти данных в состав процессора включены внутрикристальное ОЗУ данных/программ и ПЗУ программ.

Адреса внутрикристального ОЗУ, соответствующие блоку B0, могут быть распределены либо на память данных, либо на память программ. Для этого соответственно применяются команды CNFD/CNFP. Программы, хранимые в блоке B0, выполняются с максимально возможной скоростью. Блоки B1 и B2 всегда используются как память данных. К внутрикристальному ОЗУ данных относят также шесть регистров, распределенных на адресное пространство памяти данных: регистры последовательного порта (DRR и DXR); таймер (TIM); регистр периода таймера (PRD), регистр маски прерываний (IMR), регистр распределения глобальной памяти (GREG).

Центральное арифметическо-логическое устройство (ЦАЛУ) процессора оперирует 16-разрядными словами, считываемыми из памяти данных или памяти программ. В состав ЦАЛУ входит умножитель, сдвигающее устройство (СУ), арифметическо- логическое устройство (АЛУ), аккумулятор. Умножитель выполняет умножение 16-разрядных слов и формирует 32-разрядный результат за один цикл. Умножитель содержит входной 16-разрядный регистр временного хранения (Т-регистр) одного из операндов и 32-разрядный регистр произведения (Р-регистр). Выходное слово Р-регистра может быть сдвинуто влево на 1 или 4 разряда и вправо на 6 разрядов с помощью сдвигающего устройства. Кроме этого, сдвигающее устройство имеет 16-разрядный вход, связанный с шиной данных и 32-разрядный выход, связанный с АЛУ. Это позволяет выполнять масштабирование данных, извлекать значения отдельных битов, предотвращать переполнения АЛУ процессора, выполнять арифметические и логические операции над 32-разрядными словами. Один из входов АЛУ связан с Р-регистром или сдвигающим устройством, другой - с аккумулятором.

Аккумулятор (ACC) предназначен для временного запоминания результатов операций, выполненных в АЛУ, и является 32-разрядным (старшее слово -биты 31-16 и младшее слово -биты 15-0).

Дополнительное арифметическое устройство (ДАУ) состоит из восьми 16-разрядных вспомогательных регистров (AR0-AR7), 3-разрядного указателя вспомогательных регистров (ARP), указателя страниц (DP), арифметического устройства вспомогательных регистров (ARAU).

ДАУ обычно используется для формирования адреса операнда памяти данных и функционирует параллельно с ЦАЛУ. В процессоре используется несколько способов адресации. Указатель страниц (DP) используется при прямой адресации, а вспомогательные регистры

(AR0-AR7), указатель вспомогательных регистров ARP, арифметическое устройство вспомогательных регистров (ARAU) - при косвенной регистровой адресации.

Восьмиуровневый аппаратный стек предназначен для запоминания содержимого счетчика команд (PC) во время прерываний и вызова подпрограмм. Внутренние прерывания генерируются 16-разрядным таймером (TIM) или последовательным портом. Внешние прерывания формируются сигналами INT0-INT2 и могут маскироваться.

Встроенный последовательный порт обеспечивает связь с другими устройствами по последовательному каналу. Два регистра RSR и XSR (сдвиговые регистры приема/передачи) последовательного порта могут функционировать либо в 8-битовом, либо в 16-битовом режиме.

Для построения многопроцессорных систем предусмотрена возможность использования глобальной памяти. Взаимодействие с этой памятью осуществляется посредством сигналов BR (запрос шины) и READY. Восьмиразрядный регистр GREG определяет какая часть внешней памяти должна рассматриваться как глобальная. Если текущая команда адресует глобальную память, то сигнал BR запрашивает доступ к шине данных глобальной памяти. Сигнал READY информирует о возможности доступа к запрашиваемой шине.

Управляющие операции поддерживаются в процессоре 16-разрядным таймером (TIM), счетчиком числа повторений (RPTC), регистрами состояния ST0, ST1 и очереди команд (QIR), регистрами команды (IR) и признаков прерываний (IFR).

7.3 Организация памяти

Процессоры семейства TMS320C2x обеспечивает работу с тремя адресными пространствами: памяти данных, памяти программ и ввода-вывода. Процессор может выполнять действия над содержимым внутренней (расположенной на кристалле) памяти, либо над содержимым внешней памяти. Ниже будем выделять следующие виды памяти [35]: внутреннюю и внешнюю память данных, внутреннюю и внешнюю память программ.

Внутренняя память данных представлена тремя блоками ОЗУ (B0, B1, B2) и шестью регистрами (DRR, DXR, TIM, PRD, IMR, GREG). Емкость ОЗУ 544 шестнадцатиразрядных слова. Из них 256 слов (блок B0) могут использоваться либо как память данных, либо как память программ. Оставшиеся 288 слов (блоки B1 и B2) всегда используются как память данных. Общая емкость памяти данных 64К шестнадцатиразрядных слова. Адреса, соответствующие внутренней памяти данных, не превышают 1024 (0400h) (рис.7.4). Адреса регистров, отображаемых на память данных приведены в табл. 7.2.

Внутренняя память программ представлена ПЗУ программ емкостью 4Кх16 и блоком В0, если он распределен на память программ командой CNFP (рис. 7.4) Внутренняя память программ позволяет процессору функционировать с максимальным быстродействием, при этом внешняя шина данных D15-D0 остается свободной для обращения к памяти данных. После поступления сигнала $\overline{RS}$ блок В0 отображается на память данных.

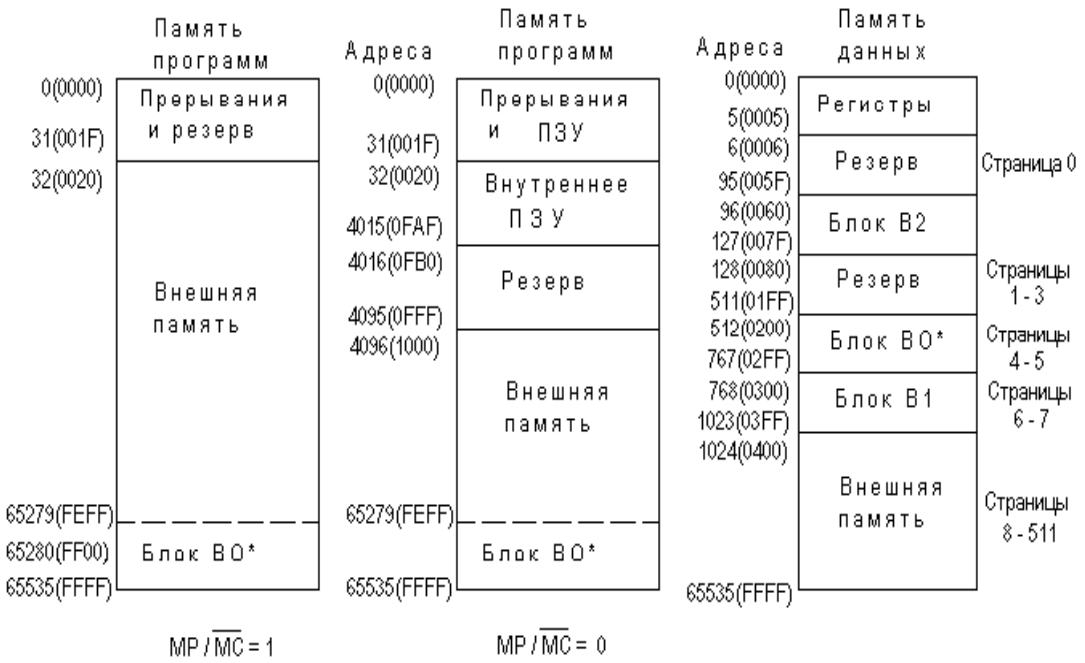


Рис.7.4. Карта распределения памяти

Таблица 7.2 - Адреса регистров

Регистр	Адрес	Назначение
DRR(16)	0	Регистр приема последовательного порта
DXR(16)	1	Регистр передачи последовательного порта
TIM(16)	2	Таймер
PRD(16)	3	Регистр периода таймера
IMR(6)	4	Регистр маски прерываний
GREG(8)	5	Регистр распределения глобальной памяти

Распределение первых 4К-слов памяти программ задается сигналом $MP / \overline{MC}$. Если $MP / \overline{MC}$ =1, то указанное адресное пространство отводится под внешнюю память программ. Если $MP / \overline{MC}$ =0, то адреса с 32 по 4015 отводятся под ПЗУ программ. Обращение к внешней памяти программ/данных происходит при активных сигналах $\overline{PS} / \overline{DS}$, $\overline{STRB}$. Во время обращения к внутренней памяти эти сигналы неактивны.

Действия, выполняемые по той или иной команде, существенно зависят от используемых видов памяти. При этом возможны четыре

комбинации видов памяти: PI/DI, PI/DE, PE/DI, PE/DE. Если учесть, что внутренняя память программ может быть представлена ПЗУ (PI) или блоком B0 (PR), то получим шесть различных комбинаций. Необходимо учитывать это обстоятельство при определении количества циклов, требуемых для выполнения команды.

7.4 Аппаратные средства адресации

К аппаратным средствам адресации, входящим в состав ДАУ, относят вспомогательные регистры AR0-AR7, указатель вспомогательного регистра ARP, буфер вспомогательного регистра ARB, арифметическое устройство вспомогательных регистров (ARAU), указатель страниц DP, мультиплексоры (MUX). Схема соединения перечисленных блоков приведена на рис.7.5.

Девятиразрядный регистр страниц DP используется при прямой адресации памяти данных. Для этого 7 младших разрядов слова команды (на рис. 7.5 - 7 м.р.к.) объединяются с содержимым DP.

Вспомогательные регистры AR0-AR7 используются при косвенной адресации памяти данных. В этом случае содержимое вспомогательных регистров рассматривается как шестнадцатиразрядный адрес памяти данных. Для выбора одного из вспомогательных регистров используется трехразрядный регистр указатель ARP, в который загружается значение от 0 до 7. Процессор имеет специальные команды для первоначальной загрузки регистров AR0-AR7 и ARP. Указатель ARP может быть загружен значением, взятым либо из памяти данных (через шину данных), либо из 3-х младших разрядов команды (через шину команд). Каждый раз, когда ARP загружается новым значением, старое значение сохраняется в буфере ARB (за исключением команды LST).

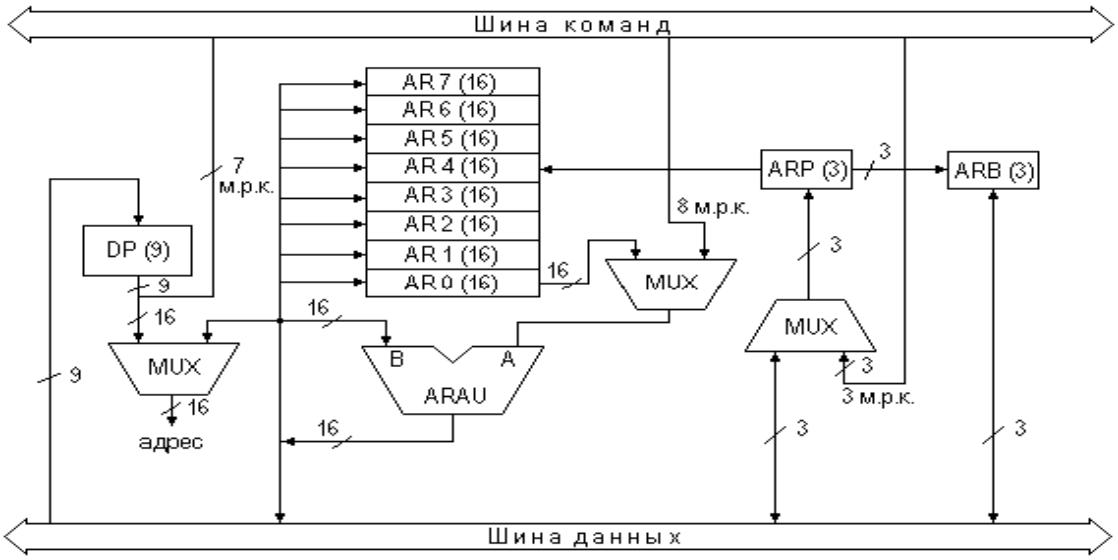


Рис.7.5. Структурная схема ДАУ

ARAU позволяет после первоначальной загрузки выполнять индексацию вспомогательного регистра, на который указывает ARP. Индексация выполняется путем увеличения (уменьшения) содержимого ARn на единицу, либо путем сложения (вычитания) содержимого AR0 с содержимым ARn. Индексация осуществляется параллельно с операциями, выполняемыми в ЦАЛУ.

Хотя ARAU разработано для поддержания манипуляций с адресами, оно может также использоваться как арифметическое устройство общего назначения. ARAU оперирует 16-разрядными аргументами без знака, а также обеспечивает реализацию команды передачи управления (BANZ), основанную на сравнении содержимого AR0 со вспомогательным регистром, на который указывает ARP. Это позволяет реализовывать с помощью ARAU циклические алгоритмы. В этом случае один из вспомогательных регистров используется в качестве счетчика циклов, а AR0 содержит его конечное значение.

7.5 Центральное арифметическо-логическое устройство

ЦАЛУ содержит 16-разрядный сдвигающий регистр, паралельный умножитель 16х16, 32-разрядное арифметическо-логическое устройство (ALU), 32-разрядный аккумулятор (ACC) и дополнительные сдвигающие устройства на выходах умножителя и аккумулятора. Функциональная схема ЦАЛУ приведена на рис. 7.6 [35].

Выполнение команды в ЦАЛУ обычно сводится к следующим действиям:

1) получению данных из памяти через шину данных;

2) обработке данных в масштабирующем сдвигающий регистре, умножителе, ALU;

3) сохранению результата в аккумуляторе.

Один из входов ALU соединен с выходом аккумулятора (ACC), а другой может быть подключен либо к выходу масштабирующего сдвигающего регистра, либо к выходу умножителя. Масштабирующий сдвигающий регистр связывает шину данных с ALU и обеспечивает сдвиг данных влево на 16 разрядов (SFL(0-16)). Младшие биты сдвигаемого слова заполняются нулями или битом знака (SX), если установлен бит режима знакорасширения (SXM) регистра состояния ST1.

ЦАЛУ содержит дополнительное сдвигающее устройство на выходе умножителя, которое позволяет выполнять сдвиг вправо на шесть разрядов (SFR(6)) с учетом бита знака SX, или влево на 1 или 4 разряда (SFL(1,4)). Кроме этого, содержимое аккумулятора может быть сдвинуто влево на 7 разрядов при выводе данных.

ALU обеспечивает выполнение большого набора арифметических и поразрядных логических операций за один командный цикл. Оно

оперирует 16-разрядными словами, извлекаемыми из памяти данных или непосредственно из команды.

ALU поддерживает операции с плавающей точкой. В систему команд процессора включена команда NORM, позволяющая нормализовать число с фиксированной точкой, содержащееся в аккумуляторе. Это достигается путем удаления лишних знаковых битов. Команда NORM реализуется с помощью операции сдвига влево. Команда LACT позволяет выполнить обратную операцию - денормализацию числа с плавающей точкой за счет сдвига мантиссы. В этом случае количество сдвигов задается четырьмя младшими битами Т-регистра. Операции сдвига выполняются в масштабирующем сдвигающем регистре. Содержимое Т-регистра определяет значение экспоненты.

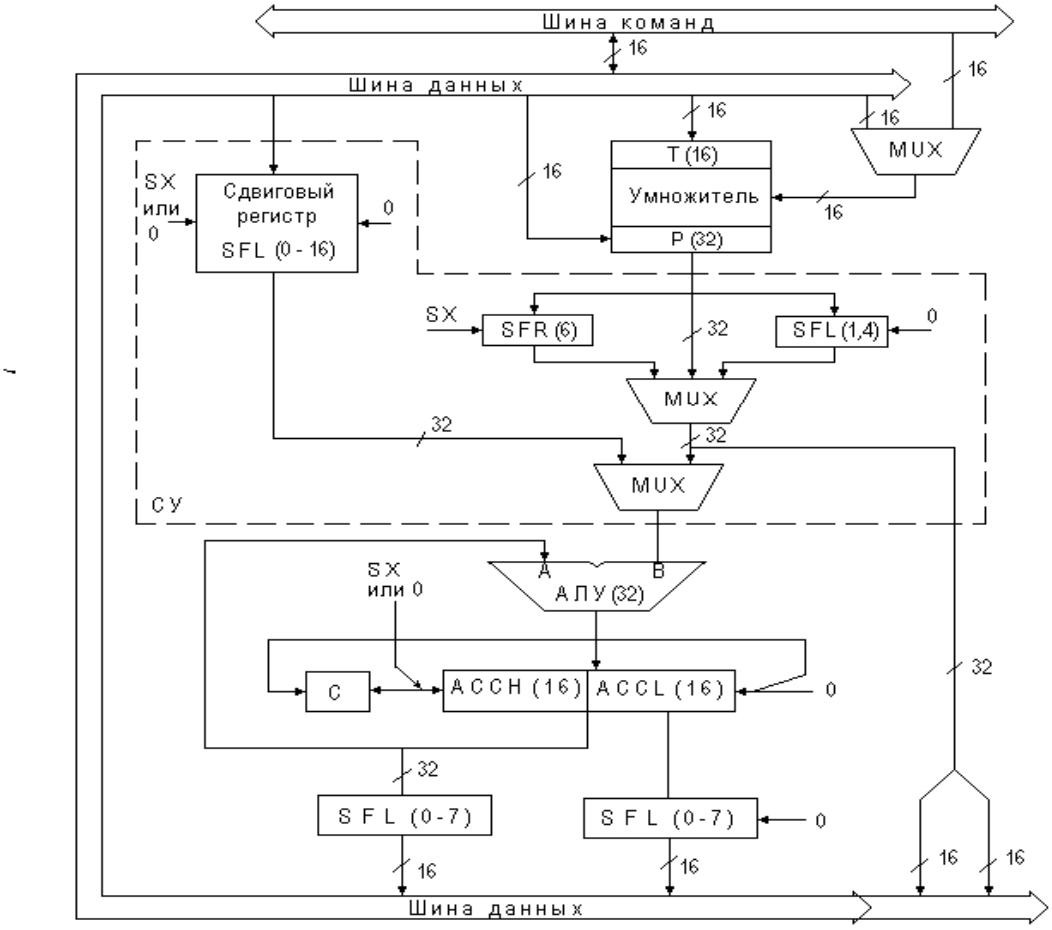


Рис.7.6. Функциональная схема ЦАЛУ

Программирование режима переполнения аккумулятора может выполняться командами SOVM и ROVM, устанавливающими или сбрасывающими (соответственно) бит режима переполнения OVM регистра состояния ST1. Если OVM=1 и происходит переполнение, то аккумулятор загружается либо максимальным положительным числом (7FFF FFFFh), либо минимальным отрицательным числом (8000 0000h) в

зависимости от направления переполнения. При OVM=0 результаты в аккумуляторе не модифицируются. Следует отметить, что логические операции не оказывают влияния на флаг переполнения.

ALU и аккумулятор позволяют выполнять широкий набор команд безусловного и условного переходов. Например, команду перехода при возникновении переполнения BV, команду перехода BZ, проверяющую на равенство нулю содержимое аккумулятора, команду безусловного перехода по адресу, содержащемуся в аккумуляторе (BACC) и др.

Аккумулятор процессора состоит из двух 16-разрядных регистров: ACCH (старшее слово аккумулятора) и ACCL (младшее слово аккумулятора). Сдвигающее устройство на выходе аккумулятора обеспечивает сдвиг на 7 разрядов влево. Сдвиг выполняется во время передачи данных в память и поэтому не оказывает влияние на содержимое аккумулятора.

Аккумулятор содержит бит переноса C, который позволяет более эффективно выполнять арифметические операции с повышенной точностью. Для этого могут применяться команды ADDC (сложить с учетом переноса) и SUBB (вычесть с учетом заема). Бит переноса может изменять свое значение под воздействием многих арифметических команд и команд сдвига. На значение бита переноса не оказывают влияние логические команды, команды загрузки аккумулятора, команды умножения MPY, MPYK, MPYU. Значение бита переноса C анализируется в командах условного перехода BC и BNC. Бит переноса C участвует в операциях сдвигов аккумулятора (команды SFL, SFR, ROL, ROR).

В ЦПОС применен аппаратный умножитель 16 x 16 бит, который позволяет получать 32-разрядный результат с учетом или без учета знака операндов команды умножения. Умножитель может находить произведение 2-х операндов, поступающих из памяти данных и памяти программ, либо только из памяти данных. Результат операции умножения помещается в P-регистр. При передаче содержимого P-регистра в АЛУ или память данных может выполняться операция сдвига произведения влево на 1 или 4 бита и вправо на 6 битов. Конкретное количество разрядов, на которое будет сдвинуто произведение, зависит от значения 2-х разрядного поля PM регистра состояния ST1. Сдвиг вправо на 6 битов позволяет реализовать 128 команд умножения с накоплением результата в аккумуляторе без опасности возникновения переполнения.

Для управления умножителем имеются команды: LT- загрузка T-регистра содержимым заданной ячейки памяти данных; MPY- перемножение операндов из T-регистра и памяти данных; MPYK вычисление произведения содержимого T-регистра и константы, заданной в команде. Кроме этого, в систему команд ЦПОС входят команды MAC, MACD, MPYA, MPYS, управляющие как умножителем, так и АЛУ с аккумулятором. Эти команды позволяют выполнить умножение с одновременным

накоплением результатов в аккумуляторе. При выполнении команд MAC и MACD операнды извлекаются из памяти данных и памяти программ за один цикл. Это позволяет командам MAC и MACD выполняться также в течение одного цикла, если они применяются совместно с командами RPT или RPTK.

7.6 Средства управления

Управление вычислительным процессом в ЦПОС осуществляется с помощью счетчика команд (PC), аппаратного стека, сигналов прерываний и сброса, регистров состояний, таймера, счетчика числа повторений [35].

7.6.1 Счетчик команд и стек

ЦПОС содержит 16-разрядный счетчик команд (PC) и восьмиуровневый аппаратный стек. Счетчик команд адресует память программ. Стек используется во время обработки прерываний и вызова подпрограмм.

Счетчик команд формирует адрес очередной команды и передает его на шину адреса команд. Команды из памяти программ загружаются в регистр команды (IR) (рис.7.7). В момент загрузки регистра IR счетчик команд содержит адрес следующей команды. Счетчик команд может также адресовать память данных. Это происходит во время исполнения команды BLKD, которая выполняет пересылку блоков данных в памяти данных.

Счетчик команд связан с шиной данных через мультиплексор (MUX) и может быть загружен содержимым аккумулятора. Это используется в командах передачи управления по адресу, находящемуся в аккумуляторе (BACC, CALA). В начале нового цикла выборки команды содержимое счетчика команд увеличивается на 1 или в него загружается адрес, по которому следует передать управление.

Если происходит вызов подпрограмм или обработка прерываний, то содержимое счетчика команд сохраняется в стеке. Для работы со стеком используются команды PUSH (запомнить в стеке) и POP (извлечь из стека). Когда содержимое счетчика команд запоминается в верхушке стека, старое значение верхушки продвигается на уровень ниже; значение, находящееся внизу стека, теряется. Старое значение верхушки стека может быть потеряно, если до извлечения его из стека команда PUSH будет выполнена 8 раз. При выталкивании значений из стека может произойти обратное переполнение стека, если применить команду POP более 7 раз. В этом случае на всех уровнях стека будет находиться одно и то же значение.

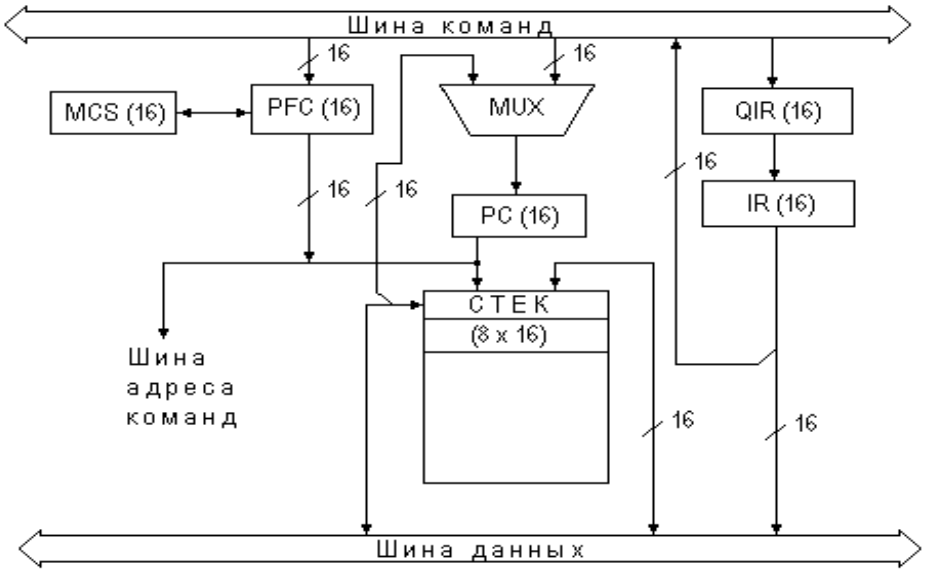


Рис.7.7. Счетчик команд и стек

ЦПОС содержит две дополнительные команды PUSHD и POPD, которые позволяют сохранять содержимое верхушки стека в памяти данных и извлекать его из памяти данных. Эти команды обеспечивают организацию стека произвольной глубины в памяти данных.

В ЦПОС TMS320C25 каждая команда выполняется в течение трех машинных циклов: предварительной выборки, декодирования и исполнения. Счетчик предварительной выборки (PFC) содержит адрес следующей команды, которая должна быть выбрана. Эта команда загружается в регистр команды (IR), однако если он будет занят исполняемой командой, то предварительно выбранная команда сохраняется в регистре очереди команд (QIR). Затем инкрементируется содержимое PFC и после выполнения текущей команды содержимое QIR загружается в IR. Таким образом, счетчик команд, хотя и содержит адрес следующей команды, но непосредственно в операциях выборки не применяется. В действительности он является указателем текущей команды программы. Содержимое PC инкрементируется после того, как завершается выполнение команды.

Благодаря наличию трехэтапного конвейера, в каждом машинном цикле обрабатывается три команды, находящиеся на разных этапах выполнения. Конвейер становится двухэтапным, если выполняемая программа считывается из внутреннего ОЗУ. При обращении к внутреннему ОЗУ выборка и декодирование команды выполняются в одном и том же машинном цикле.

В качестве примера рассмотрим типовую временную диаграмму внутренних операций процессора (рис. 7.8).

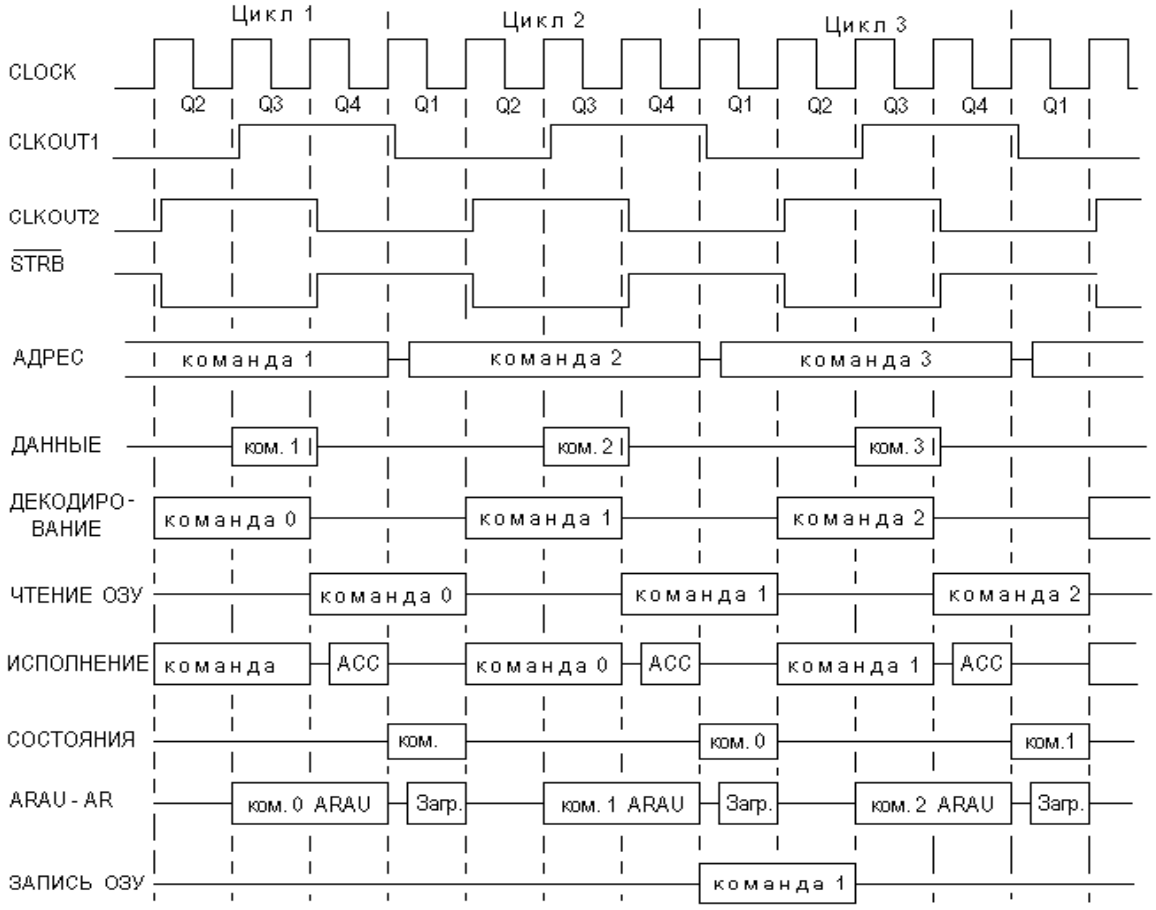


Рис. 7.8. Временные диаграммы основных операций

Машинный цикл процессора состоит из четырех тактов (фаз) сигнала CLOCK, обозначаемых Q1-Q4. Начало машинного цикла соответствует заднему фронту сигнала CLKOUT1. В первом машинном цикле происходит выборка команды 1 (фаза Q3). Во втором цикле в течение фаз Q2 и Q3 осуществляется декодирование команды, которое приводит к формированию управляющих сигналов АЛУ и выборке операндов команды в фазе Q4 цикла 2 и Q1 цикла 3. Параллельно с этим в цикле 2 в фазах Q3 и Q4 вычисляется новое содержимое вспомогательного регистра в ARAU (если используется косвенная адресация). Загрузка вспомогательного регистра и ARP выполняется в фазе Q1 третьего цикла. Во время действия фаз Q2 и Q3 цикла 3 производятся необходимые вычисления в ЦАЛУ. В фазе Q4 третьего цикла результаты вычислений помещаются в аккумулятор. Биты регистров состояния обновляются в фазе Q1 четвертого цикла. Биты, загружаемые в регистр состояния, определяют текущее состояние АЛУ и новое значение ARP, соответствующее следующей команде. Временные диаграммы, изображенные на рис.7.8, верны, если обращение к памяти не требует дополнительных циклов ожидания.

При работе с "медленной" памятью цикл выполнения команды удлиняется на требуемое число циклов ожидания.

7.6.2 Таймер, регистр периода, счетчик числа повторений

ЦПОС содержит 16-разрядный таймер (TIM) и регистр периода таймера (PRD, рис. 7.3), отображаемые на память данных. Таймер управляется сигналом CLKOUT1. При сбросе в таймер TIM и регистр периода PRD записывается максимальное число FFFFh. После того как сигнал сброса RS примет высокий уровень, содержимое таймера уменьшается на 1 при каждом поступлении импульса CLKOUT1. Начальное значение таймера определяется содержимым регистра периода. Как только значение в таймере станет равным нулю, формируется прерывание таймера TINT и он перезагружается значением, содержащимся в регистре периода PRD. Это происходит в течении первого машинного цикла после формирования прерывания TINT. Таким образом, прерывания будут происходить через интервалы времени, равные $[(PRD) + 1] \cdot T_{clkout1}$, где $T_{clkout1}$ - период сигнала CLKOUT1.

Обычно прерывание TINT используют для считывания выборок входного сигнала, обрабатываемого процессором. Таймер и регистр периода могут быть инициализированы в любом машинном цикле. Нулевое значение регистра периода недопустимо.

Если прерывание TINT не используется, то необходимо использовать маску или запретить прерывания командой DINT. В этом случае регистр периода PRD может использоваться как обычная ячейка памяти. Если требуется вновь использовать прерывание TINT, то регистр PRD и таймер следует предварительно инициализировать, а затем разрешить прерывание TINT.

Счетчик числа повторений RPTC содержит 8-разрядное двоичное число N, которое определяет количество повторений некоторой команды. Число повторений команды равно N + 1. С помощью команды RPT или RPTK счетчик числа повторений может быть загружен значением, находящимся в диапазоне от 0 до 255. Это позволяет повторять команду, находящуюся за командой RPT или RPTK, максимум 256 раз.

Счетчик числа повторений может использоваться командами: умножения с накоплением, пересылки блоков, ввода-вывода, чтения/записи таблиц. Те команды, которые требуют для своего выполнения нескольких циклов , при записи их после команды повторений (RPT или RPTK) выполняются за один цикл. Например, команда чтения таблицы TBLR обычно требует трёх и более циклов конвейера. При повторении с помощью RPTC она выполняется за один цикл.

7.6.3 Регистры состояний ST0 и ST1

Регистры состояний ST0 и ST1 (рис. 7.3) предназначены для сохранения различных условий и режимов работы процессора. Содержимое регистров состояний может быть запомнено в памяти данных. Это позволяет восстанавливать состояние процессора после обработки прерываний и вызова подпрограмм. Все биты состояний могут быть запомнены и считаны из регистров состояний ST0 и ST1 с помощью команд LST/LST1 (загрузить регистр состояния) и SST/SST1 (запомнить регистр состояния). Бит INTM (режим прерываний) не может быть установлен с помощью команды LST.

На рис. 7.9 показана структура регистров состояний. Рассмотрим назначение отдельных полей регистров ST0 и ST1.

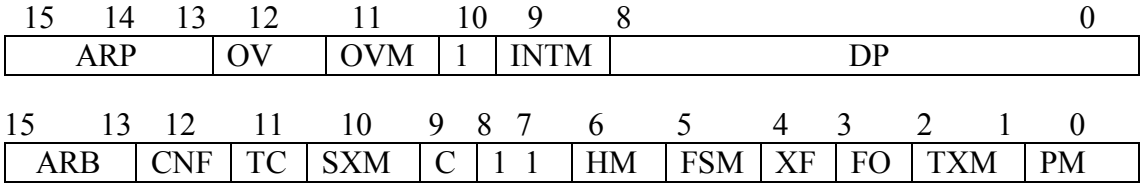


Рис.7.9. Структура регистров состояний

Поля ARP, ARB, DP копируют содержимое соответствующих регистров, показанных на рис. 7.5. Для указанных регистров нет специальных команд, сохраняющих их значения в памяти. После загрузки регистров ST0 и ST1 командами LST/LST1 регистры ARP, ARB, DP инициализируются значениями соответствующих полей регистров состояний.

Поля OV и OVM имеют отношение к переполнению АЛУ. Бит флага переполнения OV устанавливается в 1, если при выполнении операции в АЛУ произошло переполнение. Бит может быть установлен в 0 командами LST, BV, BVN или сигналом сброса. Бит режима переполнения OVM определяет способ аппаратной обработки переполнения. Если OVM=1, то в случае возникновения переполнения в аккумулятор загружается максимальное положительное или минимальное отрицательное число. Команды SOVM и ROVM позволяют соответственно сбросить и установить бит OVM.

Бит переноса C устанавливается в 1, если в результате выполнения суммирования в АЛУ происходит перенос, и устанавливается в 0, если при выполнении вычитания происходит заем. В любом случае (когда нет переноса или заема), после выполнения операции суммирования, бит C устанавливается в ноль, а после операции вычитания в - 1 (за исключением команд ADDH и SUBH). На бит переноса также воздействуют команды: SC, RC, LST1, команды сдвигов. Имеются команды передачи управления BC и BNC, реагирующие на значение бита переноса. При поступлении сигнала сброса бит переноса устанавливается в 1.

Бит режима знакорасширения SXM разрешает (SXM=1) или подавляет (SXM=0) дополнение числа с левой стороны до требуемого формата битом знака. Бит устанавливается и очищается командами SSXM и RSXM. При поступлении сигнала сброс бит устанавливается в 1.

Бит режима прерывания INTM разрешает или запрещает прерывания. Если INTM=0, то все прерывания, на которые не наложена маска, разрешены. В случае INTM=1 запрещены все маскируемые прерывания. Бит INTM устанавливается и очищается соответственно командами DINT и EINT. Сигналы RS и IACK устанавливают бит INTM в 1.

Бит флага контроль/управление TC меняет свое значение под воздействием команд BIT, BITT, CMPR, LST, NORM (см. ниже описание команд). Имеются две команды условного перехода BBZ и BBNZ, реагирующие на значение бита TC.

Бит управления конфигурацией внутрикристального ОЗУ CNF определяет размещение блока B0 . Если CNF=0, то B0 соответствует памяти данных, в противном случае - памяти программ. Бит модифицируется командами CNFD, CNFP, LST1. Сигнал RS устанавливает CNF=0.

Бит режима захвата HM определяет реакцию процессора на входной сигнал HOLD. Если HM=0, то после обработки сигнала HOLD процессор переводит шины в третье состояние, но может при этом выполнять команды из внутренней памяти программ. Когда HM=1 процессор приостанавливает внутренние операции.

Биты режима сдвига произведения PM управляют сдвиговым устройством при передаче данных из P-регистра в АЛУ или на шину данных. Если PM=00, то сдвиг не выполняется. Значения PM=01 и PM=10 соответствуют сдвигу влево на 1 и 4 разряда; значение PM=11 соответствует сдвигу вправо на 6 разрядов. Биты PM устанавливаются командами SPM и LST1. Сигнал RS обнуляет поле PM.

Бит состояния вывода XF используется в мультипроцессорных системах. Бит устанавливается и сбрасывается командами SXF и RXF. Сигнал RS устанавливает бит XF в единицу.

Биты FO, FSM и TXM управляют работой последовательного порта. Бит формата FO определяет длину регистров порта. Если FO=1, то регистры 16-разрядные , если FO=0, то 8-разрядные. Бит FO модифицируется командами FORT и LST1. Сигнал RS устанавливает бит FO в ноль. Бит режима кадровой синхронизации FSM определяет использование импульсов кадровой синхронизации. Если FSM=1, то последовательный порт синхронизируется сигналами на входах FSX/FSR. Сигнал RS устанавливает бит FSM в 1. Бит режима передачи TXM управляет назначением вывода FSX. Если TXM=1, то FSX - это выходной вывод, в противном случае - входной. Бит TXM устанавливается и сбрасывается командами STXM и RTXM. Сигнал RS устанавливает бит TXM в ноль.

