МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
Национальный университет кораблестроения
имени адмирала Макарова

В. М. РЯБЕНЬКИЙ, А. О. УШКАРЕНКО

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Монография

Рекомендовано Ученым советом НУК

Николаев • НУК • 2016

УДК 681.3.06(075.8)
ББК 32.884.1
Р 98

Авторы: В. М. Рябенький, доктор технических наук, профессор, зав. кафедры теоретической электротехники и электронных систем;
А. О. Ушкаренко, кандидат технических наук, доцент кафедры теоретической электротехники и электронных систем

Рецензенты: В. С. Смирнов, доктор технических наук, профессор;
Н. Т. Фисун, доктор технических наук, профессор

Рекомендовано Ученым советом
Национального университета кораблестроения
имени адмирала Макарова
как учебное пособие для студентов высших учебных заведений
(*протокол № 8 от 04.07.2016 г.*)

Рябенький В. М.

Р 98 Программная реализация алгоритмов цифровой обработки сигналов : монография / В. М. Рябенький, А. О. Ушкаренко. – Николаев : НУК, 2016. – 244 с.

В монографии рассмотрены базовые вопросы дискретизации сигналов по времени, их цифрового представления, спектрально-корреляционного анализа, принципов модуляции сигналов, цифровой обработки изображений. Основной акцент делается на программную реализацию алгоритмов цифровой обработки сигналов, в частности цифровых фильтров, которые могут использоваться в системах мультимедиа, цифровой связи, телекоммуникациях, обработке изображений и цифровом телевидении. Приводится большое количество примеров программ, рассмотрены теоретические основы цифровой обработки сигналов и предлагаются задачи для самостоятельного решения.

Монография рекомендуется специалистам в области цифровой обработки сигналов, студентам, обучающимся по направлению «Телекоммуникации», а также студентам других направлений и специальностей, в образовательных программах которых предусмотрено изучение дисциплины «Цифровая обработка сигналов» или родственных дисциплин.

УДК 681.3.06(075.8)
ББК 32.884.1

Введение

Цифровая обработка сигналов имеет большое фундаментальное и прикладное значение в телекоммуникациях, современной радиотехнике и смежных с нею областях. Ее методы используются для разработки и исследования радиоэлектронных устройств и систем различного назначения, а ее средства – для их аппаратно-программной реализации.

Цифровой обработкой сигналов принято называть в вычислительной технике арифметическую обработку последовательностей равноотстоящих во времени отсчетов. Под цифровой обработкой понимают также обработку одномерных и многомерных массивов данных. Цифровая обработка сигналов, в отличие от аналоговой обработки, традиционно используемой во многих радиотехнических устройствах, является более дешевым способом достижения результата, обеспечивает более высокую точность, миниатюрность и технологичность устройства, температурную стабильность. Таким образом, цифровая обработка сигналов – это математический аппарат, алгоритмы и технологии, применяемые для работы с сигналами, после их преобразования в цифровую форму.

В системах обработки звука цифровые процессоры обработки сигнала решают задачи анализа, распознавания и синтеза речи, сжатия речи в системах телекоммуникации. Для систем обработки изображений типовыми задачами являются улучшение изображений, сжатие информации для передачи и хранения, распознавание образов. При обработке цифровых звуковых сигналов используются алгоритмы цифровой фильтрации и спектрального анализа, алгоритмы корреляционного анализа, обратной свертки, специальные алгоритмы линейного предсказания.

В монографии рассмотрены базовые вопросы дискретизации сигналов по времени, их цифрового представления, спектрально-корреляционного анализа, принципов модуляции сигналов, цифровой обработки изображений. Основной акцент делается на программную реализацию алгоритмов цифровой обработки сигналов, в частности цифровых фильтров, которые можно будет использовать в системах мультимедиа, цифровой связи, телекоммуникациях, обработке изображений и цифровом телевидении.

Научная новизна полученных результатов состоит в следующем:

1. Получили дальнейшее развитие методы программной реализации алгоритмов цифровой обработки сигналов за счет сочетания в одном программном модуле средств реализации, визуализации и валидации цифровых фильтров.

2. Улучшена методика подготовки специалистов по цифровой обработке сигналов за счет использования сквозного проектирования цифровых фильтров, их программной реализации, визуализации результатов обработки сигналов с помощью развитых средств графического интерфейса пользователя, что позволяет наиболее полно понять работу алгоритмов ЦОС.

