УЗБЕКСКОЕ АГЕНТСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ
ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

На правах рукописи

Бурганов Ильмир Ильхамович

ИССЛЕДОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ И УСТРОЙСТВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Специальность: 5А521907 – Прикладная информатика

На соискание академической степени магистра

Работа рассмотрена
и допускается к защите
зав. кафедрой «Компьютерные системы»
профессор Мусаевым М.М.

_____

« ___ » _____ 2009 г.

Научный руководитель
к.т.н., доц. Каххаров А.А.

_____

Ташкент – 2009

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Преобразование и обработка сигналов осуществляется в системах. Понятия сигнала и системы неразрывны, так как любой сигнал существует в какой-либо системе его обращения. Система обработки сигналов может быть реализована как в материальной форме (специальное устройство, измерительный прибор и т.п.), так и программно на ЭВМ или на любом другом вычислительном устройстве. Существуют и комплексные измерительно-вычислительные системы (ИВС), которые выполняют как регистрацию и первичную обработку сигналов непосредственно в материальной форме их представления, так и преобразование сигналов в цифровую форму, и последующую программную обработку. Форма реализации систем существенного значения не имеет и определяет только их возможности при анализе и обработке сигналов.

Успешное воплощение перспектив развития инфокоммуникационных технологий во многом базируется на достижениях цифровой обработки сигналов (ЦОС), призванной решать задачи приема, формирования, обработки и передачи информации в реальном масштабе времени [3]. Осуществление сложных алгоритмов ЦОС требует, в свою очередь, применения эффективных базовых алгоритмов ЦОС (фильтрации, спектрального анализа и синтеза сигналов), экономично использующих соответствующие технические ресурсы.

Основная научная проблематика в области ЦОС заключена в разработке путей преодоления ограничений обусловленных имеющимися ресурсами: возможностями элементной базы, допустимой величиной программно-аппаратных затрат. Методы проектирования инструментальных средств ЦОС, объединяющие синтез в спектральной области по заданным величинам рабочих параметров с приемами, учитывающими эти ограничения, позволяют получить решения, близкие к оптимальным в смысле минимизации результирующих затрат.

Задача исследования эффективных алгоритмов и устройств цифровой фильтрации и синтеза сигналов, базирующаяся на последних достижениях теории цифровой обработки сигналов, является весьма актуальной, тем более что накопленный опыт разработки и использования цифровых сигнальных процессоров стимулируют создание новых более совершенных и мощных типов этих процессоров, в архитектуре которых должны быть заложены возможности воплощения эффективных алгоритмов ЦОС [5].

Таким образом, в настоящее время существует **актуальная научно-техническая проблема** совершенствования алгоритмов и устройств ЦОС.

Цель настоящей диссертационной работы – анализ линейных систем и сигналов, повышение эффективности линейных алгоритмов и устройств ЦОС путем разработки программных и аппаратных средств.

Задачи исследования

1. Исследование свойств цифровых фильтров и характеристик алгоритмов ЦОС.
2. Разработка методов и совершенствования алгоритмов и устройств ЦОС, определение условий целесообразности их использования.
3. Создание методики проектирования алгоритмов и устройств ЦОС с наименьшей программно-аппаратной затратой.
4. Разработка и реализация методик анализа программного обеспечения и инструментальных средств ЦОС.

Научная новизна:

1. Анализированы линейные методы обработки сигналов.
2. Тестированы различные типы сигнальных процессоров по производительность, время выполнение задач и объема памяти.
3. Разработана методика расчета, инструментальных средства и программного обеспечение для проектирования алгоритмов и устройств ЦОС, удовлетворяющих критери минимума показателей сложности вычислений.

Практическая ценность

Разработанные программные продукты обеспечивают создание аппаратных средств ЦОС повышенной эффективности, оптимизированных по критерию минимума показателя вычислительной сложности.

Апробация работы

Основные положения диссертационной работы докладывались и одобрены на научно-технических конференциях: Научно-практическая конференция аспирантов, магистров и одаренных студентов, г. Ташкент, ТУИТ, 2008-2009г.

Публикации. По результатам выполненных исследований опубликовано 2 тезисов докладов в трудах научно-практической конференции.

Диссертационная работа состоит из введения, четырех глав с выводами и составляет 99 стр, литературы из более 34 наименований, включает 42 рисунков.

ГЛАВА 1. АНАЛИЗ ЛИНЕЙНЫХ СИСТЕМ И СИГНАЛОВ

1.1. Исследование свойств линейных систем

Безотносительно к назначению и исполнению система всегда имеет *вход*, на который подается входной сигнал или входное воздействие, в общем случае многомерное, и *выход*, с которого снимается обработанный выходной сигнал. Если устройство системы и внутренние операции преобразований принципиального значения не имеют, то система в целом может восприниматься как “черный ящик”, в формализованном виде. Формализованная система представляет собой определенный *системный оператор* (алгоритм) преобразования входного сигнала – *воздействия* s(t), в сигнал на выходе системы y(t) – *отклик* или *выходную реакцию* системы. Символическое обозначение операции преобразования (трансформации):

$$y(t) = T[s(t)].$$

Системный оператор T - это правило (набор правил, алгоритм) преобразования сигнала s(t) в сигнал y(t). Для общеизвестных операций преобразования сигналов применяются также расширенные символы операторов трансформации, где вторым символом и специальными индексами обозначается конкретный вид операции (как, например, TF - преобразование Фурье, TF^{-1} - обратное преобразование Фурье) [1, 2, 9].

Входной сигнал системы может представлять собой m - мерный вектор (m входных сигналов), а выходной сигнал n - мерный вектор, при этом система будет иметь m входов и n выходов. Пример такой системы в геофизике: трехканальный гамма-спектрометр, на три входа решающего блока которого поступают сигналы от калиевого, радиевого и ториевого каналов спектрометра, а на три выхода выводятся сигналы содержаний калия, урана и тория, при этом системный оператор реализует алгоритм решения системы трех линейных уравнений с тремя неизвестными.

Для детерминированных входных сигналов соотношение между выходными и входными сигналами однозначно задается системным

оператором. В случае реализации на входе системы случайного входного процесса также существует однозначное соответствие процессов на выходе и входе системы, однако при этом одновременно происходит изменение статистических характеристик выходного сигнала (математического ожидания, дисперсии, корреляционной функции и пр.), которое также определяется системным оператором [14, 18, 20].

Для определения системы необходимо задать характер, тип и области допустимых величин входных и выходных сигналов. Как правило, системы выполняются на сигналы одного типа по входу/выходу и подразделяются на системы непрерывного времени (аналоговые или дискретные сигналы на входе и выходе) и цифровые системы. Совокупность системного оператора T и пространства сигналов образует математическую модель системы.

Линейные системы. Любые преобразования сигналов сопровождаются изменением их спектра и по характеру этих изменений разделяются на два вида: линейные и нелинейные. К нелинейным относят изменения, при которых в составе спектра сигналов появляются новые гармонические составляющие. При линейных изменениях сигналов изменяются амплитуды и/или начальные фазы гармонических составляющих спектра. Оба вида изменений могут происходить как с сохранением полезной информации, так и с ее искажением. Это зависит не только от характера изменения спектра сигналов, но и от спектрального состава самой полезной информации.

Линейные системы составляют основной класс систем обработки сигналов. Термин линейности означает, что система преобразования сигналов должна иметь произвольную, но в обязательном порядке линейную связь между входным сигналом (возбуждением) и выходным сигналом (откликом). В нелинейных системах связь между входным и выходным сигналом определяется произвольным нелинейным законом.

Система считается линейной, если в пределах установленной области входных и выходных сигналов ее реакция на входные сигналы аддитивна (выполняется принцип суперпозиции сигналов) и однородна (выполняется

принцип пропорционального подобия).

Принцип *аддитивности* требует, чтобы реакция на сумму двух входных сигналов была равна сумме реакций на каждый сигнал в отдельности:

$$T[a(t)+b(t)] = T[a(t)]+T[b(t)].$$

Принцип *однородности* или пропорционального подобия требует сохранения однозначности масштаба преобразования при любой амплитуде входного сигнала:

$$T[c \times a(t)]= c \times T[a(t)].$$

Другими словами, отклик линейной системы на взвешенную сумму входных сигналов должен быть равен взвешенной сумме откликов на отдельные входные сигналы независимо от их количества и для любых весовых коэффициентов, в том числе комплексных.

При программной реализации линейных систем на ЭВМ особых затруднений с обеспечением линейности в разумных пределах значений входных и выходных сигналов, как правило, не возникает. При физической (аппаратной) реализации систем обработки данных диапазон входных и/или выходных сигналов, в котором обеспечивается линейность преобразования сигналов, всегда ограничен и должен быть специально оговорен в технической документации или методической инструкции.

Основные системные операции. К базовым линейным операциям, из которых могут быть сформированы любые линейные операторы преобразования, относятся операции скалярного умножения, сдвига и сложения сигналов:

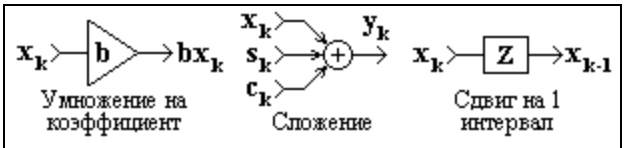

Рис.1.1. Графика системных операций

$$y(t) = b \times x(t),$$

$$y(t) = x(t-\Delta t),$$

$$y(t) = a(t)+b(t).$$

Графическое отображение операций (цифровая форма) приведено на рис.1.1.

Отметим, что операции сложения и умножения являются линейными только для аналоговых и дискретных сигналов. В случае цифровых сигналов они линейны относительно самих цифровых сигналов, но если последние - результат операции амплитудно-цифрового преобразования, то сложение и умножение не может считаться линейным абсолютно точно по отношению к исходным сигналам [14, 18].

Для систем, с размерностью 2 и более существует также еще одна базовая операция, которая называется операцией *пространственного маскирования*, которая может рассматриваться как обобщение скалярного умножения. Так, для двумерных систем:

$$z(x,y) = c(x,y)\cdot u(x,y),$$

где u(x,y) – двумерный входной сигнал, c(x,y) – пространственная маска постоянных (весовых) коэффициентов. Пространственное маскирование представляет собой поэлементное произведение значений сигнала с коэффициентами маски.

Инвариантность систем к сдвигу. Система называется инвариантной к сдвигу (инвариантной во времени, а равно и по любым другим аргументам), если сдвиг входного сигнала по аргументам вызывает соответствующий сдвиг выходного сигнала:

$$s(x,t) = T[a(x,t)], \quad T[a(x-\Delta x,t-\Delta t)] = s(x-\Delta x,t-\Delta t).$$

Линейность и инвариантность к сдвигу являются независимыми свойствами систем и не определяют друг друга. Так, например, операция квадратирования сигнала (возведения в квадрат всех значений сигнала) инвариантна к сдвигу, но нелинейна.

В теории анализа и обработки данных основное место занимают системы, линейные и инвариантные к сдвигу (ЛИС - системы). Они обладают достаточно широкими практическими возможностями при относительной простоте математического аппарата. В дальнейшем, если это специально не оговаривается, будем иметь в виду именно такие системы.

Преимущество, которое отдается ЛИС - системам в методах обработки

информации, базируется на возможности разложения входного сигнала любой, сколь угодно сложной формы, на составляющие простейших форм, отклик системы на которые известен и хорошо изучен, с последующим вычислением выходного сигнала в виде суммы откликов на все составляющие входного сигнала. В качестве простейших форм разложения сигналов используются, как правило, единичные импульсы и гармонические составляющие. Первая применяется при представлении сигнала в динамической форме и использует преобразование свертки, вторая - частотное представление сигнала и преобразование Фурье.

Другой важной особенностью ЛИС - систем является то, что любые их комбинации также являются ЛИС - системами, а любую сложную ЛИС - систему можно разложить на комбинации простых систем. Так, например, при последовательном (каскадном) соединении систем, когда выходной сигнал одной системы служит входным сигналом для второй и т.д., образуемая система в целом также является ЛИС - системой, если линейны и инвариантны к сдвигу все системы, в нее входящие, при этом по отношению к общей системной операции преобразования порядок соединения входящих в нее систем значения не имеет.

Математическая модель системы задает связь физических (программных) элементов, образующих техническое устройство, способное воспринимать внешнее воздействие s(t) и формировать выходную величину y(t), определенным образом зависимую от воздействия x(t). В аналитической форме эта связь в аналоговой одномерной линейной системе обычно выражается линейным дифференциальным уравнением:

$$\sum_{m=0}^{M} a_m \frac{d^m y(t)}{dt^m} = \sum_{n=0}^{N} b_n \frac{d^n s(t)}{dt^n}. \qquad (1.1)$$

При нормировке к $a_o = 1$, отсюда следует:

$$y(t) = \sum_{n=0}^{N} b_n \frac{d^n s(t)}{dt^n} - \sum_{m=1}^{M} a_m \frac{d^m y(t)}{dt^m}. \qquad (1.1')$$

По существу, правой частью этого выражения в самой общей

математической форме отображается содержание операции преобразования входного сигнала, т.е. задается оператор трансформации входного сигнала в выходной. Для однозначного решения уравнений (1.1) должны задаваться начальные условия: значения решения y(0) и его производных по времени в начальный (нулевой) момент времени. В физической системе эти значения определяются начальной энергией в элементах, способных ее накапливать. Входным воздействием может быть произвольный сигнал. Коэффициенты a_m и b_n, которые определяются составом и свойствами системы, должны быть известными.

Как правило, решение уравнения (1.1) складывается из суммы принужденной $y_{пр}(t)$ и свободной $y_{св}(t)$ составляющих:

$$y(t) = y_{пр}(t) + y_{св}(t).$$

В устойчивых системах $y_{св}(t)$ определяется параметрами системы (реакция на "ударное" воздействие – единичный "дельта-импульс" или скачок) и с течением времени затухает. Если входное воздействие достаточно гладкое и не содержит скачков (разрывов первого рода), то по истечении определенного времени затухания $y_{св}(t)$, которое обычно называют временем переходного процесса, система переходит в установившийся динамический режим, а выходной сигнал y(t) определяется только значением $y_{пр}(t)$ и линейно связан с входным воздействием x(t).

Аналогичная модель в цифровой системы описывается разностными уравнениями:

$$\sum_{m=0}^{M} a_m\, y((k-m)\Delta t) = \sum_{n=0}^{N} b_n\, s((k-n)\Delta t). \qquad (1.2)$$

$$y(k\Delta t) = \sum_{n=0}^{N} b_n\, s((k-n)\Delta t) - \sum_{m=1}^{M} a_m\, y((k-m)\Delta t). \qquad (1.2')$$

Последнее уравнение можно рассматривать как алгоритм последовательного вычисления значений $y(k\Delta t)$, $k = 0,1,2, \ldots$, по значениям входного сигнала $s(k\Delta t)$ и предыдущих вычисленных значений $y(k\Delta t)$ при известных значениях коэффициентов a_m, b_n и с учетом задания определенных

начальных условий - значений $s(k\Delta t)$ и $y(k\Delta t)$ при $k < 0$. Интервал дискретизации в цифровых последовательностях отсчетов обычно принимается равным 1, т.к. выполняет только роль масштабного множителя.

Нерекурсивные цифровые системы. При нулевых значениях коэффициентов a_m уравнение (1.2') переходит в уравнение дискретной свертки x(k) с оператором b_n:

$$y(k) = \sum_{n=0}^{N} b_n x(k-n). \qquad (1.3)$$

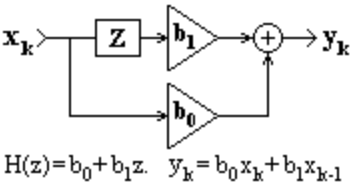

Рис. 1.2. Пример НЦС.

При установленных значениях коэффициентов b_n значения выходных отсчетов свертки для любого аргумента k определяются текущим и "прошлыми" значениями входных отсчетов. Такая система называется нерекурсивной цифровой системой (НЦС). Пример простейшей НЦС приведен на рис. 1.2. Интервал суммирования по n получил название "окна" системы. Окно системы (1.3) составляет N+1 точку, система является односторонней каузальной, причинно обусловленной текущими и "прошлыми" значениями входного сигнала, выходной сигнал не опережает входного. Каузальная система может быть реализована аппаратно в реальном масштабе времени. При $k<n$ проведение обработки входных данных возможно только при задании определенных начальных условий для точек x(-k), k=1,2,..,N. Как правило, в качестве начальных условий принимаются нулевые значения или значения отсчета x(0). Применяется также четное или нечетное продление функции x(k) на интервал отрицательных значений k. Если при обработке данных начальные интервалы массивов x(k) существенного значения не имеют, то обработку можно начинать с отсчета k=N.

При обработке данных на ЭВМ ограничение по каузальности системного оператора снимается. В программном распоряжении системы могут находиться как "прошлые", так и "будущие" значения входных отсчетов, при этом уравнение (1.3) будет иметь вид:

$$y(k) = \sum_{n=-N'}^{N} b_n x(k-n). \qquad (1.3')$$

При N' = N система называется двусторонней симметричной. Симметричные системы, в отличие от каузальных, не изменяют фазы обрабатываемых сигналов.

Техника выполнения свертки в координатной области не отличается от техники выполнения обычной дискретной свертки двух массивов данных.

Представим, что на одной полоске бумаги выписаны по порядку сверху вниз значения данных x(k). На второй полоске бумаги находятся записанные в обратном порядке значения коэффициентов системы b_n. Для вычисления y(k) располагаем вторую полоску против первой таким образом, чтобы значение b_0 совпало со значением x(k), перемножаем все значения b_n с расположенными против них значениями x(k-n) и суммируем результаты перемножения. Результат суммирования является выходным значением сигнала y(k). Сдвигаем окно системы - полоску коэффициентов b_k, на один отсчет последовательности x(k) вниз (по порядку возрастания номеров k) или массив x(k) сдвигаем на отсчет вверх и вычисляем аналогично следующее значение, и т.д.

Описанный процесс свертки в вещественной области массива данных x(k) с нерекурсивным оператором системы b_n (массивом весовых коэффициентов системы) обычно называют нерекурсивной цифровой фильтрацией данных, а саму систему, если она выполняет только данную операцию, нерекурсивным цифровым фильтром (НЦФ).

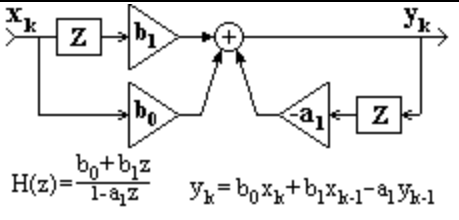

Рис.1.3. Пример РЦС.

Рекурсивные цифровые системы. Системы, которые описываются полным разностным уравнением (1.2), принято называть рекурсивными цифровыми системами (РЦС) или рекурсивными цифровыми фильтрами (РЦФ), так как в вычислении текущих значений выходного сигнала участвует не только входной сигнал, но и значения выходного

сигнала, вычисленные в предшествующих циклах расчетов. С учетом последнего фактора рекурсивные системы называют системами с обратной связью. Пример рекурсивной системы приведен на рис.1.3.

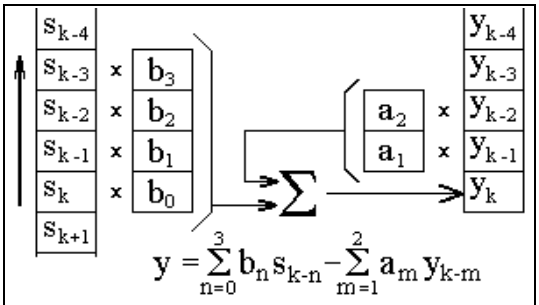

Рис.1.4. Техника вычислений в РЦС.

Полное окно рекурсивной системы состоит из двух составляющих: нерекурсивной части b_n,, аналогичной окну нерекурсивной системы и ограниченной в работе текущими и "прошлыми" значениями входного сигнала (при реализации на ЭВМ возможно использование и “будущих” отсчетов сигнала), и рекурсивной части a_m, которая работает только с "прошлыми", ранее вычисленными значениями выходного сигнала. Техника вычислений для РЦС приведена на рис. 1.4.

Из примера можно видеть, что реакция РЦС на конечный входной сигнал, в принципе, может иметь бесконечную длительность, в отличие от реакции НЦС, которая всегда ограничена количеством членов b_k (окном системы).

Стационарные и нестационарные системы. Система считается *стационарной* и имеет постоянные параметры, если ее свойства (математический алгоритм оператора преобразования) в пределах заданной точности не зависят от входного и выходного сигналов и не изменяются ни во времени, ни от каких-либо других внешних факторов. Математически это означает задание системы уравнениями типа (1.2) с постоянными значениями коэффициентов a_j и b_i и реакция системы на какое-либо воздействие не зависит от времени (координат) его приложения. В противном случае система является нестационарной или *параметрической* (системой с переменными параметрами). Среди последних большое значение имеют так называемые адаптивные системы обработки данных. В этих системах производится, например, оценивание определенных параметров входных и выходных сигналов, по результатам сравнения которых осуществляется

подстройка параметров преобразования (переходной характеристики системы) таким образом, чтобы обеспечить оптимальные по производительности условия обработки сигналов или минимизировать погрешность обработки.

1.2. Анализ структурные схем линейных систем

Структурные схемы. Алгоритмы обработки сигналов в системах наглядно отображаются в виде структурных схем. Базовые элементы схем и примеры построения структурных схем приводились ранее на рис.1.3. Как правило, структурные схемы соответствуют программной реализации систем и будут рассматриваться ниже применительно к цифровым системам, но не определяют аппаратной реализации в специальных радиотехнических устройствах, которая может существенно отличаться от программной реализации, особенно для аналоговых систем [5,20].

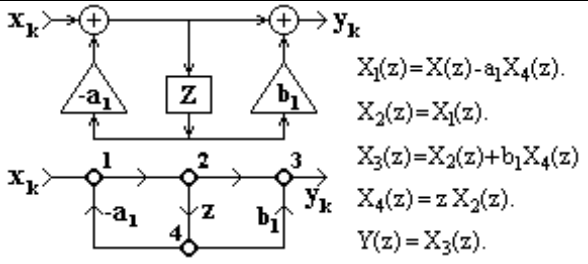

Рис. 1.5. Структурная схема и граф системы.