7.6.4 Прерывания

Процессор имеет три источника внешних маскируемых прерываний (INT2-INT0) и четыре источника внутренних прерываний, генерируемых последовательным портом (RINT и XINT), таймером (TINT), программно с помощью команды TRAP. Сигнал RS также обрабатывается как прерывание. В табл. 7.3 приведены векторы прерываний и их приоритеты.

Когда возникает запрос прерывания, он запоминается в 6-разрядном регистре признаков прерываний (IFR). Отдельные разряды этого регистра управляются внешними прерываниями $\overline{INT}0 - \overline{INT}2$, а также внутренними прерываниями XINT, RINT и TINT. Каждое прерывание хранится в IFR пока не будет распознано, а затем автоматически очищается сигналом IACK или RS. RS прерывание не сохраняется в IFR. Внешние и внутренние прерывания можно маскировать с помощью регистра маски прерываний IMR (рис.7.10). Прерывания RS и TRAP не маскируются.

Таблица 7.3 - Векторы прерываний

Прерывание	Вектор	Приоритет
$\overline{RS}$	0	1 (наивысший)
$\overline{INT}0$	2	2
$\overline{INT}1$	4	3
$\overline{INT}2$	6	4
TINT	24	5
RINT	26	6
XINT	28	7 (низший)
TRAP	30	Нет

Если соответствующий разряд IMR содержит 0, то данное прерывание замаскировано. Регистр маски прерываний распределен на область памяти данных и доступен для чтения и записи по адресу 4. Бит INTM регистра состояний ST0 разрешает или запрещает маскируемые прерывания. Процессор имеет встроенный механизм, запрещающий прерывания при выполнении "длинных" команд, требующих нескольких машинных циклов. Прерывание обрабатывается только после того, как завершается выполнение команды.

15 — 6	5	4	3	2	1	0
Резерв	XINT	RINT	TINT	$\overline{INT}2$	$\overline{INT}1$	$\overline{INT}0$

Рис. 7.10. Регистр маски прерываний (IMR)

Прерывания запрещены внутри процесса, связанного с обработкой команды RPT или RPTK. В этом случае прерывание сохраняется в регистре IFR, пока содержимое счетчика числа повторений RPTC не

станет равным нулю. Прерывание не может быть обработано между командой EINT, разрешающей прерывания, и следующей по порядку командой.

Внешние прерывания, поступающие в процессор, синхронизируются ближайшей фазой Q2 сигнала CLKOUT и записываются в IFR в фазе Q1 следующего цикла. Если прерывания разрешены, то формируется сигнал подтверждения прерываний IACK, который устанавливает бит INTM в 1. Пока бит INTM не будет сброшен (команда EINT) прерывания будут запрещены. На рис. 7.11 приведена временная диаграмма обработки прерывания.

На диаграмме прерывание поступает в цикле выборки команды N. Конвейер процессора функционирует еще в течение двух машинных циклов, пока команда N не будет обработана. При этом из памяти выбираются команды N+1 и N+2, которые будут находиться соответственно в регистрах IR и QIR. В цикле адресации команды N+3 выборки и исполнения не происходит. В следующем цикле сохраняется содержимое счетчика команд PC (оно соответствует адресу команды N+1) в стеке, в PFC загружается адрес, соответствующий вектору прерываний и далее конвейер загружается командами подпрограммы обработки прерываний (I, I+1, I+2 и т.д.). В момент выборки первой команды обработки прерываний при низком уровне сигнала CLKOUT1 формируется сигнал IACK. Внешнее устройство, сформировавшее запрос прерывания, может установить вид прерывания путем "защелкивания" адреса, действующего на линиях A1-A4, при совпадении низкого уровня сигнала IACK и переднего фронта сигнала CLKOUT2. Этот момент соответствует выборке первого слова вектора прерывания.

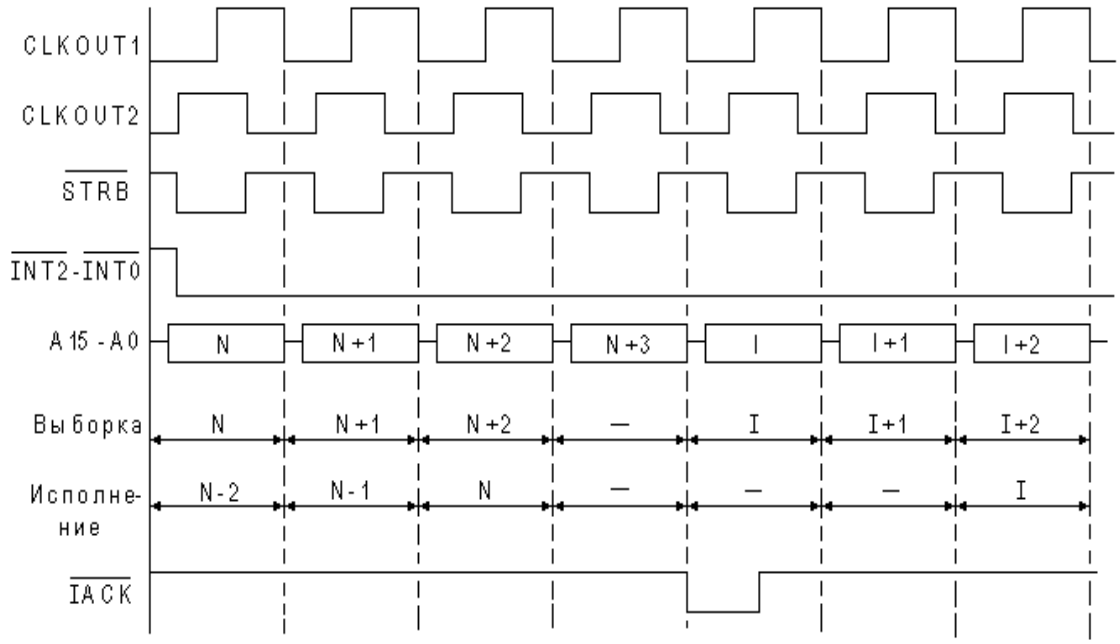


Рис.7.11. Временная диаграмма обработки прерываний

Прерывание от сигнала RS не может быть замаскировано. Длительность импульса сброса должна быть не менее длительности трех периодов сигнала CLKOUT. После положительного фронта RS процессор начинает выборку команд обработки сброса. Поскольку конвейер пустой, то выполнение команды из нулевой ячейки произойдет спустя два машинных цикла. Обычно в этой ячейке содержится команда передачи управления подпрограмме инициализации процессора.

7.6.5 Мультипроцессорные средства и прямой доступ к памяти

В мультипроцессорных системах решение задачи распределяется между несколькими параллельно работающими процессорами. За счет этого достигается существенное увеличение производительности системы, а также повышается надежность и живучесть системы.

На основе процессоров TMS320C2x можно проектировать мультипроцессорные системы различных конфигураций: с параллельными ЦПОС, использующими локальную память; с основным и вспомогательным процессорами ЦПОС, разделяющими глобальную память; с универсальным процессором, управляющим ЦПОС. Это возможно благодаря синхронизации процессоров посредством сигнала SYNC, интерфейсу с глобальной памятью, режиму захвата шин.

Синхронизация нескольких ЦПОС выполняется сигналом SYNC. Отрицательный фронт сигнала SYNC переводит каждый процессор мультипроцессорной системы в фазу Q1. При этом имеет место задержка, равная трем периодам сигнала CLKIN. Обычно сигнал SYNC формируется во время действия сигнала RS.

Работа с глобальной памятью данных выполняется с помощью сигналов BR (запрос шины) и READY. Глобальная память - это память разделяемая между несколькими процессорами. Для доступа к такой памяти используется арбитр. При этом пространство адресов каждого процессора разделяется на два раздела - локальный и глобальный.

Локальный раздел используется процессором для выполнения внутренних функций, а глобальный раздел для связи с другими процессорами. Регистр распределения глобальной памяти (GREG) определяет часть памяти данных процессора как глобальную внешнюю память. GREG это 8-разрядный регистр. Содержимое регистра GREG определяет объем глобальной памяти (табл. 7.4).

Когда происходит обращение к пространству адресов глобальной памяти, вместе с сигналом DS формируется сигнал запроса доступа к глобальной памяти BR. Внешний арбитр в ответ на BR вырабатывает сигнал READY для разрешения доступа.

Таблица 7.4 - Распределение глобальной памяти

Содержимо eGREG	Локальная память		Глобальная память	
	Диапазон	Объем	Диапазон	Объем
000000xx	0h-FFFFh	65536	----------	----------
10000000	0h-7FFFh	32768	8000-FFFFh	32768
11000000	0h-BFFFh	49152	C000h-FFFFh	16384
11100000	0h-DFFFh	57344	E000h-FFFFh	8192
11110000	0h-EFFFh	614440	F000h-FFFFh	4056
11111000	0h-F7FFh	63488	F800h-FFFFh	2048
11111100	0h-FBFFh	64512	FC00h-FFFFh	1024
11111110	0h-FDFFh	65024	FE00h-FFFFh	512
11111111	0h-FEFFh	65280	FF00h-FFFFh	256

Процессор TMS320C2x поддерживает режим прямого доступа к памяти данных и программ, пространству адресов ввода-вывода. После поступления сигнала HOLD от устройства, запрашивающего прямой доступ, ЦПОС переводит шину адреса A15-A0, шину данных D15-D0, управляющие выводы (PS, DS, IS, R/W, STRB) в высокоимпедансное состояние и формирует сигнал подтверждения доступа HOLDA. Обычно требуется не менее трех машинных циклов прежде, чем шины перейдут в третье состояние. В отличие от прерываний сигнал HOLD не запоминается, поэтому на выводе HOLD должен обеспечиваться низкий уровень до тех пор, пока внешнее устройство не завершит прямой доступ к памяти. Если при поступлении сигнала HOLD процессор выполняет некоторую команду, то он сначала завершает выполнение текущей команды, а затем переводит шины в третье состояние.

После подачи на вход HOLD сигнала высокого уровня продолжается выполнение программы с той же точки, где она была приостановлена. При этом синхронно с HOLD восстанавливается высокий уровень сигнала HOLDA. Требуется не менее двух машинных циклов для возобновления работы ЦПОС.

После перехода в режим прямого доступа процессоры TMS320C2x могут продолжать выполнение программы, находящейся во внутреннем ПЗУ или ОЗУ. Для этого необходимо предварительно установить бит HM регистра состояния ST1 в ноль. Если HM=1, то во время действия сигнала HOLD запрещены все прерывания. Если HM=0 , то прерывания обрабатываются обычным образом.

7.7 Режимы адресации и форматы команд

7.7.1 Режимы адресации

При выполнении любой команды процессор обращается к памяти программ или памяти данных. Память программ содержит команды, вы-полняемые процессором, а память данных - операнды, над которы-ми вы-полняются необходимые преобразования, задаваемые полем "код операции" (КОП) команды. Способ или метод определения в команде адреса операнда или адреса передачи управления называют режимом адресации. Процессоры TMS320C2x поддерживают три режима адре-сации: прямой, косвенный и непосредственный.

В режиме *прямой адресации* семь младших разрядов слова коман-ды объединяются с девятью разрядами регистра указателя страниц (DP) для получения полного 16-разрядного адреса памяти данных (dma рис.7.12). Нулевое значение седьмого бита слова команды иден-тифицирует режим прямой адресации.

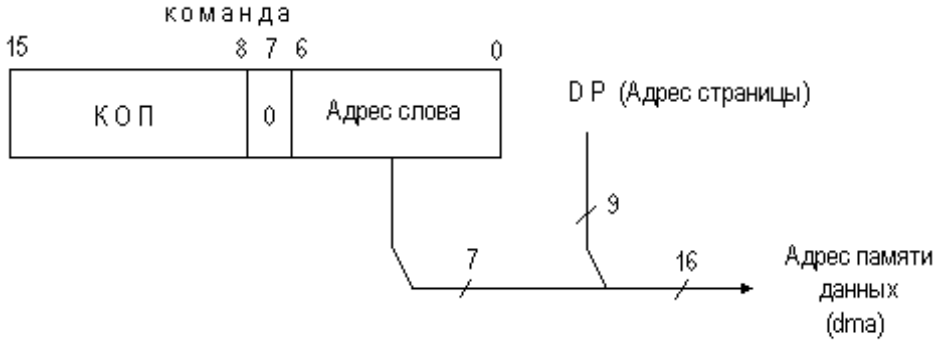


Рис.7.12. Прямая адресация

Указатель страниц позволяет выбрать одну из 512 страниц па-мяти данных. Каждая страница содержит 128 слов. Адрес слова на странице определяется младшими разрядами команды. Загрузка указате-ля страниц DP выполняется командами LDP (загрузить указатель страниц DP) и LDPK (загрузить указатель страниц непосредственно). Необходимо отметить, что указатель страниц не инициализируется при сбросе процессора.

В случае *косвенной адресации* адрес памяти данных определяется содержимым одного из вспомогательных регистров AR0-AR7. Выбор необ-ходимого регистра AR0-AR7 осуществляется с помощью трех-разрядного указателя вспомогательного регистра ARP, в который поме-щается число, определяемое тремя младшими битами (NARP) коман-ды (рис.7.13). При описании команд для обозначения текущего вспомо-гательного регистра, на который ссылается указатель ARP, используют запись - AR(ARP).

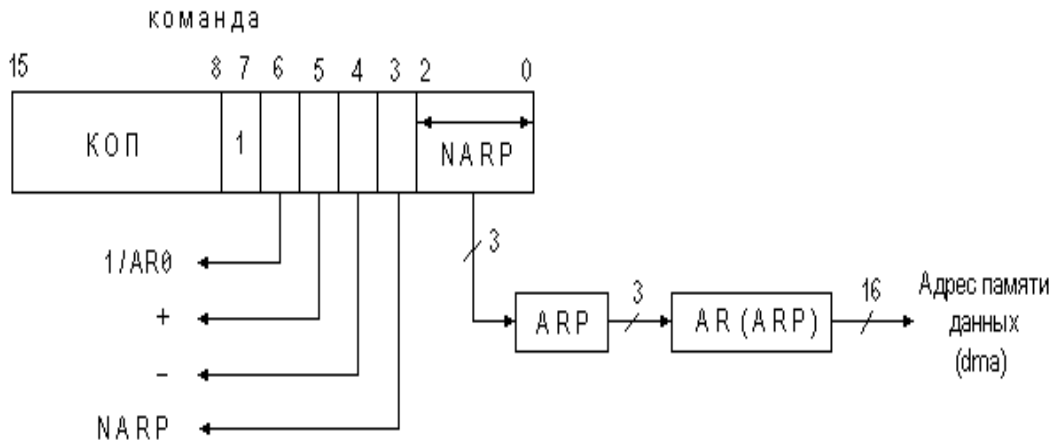


Рис.7.13. Косвенная адресация

Над содержимым регистров AR0-AR7 можно выполнять арифметические операции с помощью арифметического устройства вспомога-тельных регистров (ARAU). Указанное арифметическое устройство вы-полняет операции над содержимым AR0-AR7 параллельно с операциями, соответствующими выполнению текущей команды в ЦАЛУ. Наличие арифметического устройства вспомогательных регистров позволяет организовать семь различных вариантов косвенной адресации. При записи команд с косвенной адресацией на языке ассемблера используют специальные обозначения, показанные в табл.7.5.

Таблица 7.5 - Режимы косвенной адресации

Обозначение	Название режима адресации
* [,NARP]	Адресация без изменения (AR)
*- [,NARP]	Адресация с уменьшением (AR) на 1
*+ [,NARP]	Адресация с увеличением (AR) на 1
*0- [,NARP]	Адресация с увеличением (AR) на (AR0)
*0+ [,NARP]	Адресация с уменьшением (AR) на (AR0)
*BR0- [,NARP]	Адресация с уменьшением (AR) на (AR0) и обратным распространением переноса
*BR0+ [,NARP]	Адресация с увеличением (AR) на (AR0) и обратным распространением переноса

В табл.7.5 операнд NARP соответствует новому значению указателя ARP, которое установится после выполнения команды. Операнд, заключенный в квадратные скобки, является необязательным и может быть опущен. Запись (AR) обозначает содержимое вспомогательного регистра. Следует отметить, что уменьшение или увеличение значения вспо-могательного регистра происходит после его использования текущей командой.

Режимы адресации, основанные на обратном распространении переноса, называют *бит-инверсной адресацией* и применяют при упорядочении данных в программах быстрого преобразования Фурье .

Как показано на рис.7.13, единичное значение седьмого разряда команды определяет косвенную адресацию. Разряды 4, 5, 6 определяют одну из разновидностей косвенной адресации. Разряд 3 определяет использование поля NARP. Если значение этого разряда команды равно нулю, то поле NARP игнорируется и значение ARP остается неизменным.

Рис.7.14. Непосредственная адресация

При *непосредственной адресации* операнд содержится прямо в команде. Различают однословные (короткие) команды с непосредственным операндом (8 и 13-разрядным) (рис.7.14,а) и двухсловные (длинные) команды с непосредственным операндом. В длинных командах в первом слове содержится код операции (рис.7.14,б), а во втором непосредственный операнд.

7.7.2 Форма описания команд ЦПОС

Для записи команд будем использовать расширенные *формулы Бэкуса-Наура*. Формат ассемблерных команд процессора зависит от режима адресации:

<команда с прямой адресацией>::=
[<метка>] <мнемоника> <ас> [,<сдвиг>]

<команда с косвенной адресацией>::=
[<метка>] <мнемоника> <ка> [,<сдвиг> [,<NARP>]]

<команда с непосредственной адресацией>::=
[<метка>] <мнемоника> [<константа>]

Метка представляет собой определяемое пользователем имя. Значением метки является текущее значение счетчика команд. Метки

являются необязательными и используются в программах для передачи управления. *Мнемоника* представляет собой заранее определенное имя, которое идентифицирует код машинной операции (КОП). В качестве мнемоник используются сокращенные английские слова или аббревиатуры английских названий команд, передающих смысл основной функции команды. Например, ADDH (add to high accumulator) - сложить со старшим словом аккумулятора, LAR (load auxiliary register) - загрузить вспомогательный регистр.

После мнемоники следуют *операнды команд*. Значениями операндов <ac>, <сдвиг>, <NARP>, <константа> являются целые числа. При этом <ac> задает адрес слова на текущей странице памяти данных, 0 <= ac < 127. Необязательный операнд <NARP> определяет новое значение указателя вспомогательного регистра ARP, 0<=NARP <= 7.

Операнд <константа> - целое 8-разрядное или 16-разрядное число. Операнд <сдвиг> указывает на количество двоичных разрядов, на которое необходимо сдвинуть данное, участвующее в операции. Значение этого операнда зависит от выполняемой команды. Обычно значение поля <сдвиг> это число в пределах от 0 до 15.

Для команд с косвенной адресацией операнд <ка> определяется следующим образом:

<ка>::= *¦ *+¦ *0+¦ *BR0+¦ *BR0- .

При записи ассемблерных команд операнды отделяются друг от друга запятой. Ниже приведены примеры записи команд в соответствии с введенными определениями:

```
 ADD   DAT1,3        (7.1)
ADD     *,3          (7.2)
ADDK  5h             (7.3)
```

Здесь ADD, ADDK мнемоники команд. В команде (7.1) значение переменной DAT1 соответствует полю <ac> формата команды с прямой адресацией. Число 3 в командах (7.1) и (7.2) определяет сдвиг адресуемого операнда на три разряда влево. В команде (7.3) задана шестнадцатиричная константа 5.

7.8 Команды аккумулятора

7.8.1 Команды арифметических операций

Команды данной группы выполняют арифметические операции над содержимым аккумулятора (ACC) и содержимым заданной ячейки памяти данных , либо над 8- или 16-разрядной константой. При этом может учитываться бит переноса и выполняться сдвиг содержимого за-

данной ячейки памяти. Сдвиг указывается либо непосредственно в команде, либо определяется содержимым Т-регистра.

Ниже приведены описания арифметических команд ЦПОС. Для каждой команды указывается формат и выполняемая операция, которая условно описывается в виде некоторого выражения, а также в текстовой форме. Для лучшего понимания действий, выполняемых командой, приводятся примеры.

ADD (add to accumulator with shift) - сложить с аккумулятором со сдвигом.

Формат команды:

ADD <ac> [,<сдвиг>] ;

ADD <ка> [,<сдвиг>[,<NARP>]]

Операция: (ACC) + (dma)* $2^{сдвиг}$ --> ACC

Содержимое памяти данных, определяемое исполнительным адресом dma, сдвигается влево на число разрядов, соответствующее значению поля <сдвиг>, и затем суммируется с содержимым аккумулятора. Результат помещается в аккумулятор. В результате выполнения команды устанавливается бит переноса C и бит переполнения OV, если перед этим были установлены бит режима переполнения OVM и бит режима знакорасширения SXM.

Операция сдвига содержимого (dma) выполняется с использованием знакорасширения, если был установлен бит SXM. Диапазон значений поля <сдвиг> от 0 до 15. По умолчанию сдвиг равен нулю.

Пример 7.1

Здесь и далее заключение некоторого имени или адреса в круглые скобки обозначает содержимое (значение) области памяти, на которую ссылается имя или адрес. Например, запись вида (AR(ARP)) обозначает значение памяти, на которую ссылается текущий вспомогательный регистр, а запись вида (ACC) обозначает значение аккумулятора и т.д.

Во всех рассматриваемых ниже примерах левый столбец показывает состояние регистров и памяти процессора до применения команды, правый столбец - после применения.

(AR(ARP)) = 1025		(1025) = 4h
(1025) = 4h	ADD *,3	(ACC) = 28h
(ACC) = 8h		(C) = 0

Здесь содержимое ячейки памяти по адресу 1025 (т.е. 4h) сдвигается влево на 3 разряда (получается 20h) и складывается с содержимым аккумулятора (20h+8h=28h)

ADDC (add to accumulator with carry) - сложить с аккумулятором с учетом переноса.

Формат команды:

ADDC <ac>

ADDC <ка> [,<NARP>]

Операция: (ACC) + (dma) + (C) --> ACC

В результате выполнения команды устанавливаются биты OV и C, если предварительно был установлен бит OVM.

Команда используется при выполнении арифметических операций с повышенной разрядностью.

Пример 7.2

(AR(ARP)) = 1025	ADDC *,3	(1025) = 4h
(1025) = 4h		(ACC) = 29h
(C) = 1		(C) = 0

ADDH(add to high accumulator) - сложить со старшим словом аккумулятора.

Формат команды:

ADDH <ac>

ADDH <ка> [,<NARP>]

Операция: (ACC) + (dma) * 2^{16} --> ACC

Содержимое памяти складывается со старшим словом аккумулятора (разряды 31..16). Значение младшего слова не изменяется. Команда воздействует на биты C и OV. Бит C может быть только установлен данной командой, но не сброшен.

Пример 7.3

(AR(ARP)) = 1025	ADDH *	(1025) = 4h
(1025) = 4h		(ACC) = 40008h
(ACC) = 8h		(C) = 1
(C) = 1		

ADDK (add to accumulator short immediate) - короткое непосредственное сложение с аккумулятором.

Формат команды:

ADDK <константа>

Операция: (ACC) + 8-разрядная константа --> ACC

В результате выполнения команды устанавливаются биты C и OV.

Пример 7.4

(ACC) = A315h	ADDK 4h	(ACC) = A319h
		(C) = 0

ADDS (add to accumulator with sign-extension suppressed) сложение с аккумулятором с подавлением знакорасширения.

Формат команды:

ADDS <ac>

ADDS <ка> [,<NARP>]

Операция: (ACC) + (dma) --> ACC, где (dma) - 16-разрядное число без знака.

В результате выполнения команды устанавливаются биты C и OV. Аккумулятор рассматривается как число со знаком, содержимое по адресу dma - как число без знака.

Пример 7.5

(AP(ARP)) = 1025	ADDS *	(1025) = F115h
(1025) = F115h		(ACC) = F11Ah
(ACC) = 5h		(C) = 0

ADDT (add to accumulator with shift specified by T register) - сложить с аккумулятором со сдвигом, определяемым T-регистром.

Формат команды:

ADDT <ac>

ADDT <ка> [,<NARP>]

Операция: (ACC) + (dma)* $2^{Treg\,(3-0)}$ -->ACC

Содержимое памяти данных по адресу dma сдвигается влево на количество разрядов, определяемое четырьмя младшими битами T-регистра, и затем суммируется с содержимым аккумулятора. В остальном команда ADDT полностью аналогична команде ADD.

Пример 7.6

(AR(ARP)) = 1025	ADDT *	(1025) = 4h
(1025) = 4h		(T) = 94h
(ACC) = 8h		(ACC) = 48h
(T) = 94h		(C) = 0

ADLK (add to accumulator long immediate with shift) - длинное непосредственное сложение с аккумулятором со сдвигом.

Формат команды:

ADLK <константа> [,<сдвиг>]

Здесь <константа> - это 16-разрядное число в двоичном представлении. При выполнении команды константа сдвигается влево и затем суммируется с аккумулятором. Если бит SXM=1, то константа интерпретируется как число со знаком в диапазоне -32768..+37767. Если SXM=0, то значение константы лежат в диапазоне 0..65535. В результате выполнения команды устанавливается бит OV (если предварительно были включены режимы OVM и SXM) и бит переноса C. Длина команды два слова.

Пример 7.7

(ACC) = 13AFh	ADLK 7,8	(ACC) = 22FAh
		(C) = 0

В группу арифметических команд входят семь команд вычитания, шесть из которых аналогичны первым шести рассмотренным командам

сложения. Отличие заключается в том, что вместо операции сложения выполняется операция вычитания. Перечень этих команд и операции, выполняемые процессором приведены в табл. 7.6

Таблица 7.6 - Аккумуляторные команды вычитания

Мнемо-ника	Описание команды	Операция
SUB	Вычесть из аккумулятора со сдвигом	(ACC)-(dma) $2^{сдвиг}$ →ACC
SUBB	Вычесть из аккумулятора с заемом	(ACC)-(dma)-(C)→ACC
SUBH	Вычесть из старшего слова аккумулято-ра	(ACC)-(dma) 2^{16} →ACC
SUBK	Короткое непосредственное вычитание из аккумулятора	(ACC)-8-разр.конст. →ACC
SUBS	Вычитание из аккумулятора с подавле-нием знакорасширения	(ACC)-(dma)→ACC
SUBT	Вычесть из аккумулятора со сдвигом , определяемым Т-регистром	(ACC)+(dma) $2^{Treg(3-0)}$ →ACC

Ниже приведены примеры выполнения команд вычитания (табл.7.6).

Пример 7.8

(AR(ARP)) = 1041 SUB * (1041) = 21h
(1041) = 21h (ACC) = 24h
(ACC) = 45h (C) = 1

Пример 7.9

(AR(ARP)) = 1041 SUBB * (1041) = 6h
(1041) = 8h (ACC) = FFFFFFFFh
(ACC) = 8h (C) = 1
(C) = 0

Пример 7.10

(AR(ARP)) = 1041 SUBH * (1041) = 6h
(1041) = 6h (ACC) = 4B003h
(ACC) = AB003h (C) = 1

Пример 7.11

(ACC) = 48h SUBK 11h (ACC) = 37h
(C) = 1

Пример 7.12

(AR(ARP)) = 2000 SUBS * (2000) = A003h
(2000) = A003h (ACC) = 104h
(ACC) = A107h (C) = 1

Пример 7.13

(AR(ARP)) = 2000 SUBT * (2000) = 8h
(2000) = 8h (T) = A96h

(T) = A96h	(ACC) = EBA6h
(ACC) = EDA6h	(C) = 1

Седьмая команда вычитания не имеет аналогов среди команд сложения и называется условным вычитанием:

SUBC (conditional substract).

Формат команды:

SUBC <ac>

SUBC <ка> [,<NARP>]

Команда SUBC выполняет условное вычитание, которое используется при выполнении операции деления. 16-разрядное делимое помещается в младшее слово аккумулятора, а старшее слово аккумулятора обнуляется. Делителем является содержимое определенной ячейки памяти данных. Команда SUBC выполняется 16 раз при 16-разрядном делении. После выполнения команды SUBC частное от деления будет находиться в младшем слове аккумулятора, а остаток - в старшем слове. Команда SUBC позволяет получить правильные результаты деления, если делимое и делитель положительны. Бит SXM для этой команды связан с делителем. Если SXM=1, то делитель должен иметь нулевой старший бит. Если SXM=0, то любой 16-разрядный делитель обеспечит получение требуемых результатов. Делимое, находящееся в аккумуляторе, должно первоначально быть положительным и оставаться положительным в процессе сдвигов, обусловленных выполнением команды SUBC.

Если делимое содержит меньше 16 разрядов , то его можно предварительно сдвинуть влево. При этом количество повторений команды SUBC сокращается на число выполненных сдвигов. Необходимо отметить, что команда SUBC устанавливает бит OV . Однако эта возможность не управляется битом режима OVM. Поэтому не возникает отрицательных или положительных переполнений во время выполнения команды SUBC.

Пример 7.14

(AP(ARP)) = 1025	RPTK 15	(1025) = 6h
(1025) = 6h	SUBC *	(ACC) = 2000Eh
(ACC) = 56h		(C) = 1

В этом примере команда RPTK обеспечивает повторное выполнение команды SUBC 16 раз.

ABS (absolute value of accumulator) - абсолютное значение аккумулятора.

Формат команды:

ABS

Операция: ¦(ACC)¦ --> ACC

Если содержимое аккумулятора равно нулю или больше нуля, то значение аккумулятора остается неизменным. Если (ACC)< 0, то в аккумулятор помещается его двоичное дополнение.

Из приведенного правила имеется исключение. В том случае, если (ACC)= 8000 0000h и OVM=0, значение аккумулятора не изменяется. Если OVM=1, то абсолютное значение 8000 0000h есть 7FFF FFFF. В каждом из этих случаев устанавливается бит OV. Бит переноса после выполнения команды ABS всегда устанавливается в ноль.

Пример 7.15

(ACC) = 2341h	ABS	(ACC) = 2341h (C) = 0
(ACC) = AFFF FFFFh	ABS	(ACC) = 5000 0001h (C) = 0

NEG (negate accumulator) - отрицание аккумулятора.

Формат команды:

NEG

Операция: -(ACC) --> ACC

Значение аккумулятора замещается на его двоичное дополнение. Команда устанавливает бит OV, когда аккумулятор содержит значение 8000 0000h. Если OVM=1, то содержимое аккумулятора заменяется на 7FFF FFFFh. Если OVM=0, то в аккумулятор будет записано значение 8000 0000h.

Пример 7.16

(ACC) = AFFF FF34h	NEG	(ACC) = 5000 00CCh (C) = 0

NORM (normalize contens of accumulator) - нормализация содержимого аккумулятора.

Формат команды:

NORM <ка>

Команда используется для нормализации значения аккумулятора. Нормализация числа с фиксированной точкой требует выделения в нем мантиссы и порядка. Для определения порядка необходимо определить количество битов знакорасширения. Указанное действие внутри команды NORM выполняется с помощью операции (ACC(31))XOR (ACC(30)). Если разряды аккумулятора 30 и 31 равны, то оба они являются знаковыми. В этом случае содержимое аккумулятора сдвигается влево для того, чтобы удалить лишний знаковый бит.

Затем модифицируется значение вспомогательного регистра в соответствии с выбранным способом косвенной адресации для того, чтобы получить в нем значение порядка. Обычно оно инкрементируется.

Для полной нормализации 32-разрядного значения может потребоваться многократное выполнение команды NORM.

7.8.2 Команды логических операций

Данная группа команд реализует поразрядные операции булевой алгебры над содержимым аккумулятора и содержимым ячейки памяти данных, либо константой заданной непосредственно в команде.

AND (and with accumulator) - логическое И с аккумулятором.

Формат команды:

AND <ac>

AND <ка> [,<NARP>]

Операция: (ACC(15-0)) AND (dma) --> ACC(15-0)

Команда выполняет поразрядную логическую операцию И над содержимым младшего слова аккумулятора и адресуемым словом памяти данных. Старшее слово аккумулятора после выполнения команды всегда обнуляется. Бит SXM не оказывает влияния на получаемые результаты .

Пример 7.20

DAT1 = 16	AND DAT1	(DAT1) = FFh
(DP) = 4		(ACC) = 0000 0078h
(DAT1) = FFh		
(ACC) = 5432 1678h		

ANDK (AND immediate with accumulator with shift) - непосредственное логическое И с аккумулятором со сдвигом.

Формат команды:

ANDK <константа> [,<сдвиг>]

Операция: (ACC(30-0)) AND (константа * $2^{сдвиг}$) --> ACC(30-0)

0 --> ACC(31)

16-разрядная константа, непосредственно заданная в команде, сдвигается влево на количество разрядов, определенных полем команды <сдвиг>. Над константой и содержимым аккумулятора выполняется логическая операция И. Биты, расположенные вне пределов, определяемых длиной константы, считаются нулевыми и поэтому соответствующие разряды аккумулятора обнуляются после выполнения команды. Старший разряд аккумулятора обнуляется всегда. Длина команды два слова.

Пример 7.21

(ACC) = 5423 1678h ANDK FFFFh,8 (ACC) = 0032 1600h

OR (OR with accumulator) - логическое ИЛИ с аккумулятором.

Формат команды:

OR <ac>

OR <ка> [,NARP>]

Операция: (ACC(15-0)) OR (dma) --> ACC(15-0)

(ACC 31-16) --> ACC(31-16)

Данная команда позволяет осуществить поразрядную логическую операцию ИЛИ над содержимым младшего слова аккумулятора и адресуемым словом данных.

Пример 7.22

(AR(ARP)) = 1025 OR * (1025) = 0F00h

(1025) = 0F00h (ACC) = 12F01h

(ACC) = 12001h

ORK (OR immidiate with accumulator with shift) - непосредственное логическое ИЛИ с аккумулятором со сдвигом.

Формат команды:

ORK <константа> [,<сдвиг>]

Операция: (ACC(30-0)) OR (константа $2^{сдвиг}$) --> ACC(30-0)

(ACC(31)) --> ACC(31)

16-разрядная константа сдвигается влево на количество разрядов, определяемое полем <сдвиг>, и затем поразрядно выполняется логическая операции ИЛИ с содержимым аккумулятора. Биты, расположенные вне пределов длины константы, считаются нулевыми. Старший бит аккумулятора не меняет своего значения.

Пример 7.23

(ACC) = 5432 1678h ORK FFFFh,12 (ACC) = 5FFF F678h

XOR (exclusive OR with accumulator) - исключающее ИЛИ с аккумулятором.

Формат команды:

XOR <ac>

XOR <ка>[,<NARP>]

Операция: (ACC(15-0)) XOR (dma) --> (ACC(15-0))

(ACC(31-16)) --> ACC(31-16)

Выполняется поразрядная логическая операция исключающее ИЛИ над содержимым младшего слова аккумулятора и адресуемой памятью.

Пример 7.24

(AR(ARP)) = 65000 XOR * (65000) = 0F00h
(65000) = 0F0Fh (ACC) = 5432 1977h
(ACC) = 5432 1678h

XORK (XOR immediate with accumulator with shift) - непосредственное исключающее ИЛИ с аккумулятором со сдвигом.

Формат команды:

XORK <константа> [,<сдвиг>]

Операция: (ACC(30-0)) XOR (константа* $2^{сдвиг}$) --> ACC(30-0)
ACC 31 --> ACC 31

16-разрядная константа сдвигается влево на количество разрядов, определяемое полем <сдвиг>, и затем поразрядно выполняется логическая операция исключающее ИЛИ с содержимым аккумулятора. Биты, расположенные вне пределов длины константы, считаются нулевыми. Старший бит аккумулятора не меняет своего значения.

. **Пример** 7.25

(ACC) = 54321678h XORK FFFFh,12 (ACC) = 5BCD E678h

CMPL (complement accumulator) - дополнение аккумулятора.

Формат команды:

CMPL

Операция: $(\overline{ACC})$ --> ACC

Содержимое аккумулятора заменяется его поразрядным логическим отрицанием.

ROL (rotate accumulator left) - циклический сдвиг аккумулятора влево.

Формат команды:

ROL

Операция: ACC(31) --> C; (ACC(30-0)) --> ACC(31-1) ; (C) --> ACC(0)

Команда ROL обеспечивает выполнение циклического сдвига содержимого аккумулятора на один бит влево. Старший бит аккумулятора перемещается в бит переноса, бит переноса перемещается в младший бит аккумулятора.

Пример 7.26

(ACC) = 0A00 4321h ROL (ACC) = 1400 8663h
(C) = 1 (C) = 0

ROR (rotate accumulator right) – циклическикий сдвиг аккумулятора вправо.

Формат команды:

ROR

Операция: ACC(0) --> C; (ACC(31-1)) --> ACC(30-0) ; (C) --> ACC(31)

Команда обеспечивает выполнение циклического сдвига содержимого аккумулятора вправо на один бит.

Пример 7.27

(ACC) = 0A00 4321h	ROR	(ACC) = 8500 2190
(C) = 1		(C) = 1

SFL (shift accumulator left) - сдвиг аккумулятора влево.

Формат команды:

SFL

Операция: (ACC(30)) --> C; (ACC(30-0)) --> ACC(31-1) ; 0 --> ACC(0)

Команда обеспечивает выполнение сдвига аккумулятора на один разряд влево. Младший бит заполняется нулем.

Пример 7.28

(ACC) = 0A004321h	SFL	(ACC) = 1400 8642h
		(C) = 0

SFR (shift accumulator right) - сдвиг аккумулятора вправо.

Формат команды:

SFR

Операция: ACC(0) --> C; (ACC(31-1)) --> ACC(30-0)

Если SXM=0, то 0 --> ACC(31), иначе (ACC(31)) --> ACC(31).

Команда обеспечивает выполнение сдвига аккумулятора на один разряд вправо.

Пример 7.29

SXM = 0	SFR	(ACC) = 4500 2190h
(ACC) = 8A00 4321h		(C) = 1
SXM = 1	SFR	(ACC) = C500 2190h
(ACC) = 8A00 4321h		(C) = 1

7.8.3 Команды загрузки и запоминания содержимого аккумулятора

Команды данной группы позволяют поместить в аккумулятор значение, извлекаемое из памяти данных или заданное непосредственно в команде, а также сохранить значение аккумулятора в памяти данных.

LAC (load accumulator with shift) - загрузить аккумулятор со сдвигом.

Формат команды:

LAC <ac> [,<сдвиг>]

LAC <ка> [,<сдвиг> [,<NARP>]]

Операция: (dma) * $2^{сдвиг}$ --> ACC

Содержимое адресуемой памяти данных сдвигается влево и помещается в аккумулятор. При сдвиге младшие разряды заполняются нулями, а старшие заполняются знаковым битом, если SXM=1, и нулями, если SXM=0.