3. Разработаны инструментальные средства для исследования алгоритмов цифровой обработки сигналов и выполнена их интеграция с объектно-ориентированным языком программирования. Таким образом, обеспечена поддержка модульности, что позволяет использовать их в различных проектах. Для цифровых фильтров, при необходимости, обеспечивается построение амплитудно-частотной и фазо-частотной характеристик. Преимуществом предлагаемого подхода является относительная простота, которая заключается в необходимости записи только передаточной функции цифрового фильтра, а с помощью программно реализованного модуля для работы с комплексными числами, происходит построение АЧХ.

4. Получили дальнейшее развитие методы моделирования цифровых фильтров за счет использования программных модулей, формирующих на входе фильтров тестовые сигналы различной формы и с разным спектром (гармонический сигнал, импульсный сигнал, белый шум), и средств спектрального анализа сигналов на выходе фильтров, что позволяет исследовать работу фильтров как во временной, так и в частотной области, и сделать выводы по корректности их работы.

Практическое значение полученных результатов заключается в том, что они успешно используются при разработке промышленных телекоммуникационных систем и учебных проектах студентов направления подготовки «Телекоммуникации».

СПИСОК УСЛОВНЫХ СОКРАЩЕНИЙ

АКФ – автокорреляционная функция
ФНЧ – фильтр нижних частот
АЦП – аналого-цифровой преобразователь
АЧХ – амплитудно-частотная характеристика
БИХ – бесконечная импульсная характеристика
БПФ – быстрое преобразование Фурье
ВКФ – взаимная корреляционная характеристика
ДПФ – дискретное преобразование Фурье
КИХ – конечная импульсная характеристика
КЧД – компрессор частоты дискретизации
НЦФ – нерекурсивный цифровой фильтр
ОДПФ – обратное дискретное преобразование Фурье
ПЛИС – программируемая логическая интегральная схема
ПФ – полосовой фильтр
ПЦОС – процессор цифровой обработки сигналов
РЦФ – рекурсивный цифровой фильтр
СФ – сглаживающий фильтр
УВХ – устройство выборки-хранения
ФВЧ и ФНЧ – фильтры верхних и нижних частот
ФЧХ – фазочастотная характеристика
ЦАП – цифро-аналоговый преобразователь
ЦОС – цифровая обработка сигналов
ЦСП – цифровой сигнальный процессор
ЦФ – цифровой фильтр
ЧХ – частотная характеристика
ЭЧД – экспандер частоты дискретизации

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ФИЛЬТРАЦИИ СИГНАЛОВ

1.1. Цифровые фильтры с конечной импульсной характеристикой

Электронный фильтр – это частотно-избирательный устройство, которое служит для передачи сигналов в заданном диапазоне частот (полосе пропускания) и подавления сигналов в других диапазонах частот (полосе задержания). Фильтры широко используются в системах связи, в схемах защиты электронных систем от помех, для разделения электрических сигналов различной частоты, уменьшение пульсаций напряжения на выходе выпрямителя, ограничения спектра модулированных сигналов, передаваемых и др.

Различают аналоговые фильтры, в которых обрабатываемый сигнал имеет аналоговую форму, и цифровые фильтры, предназначенные для обработки цифровых сигналов.

Фильтры принято классифицировать по виду амплитудно-частотной характеристики (АЧХ) на фильтры нижних частот (ФНЧ), фильтры верхних частот (ФВЧ), полосовые и заградительные фильтры.

Модуль передаточной функции фильтра остается практически постоянным в специальной области частот, которая называется полосой пропускания, и довольно резко падает с удалением от границ этого участка. Границы участка пропускания называют предельными частотами. Участок частот с достаточно большим подавлением амплитуды сигнала называется полосой заграждения. Между полосами пропускания и заграждения находится переходный участок.

На рис. 1.1, а-г, показаны идеальные АЧХ фильтров: нижних частот, верхних частот, полосовой и заградительный.