Графы систем. Наряду со структурной схемой система может быть представлена в виде графа, который отображает диаграмму прохождения сигналов и состоит из направленных ветвей и узлов.

Пример структурной схемы системы с передаточной функцией $H(z) = (1+b_1z)/(1+a_1z)$ и графа, ей соответствующего, приведен на рисунке 1.5. С каждым i узлом графа связано значение сигнала $x_i(k)$ или его образа $X_i(z)$, которые определяются суммой всех сигналов или их z-образов входящих в узел ветвей. В каждой ij - ветви (из узла i в узел j) происходит преобразование сигнала в соответствии с передаточной функцией ветви, например, задержка сигнала или умножение на коэффициент.

Соединения систем. Различают следующие соединения систем (рис. 1.6-1.8).

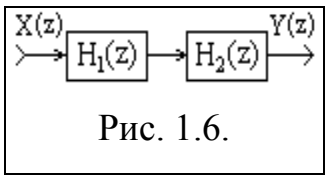

Рис. 1.6.

1. *Последовательное соединение.* Выходной сигнал предшествующей системы является входным для последующей. Эквивалентная передаточная функция общей системы равна произведению передаточных функций систем, в нее входящих:

$$H(z) = H_1(z)\cdot H_2(z)\cdot\ldots\cdot H_N(z).$$

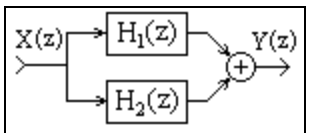

Рис. 1.7.

2**.** *Параллельное соединение.* Сигнал подается на входы всех параллельно соединенных систем одновременно, выходные сигналы систем суммируются. Эквивалентная передаточная функция общей системы равна сумме передаточных функций систем, в нее входящих:

$$H(z) = H_1(z)+H_2(z)+\ldots+H_N(z).$$

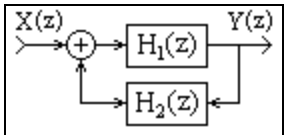

Рис. 1.8.

3. *Соединение обратной связи.* Выходной сигнал первой системы подается на выход системы и одновременно на вход системы обратной связи, выходной сигнал которой суммируется, со знаком плюс или минус в зависимости от вида связи (отрицательной или положительной), с входным сигналом первой системы. Эквивалентная передаточная функция общей системы: $H(z) = H_1(z)/(1\pm H_1(z)H_2(z))$.

Схемы реализации систем. По принципам структурной реализации систем различают следующие схемы:

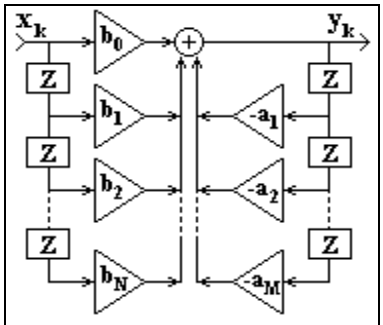

Рис. 1.9.

1. *Прямая форма.* Реализуется непосредственно по разностному уравнению

$$y_k = \sum_{n=0}^{N} b_n x_{k-n} - \sum_{m=1}^{M} a_m y_{k-m},$$

или по передаточной функции

$$H(z) = \sum_{n=0}^{N} b_n z^n /(1+ \sum_{m=1}^{M} a_m z^m).$$

Пример прямой системы приведен на рис. 1.9.

2. *Прямая каноническая форма* содержит минимальное число элементов задержки. Передаточную функцию РЦС можно представить в

виде:

$$H(z) = Y(z)/X(z) = H_1(z)H_2(z),$$

$$H_1(z) = V(z)/X(z) = 1/(1+\sum_{m=1}^{M} a_m z^m), \qquad H_2(z) = Y(z)/V(z) = \sum_{n=0}^{N} b_n z^n.$$

Отсюда:

$$v(k) = x(k) - \sum_{m=1}^{M} a_m v(k-m), \qquad (1.4)$$

$$y(k) = \sum_{n=0}^{N} b_n v(k-n). \qquad (1.5)$$

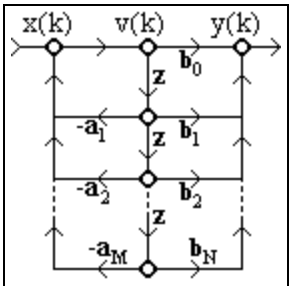

Рис. 1.10.

В разностных уравнениях (1.4-1.5) осуществляется только задержка сигналов v(k). Граф реализации РЦС приведен на рисунке 1.10.

3. *Каскадная (последовательная) форма***.** Соответствует представлению передаточной функции в виде произведения:

$$H(z) = \prod_{i=1}^{k} H_i(z).$$

$H_i(z)$ - составляющие функции типа $(1-r_iz)/(1-p_iz)$ при представлении H(z) в факторизованной форме, где r_i и p_i- нули и полюсы функции H(z). В качестве функций $H_i(z)$ обычно используются передаточные функции биквадратных блоков - фильтров второго порядка:

$$H_i(z) = (b_{0i}+b_{1i}\cdot z+b_{2i}\cdot z^2)/(1+a_{1i}\cdot z+a_{2i}\cdot z^2).$$

4. *Параллельная форма.* Используется много реже и соответствует представлению передаточной функции в виде суммы биквадратных блоков или более простых функций.

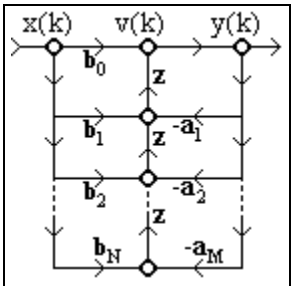

Рис. 1.11.

Обращенные формы. В теории линейных сигнальных графов существуют процедуры преобразования исходных графов с сохранением передаточных функций. Одна из таких процедур - обращение графов, которое выполняется путем изменения направления всех ветвей цепи, при этом вход и

выход графа также меняются местами. Для ряда систем такая транспозиция позволяет реализовать более эффективные алгоритмы обработки данных. Пример обращения графа прямой канонической формы рекурсивной системы (с перестроением на расположение входа с левой стороны) приведен на рис. 1.11.

1.3. Основные задачи и направления цифровой обработки сигналов

В технических отраслях знаний термин "сигнал" (signal, от латинского signum – знак) очень часто используется в широком смысловом диапазоне, без соблюдения строгой терминологии. Под ним понимают и техническое средство для передачи, обращения и использования информации - электрический, магнитный, оптический сигнал; и физический процесс, представляющий собой материальное воплощение информационного сообщения - изменение какого-либо параметра носителя информации (напряжения, частоты, мощности электромагнитных колебаний, интенсивности светового потока и т.п.) во времени, в пространстве или в зависимости от изменения значений каких-либо других аргументов (независимых переменных); и смысловое содержание определенного физического состояния или процесса, как, например, сигналы светофора, звуковые предупреждающие сигналы и т.п. Все эти понятия объединяет конечное назначение сигналов. Это определенные сведения, сообщения, информация о каких-либо процессах, состояниях или физических величинах объектов материального мира, выраженные в форме, удобной для передачи, обработки, хранения и использования этих сведений.

Наиболее распространенное представление сигналов - в электрической форме в виде зависимости напряжения от времени U(t). Так, например, сигнал изменения напряженности магнитного поля по профилю аэросъемки – это и временная последовательность изменения электрического напряжения на выходе датчика аэромагнитометра, и запись этого напряжения на ленте

регистратора, и последовательные значения цифровых отсчетов при обработке лент регистратора и вводе сигнала в ЭВМ.

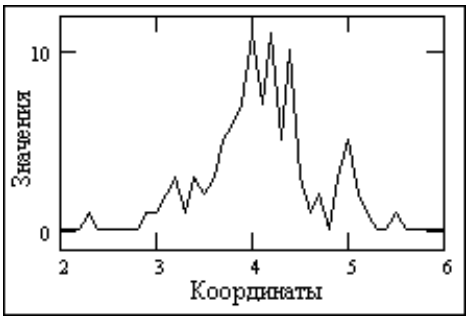

Рис. 1.12. Сигнал.

С математической точки зрения сигнал представляет собой функцию, т.е. зависимость одной величины от другой, независимой переменной.

Под "анализом" сигналов (analysis) имеется в виду не только их чисто математические преобразования, но и получение на основе этих преобразований выводов о специфических особенностях соответствующих процессов и объектов. Целями анализа сигналов обычно являются:

- Определение или оценка числовых параметров сигналов (энергия, средняя мощность, среднее квадратическое значение и пр.).

- Разложение сигналов на элементарные составляющие для сравнения свойств различных сигналов.

- Сравнение степени близости, "похожести", "родственности" различных сигналов, в том числе с определенными количественными оценками.

Математический аппарат анализа сигналов весьма обширен, и широко применяется на практике во всех без исключения областях науки и техники.

Классификация сигналов осуществляется на основании существенных признаков соответствующих математических моделей сигналов**.** Все сигналы разделяют на две крупных группы: детерминированные и случайные. Классификация сигналов внутри групп приведена на рис. 1.13. [5,20].

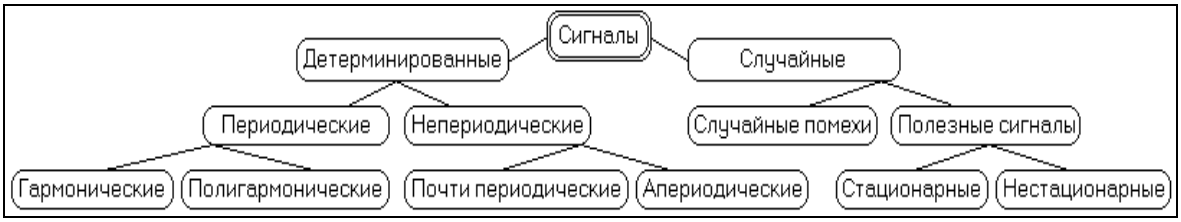

Рис. 1.13. Классификация сигналов.

С математических позиций группы сигналов обычно называют множествами, в которые объединяют сигналы по какому-либо общему свойству. Принадлежность сигнала s к множеству L_P записывается в виде $L_P = \{s; P\}$, где P – определенное свойство данного множества сигналов.

Типы сигналов. Выделяют следующие типы сигналов, которым соответствуют определенные формы их математического описания.

Аналоговый сигнал (analog signal) является непрерывной функцией непрерывного аргумента, т.е. определен для любого значения аргументов. Источниками аналоговых сигналов, как правило, являются физические процессы и явления, непрерывные в динамике своего развития во времени, в пространстве или по любой другой независимой переменной, при этом регистрируемый сигнал подобен (“аналогичен”) порождающему его процессу.

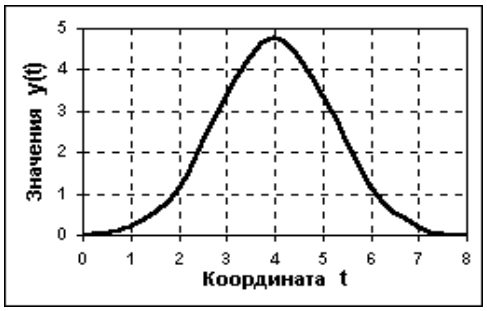

Рис. 1.14. Аналоговый сигнал.

Пример математической записи сигнала: $y(t) = 4.8 \exp[-(t-4)^2/2.8]$. Пример графического отображения данного сигнала приведен на рис. 1.14.

Дискретный сигнал (discrete signal) по своим значениям также является непрерывной функцией, но определенной только по дискретным значениям

аргумента. По множеству своих значений он является конечным (счетным) и описывается дискретной последовательностью отсчетов (samples) $y(n\Delta t)$, где $y_1 \le y \le y_2$, Δt - интервал между отсчетами (интервал или шаг дискретизации, sample time), n = 0, 1, 2,...,N. Величина, обратная шагу дискретизации: $f = 1/\Delta t$, называется частотой дискретизации (sampling frequency). Если дискретный сигнал получен дискретизацией (sampling) аналогового сигнала, то он представляет собой последовательность отсчетов, значения которых в точности равны значениям исходного сигнала по координатам $n\Delta t$.

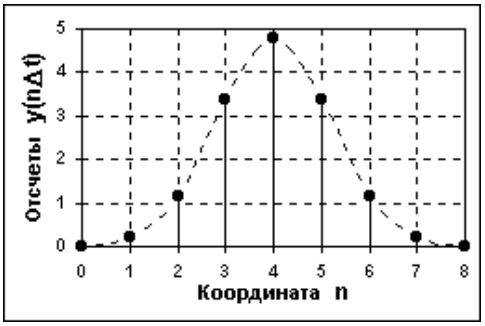

Рис. 1.15. Дискретный сигнал

Пример дискретизации аналогового сигнала, приведенного на рис. 1.14, представлен на рис. 1.15.

Цифровой сигнал (digital signal) квантован по своим значениям и дискретен по аргументу. Он описывается квантованной решетчатой функцией $y_n = Q_k[y(n\Delta t)]$, где Q_k - функция квантования с числом уровней квантования k, при этом интервалы квантования могут быть как с равномерным распределением, так и с неравномерным, например - логарифмическим. Задается цифровой сигнал, как правило, в виде дискретного ряда (discrete series) числовых данных - числового массива по последовательным значениям аргумента при Δt = const, но в общем случае сигнал может задаваться и в виде таблицы для произвольных значений аргумента.

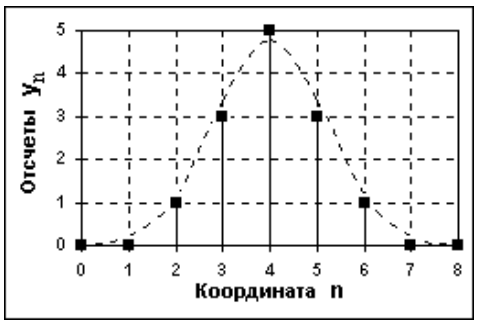

Рис. 1.16. Цифровой сигнал

По существу, цифровой сигнал по своим значениям (отсчетам) является формализованной разновидностью дискретного сигнала при округлении отсчетов последнего до определенного количества цифр, как это показано на рис 1.16. Цифровой сигнал конечен по множеству своих значений.

В системах цифровой обработки данных и в ЭВМ сигнал всегда представлен с точностью до определенного количества разрядов, а, следовательно, всегда является цифровым.

Основные задачи и направление цифровой обработки сигналов

Среди многочисленных задач, решаемых на базе цифровой обработки сигналов (ЦОС), можно выделить группу наиболее полно характеризующих как традиционные, так и нетрадиционные области применения ЦОС. Каждая задача — в зависимости от конкретного приложения — может решаться с использованием различных методов и алгоритмов; например, задача выделения сигнала из помех может решаться методами линейной, адаптивной и нелинейной фильтрации.

В настоящее время выделяют следующие основные направления ЦОС:

– *линейная фильтрация*. Селекция сигнала в частотной области; синтез фильтров, согласованных с сигналами; частотное разделение каналов; цифровые преобразователи Гильберта и дис}хреренциаторы; корректоры характеристик каналов;

- *спектральный анализ.* Обработка речевых, звуковых, сейсмических, гидроакустических сигналов; распознавание образов;
- *частотно-временной анализ.* Компрессия изображений, тидро- и радиолокация, разнообразные задачи обнаружения;
- *адаптивная фильтрация.* Обработке речи, изображений, распознавание образов, подавление шумов, адаптивные антенные решетки;
- *нелинейная обработка.* Вычисление корреляций, медианная фильтрация: синтез амплитудных, фазовых, частотных детекторов, обработка речи, векторное кодирование;
- *многоскоростная обработка.* Интерполяция (увеличение) и децимация (уменьшение) частоты дискретизации в многоскоростных системах телекоммуникации, аудиосистемахё

1.4. Области применения цифровой обработки сигналов

К числу областей применения цифровой обработки сигналов можно отнести следующие.

Обработка изображений: распознавание образцов; машинное зрение; улучшение изображения; факсимиле; спутниковые карты погоды; анимация.

Инструментальные средства/контроль: спектральный анализ; контроль положения и скорости; снижение шума; сжатие информации.

Речь/аудио: распознавание речи; синтез речи; озвучивание текста; цифровые аудиосистемы; выравнивание.

Военные цели: безопасная связь; работа с радарами; работа с сонарами; управление ракетами.

Телекоммуникации: устранение эха; адаптивное выравнивание; транскодеры ADPCM; расширение спектра; видсоконференц-связь; передача данных.

Биомедицина: наблюдение за пациентами; сканеры; карты электроэнцефалограммы мозга; анализ электрокардиограмм; хранение (улучшение) рентгеновских снимков.

Потребительские цели: цифровые сотовые мобильные телефоны; универсальная мобильная система связи; цифровое телевидение; цифровые камеры; телефонная связь, музыка и видео через Internet; цифровые автоответчики, факсы и модемы; системы голосовой почты; интерактивные развлекательные системы; активная подвеска в автомобилях.

С одного взгляда на этот отнюдь не полный список можно понять огромное значение ЦОС. Об осознании значения ЦОС свидетельствует постоянное внедрение мощных средств ЦОС производителями полупроводниковой техники. В то же время не хватает инженеров, обладающих достаточными знаниями в данной сфере. Цель этой книги — обеспечить понимание методов ЦОС и их реализации, дать читателям возможность овладеть практическими знаниями по столь важному предмету.

Выводы

В первой главе получены следующие исследования:

1. Свойства линейных систем анализированы структурные схемы линейных систем

2. Определены основные задачи и направления цифровой обработки сигналов.

3. Анализированы области применения цифровой обработки сигналов.

ГЛАВА 2. АЛГОРИТМЫ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

2.1. Анализ ключевых операции цифровой обработки сигналов

Существует несколько алгоритмов ЦОС, еще больше находится в стадии разработки или ждет своего открывателя. Однако для всех этих алгоритмов, включая самые сложные, необходимы одни и те же основные операции. Для начала будет полезно рассмотреть некоторые из них, чтобы оценить простоту реализации ЦОС. Итак, основные операции ЦОС — это свертка, корреляция, фильтрация, преобразования и модуляция. Следует заметить, что для всех основных операций ЦОС потребуется выполнение только простых арифметических действий — умножения, сложения, вычитания и операции сдвига. Также отметим сходство между многими операциями [5,12].

Свертка. Свертка — это одна из наиболее используемых операций в ЦОС. Например, это основная операция цифровой фильтрации. Для двух данных конечных и причинных последовательностей x(n)и h(n) с длиной N1 и N2 соответственно их свертка определяется как

$$y(n) = h(n) \circledast x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) =$$

$$= \sum_{k=0}^{\infty} h(k)x(n-k), \quad n = 0, 1, \ldots, (M-1), \tag{2.1}$$

где символ $\circledast$ используется для обозначения свертки, а $M = N_1 + N_2 - 1.$ Как будет видно из последующих глав, производители устройств ЦОС разработали процессоры для обработки сигналов, которые эффективно выполняют операции умножения с накоплением, задействованные в свертке.

Значение свертки становится более очевидным, если рассматривать ее в частотных координатах и использовать тот факт, что свертка во временных координатах эквивалентна умножению в частотной области.

Корреляция. Существует две формы корреляции: автокорреляция и взаимная корреляция.

Взаимно-корреляционная функция (ВКФ) — это показатель сходства или общих свойств двух сигналов. В число областей применения ВКФ входят: взаимный спектральный анализ, детектирование (восстановление) сигналов, скрытых в шуме, например, детектирование ответных сигналов радара, подбор по образцу и измерение задержки. Автокорреляционная функция (АКФ) подразумевает существование только одного сигнала и дает информацию о структуре сигнала или его поведении во времени. Это частный случай ВКФ, и их сферы применения сходны. Автокорреляционная функция особенно полезна для выявления скрытой периодичности.

Примеры ВКФ и АКФ некоторых сигналов приведены на рис. 2.1 и 2.2. Стоит заметить, например, что по АКФ искаженного шумом сигнала ясно видно, что за шумом скрывается периодический сигнал (рис. 2.1). На рис. 2.2 показано, как с помощью ВКФ измеряется задержка, которую вводит система, — ее можно узнать, измерив, время от начала отсчета до большого максимума.

Обратите внимание на то, что на панели в периодический характер сигнала, скрытого за шумом, все это заметен: по этой причине автокорреляции часто используют дня выявления скрытой периодичности.

Цифровая фильтрация. Как станет очевидным при изучении одной из операций ЦОС. имеющих первостепенное значение, является линейная цифровая фильтрация, которая определяется как

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k), \qquad (2.1)$$

где $h(k), k = 0, 1, \ldots, N-1$ — коэффициенты фильтра, а *x(k)* и *y(k*) соответственно — вход н выход фильтра. Для заданного фильтра коэффициенты, определяющие характеристики этого фильтра, однозначны [1,4,5,20].