Пример 7.30

(AR(ARP)) = 1027	LAC *,4,5	(1027) = 3h
(1027) = 3h		(ACC) = 30h
(ACC) = 5432 1678h		(ARP) = 5

LACK (load accumulator immediate short) - короткая непосредственная загрузка аккумулятора.

Формат команды:

LACK <константа>

Операция: 8-разрядная константа --> ACC

8-разрядная константа помещается в аккумулятор. Старшие 24 разряда аккумулятора обнуляются.

Пример 7.31

(ACC) = 43h	LACK 25h	(ACC) = 25h

LACT (load accumulator with shift specified by T register) загрузить аккумулятор со сдвигом, определяемым Т регистром.

Формат команды:

LACT <ac>

LACT <ка> [,<NARP>]

Операция: (dma)* $2^{Treg(3-0)}$ --> ACC

Команда LACT обеспечивает загрузку аккумулятора значением, считанным из памяти данных и сдвинутым влево на количество разрядов, определяемое четырьмя младшими разрядами Т регистра. Если бит режима знакорасширения SXM=1, то значение, загружаемое в аккумулятор, дополняется слева до необходимого числа разрядов знаковым битом.

Пример 7.32

(AR(ARP)) = 1300	LACT *	(1300) = 276h
(1300) = 276h		(ACC) = 2760h
(ACC) = 8432 1AB5h		(T) = 2014
(T) = 2014h		

LALK (load accumulator long immediate with shift) - длинная непосредственная загрузка аккумулятора со сдвигом.

Формат команды:

LALK <константа> [,<сдвиг>]

Операция: (16-разрядная константа) * $2^{сдвиг}$ --> ACC

Сдвинутое влево значение 16-разрядной константы помещается в аккумулятор. Количество разрядов, на которое сдвигается константа, определяется полем <сдвиг>. Если бит SXM=1, то значение, загружаемое в аккумулятор, дополняется слева до необходимого числа разрядов знаковым битом. Необходимо отметить, что значение старшего бита аккумулятора может быть установлено в 1 только в режиме SXM=1. Тогда в аккумулятор может быть загружено значение в диапазоне -32768..32767. Если SXM=0, то загружаемое в аккумулятор значение находится в пределах от 0 до 65535. Длина команды-два слова.

Пример 7.33

SXM = 1 LALK F634h,4 (ACC) = FFFF 6340h
(ACC) = 5432 6781h

Пример 7.34

SXM = 0 LALK F634h,4 (ACC) = 000F 6340h
(ACC) = 5432 6781h

ZAC (zero accumulator) - обнулить аккумулятор.

Формат команды:

ZAC

Операция: 0 --> ACC

Данная команда соответствует команде LACK 0.

ZALH (zero low accumulator and load high accumulator) - обнулить младшее слово аккумулятора и загрузить старшее слово значением, извлекаемым из адресуемой памяти данных.

Пример 7.35

(AR(ARP)) = 4033 ZALH * (4033) = 2A01h
(4033) = 2A01h (ACC) = 2A01 0000h
(ACC) = FFFF FFFFh

ZALR (zero low accumulator, load high accumulator with rounding) - обнулить младшее слово аккумулятора и загрузить старшее с округлением.

Формат команды:

ZALR <ac>

ZALR <ка> [,<NARP>]

Операция: 8000h --> ACC(15-0); (dma) --> ACC(31-16)

Команда ZALR позволяет загрузить в аккумулятор значение из памяти данных с точностью до 1/2 младшего бита (старшего слова).

Пример 7.36

(AR(ARP)) = 4033 ZALR * (4033) = 2A01h
(4033) = 2A01h (ACC) = 2A01 8000h
(ACC) = AFAF AFAFh

ZALS (zero accumulator, load low accumulator with sign extension suppressed) - обнулить аккумулятор и загрузить младшее слово с подавлением знакорасширения.

Формат команды:

ZALS <ac>

ZALS <ка> [,<NARP>]

Операция: 0 --> ACC(31-16) ; (dma) --> ACC(15-0)

Значение из памяти данных загружается в младшее слово аккумулятора. Старшее слово аккумулятора обнуляется.

Пример 7.37

(AR(ARP)) = 4033 ZALS * (4033) = 2A01
(4033) = 2A01 (ACC) = 0000 2A01h
(ACC) = FAFA FAFAh

7.9 Команды вспомогательных регистров и указателя страниц

7.9.1 Команды арифметических операций и операций отношений

Эта группа команд выполняет арифметические операции над содержимым вспомогательных регистров и операции сравнения вспомогательных регистров.

ADRK(add to auxilary register short immediate) - короткое непосредственное сложение вспомогательного регистра.

Формат команды:

ADRK <константа>

Операция: (AP(ARP)) + 8-разрядная константа --> AR(ARP)

Содержимое текущего вспомогательного регистра складывается в ARAU с 8-разрядной константой. Результат помещается в текущий вспомогательный регистр.

Пример 7.38

(AR(ARP)) = 2341h ADRK 40h (AR(ARP)) = 2381h

SBRK (substract from auxilary register short immediate) - короткое непосредственное вычитание вспомогательного регистра.

Формат команды:

SBRK <константа>

Операция: (AR(ARP)) - 8-разрядная константа --> AR(ARP)
Команда соответствует ADRK.

Пример 7.39

(AR(ARP)) = 0h SBRK 03Fh (AR(ARP)) = FFC1h

MAR (modify auxilary register) - модификация вспомогательного регистра.

Формат команды:

MAR <ac>

MAR <ка> [,<NARP>]

В случае прямой адресации команда соответствует пустой операции. В случае косвенной адресации производится модификация вспомогательного регистра и указателя ARP, так как указано в команде. Никаких операций с памятью данных не выполняется. Если в команде задано значение поля NARP, то старое содержимое ARP копируется в поле ARB регистра состояния ST1.

Пример 7.40

(ARP) = 1h MAR *+,4 (AR1) = 38h
(AR1) = 37h (ARP) = 4

Следует отметить, что действия, выполняемые по команде MAR, можно также выполнить с помощью любой команды с косвенной адресацией.

CMPR (compare auxilary register with auxilary register AR0) - сравнение вспомогательного регистра со вспомогательным регистром AR0.

Формат команды:

CMPR <константа>

Операция: if ARn <константа> AR0 then 1 --> TC
else 0 --> TC

Команда выполняет сравнение содержимого текущего вспомогательного регистра с содержимым вспомогательного регистра AR0. Вид операции сравнения зависит от константы, которая может принимать значения в диапазоне 0..3. При этом имеет место следующее соответствие между значением константы и видом операции:

0 - (ARn = AR0);
1 - (ARn < AR0);
2 - (ARn > AR0);
3 - (ARn ≠ AR0).

Если результат сравнения будет иметь значение "истина", то устанавливается бит TC регистра состояния ST1.

Пример 7.41

(AR0) = 00F7h	CMPR 1	(AR0) = 00F7h
(AR(ARP)) = 00A1h		(AR(ARP)) = 00A1h
(TC) = 0h		(TC) = 1h

7.9.2 Команды загрузки и сохранения значений регистров

Команды данной группы позволяют загрузить заданные значения в регистры ARn, ARP, DP и производят обмен данными между вспомогательными регистрами и памятью данных.

LARP(load auxiliary register pointer) - загрузить указатель вспомогательного регистра.

Формат команды:

LARP <константа>

Операция: 3-х разрядная константа --> ARP, (ARP) --> ARB

В регистр ARP загружается значение 3-разрядной константы, определяющей, какой из вспомогательных регистров AR0..AR7 будет использоваться в дальнейших операциях. Прежнее содержимое (ARP) сохраняется в буфере указателя вспомогательного регистра ARB. Отметим, что содержимое ARP может быть изменено любой командой с косвенной адресацией.

LDP (load data memory page pointer) - загрузить указатель страниц памяти данных.

Формат команды:

LDP <ac>

LDP <ка> [,<NARP>]

Операция: (dma) --> DP

Содержимое девяти младших разрядов памяти данных (dma) помещается в регистр DP.

Пример 7.42

(AR(ARP)) = 1025	LDP *	(1025) = CAEDh
(1025) = CAEDh		(DP) = 0EDh
(DP) = 1AFh		

LDPK (load data memory page pointer immediate) - непосредственная загрузка указателя страниц памяти данных.

Формат команды:

LDPK <константа>

Операция: 9-ти разрядная константа --> DP

Значение константы, заданное в команде, помещается в регистр DP.

LAR (load auxiliary register) - загрузить вспомогательный регистр.

Формат команды:
LAR <ARn>, <ас>
LAR <ARn>, <ка> [,<NARP>]
Операция: (dma) --> ARn
Содержимое адресуемой памяти данных помещается в регистр ARn, указанный в команде.

Пример 7.43
(DP) = 8h LAR AR2,DAT (AR2) = A8h
DAT = 001Fh
dma = 041Fh
(dma) = A8h

В случае косвенной адресации модификация вспомогательного регистра, заданная полем <ка>, не выполняется, если команда LAR загружает значение в текущий регистр ARn, используемый для косвенной адресации.

Пример 7.44
(AR6) = 1025 LARP AR6 (1025) = 34h
(1025) = 34h LAR AR6,*+ (AR6) = 34h

LARK (load auxiliary register immediate short) - непосредсвенная короткая загрузка вспомогательного регистра.
Формат команды:
LARK <ARn>,<константа>
Операция: 8 разрядная константа --> ARn
Значение восьми разрядной константы, заданное в команде, помещается в регистр ARn. Старшие разряды регистра ARn заполняются нулями.

Пример 7.45
(AR2) = 0h LARK AR2,20h (AR2) = 20h

LRLK (load auxiliary register long immediate) - длинная непосредственная загрузка вспомогательного регистра.
Формат команды:
LRLK <ARn>,<константа>
Операция: 16 разрядная константа --> ARn
Значение шестнадцатиразрядной константы, заданное в команде, помещается в регистр ARn. Константа должна быть целым положительным числом.

Пример 7.46
(AR2) = 7A80h LRLK AR2,5073h (AR2) = 5073h
SAR (store auxiliary register) - запомнить вспомогательный регистр.

Формат команды:
SAR <ARn>,<ac>
SAR <ARn>,<ка>[,<NARP>]
Содержимое вспомогательного регистра запоминается в памяти данных по адресу dma.

Пример 7.47
(AR(ARP)) = 1031 SAR AR1,* (1031) = 15h
(AR1) = 15h
(1031) = FAh

Пример 7.48
(AR0) = 1031 LARP AR0 (1031) = 407h
(1031) = 0h SAR AR0,*0+ (AR0) = 80Eh

7.10 Команды T и P регистров, команды умножения

7.10.1 Команды загрузки, запоминания, сложения и вычитания

Команды данной группы предназначены для обмена данными между памятью и регистрами умножителя T и P. Некоторые из команд данной группы распространяют свое действие не только на T и P регистры, но и на аккумулятор. Это позволяет наряду с передачей данных выполнять операции сложения и вычитания аккумулятора с регистрами T и P.

LPH (load high P register) - загрузить старшее слово P регистра.
Формат команды:
LPH <ac>
LPH <ка>,[,<NARP>]
Операция: (dma) --> Preg (31-16)
Содержимое памяти данных по адресу dma загружается в старшее слово P-регистра. Младшее слово не изменяется.

Пример 7.49
(AR(ARP)) = 1025 LPH * (1025) = 3A64h
(1025) = 3A64h (P) = 3A64 5214h
(P) = 3041 5214h

LT (load T register) - загрузить T регистр.
Формат команды:
LT <ac>
LT <ка>[,<NARP>]
Операция: (dma) --> Treg
Содержимое памяти данных по адресу dma загружает в T регистр.

Пример 7.50

(AR(ARP)) = 1032	LT *	(1032) = 4AFFh
(1032) = 4AFFh		(T) = 4AFFh
(T) = 0h		

LTA (load T register and accumulate previor product) - загрузить T регистр и сложить предыдущее произведение.

Формат команды:

LTA <ac>

LTA <ка>[,<NARP>]

Операция: (dma) --> Treg, (ACC) + (Preg)* $2^{сдвиг}$ -->ACC

Содержимое памяти данных по адресу dma загружается в T регистр. Содержимое P регистра сдвигается в соответствии со значением битов PM регистра состояния ST1 и складывается с содержимым аккумулятора. Результат помещается в аккумулятор. Если PM=00, то сдвиг содержимого P регистра не производится. Если PM=01 или PM=10, то содержимое P регистра сдвигается влево соответственно на 1 или 4 разряда. При этом младшие биты заполняются нулями. В случае, если PM=11, то выполняется сдвиг вправо на 6 разрядов, старшие биты заполняются значением знакового бита.

Следует отметить, что сдвиг выполняется в процессе передачи содержимого P регистра в аккумулятор, поэтому операции сдвига не оказывают влияния на содержимое P регистра.

Сдвиг влево используется для выравнивания значения произведения в операциях с дробными числами. Сдвиг вправо на 6 разрядов позволяет выполнить 128 операций типа "умножение с накоплением" без опасности переполнения.

Команда LTA воздействует на биты C и OV. Последний устанавливается, если задан режим OVM.

Пример 7.51

(AR(ARP)) = 768	LTA *	(768) = 30h
(768) = 30h		(T) = 30h
(T) = 4h		(P) = AFh
(P) = AFh		(ACC) = B2h
(ACC) = 3h		(C) = 0
PM = 00		

LTD (load T register accumulate, previous product and move data) - загрузить T регистр, сложить предыдущее произведение, переслать данные.

Формат команды:

LTD <ac>

LTD <ка>[,<NARP>]

Операция: (dma) --> Treg, (ACC)+(Preg) * $2^{сдвиг}$ --> ACC,
(dma) --> dma + 1

Действия, выполняемые по данной команде, аналогичны действиям команды LTA. Дополнительно выполняется пересылка данных из ячейки с адресом dma в ячейку с адресом dma+1. Данная команда действительна только при работе с блоками памяти B0, B1, B2.

Пример 7.52

(AR(ARP)) = 768	LTD *	(768) = 30h
(768) = 30h		(769) = 30h
(769) = 0h		(T) = 30h
(T) = 4h		(P) = AFh
(P) = AFh		(ACC) = B2h
(ACC) = 3h		(C) = 0
PM = 00		

LTP (load T register and store P register in accumulator) - загрузить T регистр и запомнить содержимое P регистра в аккумуляторе.

Формат команды:

LTP <ac>

LTP <ка>[,<NARP>]

Операция: (dma) --> Treg ; (Preg)* $2^{сдвиг}$ --> ACC

Содержимое памяти (dma) загружается в T-регистр, содержимое P-регистра помещается в аккумулятор. При этом выполняется сдвиг P-регистра на число разрядов, определяемых полем PM .

LTS (load T register, subsract previous product)- загрузить T-регистр, вычесть предыдущее произведение.

Формат команды:

LTS <ac>

LTS <ка> [,<NARP>]

Операция: (dma) --> Treg, (ACC) - (Preg)* $2^{сдвиг}$ -->ACC

Команда аналогична команде LTA. Отличие заключается в том, что вместо сложения выполняется вычитание.

PAC (load accumulator with P register) - загрузить аккумулятор содержимым P регистра.

Формат команды:

PAC

Операция: (Preg)* $2^{сдвиг}$ --> ACC

Содержимое P регистра, сдвинутое на 1, 4 или 6 разрядов в соответствии со значением битов PM, помещается в аккумулятор.

SPM (set P register output shift mode) - установить режим сдвига выхода P регистра.

Формат команды:

SPM <константа>

Операция: 2-х разрядная константа --> PM

Константа, заданная в команде, копируется в поле PM регистра состояния ST1. Поле PM определяет сдвиг содержимого P регистра при передаче его в аккумулятор.

SPH (store high P register) - запомнить старшее слово P регистра.

Формат команды:

SPH <ac>

SPH <ка>[,<NARP>]

Операция: (Preg(31-16)) $2^{сдвиг}$ --> dma

Содержимое старшего слова P регистра, сдвинутое в соответствии с содержимым поля PM регистра ST1, помещается в ячейку памяти данных по адресу dma. При этом, если сдвиг выполняется вправо, то старшие биты заполняются знаковым битом. Если сдвиг выполняется влево, то младшие биты заполняются соответствующими значениями из младшего слова P регистра.

Пример 7.53

(P) = AF07 8354h	SPH *	(P) = AF07 8354h
(AR(ARP)) = 523		(523) = F078h
(523) = 3041h		
PM = 2		

SPL (store low P register) - запомнить младшее слово P регистра.

Формат команды:

SPL <ac>

SPL <ка>[,<NARP>]

Операция: (Preg(15-0)) $2^{сдвиг}$ --> dma

Сдвинутое на 1, 4 или 6 разрядов содержимое младшего слова P регистра помещается по адресу dma памяти данных. При сдвиге вправо (PM=11) старшие биты заполняются соответствующими значениями из старшего слова P регистра. Если сдвиг выполняется влево (PM=01 или 10), то младшие биты P регистра заполняются нулями.

APAC (add P register to accumulator) - сложить содержимое P регистра с аккумулятором.

Формат команды:

APAC

Операция: ACC + (Preg)* $2^{сдвиг}$ --> ACC

Содержимое P регистра сдвигается в соответствии со значением поля PM регистра ST1 и затем складывается с содержимым аккумулятора.

Команда воздействует на биты C и OV. Последний устанавливается, если задан режим OVM.

SPAC (subctract P register from accumulator) - вычесть содержимое P регистра из аккумулятора.

Формат команды:

SPAC

Операция: (ACC) - (Preg) * $2^{сдвиг}$ --> ACC

Аналогично предыдущей команде. Только выполняется вычитание.

7.10.2 Команды умножения

Команды данной группы обеспечивают выполнение операций умножения, умножения с накоплением результатов в аккумуляторе, умножения с накоплением и пересылкой данных. Обычно перемножаются аргументы, извлекаемые из памяти программ и данных, либо из T регистра и памяти данных.

MAC (multiply and accumulate) - умножить и накопить.

Формат команды:

MAC <pma>,<ac>

MAC <pma>,<ка>[,<NARP>]

Операция: (ACC) + (Preg) * $2^{сдвиг}$ --> ACC, (dma) --> Treg

(pma)* (dma) --> Preg

По команде MAC вычисляется произведение содержимого памяти данных по адресу dma и памяти программ по адресу pma. Команда также обеспечивает накопление предыдущего произведения в аккумуляторе. При этом сдвиг содержимого P-регистра выполняется в соответствии с содержимым поля PM регистра ST1. Адрес памяти программ pma, задаваемый в команде, должен быть в диапазоне 0..65535.

Команда MAC может использоваться совместно с командами цикла RPT, RPTK. В этом случае после каждого очередного выполнения MAC-операции модифицируется содержимое вспомогательного регистра в соответствии со значением поля <ка>, а также инкрементируется содержимое счетчика команд: (PFC) + 1 --> PFC. Это позволяет адресовать цепочку операндов из памяти программ.

В общем случае команда MAC занимает два слова и для ее выполнения требуется два командных цикла. Однако в случае применения команд RPT, RPTK она выполняется за один цикл. Поскольку при повторении адрес pma более не считывается из памяти, а вычисляется так, как

указано выше. В результате выполнения команды устанавливается бит OV, если задан режим OVM.

Пример 7.54

Ниже приведен фрагмент программы совместного использования команд MAC и RPTK.

```
SPM  3            ; установить сдвиг вправо на 6 разрядов
CNFP              ; B0 - блок памяти программ
LARP  3           ; текущий регистр AR3
LRLK  3,300h      ; 300h - начальный адрес B1
RPTK  255         ; повторять 256 раз
MAC  FF00h,*+     ; суммирование произведений
```

Здесь блок B0 сконфигурирован как память программ. В качестве памяти данных используется блок B1, который адресуется с помощью регистра AR3. В приведенном примере выполняется суммирование произведений аргументов, извлекаемых из блоков памяти B0 и B1. Загрузка поля PM регистра ST1 константой 3 (команда SPM 3) позволяет исключить переполнение аккумулятора, которое может возникнуть в результате операции сложения.

MACD (multiply and accumulate with data move) - умножение и накопление с передачей данных.

Формат команды:

MACD <pma>,<ac>

MACD <pma>,<ка>[,<NARP>]

Операция: (ACC) + (Preg) $2^{сдвиг}$ --> ACC

(pma) * (dma) --> Preg

(dma) --> dma + 1

Команда MACD аналогична команде MAC и дополнительно обеспечивает пересылку данных (dma) --> dma + 1. Пересылка данных будет выполняться, если в качестве памяти данных используются блоки B0, B1, B2. В противном случае команда MACD аналогична команде MAC.

MPY (multiply) - умножить.

Формат команды:

MPY <ac>

MPY <ка>[,<NARP>]

Операция: (Treg) * (dma) --> Preg

Содержимое T регистра перемножается с содержимым памяти данных dma. Результат помещается в P регистр.

MPYA (multiply and accumulate previous product) - умножить и накопить предыдущее произведение.

Формат команды:

MPYA <ac>

MPYA <ка>[,<NARP>]

Операция: (ACC) + (Preg) $2^{сдвиг}$ --> ACC

(Treg) * (dma) --> Preg

Предыдущее произведение сдвигается и суммируется с аккумулятором. Содержимое Т регистра перемножается с содержимым памяти данных dma. Результат помещается в Р регистр. Команда воздействует на биты С и OV.

Пример 7.55

(AR(ARP)) = 737	MPYA *	(737) = 6h
(737) = 6h		(T) = 7h
(T) = 7h		(P) = 2Ah
(P) = 34h		(ACC) = 8Bh
(ACC) = 57h		(C) = 0
(PM) = 0		

MPYK (multiply immediate) - непосредственное умножение.

Формат команды:

MPYK <константа>

Операция: (Treg) * (13-разрядная константа) --> Preg

Содержимое Т регистра перемножается с 13 разрядной константой, имеющей значение в диапазоне -4096..4096. Перед умножением константа выравнивается по правой границе, старшие биты заполняются битом знака.

Пример 7.56

(T) = 9h	MPYK -9	(T) = 9h
(P) = 3Ch		(P) = FFFF FFAFh

MPYS (multiply and substract previous product) - умножить и вычесть предыдущее произведение.

Формат команды:

MPYS <ac>

MPYS <ка>[,<NARP>]

Операция: (ACC) - (Preg) * $2^{сдвиг}$ --> ACC

(Treg) * (dma) --> Preg

Команда аналогична команде MPYA. Отличие заключается в том, что по команде MPYS выполняется вычитание, тогда как в команде MPYA - сложение.

MPYU (multiply unsigned) - беззнаковое умножение.

Формат команды:

MPYU <ac>

MPYU <ка>[,<NARP>]

Операция: Usgn(Treg) * Usgn(dma) --> Preg

Команда выполняет умножение беззнакового содержимого T регистра и памяти данных dma. Напомним, что умножитель перемножает 17-битные аргументы. В случае команды MPYU старшие биты каждого из операндов, поступающих на вход умножителя, считаются нулевыми. С командой MPYU не следует использовать режим PM=3, поскольку при этом происходит знакорасширение. Команда MPYU может использоваться для вычисления произведения 32-разрядных аргументов.

SQRA (square and accumulate previous product) - возвести в квадрат и накопить предыдущее произведение.

Формат команды:

SQRA <ac>

SQRA <ка>[,<NARP>]

Операция: ACC + (Preg) * $2^{сдвиг}$ --> ACC

(dma) x (dma) --> Preg

Содержимое P регистра сдвигается в соответствии с содержимым поля PM регистра ST1 и добавляется к аккумулятору. Затем значение (dma) загружается в T регистр возводится в квадрат и помещается в P регистр. Команда воздействует на биты C и OV.

SQRS (square and substract previous product) - возвести в квадрат и вычесть предыдущее произведение.

Формат команды:

SQRS <ac>

SQRS <ка>[,<NARP>]

Операция: (ACC) - (Preg) * $2^{сдвиг}$ --> ACC

(dma) x (dma) --> Preg

Команда SQRS аналогична команде SQRA. Однако в отличие от последней команда SQRS выполняет вычитание.

7.11 Команды переходов и вызова подпрограмм

В программах линейной структуры выполняется последовательная выборка команд из смежных ячеек памяти программ. В программах с разветвляющейся и циклической структурой необходимо выполнять не следующую по порядку команду, а команду, находящуюся в другой ячейке памяти программ. Для этого в счетчик команд загружается адрес новой ячейки памяти программ, называемый адресом перехода. Команды, реализующие указанную процедуру, называют командами передачи управления и подразделяют на команды переходов и вызова подпрограмм. В системе команд ЦПОС TMS320C2x предусмотрен обшир-

ный перечень команд передачи управления. Большинство команд данной группы занимают в памяти два слова.

B (branch unconditionally) - безусловный переход.
Формат команды:
B <pma>[,<ка>[,<NARP>]]
Операция: pma --> PC
Текущий вспомогательный регистр ARn и указатель ARB модифицируются так, как указано в команде. Управление передается команде, находящейся по адресу pma, 0 < pma < 65535. Команда занимает в памяти два слова.

BACC (branch on adress specified by accumulator) - переход по адресу, заданному аккумулятором.
Формат команды:
BACC
Операция: (ACC(15-0)) --> PC
Управление программой передается команде, адрес которой определяется младшим словом аккумулятора. Команда занимает в памяти одно слово.

BANZ (branch on auxiliary register not zero) - переход по не нулевому значению вспомогательного регистра.
Формат команды:
BANZ <pma>,[<ка>[,<NARP>]]
Операция: if AR(ARP) <> 0 then pma --> PC
else (PC) + 2 --> PC
Если содержимое текущего вспомогательного регистра не равно нулю, то управление передается команде, находящейся по адресу pma. В противном случае выполняется следующая команда. Следует отметить, что по умолчанию, если не указано поле <ка>, содержимое текущего вспомогательного регистра уменьшается на единицу, т.е. выполняется инструкция *-. Это сде-лано для совместимости с процессором TMS320C1x. Команду BANZ можно использовать для программирования циклических алгоритмов. В этом случае вспомогательный регистр выступает в качестве счетчика циклов.

Пример 7.57

(AR(ARP)) = 1h	BANZ 40h,*-	(AR(ARP)) = 0
(PC) = 58h		(PC) = 40h

Приведенные ниже команды условного перехода имеют общий формат: мнемоника <pma>[<ка>,[,<NARP>]] и выполняют передачу

управления команде, находящейся по адресу pma, если выполняется некоторое условие. В противном случае управление передается следующей команде. Затем модифицируется содержимое вспомогательного регистра и указателя ARP, если заданы поля <ка> и <NARP>. Поэтому ниже приведем упрощенное описание остальных команд условного перехода. В табл.7.7 указаны мнемоники команд и проверяемые условия, выполнение которых приводит к загрузке значения поля pma в счетчик команд (pma --> PC).

Как видно из табл.7.7, команды условного перехода позволяют организовать передачу управления, основываясь на анализе содержимого аккумулятора (>= 0, > 0, <= 0, < 0, <> 0, = 0), сигнала BIO и состояний битов: тестирования/управления TC, переноса C, переполнения OV. Все команды выполняются сходным образом. Некоторую особенность имеют команды BNV и BV, которые воздействуют на бит OV. Команда BNV устанавливает бит OV в нулевое состояние, если условие не реализуется. Команда BV, наоборот, очищает бит OV, если условие реализуется.

Таблица 7.7 - Команды условного перехода

Мнемоника	Описание команды	Условие
BBNZ	Переход, если бит TC<>0	(TC)=1
BBZ	Переход, если бит TC=0	(TC)=0
BC	Переход, если бит C=1	(C)=1
BNC	Переход, если бит C=0	(C)=0
BGEZ	Переход, если аккумулятор >=0	(ACC)>=0
BGZ	Переход, если аккумулятор >0	(ACC)>0
BLEZ	Переход, если аккумулятор <=0	(ACC)<=0
BLZ	Переход, если аккумулятор <0	(ACC)<0
BNZ	Переход, если аккумулятор <>0	(ACC)<>0
BZ	Переход , если аккумулятор =0	(ACC)=0
BNV	Переход, если бит OV=0	(OV)=0
BV	Переход, если бит OV=1	(OV)=1
BIOZ	Переход, если $\overline{B\,\overline{IO}} = 0$	$(\overline{B\,\overline{IO}}) = 0$

CALA (call subroutine indirect) - косвенный вызов подпрограммы.

Формат команды:

CALA

Операция: (PC) + 2 --> TOS; pma --> PC

Счетчик команд увеличивается на два и загружается в верхушку стека. Управление передается команде, находящейся по адресу pma.

RET (return from subroutine) - возврат из подпрограммы.

Формат команды:

RET

Операция: (TOS) --> PC

Содержимое верхушки стека копируется в счетчик команд. Из стека выталкивается одно значение. Команда используется для возврата из подпрограмм.

7.12 Команды ввода-вывода и пересылок данных

В данную группу команд входят команды пересылки блоков данных как внутри памяти данных, так и между памятью программ и памятью данных. Кроме этого, в данную группу команд включены команды работы с последовательным и паралельным портами ввода-вывода, а также команды, воздействующие на отдельные выводы ЦПОС.

7.12.1 Команды пересылки данных

BLKD (block move from data memory to data memory) - переслать блок из памяти данных в память данных.

Формат команды:

BLKD <dma1><ac>

BLKD <dma1><ка>[,<NARP>]

Операция: (dma1, содержащийся в PC) --> dma2

Команда выполняет пересылку блока данных между источником данных и приемником данных. Начальный адрес источника данных определяется полем dma1 и лежит в диапазоне 0..65535. Начальный адрес блока приема данных задается полями <ac>, либо <ка>, которые определяют адрес dma2. При прямой адресации адрес приемника данных не может модифицироваться автоматически, если команда BLKD входит в циклический оператор, например RPTK. В этом случае получателем будет одна и та же ячейка памяти данных.

Для правильной передачи данных команды RPT и RPTK должны использоваться с командой BLKD, заданной в формате с косвенной адре-сацией. Количество пересылаемых слов будет на единицу больше, чем начальное число, содержащееся в счетчике циклов RPTC. На момент окончания цикла (RPTC)=0, а AR(ARP) будет содержать адрес, следующий после конечного адреса блока приема .

Команда BLKD не может быть использована для передачи данных из внутренних регистров. Во время циклического выполнения команды BLKD с помощью операторов RPT или RPTK запрещены прерывания. После выполнения команды BLKD счетчик команд будет содержать адрес следующей команды.

Пример 7.58

Пусть (AR(ARP)) = 1025. Тогда фрагмент программы

```
RPTK  9
  BLKD A200h,*+
```

обеспечит пересылку блока из 10 значений, начиная с адреса A200h и заканчивая адресом A209h. Пересылка будет выполнена по адресам 1025..1034.

BLKP (blok move from program memory to data memory) - переслать блок из памяти программ в память данных.

Формат команды:

BLKP <pma>,<ac>

BLKP <pma>,<ка>[,<NARP>]

Операция: (pma) --> dma

Данная команда аналогична команде BLKD. Отличие заключается в том, что здесь источником данных является память программ.

Команды BLKD и BLKP занимают в памяти два слова.

TBLR (table read) - чтение таблицы.

Формат команды:

TBLR <ac>

TBLR <ка>[,<NARP>]

Операция: (pma, из ACC(15-0)) --> dma

Команда TBLR обеспечивает передачу данных из памяти программ в память данных. Команда аналогична команде BLKP с небольшим отличием: начальный адрес источника данных определяется младшим словом аккумулятора.

TBLW (table write) - запись таблицы.

Формат команды:

TBLW <ac>

TBLW <ка>[,<NARP>]

Операция: (dma) --> pma, из ACC(15-0)

Команда TBLW обеспечивает передачу данных из памяти данных в память программ. Общая схема выполнения команды аналогична схемам команд BLKD, BLKP, TBLR. Отличие заключается в том, что адрес источника данных определяется с помощью полей <ac> или <ка>, а начальный адрес приемника данных соответствует содержимому младшего слова акку-мулятора.

Команды TBLR и TBLW занимают в памяти одно слово.

DMOV (data move in data memory) - пересылка данных в памяти данных.

Формат команды:

DMOV <ac>
DMOV <ка>[,<NARP>]
Операция: (dma) --> dma + 1

Содержимое памяти данных по адресу dma копируется по следующему адресу dma + 1. Команда DMOV применима только при работе с блоками памяти B0, B1, B2. Команда полезна для организации задержек на шаг дискретизации при программировании цифровых фильтров.

7.12.2 Команды ввода-вывода

IN (input from port) - ввод из порта.
Формат команды:
IN <ac>,<PA>
IN <ка>,<PA>[,<NARP>]
Операция: PA --> (A3-A0), 0 --> (A15-A4)
(D15-D0) --> dma

Команда IN копирует 16 разрядное значение, устанавливаемое на шине данных D15-D0 внешним устройством, в память данных по адресу dma. Значение поля <PA> лежит в диапазоне 0..15 и представляет адрес порта.

Пример 7.59

```
(AR(ARP)) = 721        IN  *+,PA15      (721) = 31h
(721) = 0h
(A15-A0) = Fh
(D15-D0) = 31h
```

OUT (output data to port) - вывод данных в порт.
Формат команды:
OUT <ac>,<PA>
OUT <ка>,<PA>[,<NARP>]
Операция: PA --> (A3-A0), 0 --> (A15-A4)
(dma) --> D15-D0

Команда устанавливает на шине адреса адрес порта PA и выводит на шину данных D15-D0 содержимое ячейки памяти данных dma.

RFSM (reset serial port frame synchronization mode) - сброс режима кадровой синхронизации последовательного порта.
Формат команды:
RFSM
Операция: 0 --> FSM

Команда RFSM устанавливает бит FSM регистра состояния ST1 в ноль. В этом режиме не требуются внешние импульсы FSR для инициирования операции приема каждого принимаемого слова; достаточно только одного импульса для инициализации "непрерывного режима".

RTXM (reset serial port transmit mode) - сброс режима передачи последовательного порта.

Формат команды:

RTXM

Операция: 0 --> TXM

Команда RTXM устанавливает бит TXM регистра состояния ST1 в ноль. В этом случае передатчик запускается, когда приходит импульс на вход FXM.

RXF (rent external flag) - сброс внешнего флага.

Формат команды:

RXF

Операция: 0 --> XF

Команда RXF устанавливает вывод XF и бит XF регистра состояний в ноль.

SFSM (set serial port frame synchronization mode) - установить режим кадровой синхронизации последовательного порта.

Формат команды:

SFSM

Операция: 1 --> FSM

Команда устанавливает бит состояния в единицу. В этом режиме для приема данных требуется подача внешних FSR импульсов. Если TXM = 0, то требуется подача импульсов на вывод FSX. Если же TXM = 1, то импульсы FSX генерируются каждый раз, когда загружается сдвиговый регистр XSR передатчика.

STXM (set serial port transmit mode) - установить режим передачи последовательного порта.

Формат команды:

STXM

Операция: 1 --> TXM

Команда устанавливает бит TXM в единицу. В этом случае вывод FXM работает как выход, на котором появляется импульс каждый раз, когда загружается регистр DXR. Передача инициируется отрицательным фронтом этого импульса.

SXF (set external flag) - установить внешний флаг.

Формат команды:

SXF

Операция: 1 --> XF

Команда переводит вывод XF и бит XF в единичное состояние.

FORT (format serial port registers) - формат регистров последовательного порта.

Формат команды:

FORT <константа>

Операция: 1-битная константа --> FO

Бит состояния FO загружается 1-битной константой. Бит FO используется для управления форматом сдвиговых регистров приемника и передатчика последовательного порта. Если FO = 0 регистры передатчика и приемника являются 16 разрядными. Если FO = 1, то регистры являются 8 разрядными. Бит FO сбрасывается в ноль при подаче сигнала сброса.

7.13 Команды управления

В данную группу команд входят команды управления стеком, количеством повторений циклов, прерываниями, команда загрузки и запоминания регистров состояний, команды управления отдельными битами: C, OVM, HM, TC.

7.13.1 Команды управления стеком

POP (pop top of stack to low accumulator) - вытолкнуть верхушку стека в младшее слово аккумулятора.

Формат команды:

POP

Операция: (TOS) --> ACC(15-0)

Содержимое верхушки стека (TOS) пересылается (выталкивается) в младшее слово аккумулятора. На освободившееся место последовательно пересылаются элементы с нижних уровней стека . Поэтому после семи пересылок все элементы стека будут иметь одно и тоже значение, соответствующее последнему элементу стека. Данная ситуация называется антипереполнением. Процессор не имеет средств, обнаруживающих антипереполнение стека.

POPD (pop stack to data memory) - вытолкнуть стек в память данных.

Формат команды:

POPD <ac>

POPD <ка>[,<NARP>]

Операция: (TOS) --> dma

Содержимое верхушки стека (TOS) пересылается по адресу адресу dma. Из стека выталкивается один элемент (рис. 7.14)

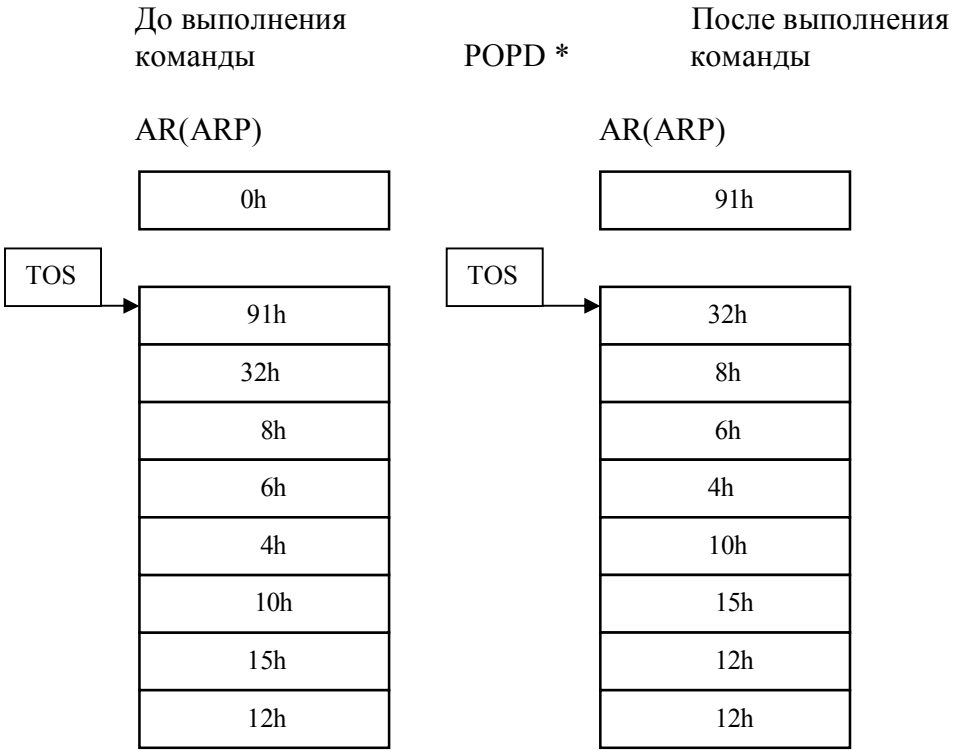


Рис.7.14. Выполнение команды POPD

PUSH (push low accumulator onto stack) - вталкивание младшего слова аккумулятора в стек.