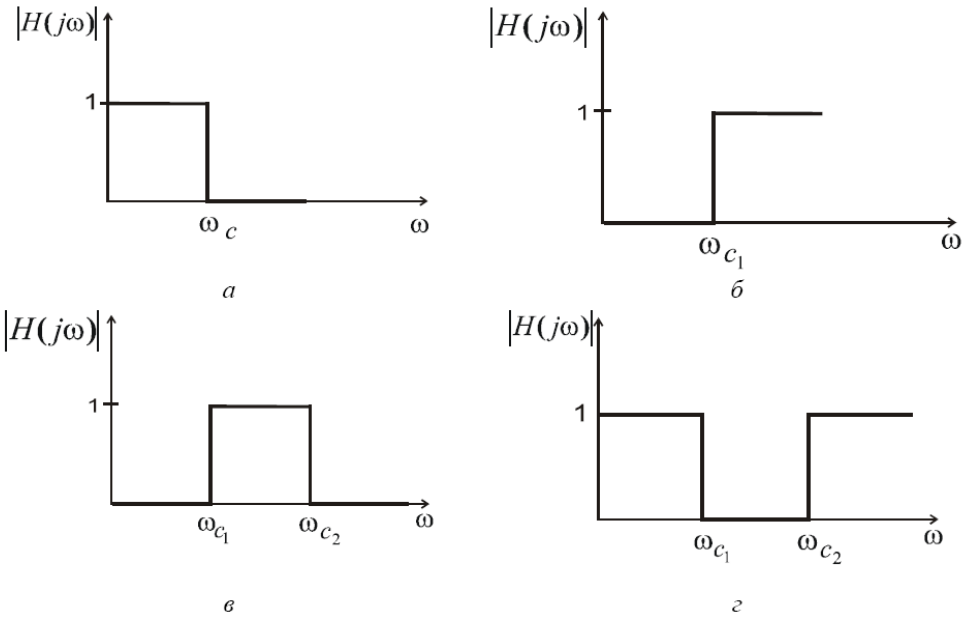


Рис. 1.1.

Амплитудно-частотная характеристика реального фильтра нижних частот показана на рис. 1.2. Поскольку с помощью реального фильтра невозможно реализовать постоянную амплитудно-частотную характеристику, задают максимальное отклонение АЧХ в полосе пропускания A_{max}. В полосе заграждения задается минимальная величина ослабления сигнала A_{min}.

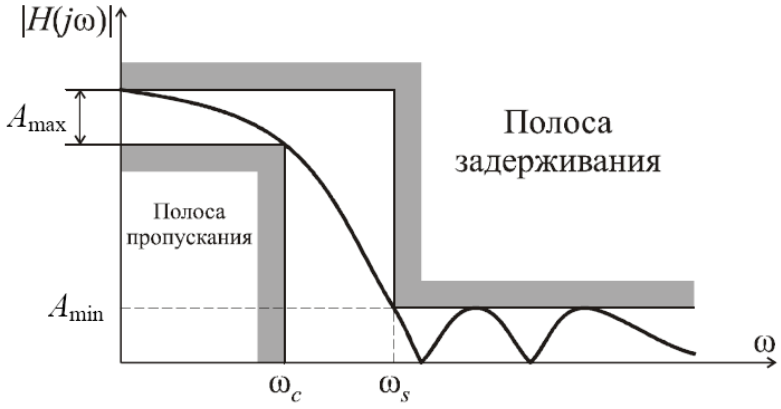


Рис. 1.2.

Физически реализованный фильтр всегда имеет переходную полосу между полосами пропускания и заграждения. Она расположена между частотой среза ω_c и граничной частотой полосы заграждения ω_s. Отношение ω_c/ω_s характеризует избирательность фильтра.

Итак, амплитудно-частотная характеристика фильтра нижних частот определяется следующими параметрами:

1) частотой среза ω_c;

2) максимальным отклонением в полосе пропускания A_{max};

3) граничной частотой полосы пропускания ω_s;

4) минимальным ослаблением в полосе заграждения A_{min}.

Цифровая обработка сигналов (ЦОС) – это преобразование сигналов, представленных в цифровой форме. Любой аналоговый сигнал может быть подвергнут дискретизации во времени и квантования по уровню, то есть представлен в цифровой форме. С помощью математических алгоритмов входной сигнал преобразуется в некоторый другой сигнал, который имеет необходимые свойства.

Обработка сигналов во временной области широко используется в современных электронных и телекоммуникационных системах. Основные задачи ЦОС:

– линейная фильтрация – селекция сигнала в частотной области, синтез фильтров, согласованных с сигналами, частотное разделение каналов, корректоры характеристик каналов;

– спектральный анализ – обработка речевых, звуковых, сейсмических и гидроакустических сигналов, распознавания образов;

– частотно-временной анализ – компрессия изображений, гидро- и радиолокация, разнообразные задачи обнаружения сигнала;

– адаптивная фильтрация – обработка речи, изображений, распознавания образов, подавление шумов;

– нелинейная обработка – вычисление корреляции, медианная фильтрация; синтез амплитудных, фазовых, частотных детекторов, обработка речи, векторное кодирование;

– высокоскоростная обработка – интерполяция (увеличение) и децимация (уменьшение) частоты дискретизации в высокоскоростных системах телекоммуникации, аудиосистемах;

– получение сверток.