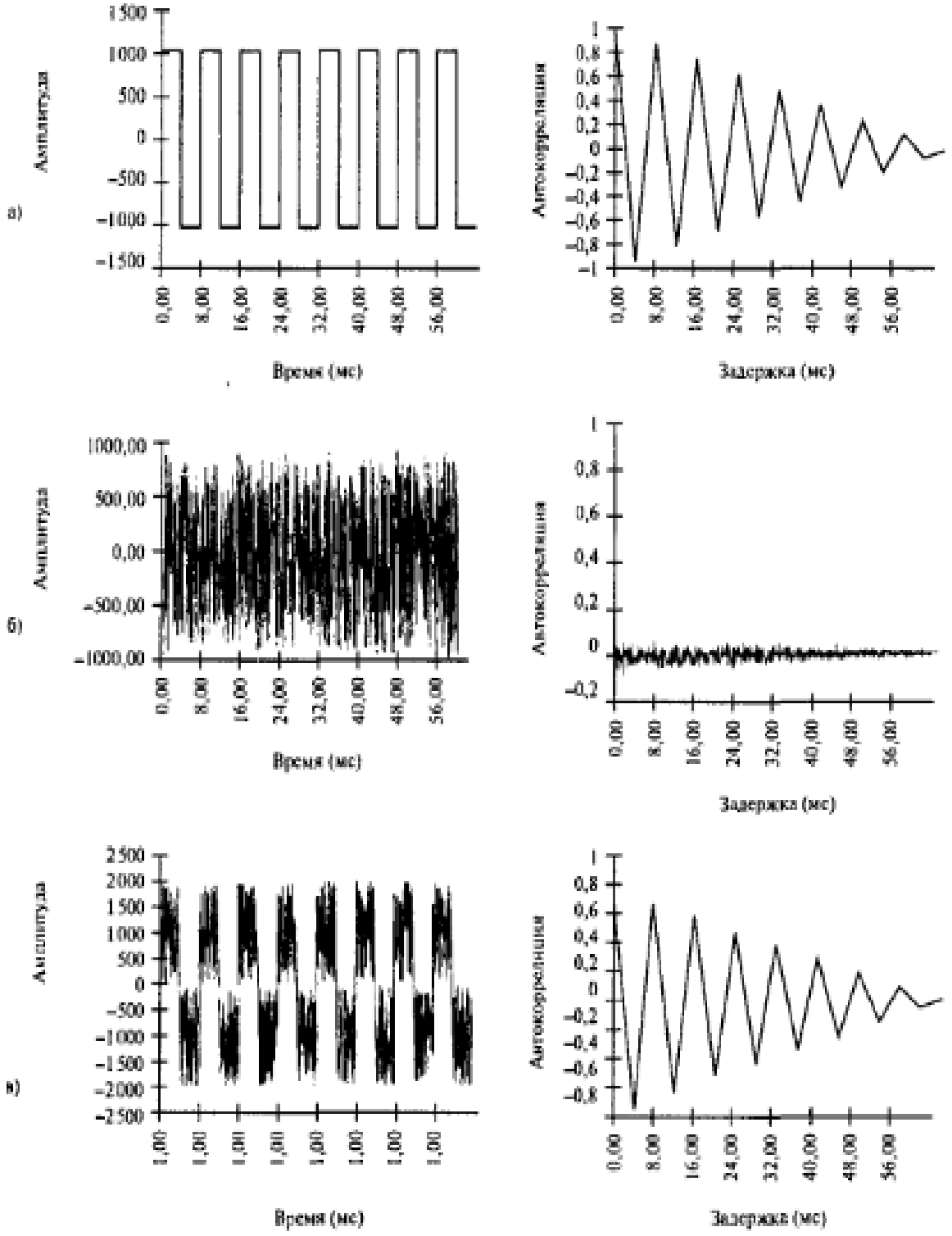

Рис. 2.1. Автокорреляция: а) периодического сигнала, б) шума;
в) периодического сигнала с шумом.

Заметим, что фильтрация — это, по сути, свертка сигнала с импульсной характеристикой фильтра во временных координатах, т.е. *h(k).* На рис. 2.3, а показана блок-схема фильтра, определение которого было дано выше. В таком виде этот фильтр широко известен как трансверсальный фильтр. На рисунке через z^{-1} обозначена задержка на один интервал дискретизации.

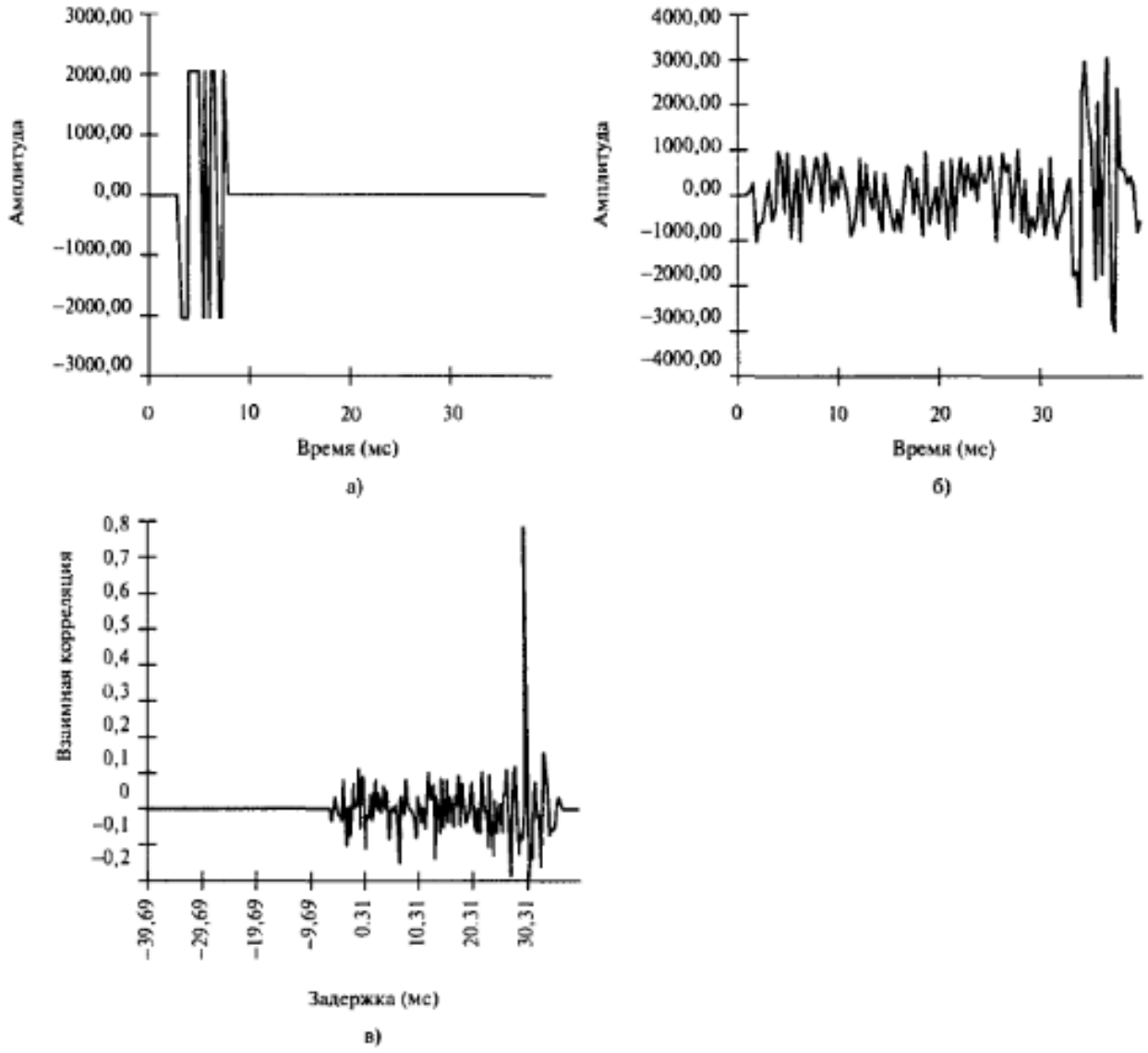

Рис. 2.2. Взаимная корреляция случайного сигнала *x(t)* с задержанной зашумленной версией того же сигнала *y(t*). Задержка между двумя сигналами равна времени от начала отсчета до времени появления максимума их взаимной корреляции.

Как правило, целью фильтрации является устранение или снижение шума в полезном сигнале. Например, на рис. 2.3, б показана низкочастотная фильтрация некоторого биомедицинского сигнала для устранения высокочастотных искажений. Вообще в подобных сферах цифровой фильтр используется в основном для минимизации искажений внутриполостных компонентов сигнала.

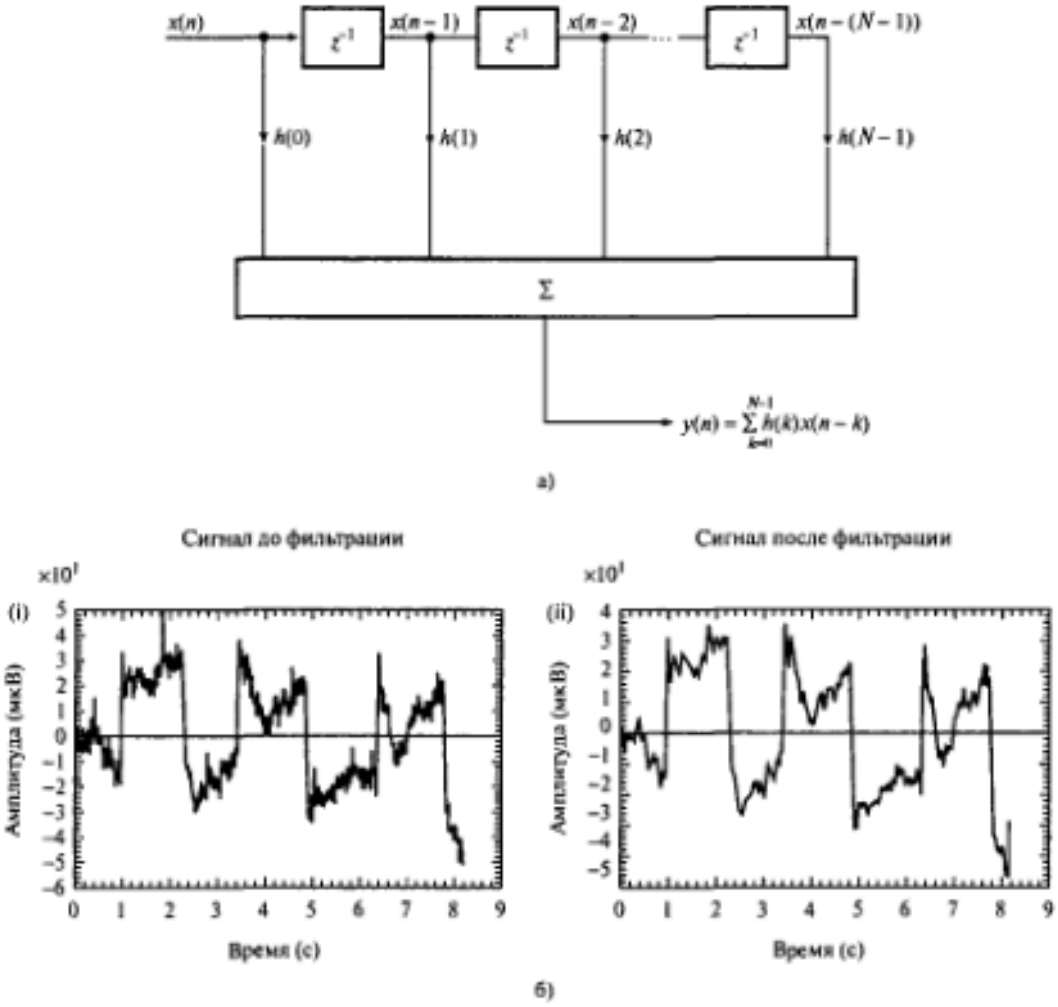

Рис. 2.3. Блок-схема трансверсального фильтра.

Здесь $h(k), k = 0, 1, \ldots, N-1$ - коэффициенты фильтра, а каждый квадрат с z^{-1} обозначает задержку на один интервал дискретизации (панель а); цифровая низкочастотная филы рация биомедицинского сигнала с целью устранения шума (панель 6).

Дискретные преобразования. Дискретные преобразования позволяют описывать сигналы с дискретным временем в частотных координатах или переходить от описания во временной области к описанию в частотной. Для получения спектра сигнал раскладывается на частотные составляющие с помощью дискретного преобразования. Знание такого спектра неоценимо, например, при определении ширины полосы, что необходимо для передачи сигнала. Переход от временных координат к частотным необходим во многих приложениях ЦОС. Например, он позволяет более эффективно реализовывать такие алгоритмы ЦОС, как цифровая фильтрация, свертка и корреляция.

Существует много дискретных преобразований, из которых самым распространенным является дискретное преобразование Фурье (ДПФ), которое определяется следующим образом:

$$X(n) = \sum_{k=0}^{N-1} x(k) W^{kn}, \text{ где } W = \exp(-2\pi i/N). \tag{2.3}$$

Пример использования ДПФ приведен на рис. 2.4. Здесь импульсная характеристика фильтра $h(n), n = 0, 1, \ldots, N-1$, с помощью ДПФ преобразуется в частотную.

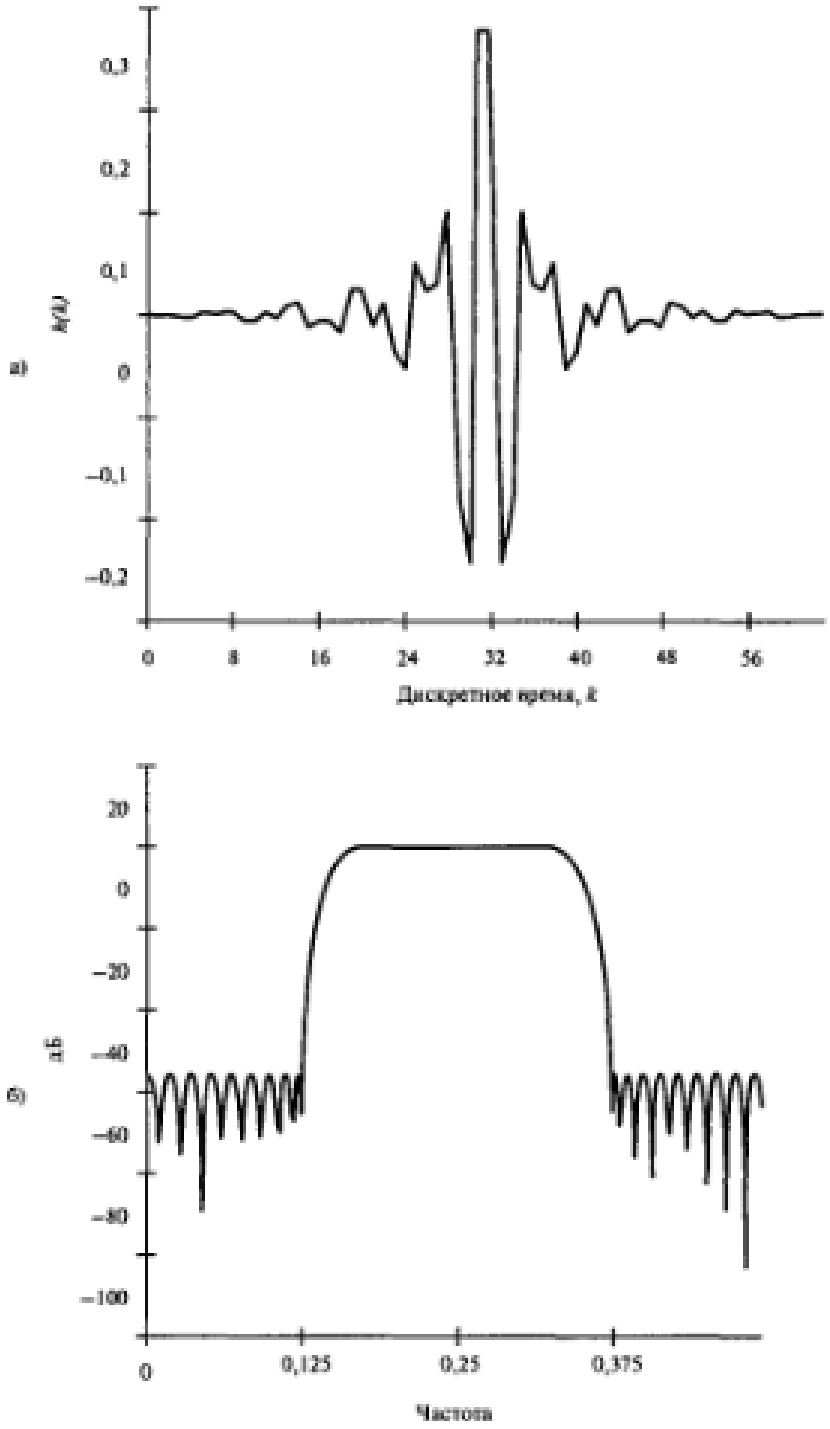

Рис. 2.4. Описание цифрового фильтра во временных и частотных координатах: а) импульсная характеристика; б) спектр фильтра.

2.2. Спектральные методы обработки сигналов

Цифровой спектральный анализ — это совокупность разнообразных методов обработки цифровых сигналов, которые позволяют оценить частотный состав (спектр) исследуемого сигнала. Задача спектрального анализа может носить как самостоятельный характер (например, в сейсмологии для определения типа сейсмического события, в геофизике для поиска месторождений полезных ископаемых и разработки новых методов поиска и т. п.), так и вспомогательный (в системах компрессии речи и изображений) [5,12,16,20].

Дискретное преобразование Фурье Спектральный анализ часто называют частотным анализом. Термин "частотный" обязан происхождением обратной переменной f=1/|t| временного представления сигналов и функций. Понятие частотного преобразования не следует связывать только с временными аргументами, т.к. математический аппарат преобразования не зависит от физического смысла независимых переменных. Так, например, при переменной "x", как единице длины, значение f будет представлять собой пространственную частоту с размерностью 1/|x| - число периодических изменений сигнала на единице длины [22].

В ЦОС важнейшими сигналами являются периодические последовательности с периодом отсчетов N и последовательности конечной длины в N отсчетов. Для периодических последовательностей вводится *дискретное преобразование Фурье (ДПФ)*.

Единственный член этого семейства, который имеет отношение к цифровой обработке сигналов, – это дискретное преобразование Фурье (ДПФ), которое оперирует дискретной по времени выборкой периодического сигнала во временной области. Для того, чтобы быть представленным в виде суммы синусоид, сигнал должен быть периодическим. Но в качестве набора входных данных для ДПФ доступно только конечное число отсчетов (N). Эту

дилемму можно разрешить, если мысленно поместить бесконечное число одинаковых групп отсчетов до и после обрабатываемой группы, образуя, таким образом, математическую (но не реальную) периодичность.

Фундаментальное уравнение для получения N-точечного ДПФ выглядит следующим образом:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk / N} = \frac{1}{N} \sum_{n=0}^{N-1} x(n)[\cos(2\pi nk / N) - j \sin(2\pi nk / N)] \quad (2.4)$$

По отношению к этому уравнению следует сделать некоторые терминологические разъяснения. X(k) (прописная буква X) представляет собой частотный выход ДПФ в k-ой точке спектра, где k находится в диапазоне от 0 до N-1. N представляет собой число отсчетов при вычислении ДПФ.

Обратите внимание, что "N" не следует путать с разрешающей способностью АЦП или ЦАП, которая в других главах данной книги также обозначается буквой N.

Значение x(n) (строчная буква x) представляет собой n-ый отсчет во временной области, где n также находится в диапазоне от 0 до N-1. В общем уравнении x(n) может быть вещественным или комплексным.

Обратите внимание, что косинусоидальные и синусоидальные компоненты в уравнении могут быть выражены в полярных или прямоугольных координатах, связь между которыми определяется формулой Эйлера:

$$e^{j\theta} = \cos\theta + j \sin\theta \quad (2.5)$$

Выходной спектр ДПФ X(k) является результатом вычисления свертки между выборкой, состоящей из входных отсчетов во временной области, и набором из N пар гармонических базисных функций (косинус и синус). Концепцию хорошо иллюстрирует рис.5.4, на котором представлена вещественная часть первых четырех точек спектра (показаны только косинусоидальные гармонические базисные функции). Подобная же

процедура используется для вычисления мнимой части спектра на основе синусоидальных функций.

Первая точка X(0) является простой суммой входных отсчетов во временной области, потому что cos(0) = 1. Коэффициент масштабирования 1/N не учитывается, но должен присутствовать в конечном результате. Обратите внимание, что X(0) – это среднее значение отсчетов во временной области, или просто смещение по постоянному току. Вторая точка ReX(1) получена умножением каждого отсчета из временной области на соответствующее значение косинусоиды, имеющей один полный период на интервале N, с последующим суммированием результатов. Третья точка ReX(2) получена умножением каждого отсчета из временной области на соответствующую точку косинусоиды, которая имеет два полных периода на интервале N, с последующим суммированием результатов. Точно так же, четвертая точка ReX(3) получена умножением каждого отсчета из временной области на соответствующую точку косинусоиды с тремя полными периодами на интервале N и суммированием результатов. Этот процесс продолжается, пока не будут вычислены все N выходных отсчетов. Подобная процедура, но с использованием синусоид, применяется для вычисления мнимой части частотного спектра. Косинусоиды и синусоиды являются базисными функциями данного преобразования.

Предположим, что входной сигнал является косинусоидальным, имеющим период N, то есть он содержит один полный период в нашей выборке. Также примем его амплитуду и фазу идентичными первой косинусоидальной базисной функции cos(2πn/8). Выходной спектр содержит одну ненулевую точку ReX(1), а все другие точки ReX(k) являются нулевыми. Предположим, что теперь входная косинусоида сдвинута вправо на 90º. Значение свертки между ней и соответствующей базисной косинусоидальной функцией равно нулю.

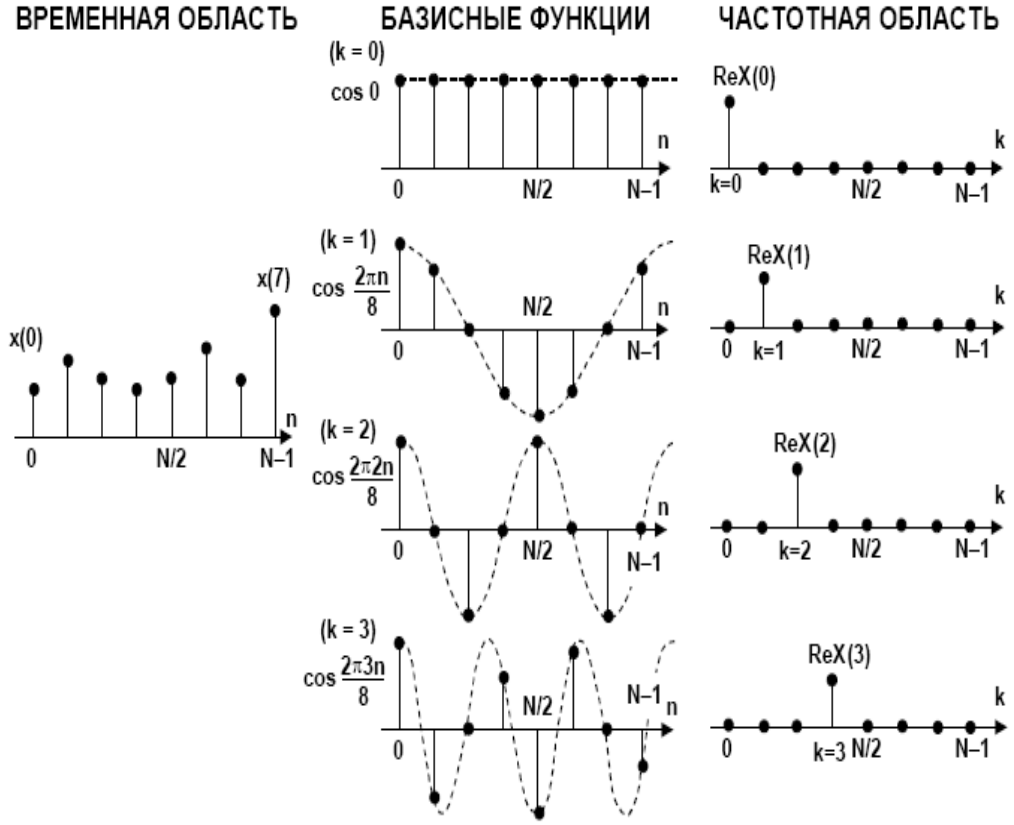

Рис. 2.5

Но алгоритм преобразования предполагает вычисление свертки с базисной функцией sin(2πn/8), необходимое для получения ImX(1). Это показывает, почему необходимо рассчитывать и вещественные, и мнимые части спектра для определения и амплитуды и фазы частотного спектра.