Формат команды:

PUSH

Операция: (ACC(15-0)) --> TOS

Элементы стека последовательно пересылаются на уровень ниже. Содержимое младшего слова аккумулятора помещается в освободившуюся верхушку стека.

PSHD (push data memory value to stack) - вталкивание значения памяти данных в стек.

Формат команды:

PSHD <ac>

PSHD <ка>[,<NARP>]

Операция: (dma) --> TOS

Содержимое памяти данных (dma) помещается в верхушку стека.

TRAP (software interrupt) - программное прерывание.

Формат команды:

TRAP

Операция: (PC) + 1 --> TOS, 30 --> PC

Команда TRAP передает управление команде, находящейся в ячейке 30. В стек вталкивается адрес команды следующей за TRAP, что позволяет вернуться в дальнейшем по команде RET в точку прерывания.

7.13.2 Команды управления циклами

RPT (repeat instruction as specified by data memory value) - команда повторения в соответствии со значением памяти данных.

Формат команды:

RPT <dma>

RPT <ка>[,<NARP>]

Операция: (dma(7-0)) --> RPTC

Команда позволяет поместить в счетчик числа повторений RPTC значения младших восьми битов из памяти данных по адресу dma. При этом коман-да, которая будет записана после RPT, будет повторяться на один раз боль-ше, чем количество повторений, заданное непосредственно числом, загруженным в RPTC. Во время циклического выполнения команды, следующей после RPT, прерывания будут замаскированы. Счетчик числа повторений RPTC очищается сигналом RS.

Пример 7.60

```
(AR(ARP)) = 1030      RPT  *          (RPTC) = Ah
(1030) = Ah
(RPTC) = 0h
```

RPTK (repeat instruction as specified by immediate value) - команда повторения в соответствии с непосредственно заданным значением.

Формат команды:

RPTK <константа>

Операция: 8-битная константа --> RPTC

Команда помещает непосредственно заданную константу в счетчик числа повторений. В остальном команда RPTK аналогична команде RPT.

Пример 7.61

```
LRLK  AR3,0400h; поместить в AR3 начальный адрес X
LARP  3          ; 3 --> ARP
ZAC              ; ACC = 0
RPTK  3          ; повторить 4 раза
ADD  *+          ; X1 + X2 + X3 + X4
```

В приведенном фрагменте программы вычисляется сумма четырех значений данных, расположенных по первым четырем адресам внешней памяти, начиная с адреса 0400h.

7.13.3 Команды управления прерываниями

DINT (disable interrupt) - запрет прерываний.
Формат команды:
DINT
Операция: 1 --> INTM

Команда устанавливает бит режима прерываний INTM в единицу. После этого маскируемые прерывания будут немедленно запрещены. Отметим, что команда LST не устанавливает бит INTM. Немаскируемые прерывания не запрещаются этой командой.

Прерывания также запрещаются при поступлении сигнала сброс.

EINT (enable interrupt) - разрешение прерываний.
Формат команды:
EINT
Операция: 0 --> INTM

Команда устанавливает бит режима прерываний в ноль. После этого будут разрешены маскируемые прерывания.

IDLE (idle until interrupt) - ожидание до прерывания.
Формат команды:
IDLE
Операция: 0 --> INTM

Выполнение команды IDLE приводит к ожиданию прерывания или сигнала сброс. Процессор переводится в режим малого потребления энергии.

7.13.4 Команды загрузки и запоминания регистров состояний

LST (load status register ST0) - загрузить регистр состояний ST0.
Формат команды:
LST <ac>
LST <ка>[,<NARP>]
Операция: (dma) --> ST0

Регистр состояний ST0 загружается содержимым памяти dma. Команда не воздействует на бит INTM и не осуществляет загрузку поле ARB. Команда LST используется для восстановления слова состояния ST0 после прерываний и вызова подпрограмм. Выполнение команды производит установку значений ARP, OV, OVM, DP. Если в команде задано поле NARP, то оно игнорируется.

LST1 (load status register ST1) - загрузить регистр состояний ST1.
Формат команды:

LST1 <ac>
LST1 <ка>[,<NARP>]
Операция: (dma) --> ST1

Регистр состояний ST1 загружается содержимым памяти данных по адресу dma. Биты памяти данных, которые загружаются в поле ARB, также загружаются в указатель ARP. Если в команде задано поле NARP, то оно игнорируется.

SST (store status register) - запомнить регистр состояний ST0.
Формат команды:
SST <ac>
SST <ка>[,<NARP>]
Операция: (ST0) --> dma

Команда позволяет сохранить содержимое регистра ST0 в памяти данных по адресу dma. В случае прямой адресации регистр ST0 сохраняется в нулевой странице независимо от значения DP.

SST1 (store status register ST1) - запомнить регистр состояний ST1.
Формат команды:
SST1 <ac>
SST1 <ка>[,<NARP>]
Операция: (ST1) --> dma

Команда полностью аналогична команде SST.

7.13.5 Команды управления отдельными битами

BIT (test bit) - бит контроля.
Формат команды:
BIT <ac><двоичный код>
BIT <ка><двоичный код>[,<NARP>]
Операция: заданный двоичным кодом бит (dma) --> TC

Команда копирует заданный бит содержимого памяти данных в бит TC регистра состояний ST1. Номер бита, который необходимо скопировать в ST1 задается полем <двоичный код>. Возможные значения этого поля 0..15.

BITT (test bit specified by T register) - бит контроля, заданный T регистром.
Формат команды:
BITT <dma>
BITT <ка>[,<NARP>]
Операция: заданный посредством Treg(3-0) бит (dma) --> TC

Команда аналогична BIT, но номер бита, копируемого в TC, задается четырьмя младшими битами T регистра.

В табл.7.8 приведены команды, воздействующие на биты CNF, C, HM, OVM, TC, SXM. Все команды имеют простейший формат, включающий только мнемонику команды. Команды являются парными – сбро-сить/установить.

Таблица 7.8 - Команды управления отдельными битами

Мнемоника	Описание команды	Операция
CNFD	Сконфигурировать B0 как память данных	0→CNF
CNFP	Сконфигурировать B0 как память программ	1→CNF
RC	Сбросить бит переноса	0→C
SC	Установить бит переноса	1→C
ROVM	Сбросить бит режима переполнения	0→OVM
SOVM	Установить бит режима переполнения	1→OVM
RSXM	Сбросить бит режима знакорасширения	0→SXM
SSXM	Установить бит режима знакорасширения	1→SXM
RHM	Сбросить бит режима захвата	0→HM
SHM	Установить бит режима захвата	1→HM
RTC	Сбросить флаг контроль/управление	0→TC
STC	Установить флаг контроль/управление	1→TC

7.14 Общие вопросы программирования ЦПОС

В данном параграфе рассматриваются общие вопросы программирования ЦПОС TMS320C2x на языке ассемблера. Основное вни-мание уделено рассмотрению полей ассемблерной строки и простейшим директивам ассемблера.

7.14.1 Программирование на языке ассемблера

Программирование на языке ассемблера в отличие от программирования в кодах предполагает запись всех элементов программы в символической форме. Это в значительной степени облегчает процесс разработки и отладки программ. Преобразование символических имен в машинные коды возлагается на специальную программу, называемую *ассемблером*.

Ассемблерные программы записываются в виде последовательности команд, называемых также операторами. Во многих случаях один оператор ассемблера порождает одну машинную команду. Поэтому программирование на ассемблере является удобной формой записи машинных команд и позволяет добиться максимальной эффективности про-

грамм. Однако это требует от программиста полного понимания работы технических средств.

Программа-ассемблер создает объектный файл. Имеется два различных формата объектных файлов: COFF-формат (common object file format) и TI-формат. Здесь и далее рассматривается ассемблер , поддерживающий COFF-формат.

Ассемблерная команда содержит следующие поля:

[<метка>] <мнемоника> <операнды> [<комментарий>]

Необязательное поле *метки* это символическое наименование 16-разрядного адреса той ячейки, в которой будет размещена помеченная ко-манда. Символическое наименование (имя, идентификатор) должно начи-наться с буквы и содержать не более 6 символов. Метки используются в качестве адресов перехода в командах и директивах передачи управления и освобождают программиста от необходимости оперировать абсолютными адресами памяти. Пример метки:

```
Метка   Мнемоника      Операнды      ;Комментарий
LOOP    LACK          0FFFh
          ...
        BGZ           LOOP
        ...
        BLEZ          LOOP
```

Команды BGZ и BLEZ передают управление одной и той же ячейке памяти программ, содержащей команду LACK.

Поле *кода* или *мнемоники* содержит символическое имя команды, которое заменяет собой значение кода выполняемой операции. Мнемоники ассемблерных команд представляют аббревиатуры предложений, характе-ризующих основные функции, выполняемые командой. Длина мнемоники не превышает пяти символов. Поле мнемоники (кода) отделяется от поля метки хотя бы одним пробелом. Мнемоники относятся к ключевым словам ассемблера. Если они записываются с ошибками, то ассемблер формирует соответствующее сообщение.

Содержимое *поля операндов* в значительной степени зависит от команды. В качестве операндов могут фигурировать абсолютные и символические имена адресов памяти, программно доступные внутренние регистры процессора (PRD, TIM, AR0 и др.), адреса портов ввода-вывода (PA1-PA15), числовые и символические константы, режимы косвенной адресации (*+, +0-, *BR0- и др.). Поле операндов для некоторых команд может быть пустым (SST, CALA, RTC и др.).

Содержащееся в поле операнда *шестнадцатиричное число* обязательно должно начинаться с цифры и завершаться буквой h (Hex).

Число, начинающееся с буквы, дополняется слева *незначащим нулем*, например, 0AFCh, 0FCEDh. В TI-формате запись шестнадцатиричных чисел начинается с символа ">". Например, >FF00, >7A и т.п. Любое другое число считается десятичным. В поле операндов разрешается указывать символические имена, которые определены в самом ассемблере и связаны с архитектурой процессора. В язык ассемблера встроены имена программно доступных регистров: AR0-AR7, TIM, PRD, IMR, GREG, DRR, DXR, метки - DATn и PRGn, ассоциирующиеся соответственно с n-ой ячейкой памяти данных и памяти программ.

Вместо имен внутренних регистров допускается указывать их адреса в десятичной или в шестнадцатиричной системе. Например, приведенные ниже команды эквивалентны:

```
SALK  3h              ;  (ACC) --> 3h
SALK  PRD             ;  (ACC) --> PRD
```

В командах передачи управления в поле операнда можно указывать метки, введенные в полях меток других команд. Метки, расположенные в поле операнда, являются *символическими адресами перехода*. В процессе ассемблирования и загрузки программы меткам ставятся в соответствие адреса. Метки, введенные в качестве операнда, должны обязательно один раз появиться в поле метки некоторой команды:

```
        LAC     0200h         ;
        SUBK    01h           ;
        BGZ     METKA1        ; Переход на метку
        ...
METKA1  LAC     0300h         ;
        CALL    SUBR          ; Вызов подпрограммы
        ...
SUBR    SST
        SST1
        ...
```

7.14.2 Директивы ассемблера

В строках ассемблерной программы могут использоваться *директивы*, передающие указания программе ассемблеру о выполнении определенных действий в процессе ассемблирования. Директивы определяют порядок ассемблирования, размещают в памяти команды и данные, присваивают численные значения символическим наименованиям, резервирует память и пр. Рассмотрим упрощенные форматы некоторых директив COFF-ассемблера.

Директива **.title**. Директива заголовка программы .title имеет формат:

.title <строка>

Поле строка представляет последовательность символов (не более 65), заключенных в двойные кавычки. Например,

.title "ПРОГРАММА ФИЛЬТРАЦИИ"

Директива **.set** (установить) выполняет операцию присваивания и имеет следующий формат:

<имя> .set <выражение>

При выполнении директивы имени, стоящему в поле метки, присва-ивается значение выражения. Обычно выражение задается шестнадцатиричным числом. Когда имя встречается в поле операндов команд, программа ассемблер подставляет вместо него присвоенное значение:

```
SADR   .set   00400h
EADR   .set   02000h
 ...
LRLK   AR0,EADR
LRLK   AR1,SADR
```

В командах LRLK вместо имен SADR и EADR будут использованы соответственно значения 400h и 2000h..

Директива **.bss** резервирует память под переменную и имеет следующий формат:

.bss <имя>,<константа>

Числовая константа определяет число ячеек памяти, резервируемых для запоминания переменной. Например, запись .bss var,1 резервирует одну ячейку памяти под переменную var.

Директива **.sect** (секция, сегмент) определяет именованный сегмент и имеет формат:

.sect <строка>

Программа может быть разбита на *сегменты* , которые представляют перемещаемые блоки кода или данных. Сегменты размещаются в памяти процессора. Конкретное размещение сегментов в адресном пространстве памяти определяется на стадии редактирования связей. Поле <строка> соответствует названию сегмента. Например,

```
.sect "reset"
B   MAIN
```

Здесь сегмент “reset” содержит одну команду передачи управления на метку MAIN, соответствующую началу программы. Абсолютный адрес, по которому будет размещена команда, определяется при вызове редактора связей.

Кроме именнованных сегментов объектные файла COFF-формата всегда содержат три стандартных сегмента: .text, .data, .bss.

Директива **.text** определяет сегмент, в котором располагается код программы. Код программы всегда будет помещаться в данный сегмент пока не будет открыт другой сегмент с помощью директив .sect , .data.

Директива **.data** открывает сегмент , в котором обычно размещаются таблицы исходных данных и другие инициализированные переменные.

Существует два основных типа сементов: инициализированные и неинициализированные. *Инициализированные сегменты* содержат данные или код. Сегменты .sect, .text, .data относятся к инициализированным. *Неинициализированные сегменты* резервируют память в адресном пространстве процессора для неинициализированных данных. Например, сегмент .bss .

Ниже приведен пример программы, содержащей директивы .set, .bss, .sect, .text. Для лучшего понимания назначения директив программа содержит дополнительно поле адреса и поле машинного кода:

```
Адрес  Код
       0400  SADR  .set   00400h
       2000  EADR  .set   02000h
                *
0000                .bss   VAR,1
                *
0000                .sect  "RESET"
0000   FF80         B    MAIN
0001   0000’
         *
0000                .text
0000   C800  MAIN   LDRK   0              ; DP <--  0
0001   CA00         LACK   0              ; ACC <--  0
              ...          ...
0005   D000         LRLK   AR0,EADR  ; AR0 <-- EADR
0006   2000
0007   D100         LRLK   AR1,SADR  ; AR1 <-- SADR
0008   0400
                    ...          ...
0012   6000         SACL   VAR           ; (ACC) --> VAR
                    ...
                    .end
```

Второе слово машинного кода команды B содержит относительный адрес перехода 0000' на метку MAIN, который может быть определен только, когда будет известно где располагается сегмент .text. Абсолютные адреса расположения секций задаются пользователем на этапе редак-тирования связей. Следовательно, тогда и заменяются все относительные адреса на абсолютные.

Директива **.bss**, определяющая количество ячеек памяти данных, отводимых для хранения переменных, также формирует относительные адреса на стадии ассемблирования. Абсолютные адреса определяются пользователем при редактировании связей.

Директива **.end** информирует программу-ассемблер о достижении физического конца входной программы.

Одна из основных трудностей программирования на языке ассемблера состоит в том, что весь процесс вычислений должен быть сформулирован в терминах команд процессора. Так как команды процессора выполняют мелкие операции по сравнению с операциями, требующимися для решения задачи на содержательном уровне, то программирование на ассемблере весьма трудоемко.

Для уменьшения трудностей, связанных с указанным обстоятельством, программист может по своему усмотрению определять более содержательные команды, которые затем используются при составлении программы наряду с обычными командами. Такие команды называют макрокомандами. *Макрокоманда* вводится с помощью макроопределения, которое по своей сути весьма близко к определению процедур в языках высокого уровня. *Формат макроопределения*:

<имя макрокоманды> $MACRO [параметр{,параметр}]
.
тело макроопреления
.
$ENDM

Здесь директива $MACRO открывает определение макрокоманды, присваивает ей имя, стоящее слева от слова $MACRO, и задает параметры макрокоманды. Параметры макрокоманды являются необязательными. Вызывается макрокоманда с помощью своего имени и операндов. Когда макрокоманда вызвана, ассемблер сопоставит первый операнд в вызове макроманды с первым параметром в определении макрокоманды и т.д. Директива $ENDM заканчивает определение макрокоманды. При вызове макрокоманды ассемблер подставляет команды, входящие в ее определение, в код исходной программы.

7.15 Программирование основных структур алгоритмов

7.15.1 Программирование разветвляющихся алгоритмов

Одной из характерных особенностей многих алгоритмов обработки сигналов является наличие разветвлений. Для реализации разветвляющихся алгоритмов в процессоре предусмотрены специальные команды условного перехода по значению некоторого признака. Большинство команд условного перехода использует в качестве признака значение, содержащееся в аккумуляторе (BGEZ, BGZ, BLEZ, BLZ, BNZ, BZ). Во всех этих командах выполняется соответствующая команда сравнения содержимого аккумулятора с нулем. Если сравнение завершается успешно, то управление передается по адресу, указанному в команде, в противном случае выполняется следующая команда.

Пусть требуется запрограммировать на языке ассемблера следующий оператор языка высокого уровня:

IF A1>B1 THEN C:=A1+B1 ELSE C:=A1-B1

Здесь A1, B1 и C целые числа, не превышающие по модулю значение 65535. Тогда программа для ЦПОС может быть реализована в следующем виде:

```
        LDPK   4h          ; DP=4
        LAC    A1          ; Загрузить (A1) в аккумулятор
        SUB    B1          ; ((A1)-(B1)) --> ACC
        BGZ    МЕТКА1      ; Переход, если (A1)>(B1)
        SACL   C           ; Сохранить ((A1)-(B1)) в C
        B      МЕТКА2      ; Переход на метку 2
МЕТКА1  LAC    A1          ; (A1) --> ACC
        ADD    B1          ; ((A1)+(B1)) --> ACC
        SACL   C           ; (ACCL) --> C
МЕТКА2  ...
```

В систему команд процессора входит значительное число команд, использующих в качестве признаков перехода значения битов TC, C, OV (BBNZ, BBZ, BNC, BC, BNV, BV). Кроме этого, имеется команда перехода BANZ, выполняющая передачу управления, если содержимое текущего вспомогательного регистра не равно нулю ((AR(ARP)) <> 0).

Рассмотренные команды являются подходящими, когда нужно выбрать одну из двух возможных ветвей алгоритма. Иногда возникает необходимость выбрать одну из k возможных ветвей, где $k > 2$. Пусть требуется составить программу, предназначенную для выбора одной из 4-х

ветвей алгоритма, коды которых располагаются по адресам B0, B4, B8, BC. Для решения задачи может использоваться команда передачи управления по адресу, вычисляемому в аккумуляторе BACC. Предположим, что выбор одной из ветвей алгоритма определяется значением переменной CHOICE (0 < CHOICE < 3), тогда поставленную задачу решает следующий фрагмент программы:

```
        ...
        LDPK   4h          ; DP=4
        LACK   0B0h        ; Инициализация аккумулятора базовым адре-
сом
        LT     CHOICE      ; Загрузить Т-регистр значением CHOICE
        MPYK   4           ; Вычислить относительный адрес перехода
(на
                           пример 2x4=8)
        APAC               ; Сложить с базовым адресом B0
        BACC               ; Переход по одному из адресов:B0, B4, B8,
BC
```

Данный фрагмент программы соответствуют использованию операторов выбора в языках высокого уровня (case в языке Паскаль, switch в языке Си). Однако для более полного соответствия лучше воспользоваться командой CALA (косвенный вызов подпрограммы), пример использования которой будет рассмотрен ниже.

В общем случае команды условного перехода требуют перезагрузки конвейера процессора и дополнительных затрат времени на формирование и проверку условий переходов. Это значительно увеличивает время выполнения программы, если передача управления происходит внутри циклов. Поэтому критические участки программ цифровой обработки сигналов стремятся писать без команд условных или безусловных переходов.

7.15.2 Программирование циклических алгоритмов

Характерной особенностью большинства используемых алгоритмов цифровой обработки сигналов является цикличность. Различают циклы с заранее известным числом повторений и с заранее неизвестным числом повторений. В последнем случае при задании цикла формулируется специальное условие, при выполнении которого циклический процесс должен завершаться. Такие циклы часто называются циклами итерационного типа.

Специфика программирования циклов связана с необходимостью особенно тщательной оптимизации тела цикла, т.е. той части программы, которая выполняется многократно. С этой целью в систему команд процессора включены команды повторений RPT и RPTK, позволяющие про-

граммировать однокомандные циклы. Каждая из этих команд обеспечивает повторное выполнение команды, непосредственно следующей за RPT или RPTK. Количество повторений определяется значением, содержащимся в счетчике RPTC.

Важной особенностью однокомандных циклов, реализованных на основе команд повторений, является сокращение времени выполнения ко-манды, находящейся в теле цикла. Достигается это аппаратными сред-ствами. Повторяющаяся команда считывается из памяти программ только один раз перед выполнением цикла. Действительно, если команда выбрана и находится в регистре команд IR, то при ее повторном выполнении отпадает необходимость в машинном цикле выборки, что сокращает время выполнения команды. Особенно существенный выигрыш получается при повторении команд обмена данными: IN, OUT, TBLR, TBLW, BLKD, BLKP.

После выборки команды и загрузки ее в IR шина команд остается свободной. Поэтому в это время её можно использовать для пересылки данных, адресуемых счетчиком команд PC. Это создает возможность параллельного использования шины данных и шины программ для передачи операндов. Например, команда MAC адресует и пересылает один из операндов посредством шины команд, а второй посредством шины данных. Такая параллельная работа шин возможна благодаря вычислению адресов операндов независимо от выполнения арифметических операций в ЦАЛУ. Адресация операндов происходит путем инкрементации PC и индексации вспомогательных регистров в ARAU. Для параллельного использования шин необходимо, чтобы один из операндов располагался в памяти программ, так как он адресуется счетчиком команд.

Рассмотрим пример. Пусть требуется вычислить скалярное произведение векторов

$$S = \sum_{i=1}^{N} x[i] \cdot y[i],$$

где $N < 256$. Предположим, что элементы вектора x расположены в блоке B0, а вектора y в блоке B1. Тогда программа вычисления S может быть записана в следующем виде:

```
 ...
LARP   AR7          ; AR7 текущий регистр
CNFP                ; B0 программная память
LRLK   AR7,300h     ; 300h начало B1
MPYK   0h           ; Обнулить P-регистр
ZAC                 ; Обнулить ACC
LDPK   0            ; DP=0
RPT    N            ; (N) --> RPTC
MAC    0FF00h,*+    ; S:=S + X[I] x Y[I]; (AR7) + 1 --> AR7
```

```
        APAC            ; Добавление последнего произведения P→ACC
```

Недостатком рассмотренного варианта организации цикла является малое число повторений (не более 256) и включение в тело цикла только одной команды.

Значительно большее число повторений можно получить, если в ка-честве счетчика циклов использовать один из вспомогательных регистров и команду передачи управления BANZ:

```
        ...
        LARP   AR1          ; Выбрать вспомогательный регистр AR1
        LPLK   AR1,3FFh     ; Загрузить в AR1 число повторений
        ZAC                 ; Обнулить ACC
LOOP    ADD    *            ; (ACC) + (dma) --> ACC
        BANZ   LOOP         ; Если (AR1) <> 0, то LOOP --> PC
                ...         ;((AR1)-1) --> AR1
```

Здесь регистр AR1 используется в качестве счетчика цикла. В цикле выполняется суммирование содержимого памяти данных, начиная с ячейки 3FF и заканчивая ячейкой 0h. Тело цикла составляют команды, находящиеся между метками LOOP и командой проверки условия выхода из цикла BANZ. Команда BANZ (по умолчанию) уменьшает содержимое текущего вспомогательного регистра на единицу. Когда содержимое AR1 станет равным нулю, произойдет выход из цикла. В приведенном примере операции по управлению циклом выполняются в ARAU, а арифметические действия, составляющие тело цикла, в ЦАЛУ. Это также сокращает время выполнения цикла.

Ассемблерная реализация итерационных циклов не содержит каких-либо существенно новых элементов программирования по сравнению с рассмотренным выше примером, поскольку определяемый ими процесс также можно задавать с помощью команд условного перехода. Отличие заключается в том, что для проверки выполнения условия выхода из цикла необходимо выполнять более сложные вычисления , которые нельзя обычно реализовать в ARAU. Например, пусть требуется определить минимальное значение i, при котором выполняется неравенство:

$$\sum_i x[i] \geq k \quad .$$

Предполагается, что все элементы $x[i]$ и число K больше нуля. Регистр AR1 будем использовать в качестве указателя памяти данных, а вычисляемую сумму элементов $x[i]$ будем хранить в ячейке SUM. Тогда поставленная задача решается следующим фрагментом программы:

```
...
   LDPK   8h            ; DP=8
   ZAC                  ; 0 --> ACC
   SACL   SUM           ; 0 --> SUM
```

```
        LARP   AR1          ; AR1 - указатель
        LRLK   AR1,0FFFh    ; 0FFFh нач. адрес массива X
REPEAT  LAC    SUM          ; (SUM) --> ACCL
        ADD    *+           ; (ACC)+(AR(ARP)) --> ACC
        SACL   SUM          ; (ACCL) --> SUM
        SUB    K            ; ((ACC)-(K)) --> ACC
        BLZ    REPEAT       ; Если ACC<0, то переход на LOOP
        SAR    AR1,TEMP     ; (AR1) --> TEMP
        LALK 0FFFh          ; 0FFFh --> ACC
        SUB    TEMP         ; FFFh - (TEMP) --> ACC, i
   ...
```

Здесь выход из цикла управляется содержимым аккумулятора. Как только содержимое аккумулятора станет равным нулю или будет больше нуля, управление передается команде SAR. Значение *i* получается путем вычитания адресов начального и конечного элемента *x*[*i*].

Рассмотренный фрагмент программы соответствует циклу с постусловием. В этом случае команды, образующие тело цикла, выполняются хотя бы один раз независимо от условия выхода из цикла. Несложно написать программу, использующую конструкцию "цикл с предусловием", когда проверка окончания цикла выполняется в начале тела цикла:

```
   ...
        LDPK   8h           ; DP=8
        ZAC                 ; 0 --> ACC
        SACL   SUM          ; 0 --> SUM
        LARP   AR1          ; ARP <--  1
        LRLK   AR1,0FFFh    ; AR1 <--  0FFFh
WHILE   SACL   SUM          ; ACC --> SUM
        SUB    K            ; (ACC)-(K) --> ACC
        BGEZ   QUIT         ; Если ACC >= 0, то выход
        LAC    SUM          ; (SUM) --> ACC
        ADD    *+           ; (ACC)+(AR(ARP)) --> ACC
        B      WHILE        ; Переход на метку WHILE
QUIT    SAR    AR1,TEMP     ; (AR1) --> TEMP
        LALK   0FFFh        ; 0FFFh --> ACC
        SUB    TEMP         ; 0FFFh-(TEMP) --> ACC, i
   ...
```

Очевидно, что рассмотренные варианты организации итерационных циклов с переходами на метки и применением команд условных переходов менее эффективны, чем однокомандные циклы, использующие команды RPT и RPTK. Поэтому при программировании алгоритмов цифровой обработки сигналов необходимо стараться применять однокомандные циклы.

7.16 Структуры данных

Под *структурами данных* понимают совокупность элементов данных, между которыми существуют отношения. В зависимости от характера отношений (связей) выделяют различные структуры данных. В области цифровой обработки сигналов наиболее часто используют массивы, стеки, очереди. Рассмотрим возможности ЦПОС по управлению указан-ными структурами.

7.16.1 Массивы

Массив является наиболее простой и распространенной структурой данных. Он представляет собой совокупность однотипных элементов, которые размещаются в смежных ячейках памяти. Положение любого элемента в массиве определяется базовым (начальным) адресом массива и порядковым номером, называемым индексом. Адрес элемента массива равен сумме базового адреса и индекса (индексная адресация).

Индексная адресация в процессоре TMS320C25 реализуется аппаратными средствами с помощью ARAU. В этом случае один из вспомо-

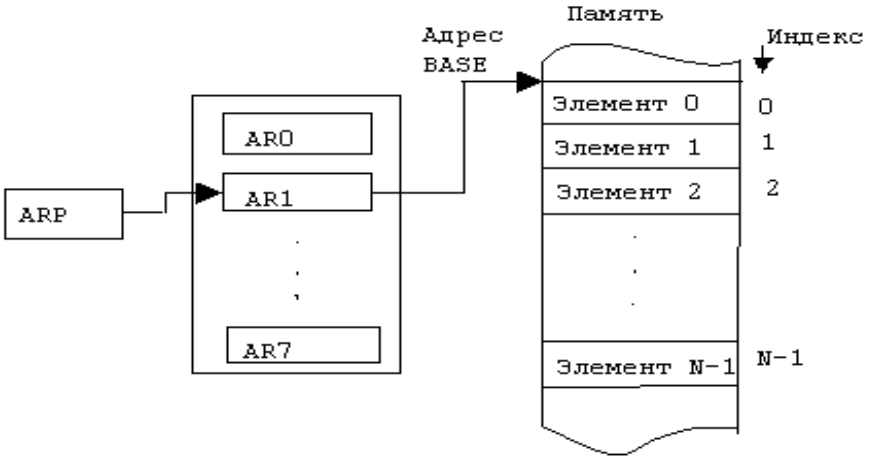


Рис. 7.16. Адресация одномерного массива

гательных регистров, например AR1 (рис.7.16), содержит начальный (базовый) адрес массива BASE и указывает на элемент массива с нулевым индексом. Для доступа к следующему элементу массива содержимое AR1 необходимо увеличивать (уменьшать) на единицу. Как известно, эта операция может быть выполнена в ARAU любой командой с косвенной адресацией.

Обычно обработка массивов осуществляется циклическими программами. Поскольку количество элементов массива известно заранее, то для программирования операций обработки массивов удобно использовать циклы с фиксированным числом повторений, рассмотренные ранее.

В качестве примера обработки массива рассмотрим следующую задачу. Пусть требуется подсчитать количество элементов массива, превы-шающих пороговое значение, хранящееся в ячейке с адресом TRES. Начальный и конечный адрес массива хранятся соответственно в ячейках с адресами BASE и KON.

```
Метка  Код     Операнды     Комментарий
       LDPK    8h           ; DP=8
       RTC                  ; 0 --> TC
       LAR     AR0,KON      ; Конечный адрес в AR0
       LARP    AR1          ; Инициализация
       LAR     AR1,BASE     ; указателя AR1
       LARK    AR2,0h       ; Инициализация счетчика элементов
REP    LAC     *+           ; x[i] --> ACC, инкремент AR1
       SUB     TRES         ; x[i]-(TRES) --> ACC
       BLEZ    METKA        ; x[i] <= TRES
       LARP    2            ; 2 --> ARP, AR2 - счетчик элементов
       MAR     *+           ; (AR2)+1 --> AR2
       LARP    1            ; 1 --> ARP
METKA  CMPR    2            ; Если (AR1) > (AR0), то 1 --> TC
       BBZ     REP          ; Переход ,если TC=0
       SAR     AR2,NUM      ; Количество элементов в ячейке NUM
```

Каждый элемент массива сравнивается с порогом. Если значение элемента массива превышает порог, то счетчик числа элементов AR2 увеличивается на 1; иначе происходит переход к следующему элементу массива. Проверка условия окончания цикла выполняется путем сравнения адреса текущего элемента массива (AR1) с конечным адресом, хранящимся в AR0.

Управление адресами элементов несколько усложняется в случае обработки многомерных массивов. Ограничимся рассмотрением двумерных массивов (матриц), поскольку методика сохраняется и для работы с массивами с большим числом измерений.

Специфика использования многомерных массивов проявляется уже в том, что элементы такого массива можно разместить в памяти различными способами: по строкам или по столбцам. При разных способах размещения получается разной и зависимость адреса элемента массива от значений индексов. Если каждый элемент массива занимает одно адресуемое слово, то адрес элемента определяется следующим образом:

$$\textit{адрес}\,\mathrm{A[i,j]} = \textit{адрес}\,\mathrm{A[0,0]} + \mathrm{i}\cdot\mathrm{nr} + \mathrm{j},\ \text{если массив записан по строкам;}$$

$$\textit{адрес}\,\mathrm{A[i,j]} = \textit{адрес}\,\mathrm{A[0,0]} + i + \mathrm{j}\cdot\mathrm{nc},\ \text{если массив записан по столбцам,}$$

где *nr, nc* - количество элементов в строке и столбце массиве. Нумерация строк и столбцов начинается с нуля.

При обработке многомерных массивов следует выделить такой случай, когда последовательно просматриваются и обрабатываются все его элементы. Например, требуется вычислить сумму всех элементов матрицы или найти максимальный элемент матрицы и т.п. В подобных случаях матрицу размером m x n удобнее трактовать как вектор размером 1 x (m x n). Это позволяет избежать программирования вложенных циклов, обусловленного необходимостью перебора индексов по строкам и столбцам. Управление адресами элементов массива выполняется так же, как и в рассмотренных выше примерах обработки массивов. При этом необходимо предварительно вычислить значение адреса конечного элемента массива.

Если же специфика задачи такова, что обработка матрицы не может быть сведена к последовательной обработке всех ее элементов, то управ-ление адресами выполняется более сложным способом. Например, пусть для квадратной матрицы A размером NxN требуется найти наибольший элемент в нижней треугольной части, включающей главную диагональ. В этом случае для решения поставленной задачи необходимо просматривать из первой строки один элемент, из второй строки - два элемента и т.д. На языке высокого уровня решение подобной задачи можно записать в виде:

```
Y:= A[0,0];
for  i:=0  to  N-1  do
     for  j:= 0  to  i  do
          if  A[i,j] > Y  then  Y:= A[i,j];
 ...
```

В процессоре TMS320C2x удобно выход из циклов организовать с помощью команды BANZ. В этом случае в один из вспомогательных регистров загружают число, значение которого соответствует количеству повторений цикла. В теле цикла уменьшают содержимое регистра до тех пор, пока оно не станет равным нулю. В соответствии с этим приведенный фрагмент программы можно переписать в виде:

```
Y:= A[N-1,N-1];
for  i:= N-1  downto  0  do
    for  j:= i  downto  0  do
         if  A[i,j] > Y  then  Y:= A[i,j];
...
```

Здесь изменен порядок обработки значений индексов. Физически это соответствует движению по адресному пространству от больших адресов к меньшим. Предположим, что массив A хранится в смежных ячейках памяти по строкам, т.е.

адрес A[i,j] = адрес A[0,0] + i N + j =адрес A[0,0] + in + j.

Управление числом повторений по переменной i удобно осуществлять с помощью значения переменной in, которое для рассматриваемой задачи изменяется от значения (N-1)N с шагом -N до значения 0. Переменная j изменяет свое значение от i до 0 с шагом -1. Поскольку верхняя граница изменения j зависит от i, то при каждом изменении i необходимо изменять и значение этой границы. В этом случае массив A будет обрабатываться с конца, т.е. от больших индексов к меньшим. И начальное значение адреса должно соответствовать конечному элемента массива.

В приведенной ниже программе использованы следующие обозначения:
- BASE - адрес ячейки, содержащей адрес последнего элемента массива A;
- JK - адрес ячейки, содержащей начальное значение переменной j.Это значение зависит от переменной i;
- INK - адрес ячейки, содержащей начальное значение переменной in;
- INS - адрес ячейки, содержащей шаг изменения переменной in,(INS) = N;
- KOR - адрес ячейки, содержащей корректирующие значение, позволяющее уточнить адрес текушего элемента массива A.

В программе адрес текущего элемента массива хранится в регистре AR1. При каждом уменьшении на единицу индекса j содержимое AR1 также уменьшается на единицу. При переходе к другой строке матрицы данный порядок нарушается. Необходимо пропустить ряд элементов. Если движение выполняется с конца матрицы, то необходимо при каждом переходе на новую строку пропускать сначала один элемент, затем два и т.д. В ячейке памяти KOR запоминается количество элементов , которое необходимо пропустить при переходе к новой строке матрицы.

```
        ...
     * инициализация
        LAR   AR1,BASE  ; AR1 -указатель на текущий элемент массива
        LAR   AR2,JK    ; Инициализация счетчика j (j=N)
        LAR   AR3,INK   ; Инициализация счетчика in (in=N*N)
        LARK  AR0, KOR      ; Корректирующее значение для AR1,
(KOR)=0

    * присвоение  Y:= A[N-1,N-1]
        LARP  AR1     ;
        LAC   *       ;
        SACL  Y       ; A[N-1,N-1] --> Y

     * вложенные циклы
REPJI  LARP  AR1     ; Текущий регистр AR1
        LAC   *       ; Загрузить A[i,j] в аккумулятор
```

```
        SUB     Y         ; Сравнить A[i,j] и Y,т.е. наити A[i,j]-Y
        BLEZ    MET       ; Если  A[i,j] <= Y, то Y не меняем
        LAC     *         ; Иначе, по адресу  Y
        SACL    Y         ; Сохраняем значение A[i,j]
MET     MAR     *-        ; Уменьшаем физ. адрес массива на 1
        LARP    AR2       ; Работа со счетчиком циклов по j
        MAR     *-        ; Уменьшаем счетчик на 1
        BANZ    REPJI     ; Если счетчик не равен 0, то цикл по j
        LAC     JK        ; Начинаем новый цикл по переменной i
        SUB     JS        ; Модификация JK, т.е. начального значения j
        SACL    JK        ; (JK)-JS --> JK- новое начальное значение j
        LAR     AR2,JK    ; (JK) --> AR2
        LARP    AR0       ;Изменяем
        MAR     *+        ; Корректирующее значение для AR1
        SAR     AR0,KOR   ; Сохраняем  коректирующее значение
        LARP    AR1       ; Корректируем AR1
        MAR     *0-       ; Пропускаем некоторые элементы
        LARP    AR3       ; Изменяем  счетчик in
        LAR     AR0,INS   ; Загружаем шаг изменения in в AR0
        MAR     *0-       ; Уменьшаем in на один шаг INS
        LAR     AR0,KOR ; Восстанавливаем  в AR0 корректирующее
                          ; значение
        BANZ    REPJI     ; Если счетчик in не равен 0, то цикл по i
```

7.16.2 Очереди и кольцевые буферы

Одной из часто используемых структур данных в алгоритмах цифровой обработки сигналов является *очередь.* Элементы, образующие очередь, заносятся и исключаются из неё в процессе выполнения программы. Очередь обслуживается по принципу FIFO (First In First Out): элемент, поступивший первым, первым и обслуживается.