Цифровая фильтрация является одним из наиболее мощных инструментальных средств ЦОС. Кроме очевидных преимуществ устранения ошибок в фильтре, связанных с флуктуациями параметров пассивных компонентов аналоговых фильтров во времени и по температуре, дрейфом характеристик операционных усилителей (в активных фильтрах) и т.д., цифровые фильтры способны удовлетворять таким техническим требованиям по своим параметрам, которых, в лучшем случае, было бы чрезвычайно трудно или даже невозможно достичь в аналоговом исполнении. Кроме того, характеристики цифрового фильтра могут быть легко изменены программно.

Цифровой фильтр, работающий в реальном масштабе времени, оперирует с дискретными по времени данными в противоположность непрерывному сигналу, который обрабатывается аналоговым фильтром. При этом очередной отсчет, соответствующий отклику фильтра, формируется по окончании каждого периода дискретизации. Вследствие дискретной природы обрабатываемого сигнала, на отсчеты данных часто ссылаются по их номерам, например, отсчет 1, отсчет 2, отсчет 3 и т.д.

На рис. 1.3, а, представлена упрощенная функциональная схема цифрового фильтра.

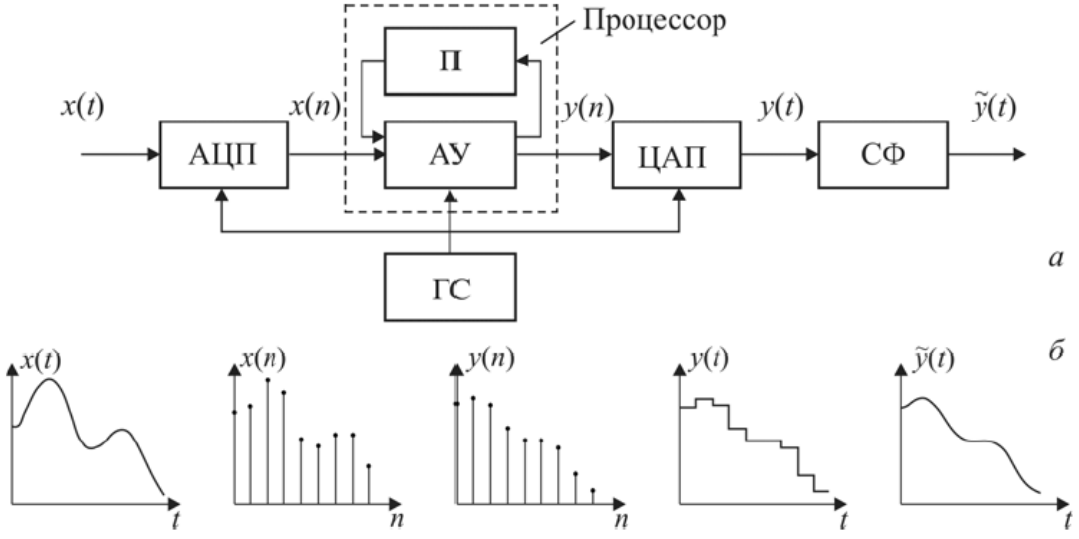


Рис. 1.3.

Непрерывный сигнал $x(t)$ поступает на вход аналого-цифрового преобразователя (АЦП), который фиксирует значения $x(n)$ сигнала в дискретные моменты времени $t=nT$, $n=0,1, ...$, и преобразует их в цифровой код в виде двоичного числа. Последовательность $x(n)$ поступает в процессор, состоящий из арифметического устройства (АУ) и памяти (П). В процессоре осуществляется преобразование последовательности $x(n)$ в соответствии с определенным алгоритмом. В результате на его выходе образуется последовательность $y(n)$.

Последовательность $y(n)$ поступает на цифро-аналоговый преобразователь (ЦАП), в котором текущее значение $y(n)$, представленное в цифровом виде, преобразуется в постоянное напряжение, удерживаемое в течение соответствующего интервала дискретности. На выходе ЦАП формируется непрерывный сигнал $y(t)$ в виде ступенчатой функции (рис. 1, б). С помощью сглаживающего фильтра (СФ) нижних частот устраняются высокочастотные колебания, и выходной сигнал $\tilde{y}(t)$ цифрового фильтра приобретает сглаженный вид.