Обратите внимание, что свертка синусоидальной / косинусоидальной функции любой частоты, отличной от частоты базовой функции, дает нулевое значение и для ReX(1), и для ImX(1).

Подобная процедура применяется при вычислении обратного ДПФ для восстановления отсчетов во временной области x(n) из отсчетов в частотной области X(k). Соответствующее уравнение выглядит следующим образом:

$$x(n) = \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N} = \sum_{k=0}^{N-1} X(k)[\cos(2\pi nk/N) + j\sin(2\pi nk/N)] \qquad (2.6)$$

Существует два основных типа ДПФ: вещественное ДПФ и комплексное ДПФ. Уравнения, представленные на рис.2.6, описывают комплексное ДПФ, где и входные, и выходные величины являются

комплексными числами. Так как входные отсчеты во временной области являются вещественными и не имеют мнимой части, мнимая часть входных отсчетов всегда принимается равной нулю. Выход ДПФ X(k) содержит вещественную и мнимую компоненты, которые могут быть преобразованы в амплитуду и фазу.

Вещественное ДПФ выглядит несколько проще и, в основном, является упрощением комплексного ДПФ. Большинство алгоритмов вычисления быстрого преобразования Фурье (БПФ) составлено с использованием формата комплексного ДПФ, поэтому важно понимать, как работает комплексное ДПФ и как оно соотносится с вещественным ДПФ. В частности, если известны выходные частоты вещественного ДПФ и требуется использовать обратное комплексное ДПФ для вычисления отсчетов во временной области, надо знать, как разместить выходные точки вещественного ДПФ в формате комплексного ДПФ перед выполнением обратного комплексного ДПФ.

Комплексное ДПФ имеет вещественные и мнимые значения и на входе, и на выходе. Практически, мнимые части отсчетов во временной области устанавливаются в ноль. При рассмотрении спектра, получаемого в результате вычисления комплексного ДПФ, полезно знать, как связать его с результатом вычисления вещественного ДПФ и наоборот. Заштрихованные области в диаграмме соответствуют точкам, которые являются общими и для вещественного, и для комплексного ДПФ.

При цифровом спектральном анализе с помощью предлагаемым методом можно решать следующие задачи: анализаторы спектра; обработка речи; обработка изображений; распознавание образов.

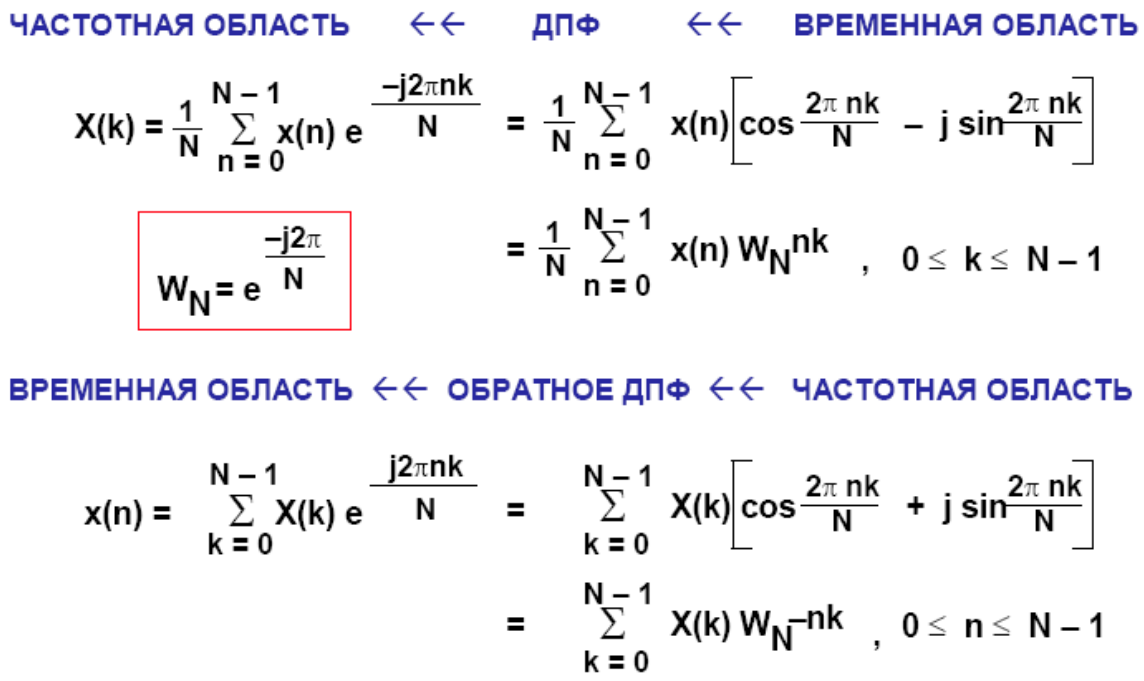

Рис. 2.6 Комплексное дискретное преобразование Фурье

Проектирование фильтров: вычисление импульсной характеристики по частотной; вычисление частотной характеристики по импульсной.

Быстрое преобразование Фурье. Из выражений ДПФ можно видеть, что для вычисления каждой гармоники нужно N операций комплексного умножения и сложения и соответственно N^2 операций на полное выполнение ДПФ. При больших объемах массивов данных это может приводить к существенным временным затратам. Ускорение вычислений достигается при использовании быстрого преобразования Фурье.

Быстрое преобразование Фурье (БПФ, fast Fourier transform - FFT). Он базируется на том, что при вычислениях среди множителей (синусов и косинусов) есть много периодически повторяющихся значений (в силу периодичности функций). Алгоритм БПФ группирует слагаемые с одинаковыми множителями в пирамидальный алгоритм, значительно сокращая число умножений за счет исключения повторных вычислений. В результате быстродействие БПФ в зависимости от N может в сотни раз превосходить быстродействие стандартного алгоритма. При этом следует

подчеркнуть, что алгоритм БПФ даже точнее стандартного, т.к. сокращая число операций, он приводит к меньшим ошибкам округления.

Для понимания принципов работы БПФ, рассмотрим ДПФ на 8 точек, представленное на рис.2.7 в развернутом виде. Обратите внимание, что для упрощения таблицы мы вводим следующее определение:

$$W_N = e^{-j2\pi/N} \qquad (2.7)$$

Это ведет к определению коэффициентов поворота (поворачивающих множителей):

$$W_N^{nk} = e^{-j2\pi nk/N} \qquad (2.8)$$

Коэффициенты поворота представляют базисные гармонические функции, записанные в экспоненциальной форме. Обратите внимание, что 8-точечное ДПФ, представленное на диаграмме, требует 64 операций умножения с комплексными числами. N-точечное ДПФ требует N2 операций умножения с комплексными числами. Знание количества умножений важно потому, что на реализацию операций умножения затрачиваются существенные вычислительные ресурсы DSP. В действительности, общее время, требуемое для вычисления ДПФ, прямо пропорционально числу умножений с учетом необходимого числа дополнительных операций.

Быстрое преобразование Фурье (FFT) является не более чем алгоритмом для ускоренного вычисления ДПФ путем сокращения требуемого числа операций умножения и сложения. Данное преобразование было предложено Кули и Таки (J.W.Cooley и J.W.Tukey) в 1960-ых годах.

Для понимания основных концепций БПФ и его происхождения, полезно обратить внимание, что ДПФ, показанное на рис.2.7 в развернутом виде, может быть сильно упрощено, если использовать свойства симметрии и периодичности коэффициентов поворота, как показано на рис.2.8.

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} = \frac{1}{N} \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \qquad W_N = e^{\frac{-j2\pi}{N}}$$

X(0) =	$x(0)W_8^0$	$+ x(1)W_8^0$	$+ x(2)W_8^0$	$+ x(3)W_8^0$	$+ x(4)W_8^0$	$+ x(5)W_8^0$	$+ x(6)W_8^0$	$+ x(7)W_8^0$
X(1) =	$x(0)W_8^0$	$+ x(1)W_8^1$	$+ x(2)W_8^2$	$+ x(3)W_8^3$	$+ x(4)W_8^4$	$+ x(5)W_8^5$	$+ x(6)W_8^6$	$+ x(7)W_8^7$
X(2) =	$x(0)W_8^0$	$+ x(1)W_8^2$	$+ x(2)W_8^4$	$+ x(3)W_8^6$	$+ x(4)W_8^8$	$+ x(5)W_8^{10}$	$+ x(6)W_8^{12}$	$+ x(7)W_8^{14}$
X(3) =	$x(0)W_8^0$	$+ x(1)W_8^3$	$+ x(2)W_8^6$	$+ x(3)W_8^9$	$+ x(4)W_8^{12}$	$+ x(5)W_8^{15}$	$+ x(6)W_8^{18}$	$+ x(7)W_8^{21}$
X(4) =	$x(0)W_8^0$	$+ x(1)W_8^4$	$+ x(2)W_8^8$	$+ x(3)W_8^{12}$	$+ x(4)W_8^{16}$	$+ x(5)W_8^{20}$	$+ x(6)W_8^{24}$	$+ x(7)W_8^{28}$
X(5) =	$x(0)W_8^0$	$+ x(1)W_8^5$	$+ x(2)W_8^{10}$	$+ x(3)W_8^{15}$	$+ x(4)W_8^{20}$	$+ x(5)W_8^{25}$	$+ x(6)W_8^{30}$	$+ x(7)W_8^{35}$
X(6) =	$x(0)W_8^0$	$+ x(1)W_8^6$	$+ x(2)W_8^{12}$	$+ x(3)W_8^{18}$	$+ x(4)W_8^{24}$	$+ x(5)W_8^{30}$	$+ x(6)W_8^{36}$	$+ x(7)W_8^{42}$
X(7) =	$x(0)W_8^0$	$+ x(1)W_8^7$	$+ x(2)W_8^{14}$	$+ x(3)W_8^{21}$	$+ x(4)W_8^{28}$	$+ x(5)W_8^{35}$	$+ x(6)W_8^{42}$	$+ x(7)W_8^{49}$

N^2 умножений с комплексными числами

$\frac{1}{N}$ Не учтенный масштабный коэффициент

Рис. 2.7. 8-точечное ДПФ (N = 8)

преобразование Фурье (FFT), которое требует только (N/2)log2(N) умножений комплексных чисел. Вычислительная эффективность БПФ по сравнению с ДПФ становится весьма существенной, когда количество точек БПФ увеличивается до нескольких тысяч, как это следует из рис.2.9. Очевидно, что БПФ вычисляет все компоненты выходного спектра (или все, или ни одного!). Если необходимо рассчитать только несколько точек спектра, ДПФ может оказаться более эффективным. Вычисление одного выходного отсчета спектра с использованием ДПФ требует только N умножений с комплексными числами.

- БПФ является лишь алгоритмом эффективного вычисления ДПФ
- Вычислительная эффективность N-точечного БПФ:
  - ДПФ: N^2 вычислений с комплексными числами
  - БПФ : $(N/2)\log_2(N)$ вычислений с комплексными числами

N	Умножений при ДПФ	Умножений при БПФ	Эффективность БПФ
256	65,536	1,024	64 : 1
512	262,144	2,304	114 : 1
1,024	1,048,576	5,120	205 : 1
2,048	4,194,304	11,264	372 : 1
4,096	16,777,216	24,576	683 : 1

Рис. 2.9 БПФ по сравнению с ДПФ

В упрощенной диаграмме операции умножения помечаются множителем возле стрелки, а под суммированием подразумеваются две стрелки, сходящиеся в точке.

2.3. Фильтрация сигналов с использованием линейных методов

Во временной области наиболее популярными методами фильтрации являются рекурсивная и нерекурсивная фильтрации.

В одномерной дискретной линейной системе связь между входом и выходом (входной и выходной дискретными последовательностями значений сигнала – отсчетами), задается линейным оператором преобразования TL:

$$y(k\Delta t) = TL\{x(k\Delta t)\} \qquad (2.10)$$

Это выражение отображает краткую запись линейного разностного уравнения:

$$\sum_{m=0}^{M} a_m\, y(k\Delta t - m\Delta t) = \sum_{n=0}^{N} b_n\, x(k\Delta t - n\Delta t), \qquad (2.11)$$

где k = 0,1,2,..- порядковый номер отсчетов, Δt - интервал дискретизации сигнала, a_m и b_n - вещественные или, в общем случае, комплексные

коэффициенты. Положим $a_0 = 1$, что всегда может быть выполнено соответствующей нормировкой уравнения (2.11), и, принимая в дальнейшем $\Delta t = 1$, приведем его к виду:

$$y(k) = \sum_{n=0}^{N} b_n x(k-n) - \sum_{m=1}^{M} a_m y(k-m) \qquad (2.12)$$

Оператор, представленный правой частью данного уравнения, получил название цифрового фильтра (ЦФ), а выполняемая им операция - цифровой фильтрации данных (информации, сигналов). Если хотя бы один из коэффициентов a_m или b_n зависит от переменной k, то фильтр называется параметрическим, т.е. с переменными параметрами. Ниже мы будем рассматривать фильтры с постоянными коэффициентами (инвариантными по аргументу).

Нерекурсивная фильтрация. При нулевых значениях коэффициентов a_m уравнение переходит в уравнение линейной дискретной свертки функции x(k) с оператором b_n:

$$y(k) = \sum_{n=0}^{N} b_n x(k-n) \qquad (2.13)$$

Значения выходных отсчетов свертки для любого аргумента k определяются текущим и "прошлыми" значениями входных отсчетов. Такой фильтр называется нерекурсивным цифровым фильтром (НЦФ). Интервал суммирования по n получил название "окна" фильтра. Окно фильтра составляет N+1 отсчет, фильтр является односторонним каузальным, т.е. причинно обусловленным текущими и "прошлыми" значениями входного сигнала, и выходной сигнал не опережает входного. Каузальный фильтр может быть реализован физически в реальном масштабе времени. При k<n, а также при k<m для фильтра, проведение фильтрации возможно только при задании начальных условий для точек x(-k), k = 1,2,..,N, и y(-k), k = 1,2,..,M. Как правило, в качестве начальных условий принимаются нулевые значения или значения отсчета x(0), т.е. продление отсчета x(0) назад по аргументу.

При обработке данных на ЭВМ ограничение по каузальности

снимается. В программном распоряжении фильтра могут находиться как "прошлые", так и "будущие" значения входной последовательности отсчетов относительно текущей точки вычислений k, при этом уравнение будет иметь вид:

$$y(k) = \sum_{n=-N'}^{N} b_n x(k-n) \qquad (2.14)$$

При N'=N фильтр называется двусторонним симметричным. Симметричные фильтры, в отличие от односторонних фильтров, не изменяют фазы обрабатываемого сигнала.

Техника выполнения фильтрации не отличается от техники выполнения обычной дискретной свертки двух массивов данных.

Представим, что на одной полоске бумаги выписаны по порядку сверху вниз значения данных $x(k) \equiv s_k$. На второй полоске бумаги находятся записанные в обратном порядке значения коэффициентов фильтра $b_n \equiv h_n$ (обозначение h для коэффициентов операторов НЦФ является общепринятым). Для вычисления $y_k \equiv y(k)$ располагаем вторую полоску против первой таким образом, чтобы значение h_0 совпало со значением s_k, перемножаем все значения h_n с расположенными против них значениями s_{k-n}, и суммируем все результаты перемножения. Результат суммирования является выходным значением сигнала y_k. Сдвигаем окно фильтра - полоску коэффициентов h_k, на один отсчет последовательности s_k вниз (или массив s_k сдвигаем на отсчет вверх) и вычисляем аналогично следующее значение выходного сигнала, и т.д.

Описанный процесс является основной операцией цифровой фильтрации, и называется сверткой в вещественной области массива данных x(k) с функцией (оператором) фильтра b_n (массивом коэффициентов фильтра). Для математического описания применяется также символическая запись фильтрации:

$$y(k) = b(n) * x(k-n) \qquad (2.15)$$

Сумма коэффициентов фильтра определяет коэффициент передачи

(усиления) средних значений сигнала в окне фильтра и постоянной составляющей в целом по массиву данных (с учетом начальных и конечных условий). Как правило, сумма коэффициентов фильтра нормируется к 1.

Рекурсивная фильтрация. Фильтры, которые описываются полным разностным уравнением, принято называть рекурсивными цифровыми фильтрами (РЦФ), так как в вычислении текущих выходных значений участвуют не только входные данные, но и значения выходных данных фильтрации, вычисленные в предшествующих циклах расчетов. С учетом последнего фактора рекурсивные фильтры называют также фильтрами с обратной связью, положительной или отрицательной в зависимости от знака суммы коэффициентов a_m. По существу, полное окно рекурсивного фильтра состоит из двух составляющих: нерекурсивной части b_n, ограниченной в работе текущими и "прошлыми" значениями входного сигнала (при реализации на ЭВМ возможно использование и “будущих” отсчетов сигнала) и рекурсивной части a_m, которая работает только с "прошлыми" значениями выходного сигнала.

Выводы

Во второй главе получены следующие основные результаты:

1. Анализированы ключевые операции цифровой обработки сигналов.

2. Исследованы спектральные методы обработки сигналов.

3. Рассмотрены методы фильтрации сигналов с использованием спектральных методов.

ГЛАВА 3. АНАЛИЗ И ВЫБОР УСТРОЙСТВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

3.1. Архитектура специализированных процессоров обработки сигналов

В конце 70-х годов появились первые специализированные процессоры, предназначенные для выполнения алгоритмов цифровой обработки сигналов (ЦОС) в реальном масштабе времени. Часто в литературе такие микропроцессоры называются сигнальными микропроцессорами (СМП) или цифровыми процессорами обработки сигналов (ЦПОС). Появление СМП было обусловлено несколькими причинами. Развитие теории ЦОС привело к появлению различных алгоритмов, которые могли практически использоваться во многих областях техники. Среди них можно отметить алгоритмы нерекурсивной и рекурсивной фильтрации, быстрое преобразование Фурье (БПФ) и другие. Первоначально внедрение этих методов основывалось на использовании интегральных микросхем малой и средней степени интеграции. При этом был достигнут качественный скачок по отношению к аппаратуре с аналоговой обработкой. Достоинства ЦОС позволили добиться высокой точности обработки сигналов, стабильности и повторяемости характеристик, высокой надежности аппаратуры. Однако использование ИМС не позволило снизить массогабаритные показатели аппаратуры, поэтому возникла необходимость реализовать данные методы на более современной элементной базе.

Другим важным моментом предшествовавшим появлению СМП явилось развитие технологии изготовления микросхем. Именно появление микронных и субмикронных технологий позволило создать СМП с несколькими десятками тысяч транзисторов на кристалле, потреблением мощности порядка одного ватта и быстродействием более 5-и миллионов операций в секунду. Эти тенденции развития науки и техники стимулировались постоянно возраставшими потребностями практики. В свою очередь практика все время ориентировалась на достижения науки и

техники. Такой взаимный процесс в конце концов и привел к появлению СМП. Уже первые СМП показали их значительные преимущества для реализации алгоритмов ЦОС перед ИМС средней и малой степени интеграции и универсальными микропроцессорами. Поэтому, в 80-е годы процесс улучшения характеристик СМП проходил очень быстрыми темпами. В настоящее время за рубежом выпускается несколько десятков различных СМП, находящих широкое применение в различных областях техники. Можно предположить, что в ближайшем будущем СМП станут одной их основных элементных баз разработчиков новой техники и, в частности, средств связи.

Алгоритмы ЦОС используются для выполнения таких операций, как фильтрация, формирование сигналов, выделение сигналов на фоне помех, распознавание образов и многих других. Ранее эти задачи в основном решались с помощью аналоговой обработки, но данные методы имеют множество недостатков, а именно: невысокую точность преобразований, нестабильность характеристик, низкую надежность, и другие. Использование методов ЦОС позволяет добиться качественного улучшения характеристик реализуемой аппаратуры. Поэтому внедрение методов цифровой обработки сигналов является насущной необходимостью во многих областях техники таких, как связь, обработка речи и изображений, распознавание образов, измерительная техника, радиолокация и многих других. Программная реализация алгоритмов ЦОС с помощью СМП дает дополнительные преимущества по сравнению с их жесткой реализацией на ИМС различной степени интеграции. В первую очередь к ним относится возможность существенного снижения массогабаритных показателей аппаратуры. Другое важное преимущество заключается в многофункциональности данного метода реализации: изменение программы приводит и к полному изменению алгоритма обработки. Кроме того, существенно упрощаются этапы разработки и отладки аппаратуры. Выгодно использовать СМП и с экономической точки зрения, поскольку происходит

значительное снижение цены как разработки, так и изготовления аппаратуры. Из всего вышесказанного следует вывод, что СМП являются перспективной элементной базой для многих областей техники. Стандартная схема применения СМП показана на рис.3.1.