Включение элементов в очередь производят с одного конца (конец очереди), а исключение с другого (начало очереди). Соответственно для работы с очередью вводят указатели начала и конца очереди. Первый адресует элемент, подлежащий исключению, а второй - последний элемент в очереди. Для хранения очереди выделяется некоторая область из смежных ячеек памяти, число которых соответствует максимально возможной длине очереди (рис.7.17,а).

При работе с очередью необходимо учитывать два особых случая: попытка включить элемент в очередь, все ячейки которой заняты; попытка исключения элемента из пустой очереди. Соответственно каждый из этих случаев называется *переполнением или антипереполнением очереди.*

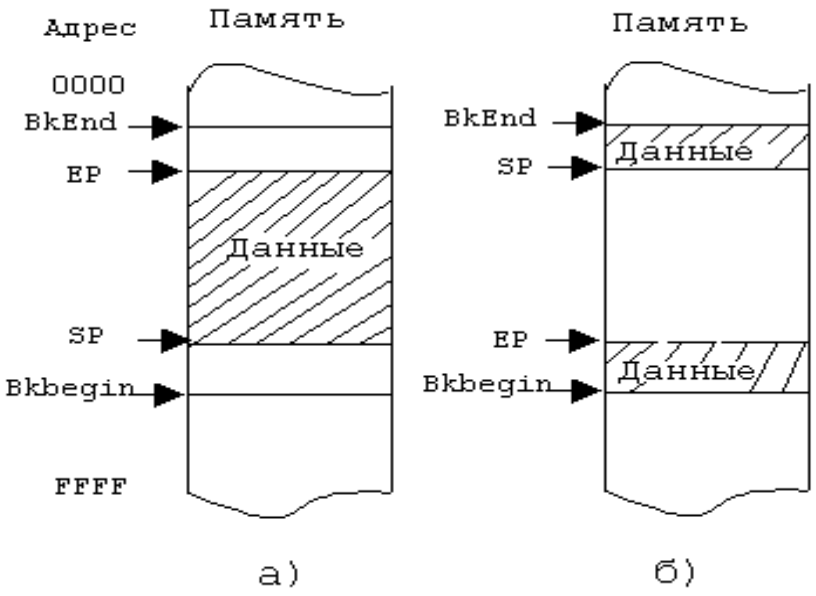


Рис. 7.17. Организация очереди

Для исключения элемента из очереди необходимо выполнить обработку элемента, адресуемого *указателем начала очереди* (SP), и осуществить декремент этого указателя. Включаемый элемент записывается в ячейку, адресуемую *указателем конца очереди* (EP). Обычно указатель конца очереди указывает на первую свободную ячейку. После включения нового элемента в очередь производится декремент указателя конца очереди. При работе с очередью необходимо следить за тем, чтобы она находилась в пределах начальной (указатель BkBegin) и конечной границ (указатель BkEnd) области памяти, отведенной для ее размещения (рис. 7.17,а). Рассмотренный принцип организации очереди соответствует линейной очереди.

В задачах цифровой обработки сигналов часто применяют *кольцевую очередь*, которую называют *кольцевым буфером* (рис.7.17,б). В этом случае значения указателей EP и SP заменяются на BkBegin, если они достигают границы BkEnd. Если указатели EP и SP становятся равными друг другу, то возникает переполнение кольцевой очереди.

Рассмотрим пример программы, выполняющей операции включения элемента в кольцевую очередь и исключения элемента из неё. Переполнение очереди будем фиксировать с помощью бита TC. После возникновения переполнения нельзя выполнять повторно операцию , которая привела к переполнению .

```
*  Включение элемента из аккумулятора в очередь
     LAR    AR1,EP            ; Указатель конца очереди в AR1
     LARP   AR1               ; AR1 текущий регистр
```

```
        SACL   *-                ; Сохранить аккумулятор в конце очереди
                                   ; (AR1)-1 --> AR1
        LAR    AR0,BKEND         ;Конечная граница в AR0
        RTC                      ; Установить бит TC в ноль
        CMPR   1                 ; Достигнута ли нижняя граница
        BBZ    M1                ; Если нет, то модификация EP не нужна
        LAR    AR1,BKBEGIN       ; Модификация EP
M1      LAR    AR0,SP            ; Проверка переполнения
        RTC                      ; 0 --> TC
        CMPR   0                 ; Если EP=SP, то 1 --> TC. Переполнение
        SAR    AR1,EP            ; Сохранить EP
          ...
*  Исключение элемента из очереди
        LAR    AR2,SP            ; Указатель начала очереди в AR2
        LARP   AR2               ; AR2 текущий регистр
        LAC    *-                ; Исключаемый элемент в аккумуляторе
        LAR    AR0,BKEND         ; Нижняя граница в AR0
        RTC                      ; Установить бит TC в ноль
        CMPR   1                 ; Достигнута ли нижняя граница?
        BBZ    M2                ; Если нет, то модификация SP не нужна
        LAR    AR2,BKBEGIN       ; Модификация SP
M2      LAR    AR0,EP            ; Проверка антипереполнения
        RTC                      ; 0 --> TC
        CMPR   0                 ; Если SP=EP,то 1-->TC.Антипереполнение
        SAR    AR2,SP            ; Запоминание SP
          ...
```

В алгоритмах цифровой обработки сигналов часто используют "синхронное" перемещение указателей SP и EP. После включения заданного количества элементов в очередь, такое же количество элементов исключается, т.е. в очереди хранится определенное число элементов, подлежащих обработке. Такая организация данных может использоваться при программировании фильтров с конечной импульсной характеристикой, вычислении преобразований Фурье и др.

Приведенные фрагменты программ легко адаптировать применительно к рассматриваемому случаю. Для этого их следует объединить и исключить проверку на переполнение и антипереполнение. Так как при некотором резерве памяти равенство указателей EP и SP исключается.

Кольцевая очередь может быть эффективно реализована с помощью бит-инверсной адресации. Наиболее просто при использовании бит-инверсной адресации реализуется кольцевая очередь длины N, где N-является степенью числа 2.

В случае бит-инверсной адресации указатель, ссылающийся на текущий элемент очереди, меняет свои значения с шагом, определяемым содержимым регистра AR0. При этом в ARAU выполняется операция суммирования *с распространием переноса в обратную сторону* (т.е. не влево, а вправо). Перенос из младшего значащего бита игнорируется. В таблице 7.9 приведен пример вычисления адресов для случая организации в памяти кольцевого буфера из 8-ми элементов. Если в AR0 поместить значение N/2, где N – длина буфера (в рассматриваемом случае (AR0)=4h) , то через N-шагов произойдет переход к начальному адресу, т.е. формируются адреса , обеспечивающие доступ к элементам кольцевого буфера. При этом адреса формируются в бит-инверсном порядке. С целью упрощения таблицы вычисление адресов показано только для трех младших значащих битов (м.з.б.). Вычисляемые адреса будут находиться в пределах, определяемых начальным адресом буфера и его длиной. Начальный адрес буфера выбирается таким образом, чтобы его младшие $\log_2 N$ разрядов имели нулевые значения.

Таблица 7.9 - Вычисление адресов бит-инверсным способом

Режим косвенной адресации	Адрес элемента	Вычисление адреса (3 м.з.б.)
Начальный адрес →	0200h	
*BR0+	0204h	000+100=100
*BR0+	0202h	100+100=010
*BR0+	0206h	010+100=110
*BR0+	0201h	110+100=001
*BR0+	0205h	001+100=101
*BR0+	0203h	101+100=011
*BR0+	0207h	011+100=111
*BR0+	0200h	111+100=000

7.16.3 Стеки

Стек функционирует по принципу LIFO (Last In First Out): элемент, поступивший последним, обслуживается первым. Такая дисциплина обслуживания целесообразна во многих случаях, в частности, при работе с различными подпрограммами. При работе со стеком можно выполнять операции включения элемента в стек и исключения элемента из стека. Максимальное количество слов, запоминаемых в стеке, называется его *глубиной*.

В процессоре имеется аппаратно реализованный стек, глубина кото-рого ограничена и равна восьми. Включение и исключение данных из аппаратного стека выполняется с помощью команд PUSH и POP. Эти команды позволяют проводить обмен данными между младшим словом аккумулятора и верхушкой аппаратного стека (TOS). Кроме этого, в

систему команд процессора включены команды PUSHD и POPD, выполняющие обмен данными между верхушкой стека и памятью данных. Эти команды обычно используют для сохранения и восстановления содержимого аппаратного стека в памяти программ.

В случае обработки данных, требующей большой глубины стека, его организуют программно в смежных ячейках памяти данных(рис.7.18) с начальной и конечной границами - BkBegin, BkEnd. Для обращения к верхушке стека используют указатель TSP. При исключении элемента из стека необходимо выполнить декремент указателя TSP и прочитать соответствующий элемент. Включаемый элемент записывается в ячейку, на которую указывает TSP, после этого выполняется его инкремент. В процессе выполнения операций со стеком необходимо обнаруживать случаи, когда указатель TSP указывает на границу BkBegin (антипереполнение) или BkEnd (переполнение). В приведенных ниже примерах при возникновении переполнения (антипе-реполнения) устанавливается бит TC. После возникновения переполнения или антипереполнения нельзя повторно выполнять операцию, которая привела к этому.

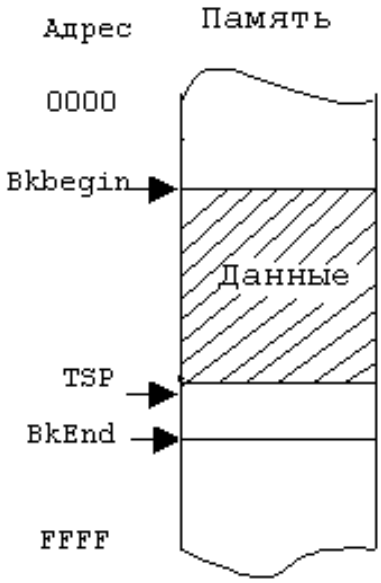


Рис.7.18. Стек

Рассмотрим пример программы, выполняющей включение элементов в стек и исключение элементов из стека:

```
*  включение в стек содержимого аккумулятора
        LARP   AR1              ; AR1 указатель TSP
        LAR    AR1,TSP          ; Загрузка указателя
        SACL   *+               ; Включение элемента в стек
        RTC                     ; Сброс бита TC
        LAR    AR0, BKEND       ; Загрузить конечную границу
        CMPR     0                          ; Если TSP=BKEND, то 1 -->
TC.Переполнение
        SAR    AR1,TSP          ; Запомнить указатель стека
        ...
   * исключение из стека в аккумулятор
        LARP   AR1              ; AR1 - указатель TSP
        LAR    AR1,TSP          ; Загрузка указателя
        MAR    *-               ; Перемещение указателя назад
```

```
        LAC    *                ; Извлечение из стека
        RTC                     ; Сбросить бит TC
        LAR    AR0,BKBEGIN      ; Загрузить нижнюю границу
        CMPR 0          ; Если TSP=BKBEGIN,то 1-->TC.Антипереполнение
        SAR    AR1,TSP          ; Запомнить указатель стека
        …
```

В этом примере ячейка с адресом BkBegin используется для записи элементов стека, а ячейка с адресом BkEnd не используется.

7.17 Подпрограммы

Если в разных местах алгоритма приходится выполнять одно и то же действие, например, вычисление функции при различных значениях аргумента, то реализацию этого действия целесообразно описать в виде подпрограммы. В том случае, если возникает необходимость в вычислении функции, осуществляется вызов подпрограммы.

Вызов подпрограммы нарушает естественный порядок исполнения команд. Из точки вызова управление передается первой команде подпрограммы, затем выполняются команды подпрограммы, а по ее завершению управление обратно передается в точку вызова.

Вызов подпрограммы выполняется с помощью команд CALL и CALA. Команда CALL передает управление подпрограмме с заданным именем, которое находится в поле метки первой команды подпрограммы. Команда CALA передает управление по адресу, находящемуся в аккумуляторе. При выполнении подпрограммы текущее содержимое PC (адрес возврата) загружается в аппаратный стек, а в счетчик команд PC загружается либо адрес, связанный с именем подпрограммы (команда CALL), либо содержимое младшего слова аккумулятора (команда CALA). Заключительной командой подпрограммы должна быть команда RET, которая извлекает адрес возврата из стека и передает его в PC. Следовательно, теперь будет выполняться команда, находящаяся в главной программе за командой CALL/CALA. На рис.7.19 показана схема взаимодействия основной программы и подпрограммы. Внутри вызванной подпрограммы может быть вызов следующей подпрограммы и т.д. Это соответствует вложению подпрограмм. Аппаратный стек процессора позволяет реализовать до восьми вложенных вызовов подпрограмм.

Применение подпрограмм требует временного запоминания состояния основной программы. Запомнить содержимое регистров основной программы можно в начале вызываемой подпрограммы. После выполнения подпрограммы восстанавливают содержимое регистров основной программы.

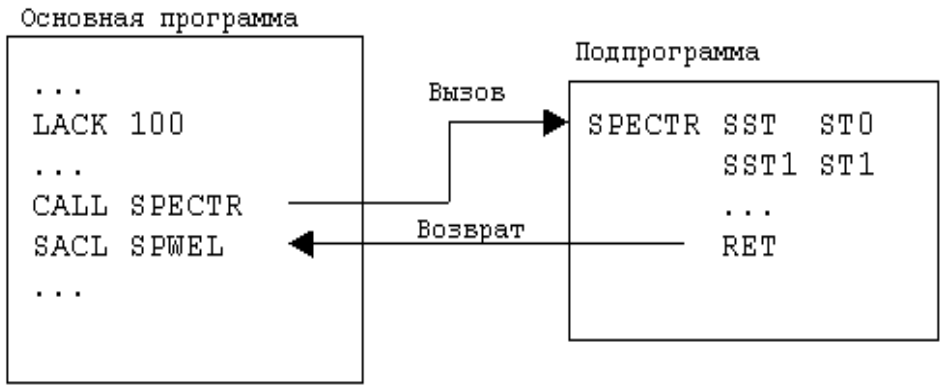


Рис.7.19. Взаимодействие основной программы и подпрограммы

Часто для временного запоминания состояния основной программы используют аппаратный или программный стек. При вызове подпрограммы необходимо временно запоминать содержимое тех регистров, которые будут использоваться подпрограммой. Например, если подпрограмма SUBR использует регистры ARP, AR1 и ACC, то она должна сохранять и восстанавливать их:

```
SUBR  LARP  AR2      ; AR2 указатель программного стека
      SST   *-       ; Запоминание ARP (ST0)
      SAR   AR1,*+   ; Запоминание AR1
      SACH  *+       ; Запоминание ACCH
      SACL  *+       ; Запоминание ACCL
      ...
      Текст    подпрограммы
      ...
      MAR   *-       ; Возврат указателя
      ZALS  *-       ; Восстановление ACCL
      ADDH  *-       ; Восстановление ACCH
      LAR   AR1,*-   ; Восстановление AR1
      LST   *        ; Восстановление ARB (ST0)
      RET
```

Здесь регистр AR2 содержит указатель на верхушку программного стека. Отметим, что из-за особенностей работы со стеком восстановление выполняется в обратном порядке.

Команда CALA передает управление по адресу, вычисляемому в аккумуляторе. Это позволяет использовать таблицу адресов подпрограмм и передавать управление различным подпрограммам из одной точки программы.

Таблица адресов, число элементов в которой равно числу вызываемых подпрограмм, размещается в фиксированной области памяти программ (рис.7.20). Элементами таблицы являются начальные адреса со-ответствующих подпрограмм. В аккумулятор загружается базо-вый адрес таблицы TBL, после чего к нему добавляется номер строки таблицы, определяющий вызов конкретной подпрограммы. Полученная сумма используется в качестве адреса памяти программ, по которому в аккумулятор считывается соответствующая строка таблицы. Команда вызова CALA включает в верхушку стека адрес возврата и передает управле-ние по адресу, находящемуся в аккумуляторе.

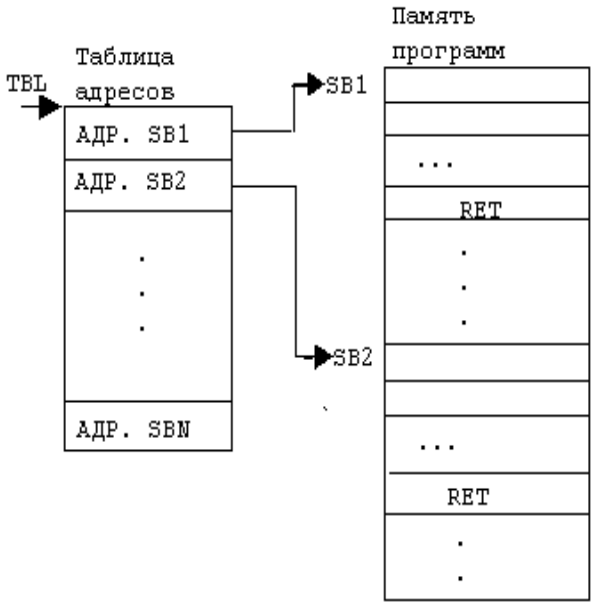


Рис.7.20 Таблица адресов

Рассмотрим пример. Пусть требуется вычислить одну из следующих функций *f0(x)*, *f1(x), fк(x*). В качестве исходных данных задаются значения аргумента *x* и номер *i* функции, подлежащей вычислению. Пусть таблица адресов содержит адреса F0, F1, ... FK подпрограмм, вычисляющих функции *f0(x), f1(x), ... fк(x).*

Тогда задача может быть решена с помощью следующего фрагмента программы:

```
...
LALK   TBL     ; Загрузить в ACC базовый адрес
ADD    I       ; Добавить номер функции (селектор)
TBLR   TEMP    ; (TBL + I) --> TEMP
LAC    TEMP    ; Загрузка адреса TEMP --> ACC.
CALA           ; Вызов соответствующей подпрограммы
...
```

7.18 Обработка прерываний

При поступлении запроса прерывания устанавливается бит регистра признаков прерываний (IFR). Если соответствующий бит регистра маски прерываний (IMR) установлен в единицу и прерывания разрешены (INTM=0), то выполняется обработка прерывания. После загрузки адреса подпрограммы обработки прерываний в программный счет-

чик, преры-вания запрещаются (INTM=1) и управление передается программе обра-ботки прерываний, которая при необходимости может разрешить преры-вания.

7.18.1 Сохранение контекста

Перед началом обслуживания прерывания программа обработки прерываний должна сохранить состояние прерванной программы, а после окончания обслуживания - восстановить состояние. Данные действия аналогичны тем, которые выполняются при вызове подпрограмм. При этом можно выполнять полное или частичное сохранение контекста прерванной программы. Это зависит от тех действий, которые необходимо выполнить для обслуживания прерывания.

В процессоре программный счетчик автоматически сохраняется в аппаратном стеке. Поэтому в программном стеке обычно сохраняют содержимое регистров состояний (ST0 и ST1), аккумулятора (ACCH,ACCL), регистра произведения (PR), регистра временного хранения (T), всех восьми регистров аппаратного стека, вспомогательных регистров (AR0-AR6). Седьмой вспомогательный регистр используется в качестве указателя программного стека. Во время сохранения или восстановления контекста все прерывания должны быть запрещены.

Ранее при рассмотрении подпрограмм были рассмотрены примеры сохранения и восстановления регистра состояний, аккумулятора, вспомогательных регистров. Здесь рассмотрим особенности сохранения и восстановления P- и T-регистров, аппаратного стека.

Так как команды SPH и SPL обеспечивают запоминание старшего и младшего слова P-регистра с возможностью сдвига, то перед запоминанием необходимо записать в поле PM регистра ST1 ноль. При этом регистр ST1 должен быть сохранен раньше, чем будет выполнено это действие:

```
        ...
ISR     LARP   AR7  ; AR7 указатель стека
        ...
        SST1   *+   ; сохранение ST1
        ...
        SPM    0    ; 0 --> PM
        SPH    *+   ; Запоминание PH
        SPL    *+   ; Запоминание PL
        ...
```

Некоторую особенность имеет сохранение T-регистра, так как процессор не имеет специальных команд, обеспечивающих эту операцию. Поэтому содержимое T-регистра умножается на 1 и передается в P-

регистр, а затем сохраняется младшее слово P-регистра, равное содержимому T-регистра:

```
...
MPYK   1    ; (T) x 1 --> P
SPL    *+   ; Запомнить T-регистр
...
```

Запоминание содержимого аппаратного стека выполняют в цикле, используя команду POPD:

```
...
RPTK   7
POPD   *+
...
```

После завершения обслуживания прерывания выполняют восстановление состоянии прерванной программы. Из-за особенностей стека это восстановление осуществляется строго в обратном порядке. Содержимое аппаратного стека восстанавливают командами:

```
...
RPTK   7
PSHD   *-
...
```

Восстановление P и T-регистров имеет особенность, обусловленную отсутствием в системе команд процессора команды загрузки P-регистра. Поэтому загрузка P-регистра выполняется через T-регистр:

```
...
*  восстановление P-регистра
   MAR    *-  ; Пропустить T-регистр
   LT     *+  ; Загрузить в T-регистр, сохр.значение P-регистра
   MPYK   1   ; (T) x 1 --> PL. Загрузка P-регистра

*  восстановление T-регистра
   LT     *    ; Загрузка T-регистра
   MAR    *-   ; Пропуск P-регистра
   ...
   LST1   *   ; Восстановить ST1, AR7 указывает на первую свободную ячейку
   EINT       ; Разрешить прерывания
   RET        ; Возврат в прерванную программу
```

Для того чтобы выполнить рассмотренную программу, необходимо при поступлении запроса на обслуживание прерывания передать управление на метку ISR.

Запросы прерываний в процессоре обрабатываются в соответствии с их приоритетами. Иногда необходимо внутри программы обработ-

ки прерываний обрабатывать другое прерывание. Например, при обработке редких запросов прерываний, требующих длительного обслуживания, целесообразно разрешить их прерывание по запросам, которые случаются чаще и требует небольшого времени обслуживания.

Пусть в программе обработки прерывания INT1 необходимо разрешить обслуживание прерывания INT2. Для этого следует модифицировать регистр маски прерываний IMR, запретив все прерывания, кроме INT2. Это потребует предварительного сохранения содержимого IMR, чтобы не нарушить вычислительный процесс основной программы.

В приведенном фрагменте программы обслуживания прерывания INT1, регистр AR7 используется в качестве указателя верхушки стека:

```
ISSR  LARP  AR7    ; AR7 указатель стека
      SST1  *+     ; Запомнить ST1
      SST   *+     ; Запомнить ST0
        ...
      LDPK  0      ; Загрузить в указатель страниц 0
      PSHD  IMR    ; Сохранить IMR в аппаратном стеке
      LACK  0004h  ; Маска для INT2
      AND   IMR    ; (ACC) AND (IMR) --> ACC
      SACL  IMR    ; ACC --> IMR
      EINT         ; Разрешить прерывания
        ...
      команды обслуживания прерывания INT1
        …
      DINT         ; Запрет всех прерываний
      LDPK  0      ;
      POPD  IMR    ; Восстановить IMR
      LARP  AR7    ; AR7 указатель стека
      MAR   *-     ; Возврат указателя
        ...
      LST          ; Восстановление ST0
      LST1         ; Восстановление ST1
      EINT         ; Разрешить прерывания
      RET          ; Возврат в прерванную программу
```

Приведенная программа может быть выполнена, если в ячейку 4 памяти программ поместить команду передачи управления на метку ISSR.

7.18.3 Прерывание таймера

Таймер процессора содержит регистры TIM и PRD, адресуемые как ячейки нулевой страницы памяти данных с адресами 2 и 3. Со-

держимое регистра TIM связано с количеством периодов сигнала CLKOUT1, поступивших на вход таймера с момента последнего прерывания TINT или RS. При поступлении каждого импульса сигнала CLKOUT1 содержимое TIM уменьшается на единицу. Начальное значение, помещаемое в регистр TIM, хранится в регистре периода PRD. Когда содержимое регистра TIM станет равным нулю, формируется прерывание TINT и в следующем периоде сигнала CLKOUT в TIM загружается начальное значение из PRD. Отсюда следует, что прерывание TINT формируется через интервалы времени, равные [(PRD) + 1] x Tclkout1.

Обычно прерывание TINT используют для получения отсчетов обрабатываемого входного сигнала, поступающего на вход ЦПОС. В этом случае содержимое PRD и частота взятия отсчетов $f_Д$ сигнала связаны соотношением:

$$(PRD) = [fclkout1/ f_Д] - 1,$$

где fclkout1 - частота следования импульсов сигнала CLKOUT1 (fclkout1 = fclkin/4); [.] - операция взятия целой части. Пусть $f_Д$ = 10 КГц и fclkin = 40 МГц, тогда (PRD)=999.

Ниже приведен фрагмент программы установки таймера для форми-рования заданного значения частоты прерываний TINT:

```
...
LACK   999  ;
LDPK   0      ; DP=0
SACL   3h     ; Загрузить регистр периода
LACK   8h     ; 1000 --> ACC
OR     4h     ;
SACL   4h     ; 1 --> IMR(4)
EINT          ; Разрешить прерывания
```

7.18.4 Инициализация процессора по прерыванию

Обычно перед началом выполнения алгоритмов цифровой обработки сигналов требуется выполнить предварительную настройку (инициализацию) процессора. Общая инициализация осуществляется при подаче на вход процессора сигнала сброс RS, который является немаскируемым прерыванием с наивысшим приоритетом, т.е. такое прерывание нельзя запретить.

Сигнал RS обеспечивает установку счетчика команд PC в нулевое состояние и тем самым управление передается команде, находящейся в нулевой ячейке. Обычно в этой ячейке помещают команду безусловного перехода к программе инициализации.

После сброса процессор должен быть настроен на выполнение функций, определяемых требованиями, предъявляемыми системой циф-

ровой обработки. При этом необходимо учитывать, что сигнал RS выполняет следующие установки:

регистр ST0:
0 → OV; 1 → INTM
регистр ST1:
0 → CNF; 1 → SXM; 1 → C; 1 → HM; 1 → FSM;
1 → XF; 0 → FO; 0 → TXM; 0 → PM
регистр GREG: 0 → GREG
счетчик циклов RPTC: 0 → RPTC
таймер TIM: FFFFh → TIM
регистр PRD: FFFFh → PRD
регистр IFR: 0 → IFR

Значения полей OVM, ARP и DP регистра состояний ST0 и полей ARB, TC регистра состояний ST1 не устанавливаются сигналом сброс. Кроме этого, не определенным остается содержимое регистра IMR и ОЗУ. Поэтому после сброса необходимо выполнить инициализацию тех полей ST0 и ST1, которые не получили значений, а также задать структуру прерываний, определив содержимое IMR. Помимо этого, обнуляют содержимое внутреннего ОЗУ, аккумулятора и меняют значения регистров, установленные сигналом RS, если они не отвечают заданным требованиям. Ниже приведен фрагмент программы инициализации процессора:

```
INIT   SOVM              ; Установить бит режима переполнения
       LDPK   0          ; DP указывает на страницу 0
       LARP   5          ; (ARP)=5
       LACK   3Fh        ; Загрузить аккумулятор константой 3F
       SACL   4          ; Разрешить все прерывания.(ACC) --> IMR
  * заполнение внутренней памяти нулями
       ZAC               ; Обнулить аккумулятор
       LARK   AR5,60h    ; Загрузить в AR5 адрес начала B2
       RPTK   31         ;
       SACL   *-         ; Записать 0 во все  ячейки B2
       LRLK   AR5,200h   ; Загрузить в AR5 адрес начала B0
       RPTK   255        ;
       SACL   *+         ; Записать 0 во все  ячейки B0
       LRLK   AR5,300h   ; Загрузить в AR5 адрес начала B1
       RPTK   255        ;
       SACL   *+         ; Записать 0 во все   ячейки B1
       EINT              ; Разрешить все прерывания
```

Приведенный фрагмент программы разрешает все прерывания. Это достигается записью во все разряды регистра IMR единиц и установкой

бита INTM в ноль (команда EINT). После выполнения программы устанавливается режим переполнения, указатель страниц DP обнуляется, в качестве текущего вспомогательного регистра выбирается регистр AR5, а внутреннее ОЗУ (блоки B0, B1, B2) заполняется нулями. Приведенная программа может быть выполнена, если в ячейку 0 памяти программ поместить команду передачи управления на метку INIT.

7.19 Прикладные программы

7.19.1 Сложение и вычитание

Выполнение операции сложения и вычитания 16-разрядных чисел (однословных чисел) не вызывает затруднений, так как процессор имеет соответствующие команды - ADD и SUB. В этих командах одним из операндов является содержимое аккумулятора , а другим - содержимое адресуемой ячейки памяти. Команды ADD и SUB устанавливают биты переполнения OV и переноса C.

Сложение многословных чисел начинается со сложения младших слов, затем складываются следущие слова чисел с учетом переноса от предыдущего сложения и т.д. В приведенной ниже подпрограмме сложения двух 64-разрядных чисел предполагается , что слагаемые размещаются в области памяти с начальным адресом , определяемым содержимым регистра AR1 (например, (AR1)=105). Результат вычислений помещается в область памяти , определяемую содержимым регистра AR2 (например, (AR2)=114). Каждое слагаемое и результат представлены в памяти четырьмя 16-разрядными словами.

```
ADD64   LARP  AR1     ;(AR1)=105
        ZAC   *       ; 0→ACC
        LAC   *+      ; ACC=00A1, (AR1)=106
        ADDH  *+      ; ACC=A2A1, (AR1)=107
        ADDS  *+      ; ACC=A2A1+00B1,(AR1)=108
        ADDH  *+      ; ACC=A2A1+B2B1, (AR1)=109
        LARP  AR2     ; (AR2)=114
        SACL  *+      ; ACCL→114, (AR2)=115
        SACH  *+      ;ACCH→115, (AR2)=116
        LARP AR1      ;
        ZAC   *       ;0→ACC
        LAC   *+      ; ACC=00A3, (AR1)=110
```

```
 ADDH   *+        ; ACC=A4A3, (AR1)=111
 ADDC    *+       ; ACC=A4A3+00B3+C,(AR1)=112
 ADDH   *+        ; ACC=A4A3+B4B3+C, (AR1)=113
LARP    AR2      ; (AR2)=116
 SACL    *+       ; ACCL→116, (AR2)=117
 SACH    *+       ;ACCH→117, (AR2)=118
 RET
```

В программе сначала обнуляется аккумулятор и загружается младшее слово слагаемого А. Команда ADDH помещает в аккумулятор второе слово слагаемого А и обнуляет бит переноса . После этого содержимое аккумулятора складывается с двумя младшими словами слагаемого В (команды ADDS, ADDH.). Команды LARP,SACH,SACL обеспечивают сохранение двух младших слов результата по адресу, хранящемуся в AR2. Затем выполняется сложение старших слов слагаемых А и В. При этом к полученной сумме добавляется значение бита переноса (команда ADDC). Окончательный результат помещается в ячейки определяемые содержимым AR2.

Вычитание многословных чисел выполняется аналогично. При этом следует использовать команду вычитания с заемом SUBB.

7.19.2 Умножение и накопление значений

Процессор содержит аппаратный умножитель, который выполняет умножение 16-разрядных чисел и формирует 32-разрядный результат. Операция умножения может выполняться с помощью команд MPY и MPYU. Команды осуществляют умножение операндов, находящихся в Т регистре и в ячейке адресуемой памяти. Команда MPYU интерпретирует операнды как беззнаковые числа и позволяет при необходимости организовать перемножение “длинных” чисел. Произведение, получаемое с помощью указанных команд, помещается в Р регистр. Таким образом, перед выполнением операции умножения требуется загрузить в Т-регистр первый операнд, а после выполнения – сохранить значение Р регистра :

```
…
LT     OP1       ;Загрузить OP1 в T регистр
MPY  OP2         ;Умножить OP1 на OP2
SPL    PRL       ;Сохранить младшее слово по адресу PRL
SPH    PRH       ; Сохранить старшее слово адресу PRH
…
```

Наряду с командами умножения, процессор имеет группу команд, выполняющих умножение и накопление произведений в аккумуляторе (MAC , MPYA). Команда MPYA осуществляет сложение содержимого Р регистра с аккумулятором, а затем перемножает операнды, находящиеся

в Т регистре и в адресуемой ячейке памяти данных. Команда MAC отличается тем, что перемножает операнды, находящиеся в памяти программ и памяти данных. При этом в качестве памяти программ используется блок B0 , а в качестве памяти данных - блоки B1 или B2. Обычно команда MAC находится в пределах области действия команды RPTK, что позволяет выполнять её за один цикл:

```
…
LARP    AR2     ;
LRLK    AR2,300h    ;Загрузить в AR2 начальный адрес
CNFP                ; B0 – память программ
ZAC                 ; Обнулить аккумулятор
MPYK    0           ; Обнулить P регистр
RPTK    255         ; 256 циклов
MAC     0FF00h,*+   ;Умножить и сложить (B0)x(B1)+ACC;
                    ; (AR2)+1→AR2
APAC                ; Добавить последнее произведение
…
```

В приведенном примере вычисляется сумма произведений содержимого соответствующих ячеек блоков B0 и B1. Адресация выполняется с помощью счетчика команд, который инкрементируется автоматически, и регистра-указателя AR2, инкрементируемого в команде MAC. Команда CNFP распределяет блок B0 на память программ.

Действия, выполняемые командой MAC, можно реализовать парой команд LTA и MPY. LTA выполняет загрузку Т-регистра и суммирование P регистра с аккумулятором, а команда MPY перемножает содержимого Т регистра с содержимым адресуемой ячейки памяти. По времени выполнения команда MAC эффективнее пары команд LTA-MPY, если число повторений более восьми. При малом числе повторений лучше использовать пару команд LTA-MPY:

```
…
ZAC             ;Обнулить аккумулятор
MPYK    0       ;Обнулить P регистр
LTA     X1      ;загрузить X1 в T регистр, (ACC)+0→ACC
MPY     Y1      ;Умножить X1xY1→P
LTA     X2      ;Загрузить X2 в T регистр, X1xY1→ACC
MPY     Y2      ;Умножить X2xY2→P
LTA     X3      ;Загрузить X3 в T регистр,
                ; X1xY1 + X2xY2→ACC
MPY     Y3      ;Умножить X3xY3→P
APAC            ; X1xY1 +X2xY2 +X3xY3→ACC
…
```

Процессор позволяет выполнять не только суммирование произведений , но и вычитание . Для этого могут использоваться команды MPYS, LTS-MPY, SPAC.

Во многих задачах цифровой обработки сигналов требуется вычислять сумму или разность квадратов. Эти операции эффективно реализуются соответственно командами SQRA и SQRS:

```
LARP        AR1       ;
LRLK        AR1,300h  ; Загрузить в AR1 начальный адрес
ZAC                   ;Обнулить аккумулятор
MPYK        0         ; Обнулить P регистр
RPTK        19        ; 20 циклов
SQRA        *+        ; (ACC)+XI**2→ACC
APAC                  ;(ACC)+X20**2→ACC
```

7.19.3 Операции над числами с плавающей запятой

Система команд процессора включает инструкции, которые могут использоваться для выполнения операций над числами с плавающей запятой: NORM, LACT, ADDT, SUBT. Команда нормализации NORM может применяться для преобразования чисел из формата с фиксированной запятой в формат с плавающей запятой. LACT применяется для выполнения обратных преобразований. Сложение и вычитание чисел с плавающей запятой выполняется при помощи команд ADDT и SUBT.

Рассмотрим пример подпрограммы умножения чисел, представленных в формате с плавающей запятой. Будем полагать , что мантисса числа представлена 16-разрядным дробным числом с фиксированной точкой (рис.1.39), а порядок является целым числом. При умножении двух чисел с плавающей запятой результат получается суммированием порядков чисел и перемножением их мантисс. Пусть перемножаются два числа $A = MA \cdot 2^{PA}$ и $B = MB \cdot 2^{PB}$, где *MA* и *MB*- соответственно мантиссы чисел *A* и *B*, *PA* и *PB* – соответственно порядки чисел *A* и *B*. Тогда результат будет равен:

$$C = MA \cdot MB \cdot 2^{PA+PB}$$

Так как мантиссы *MA* и *MB* считаются нормализованными, то для нормализации мантиссы результата его необходимо сдвинуть на бит влево, выполнив после этого необходимую коррекцию порядка результата (команда NORM *-):

```
*
*подпрограмма умножения чисел с плавающей запятой
*
MULT    LAC     PA      ;Загрузить порядок числа A
        ADD     PB      ;Сложить с порядком числа B
        SACL    PC      ;Сохранить порядок результата
        LT      MA      ;Загрузить в T регистр мантиссу A
        MPY     MB      ;Умножить на мантиссу B
        PAC             ;Поместить произведение в ACC
        SFL             ;Удаление лишнего знаков.бита
        LARP    AR1             ;AR1 текущий указатель
        LAR     AR1,PC          ;Загрузить порядок результата
        NORM    *-      ;Нормализац. ACC , коррекция PC
        SACH    MC      ;Сохранить мантиссу результата
        SAR     AR1,PC          ;Сохранить порядок
        RET
```

При работе с числами в форме с плавающей запятой иногда требуется их преобразование в форму с фиксированной запятой . Указанную операцию удобно выполнять с помощью команды LACT. Ниже приведен пример подпрограммы денормализации числа с плавающей точкой . Предполагается , что мантисса находится в аккумуляторе , а порядок в регистре AR2.

```
*
*Подпрограмма денормализации числа с плавающей запятой
*
DEN     LARP    AR1             ;AR1-текущий указатель
        LRLK    AR1,200h        ;
        SAR     AR2,*+          ;Сохранить порядок по адресу 200
        SACH    *-              ;Сохран. мантиссу по адресу 201
        LAC     *               ;Загрузить в ACC порядок
        BZ      EX1     ;Если порядок равен 0, то переход
        LT      *+      ;Поместить порядок в T регистр
        LACT    *               ; Мантисса * 2**(T(3-0))→ACC
        RET                     ;Возврат
EX1     MAR     *+              ;Установить указат. на мантиссу
        ZALH    *               ;Поместить результат в ACC
        RET
```

Данная подпрограмма может использоваться в том случае если порядок занимает в памяти не более 4-х бит . Это связано с особенностями команды LACT.

Рассмотренные примеры обработки чисел с плавающей запятой являются простейшими и имеют ряд ограничений. Перемножаемые числа должны быть одного знака, а порядок – положительным. Имеются также ограничения на длину мантиссы и порядка. Для общепринятых операций с числами в форме с плавающей запятой разработаны специальные библиотеки, снимающие указанные ограничения.