Существует два основных типа цифровых фильтров: фильтры с конечной импульсной характеристикой (КИХ) и фильтры с бесконечной импульсной характеристикой (БИХ). Как следует из терминологии, эта классификация относится к импульсным характеристикам фильтров. Изменяя веса коэффициентов и количество звеньев КИХ-фильтра, можно реализовать практически любую частотную характеристику. БИХ-фильтры могут иметь такие свойства, которые невозможно достичь методами аналоговой фильтрации (в частности, совершенно линейную фазовую характеристику). Но высокоэффективные КИХ-фильтры строятся с большим числом операций умножения и накопления, и поэтому требуют использования быстрых и эффективных процессоров цифровой обработки сигналов (DSP). С другой стороны, БИХ-фильтры имеют тенденцию имитировать принцип действия традиционных аналоговых фильтров с обратной связью. Поэтому их импульсная характеристика имеет бесконечную продолжительность. Благодаря использованию обратной связи, БИХ-фильтры могут быть реализованы с меньшим количеством коэффициентов, чем КИХ-фильтры. Другим способом реализации КИХ- или БИХ-фильтрации являются решетчатые фильтры, которые часто используются в задачах обработки речи. Цифровые фильтры применяются в системах адаптивной фильтрации, благодаря своему быстродействию и простоте изменения характеристик воздействием на его коэффициенты. Основные типы цифровых фильтров:

– фильтр скользящего среднего;

– фильтры с конечной импульсной характеристикой (линейная фаза, легкость проектирования, значительные вычислительные затраты);

– фильтры с бесконечной импульсной характеристикой (основаны на классических аналоговых фильтрах, высокая вычислительная эффективность);

– решетчатые фильтры (могут быть сих или НИП);

– адаптивные фильтры.

Фильтр с конечной импульсной характеристикой (Нерекурсивный фильтр, КИХ-фильтр) или FIR-фильтр (FIR сокр. от finite impulse response – конечная импульсная характеристика) – один из видов линейных цифровых фильтров, характерной особенностью которого является ограниченность по времени его импульсной характеристики (с какого-то момента времени она становится точно равной нулю). Такой фильтр называют ещё нерекурсивным из-за отсутствия обратной связи.

Цифровой фильтр описывается тремя характеристиками – *разностным уравнением*, которому может быть поставлена в соответствие *передаточная функция* $K(z)$ или полученная в результате обратного z-преобразования $K(z)$ его *дискретная импульсная характеристика* $h[k]$.

Нерекурсивные фильтры описываются разностным уравнением, в котором выходная величина $y[k]$ выражается только через конечное число значений входного сигнала $x[n]$.

$$y(k) = \sum_{m=0}^{k} b_m x(k-m) .$$

КИХ-фильтры могут быть реализованы с использованием трех элементов: умножитель, сумматор и блок задержки.

На рис. 1.4 показаны обозначения основных блоков для построения цифрового фильтра.

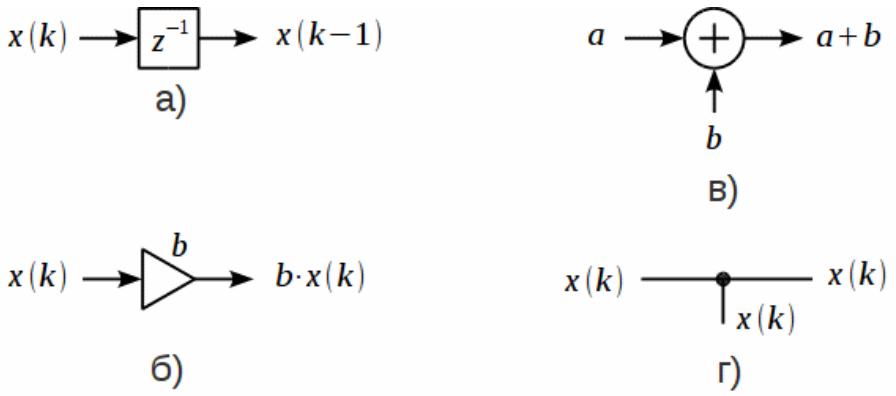


Рис. 1.4.

Вариант, показанный на рис. 1.5, есть прямая реализация КИХ-фильтров.

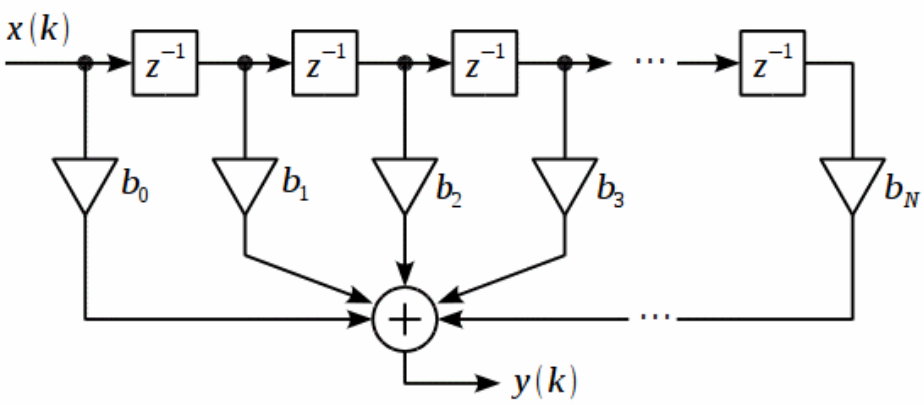


Рис. 1.5.