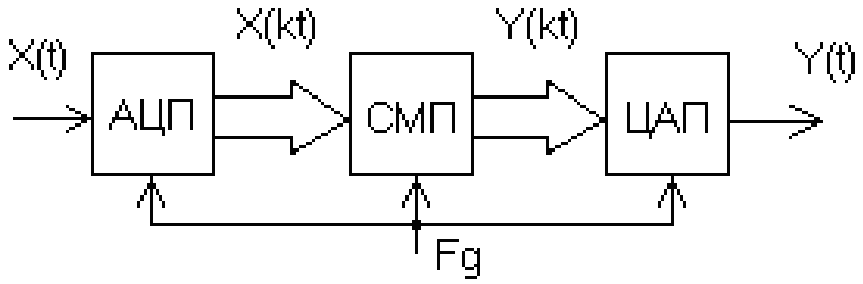

Рис. 3.1. Стандартная схема применения СМП

Аналоговый сигнал X(t) преобразуется в цифровую форму в аналого-цифровом преобразователе (АЦП) и в виде последовательности многоразрядных двоичных слов X(kT) поступает в СМП. СМП производит преобразование входной последовательности в соответствии с определенным алгоритмом ЦОС. Выходная последовательность двоичных слов Y(kT) поступает в цифроаналоговый преобразователь (ЦАП), где формируется выходной аналоговый сигнал Y(t). Следует отметить, что данная схема условна, поскольку у СМП в зависимости от особенностей аппаратуры может быть либо цифровой вход, либо цифровой выход, либо и цифровой вход, и цифровой выход. Но независимо от этого основное назначение СМП заключается в выполнении в реальном масштабе времени алгоритмов ЦОС.

В настоящее время выпускается большое число СМП. Среди них можно выделить СМП фирмы "Texas Instruments Incorporated", Motorola и Analog Devices.

Выбор цифрового процессора обработки сигналов. В последние годы вопрос выбора процессора ЦОС для конкретного приложения становится важным, поскольку число доступных процессоров продолжает расти. В число специфических факторов, которые следует рассмотреть при выборе процессора ЦОС для данного приложения, входят архитектурные

особенности, скорость выполнения, тип арифметики и длина слова.

Архитектурные особенности. Ключевыми характеристиками процессоров является размер встроенной памяти, наличие специальных команд и возможности ввода-вывода. Наличие встроенной памяти – необходимое требование в большинстве приложений ЦОС реального времени, поскольку это означает быстрый доступ к данным и быстрое исполнение программы. Для приложений со строгими требованиями к памяти (цифровое аудио, система Dolby, факс-модем, кодирование-декодирование MPEG) размер внутренней памяти ОЗУ может стать решающим фактором при выборе процессора. Если внутренней памяти недостаточно, ее можно дополнить высокоскоростной внешней памятью, хотя это может привести к увеличению стоимости системы. Для приложений, требующих быстрых и эффективных вычислений или обмена потоками данных с внешним миром, весьма важны такие средства ввода-вывода, как интерфейсы к АЦП и ЦАП, возможность прямого доступа к памяти и поддержка многопроцессорной обработки. В зависимости от приложения важен богатый набор специальных команд поддержки операций ЦОС, например, возможность организации циклов с нулевыми служебными издержками, специализированные команды ЦОС и круговая адресация.

Скорость выполнения. Поскольку большинство задач ЦОС требует срочного решения, важной мерой производительности является скорость процессоров ЦОС. Традиционно двумя основными единицами измерения этой величины являются тактовая частота процессора в мегагерцах и число выполняемых команд в миллионах команд за секунду (Million Instructions per Second - MIPS) или, если используются процессоры ЦОС с плавающей запятой, в миллионах операций с плавающей запятой в секунду (Million Floating-point Operations per Seconds – MFLOPS). Впрочем, подобные меры могут в некоторых случаях не подходить из-за значительных отличий в принципах работы различных процессоров ЦОС, большинство из которых может выполнять несколько операций в одной машинной команде.

Альтернативная мера основана на скорости выполнения контрольных алгоритмов ЦОС – например БПФ.

Тип арифметики. Арифметика с плавающей запятой – это естественный выбор в приложениях с широкими и переменными требованиями к динамическому диапазону (динамический диапазон – разность между наибольшим и наименьшим уровнем сигнала, который можно представить, или как разность между наибольшим сигналом и минимальным уровнем шума, измеренным в децибелах). Процессоры с фиксированной запятой предпочтительны с точки зрения низкой стоимости, они подходят для массового производства (например, сотовые телефоны и компьютерные дисководы). Процессоры с плавающей запятой дороже процессоров с фиксированной запятой, хотя в последние годы разница в цене существенно уменьшилась. Отметим также, что большинство существующих процессоров ЦОС с плавающей запятой также поддерживают арифметику с фиксированной запятой.

Длина слова. Этот параметр определяет, насколько точно можно представить параметры и результаты операций ЦОС. Вообще чем длиннее слово данных, тем меньше ошибки при цифровой обработке сигнала. В процессорах ЦОС с фиксированной запятой, нацеленных на рынок телекоммуникаций, обычно используются слова 16-битовой длины (например, TMS320C54x), тогда как в процессорах, нацеленных на аудиоприложения высокого качества, используются 24 бит (например, Motorola DSP56300) или 32 бит (например, ADSP-TS101). В последние годы отмечается тенденция к использованию большего числа бит для АЦП и ЦАП, поскольку стоимость подобных устройств постоянно снижается.

3.2. Выбор сери сигнального процессора

При тестировании использованы две программ: программа, реализующая фильтр с конечной импульсной характеристикой (КИХ-фильтр) при операциях с блоками данных, представленными вещественными числами

(Real Block FIR Filter) и тестовая программа прямое и обратное преобразования Хаара (ПХ). В результате выполнения программ Real Block FIR Filter, оптимизированных для указанных типов процессоров, было определено время их выполнения, а также вычислен показатель “быстродействие”. Кроме того, был определен объем памяти, необходимый для выполнения тестовых программ ПХ.

Для каждой тестовой программы вычисляется показатель “быстродействие”, а также определяются: время ее выполнения, число затрачиваемых машинных тактов, эффективность использования энергоресурсов и памяти.

TMS 320 C 6 4x x — высокопроизводительные сигнальные процессоры с фиксированной точкой, созданные на базе архитектуры VLIW (Very Long Instruction Word). Первый процессор этого семейства был анонсирован в 2000 году [26,28,30,31]. В 2003 году начался серийный выпуск процессоров TMS320C6414, TMS320C6415, TMS320C6416. Тактовая частота этих процессоров достигает 1000 МГц (к примеру, TMS320C6416TGLZ1), а напряжение питания процессорного ядра составляет 1,2...1,4 В. Стоимость процессора TMS320C6416TGLZ1 — 247 долларов США [28, 30]. Кроме того, начиная с 2003 года, производятся процессоры TMS320DM640, TMS320DM641 и TMS320DM642, разработанные специально для применения в мультимедийных приложениях, а в 2005 году начался серийный выпуск новых сигнальных процессоров TMS320DM643. Эти процессоры содержат встроенные видеопорты (TMS320DM642 имеет три видеопорта) и один или два многоканальных последовательных порта MsBSP, предназначенных для обмена данными с источниками аудиосигналов. Кроме того, процессоры TMS320DM640/641/642/643 имеют 64-канальный контроллер прямого доступа к памяти (DMA), хост- и PCI-интерфейсы. При тактовой частоте 720 МГц пиковая производительность TMS320DM642 составляет 5760 MIPS (Million Instructions per Second – миллион инструкций в секунду). Стоимость самого мощного процессора

TMS320DM642GNZ720 с тактовой частотой 720 МГц — 67,79 долларов США [28, 30]. Соответственно менее мощные процессоры имеют меньшую стоимость.

Базовое процессорное ядро TMS320C64x x содержит восемь операционных блоков: два блока MAC и шесть ALU, четыре из которых используются для арифметических вычислений, а два — для вычислений адресов. В вычислительном ядре TMS320C64xx, как и в ранее выпущенных процессорах TMS320C62xx, выполняются операции с 8-, 16-, 32- и 40-разрядными числами, а, кроме того, появилась возможность работы с 64 -разрядными. В течение одного цикла могут выполняться четыре операции умножения с 16-разрядными и восемь операций с 8-разрядными числами. В процессе вычислений все операционные блоки могут использоваться одновременно, что дает возможность выполнять параллельно восемь 32-разрядныхинструкций. Ориентированные на применение в высокопроизводительных телекоммуникационных системах сигнальные процессоры TMS320C6416 и TMS320C6418 содержат встроенный сопроцессор для реализации декодера Витерби (Viterbi decoder Co-Processor – VCP). Процессор TMS320C6416, кроме того, имеет еще один встроенный сопроцессор TCP (Turbo Decoder Coprocessor). К периферийным устройствам процессоров TMS320C64xx относятся: 16- или 32-разрядный хост-порт, многоканальный контроллер DMA, контроллер PCI-шины, последовательные порты MsBSP, три 32-разрядных таймера и другие. В табл.3.1 приведены основные параметры сигнальных процессоров семейства TMS320C64xx [28, 30].

Летом 2005 года фирма Texas Instruments анонсирова ла новые изготовленные по CMOS-технологии 90 нм высокопроизводительные сигнальные процессоры TMS320C6455 с тактовой частотой до 1000 МГц и максимальной производительностью до 8000 MIPS [31]. При тактовой частоте 1000 МГц и работе с 16-разрядными числами в процессорном ядре может выполняться до 8000 млн операций умножения в секунду. Новые

процессоры созданы на базе модифицированного процессорного ядра, получившего название C64+. Новое ядро позволило увеличить суммарную вычислительную мощность процессоров этого типа.

Таблица 3.1. Основные параметры сигнальных процессоров TMS320C64xx

Наименование параметра	**Тип процессора TMS320…**					
	C6414	**C6415**	**C616**	**DM641**	**DM642**	**DM643**
Макс. тактовая частота, МГц	100			600	720	600
Максимальная производительность, MIPS	8000			4800	5760	4800
Объем встроенной памяти, Мбайт	1,032			0,16	0,288	0,288
Число каналов DMA	64			64		
Число таймеров	3			3		
Напряжение питания ядра (схем ввода/вывода), В	1,2 (3,3)			1,2 (3,3)	1,2…1,4 (3,3)	
Потребляемая мощность, Вт (тактовая частота, МГц)	1,7 (720)			1,9 (600)	2,15 (720)	1,9 (600)
Интерфейсы (число)	HPI 16-/32-разр.; MsBSP(3)	PCI; HPI 16-/32-разрядн.; MsBSP(2)		HPI 16-раз-рядн.; EMAC; два видео-порта	HPI 32-раз-рядн.; EMAC; PCI; три видеопорта	HPI 32-раз-рядн.; EMAC; два видео-порта
Число выводов и тип корпуса (размеры, мм)	532-BGA (23Ч23)			548-BGA (27Ч27)		
Диапазон рабочих температур, °С	0...90/-40...105					
Стоимость, $ в партии 1000 шт.	85,85...213,63	90,37...224,87	99,41...247,36	30,77...33,84	42,89...67,79	31,95...34,95

Сигнальные процессоры TMS320C6455 имеют объем встроенной памяти 2128 кбайт и содержат высокоскоростные коммуникационные порты Serial RapidIO и Gigabit EMAC (Ethernet MAC), а также контроллер PCI-шины

(тактовая частота 33,66 МГц, 32-разрядная шина данных) и 16-/32-разрядный контроллер HPI (Host-Port Interface). Пиковая скорость передачи данных через порт Serial RapidIO составляет 3,125 Гбит/с. Как и Link-порты, применяемые в процессорах ADSPTS201/202/203, порт Serial RapidIO предназначен для использования в первую очередь для обмена данными в мультипроцессорных системах. Четыре независимых полнодуплексных канала Serial RapidIO дают возможность построить многопроцессорную систему, состоящую из двумерного массива процессоров. Кроме того, имеются два традиционных порта MsBSP и порт I2C. Контроллер внешней памяти поддерживает обмен данными с памятью типа SR AM, ROM, FLASH, SBSRAM объемом до 32 Мбайт. Кроме того, новые процессоры TMS320C6455 поддерживают работу с 32-разрядной памятью типа DDR2-500 SDRAM объемом до 256 Мбайт. Как и все процессоры семейства TMS320C64xx, новые содержат 64-канальный контроллер прямого доступа к памяти. Для формирования сигналов тактовых частот в TMS320C6455 реализованы две системы ФАПЧ (PLL). Одна из них (программно управляемая PLL1) генерирует тактовый сигнал частотой до 1000 МГц для работы процессорного ядра. Вторая (PLL2) с фиксированным коэффициентом умножения 10 используется для формирования сигналов тактовых частот необходимых для работы контроллеров DDR2 SDRAM и Ethernet MAC. Основные параметры сигнальных процессоров TMS320C64xx приведены в табл. 3.1. Напряжение питания процессорного ядра составляет 1,2 В, схем ввода/вывода – 3,3 В. Периферийные контроллеры Serial RapidIO, DDR2 SDRAM и Ethernet MAC имеют соответственно напряжение питания 1,2 и 1,5/1,8 В. Сигнальные процессоры TMS320C6455 выпускаются в корпусе типа 697-PBGA (размерами 24×24 мм с шагом сферических выводов 0,8 мм) и предназначены для работы в диапазоне температур от 0 до 90°C. Стоимость новых процессоров от 202 до 292 долларов США [30].

ADSP-TS201/202/203 — сигнальные процессоры семейства Tiger SHARC. В это семейство включены процессоры первого (ADSPTS101) и

второго поколения (ADSPTS201/202/203). В табл.3.2 приведены основные параметры сигнальных процессоров семейства Tiger SHARC [29]. Архитектура вычислительного ядра ADSP-TS201/202/203 создана на базе архитектуры процессорного ядра ADSP-TS101.

Таблица 3.2. Основные параметры сигнальных процессоров семейства Tiger SHARC

Наименование параметра	**тип процессора ADSP-…**			
	TS101S	**TS201S**	**TS202S**	**TS203S**
Макс. тактовая частота, МГц	300	600	500	
Максимальная производительность, MIPS	2400	4800	4000	
Объем (тип) встроенной памяти, Мбайт	0,75 (SRAM)	3 (DRAM)	1,5 (DRAM)	0,5 (DRAM)
Разрядность внешней шины данных, бит	32/64	32		
Число каналов DMA	14	10		
Скорость обмена данными через Link-порт, Гбайт/с	0.25	1.0	0.5	
Интерфейс (число)	Link (4)			Link (2)
Напряжение питания ядра (схем ввода/вывода), В	1,2 (3,3)	1,05…1,2 (2,5); 1,2 (2,5)	1,0 (2,5)	
Ток потребления, типов., А				
Число выводов и тип корпуса (габаритные размеры, мм)	484-PBGA (19Ч19); 625-PBGA (27Ч27)	576-PBGA (25Ч25)		
Число таймеров	2			
Диапазон рабочих температур, °С	-40…85			
Стоимость, $ в партии 1000 шт.	159…193	205…223	149	47

Особенность процессоров ADSP-TS201/202/203 — большой объем встроенной памяти типа DRAM [29]. К примеру, объем памяти сигнального процессора ADSP-TS201 составляет 24 Мбит. Все процессоры семейства ADSP-TS20x содержат: высокопроизводительное вычислительное ядро,

которое относится к системам типа SIMD (Single Instruction Multiple Data); большой объем динамической памяти; мощные периферийные контроллеры Link-портов (до четырех 8-разрядных полнодуплексных портов), поддерживающие через ка ж дый из портов обмен данными со скоростью до 1 Гбайт/с. Архитектура процессорного ядра сочетает все достоинства RISC (Reduced Instruction Set Computer), VLIW и тра диционной архитектуры цифровых сигнальных процессоров. Для обработки потоков данных в этих сигнальных процессора х имеются два полноценных вычислительных устройства, содержащих: ALU, умножитель/накопитель 32 × 32 разряда с 8 0 –разрядным аккумулятором, 64-разрядное устройство сдвига, регистровый файл объемом тридцать два 32-разрядных регистра. Кроме того, имеются еще два дополнительных целочисленных 32-разрядных ALU: JALU и K ALU. Таким образом, четыре ALU позволяют выполнять параллельно четыре операции с 32-разрядными числами. Вместе с тем, хотя сигнальные процессоры семейства Tiger SHARC относятся к 32-разрядным процессорам с плавающей точкой, возможности их архитектуры и организация работы вычислительных устройств позволяют выполнять также операции с 8-, 16-, 32- и 64-разрядными числами с фиксированной точкой. В течение каждого цикла в процессорном ядре может выполняться четыре инструкции и при этом совершаться до двадцати четырех операций с 16-разрядными числами с фиксированной точкой или шесть операций с числами с плавающей точкой. Возможность работы с данными, представленными в разных форматах, позволяет значительно увеличить производительность этих сигнальных процессоров. Кроме того, дополнительные целочисленные ALU могут работать в двух режимах. В первом устройства JALU и KALU используются в качестве генераторов адресов при косвенной адресации к встроенной и внешней памяти, во втором они используются для целочисленной обработки данных (выполнения операций сложения, вычитания и т.п.). Максимальная производительность самого мощного ADSP-TS201 состав ляет 4800 MMACS (Million Multiplication Accumulation per Second — миллионов операций

умножения с накоплением в секунду). Архитектура сигнальных процессоров TigerSHARC ориентирована в первую очередь на создание высокопроизводительных мультипроцессорных систем.

Быстродействие

На рис. 3.2 приведено время выполнения программы Real Block FIR Filter для разных типов сигнальных процессоров. В этом тесте данные и программный код размещаются в кэш-памяти процессоров. Как следует из приведенных данных, сигнальный процессор TMS320C6414 с тактовой частотой 720 МГц затрачивает на выполнение программы Real Block FIR Filter чуть больше времени, чем процессор ADSP-TS201S, работающий с тактовой частотой 600 МГц. По сравнению с MSC8103 (300 МГц) скорость выполнения этого теста процессором TMS320C6414 более чем в два раза выше. Это обусловлено не только высокой так товой частотой (720 МГц), но и возможностью параллельного выполнения нескольких операций. Для выполнения данного теста и в MSC8103, и в процессоре TMS320C6414 затрачивается примерно одинаковое число машинных тактов, однако тактовая частота TMS320C6414 более чем в два раза выше, чем в MSC8103, поэтому процессор TMS320C6414 и показал лучший результат в сравнении с MSC8103. Программа Real Block FIR Filter отличается тем, что в процессе ее выполнения необходимо производить большое число операций умножения с суммированием.

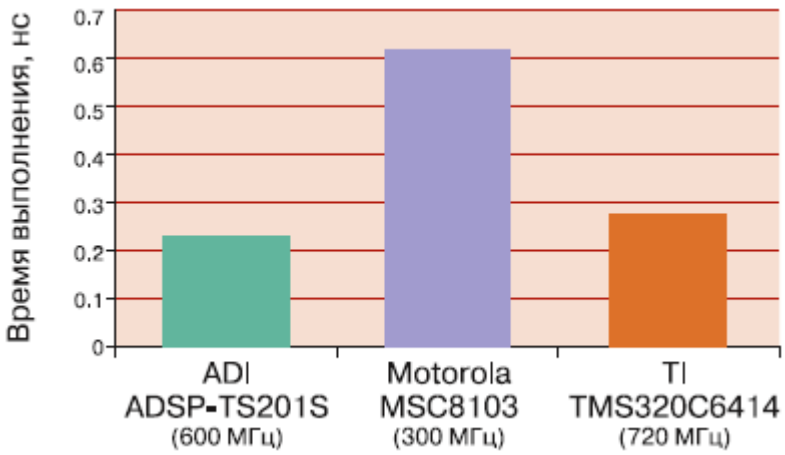

Рис. 3.2. Время выполнения программы Real Block FIR Filter

Однако не только эта особенность существенно влияет на конечный результат. К примеру, в сигнальном процессоре ADSP-TS201S может в течение одного цикла выполняться восемь 16-разрядных операций умножения. Это вдвое больше, чем в MSC8103 и TMS320C6414. Вместе с тем в ADSP-TS201S затрачивается много “пустых” машинных тактов на упорядочивание результатов умножения, что, в конечном счете, приводит к снижению уровня “полезной” производительности этого процессора. Именно поэтому результат, полученный в данном тесте, не такой высокий, какого можно было бы ожидать от процессора ADSP-TS201S, сравнивая его с другими рассматриваемыми в статье процессорами только по показателю производительности, выраженному в числе операций умножения с накоплением, выполняемых в секунду (MMACS).

Показатель “быстродействие”

Чтобы получить числовое значение показателя “быстродействие”, используется результаты время выполнения тестовой программы Real Block FIR Filter. Числовые значения показателей “быстродействие” для разных типов сигнальных процессоров приведены на рис. 3.3.

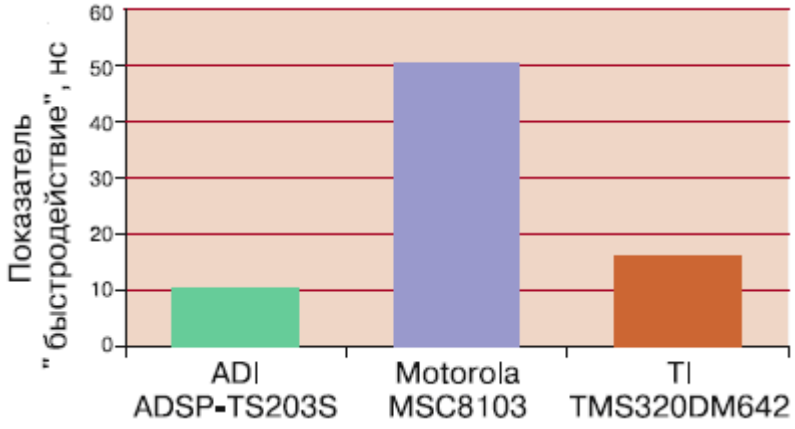

Рис. 3.3. Показатели “быстродействие” сигнальных процессоров

Принимая во внимание полученные результаты, сигнальный процессор ADSP-TS203S с тактовой частотой 500 МГц по сравнению с другими

рассматриваемыми в работе процессорами имеет наилучший показатель “быстродействие”. Этот показатель ADSP-TS203S в пять раз лучше, чем процессора MSC8103 (300 МГц) и примерно на 30% лучше, чем TMS320DM642 (500 МГц). Объем встроенной памяти и число периферийных контроллеров, интегрированных на кристалле, в значительной мере оказывают влияние на общую стоимость. Однако эти факторы не учитываются при вычислении количественных показателей “быстродействие” для рассматриваемых в работе типов сигнальных процессоров.