7.19.4 Реализация алгоритмов цифровой фильтрации

Для вычисления одного выходного отсчета y[n] нерекурсивного цифрового фильтра (НРФ) в соответствии с (1.212) необходимо вычислить N раз произведение $b[k]\cdot x[n-k]$. Вычисление следующего отсчета выходного сигнала $y[n]$ требует перемещения входной последовательности $x[n]$ вдоль элементов задержки (рис.1.34). Это соответствует “скольжению” отсчетов входного сигнала относительно отсчетов импульсной характеристики. Простейшая реализация НРФ основана на циклическом применении команды MACD (умножение и накопление с пересылкой данных):

```
RPTK        N-1
MACD    COEFB,*-
```

В этом случае коэффициенты фильтра $b[k]$ должны располагаться в памяти программ (COEFB-начальный адрес этой области памяти). Для этих целей может использоваться встроенное ПЗУ/ППЗУ, ОЗУ B0, или внешнее ОЗУ. Отсчеты входного сигнала x[n] должны располагаться во внутрикристальном ОЗУ (блоки B0 и B1), иначе пересылка данных не выполняется. Если коэффициенты фильтра хранятся во внешнем ОЗУ или во внутреннем ПЗУ/ППЗУ, то для хранения отсчетов входного сигнала можно использовать весь объём внутреннего ОЗУ. Это позволяет реализовать нерекурсивные фильтры, импульсная характеристика содержит 512 отсчетов.

Если память, в которой хранятся коэффициенты $b[k]$ фильтра, не вносит задержек, то команда MACD обрабатывает один отсчет импульсной характеристики фильтра за 100 нс. Ниже приведен фрагмент программы, реализующей нерекурсивный фильтр с 256 коэффициентами, хранящимися в блоке ОЗУ B0, распределенном на адресное пространство памяти программ. Вспомогательный регистр AR1 к моменту окончания выполнения программы содержит адрес текущего входного отсчета.

```
…
      CNFP                ;Распределить B0 на память программ
      LARP    AR1         ;AR1-текущий указатель
      LRLK    AR1,3FFh    ;Конечный адрес блока B1
```

```
        MPYK    0           ;Обнулить P регистр
        ZAC                 ;Обнулить аккумулятор
        RPTK    255         ;256 циклов
        MACD    0FF00h,*-   ;Сумма произведений b[k]*x[n-k]
        APAC                ;Добавить последнее произведение
        SACH    Y           ;Сохранить выходной отсчет в Y
…
```

При практической реализации НРФ данный фрагмент программы повторяется для каждого нового отсчета входного сигнала, помещаемого в блок ОЗУ B1 по адресу 0300h. Во время циклического выполнения команды MACD осуществляется последовательная пересылка отсчетов входного сигнала , хранящихся в блоке B1, начиная со старших адресов. Это соответствует продвижению отсчетов $x[n]$ вдоль элементов задержки (рис.1.34). Начальное содержимое блока ОЗУ B1 оказывает влияние на ход переходного процесса в фильтре. После заполнения всех ячеек блока B1 соответствующими осчетами входного сигнала на выходе фильтра будет установившийся режим.

В том случае, если блок B1 не может быть использован, в программу приходится включать дополнительную команду BLKD , обеспечивающую необходимое перемещение отсчетов $x[n]$ вдоль элементов задержки. Поскольку такую команду приходится выполнять циклически для каждого из входных отсчетов, то время выполнения программы нерекурсивной фильтрации значительно увеличивается.

Если входные отсчеты разместить в кольцевом буфере и перемещать не значения $x[n]$, а соответствующие указатели, то указанный недостаток можно устранить. Ниже приведена программа нерекурсивной фильтрации, в которой для хранения значений входного сигнала используются кольцевой буфер с бит-инверсной адресацией (см.§ 7.16.2).

```
SEGM    .set    128             ;длина фильтруемого участка
        .bss    BUFFER,32       участок памяти под буфер
        .bss    Y,1             ;выходная переменная

COEFF   .sect   “coefficients”  ; сегмент памяти программ
        .word 4231h             ;для размещения коэф.фильтра
        .word 6F34h             ; и т.д. 32 коэффициента
        …

        .text
INIT    LARK    AR0,10h         ;10000→AR0
        LRLK    AR1,BUFFER ; AR1 ссылается на начало
                                ; буфера
```

```
        LRLK    AR2,SEGM  ;AR2-счетчик вх. отсчетов
        LARP    AR1       ;AR1 текущий указатель
        LDP     Y         ;DP указ. на текущую стр.

FIRFILT IN      *,PA1             ;Ввод вх. отсчета
        MPYK    0h                ;Обнулить P регистр
        ZAC                       ;Обнулить аккумулятор
        RPTK    31                ; 32 цикла
        MAC     COEFF,*BR0+ ;b[k]*x[n-k]+(ACC)→ACC
        APAC                      ;Добавить послед. произв.
        SACH    Y,1               ;Сохранить результат
        OUT     Y,PA2             ;Вывести результат
        MAR     *BR0-,AR2         ;Указатель назад
        BANZ    FIRFILT,*-,AR1    ;Перейти на FIRFILT
```

В программе с помощью директивы ассемблера .bss выделяется память под буфер и выходную переменную Y. По умолчанию редактор связей разместит сегмент .bss в памяти данных. Инициализированный именованный сегмент “coeffcients” по умолчанию размещается в памяти программ. Директива .word размещает 16-разрядные коэффициенты фильтра в ячейки памяти программ, выделенные текущему сегменту. Доступ к этой области памяти осуществляется через метку COEFF.

В сегменте, начинающемся директивой .text, приведен текст программы. Часть программы от метки INIT до метки FIRFILT обеспечивает необходимую инициализацию вспомогательных регистров и указателя страниц. В соответствии с принципом организации кольцевого буфера , использующего бит-инверсную адресацию, в регистр AR0 помещается значение , равное половине длины буфера, т.е. 16 (10000h). Регистр AR1 настраивается таким образом, чтобы указывать на начало буфера. В регистр AR2 записывается константа, определяющая длину реализации фильтруемого сигнала и соответсвенно число циклов, которые необходимо выполнить, чтобы вычислить все значения выходного сигнала Y.

Команды, реализующие непосредственно алгоритм нерекурсивной фильтрации и действия по управлению кольцевым буфером, записаны после метки FIRFILT. Свертка входной последовательности и последовательности коэффициентов фильтра вычисляется с помощью команды MAC. При этом значения отсчетов входной последовательности $x[n]$ извлекаются из буфера в бит-инверсном порядке. Это предполагает запись значений $x[n]$ в буфер в таком же порядке. Ввод значений в буфер выполняется командой IN. Ячейка буфера, в которую будет записано значение $x[n]$, определяется указателем AR1. Благодаря бит-инверсной адресации этот указатель всегда имеет значения в пределах выделенного буфера памяти. Каждый раз вновь вводимое значение $x[n]$ замещает в

буфере самое раннее значение, которое было в нем сохранено. Для этого, после циклического выполнения команды MAC и использования всех значений буфера, выполняется возврат указателя AR1 на последнее использованное значение буфера. Это значение является самым ранним значением, сохранённым в буфере. Указанный возврат выполняется командой MAR *BR0- , учитывающей бит-инверсную упорядоченность буфера. После выполнения возврата указателя в команде BANZ проверяется условие завершения цикла (AR2)=0. Если условие не выполняется, то происходит переход на метку FIRFILT и команда IN вводит из порта PA1 очередное значение $x[n]$ и помещает его по адресу, на который указывает AR1. Вычисленные значения выходного сигнала $y[n]$ выводятся с помощью команды OUT через порт PA2.

Если сравнить ядро программы нерекурсивной фильтрации, использующей кольцевой буфер, с предыдущим вариантом программы, базирующейся на применении команды MACD, то видно , что применение кольцевого буфера добавило лишь одну лишнюю команду по управлению буфером (MAR). Важным преимуществом последнего варианта программы является возможность реализации фильтров с большим числом коэффициентов (более 512). Однако при этом приходится использовать более медленную внешнюю память.

При реализации рекурсивных фильтров (РФ) требуются меньшие объемы памяти. С целью уменьшения шумов квантования РФ реализуют в виде последовательно соединенных биквадратных блоков в канонической форме (1.217)-(1.218). Для биквадратного блока уравнения (1.217)-(1.218) можно записать в виде:

$$u[n] = x[n] - a[1] \cdot u[n-1] - a[2] \cdot u[n-2],$$
$$y[n] = b[0] \cdot u[n] + b[1] \cdot u[n-1] + b[2] \cdot u[n-2].$$

Введение вспомогательной последовательности $u[n]$ позволяет сократить объем требуемой памяти по сравнению с прямой формой реализацией РФ. Приведенные уравнения биквадратного РФ можно реализовать либо с помощью команды MACD, либо с помощью пары команд LTD-MPY. При реализации РФ последний вариант предпочтительнее (см.§ 7.19.2).

Рассмотрим фрагмент программы, реализующей биквадратный блок РФ. Будем полагать, что текущий отсчет входной последовательности хранится в ячейке с адресом X, а выходной отсчет в ячейке с адресом Y. Коэффициенты фильтра располагаются соответственно в ячейках памяти данных: A1, A2, B0, B1, B2. Отсчеты вспомогательной последовательности хранятся соответственно в ячеках UN, UN1, UN2. Пересылка данных UN→UN1→UN2 осуществляется при выполнении коман-

ды LTD. Коэффициенты A1 и A2 хранятся в дополнительном коде. Это позволяет заменить операции вычитания операциями сложения:

```
…
LAC     X,15    ;Загрузить в ACC x[n]**(2**15)
LT      UN1     ;u[n-1]→T
MPY     A1      ;a[1]*u[n-1]
LTA     UN2     ;a[1]*u[n-1]+x[n], u[n-2]→T
MPY     A2      ;a[2]*u[n-2]
APAC            ; a[2]*u[n-2]+a[1]*u[n-1]+x[n]→ACC,
SACH    UN,1    ;u[n]=( a[2]*u[n-2]+a[1]*u[n-1]+x[n])*2
ZAC             ;Обнулить ACC
MPY     B2      ;b[2]*u[n-2]
LTD     UN1     ; b[2]*u[n-2]→ACC, u[n-1]→T
MPY     B1      ;b[1]*u[n-1]
LTD     UN      ; b[1]*u[n-1]+ b[2]*u[n-2]→ACC,u[n]→T
MPY     B0      ;b[0]*u[n]
APAC            ; b[0]*u[n]+ b[1]*u[n-1]+ b[2]*u[n-2]→ACC
SACH    Y,1     ; (ACC)*2, ACCH→Y
```

Программа вычисляет для каждого входного отсчета $x[n]$ выходной отсчет $y[n]$. Для получения последовательности отсчетов $y[n]$ необходимо периодически повторять вычисления. Поля сдвигов, указанные в командах LAC и SACH, предназначены для масштабирования и удаления лишнего знакового бита.

7.19.5 Реализация алгоритма БПФ

Рассмотрим пример программы, реализующей базовую операцию БПФ в соответствии с уравнениями (1.263) и (1.264). Будем полагать, что все входные и выходные значения представлены в 16-разрядном формате в виде дробных чисел с фиксированной запятой, которая следует сразу после знакового бита. Соответственно для представления значений без учета знака отводится 15 разрядов.

В программе выполняется масштабирование с коэффициентом 2. При этом полагается, что значения поворачивающего множителя W_N^k уже масштабированы, т.е.

$$W_N^k = (1/2)\cdot[\cos(2\pi k/N) - j\cdot\sin(2\pi k/N)] = (1/2)\cdot[\cos(x) - j\cdot\sin(x)].$$

Действительная и мнимая части W_N^k запоминаются соответственно в программных переменных WR и WI. Переменные PR,PI,QR,QI используются как для хранения соответствующих значений входных переменных выражений (1.263) и (1.264) , так и для хранения результи-

рующих значений этих выражений. Иными словами, при реализации операции “бабочка” для хранения входных и выходных значений используются одни и те же ячейки памяти.

Программа оформлена в виде макроса, который можно вызывать при различных значениях параметров-переменных PR,PI,QR,QI,WR,WI. В теле макроса указанные параметры выделяются с двух сторон двоеточием.

Макрос состоит из четырех блоков. В первом блоке вычисляется действительная часть произведения $Q_m \cdot W_N^k$, которая с учетом масштабирующего коэффициента равна $(1/2)\cdot[QR\cdot\cos(x)+QI\cdot\sin(x)]$. Полученное значение сохраняется в переменной QR, так как текущее значение QR более не требуется для дальнейших вычислений.

```
BUTFLY    $MACRO   PR,PI,QR,QI,WR,WI
*
*WR=(1/2)*cos(x), WI=(1/2)*sin(x)
*
*Вычисление (QR*WR+QI*WI), сохранение результата в QR
*
LT        :QR:     ; Загрузить QR в T-регистр
MPY       :WR:     ;(1/2)*QR*cos(x)
LTP       :QI:     ;(1/2)*QR*cos(x)→ACC, QI→T
MPY       :WI:     ;(1/2)*QI*sin(x)→P
APAC               ;(1/2)*[QR*cos(x)+QI*sin(x)]→ACC
LT        :QR:     ;Сохранить QR в T-регистре
SACH      :QR:     ;(1/2)*[QR*cos(x)+QI*sin(x)]→QR

* Вычисление (QI*WR-QR*WI), сохранение результата в QI
*
ZAC                ; Обнулить ACC
MPY       :WI:     ;(1/2)*QR*sin(x)
SPAC               ; -(1/2)*QR*sin(x)→ACC
LT        :QI:     ; QI→T
MPY       :WR:     ;(1/2)*QI*cos(x)→P
APAC               ;(1/2)*[QI*cos(x)-QR*sin(x)]→ACC
SACH      :QI:     ;(1/2)*[QI*cos(x)-QR*sin(x)]→QI
```

*Вычисление действительных частей Pm+1 и Qm+1,*сохранение в PR и QR

```
LAC       :PR: ,14    ; (1/4)*PR→ACC
ADD       :QR: ,15    ;(1/4)*(PR+[QR*cos(x)+QI*sin(x)])→ACC
SACH      :PR: ,1     ;(1/2)*(PR+[QR*cos(x)+QI*sin(x)])→PR
SUBH      :QR:        ;(1/4)*(PR-[QR*cos(x)+QI*sin(x)])→ACC
SACH      :QR: ,1     ;(1/2)*(PR-[QR*cos(x)+QI*sin(x)])→QR
```

*Вычисление мнимых частей Pm+1 и Qm+1, сохранение в PI и QI

```
LAC     :PI: ,14    ; (1/4)*PI→ACC
ADD     :QI: ,15    ;(1/4)*(PI+[QI*cos(x)-QR*sin(x)])→ACC
SACH    :PI: ,1     ;(1/2)*(PI+[QI*cos(x)-QR*sin(x)])→PI
SUBH    :QI:        ;(1/4)*(PI-[QI*cos(x)-QR*sin(x)])→ACC
SACH    :QI: ,1     ;(1/2)*(PI-[QI*cos(x)-QR*sin(x)])→QI
$END
```

Во втором блоке вычисляется масштабированное значение мнимой части произведения $Q_m \cdot W_N^k$, равное $(1/2)\cdot[QI\cdot\cos(x)-QR\cdot\sin(x)]$. Вычисленное значение запоминается в переменной QI.

В третьем блоке вычисляются действительные значения выходных переменных P_{M+1} и Q_{M+1} и обеспечивается деление полученных значений на 2. Здесь значение переменной PR сначала масштабируется с коэффициентом 4. Для этого значение PR сдвигается вправо на два разряда. Данная операция выполняется при загрузке PR в аккумулятор. Сдвиг 16-разрядного двоичного слова вправо на два разряда соответствует его размещению в 32-разрядном аккумуляторе с предварительным сдвигом слова влево на 14 разрядов (команда LAC PR,14). После сдвига к значению PR добавляется действительная часть произведения $Q_m \cdot W_N^k$, хранящаяся в переменной QR , которая также масштабируется с коэффициентом 4. Поскольку в QR хранится значение действительной части $Q_m \cdot W_N^k$ уже масштабированное с коэффициентом 2, то при добавлении QR к PR необходимо выполнить сдвиг двоичного значения QR вправо на один разряд. Для этого при выполнении команды сложения QR с содержимым 32-х разрядного аккумулятора выполняется эквивалентный сдвиг QR влево на 15 разрядов. Старшее слово аккумулятора сдвигается влево на один разряд, чтобы получить значение $(1/2)\cdot[PR+[QR\cdot\cos(x)+QI\cdot\sin(x)])$, точно равное действительной части P_{M+1}, и запоминается в переменной PR. Последний сдвиг осуществляеется при передаче значения из аккумулятора в переменную PR . В аккумуляторе остается значение, масштабированное с коэффициентом 4. Для вычисления действительной части Q_{M+1} из содержимого аккумулятора вычитается значение, хранящееся в переменной QR , т.е.

$$(1/4)[PR+[QR\cos(x)+QI\sin(x)])-(1/2)[QR\cos(x)+QI\sin(x)]=$$
$$=(1/4)[PR-[QR\cos(x)+QI\sin(x)])\ .$$

Полученное значение запоминается в переменной QR. В четвертом блоке по аналогичной схеме вычисляются значения мнимых частей P_{M+1} и Q_{M+1}.

Отметим , что все масштабирования выполняются сдвигающими устройствами процессора и не требуют дополнительных затрат времени.

Рассмотренный макрос BUTFLY , реализует одну операцию “бабочка”. При вычислении БПФ последовательности , имеющей длину N, макрос должен вызываться $(N/2)\log_2 N$ раз. В частности, при вычислении 8-точечного БПФ макрос должен вызываться двенадцать раз. При этом входная последовательность x[n] должна быть упорядочена в соответствии с принципом бит-инверсной адресации. Ниже приведена программа, реализующая 8-точечное БПФ. В программе значения исходной последовательности вводятся из порта PA0 и размещаются в бит-инверсном порядке в блоке B0. При этом вводятся только действительные значения, которые размещаются от начала блока по четным адресам. Результаты выводятся через порт PA1 в естественном порядке. Вызовы макросов выполняются в соответствии с граф-схемой 8-точечного БПФ, изображенной на рис . 1.21. Значения поворачивающих коэффициентов хранятся во внешней памяти данных, начиная с адреса 0F400h.

```
*8-точечное БПФ с прореживанием по времени
*
* Распределение входных и выходных переменных
X0R         .set        0;
X0I         .set        1;
X1R         .set        2;
X1I         .set        3;
X2R         .set        4;
X2I         .set        5;
…
X7R         .set        14;
X7I         .set        15;

* Распределение значений  поворачивающих коэффициентов
W0R         .set        16;
W0I         .set        17;
W1R         .set        18;
W1I         .set        19;
W2R         .set        20;
W2I         .set        21;
W3R         .set        22;
W3I         .set        23;
```

```
            .text
* Инициализация
FFT8        CNFD
            SPM         0
            SSXM
            LDPK        4
*Ввод значений и сохранение в бит инверсном порядке
            LARK        AR0,8       ;Загрузить в AR0  N/2
            LRLK        AR1,200h    ;Загрузить в AR адрес B0
            LARP        AR1
            RPTK        7
            IN          *BR0+,PA0 ; Ввести веществ. значения x[n]
            LRLK        AR1,210h    ;Загрузить в AR адрес W0R
            RPTK        7
            BLKD        0F400,*+    ;Прочитать поворотные коэф.

*Первый этап БПФ
            BUTFLY X0R,X0I,X4R,X4I,W0R,W0I
            BUTFLY X2R,X2I,X6R,X6I,W0R,W0I
            BUTFLY X1R,X1I,X5R,X5I,W0R,W0I
            BUTFLY X3R,X3I,X7R,X7I,W0R,W0I
*Второй этап БПФ
            BUTFLY X0R,X0I,X2R,X2I,W0R,W0I
            BUTFLY X4R,X4I,X6R,X6I,W2R,W2I
            BUTFLY X1R,X1I,X3R,X3I,W0R,W0I
            BUTFLY X5R,X5I,X7R,X7I,W2R,W2I
*Третий этап БПФ
            BUTFLY X0R,X0I,X1R,X1I,W0R,W0I
            BUTFLY X4R,X4I,X5R,X5I,W1R,W1I
            BUTFLY X2R,X2I,X3R,X3I,W2R,W2I
            BUTFLY X6R,X6I,X7R,X7I,W3R,W3I
*Вывод результатов в последовательной форме
            LRLK        AR1,200
            RPTK        15
            OUT          *+,PA1
            RET
```

Приведенную программу можно улучшить, если учесть то обстоятельство, что в некоторых случаях значения поворачивающего множителя равны 1, либо –j. В этом случае не требуется умножение на поворачивающий множитель. Кроме того, эффективность первого этапа БПФ можно повысить, если применить БПФ по основанию 4. Соответствующий вариант программы приведен в [35].

Глава 8
ЦПОС С ПЛАВАЮЩЕЙ ЗАПЯТОЙ СЕМЕЙСТВА *ADSP*-21XXX

8.1 Общая характеристика процессора ADSP-2106x

Цифровой сигнальный процессор *ADSP*-2106x *SHARC (Super Harvard Architecture Computer)* представляет собой высокопроизводительную 32-х разрядную однокристальную специализированную микро-ЭВМ с плавающей запятой, предназначенную для использования в различных областях науки и техники для обработки ультразвуковых и звуковых сигналов, речи, радиотехнических сигналов, неподвижных и подвижных изображений. Процессор принадлежит к семейству цифровых сигнальных процессоров с плавающей запятой *ADSP*-21000, которые являются совместимыми вниз с ЦПОС семейства 16-разрядных процессоров *ADSP*-2100 с фиксированной запятой.

SHARC строится на основе ядра цифрового сигнального процессора семейства микросхем *ADSP*-21000. Для формирования завершенной однокристальной микро-ЭВМ к нему добавляется (рис.8.1) двухпортовая память (статическое ОЗУ), находящаяся на кристалле, процессор ввода/вывода, контроллер ПДП, параллельный порт системной шины и ряд других устройств.

Разработано два *ADSP SHARC*-процессора: *ADSP*-21060 и *ADSP*-21062. Первый содержит 4 Мегабита ОЗУ на кристалле, а второй - 2 Мегабита. Во всех других отношениях процессоры одинаковы и являются функционально- и кодово-совместимыми с *ADSP*-21020. Устройство *SHARC* представляет собой новый стандарт цифровых сигнальных процессоров для выполнения операций с плавающей запятой. Процессоры *ADSP*-21000 выполняют все инструкции в течение одного машинного цикла. Они обеспечивают малое время цикла и полный набор арифметических операций, включая операции вычисления выражений 1/x, $\sqrt{1/x}$, а также нахождение min и max, “вырезание” фрагмента изображения *(clip),* и вдобавок к традиционным операциям сложения, вычитания и умножения введена комбинированная операция умножение со сложением (*MAC*-операции). Процессоры являются совместимыми с *IEEE*-стандартом для операций с плавающей запятой.

Рис.8.1. Обобщенная структура СБИС семейства ADSP-2106x

На рис.8.2. изображена расширенная структурная схема *ADSP*-2106x. В его состав входят собственно цифровой сигнальный процессор (ядро), двухпортовое статическое ОЗУ, процессор ввода/вывода и интерфейсный процессор (внешний порт). Ядро состоит из следующих блоков:

- 32-разрядного вычислительного устройства для выполнения операций с плавающей запятой в соответствии со стандартом *IEEE*, состоящего из умножителя, АЛУ и многорегистрового устройства циклического сдвига;
- блока регистров общего назначения (файла данных);
- двух генераторов адресов данных (ГА-1 и ГА-2);
- контроллера микропрограмм;
- кэш-памяти команд;
- таймера;
- коммутатора шин.

Статическое ОЗУ построено на основе двух независимых двухпортовых блоков оперативной памяти. Интерфейсный процессор состоит из схемы мультиплексора шины адреса и интерфейса главного процессора.

Процессор ввода/вывода содержит:

- блок управления с буфером данных и регистрами ввода/вывода;
- контроллер ПДП;
- два последовательных порта;
- шесть линейных портов связи.

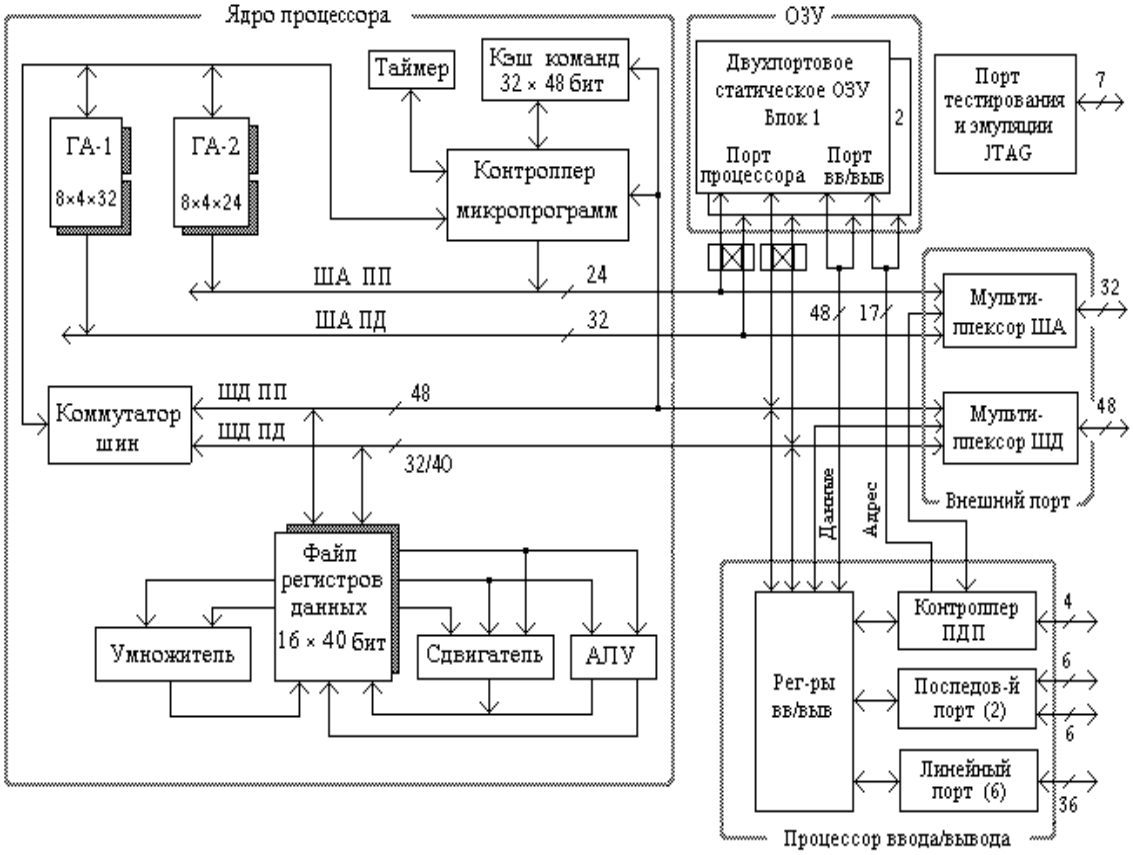


Рис.8.2. Расширенная структурная схема *ADSP*-21060 *SHARC*

В состав микросхемы *ADSP*-2106x входит также порт тестирования и эмуляции с интерфейсом *JTAG*.

Внешний порт (интерфейсный процессор) микросхемы обеспечивают связь ядра процессора с внешней памятью, с устройствами ввода/вывода, отображаемых на память, а также связь с главным процессором и дополнительными *ADSP*-2100x, входящими в состав многопроцессорной системы. Он также выполняет внутренний и внешний шинный арбитраж.

Таким образом, использование в *ADSP*-2106x модифицированной гарвардской архитектуры в комбинации с 10-портовым блоком (файлом) регистров данных общего назначения позволяет на каждом цикле одновременно выполнять:

- чтение или запись двух операндов в файл регистров данных;
- пересылку двух операндов в АЛУ;
- подачу двух операндов на умножитель;
- получение результатов операций из АЛУ и умножителя.

Использование 48-битовых команд процессора обеспечивает параллельную передачу данных и кодов арифметических операции в одной и

той же инструкции, что позволило существенно увеличить скорость обработки.

Процессоры семейства *ADSP*-21000 обрабатывают 32-битный *IEEE* формат с плавающей запятой, 32-битные целочисленные и дробные форматы (без знаковые и в дополнительном коде) и 40-битовый *IEEE* формат расширенной точности с плавающей запятой. Процессоры распространяют расширенную точность на все свои устройства счета, уменьшая промежуточные ошибки округления данных. При работе с данными внутри ЦПОС 32-разрядная мантисса с расширенной точностью может быть передана во все (или получена из всех) устройства счета. 40-разрядная шина данных может быть, при желании, организована вне кристалла. Форматы с фиксированной запятой используют 80-разрядный аккумулятор для обработки 32-битовых слов с фиксированной запятой.

Рассмотрим несколько подробнее назначение и особенности функционирования основных боков микросхемы *SHARC*. *ADSP*-2106х содержит три независимых устройства счета*: АЛУ, умножитель с аккумулятором и сдвигающее устройство.* Для удовлетворения широкого ряда потребностей обработки, вычислительные устройства обрабатывают данные в трех форматах: 32 - битовые слова с фиксированной запятой, 32 - битовые и 40-битовые слова с плавающей запятой. Операции над числами с плавающей запятой являются *IEEE*-совместимыми. 32-разрядный формат с плавающей запятой является стандартным *IEEE*-форматом, в то время как 40-битный *IEEE*-формат расширенной точности имеет 8 дополнительных бит мантиссы для обеспечения большей точности вычислений.

АЛУ выполняет стандартный набор арифметических и логических операций как в формате с фиксированной запятой, так и в формате с плавающей запятой. Умножитель производит умножение с плавающей и с фиксированной запятой, а также совместное умножение/сложение и умножение/вычитание с фиксированной запятой. Сдвигающее устройство осуществляет логические и арифметические сдвиги, манипуляции с битами, копирование полей, операции выделения (маскирование) и нормализацию 32-битовых операндов. Счетные устройства выполняют одноцикловые операции без использования конвейера. Выход любого устройства счета может быть входом любого другого в следующем цикле. При умножении АЛУ и умножитель выполняют независимые операции одновременно.

Файл регистров данных общего назначения используется для пересылки данных между устройствами счета и шинами данных и для хранения промежуточных результатов. Файл регистров данных имеет два набора регистров (основной и дополнительный) по 16 регистров в каж-

дом, для использования быстрого контекстного переключения при применении мультизадачного режима. Все регистры 40-разрядные.

Контроллер микропрограмм (КМП) и **генераторы адреса данных** (ГА-1 и ГА-2). Два генератора адреса и КМП генерируют адреса для доступа к памяти. Вместе КМП и генераторы адреса данных позволяют вычислительным операциям выполняться с наибольшей эффективностью, т.к. устройства вычисления используются исключительно для обработки данных. Имея в своем составе кэш-команд, *ADSP*-2106x может одновременно выбирать инструкцию из кэш-памяти и выбирать 2 операнда из памяти. Генераторы адреса данных аппаратно реализуют циклические буферы данных. Контроллер микропрограмм поставляет адрес команды в программную память. Он управляет итерациями и оценивает выполнение результатов условных команд. Имея внутренний счетчик циклов и стек циклов, *ADSP*-2106x выполняет циклические операции с нулевыми непроизводительными затратами. При этом не требуется явных инструкций перехода для зацикливания или декрементации и проверки счетчика.

Высокое быстродействие *ADSP*-2106x достигается за счет конвейерного режима работы, предусматривающего выборку, декодирование и выполнение команды. Генераторы адресов обеспечивают формирование адреса памяти при передаче данных между памятью и регистрами. Пара генераторов адреса данных позволяет процессору выводить адреса для одновременного выполнения двух операций чтения или записи операндов. ГА-1 посылает 32-разрядный адрес в память данных, а ГА-2 - 24-разрядный адрес в память программ для доступа к данным программной памяти. Каждый из генераторов адресов обеспечивает управление 8-ю указателями адресов, 8-ю модификаторами адресов и 8-ю регистрами значений длины. Указатель, используемый для косвенной адресации, при необходимости модифицируется значением, расположенным в специальном регистре. Модификация может выполняться до или после осуществления доступа к памяти. С каждым указателем может быть связан определенный объем адресуемых данных , например, чтобы выполнять автоматическую адресацию кольцевых буферов. Каждый из регистров обоих генераторов адреса имеет регистр-дублер, который активизируется при осуществлении быстрого контекстного переключения. Кольцевые буферы обычно используются в цифровых фильтрах и преобразованиях Фурье.

Кэш-команд. Контроллер микропрограмм содержит кэш-команд на 32-слова. В кэш-память помещаются только те инструкции, которые конфликтуют при доступе к данным программной памяти. Это допускает полноскоростное функционирование ядра при циклическом выполнении многих операций.

Прерывания. В *ADSP* - 2106x используется 4 внешних аппаратных прерывания: 3 прерывания общего назначения (*IRQ*2-0) и специальное прерывания для сброса. Процессор также имеет внутренние генерируемые прерывания таймера, арифметических исключений, мультипроцессорные векторные прерывания и прерывания, определяемые программным обеспечением пользователя. При внешних прерываниях общего назначения и внутреннем прерывании по таймеру в стеке автоматически сохраняются арифметические состояния и содержимое регистра режима (*MODE*1). Допускаются прерывания с четырьмя уровнями вложения.

Таймер. Программируемый таймер обеспечивает периодическую генерацию прерываний. Если прерывания разрешены, таймер декрементирует 32-разрядный регистр счета на каждом тактовом цикле. При переходе регистра счета в нулевое состояние, *ADSP*-2106x генерирует прерывание, активизируя свой выход *TIMEXP*. Регистр счета автоматически перезагружается из 32-разрядного регистра периода и счет возобновляется.

Шины ядра процессора. Процессор имеет 4 шины: адреса памяти программ (ШАПП), адреса памяти данных (ШАПД), данных памяти программ (ШДПП) и данных памяти данных (ШДПД). В процессорах ADSP-2106x в памяти данных хранятся операнды данных, в то время как программная память используется для как для хранения инструкций, так и данных (например, коэффициентов фильтрации). Это позволяет осуществлять двойную выборку данных в случае, когда исполняемая инструкция находится в кэш-памяти.

Шина адреса программной памяти используются для пересылки адресов с целью осуществления выборки инструкций или данных. Шины данных программной памяти ШДПП и памяти данных ШДПД используются для передачи данных и кодов операций. ШАПП является 24-разрядной и позволяет осуществить выборку до 16М слов инструкций и данных. ШДПП - 48-разрядная и предназначена для пересылки инструкций разрядностью 48 бит. Данные формата с фиксированной запятой и данные формата с плавающей запятой одинарной точности выставляются на старших 32-х разрядах ШДПП. ШАПД является 32-разрядной шиной и позволяет осуществить выборку вплоть до 4 Г слов данных. Шина данных памяти данных ШДПД имеет 40 разрядных линий. Данные с фиксированной запятой и одинарной точностью выставляются на старших 32 разрядах этой шины. ШДПД обеспечивает пересылку содержимого любого регистра процессора в любой другой регистр или в любую область памяти данных в течение одного машинного цикла. Адреса памяти данных поступают от одного из двух источников: абсолютное значение, заданное в коде команды (прямая адресация) или с выхода генератора адреса данных (косвенная адресация).

Двухпортовое статическое ОЗУ (внутренняя память). На кристалле *ADSP*-21060 расположено ОЗУ емкостью 4 Мбита, организованное в виде двух независимых блоков по 2 Мбит каждый, которые могут быть конфигурированы как для хранения данных, так и кодов. *ADSP*-21062 содержит 2Мбитовое статическое ОЗУ, разделенное на блоки по 1 Мбит.

Каждый блок памяти является двухпортовым. Доступ к ОЗУ осуществляется в течение одного машинного цикла как для ядра процессора и процессора ввода/вывода, так и для контроллера ПДП. Двухпортовая память и раздельные внутрикристальные шины позволяют в течение одного машинного цикла одновременно пересылать данные из ядра и одного из портов ввода/вывода. В *ADSP*-21060 память может иметь две конфигурации: максимум 128К 32-разрядных слов данных или 80К 48-разрядных слов-инструкций (или 40-разрядных данных) с общим объемом до 4 Мбит. Содержимое памяти может быть выбрано словами по 16, 32 или 48 разрядов. Кроме этого, может быть обеспечен 16-разрядный формат хранения с плавающей запятой, при котором удваивается количество данных, которое может храниться на кристалле. Преобразование между 32-разрядным форматом с плавающей запятой и 16-разрядным с плавающей запятой осуществляется с помощью одной команды. Несмотря на то, что каждый блок может хранить комбинации кодов и данных, доступ к памяти наиболее эффективен только в случае, когда один блок хранит данные, используя для пересылок шину данных ПД, а другой - инструкции и данные, передавая их по шине данных ПП. Только при таком использовании шин, когда одна шина (ПД) выделена одному блоку, а другая (ПП) - другому, гарантируется выполнение двух пересылок данных в течение одного машинного цикла. В этом случае исполняемая инструкция должна находиться в кэш-памяти. Выполнение передачи данных в течение одного машинного цикла сохраняется также в том случае, когда один из операндов передается во внутреннюю память из внешнего устройства через внешний порт *ADSP*-2106x или в обратном направлении.

Внешние порты обеспечивают связь процессора *ADSP*-2106x с внекристальной памятью и внешними устройствами При этом в объединенное адресное пространство *ADSP*-2106x может быть включено до 4Г слов внекристального адресного пространства. Раздельные шины внутри микросхемы для адресов ПП, данных ПП, адресов ПД, данных ПД, адресов ввода/вывода и данных ввода/вывода - мультиплексируются в интерфейсном процессоре для создания внешней системной шины с одной 32-разрядной ША и одной 48-разрядной ШД. Внешнее статическое ОЗУ может быть 16-и, 32-х или 48-разрядным. При этом интегрированный в цифровой сигнальный процессор контроллер ПДП автоматически упаковывает внешние данные в слово подходящей ширины (48 разрядов для инструкций или 32 разряда для данных).

Адресация внешних устройств памяти упрощается за счет декодирования внутри самого кристалла старших разрядов линий адреса и генерации

сигнала выбора банка памяти. Для упрощения организации страничной адресации СОЗУ имеется возможность активизации отдельных управляющих линий процессора. В *ADSP*-2106x позволяет программировать состояния ожидания памяти и контроль подтверждения внешней памяти и внешних устройств.

Интерфейс главного *(host)* процессора позволяет осуществлять простую связь со стандартными микпроцессорными 16- или 32-разрядными шинами. Для этого требуются небольшое дополнительное аппаратное обеспечение. При этом поддерживается асинхронная передача со скоростью, соответствующей максимальной частоте тактовых импульсов ЦПОС. Интерфейс главного процессора доступен через внешний порт *ADSP*-2106x, отображаемый на память в объединенном адресном пространстве. Четыре канала ПДП поддерживают пересылку кодов и данных с минимальными программными затратами.