КИХ-фильтр порядка N содержит N линий задержки и N+1 коэффициент. Импульсная характеристика КИХ-фильтра всегда конечна и полностью совпадает с коэффициентами фильтра.

1.2. Программная реализация цифрового фильтра скользящего среднего

В редакторе ресурсов среды разработки Visual Studio 6.0 проектируется интерфейс диалогового окна, примерный вид которого показан на рис. 1.6. Осциллограммы сигналов до и после применения фильтра будут отображаться на элементе Satic Text. С помощью элементов управления Radio Button выбирается тип сигнала: синусоида (sin), синусоида с шумом (sin & noise), отфильтрованная синусоида (sin <– filter), импульсный сигнал (pulse), импульсный сигнал с шумом (pulse & noise), отфильтрованный импульсный сигнал с шумом (pulse <– filter). С помощью элемента текстового поля ввода Edit Box задается уровень шума, а в выпадающем списке Combo box пользователь может выбрать порядок фильтра.

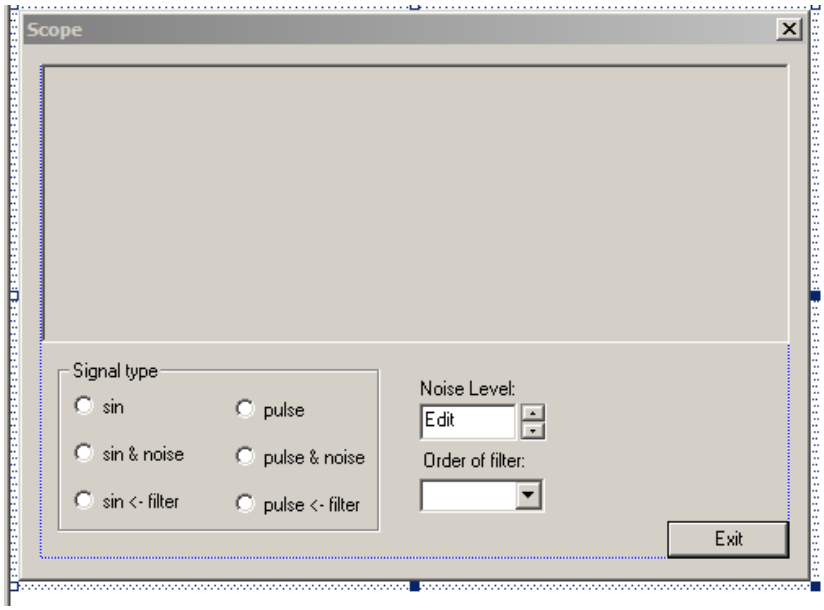


Рис. 1.6.

С элементами управления на диалоговом окне связываем переменные. Для этого вызываем окно Class Wizard нажатием сочетания клавиш Ctrl+W, и на закладке Member Variables с каждым элементом управления связываем переменную, как показано на рис. 1.7.