Уровень энергопотребления

Уровень энергопотребления (Ватт/мкс) оценивается по количественному показателю, который получается в результате умножения типового значения потребляемой процессорами мощности на время выполнения программы Real Block FIR Filter. При определении этого показателя используются характеристики тех моделей процессоров из рассматриваемых семейств, которые отличается наилучшей эффективностью использования энергоресурсов. На рис.3.4 приведен уровень энергопотребления сигнальных процессоров при выполнении ими тестовой программы Real Block FIR Filter.

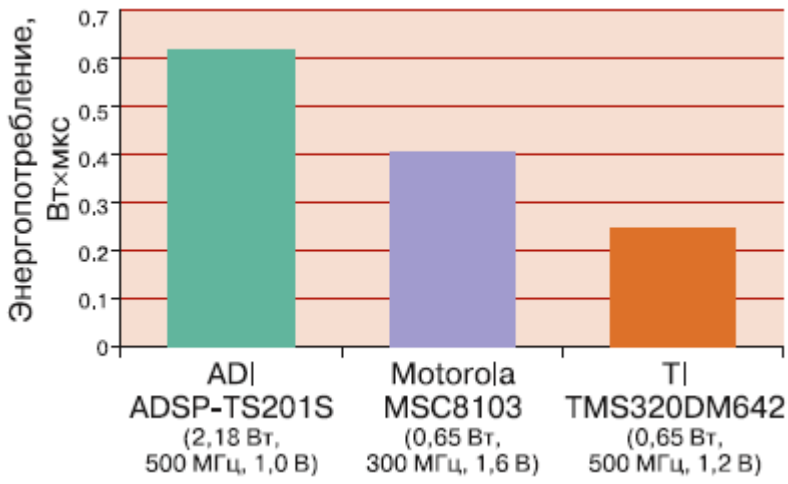

Рис. 3.4. Уровень энергопотребления сигнальных процессоров

Основываясь на данных, полученных в результате тестирования, можно отметить следующее. Хотя процессор TMS320DM6 42 имеет меньшее быстродействие по сравнению с ADSP-TS201S при работе с тактовой частотой 500 МГц, уровень его энергопотребления почти в 2,5 раза меньше, чем процессора ADSPTS201S, и почти в два раза меньше, чем MSC8103 (30 0 МГц). Вместе с тем процессор TMS 320 DM6 42 (500 МГц) имеет на 35% большее быстродействие почти такие же показатели по потребляемой мощности, что и MSC8103 (300 МГц).

Используемый объем памяти

Тестовая программа ПХ создана специально для оценки объема памяти. Программный код, необходимый для выполнения таких задач, как правило, занимает львиную долю в суммарном объеме памяти пользовательской программы. В то же время длительность выполнения таких программ составляет лишь малую часть общего времени выполнения пользовательской программы. Таким образом, проблема минимизации объема памяти, необходимого для хранения программ, ориентированных на решение задач управления/контроля, значительно более важная, чем минимизация времени ее выполнения. Главная цель при разработке тестовых программ ПХ — минимизировать используемый объем памяти. Этот подход в полной мере отражает общепринятый принцип, используемый прикладными программистами. Следует особо подчеркнуть, что полученные при использовании теста ПХ результаты никоим образом не связаны с объемом памяти, используемой при выполнении программ цифровой обработки сигналов.

На рис. 3.5 приведен объем используемой памяти, необходимый для выполнения тестовых программ ПХ для разных типов сигнальных процессоров. Большое различие в показателях для разных типов процессоров обусловлено в первую очередь отличиями в размерах исполняемых инструкций. Как видно из данных, приведенных на рис. 3.5, для выполнения тестовой программы ПХ сигнальные процессоры MSC810x (SC140)

используют наименьший объем памяти. Это обусловлено тем, что в этих процессорах используются как 16-разрядные, так и 32-разрядные инструкции. При выполнении этого теста используются преимущественно 16-разрядные инструкции. В системе команд, реализованной в процессорах ADSPTS201S и TMS320C64xx, используются инструкции, имеющие длину 32 разряда. Как результат, при выполнении данного теста эти процессоры имеют примерно одинаковые показатели, которые намного хуже в сравнении с MSC810x.

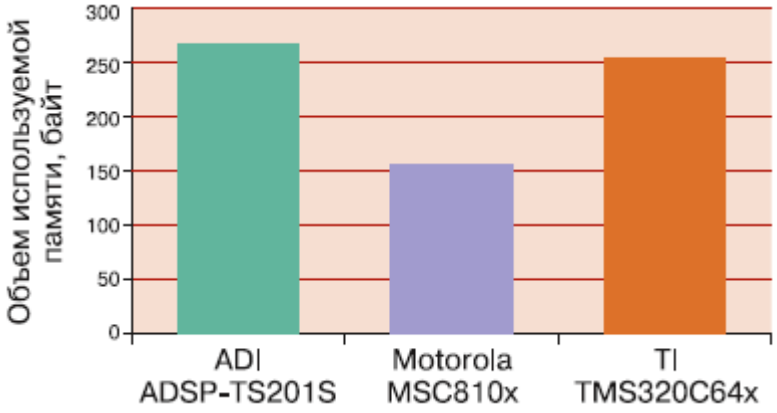

Рис.3.5. Объем используемой памяти

На рис. 3.6 приведены обобщенные показатели производительности рассматриваемых в работе процессоров, полученные в результате их тестирования.

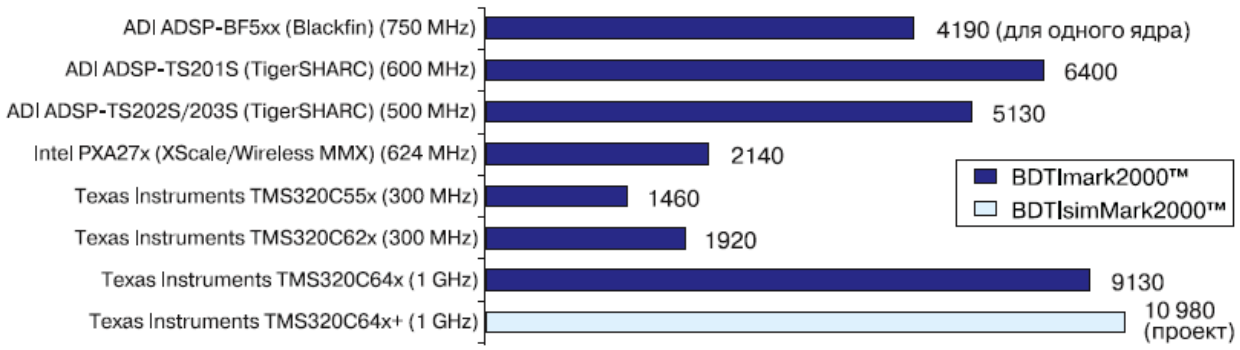

Рис. 3.6. Показатели производительности сигнальных процессоров разных типов

Для обработки сигналов с использованием линейных алгоритмов целесообразно выбрать процессора ADSP (типа TS201) от производителей Analog Devices, который удовлетворяет все высшее сказанных требование.

3.3. Методы, алгоритмы и программные средства цифровой обработки сигналов для ADSP 102

Современные процессоры ЦОС по своим возможностям не уступают микропроцессорам общего назначения, таким, например, как Intel Pentium. Они обладают развитыми системами команд и широкими возможностями управления периферийными устройствами. Большое число программируемых ресурсов требует развитых средств разработки и отладки прикладных программ, которые позволяют эффективно использовать все предоставляемые возможности.

В настоящее время для наиболее популярных процессоров ЦОС предлагаются специальные интегрированные среды разработки, включающие в свой состав развитые редакторы исходного текста, средства ведения проектов, компиляторы, компоновщики и симуляторы.

Характер задач, решаемых на процессорах ЦОС, требует отладочных средств, позволяющих производить отладку в реальном масштабе времени. Интегрированные среды могут быть расширены для такой отладки при использовании соответствующей аппаратной поддержки в виде дополнительной аппаратуры. Как правило, такая аппаратура использует интерфейс JTAG, ставший индустриальным стандартом не только для процессоров ЦОС, но и для всех микропроцессорных приборов.

Наиболее популярные процессоры ЦОС универсального назначения - это семейства TMS320 (Texas Instruments) и ADSP (Analog Devices). Для разработки и отладки программ, выполняемых этими процессорами, предлагаются пакеты Code Composer Studio и VisualDSP++, соответственно.

Эти пакеты исполняются под Windows 9x/Me/NT/2000/XP и представляют собой полнофункциональные интегрированную среды (IDE),

которые обеспечивают создание проекта, редактирование исходных текстов, задание ключей компиляции и компоновки, и отладку прикладных программ в исходных текстах на языке Си и ассемблере. Также обеспечивается просмотр памяти в любом формате, в том числе и в графическом виде. Для многопроцессорных систем специальный менеджер обеспечивает общее управление всеми процессорами в системе – старт, стоп и т.д.

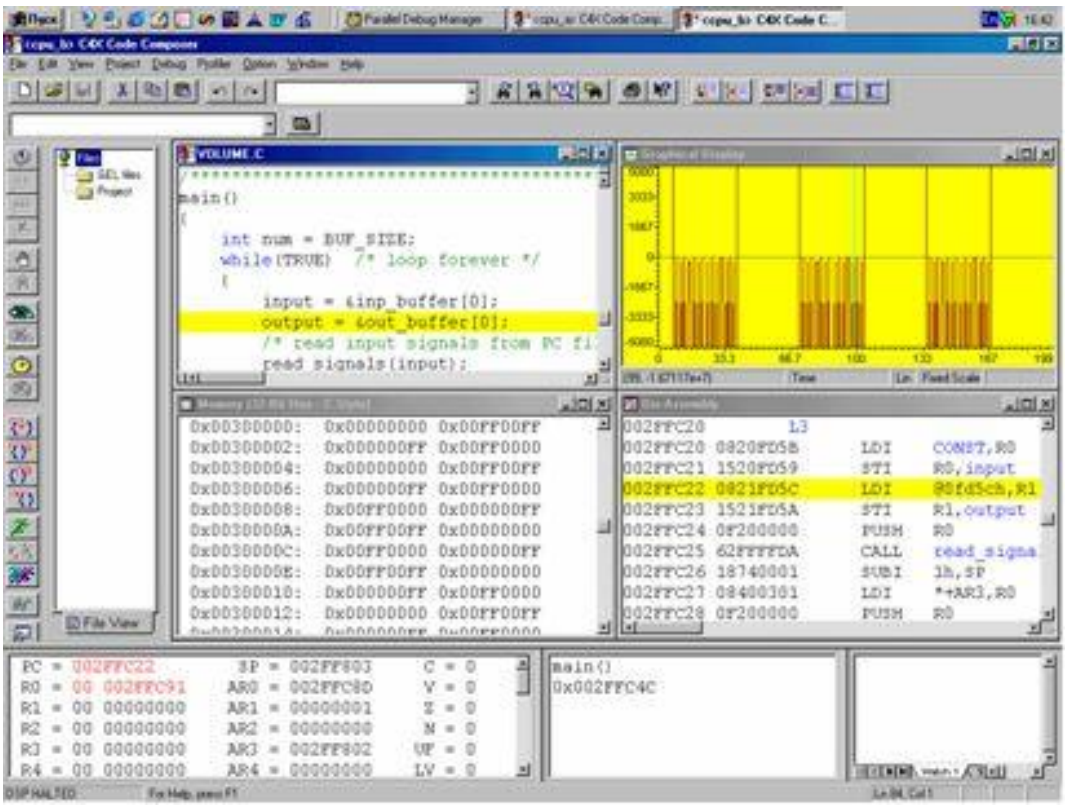

Рис.3.7. Внешний вид среды Code Composer

На рис.3.7. представлен внешний вид среды Code Composer. Это многооконное Windows-приложение. Можно в различных окнах производить редактирование исходного текста, просматривать содержимое памяти и регистров, производить настройку ключей компилятора и компоновщика и т.д. По команде из меню или с помощью горячих клавиш можно выполнять компиляцию и компоновку, после которых в отдельном окне отображается результат их выполнения.

После того как программа скомпилирована и скомпонована, требуется ее отладка. При отсутствии эмулятора можно воспользоваться программным симулятором, который позволяет выполнить программу, используя математическую (и логическую) модель соответствующего процессора ЦОС.

Для отладки прикладных программ в реальном времени требуется специальная аппаратура. Для процессоров серии TMS320 фирмой «Инструментальные системы» предлагается эмулятор EMU510PCI, который является полным аналогом эмулятора XDS-510, производимого фирмой Texas Instruments. Это специальная плата, устанавливаемая с любой свободный слот шины PCI персонального компьютера. С помощью специального кабеля она подключается к интерфейсу JTAG целевой системы, построенной на любом из процессоров семейства TMS320. Для отладки используется специальное программное обеспечение, входящее в состав Code Composer Studio.

Как было указано выше, для процессоров семейства ADSP предлагается программный пакет VisualDSP++.

На рис.3.8. представлен внешний вид среды VisualDSP++. Это, также как и Code Composer, многооконное Windows-приложение. Возможности пакета VisualDSP аналогичны возможностям Code Composer.

Для отладки прикладных программ на процессорах ADSP в реальном времени также требуется специальная аппаратура. Фирмой «Инструментальные системы» предлагается эмулятор EMU-ADPCI, полностью совместимый с эмулятором EZ-ICE фирмы Analog Devices. Это специальная плата, устанавливаемая с любой свободный слот шины PCI персонального компьютера. С помощью специального кабеля она подключается к интерфейсу JTAG целевой системы, построенной на любом из процессоров семейства ADSP. Для отладки используется специальное программное обеспечение, входящее в состав VisualDSP.

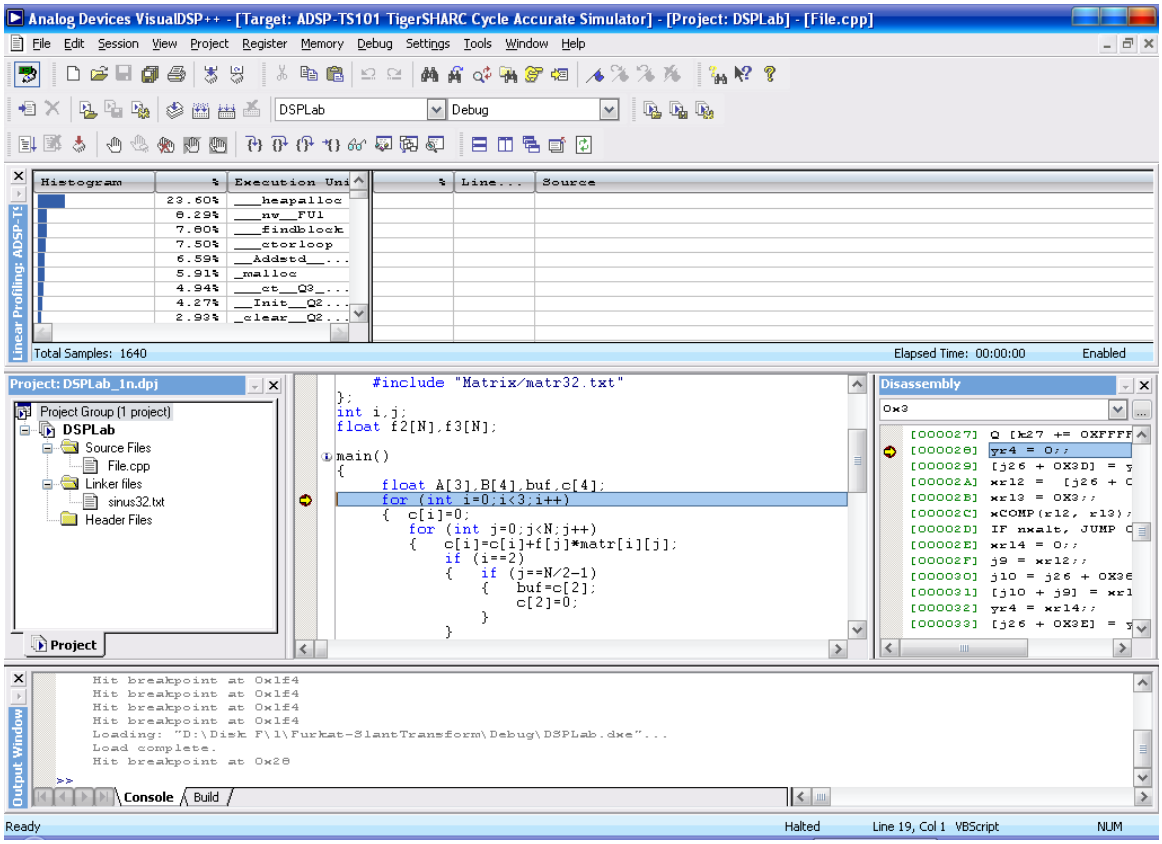

Рис.3.8. Внешний вид среды VisualDSP++

Наличие полного набора средств аппаратной и программной отладки весьма важно для любого проекта, основанного на DSP. Рассмотрим этапы проектирования программы выполнения алгоритма на ADSP-TS201 с применением средств отладки.

Первый шаг в процессе проектирования – это описание архитектуры системы, которое включает такую информацию, как тип процессора, периферийные устройства (внешняя память, кодеки, хост-процессор, каналы связи), конфигурацию. Эта информация помещается в файл, называемый LDF (файл описания связей).

Следующий шаг в процессе проектирования – генерирование выполняемой программы. Программа создана на языке высокого уровня C++. Программа разработанная на C, должна быть откомпилирована для получения кода на языке ассемблера. Нужно учитывать, что преимуществом использования языка C является простота программирования, в то время как результат компилирования такой программы не так эффективен, как при

программировании непосредственно на ассемблере. Язык ассемблера компании Analog Devices для DSP использует алгебраический синтаксис и достаточно прост при непосредственном использовании. В конце этапа компиляции компоновщик генерирует исполняемый файл.

Для конкретной реализации контрольных примеров по исследованию анализированного метода и алгоритма обработки будим использовать интегрированная среда разработки VisualDSP++, которая обладает следующими особенностями.

Среда разработки VisualDSP++ позволяет создавать и изменить исходные файлы, используя многовариантную подсветку синтаксиса языка, применять закладки, метод drag-and-drop, и другие стандартные приемы редактирования. Можно просмотреть и отредактировать файлы, создаваемые встроенными средствами автоматической генерации кода.

Среда VisualDSP++ позволяет использовать следующие инструменты разработки: компилятор C/C++, VIDL компилятор (компилятор специального языка интерфейса с ранее разработанными программными компонентами), ассемблер, компоновщик (линкер), загрузчик и симулятор – отладчик. Опции, определяющие варианты использования этих инструментальных средств, могут быть заданы как в диалоговых окнах, так и указаны в командной строке в виде ключей. Можно указать опции для каждого отдельного файла и для всего проекта в целом, а впоследствии их изменить.

Выводы

В третей главе получены следующие основные результаты:

1. Проделан анализ архитектур специализированных процессоров обработки сигналов.

2. Выбрана серия сигнального процессора.

3. Выбраны методы, алгоритмы и программные средства цифровой обработки сигналов для ADSP 102.

ГЛАВА 4. РАЗРАБОТКА ПРОГРАММ ДЛЯ ИССЛЕДОВАНИЯ АЛГОРИТМОВ И УСТРОЙСТВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

4.1. Реализация программ для фильтрации сигналов от помех с использованием КИХ-фильтров

На основе вышеописанного алгоритма нерекурсивной фильтрации сигналов разработаны программные средства. Разработка программы производилась на языке программирования С. С помощью разработанных программ проводины тестирование сигнальных процессоров, которые подробно дана информация в разделе 3.2. В качестве компилятора была выбрана платформа VisualDSP++.

Разработанная программа реализована в виде проекта. Платформа VisualDSP++ запускает проекта на виртуальном процессоре. Тип процессора выбираются из списка библиотек.

В качестве исходный информации в данной программе будет использоваться сигналы, полученные с помощью математических формул.

Проект программы включает в себя 5 пар основных файлов, из которых 3 – стандартные файлы, а 2 – файлы, разработанные:

Стандартные файлы, создающиеся автоматически:

1. DSPLab.dpj.
2. DSPLab.mak.
3. DSPLab.pcf.

Разработанные файлы:

1. File.cpp;
2. cache_macros.h.

Проект включает в себя 7 класс, из которых 6 классов являются стандартным для многодокументного приложения, 1 классов предназначен для работы с цифровыми сигналами:

Стандартные классы:

1. Класс CAboutDlg – предназначен для вывода на экран информации о программе и другой необходимой информации.
2. Класс CBMPApp – этот класс отвечает за всё приложение.
3. Класс CBMPDoc – предназначен для хранения и обработки данных каждого документа приложения.
4. Класс CBMPView – предназначен для представления и вывода на экран необходимых данных.
5. Класс CChildFrame – отвечает за каждое дочернее окно приложения.
6. Класс CMainFrame – отвечает за работу и функционирование главного окна приложения.

Класс, предназначенный для работы с цифровыми сигналами: Класс Cache_macros – здесь разработаны методы, осуществляющие преобразования сигнала.

Основная структура проекта программы предназначенной для преобразования Хаара имеет следующий вид:

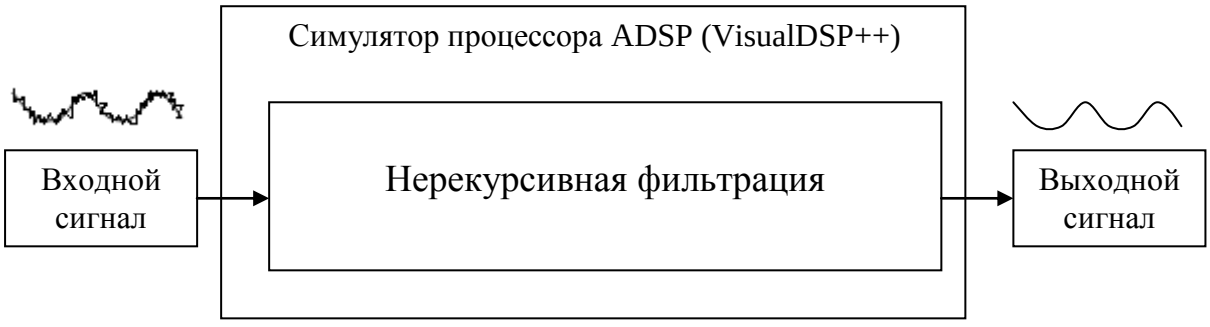

Рис. 4.1. Структура программы.