Микросхема располагает мощными средствами для работы в многопроцессорных системах цифровой обработки сигналов. Объединенное адресное пространство позволяет осуществить прямой доступ к памяти каждого используемого в многопроцессорной системе *ADSP*-2106x. В микросхему интегрированна логика шинного арбитража, которая обеспечивает управление системой, состоящей из одного ведущего и до 6 подчиненных *ADSP*-2106x. Смена ведущего процессора требует только одного непроизводственного цикла затрат. Шинный арбитраж может функционировать на основе фиксированного или циклического приоритета. Разрешается захват шин процессора для осуществления неделимых последовательностей чтение-изменение-запись, а также обеспечивается выполнение векторных прерываний для межпроцессорных команд. Максимальная скорость передачи для межпроцессорных пересылок данных через порты связи или внешний порт составляет 240 Мбит/с.

Процессор ввода-вывода содержит 2 последовательных порта, шесть 4-битовых линейных портов связи и контролер ПДП. Последовательные порты *ADSP*-2106x являются синхронными, которые обеспечивают интерфейс с различными типами внешних устройств. Последовательные порты функционируют с частотой тактирующих импульсов процессора, обеспечивая максимальную скорость передачи данных 40 Мбит/с. Тактирование портов может быть внутренним или внешним. Независимость процессов приема и передачи обеспечивает большую гибкость последовательной связи. Обмен через последовательные порты также может производиться в режиме ПДП.

Порты связи. В *ADSP*-2106x имеется шесть 4-разрядных портов связи, которые обеспечивают дополнительные возможности ввода/вывода. Их можно использовать в мультипроцессорных системах для межпроцессорного взаимодействия по двухточечной схеме *(point-to point)*.

Порты связи могут функционировать параллельно с максимальной скоростью передачи данных 240 Мбайт/с. Данные портов связи могут быть упа-

ковываны в 32-разрядные или 48-разрядные слова и непосредственно считаны ядром процессора или переданы во внутрикристальную память через ПДП. Каждый порт связи имеет свои собственные входные и выходные регистры синхронизации.

Контроллер ПДП. Внутрикристальный контроллер ПДП позволяет осуществлять пересылки данных без вмешательства процессора. Контроллер ПДП функционирует независимо от ядра процессора, выполняя операции передачи данных в то время, когда ядро исполняет программу. Код и данные могут быть загружены в *ADSP*-2106x с использованием режима передачи ПДП.

ПДП-передачи могут осуществляться между внутренней и внешней памятью *ADSP*-2106x, периферией и главным процессором, а также между внутренней памятью *ADSP*-2106x и его последовательными портами связи. Упаковка данных в слова по 16, 32 или 48 разрядов осуществляется автоматически во время ПДП-передачи.

Имеется десять каналов ПДП. Два из них доступны через линейные порты связи, четыре - через последовательные порты процессора и четыре через внешний порт. Внешние асинхронные устройства могут управлять двумя каналами ПДП, используя линии запроса/предоставления ПДП *(DMAR* $_{1\text{-}2}$ *DMAG* $_{1\text{-}2}$). Нумерация каналов ПДП приведена в табл.8.1.

Таблица 8.1 – Нумерация каналов ПДП

Номер канала ПДП	Буфер данных	Описание
0	RX0	Последовательный порт 0, прием
1	RX1	Последовательный порт 1, прием (или линейный буфер 0)
2	TX0	Последовательный порт 0, передача
3	TX1	Последовательный порт 1, передача (или линейный буфер 1)
4	LBUF2	Линейный буфер 2
5	LBUF3	Линейный буфер 3
6	ERB0	Внешний порт FIFO буфер 0 (или линейный буфер 4)
7	ERB1	Внешний порт FIFO буфер 1 (или линейный буфер 5)
8	ERB2	Внешний порт FIFO буфер 2
9	ERB3	Внешний порт FIFO буфер 3

Начальная загрузка процессора. Внутренняя память *ADSP*-2106x может быть загружена данными, поступающими с 8 разрядного ППЗУ, главного процессора или через один из портов связи. Выбор источника

загрузки контролируется по линиям *B, BMS, EBOOT*, *LBOOT*. Для начальной загрузки может быть использован как 16-ти, так и 32-разрядный хост-процессор.

На рис.8.3. изображена типовая схема однопроцессорной системы обработки. Обязательным, кроме *ADSP*, является наличие в системе только генератора тактовых импульсов (ГТИ), остальные блоки устанавливаются по мере необходимости.

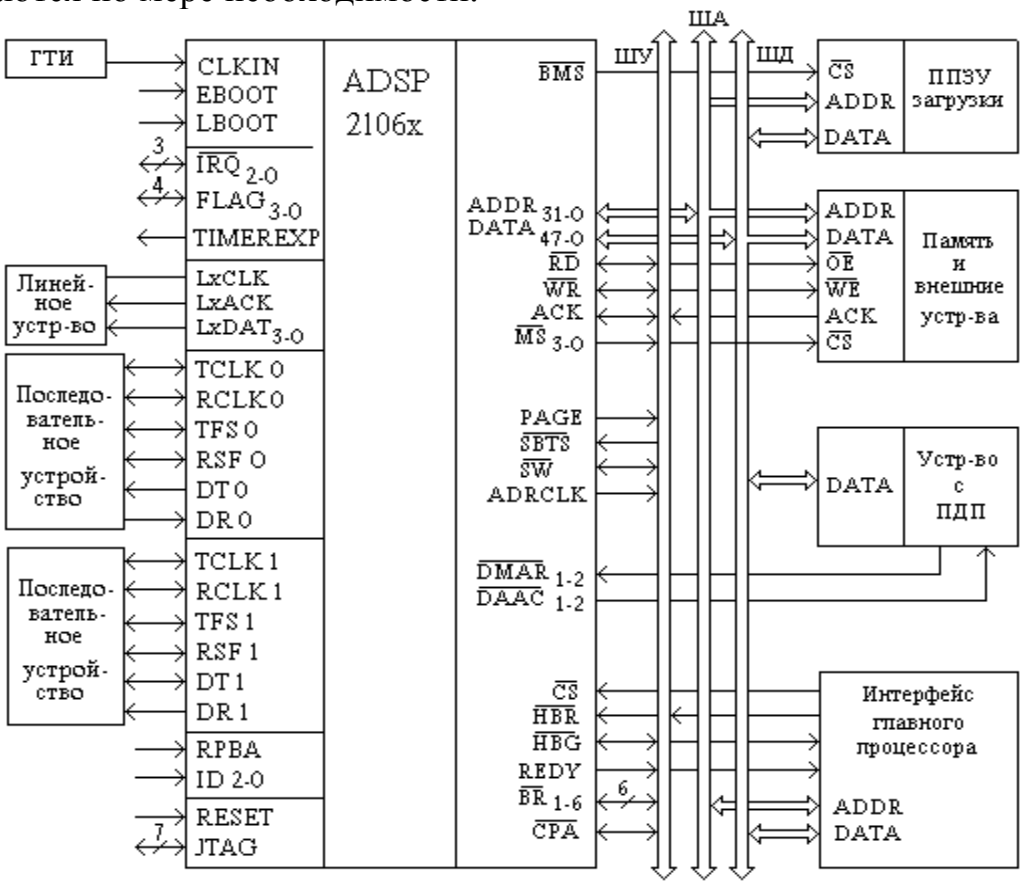


Рис.8.3. Типовая схема однопроцессорной системы

Все выводы процессоров *ADSP*-21060 и *ADSP*-21062 идентичны. Неиспользованные входы должны быть связаны с шиной питания или общим проводом, исключая *ADDR*(31-0), *DATA*(47-0), *FLAG*(3-0), $\overline{SW}$. Выводы процессора имеют следующее функциональное назначение.

ADDR (31-0) - внешняя шина адреса.

DATA (47-0) - внешняя шина данных.

$\overline{MS}$(3-0) - линии выбора памяти. Эти линии устанавливаются (низкий потенциал) при выборе микросхемы соответствующих банков внешней памяти. Они становятся активными при появлении сигнала доступа к внешней памяти.

$\overline{RD}$ - строб чтения памяти, в системе мультиобработки – выходной сигнал для главного *ADSP*-2106x и входной для всех других *ADSP*-2106x.

PAGE – сигнал конца страницы памяти.
ADRCLK - синхронизация адреса. Адрес действительный в момент переднего фронта этого сигнала.
$\overline{SW}$ - выбор синхронизации записи.
ASK - подтверждение памяти. Внешнее устройство может установить ACK, чтобы подключиться к очереди ждущих разрешения доступа к внешней памяти других устройств.
$\overline{SBTS}$ - переключение в третье состояние шин адреса, данных и стробов.
$\overline{IRQ}$(2-0) - линии запроса прерываний.
$\overline{FLAG}$(3-0) -флаговые выводы. Каждый вывод может быть определен как вход или выход. Если вывод определен как вход, то проверяется его состояние, если как выход, то он используется для взаимодействия с внешним устройствами.
TIMEXP - истечение времени счета таймера.
$\overline{HBR}$ - запрос системной шины главным процессором.
$\overline{HBG}$ - предоставление системной шины.
$\overline{CS}$ - выбор микросхемы.
REDY o/d - подтверждение системной шины, открытый выход стока (o/d) может быть запрограммирован на активный выход.
$\overline{DMAR}$1- запрос ПДП1 (Канал 7). ПДП2 – канал 8.
$\overline{DMAG}$1 - предоставление ПДП1.
BR(6-1) - запрос шины при мультиобработке.
ID(2-0) - идентификатор мультиобработки, определяет, какая из шин запроса ($\overline{BR}$1-6) используется процессором *ADSP*-2106x; *ID*=001 соответствует $\overline{BR}$1, *ID*=010 - $\overline{BR}$2 и т.д.
RPBA - выбор ротации приоритетности шин.
$\overline{CPA}$ - приоритетный доступ ядра процессора подчиненного *ADSP*.
DTx - последовательная передача данных.
Rx - прием последовательных данных.
TCLKx - синхронизация передачи.
RCLKx - синхронизация приема.
TFSx - синхронизация передачи кадра.
RFSx - синхронизация приема кадров.
LxDAT(3-0) - линейные порты данных.
LxCLK - синхронизация линейных портов.
LxACK - подтверждение линейного порта.
EBOOT - выбор загрузки ППЗУ.
LBOOT - выбор загрузки линии – загрузки главного процессора.
$\overline{BMS}$ - селекция загрузки памяти.
CLKIN - вход внешней синхронизации.
$\overline{RESET}$ - сброс процессора.
TCK - тест генератора тактовых импульсов (по стандарту *JTAG*).

TMS - выбор режима тестирования *(JTAG).*
TDI - вход тестовых данных *(JTAG).*
TDO - выход тестовых данных *(JTAG).*
$\overline{TEST}$ - сброс тестирования *(JTAG).*

8.2 Система команд и регистры *ADSP*-2106x

Набор команд ЦПОС семейства *ADSP*-21000 обеспечивает широкий выбор возможностей программирования. Многофункциональные команды позволяют осуществлять вычисления одновременно с передачей или модификацией данных, а также одновременные операции в умножителе и АЛУ. Каждая команда выполняется в течение одного машинного цикла. Развитая система адресации обеспечивает гибкость перемещения данных как внутри кристалла, так и вне его. Для облегчения кодирования и читаемости программ ассемблер процессора использует алгебраический синтаксис.

Команды цифрового сигнального процессора *SHARC* можно разделить на 4 группы:

- команды вычисления и передачи или модификации данных, которые специфицируют вычислительную операцию с параллельной передачей одного или двух полей данных или модификацией индексного регистра;

- команды, управляющие выполнением программы, специфицирующие разнообразные типы ветвлений, циклов, вызовов подпрограмм и возвратов;

- команды непосредственной пересылки данных, в которых поля инсрукций используются в качестве операндов, или непосредственно для адресации;

- прочие команды, в частности, проверка и модификация отдельных битовых полей, отсутствие операции и проч.

Многие команды содержат поля для специфических операций вычисления, которые используются в АЛУ, умножителе или в сдвигающем устройстве.

В процессоре имеется большая группа программно доступных регистров (табл.8.2). Они являются универсальными, т.е. позволяют как читать, так модифицировать их содержимое.

Все регистры цифрового сигнального процессора можно разделить на 6 групп:

- регистры общего назначения;
- регистры управления последовательностью команд;
- адресные регистры;
- регистры управления шинами;
- регистры таймера;
- системные регистры.

Таблица 8.2 – Программно доступные регистры

Обозначение регистра	Название и назначение регистра
1. *Регистры общего назначения (расположенные в файле регистров данных)*	
R15 – R0	регистры для операций с фиксированной точкой;
F15 – F0	регистры для операций с плавающей точкой.
2.*Регистры управления последовательностью команд*	
PC	Программный счетчик (ПС)
PCSTK	Верхушка стека
PCSTKP	Указатель стека
FADDR	Регистр адреса выборки
DADDR	Декодируемый адрес
LADDR	Адрес окончания цикла
CURLCNTR	Счетчик текущего цикла
LCNTR	Счетчик циклов для вложенных циклов
3. *Адресные регистры*	
I7 – I0	Индексные регистры ГА-1
M7 – M0	Регистры модификации ГА-1
L7 – L0	Регистры длины ГА-1
B7 – B0	Базовые регистры ГА-1
I15 – I8	Индексные регистры ГА-2
M15 – M8	Регистры модификации ГА-2
L15 – L8	Регистры длины ГА-2
B15 – B8	Базовые регистры ГА-2
4. *Регистра управления шинами*	
PX1	Обмен шин 1 ШДПП-ШДПД (16 бит)
PX2	Обмен шин 2 ШДПП-ШДПД (32 бита)
PX	48-битовая комбинация PX1 и PX2
5. *Регистры таймера*	
TPERIOD	Период таймера
TCOUNT	Счетчик таймера
6. *Системные регистры*	
MODE1	Управление режимом и состоянием 1
MODE2	Управление режимом и состоянием 2
IRPTL	Фиксатор прерываний
IMASK	Маскирование прерываний
IMASKP	Указатель маски прерываний (для вложенности)
ASTAT	Флаги арифметического состояния
STKY	Флаги состояния ошибок
USTAT1	Регистр 1 состояния пользователя
USTAT2	Регистр 2 состояния пользователя

Для локального хранения данных используются регистры общего назначения (РОН), которые по терминологии разработчика процессора носят название "файл регистров". Файл регистров состоит из шестнадцати 40- битовых основных и 16 альтернативных регистров. Номера реги-

стров обозначаются символом *R* при вычислениях с фиксированной запятой, и символом *F* – в операциях с плавающей запятой. АЛУ может выбирать два операнда *X* и *Y* из любого РОН. Результат операции помещается в любом из регистров файла регистров. В зависимости от результата операции устанавливаются соответствующие флаги регистра состояния АЛУ – *ASTAT*.

Команды АЛУ на языке ассемблера *ADSP*-2106x записываются следующим образом:

```
Rn=Rx+Ry ;     /* Суммирование содержимого регистров в формате */
                /* с фиксированной запятой */
Fn=Fx+Fy ;     /* Суммирование содержимого регистров в формате */
               /* с плавающей запятой */
Rn=(Rx+Ry)/2 ;        /* Определение среднего значения */
Rn=MIN(Rx,Ry) ;       /* Выделение минимального операнда */
Fn=RSQRTS Fx ;        /* Вычисляет  обратное значение квадратного корня */
                       /* из содержимого Fx в формате с плав.запятой
*/ Ra=Rx+Ry, Rs=Rx-Ry ; /*Одновременное сложение и вычитание  */
Fa=Fx+Fy, Fs=Fx-Fy ;     /*Rx, Fx, Ry, Fy – любые РОН регистры для*/
                      /* операций с фиксированной или плав.запятой */
```

Для выполнения операций умножения с фиксированной и плавающей запятой, а также операций умножения с накоплением используется аппаратный умножитель, который перемножает два входных операнда, обозначаемых *X* и *Y,* из файла регистров. Аккумулирование результата происходит в двух 80-разрядных регистрах *MRB* и *MRF* , допускающих контекстное переключение. Оба регистра имеют одинаковые структуру, состоящую из трех частей - *MR*0, *MR*1, *MR*2. Каждый из регистров может быть независимо считан или записан в файл регистров. Регистр *MR*2 имеет только 16 разрядов, но при чтении его содержимое знакорасширяется до 32 разрядов. В зависимости от результата умножения устанавливаются 4 флага регистра *ASTAT* и четыре флага регистра состояния ошибки *STKY*.

Вдобавок к вычислениям, выполняемым каждым вычислительным устройством отдельно, *ADSP*-2106x позволяет также осуществлять многофункциональные вычисления, при которых имеет место параллельное функционирование умножителя и АЛУ. Две операции выполняются тем же способом, как если бы они исполнялись в соответствующих однофункциональных вычислителях. Каждый из 4-х входных операндов многофункциональных команд ограничивается определенным местом расположения в файле регистров (рис.8.3). Во всех других операциях, входными операндами могут быть любые регистры файла регистров.

На языке ассемблера ADSP-2106x команды умножения и многофункциональные команды записываются следующим образом:

Rn = Rx * Ry
MRF= Rx * Ry
MRB= Rx * Ry
Fn = Fx * Fy
Rn = MRF + Rx * Ry
Rn = MRB - Rx * Ry
MRF= MRF - Rx * Ry .

Синтаксис команд прост и поэтому дополнительных пояснений не требуется. При выполнении однофункциональных команд в качестве регистров Rn, Rx или Ry может использоваться любой из регистров файла регистров, а при многофункциональных – только регистры в соответствии со схемой, изображенной на рис.8.3.

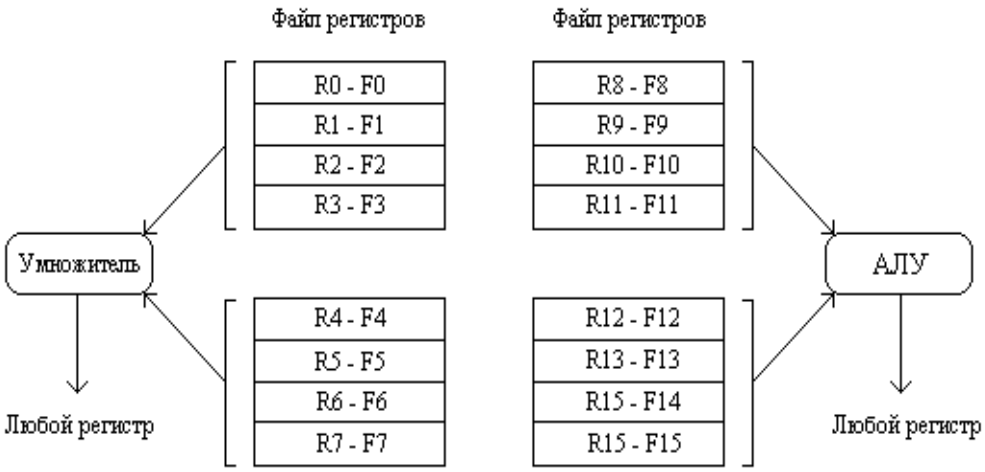


Рис.8.3. Схема закрепления регистров при многофункциональных операциях

Регистры управления последовательностью команд расположены в контроллере микропрограмм и являются универсальными т.е. доступ к ним осуществляется так же как и для других универсальных регистров. *ADSP*-2106x исполняет команды в течение трех тактов: выборки, декодирования и выполнения команды. За счет конвейеризации, пока происходит выборка текущей команды, осуществляется декодирование инструкции, выбранной в предыдущем цикле и выполнение команды, выбранной два такта назад. В связи с этим производительность сигнального процессора составляет одну команду за цикл. Любое ветвление программы может снизить производительность устройства.

Регистр адреса выборки *FADDR*, регистр адреса декодирования *DADDR* и программный счетчик *PC* содержат соответственно адреса ко-

манд, выбираемой, декодируемой и выполняемой в данный момент времени. Указатель стека программного счетчика *PCSTKP* представляет собой регистр, доступный по чтению и записи, который содержит адрес верхушки стека. При пустом стеке значение *PCSTKP*=0, и *PCSTKP*=1,2,...,30 при наличии в стеке данных. В случае, если *PCSTKP*=31 возникает переполнение стека.

Контроллер микропрограмм оперирует с двумя отдельными счетчиками циклов: текущего цикла (*CURLCNTR*), который отслеживает итерации текущего цикла, и обычного счетчика циклов *(LCNTR),* задающего количество повторений. Наличие двух счетчиков необходимо для поддержки вложенных циклов. Регистр *LADDR* хранит адрес верхушки стека адресов циклов. Он доступен по чтению и записи через ШД ПД. В случае, если стек адресов циклов пустой, он принимает значение 0xFFFFFFFF.

ADSP-2106x поддерживает программные циклы с инструкцией *DO UNTIL*. Эта команда вызывает повторение последовательности инструкций до тех пор, пока проверяемые условия не станут истинными *(true)*. Контроллер микропрограмм анализирует условия для определения момента прекращения выполнения цикла путем проверки битов регистров состояния *ASTAT*, управления режимом *MODE*1, счетчика циклов *CURLCNTR* и входных флагов. Каждое условие, которое оценивает *ADSP*-2106x, имеет ассемблерную мнемонику и уникальный код, который используется в коде операции. Для большинства условий, контроллер может проверять оба состояния: истину и ложь. Условие LCE (истечение счета счетчика циклов) чаще всего используется в инструкции DO UNTIL.

Инструкция DO UNTIL обеспечивает эффективный цикл без непроизводительных издержек (дополнительных команд перехода, проверки состояния или уменьшения счетчика). Например:

```
                LCNTR=30,       DO  label  UNTIL LCE;
                R0=DM(I0,M0),   F2=PM(I8,M8);
                R1=R0-R15;
        label:  F4=F2+F3;
```

В процессе выполнения команды DO UNTIL контроллер микропрограмм помещает в стек адреса цикла адрес последней команды цикла и условие завершения для выхода из него. Он также помещает в стек программного счетчика адрес верхушки цикла, который представляет собой адрес инструкции, непосредственно следующей за строкой DO UNTIL.

Адресные регистры располагаются в генераторах адреса ГА-1 и ГА-2. Оба генератора адреса имеют 4 группы регистров по 8 однотипных регистров каждой из групп: индексные *(I),* модификации *(M),* базы *(B)* и длины *(L)*. В 32-разрядном ГА-1 регистры нумеруются цифрами 0-7, а в 24-х разрядном ГА-2 они имеют номера 8-15. *I* – регистр выступает в ро-

ли указателя на память, а *M*-регистр содержит значение приращения указателя. Регистры *B* и *L* используются только для адресации кольцевых буферов данных. В частности, в регистре *B* располагается начальный (базовый) адрес кольцевого буфера, а в регистре *L* – длина буфера. Каждый из регистров ГА имеет альтернативный набор регистров, используемый при контекстном переключении.

Сдвигающее устройство. Устройство сдвига работает с 32-разрядными операндами с фиксированной точкой и выполняет следующие функции:

-сдвиги и циклические сдвиги ;

-операции манипулирования битами;

-манипулирование полями битов, включая выделение и копирование во внешнюю память;

-поддержка операций совместимости процессоров семейства *ADSP*-2100 (фиксированная/плавающая точка).

Сдвигающее устройство принимает от одного до трех операндов: *X*-операнд, над которым производится действие, *Y*-определяет величины сдвига s, длины полей битов или положение битов, *Z*-операнд над которым производится действие и корректировка. Устройство сдвига возвращает значение результата в один из регистров файла регистров.

Входные операнды выбираются из старших 32 разрядов регистра файла регистров (биты 39-8), или как непосредственное значение из поля инструкции. Операнды пересылаются в течение первой половины цикла. Результат передается в старшие 32 разряда регистра во время второй половины цикла, причем младшие 8 битов обнуляются. Таким образом, сдвигающее устройство может осуществлять чтение и запись одного и того же места из файла регистров в течение 1 машинного цикла. *X* и *Z* всегда 32-разрядные числа с фиксированной точкой. Вход *Y* - 32-разрядное значение с фиксированной точкой или 8-битовое поле.

Например, операция выделение и депонирование полей битов (команда FDEP) позволяет манипулировать с группами битов в пределах 32-разрядного слова фиксированной длины. Вход *Y* для этих инструкций определяет два 6-разрядных значения: *bit*6 и *len*6, расположенных в *Ry*. *Bit*6 - начальная позиция для выделения или депонирования, а *len*6 - длина поля битов, которая определяет количество выделяемых или депонируемых битов. Так при выполнении команды R0=FDEP R1 BY R2 (R1=0x000000FF00 и R2=0x0000021000) в регистр R0 будет помещен фрагмент содержимого регистра R1, состоящий из 8 битов, начиная от начала этого регистра (*len*6=8) и отстоящий от начала регистра результата R0 на 16 разрядов (*bit*6=16). Поскольку в данном случае обработка 32-разрядных слов производится в 40-разрядных регистрах, то отсчет битов в регистрах производится начиная с 8-го разряда (биты с 0 по 7 игно-

рируются и обнуляются). Т.е. в результате такой операции в регистре R0 окажется содержимое 0x00FF000000.

Регистры управления шинами. Для обеспечения возможности передачи информации между 48-разрядной ШД памяти программ и 32-разрядной шиной памяти данных используется *PX* регистр. Этот регистр состоит их двух регистров: 16-разрядного *PX*1 и 32-разрядного *PX*2. Они могут использоваться независимо или рассматриваться как составляющие регистра *PX*. Регистр *PX* позволяет осуществлять обмен данными по шине данных памяти программ или по ШД памяти данных с памятью или регистрами файла регистров. Так, например, чтобы записать 48-разрядное слово в ячейку памяти с именем *Port*1 по ШД ПП могут быть использованы следующие команды:

```
R0=0x9A00;        /* загрузка в R0 16-ти младших битов */
R1=0x12345678;    /* загрузка в R1 32-х старших битов   */
PX1=R0;
PX2=R1;
PM(Port1)=PX  .
```

Регистры таймера. Для управления таймером используются два регистра: *TPERIOD* и *TCOUNT*. Оба эти регистры доступны как для чтения, так и для записи. В первый из них заносится число периодов счета, которое после запуска таймера переносится в регистр счета *TCOUNT*. На каждом периоде счета таймер уменьшает содержимое *TCOUNT* на 1. Когда содержимое регистра счета становится равным нулю, таймер генерирует прерывание и выставляет на вывод *TIMEXP* логическую единицу длительностью 4 цикла тактовых импульсов (если работа таймера не запрещена установкой соответствующего флажка регистра режимов *MODE*2). Затем происходит автозагрузка числа периодов счета из *TPERIOD* в *TCOUNT*.

Системные регистры. Эти регистры служат для задания различных режимов работы процессора, управление операциями прерывания, индикации результатов логических и арифметических операций. Для управления режимом и состоянием процессора используются два регистра режима: *MODE*1 и *MODE*2. Путем установки и сброса определенных битов этих регистров можно изменять режимы адресации, переключать файл регистров на альтернативные регистры, управлять прерываниями, разрешать работу таймера, определять тип сигнального процессора и др.

Название и назначение битов регистра *MODE*1 приведено в табл.8.3. Регистр управления *MODE*2 используется для управления таймером, запрещает выход на внешнюю шину, блокирует кэш, содержит информацию о типе используемого процессора, а также ряд операций управления прерываниями.

Таблица 8.3 – Назначение битов регистра MODE1

№ бита	Обозначение бита	Назначение разрядов регистра *MODE*1
0	*BR8*	Инвертирование разрядов регистра I8 (ГА-2)
1	BR0	Инвертирование разрядов регистра I0 (ГА-1)
2	SRCU	Выбор альтернативных регистров вычислителя
3	SRD1H	Выбор альтернативных регистров ГА-1 (7-4)
4	SRD1L	Выбор альтернативных регистров ГА-1 (3-0)
5	SRD2H	Выбор альтернативных регистров ГА-2 (15-12)
6	SRD2L	Выбор альтернативных регистров ГА-2 (11-8)
7	SRRFH	Выбор альтернативных файловых регистров (R15-R8)
8-9	-	Зарезервированы
10	SRREL	Выбор альтернативных файловых регистров (7R-R0)
11	NESTM	Разрешение вложенности прерываний
12	IRPTEN	Разрешение глобальных прерываний
13	ALUSAT	Разрешение насыщения АЛУ
14	SSE	Разрешение знакового расширения короткого слова
15	TRUNC	1-усечение с плавающей точкой; 0-округление
16	RND32	1-округление с плавающей точкой до 32 бит; 0- до 40 бит
17-18	CSEL	00 – выбор основной шины
19-31	-	Зарезервированы

Регистр арифметического состояния *ASTAT* отображает результат выполнения операции не только в АЛУ, но и в умножителе и сдвигателе. Обозначение и назначение флагов состояния приведено в табл.8.4.

Регистр состояния ошибки индицирует дополнительные признаки состояния выполнения операций или уточняет причину установки одного или нескольких флагов регистра *ASTAT*. Флаги в регистре *STKY* проверяются после окончания серии операций. Если любой из флагов установлен, то некоторые из результатов ошибочны. Обозначение флагов и пояснение их значения приведены в табл.8.5.

8.3 Организация и адресация памяти

Процессор *ADSP*-21060 содержит внутрикристальное статическое ОЗУ емкостью 4 Мбит, разделенное на 2 независимых блока по 2 Мбит, в то время как *ADSP*-21062 имеет 2-Мбитовое СОЗУ по 1 Мбит в блоке. Оба эти блока нумеруются как блок 0 и блок 1. 2-Мбитовый блок памяти, содержащий 48-разрядные слова инструкций, может хранить максимум 40К слов, а такой же 2-Мбитовый блок, содержащий 32-разрядные слова данных, хранит до 64К слов. В процессоре также обеспечивается адресация до 4-х Гигаслов дополнительной, внекристальной, памяти через внешний порт. 32-разрядные слова используются для хранения данных формата *IEEE* с плавающей точкой одинарной точности; 48-разрядные слова применяются для хранения инструкции или 40-битовых данных с плавающей точкой расширенной точности. Независимые шины памяти

программ (ПП) и памяти данных (ПД) позволяют ядру процессора одновременно выбирать инструкции и данные из обоих блоков.

Все пространство памяти процессора разделяется на 3 секции: пространство внутренней памяти, пространство мультипроцессорной памяти и пространство внешней памяти. Пространство внутренней памяти *ADSP*-2106x состоит из внутрикристальной памяти и собственных ресурсов. Мультипроцессорное пространство памяти соответствует внутрикристальной памяти и ресурсам другого *ADSP*-2106x в мультипроцессорной системе. Пространство внешней памяти соответствует внешней памяти и устройствам ввода/вывода, отображенным на память. Для обращения к памяти в процессоре генерируются адреса, формат которых изображен на рис.8.4.

Таблица 8.4 – Назначение битов регистра ASTAT

№ бита	Обозначение флага	Индикация состояния (регистр *ASTAT*)
0	AZ	Содержимое АЛУ равно 0 или потеря значимости при операциях с плавающей точкой
1	AV	Переполнение АЛУ
2	AN	Содержимое АЛУ меньше 0
3	AC	Бит переноса АЛУ с фиксированной точкой
4	AS	Знак входа X АЛУ
5	AI	Недопустимая операция в режиме плавающей точки
6	MN	Знак произведения отрицательный
7	MV	Переполнение умножителя
8	MU	Потеря значимости умножителя при операциях с плавающей точкой
9	MI	Недопустимая операция умножителя при вычислениях с плавающей точкой
10	AF	Индикация операции АЛУ с плавающей точкой
11	SV	Переполнение сдвигателя
12	SZ	Содержимое сдвигателя равно нулю
13	SS	Знак операнда на входе сдвигателя
14-17		Зарезервировано
18	BTF	Флаг тестирования бита
19	FLG0	Значение флага 0
20	FLG1	Значение флага 1
21	FLG2	Значение флага 2
22	FLG3	Значение флага 3
23		Зарезервировано
24-31	CACC	Сравнение битов аккумулятора

Таблица 8.5 – Назначение битов регистра STKY

№ бита	Обозначение флага	Индикация состояния (регистр *STKY*)
0	AUS	Потеря значимости АЛУ с плавающей точкой
1	AVS	Переполнение АЛУ с плавающей точкой
2	AOS	Переполнение АЛУ с фиксированной точкой
3-4		Зарезервированы
5	AIS	Неверная операция в АЛУ с плавающей точкой
6	MOS	Переполнение умножителя с фиксированной точкой
7	MVS	Переполнение умножителя с плавающей точкой
8	MUS	Потеря значимости умножителе с плавающей точкой
9	MIS	Неверная операция в умножителе с плавающей точкой
10-16		Зарезервированы
17	CB7S	Переполнение циркулярного буфера 7 ГА-1
18	CB15S	Переполнение циркулярного буфера 15 ГА-2
19-20		Зарезервированы
21	PCFL	Переполнение указателя стека
22	PCEM	Указатель стека пустой
23	SSOV	Переполнение стека состояния (*MODE*1 и *ASTAT*)
24	SSEM	Состояние регистра стека пустое
25	LSOV	Переполнение стека циклов
26	LSEM	Стек цикла пустой
27-31		Зарезервированы

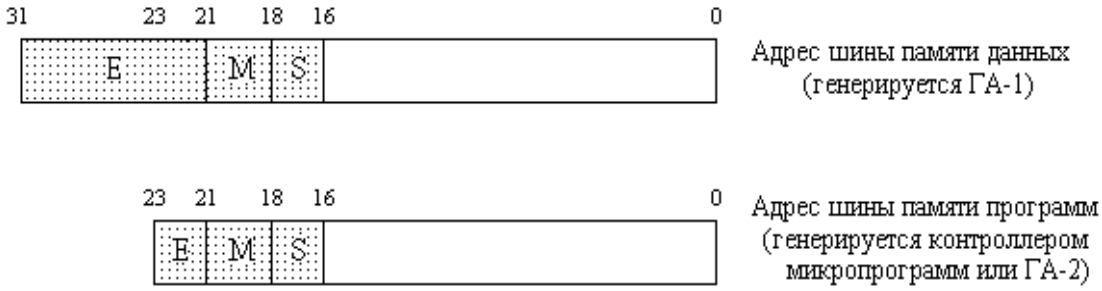


Рис.8.4 Формат адресов процессора *ADSP*-2106x

Часть разрядов адресного слова выделяется для идентификации используемой секции памяти: *E* – внешняя; *M* – мультипроцессорная и *S* – внутренняя память. Если поле *E* содержит все нули, то поля *S* и *M* становятся активными и используются для адресации либо собственной внутренней памяти (*M*=000) или для пересылки во внутреннюю память другого процессора, входящего в состав многопроцессорной системы. В последнем случае поле *M* идентифицирует номер другого процессора.

Секции памяти ограничены следующими адресами:

- внутренняя память от 0x00000000 до 0x0007FFFF;
- мультипроцессорная память от 0x00080000 до 0x003FFFFF;
- внешняя память от 0x00400000 до 0xFFFFFFFF.

Карта распределения памяти ADSP-2106x приведена на рис.8.5. Внутренняя память процессора имеет три области адресов:

- регистры процессора ввода/вывода 0x00000000 до 0x000000FF;
- адреса нормальных слов 0x00020000 до 0x0003FFFF;
(таблица векторов прерываний 0x00020000 до 0x0002007F)
- адреса коротких слов 0x00040000 до 0x0007FFFF.

Регистры процессора ввода/вывода представляют собой 256 регистров, отображенных на память, которые контролируют конфигурацию системы *ADSP*-2106x, а также различные операции ввода/вывода. Адресное пространство между регистрами ввода/вывода и адресами нормальных слов (ячейки 0x00000100 до 0x0001FFFF) зарезервировано и информация в них записываться не должна.

Блок памяти 0 начинается с начала пространства нормальных слов (с адреса 0x00020000), а блок 1 - с середины пространства нормальных слов (с адреса 0x00030000). Таким образом, для различных областей внутренней памяти используются следующие диапазоны адресов:

0x00000000-0x000000FF регистры ввода/вывода (IOP);
0x00000100-0x0001FFFF резервные адреса;
0x00020000-0x0002FFFF блок 0-адресация нормальных слов (32-р.,48-р.);
0x00030000-0x0003FFFF блок 1-адресация нормальных слов (32-р.,48-р.);
0x00040000-0x0005FFFF блок 0 - адресация коротких слов (16-р.);
0x00060000-0x0007FFFF блок 1 - адресация коротких слов(16-р.).

Пространство адресов нормальных и коротких слов в действительности представляет собой одну и ту же физическую память. Например, выборка слова с адресом 0x00020000 представляет те же самые ячейки, что и доступ к коротким словам с адресами 0x00040000 и 0x00040001 (для 32-разрядных данных в пространстве нормальных слов). Адресация коротких слов увеличивает количество 16-разрядных данных, которое может храниться во внутренней памяти. Она широко используется, в частности, в системах, производящих операции с матрицами.

Пространство мультипроцессорной памяти в мультипроцессорной системе отображается во внутреннюю память другого *ADSP*-2106x. Это позволяет каждому из *ADSP*-2106x обращаться ко внутренней памяти и регистрам процессора ввода/вывода, отображенными на память, другого процес-

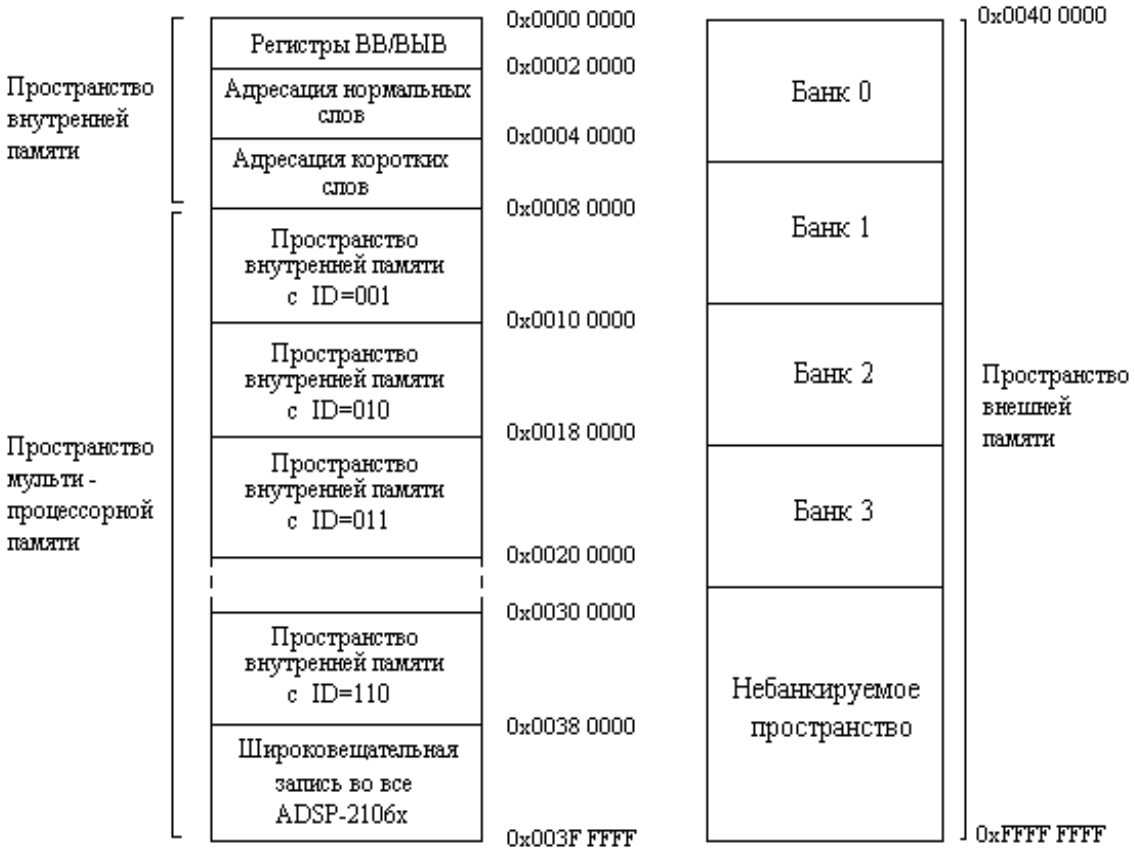


Рис.8.5. Карта распределения памяти ADSP-2106x

сора. Значение поля *M* адресной части определяет идентификационный номер (*ID*2-0) внешнего *ADSP*-2106x процессора, к которому осуществляется доступ, и поэтому, только этот процессор будет реагировать на цикл чтения/запись. Если поле *M*=111, то осуществляется циркулярная запись во все процессоры многопроцессорной системы.