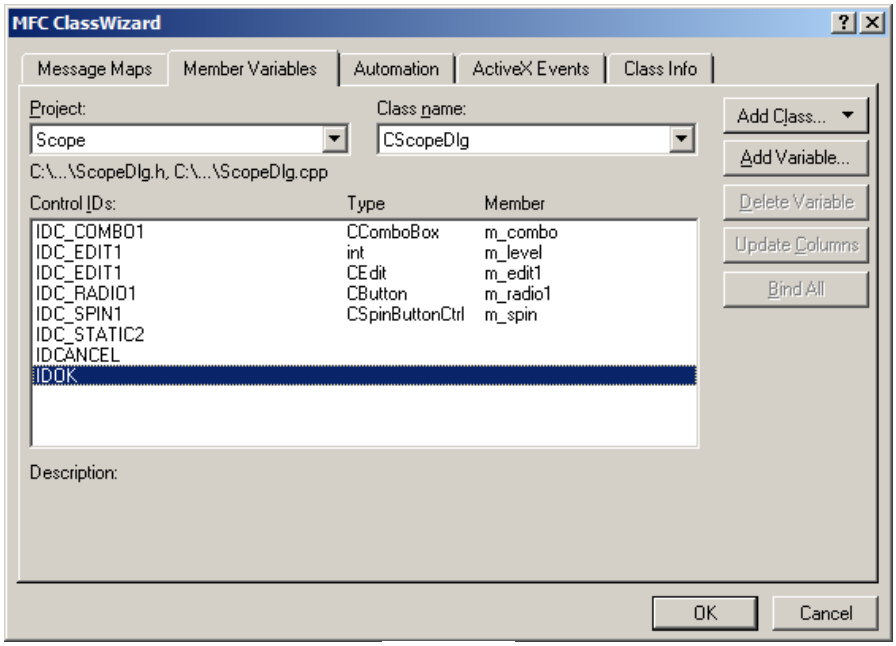


Рис. 1.7.

В классе диалога необходимо объявить следующие переменные:

```
CPoint* m_pulse_filter;
CPoint* m_pulse_noise;
CPoint* m_pulse;
int m_order;
CPoint* m_filter;
CPoint* m_noise;
CPoint* m_sin;
CStatic* m_scope;
CBrush* m_scope_brush;
CBrush* m_combo_brush;
int m_result[16];
int m_data[64];
CScopeDlg(CWnd* pParent = NULL);        // standard constructor
CPen m_scope_pen;
```

В функцию OnInitDialog добавляем код:

```
m_scope_brush=new CBrush;
m_scope_brush->CreateHatchBrush(4,RGB(0,255,0));
m_combo_brush=new CBrush;
m_combo_brush->CreateSolidBrush(RGB(255,255,255));
m_scope=(CStatic*)GetDlgItem(IDC_STATIC2);
```

```
m_scope_pen.CreatePen(PS_SOLID,2,RGB(230,255,230));
CRect r;
m_scope->GetClientRect(&r);
m_sin=new CPoint [r.right];
m_noise=new CPoint [r.right];
m_filter=new CPoint [r.right];
m_pulse=new CPoint [r.right];
m_pulse_noise=new CPoint [r.right];
m_pulse_filter=new CPoint [r.right];

for(int q=0; q<r.right-1;q++)
{
        m_pulse[q].x=q;
        if(q<r.right/4) m_pulse[q].y=r.bottom*2/3;
        if(q>r.right/4 && q<r.right/2) m_pulse[q].y=r.bottom/4;
        if(q>r.right/2 && q<3*r.right/4) m_pulse[q].y=r.bottom*2/3;
        if(q>r.right*3/4) m_pulse[q].y=r.bottom/4;
}

srand(time(NULL));
int random;
int k;
        for(int i=0; i<r.right-1;i=i+2)
        {
                random=rand()%(r.bottom/10)*pow(-1,rand()%3);
                k=(r.bottom/3)*sin(i*3.14*2/180);
                m_sin[i].x=i;
                m_sin[i].y=r.bottom/2+(r.bottom/3)*sin(i*3.14*2/180);
                m_noise[i].x=i;
        m_noise[i].y=r.bottom/2+(r.bottom/3)*sin(i*3.14*2/180)+random;
                m_pulse_noise[i].x=i;
                m_pulse_noise[i].y=m_pulse[i].y+random;
        }

m_spin.SetBase(10);
m_spin.SetRange(1,10);
m_spin.SetPos(5);
m_edit1.SetWindowText("5");
m_spin.SetBuddy(&m_edit1);

for(int n=0; n<=m_combo.GetCount();n++)
{
        m_combo.SetItemData(n,(n+1)*2);
}
m_combo.SetCurSel(1);
m_order=4;
```

Обработчик нажатия на Radio Button 1:

```
void CScopeDlg::OnRadio1()
{
        RedrawWindow();
```

```
        CStatic* st;
        st=(CStatic*)GetDlgItem(IDC_STATIC2);
        CClientDC dc(st);
        dc.SelectObject(HPEN(m_scope_pen));
        CRect r;
        st->GetClientRect(&r);
        dc.MoveTo(0,r.bottom/2);
        for(int i=0; i<r.right-1;i=i+2)
        {
                dc.LineTo(m_sin[i]);
        }
}
```

Обработчик нажатия на Radio Button 2:

```
void CScopeDlg::OnRadio2()
{
        RedrawWindow();
        CStatic* st;
        st=(CStatic*)GetDlgItem(IDC_STATIC2);
        CClientDC dc(st);
        dc.SelectObject(HPEN(m_scope_pen));
        CRect r;
        st->GetClientRect(&r);
        dc.MoveTo(0,r.bottom/2);
        for(int i=0; i<r.right-1;i=i+2)
        {
                dc.LineTo(m_noise[i]);
        }
}
```