В блоке **Симулятор процессора ADSP (VisualDSP++)** вычисляется коэффициенты преобразования Хаара, в этапе прямое преобразование. С

помощью полученных коэффициентов обратно восстанавливается исходный сигнал.

4.2. Пример практической реализации на контрольном примере

Рассмотрим работы программы:

Для работы с программой нужно сгенерировать случайный сигнал с использованием тригонометрических функции sin или cos (рис.4.2). В нашем примере $f(x) = \sin(2\pi x) + \cos(\pi x / 5)$, число отсчетов $N=64$.

Рис.4.2. Генератор случайных сигналов

После получения исходного информации (цифровой сигнал) запускается программа симулятора (VisualDSP++) и откроется проект. После открытия проекта преобразование Хаара симулятор имеет следующий вид:

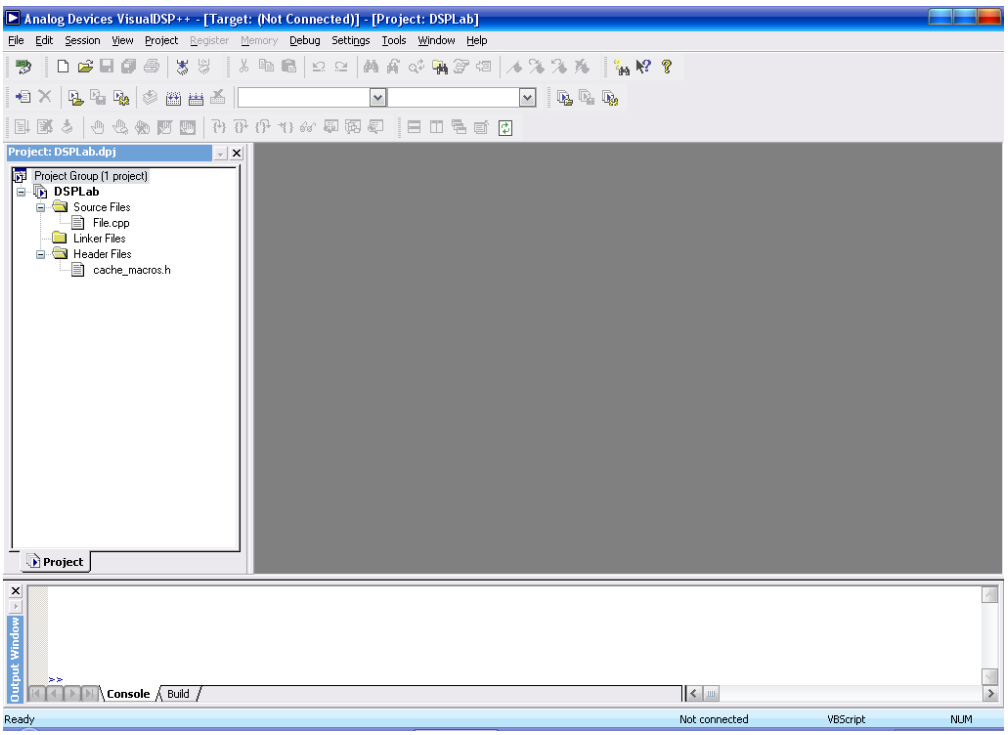

Рис.4.3.

Чтобы компилировать проекта нужно нажат кнопу Rebuild All (⌨) который расположен на панели инструментов. После компиляции симулятор имеет следующий вид:

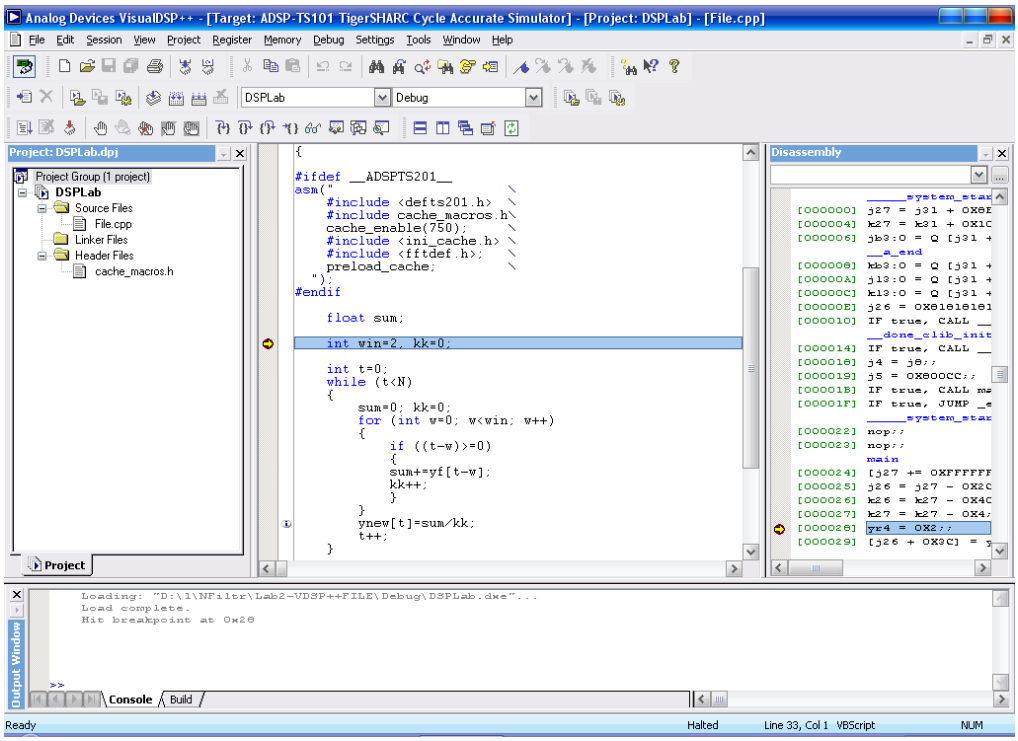

Рис.4.4.

На рис.4.4. показана структура проекта (окно радела Project: DSPLab.dpj), исходный код которого на языке C++, машинный код (окно раздела Disassembly) и окно результатов (Output Window).

С нажатием копки Run ( ) или с помощью F5 из клавиатуры программа запускается и результаты будут высвечиваться на окне Output Window (рис.4.5).

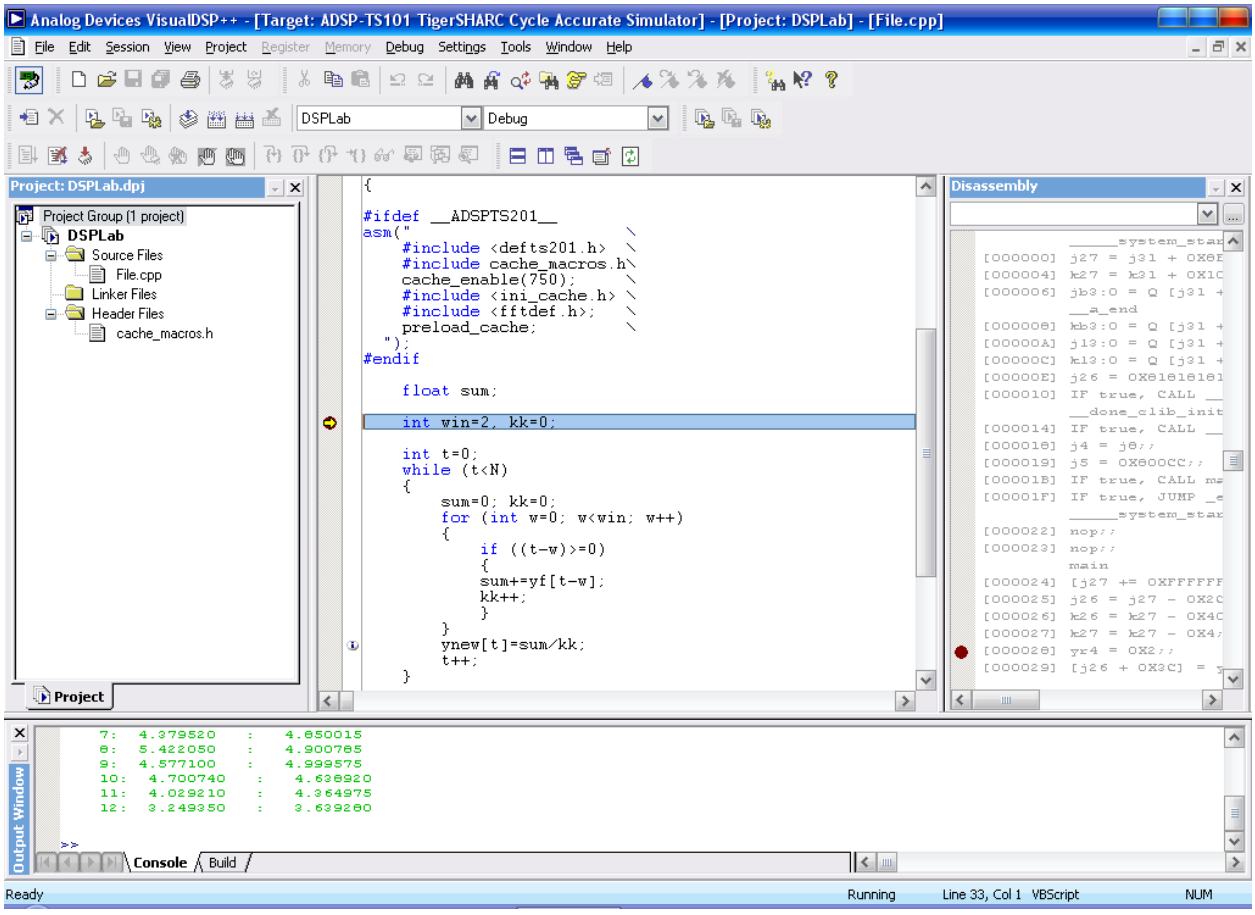

Рис.4.5.

Основные полученные результаты:

N: y[N] : ynew[N]

0: 1.000000 : 1.000000

1: 0.003354 : 0.501677

2: 2.907260 : 1.455307

3: 1.912080 : 2.409670

4: 4.430630 : 3.171355

5: 3.447670 : 3.939150

6: 5.320510 : 4.384090

7: 4.379520 : 4.850015

8: 5.422050 : 4.900785

9: 4.577100 : 4.999575

10: 4.700740 : 4.638920

11: 4.029210 : 4.364975

12: 3.249350 : 3.639280

13: 2.844150 : 3.046750

14: 1.274960 : 2.059555

15: 1.230920 : 1.252940

16: -0.932015 : 0.149452

17: -0.535551 : -0.733783

18: -3.041910 : -1.788730

19: -2.159420 : -2.600665

20: -4.736430 : -3.447925

21: -3.371980 : -4.054205

22: -5.757660 : -4.564820

23: -3.975060 : -4.866360

24: -5.948070 : -4.961565

25: -3.872870 : -4.910470

26: -5.275080 : -4.573975

27: -3.087140 : -4.181110

28: -3.836360 : -3.461750

29: -1.753260 : -2.794810

30: -1.845150 : -1.799205

31: -0.098018 : -0.971584

32: 0.402316 : 0.152149

33: 1.597530 : 0.999923

34: 2.572150 : 2.084840

35: 3.043970 : 2.808060

36: 4.343850 : 3.693910

37: 3.991000 : 4.167425

38: 5.459120 : 4.725060

39: 4.268620 : 4.863870

40: 5.760970 : 5.014795

41: 3.815680 : 4.788326

42: 5.216980 : 4.516330

43: 2.690880 : 3.953930

44: 3.923180 : 3.307030

45: 1.064500 : 2.493840

46: 2.088100 : 1.576300

47: -0.807845 : 0.640128

48: -0.000487 : -0.404166

49: -2.625530 : -1.313008

50: -2.020850 : -2.323190

51: -4.090740 : -3.055795

52: -3.667480 : -3.879110

53: -4.956370 : -4.311925

54: -4.698200 : -4.827285

55: -5.066290 : -4.882245

56: -4.971020 : -5.018655

57: -4.381540 : -4.676280

58: -4.464910 : -4.423225

59: -2.988210 : -3.726560

60: -3.281320 : -3.134765

61: -1.085620 : -2.183470

62: -1.626500 : -1.356060

63: 1.043560 : -0.291470

где,

N – число отсчетов,

y[N] – входная информация,

ynew[N] – выходная информация (результат обратного преобразования).

Теперь посмотрим все результаты на графиках и диаграммах (рис.4.6-10).

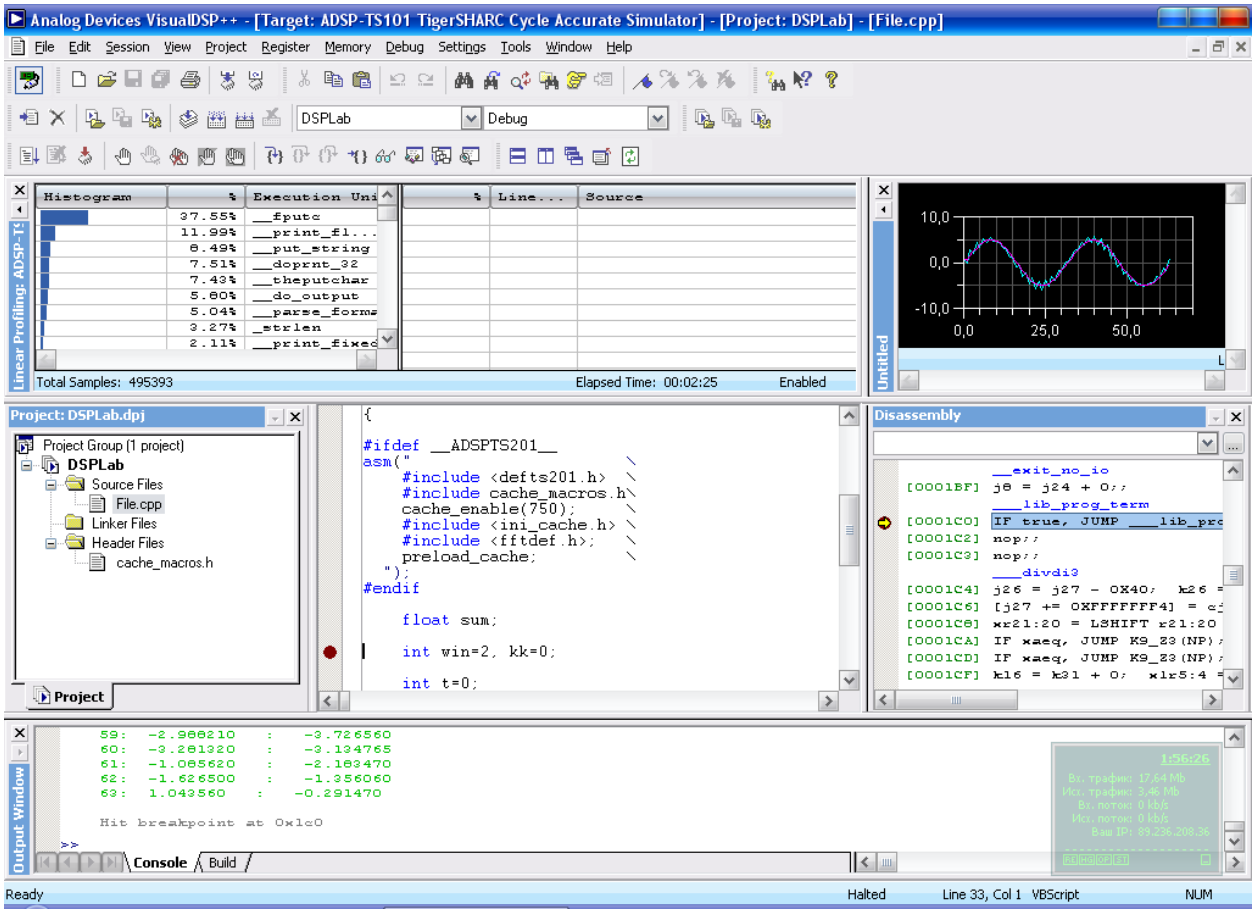

Рис.4.6. Графические представление результатов

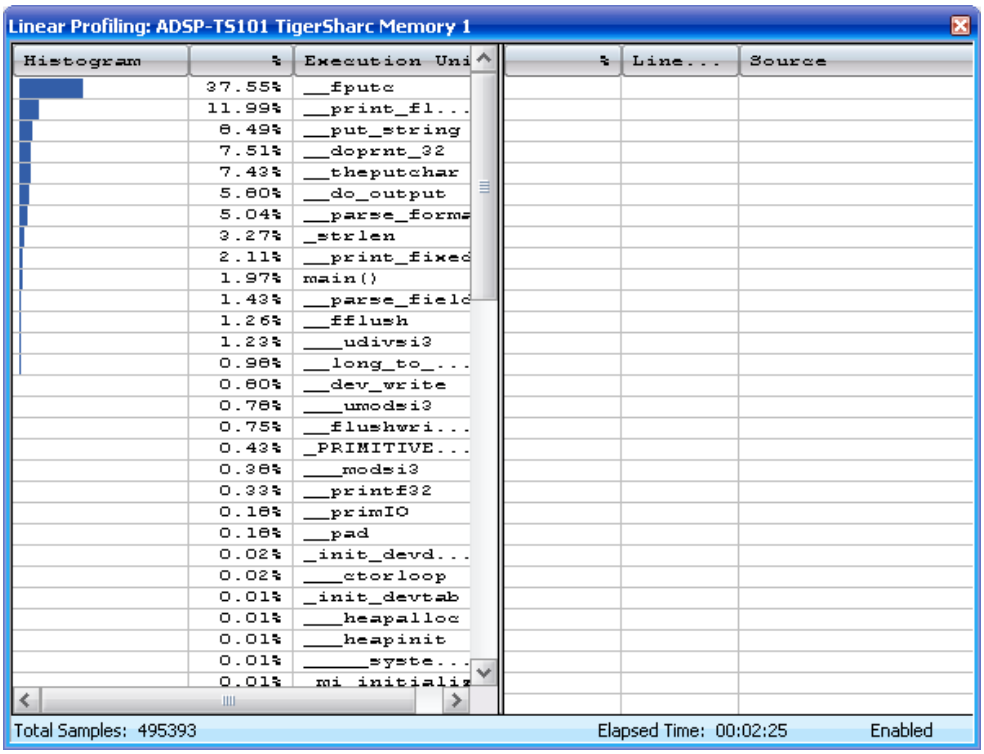

Рис.4.7. Диаграмма использование ресурсов

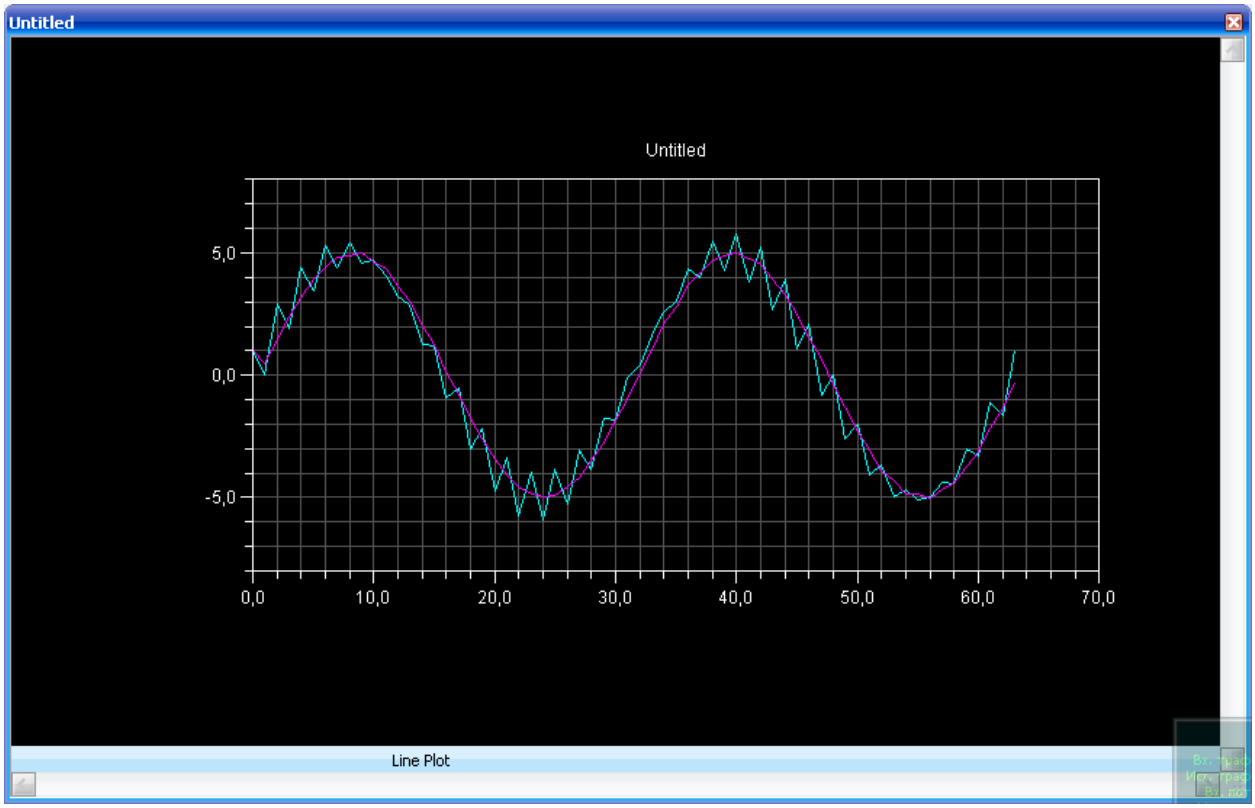

Рис.4.8. Результат работы нерекурсивной фильтрации сигналов

4.3. Инструкция для пользователей

Для начала работы с программой её необходимо скопировать на диск C: или D:. Также по указанному пути помещается папка с демонстрационными и пробными сигналами для проверки работоспособности приложения. Сигнал, поступающие на вход приложения должны иметь размер 64 точек.