Обращение к внешней памяти может быть осуществлено через внешний порт по шинам ПП, ПД и ВВ/ВЫВ. Этими шинами управляет ГА-1, контроллер микропрограмм (ГА-2) или процессор ввода/вывода. ГА-1 и процессор ввода/вывода выдают 32-разрядные адреса на ША ПД и ША ВВ/ВЫВ, при этом адресуется 4 Гигаслов памяти. Контроллер микропрограмм и ГА-2 генерируют 24-разрядные адреса по ША ПП, ограничивая адресацию младшими 12-ю Мегасловами памяти (0x00400000 - 0x00FFFFFF).

Каждый блок внутренней памяти может конфигурироваться для хранения либо 32-разрядных данных с одинарной точностью или 40-разрядных повышенной точности. Это конфигурирование инициируется установкой или очищением битов *IMDW*0 и *IMDW*1 в регистре системной конфигурации *SYSCON*.

В дополнение к внутрикристальной памяти, *ADSP*-2106x обеспечивает адресацию до 4 Гигаслов внешней памяти через внешний порт. Эта память может хранить как инструкции, так и данные. Внешнее адресное пространство включает в себя пространство мультипроцессорной памяти, внутреннюю память всех других *ADSP*-2106x, соединенных в многопроцессорную систему, а также пространство внешней памяти и область для стандартной адресации внекристальной памяти.

Адресация памяти. В сигнальном процессоре применяется непосредственная, относительная и косвенная регистровая адресация. В первом случае адрес ячейки памяти указывается непосредственно в поле операнда команды, например

dm (0x000047E0) = *astat* .

По этой команде значение *astat* заносится в ячейку памяти данных с адресом 000047E0*h*.

При относительной адресации исполнительный адрес вычисляется путем суммирования программного счетчика *PC* и смещения, задаваемого в поле команды. Например, call (pc,20) .

В случае использования косвенной регистровой адресации применяются регистры адресных генераторов ГА-1 и ГА-2. В команде указывается регистр, в котором содержится адрес требуемой ячейки памяти. При косвенной адресации возможны два режима работы:

а) предварительная модификация *(Pre-modify)*. Эффективный адрес определяется суммированием содержания индексного регистра I и регистра M или непосредственного операнда. При этом содержимое регистра I не изменяется;

б) формирование адреса с последующей модификацией индексного регистра *(Post-modify)*. ГА выдает на шину адреса содержимое индексного регистра I, а затем модифицирует его путем прибавления содержимого регистра M или непосредственного значения модификатора.

В ассемблере ADSP-2106x операции пре- и постмодификации различаются позициями индекса и модификатора (*M*-регистра или непосредственного значения) в инструкции. Расположение первым в поле команды *I*-регистра индицирует режим постмодификаци. Например,

*R*6=*PM(I*15,*M*12*)*.

Эта инструкция осуществляет обращение к ячейке программной памяти с адресом, находящемся в *I*15, затем значение *I*15+*M*12 записывается обратно в *I*15. Если же модификатор записан первым, это говорит об использовании предварительной модификации без операции обновления содержимого регистра *I*. Так по команде

R6=PM(M12,I15)

выбирается ячейка программной памяти с адресом *I*15+*M*12, но при этом содержимое *I*15 не изменяется. Изменение любого *I*-регистра производится в пределах одного ГА. Так команда

DM(M0,I2)=TPERIOD

записана правильно и позволяет выбрать ячейку памяти данных <M0+I2> , но команда DM(M0,I14)=TPERIOD – ошибочная, т.к. M и I принадлежат различным генераторам адреса.

Максимальное непосредственное значение, модифицирующее I-регистр, зависит от типа инструкции и от того, где размещается I регистр в ГА-1 или ГА-2 . При использовании ГА-1 максимальное непосредственное значение может быть равно 2^{32}, а при применении ГА-2 - 2^{24}. Некоторые инструкции с параллельными операциями разрешают только 6-разрядное значения модификатора. Так в приведенных ниже командах соответственно используются 32-разрядный и 6-разрядный модификаторы

R1=DM(0,x40000000,I1); DM *адрес*=I1+0x40000000;
F6=F1+F2,PM(I8,0x0B)=ASTAT; PM *адрес*=I8; I8=I8+0x0B.

8.4 Программирование процессоров семейства ADSP-21000*

При разработке микропроцессорных систем на базе процессоров семейства *ADSP*-21000 используется широкий спектр аппаратных и программных инструментов. Описания и назначения основных средств разработки представлены в табл.8.6.

Разработку программного обеспечения для семейства *ADSP*-21000 целесообразно выполнять в соответствии со следующей последовательностью :

1) описание архитектуры микропроцессорного комплекса в файле архитектуры системы ;
2) написание исходных программ на языке С или ассемблера, компиляция в объектный код;
3) соединение объектных модулей и файла архитектуры в один исполняемый файл;
4) отладка программы с использованием программного эмулятора или EZ-Lab® Evaluation Board;
5) отладка программы с использованием EZ-ICE® In-Circuit Emulator;

---

* Параграф написан Фендикевичем И.М.

6) подготовка для записи программы в ПЗУ и отладка программы на целевой системе.

Таблица 8.6 - Аппаратные и программные средства разработки ADSP21xxx

Аппаратные средства разработки	
EZ-Lab®Evaluation Board	Аппаратный модуль для оценки параметров разработанных программ
EZ-ICE® In-Circuit Emulator	Аппаратный эмулятор разрабатываемой микропроцессорной системы
Программные средства разработки	
ANSI C Compiler	Компилятор с языка С, поддерживающий индустриальный стандарт *GNU C* Compiler и расширение *Numerical C*, и RTL с функциями цифровой обработки сигналов.
CBUG® C Source-level Debugger	Символьный отладчик для программ, написанных на языке С. Поддерживается просмотр переменных и точи прерывания
Assembler/Linker	Компилятор с языка ассемблера, программа компоновки модулей в один исполняемый файл.
Simulator	Программа, эмулирующая функции процессора. Предназначена для отладки программного обеспечения без наличия цифрового сигнального процессора.
PROM Splitter	Преобразование полученной исполняемой программы, в вид пригодный для записи в ПЗУ.

При создании программ для процессоров обработки сигналов следует акцентировать внимание на сокращение времени выполнения того или иного участка кода. Также желательно максимально эффективно использовать все возможности архитектуры процессора.

В качестве примера рассмотрим разработку программы для двумерной фильтрации фрагмента изображения. В главе 5 были введены все необходимые формулы для выполнения этой операции. Как видно, основной операцией при выполнении фильтрации является свертка исходного фрагмента изображения X с прямоугольным окном H. Посредством подбора тех или иных весовых коэффициентов окна можно получить различные эффекты в обработанном фрагменте. Зададим размеры изображения: $M = 16$, $N = 32$. Будем использовать прямоугольное окно размером 3x3.

Создание программы начинается с описания файла архитектуры. В данном файле необходимо определить тип процессора и описать сегменты памяти для хранения данных и программы. Выберем тип процессора - ADSP21062. Для решения поставленной задачи потребуются следующие сегменты памяти:

- сегмент кода для обработчика прерывания RESET ;
- сегмент кода для программы ;

- сегмент данных для весовых коэффициентов;
- сегмент данных для исходного и обработанного изображения.

Первые три сегмента будут располагаться в памяти программ, а последний - в памяти данных. Размещение весовых коэффициентов в памяти программ даст возможность производить одновременный доступ и к исходному изображению и к этим коэффициентам. Ниже приведен полный текст полученного файла архитектуры.

```
.system ImageProcessing; название системы
.processor = ADSP21062; тип процессора
.segment /ram /begin=0x20004 /end=0x20008 /pm pm_rsti /width 48; для каждого сегмента :
.segment /ram /begin=0x20100 /end=0x20128 /pm pm_code /width 48; адрес начала, окончания,
.segment /ram /begin=0x23000 /end=0x23009 /pm pm_data /width 32; тип, название,
.segment /ram /begin=0x23010 /end=0x27000 /dm dm_data /width 32; ширина слова
.endsys;
```

Перейдем непосредственно к написанию программы фильтрации изображения. Сначала необходимо определить структуры данных для хранения исходных данных и результатов. В сегменте данных *PM_DATA* будет размещен массив весовых коэффициентов *h*, а в сегменте данных *DM_DATA* - массив исходного изображения *input* и полученного изображения *output*. В языке ассемблера предусмотрена возможность инициализации переменных значениями из внешнего файла. Подобным образом из файла «h.dat» на этапе компиляции инициализируется массив весовых коэффициентов. Размер массива *output* меньше размера массива *input*, поскольку на шаге *M*-2 в результате операции свертки с матрицей коэффициентов размером 3x3 будут обработаны все исходные данные.

```
.SEGMENT/DM dm_data; расположение сегмента в памяти данных и его название DM_DATA
.VAR        input[m*n]; исходное изображение размером M*N
.VAR        output[(m-2)*(n-2)]; обработанное изображение размером (M-2)*(N-2)
.ENDSEG; конец сегмента

.SEGMENT/PM pm_data; расположение сегмента в памяти программ и его название PM_DATA
.VAR        h[3*3]="h.dat"; массив весовых коэффициентов
.ENDSEG; конец сегмента
```

Прежде чем рассмотреть основную часть программы фильтрации, уделим внимание фрагменту кода обработчика прерывания *RESET*. Данные инструкции выполняются процессором после сброса и необходимы для начальной инициализации некоторых регистров и перехода на начало основной программы. В данном случае в обработчике прерывания *RESET* выполняется установка режимов ожидания внешней памяти (регистр *WAIT*) и переход на начало основной программы. Предполагается, что банки памяти 0, 1, 2, 3 и память без банков функционируют без дополнительных циклов ожидания и процессор будет игнорировать входной сигнал *ACK*.

```
.SEGMENT/PM pm_rsti; расположение сегмента в памяти программ и его название PM_RSTI
        NOP; нет операции
        USTAT2= 0x108421;  значения для регистра WAIT
        DM(WAIT)=USTAT2;  установка внешней памяти без циклов ожидания
        JUMP setup; безусловный переход (метка setup)
.ENDSEG; конец сегмента
```

В начале программы фильтрации необходимо произвести начальную установку регистров генераторов адреса. Поскольку исходное изображение располагается в памяти данных, то для его адресации будем использовать первый генератор адреса с регистрами *B0*, *I0*, *M0* и *L0*. Аналогичным образом для массива коэффициентов фильтра - генератор адреса с регистрами *B8*, *I8*, *M8* и *L8*.

Обработка изображения выполняется в двух вложенных циклах. Во внешнем цикле определяется начальная точка для вычислений и производятся изменения индексов массива, во внутреннем цикле происходят все основные вычисления. Во избежание дополнительного цикла чтения памяти программ, третий коэффициент фильтра до начала вычислений записывается в регистр *F5*. Регистр *F12* используется для хранения частичных произведений, а регистр *F8* - для хранения частичных сумм. Для сокращения количества повторений внутреннего цикла, запись частичной суммы происходит во второй инструкции, а не в конце цикла.

Ниже приведен текст программы фильтрации.

```
/* ---------------------------------------------------------------
CONV3x3.ASM  Двумерная свертка изображения с фильтром 3x3
---------------------------------------------------------------*/
#include "def21060.h"          /* Определение регистров IOP */
#define m 16 /* задание максимальных */
#define n 32  /* размеров изображения  */
.SEGMENT/DM dm_data;  /* сегмент данных в памяти данных */
.VAR            input[m*n]; /* массив, содержащий входное изображение */
.VAR            output[(m-2)*(n-2)]; /* массив, содержащий обработанное изображение
*/
.ENDSEG;
```

```
.SEGMENT/PM pm_data; /* сегмент данных в памяти программ */
.VAR                h[3*3]="h.dat"; /* массив коэффициентов фильтра */
.ENDSEG;

.SEGMENT/PM pm_rsti;            /* сегмент обработчика вектора прерывания RESET
*/
        NOP; /* нет операции */
        USTAT2= 0x108421;  /* Первая инструкция */
        DM(WAIT)=USTAT2;   /* Установка внешней памяти без циклов ожидания
*/
        JUMP setup;       /* Переход на начало программы */

.ENDSEG;

.SEGMENT/PM pm_code; /*сегмент кода программы */
setup:  M0=1; /* установка регистров для генераторов адреса */
      M1=n-2; /* памяти данных */
      M2=-(2*n+1);
      M8=1; /* и памяти программ */
      M9=2;
      B0=input;           L0=0;
      B8=h;               L8=@h;
      B9=output+(m-2)*(n-2); L9=@output;
      F5=PM(h+2); /* в регистре будет храниться третий коэффициент фильтра */
h_conv: F0=DM(I0,M0), F4=PM(I8,M8);
      LCNTR=m-2, DO in_row UNTIL LCE; /* внешний цикл по строкам
                                          исходного изображения */
         LCNTR=n-2, DO in_col UNTIL LCE; /* внутренний цикл  по
                                            столбцам исходного изображения */
  F8=F0*F4,  F8=F8+F12, F0=DM(I0,M0), F4=PM(I8,M9); /* в каждой команде
                                                       выполнение операций
*/
        F12=F0*F4,          F0=DM(I0,M1), PM(I9,M8)=F8;
        F12=F0*F5, F8=F8+F12, F0=DM(I0,M0), F4=PM(I8,M8); /* умножения, */
        F12=F0*F4, F8=F8+F12, F0=DM(I0,M0), F4=PM(I8,M8); /* накопления и */
        F12=F0*F4, F8=F8+F12, F0=DM(I0,M1), F4=PM(I8,M8); /* чтения новых */
        F12=F0*F4, F8=F8+F12, F0=DM(I0,M0), F4=PM(I8,M8); /* исходных данных
*/
        F12=F0*F4, F8=F8+F12, F0=DM(I0,M0), F4=PM(I8,M8); /* и коэффициентов
                                                             фильтра */
          F12=F0*F4, F8=F8+F12, F0=DM(I0,M2), F4=PM(I8,M8);
in_col:   F12=F0*F4, F8=F8+F12, F0=DM(I0,M0), F4=PM(I8,M8); /* окончание
                                                           внутреннего цикла */
          MODIFY(I0,M0); /* переход на новую строку */
in_row:   F0=DM(I0,M0); /* окончание внешнего цикла */
    RTS(DB), F8=F8+F12; /* отложенный выход их подпрограммы */
    PM(I9,M8)=F8; /* вычисление и запись последней частичной суммы */
    NOP;
.ENDSEG;
```

8.5 Многопроцессорные системы ADSP-2106x

Многочисленные приложения цифровой обработки сигналов (ЦОС) требуют очень высокой скорости вычислений, в которых даже самый быстродействующий сигнальный процессор не в состоянии в одиночку справиться с обработкой в реальном времени. В то же время многие алгоритмы ЦОС возможно разделить на отдельные задачи, которые, с целью повышения быстродействия, выполняются параллельно различными процессорами в единой мультипроцессорной системе.

При разработке процессоров семейства *ADSP*-2106x в его схему были введены функциональные особенности, которые позволяют использовать ЦПОС в многопроцессорных системах ЦОС. Эти особенности включают наличие встроенного арбитража доступа к ведущей шине и возможность обращения к ОЗУ и регистрам процессора ввода/вывода других *ADSP*-2106x.

В многопроцессорной системе с несколькими *ADSP*-2106x, функционирующими в режиме разделения внешней шины, любой из процессоров может стать ведущим и управлять системной шиной, которая состоит из 40-битовой ШД, 32-разрядной ША и группы управляющих линий. При этом ОЗУ и регистры процессора ввода/вывода (ПВВ) всех *ADSP*-2106x, входящих в систему, образуют пространство памяти многопроцессорной системы. Адреса внутреннего ОЗУ и регистры ПВВ каждого сигнального процессора отображаются на это пространство. Как только один из ЦПОС становится ведущим, он может непосредственно работать с ОЗУ и регистрами процессора ввода/вывода любого подчиненного *ADSP*-2106x, включая внешний порт *FIFO*-буферов данных. Ведущий процессор может записывать данные в регистры ПВВ для установки режима ПДП или генерации векторного прерывания.

В многопроцессорных системах обычно применяются две схемы связей между процессорами. В первой из них используется выделенный канал для обмена данными между ЦПОС, а во второй - единая разделяемая глобальная память, доступ к которой осуществляется через параллельную шину. В случае выделения канала обмена *ADSP*-2106x *SHARC* может быть соединен с другими процессорами через собственные линейные порты, либо подключаться к разделяемой параллельной шине (так называемое *кластерное соединение*).

Эффективность многопроцессорных систем во многом зависит от схемы обмена информацией между процессорами, которая определяет затраты на межпроцессорные связи и скорость передачи данных. Наличие в *ADSP*-2106x встроенных аппаратных средств позволяет строить мультипроцессорные системы на основе одной их трех топологий: потоковой, кластерной или матричной. Так для обработки больших межпроцессорных потоков данных больше всего подходит потоковая топология

(Data Flow Multiprocessing). Мультипроцессорные системы, построенные по схеме обработки потока данных, состоят из нескольких процессоров *SHARC*, соединенных последовательно с помощью линейных портов (рис.8.6). В случае использования такой топологии программист разделяет алгорим обработки на несколько процессов и пропускает данные последовательно по конвейеру. *ADSP*-2106x *SHARC* идеально подходит для систем такого типа, т.к. нет необходимости во внешней памяти и в буферах *FIFO* для связи между процессорами. Емкости встроенной памяти процессора оказывается вполне достаточно для размещения кодов команд и данных большинства прикладных задач. Кроме того, схема обработки проста в реализации и имеет невысокую стоимость. Недостаток этой топологии состоит в ограниченной гибкости системы.

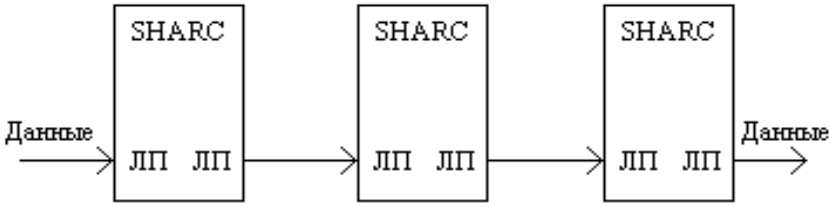


Рис.8.6. Потоковая микропроцессорная система

Кластерная топология *(Cluster Multiprocessing)* наиболее подходит для систем, от которых требуется максимальная гибкость, в частности, при поддержке нескольких задач, часть из которых может конкурировать. Кластерные многопроцессорные системы содержат несколько процессоров *SHARC*, соединенных параллельной шиной, которая позволяет осуществлять межпроцессорный доступ как к встроенному в кристалле ОЗУ, так и к разделяемой глобальной памяти (рис.8.7).

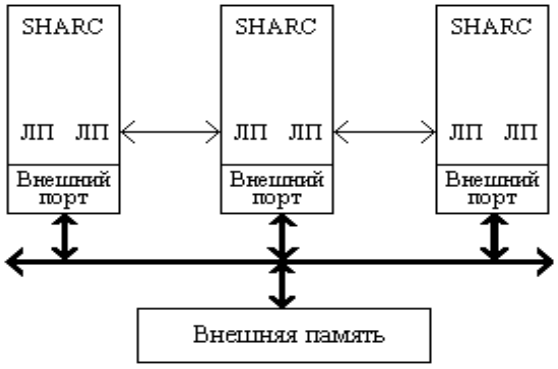


Рис.8.7. Кластерная многопроцессорная система

В типичном кластере (блоке) на процессорах *ADSP*-2106x может содержаться до шести процессоров и один главный (*хост-процессор*), осу-

ществляющий арбитраж шины. Встроенная в кристалл логика позволяет сигнальным процессорам разделять общую шину без дополнительной аппаратной части. Запрос шины генерируется автоматически каждый раз, когда процессор обращается к внешнему адресу. Как только *ADSP-2106x* становится ведущим, он может получить доступ не только к внешней, но и внутренней памяти и регистрам других процессоров. Ведущий процессор напрямую перемещает данные в другой процессор или инициализирует для этого канал ПДП. Кластерная конфигурация позволяет *SHARC*-процессорам работать на максимальной скорости межпроцессорного обмена.

Матричная топология представляет собой несколько процессоров, соединенных в двух- или трехмерную сеть. Такая топология может оказаться весьма эффективной в ряде приложений (система отображения данных в радио- и гидролокаторах и пр.). На рис.8.8 изображена двухмерная матрица сигнальных процессоров. Линейные порты обеспечивают связь с соседними процессорами и маршрутизацию данных. Один ведущий процессор обеспечивает поток команд, который выполняет массив.

Несколько сигнальных процессоров *ADSP*-2106x могут совместно использовать внешнюю шину без дополнительной схемной части для арбитража шин. Логика арбитража, встроенная в микросхему, позволяет соединить до шести *ADSP*-2106x и один главный *(host)* компьютер. Электрическая схема многопроцессорной системы изображена на рис.8.9. В этой схеме интерфейс главного процессора, глобальная память и ППЗУ загрузки являются не обязательными .

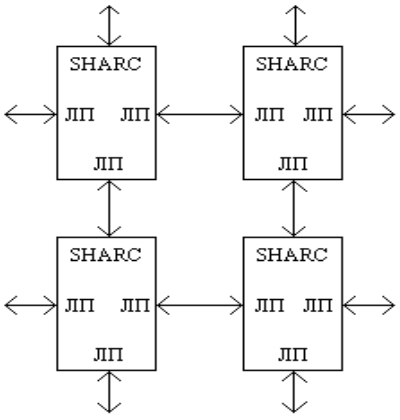


Рис.8.8. Матричная топология микропроцессорной системы

Арбитраж шины осуществляется с использованием сигналов $BR_{1\text{-}6\_}$, *HBR* и *HBG*. Линии BR_1-BR_6 выполняют арбитраж между несколькими *ADSP*-2106x, а *HBR* и *HBG* используются для передачи управляющих сигналов от ведущего *ADSP*-2106x соответственно к главному процессору и обратно. Вид арбитража определяется уровнем сигнала на выводе

RPBA. Низкий уровень соответствует фиксированному приоритету каждого из *ADSP*-2106x, а высокий – циклической ротации приоритета процессоров.

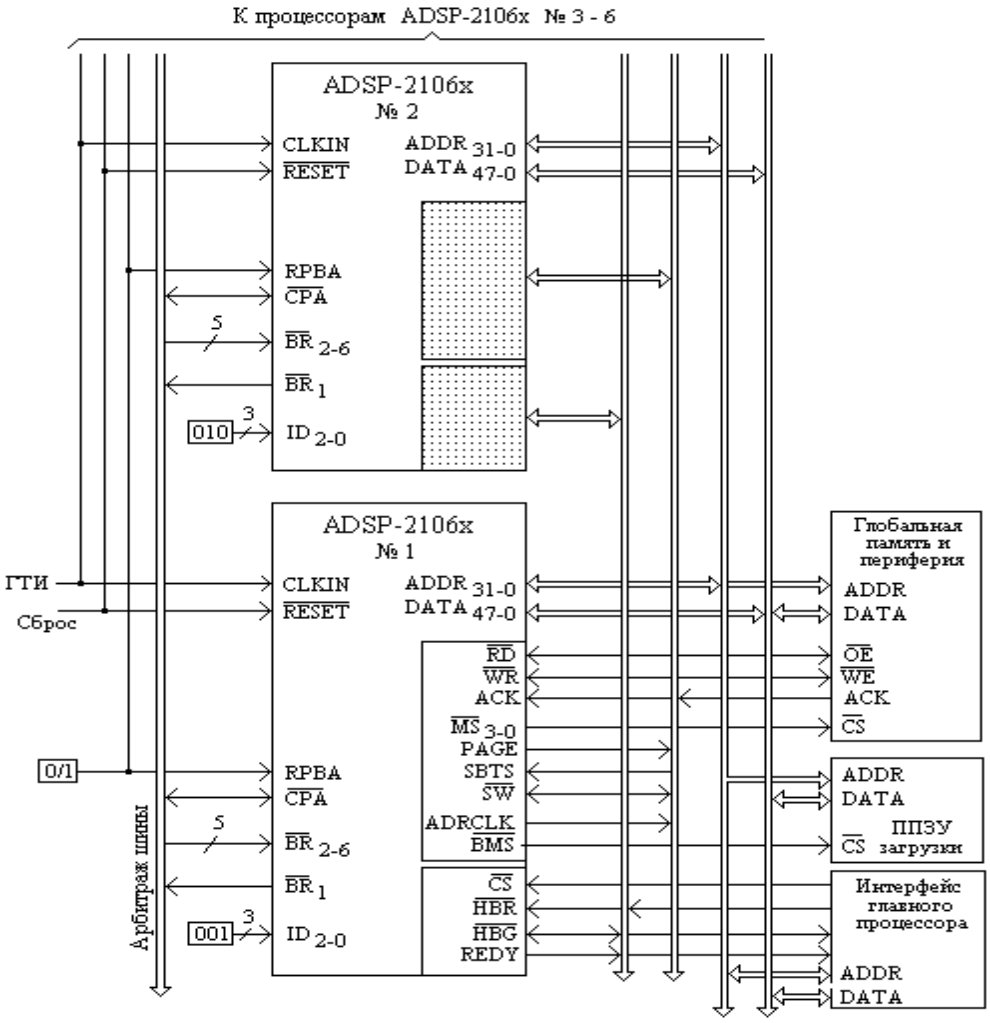


Рис.8.9. Схема многопроцессорной системы на базе ЦПОС ADSP-2106x

Уровни напряжений на выводах $ID_{2\text{-}0}$ обеспечивают идентификацию каждого процессора в многопроцессорной системе. Первый процессор должен иметь идентификационный номер 001, второй 010 и т.д. (см.рис.8.9). Один из процессоров должен обязательно иметь номер 001 для того, чтобы схема синхронизации шины работала правильно. Когда на входах *ID* процессоров установлены уровни 001-110, то ЦПОС самостоятельно конфигурируются для многопроцессорной системы и отображают свою встроенную память и регистры ввода/вывода на пространство памяти многопроцессорной системы. *ID*=000 конфигурирует данный *ADSP*-2106x для работы в однопроцессорной системе. Состояние *ID*=111 зарезервировано и использованию не подлежит. В схеме с фиксированными приоритетами высший приоритет имеет процессор с меньшим *ID*. Каждый из процессоров многопроцессорной системы пу-

тем анализа битов *CRBM*(2-0) регистра *SYSTAT* может определить, какой из ЦПОС является ведущим в системе. Эти биты показывают номер ведущего процессора в текущий момент времени.

Используя команды проверки выполнения условия (*IF condition compute*), можно программно определить, является ли данный процессор ведущим *(Bus Master)* или подчиненным. Мнемоника ассемблера по определению ведущего процессора - *BM* (либо *NBM* – отрицание ведущего). Каждый раз, когда ведущий ЦПОС прекращает использовать шину, он снимает сигнал BR_i , позволяя другому *ADSP*-2106x запросить управление системной шиной.

В большинстве многопроцессорных систем желательно ограничить время занятия системной шины только одним процессором. Для этого программным способом загружают регистр *BMAX* максимальным количеством циклов минус 2, после которых процессор теряет статус ведущего. Когда в результате декремента это число станет равным нулю, текущий ведущий процессор убирает свой сигнал BR_i , что позволяет произвести передачу управления системной шиной другому процессору. Цикл, в котором ведущий процессор передает управление шиной другому сигнальному процессору, называется *циклом перехода шины*. Он происходит в момент, когда текущий управляющий процессор убирает активный уровень со своего контакта BR_i, а один из подчиненных процессоров устанавливает активный уровень на аналогичном выводе. Для приобретения статуса ведущего подчиненный процессор ждет цикла перехода шины. Если запросивший процессор имеет наивысший приоритет в этот момент, то он станет ведущим в следующем цикле. В противном случае он продолжает ждать. В связи с тем, что *ADSP*-2106x в процессе функционирования наблюдает за всеми линиями BR_i многопроцессорной системы, каждый ЦПОС может определить, когда наступает цикл перехода шины и какой процессор стал ведущим после его завершения.

В многопроцессорной системе ведущий процессор может осуществить непосредственный доступ к внутренней памяти и регистрам ввода/вывода других процессоров. Он производит чтение или запись ячеек памяти, входящих в пространство памяти мультипроцессорной системы. Такой режим доступа получил название *прямая запись* или *прямое чтение*. Когда происходит прямая запись в подчиненный *ADSP*-2106x данные и адреса сохраняются в четырехуровневом *FIFO*-буфере процессора ввода/вывода. В случае переполнения буфера подчиненный процессор снимает сигнал подтверждения *ACK* на время, пока буфер не освободится. Такой доступ невидим для подчиненных процессоров, т.к. он осуществляется через внешний порт и внутрикристальную шину процессора ввода/вывода. Это позволяет подчиненному процессору продолжать выполнение программы без перерывов. В процессе прямого чтения подчиненного процессора адрес ячейки памяти фиксируется процессором вво-

да/вывода, при этом происходит сброс сигнала *ACK*. После завершения процедуры чтения этот сигнал снова устанавливается в активное состояние. Т.о. прямое чтение не конвейеризируется. В связи с этим прямое чтение не является самым эффективным способом перемещения данных из подчиненного процессора.

Для одновременного перемещения данных во все процессоры многопроцессорной системы используется *широковещательная* запись. Ведущий процессор может производить широковещательную запись в одинаковые области памяти или регистры процессора ввода/вывода во все подчиненные процессоры и в себя самого. Такой вид записи используется для одновременной загрузки данных и программ на несколько процессоров.

Наиболее эффективным способом перемещения данных между ведущим и подчиненным процессорами является прямой доступ в память (ПДП). Ведущий процессор может инициировать операции ПДП путем записи управляющих слов в регистры параметров и управления ПДП, а также установить внешний порт каналов ПДП для пересылки блоков данных как из, так и во внутреннюю память подчиненного процессора. После того, как канал ПДП установлен, ведущий ЦПОС может читать (или писать) из соответствующего буфера подчиненного процессора или запрограммировать для этого свой собственный контроллер ПДП. Ведущий процессор может установить внешний порт ПДП для перемещения данных во внешнюю память, используя линии запроса и подтверждения ПДП.

Для разделения вычислительных ресурсов и синхронизации в многопроцессорной системе используются семафоры. Так как обе памяти, внешняя и встроенная, в *ADSP*-2106x доступны со стороны любого процессора, то семафоры располагаются практически где угодно. На практике для семафоров чаще всего используются регистры общего назначения процессора ввода/вывода MSGR0-MSGR7. Используя возможность блокировки шины *ADSP*-2106x может читать и модифицировать семафор в течение одной команды.

Для передачи управления в многопроцессорной системе *ADSP*-2106x используются векторные прерывания. Ведущий процессор выдает вектор прерывания подчиненному ЦПОС путем записи адреса процедуры обслуживания прерывания в регистр *VIRPT* подчиненного процессора. Это обуславливает высокоприоритетное прерывание, которое переключает подчиненный процессор на выполнение требуемой процедуры. Младшие 24 бита вектора содержат адрес подпрограммы, а старшие 8 бит могут быть дополнительно использованы в качестве данных для чтения процедурой обслуживания прерывания.

БИБЛИОГРАФИЯ

REFERENCES

1. Баскаков С.И. Радиотехнические цепи и сигналы.-М.: Высшая школа, 1983.-536с.
2. Бендат Дж., Пирсол А. Применения корреляционного и спектрального анализа.-М.:Мир,1983.-312с.
3. Вайдьянатхан П.П. Цифровые фильтры, блоки фильтров и полифазные цепи с многочастотной дискретизацией. Методический обзор// ТИИЭР.-1990.-N3.-С.77-120.
4. Гольденберг Л.М., Матюшкин Б.Д., Поляк М.Н. Цифровая обработка сигналов: Справочник.-М.:Радио и Связь,1985.-312с.
5. Гоноровский И.С. Радиотехнические цепи и сигналы.-М.:Сов.радио,1977.-608с.
6. Гутников В.С. Фильтрация измерительных сигналов.-Л.: Энергоатомиздат, Ленингр. отд-ние, 1990.-192с.
7. Даджион Д., Марсеро Р. Цифровая обработка многомерных сигналов.-М.:Мир,1988.-488с.
8. Зубарев Ю.Б. , Глориозов Г.Л. Передача изображений.-М.: Радио и связь,1989.-332с.
9. Иконика. Цифровая обработка видеоинформации.-М.:Наука,1989.-186с.
10. Каппелини В., Костантенидис А. Дж., Эмилиани П. Цифровые фильтры и их применение.-М.: Энергоатомиздат ,1983.-360с.
11. Марпл С.Л. Цифровой спектральный анализ и его приложения.-М.:Мир,1990.-584с.
12. Методы передачи изображений. Сокращение избыточности/ Под ред. У.К. Прэтта.-М.:Радио и связь,1983.-264с.
13. Мошиц Г., Хорн П. Проектирование активных фильтров: Пер. с англ.-М.: Мир, 1984.-320с.
14. Назаров М.В. Прохоров Ю.Н. Методы цифровой обработки и передачи речевых сигналов.-М.:Радио и связь,1985.-176с.
15. Применение цифровой обработки сигналов / Под ред. Э.Оппенгейма.-М.: Мир,1980.-550с.
16. Рабинер Л., Гоулд Б. Теория и применение цифровой обработки сигналов.-М.:Мир,1978.-848с.

17. Рабинер Л.Р., Шафер Р.В. Цифровая обработка речевых сигналов.-М.: Радио и связь, 1981.-495с.
18. Радиотехнические цепи и сигналы/ Д.В. Васильев, М.Р. Витоль, Ю.Н. Горщенков и др.; Под ред. К.А.Самойло .- М.:Радио и связь,1982.-528с.
19. Справочник по расчету и проектированию ARC-схем/ С.А.Букашкин, В.П.Власов, Б.Ф.Змий и др.; Под ред. А.А.Ланнэ.-М.:Радио и связь,1984.-368с.
20. Цифровая обработка телевизионных и компьютерных изображений / А.В.Дворкович, В.П.Дворкович, Ю.Б.Зубарев и др.; Под ред. Ю.Б. Зубарева и В.П. Дворковича.-М.: Международный центр научной и технической информации,1997.-212с.
21. Цифровые фильтры в электросвязи и радиотехнике/ А.В.Брунченко, Ю.Г.Бутыльский, Л.М.Гольденберг и др.; Под ред. Л.М. Гольденберга .- М.:Радио и связь, 1982.-224с.
22. Хэмминг Р.В. Теория кодирования и теория информации.-М.:Радио и связь,1983.-176с.
23. Чернега В.С. Сжатие информации в компьютерных сетях.-Севастополь: Учеб.пособие для вузов; Под ред. В.К. Маригодова.-Изд-во СевГТУ,1997.-214с.
24. Яншин В. Обработка изображений на языке Си для IBM PC: Алгоритмы и программы.-М.:Мир,1994.-240с.
25. ADSP-2100 Family DSP Microcoputers .-Norwood:Analog Devices,1994.
26. ADSP-2106x SHARC User's Manual .-Norwood:Analog Devices,1995.
27. Baudendistel K. An Improved Method of Scaling for Real-Time Signal Processing Applications// IEEE Trans. of Education .-1994.-V.37.-N-3.-P.281-288.
28. CCITT G.726 General aspects of digital transmission systems; Terminal equepments. 40,32,24,16 K/s Adaptive differential pulse code modulation (ADPCM) Recomendation G.726.-Geneva,1990.- 57p.
29. Klette R., Zamperoni P. Handbuch der Operatoren fuer die Bildbearbeitung.-Braunschweig: Vieweg, 1992. – 303 s.
30. Kroon P., Deprettere E.F. A class of analysis-by-synthesis predictive coders for high quality speech coding at rates between 4.8 and 16 kbits/s // IEEE J. Select. Areas Commun.-1988.-V.6.-N2.-P. 353-363.
31. Lasley P., Bier J., Shoman A.,Lee E. DSP Processor Fundamentals.-New York: IEEE Press,1997.-210p.
32. Nelson M. Daten Komprimierung. Effiziente Algorithmen in C.-Hannover: Verlag Heinz Heise,1993.-475p.
33. Noll P. MPEG Digital Audio Coding// IEEE Signal Processing Magazine.-1997.-V.14.-N9.-P.59-81.
34. Sayood Khalid . Introduction to Data Compression.- San Francisco:Morgan Kaufmann Publishers Inc.,1995.-453p.

35. Second-Generation TMS320 User's Guide.-Avril:TI,1990.
36. Spanias A. Speech coding: A tutorial review // Proc. IEEE.-1994.-V.82.-P.1541-1582.
37. Wallace G.K. The JPEG Still Picture Compression Standard // Communication in ACM.-1991.-V.34.-N4.-P.31-44.

СОДЕРЖАНИЕ

Учебное издание
Бондарев Владимир Николаевич,
Трёстер Герхард,
Чернега Виктор Степанович

Цифровая обработка сигналов: Методы и средства

Bondarev V.N., Troester G., Chernega V.S.
Digital signal processing: Methods and Insruments

Полписано в печать 15.07.2001г. Формат 60x90/16.
Гарнитура Таймс. Печать офсетня. Бумага газетная.
Усл.п.л. 25. Тираж 550 экз.
Харьков, Издательство «Конус»

Севастполь, Издательство СевГТУ, Студгородок, НМЦ, 1999г.