Начало работы производится с запуска программы. На экран выводится главное окно приложения.

В меню «Файл» выбираем пункт «Открыть» или нажимаем комбинацию клавиш «Ctrl+O». В появившемся модальном диалоговом окне необходимо выбрать основной файл проекта. После нажатия на клавишу «OK» откроется проект.

С нажатием копки Run () или с помощью F5 из клавиатуры программа запускается.

Чтобы изменить отсчетов функции и соответственно его матрицы нужно изменит файлы, который расположен внутри папка проекта.

Выводы

В четвертой главе получены следующие основные результаты:

1. Приводится реализация разработанной программы для фильтрации сигналов от помех с использованием КИХ-фильтров.

2. Даётся пример практической реализации на контрольном примере.

3. Разработана инструкция для пользователей.

Заключение

В ходе выполнения диссертационной работы достигнуты следующие результаты.

В работе дан обзор существующих на настоящее время линейные системы, сигналов и алгоритмов цифровой обработки сигналов.

В первой главе исследованы свойства линейных систем, анализированы структурные схемы линейных систем, определены основные задачи и направления цифровой обработки сигналов, анализированы области применения цифровой обработки сигналов.

Во второй главе анализированы ключевые операции цифровой обработки сигналов, исследованы спектральные методы обработки сигналов, рассмотрены методы фильтрации сигналов с использованием спектральных

В третей главе проделан анализ архитектур специализированных процессоров обработки сигналов, выбран серия сигнального процессора, выбраны методы, алгоритмы и программные средства цифровой обработки сигналов для ADSP 102.

В четвертой главе приводится реализация разработанных программ для фильтрации сигналов от помех с использованием КИХ-фильтров, даётся пример практической реализации на контрольном примере, для удобства пользователя разработана инструкция для пользователей.

Список литературы

1. Антонью А. Цифровые фильтры: анализ и проектирование. – М.: Радио и связь, 1983. – 320 с.
2. Баскаков С.И. Радиотехнические цепи и сигналы: Учебник для вузов. - М.: Высшая школа, 1988.- 448 с.
3. Бендат Дж., Пирсол А. Прикладной анализ случайных данных. – М.: Мир, 1989. – 540 с.
4. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. – М.: Мир, 1989. – 448 с.
5. Гольденберг Л.М. и др. Цифровая обработка сигналов: Справочник. - М.: Радио и связь, 1985.- 312 с.
6. Гольденберг Л.М. и др. Цифровая обработка сигналов: Учебное пособие для вузов. - М.: Радио и связь, 1990.- 256 с.
7. Гутников В.С. Фильтрация измерительных сигналов. – Л.: Энергоатомиздат, 1990. – 192 с.
8. Даджион Д., Мерсеро Р. Цифровая обработка многомерных сигналов. – М.: Мир, 1988. – 488 с.
9. Кулханек О. Введение в цифровую фильтрацию в геофизике. – М.: Недра, 1981. – 198 с.
10. Купер Дж., Макгиллем А. Вероятностные методы анализа сигналов и систем. – М.: Мир, 1989. – 376 с.
11. Марпл-мл. С.Л. Цифровой спектральный анализ и его приложения. – М.: Мир, 1990. – 584 с.
12. Оппенгейм А.В., Шафер Р.В. Цифровая обработка сигналов. – М.: Связь, 1979. – 416 с.
13. Отнес Р., Эноксон Л. Прикладной анализ временных рядов. – М.: Мир, 1982. – 428 с.
14. Рабинер Л., Гоулд Б. Теория и применение цифровой обработки сигналов. – М.: Мир, 1978. – 848 с.

15. Сиберт У.М. Цепи, сигналы, системы. – М.: Мир, 1988. – 336 с.
16. Хемминг Р.В. Цифровые фильтры. – М.: Недра, 1987. – 221 с.
17. Дьяконов В., Абраменкова И. MATLAB. Обработка сигналов и изображений. Специальный справочник. – СПб.: Питер, 2002, 608 с.
18. Астафьева Н.М. Вейвлет-анализ: Основы теории и примеры применения. / Успехи физических наук, 1996, т.166, № 11, стр. 1145-1170.
19. Петухов А.П. Введение в теорию базисов всплесков. – СПб.: Изд. СПбГТУ, 1999, 132 с.
20. Адаптивные фильтры. /Под ред. К.Ф.Н.Коуэна и П.М.Гранта. – М.: Мир, 1988, 392 с.
21. Корн Г., Корн Е. Справочник по математике для научных работников и инженеров. – М.: Наука, 1984.
22. Айфичер Э., Джервис Б. Цифровая обработка сигналов. Практический подход. / М., "Вильямс", 2004, 992 с.
23. Солонина А.И. и др. Основы цифровой обработки сигналов. Учебное пособие. – СПб.: БХВ Петербург, 2005. – 768 с. URL: http://lord-n.narod.ru/download/books/walla/dsp/Solonin.Osnovu.DSP.rar
24. Хуанг Т.С. и др. Быстрые алгоритмы в цифровой обработке изображений. – М.: Радио и связь, 1984. – 224 с.
25. Лукин А. Введение в цифровую обработку сигналов (Математические основы).- М.: МГУ, Лаборатория компьютерной графики и мультимедиа, 2002.
26. A BDTI Analysis of the Texas Instruments TMS320C64xx. – BDTI, 2004 (http://www.bdti.com).
27. The BDTImark2000. A Summary Measure of Signal Processing Speed. A White Paper by Berkeley Design Technology, Inc. – BDTI, September, 2004 (http://www.bdti.com).
28. http://www.ti.com.
29. http://www.analog.com.

30. DSP Selection Guide. Digital Signal Processors, OMAP Processors, System Solutions, Development Tools. – Texas Instruments, 3Q, 2005, (http://www.ti.com).
31. TMS320C6455 Fixed-Point Digital Signal Processor. – Texas Instruments, 2005, (http://www.ti.com).
32. http://pv.bstu.ru/dsp/dspcourse.pdf
33. http://dsp-book.narod.ru/dspcourse.djvu
34. http://geogin.narod.ru/arhiv/dsp/dsp4.pdf

Приложение

File.cpp

```
#ifdef __ADSPTS201__
#include <defts201.h>
#endif
#define N 64
#include "cache_macros.h"
#include <stdio.h>
#include <math.h>

float yf[N] = {
        #include "Function/Function64.txt"
};
float ynew[N];
main()
{

#ifdef __ADSPTS201__
asm("                    \
   #include <defts201.h>  \
   #include cache_macros.h\
   cache_enable(750);     \
   #include <ini_cache.h> \
   #include <fftdef.h>;   \
   preload_cache;         \
  ");
#endif

        float sum;
```

```
        int win=2, kk=0;
        int t=0;
        while (t<N)
        {
                sum=0; kk=0;
                for (int w=0; w<win; w++)
                {
                        if ((t-w)>=0)
                        {
                        sum+=yf[t-w];
                        kk++;
                        }
                }
                ynew[t]=sum/kk;
                t++;
        }

        printf("N:  y[N]       :   ynew[N]\n");
        for (t=0; t<N; t++)
        {
                printf("%d:  %f   :   %f\n",t,yf[t],ynew[t]);
        }

        return 0;
}
```

DSP.cpp

```
// DSP.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "DSP.h"
#include "MainFrm.h"
#include "DSPDoc.h"
#include "DSPView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDSPApp

BEGIN_MESSAGE_MAP(CDSPApp, CWinApp)

        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

CDSPApp::CDSPApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}
```

```
CDSPApp theApp;
BOOL CDSPApp::InitInstance()
{
	AfxEnableControlContainer();
#ifdef _AFXDLL
	Enable3dControls();
#else
	Enable3dControlsStatic();
#endif
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));
	LoadStdProfileSettings();  // Load standard INI file options (including MRU)
	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CDSPDoc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
		RUNTIME_CLASS(CDSPView));
	AddDocTemplate(pDocTemplate);
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}
```

```
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();
	enum { IDD = IDD_ABOUTBOX };
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	protected:
		DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
			//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
void CDSPApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

DSPView.cpp

```
// DSPView.cpp : implementation of the CDSPView class
//
#include "stdafx.h"
#include "DSP.h"
#include "DSPDoc.h"
#include "DSPView.h"
#include "MainFrm.h"
#include <math.h>
#include "fstream.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDSPView

IMPLEMENT_DYNCREATE(CDSPView, CView)
BEGIN_MESSAGE_MAP(CDSPView, CView)
	//{{AFX_MSG_MAP(CDSPView)
	ON_BN_CLICKED(IDC_BUTTON_OK, OnButtonOk)
	ON_EN_CHANGE(IDC_EDIT_SCALE, OnChangeEditScale)
```

```
        ON_BN_CLICKED(IDC_BUTTON_FILE, OnButtonFile)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW,
CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDSPView construction/destruction

CDSPView::CDSPView()
{
        // TODO: add construction code here
        K=10;
        xCenter=400;
        yCenter=200;
}

CDSPView::~CDSPView()
{
}

BOOL CDSPView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CView::PreCreateWindow(cs);
```

```
}

/////////////////////////////////////////////////////////////////////////////
// CDSPView drawing

void CDSPView::OnDraw(CDC* pDC)
{
	CDSPDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	// TODO: add draw code for native data here

	Osi();
//	Grafik(pDC);

}

/////////////////////////////////////////////////////////////////////////////

BOOL CDSPView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CDSPView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CDSPView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
```

```
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CDSPView diagnostics
#ifdef _DEBUG
void CDSPView::AssertValid() const
{
	CView::AssertValid();
}
void CDSPView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CDSPDoc* CDSPView::GetDocument()
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CDSPDoc)));
	return (CDSPDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
void CDSPView::Osi()
{

/*
	//----- Фон ------
	CClientDC Fon(this);
```

```
CBrush* brush;
    brush=new CBrush(RGB(0,0,0));
Fon.SelectObject(brush);

    Fon.Rectangle(-100,-100,2000,2000);
        //-------- Побочные оси -------

    CPen pen1(PS_SOLID,1,RGB(0,100,0));
pDC->SelectObject(&pen1);

    for (int i=xCenter; i>=0; i-=K)
    {
        pDC->MoveTo(i,0);
        pDC->LineTo(i,2*yCenter);
    }
    for (i=xCenter; i<=2*xCenter; i+=K)
    {
        pDC->MoveTo(i,0);
        pDC->LineTo(i,2*yCenter);
    }
    for (i=yCenter; i>=0; i-=K)
    {
        pDC->MoveTo(0,i);
        pDC->LineTo(2*xCenter,i);
    }

    for (i=yCenter; i<=2*yCenter; i+=K)
    {
        pDC->MoveTo(0,i);
```

```
            pDC->LineTo(2*xCenter,i);
        }
        //----- Главные оси ------

        CPen pen(PS_SOLID,1,RGB(128,128,128));
    pDC->SelectObject(&pen);
        pDC->MoveTo(xCenter,0);
    pDC->LineTo(xCenter,2*yCenter);
        pDC->MoveTo(0,yCenter);
    pDC->LineTo(2*xCenter,yCenter);

*/
        CClientDC hPoint(this);

        CBrush* brush;
        brush=new CBrush(RGB(0,0,0));
    hPoint.SelectObject(brush);

        hPoint.Rectangle(-100,-100,2000,2000);
            //-------- Побочные оси -------
            CPen pen1(PS_SOLID,1,RGB(0,100,0));
    hPoint.SelectObject(&pen1);

        for (int i=xCenter; i>=0; i-=K)
        {
            hPoint.MoveTo(i,0);
            hPoint.LineTo(i,3*yCenter);
        }
        for (i=xCenter; i<=3*xCenter; i+=K)
        {
```

```
            hPoint.MoveTo(i,0);
            hPoint.LineTo(i,3*yCenter);
        }
        for (i=yCenter; i>=0; i-=K)
        {
            hPoint.MoveTo(0,i);
            hPoint.LineTo(3*xCenter,i);
        }
        for (i=yCenter; i<=3*yCenter; i+=K)
        {
            hPoint.MoveTo(0,i);
            hPoint.LineTo(3*xCenter,i);
        }
        //----- Главные оси ------
            CPen pen(PS_SOLID,1,RGB(128,128,128));
   hPoint.SelectObject(&pen);
        hPoint.MoveTo(xCenter,0);
   hPoint.LineTo(xCenter,3*yCenter);
        hPoint.MoveTo(0,yCenter);
   hPoint.LineTo(3*xCenter,yCenter);
}
double CDSPView::y(double t)
{
            CClientDC hPoint(this);

        CString Numbers ("1234567890");
        //double y,t=-xCenter/K;
        double a,b,c;
        CString Slag[10], Mnoj[10][10];
        int L=Str.GetLength(), pl, um, sk;
```

```
	char kx[5];

	double A=0, B=0, C, D;

	if (Str.GetLength()<=2)
	{
		if (Str.GetAt(Str.GetLength()-1)=='t')  A=t;
		if 			(Numbers.Find(Str.GetAt(Str.GetLength()-1))>=0)
A=atoi((CString)Str.GetAt(Str.GetLength()-1));
		if (Str.GetAt(0)=='-') A=-A;
	}

	if (Str.GetLength()==3)
	{
		if (Str.GetAt(0)=='t')  a=t;
		if (Numbers.Find(Str.GetAt(0))>=0) a=atoi((CString)Str.GetAt(0));
		if (Str.GetAt(2)=='t')  b=t;
		if (Numbers.Find(Str.GetAt(2))>=0) b=atoi((CString)Str.GetAt(2));
		if (Str.GetAt(1)=='*') A=a*b;
		if (Str.GetAt(1)=='/') A=a/b;
		if (Str.GetAt(1)=='+') A=a+b;
		if (Str.GetAt(1)=='-') A=a-b;
	}
	if (Str.GetLength()>3)
	{
			pl=Str.Find('+');
//			hPoint.TextOut(10,30,itoa(pl,kx,10));
		if (pl>0)
		{
			Slag[0]=Str.Left(pl);
```

```
//                      hPoint.TextOut(10,50,Slag[0]);

                Slag[1]=Str.Right(L-pl-1);
//                      hPoint.TextOut(10,70,Slag[1]);
            }
            else
            {
                Slag[0]=Str;
                Slag[1]=Str;
            }
                um=Slag[0].Find('*');
                sk=Slag[0].Find('(');

//              if ((um<0)||(um>sk))
//              {
                    C=1;
                    if                          ((um>0)&&(um<sk))
Mnoj[0][0]=Slag[0].Right(Slag[0].GetLength()-um-1);
                    else Mnoj[0][0]=Slag[0];

                    if ((Mnoj[0][0].Find('sin')>0)||(Mnoj[0][0].Find('cos')>0))
                    {
//                      hPoint.TextOut(10,50,Mnoj[0][0]);

                        if (Mnoj[0][0].GetLength()-5<=2)
                        {
                            if
(Mnoj[0][0].GetAt(Mnoj[0][0].GetLength()-2)=='t')  A=t;
                            if
(Numbers.Find(Mnoj[0][0].GetAt(Mnoj[0][0].GetLength()-2))>=0)
```

```
A=atoi((CString)Mnoj[0][0].GetAt(Mnoj[0][0].GetLength()-2));
                                        //if (Mnoj[0][0].GetAt(0)=='-') A=-A;
                                }
                                if (Mnoj[0][0].GetLength()-5==3)
                                {
                                        if (Mnoj[0][0].GetAt(4)=='t')  a=t;
                                        if  (Numbers.Find(Mnoj[0][0].GetAt(4))>=0)
a=atoi((CString)Mnoj[0][0].GetAt(4));
                                        if (Mnoj[0][0].GetAt(6)=='t')  b=t;
                                        if  (Numbers.Find(Mnoj[0][0].GetAt(6))>=0)
b=atoi((CString)Mnoj[0][0].GetAt(6));
                                        if (Mnoj[0][0].GetAt(5)=='*') A=a*b;
                                        if (Mnoj[0][0].GetAt(5)=='/') A=a/b;
                                        if (Mnoj[0][0].GetAt(5)=='+') A=a+b;
                                        if (Mnoj[0][0].GetAt(5)=='-') A=a-b;
                                }
                                if (Mnoj[0][0].Find('sin')>0) A=sin(A);
                                if (Mnoj[0][0].Find('cos')>0) A=cos(A);
                        }
                if ((um>0)&&(um<sk))
                {
                        Mnoj[0][1]=Slag[0].Left(um);
                        if      (Mnoj[0][1].GetAt(Mnoj[0][1].GetLength()-1)=='t')
C=t;
                        if
(Numbers.Find(Mnoj[0][1].GetAt(Mnoj[0][1].GetLength()-1))>=0)
C=atoi((CString)Mnoj[0][1].GetAt(Mnoj[0][1].GetLength()-1));
                }
                A=C*A;
                        //Mnoj[0][1]='1';
```

```
//                  }
                        um=Slag[1].Find('*');
                    sk=Slag[1].Find('(');

//                  if ((um<0)||(um>sk))
//                  {
                        D=1;

                        if                              ((um>0)&&(um<sk))
Mnoj[1][0]=Slag[1].Right(Slag[1].GetLength()-um-1);
                        else Mnoj[1][0]=Slag[1];

                        if ((Mnoj[1][0].Find('sin')>0)||(Mnoj[1][0].Find('cos')>0))
                        {

                            if (Mnoj[1][0].GetLength()-5<=2)
                            {
                                if
(Mnoj[1][0].GetAt(Mnoj[1][0].GetLength()-2)=='t')  B=t;
                                if
(Numbers.Find(Mnoj[1][0].GetAt(Mnoj[1][0].GetLength()-2))>=0)
B=atoi((CString)Mnoj[1][0].GetAt(Mnoj[1][0].GetLength()-2));
                            //      if (Mnoj[1][0].GetAt(0)=='-') B=-B;
                            }
                            if (Mnoj[1][0].GetLength()-5==3)
                            {
                                if (Mnoj[1][0].GetAt(4)=='t')  a=t;
                                if  (Numbers.Find(Mnoj[1][0].GetAt(4))>=0)
a=atoi((CString)Mnoj[1][0].GetAt(4));
                                if (Mnoj[1][0].GetAt(6)=='t')  b=t;
```

```
                                        if  (Numbers.Find(Mnoj[1][0].GetAt(6))>=0)
b=atoi((CString)Mnoj[1][0].GetAt(6));
                                        if (Mnoj[1][0].GetAt(5)=='*') B=a*b;
                                        if (Mnoj[1][0].GetAt(5)=='/') B=a/b;
                                        if (Mnoj[1][0].GetAt(5)=='+') B=a+b;
                                        if (Mnoj[1][0].GetAt(5)=='-') B=a-b;
                                    }

                                    if (Mnoj[1][0].Find('sin')>0) B=sin(B);
                                    if (Mnoj[1][0].Find('cos')>0) B=cos(B);
                                    //B=cos(B);
                                }

                        if ((um>0)&&(um<sk))
                        {
                            Mnoj[1][1]=Slag[1].Left(um);
                            if      (Mnoj[1][1].GetAt(Mnoj[1][1].GetLength()-1)=='t')
D=t;
                            if
(Numbers.Find(Mnoj[1][1].GetAt(Mnoj[1][1].GetLength()-1))>=0)
D=atoi((CString)Mnoj[1][1].GetAt(Mnoj[1][1].GetLength()-1));
                        }
                        B=D*B;
//                      }
        //                      y=A+B;

//              }
                        if (pl<0) B=0;
                }
        double function=A+B;
```

```
	return function;
}
void CDSPView::Grafik(CDC *pDC)
{
	CClientDC hPoint(this);
	CPen pen(PS_SOLID,1,RGB(255,0,0));
	hPoint.SelectObject(&pen);
	double t=-xCenter/K;
	while (t<=xCenter/K)
	{
			hPoint.MoveTo(xCenter+t*K, yCenter-y(t)*K);
		t=t+1/K;
		hPoint.LineTo(xCenter+t*K, yCenter-y(t)*K);
		}

}
void CDSPView::OnButtonOk()
{
//	CDC *pDC (this);
//		Osi(pDC);
	// TODO: Add your control notification handler code here
	Osi();
	CClientDC hPoint(this);
	CEdit *pEditFunction = (CEdit*) ((CMainFrame*)(AfxGetApp()->m_pMainWnd))->m_wBar.GetDlgItem(IDC_EDIT_FUNCTION);
	pEditFunction->GetWindowText(Str);
//	hPoint.TextOut(10,10,Str);
//	double x;
//	function=sin(x);
	//	CClientDC hPoint(this);
```

```
        CPen pen(PS_SOLID,1,RGB(255,0,0));
        hPoint.SelectObject(&pen);
        double t=-xCenter/K;
while (t<=xCenter/K)
        {
                hPoint.MoveTo(xCenter+t*K, yCenter-y(t)*K);
                t=t+1/K;
                hPoint.LineTo(xCenter+t*K, yCenter-y(t)*K);
}
}
void CDSPView::OnChangeEditScale()
{
        // TODO: If this is a RICHEDIT control, the control will not
        // send this notification unless you override the CView::OnInitDialog()
        // function and call CRichEditCtrl().SetEventMask()
        // with the ENM_CHANGE flag ORed into the mask.

                CClientDC hPoint(this);
        CEdit *pEditScale = (CEdit*) ((CMainFrame*)(AfxGetApp()-
>m_pMainWnd))->m_wBar.GetDlgItem(IDC_EDIT_SCALE);
        CString scl;
//      pEditScale->SetWindowText("10");
        pEditScale->GetWindowText(scl);
        //hPoint.TextOut(10,10,scl);
        K=atoi(scl);
}

void CDSPView::OnButtonFile()
{
```

```
	CClientDC hPoint(this);
	CEdit *pEditCount = (CEdit*) ((CMainFrame*)(AfxGetApp()->m_pMainWnd))->m_wBar.GetDlgItem(IDC_EDIT_COUNT);
	CString count;
//	pEditCount->SetWindowText("64");
	pEditCount->GetWindowText(count);
	//hPoint.TextOut(10,10,scl);
	int N;
	N=atoi(count);

	CString filename="Function"+count+".txt";
	ofstream file(filename,ios::out);

	for (int t=0; t<N; t++)
	{
		file<<y(t)<<","<<endl;
	}
}
```