Обработчик нажатия на элемент Spin:

```
void CScopeDlg::OnDeltaposSpin1(NMHDR* pNMHDR, LRESULT* pResult)
{
  NM_UPDOWN* pNMUpDown = (NM_UPDOWN*)pNMHDR;

  CButton *b1, *b2;
  RedrawWindow();
  b1=(CButton*)GetDlgItem(IDC_RADIO2);
  b2=(CButton*)GetDlgItem(IDC_RADIO5);
  RedrawWindow();
  CStatic* st;
  st=(CStatic*)GetDlgItem(IDC_STATIC2);
  CClientDC dc(st);
  dc.SelectObject(HPEN(m_scope_pen));
  CRect r;
  st->GetClientRect(&r);
  dc.MoveTo(0,r.bottom/2);
```

```
  int k;
  int random;

  for(int i=0; i<r.right-1;i=i+2)
  {
  random=rand()%(int)(r.bottom/(13-m_spin.GetPos())*pow(-1,rand()%3));
          k=(r.bottom/3)*sin(i*3.14*2/180);
          m_noise[i].x=i;
          m_noise[i].y=m_sin[i].y+random-25;
          m_pulse_noise[i].y=m_pulse[i].y+random-15;
          if(b1->GetCheck())  dc.LineTo(m_noise[i]);
          else if(b2->GetCheck()) dc.LineTo(m_pulse_noise[i]);
          else
                  {
                          b1=(CButton*)GetDlgItem(IDC_RADIO3);
                          b2=(CButton*)GetDlgItem(IDC_RADIO6);
                          if(b1->GetCheck())
                          OnRadio2();
                          else if(b2->GetCheck())
                          OnRadio5();
                  }
          }
  *pResult = 0;
}
```

Обработчик нажатия на Radio Button 3:

```
void CScopeDlg::OnRadio3()
{
  CButton *b;
  b=(CButton*)GetDlgItem(IDC_RADIO2);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO1);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO3);
  b->SetCheck(1);
  b=(CButton*)GetDlgItem(IDC_RADIO4);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO5);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO6);
  b->SetCheck(0);
  int temp;
  RedrawWindow();
  CStatic* st;
  st=(CStatic*)GetDlgItem(IDC_STATIC2);
  CClientDC dc(st);
  dc.SelectObject(HPEN(m_scope_pen));
  CRect r;
  st->GetClientRect(&r);
```

```
  dc.MoveTo(0,r.bottom/2);
  temp=0;
  for(int i=0; i<r.right-m_order*2;i=i+2)
          {
  m_filter[i].x=i;
  m_filter[i].y=(m_noise[i].y+m_noise[i+2].y+m_noise[i+4].y+m_noise[i+6].y)/4;
          for(int j=0; j<m_order*2; j=j+2)
          {
                  temp=temp+(m_noise[i+j].y)/m_order;
          }
          m_filter[i].y=temp;
          temp=0;
          }
          for(int k=0; k<r.right-m_order*2;k=k+2)
          {
                  dc.LineTo(m_filter[k]);
          }
}
```

Обработчик изменения выбора из выпадающего списка:

```
void CScopeDlg::OnSelchangeCombo1()
{
  m_order=m_combo.GetItemData(m_combo.GetCurSel());
  OnRadio3();
}
```

Обработчик нажатия на Radio Button 4:

```
void CScopeDlg::OnRadio4()
{
  RedrawWindow();
  CStatic* st;
  st=(CStatic*)GetDlgItem(IDC_STATIC2);
  CClientDC dc(st);
  dc.SelectObject(HPEN(m_scope_pen));
  CRect r;
  st->GetClientRect(&r);
  dc.MoveTo(0,r.bottom/2);
  for(int i=0; i<r.right-1;i=i+2)
  {
          dc.LineTo(m_pulse[i]);
  }
}
```

Обработчик нажатия на Radio Button 5:

```
void CScopeDlg::OnRadio5()
{
  RedrawWindow();
```

```
  CStatic* st;
  st=(CStatic*)GetDlgItem(IDC_STATIC2);
  CClientDC dc(st);
  dc.SelectObject(HPEN(m_scope_pen));
  CRect r;
  st->GetClientRect(&r);
  dc.MoveTo(0,r.bottom/2);
  int random;
  for(int i=0; i<r.right-1;i=i+2)
  {
  random=rand()%(int)(r.bottom/(13-m_spin.GetPos())*pow(-1,rand()%3));
  m_pulse_noise[i].x=i;
  m_pulse_noise[i].y=m_pulse[i].y+random-25;
  dc.LineTo(m_pulse_noise[i]);
  }
}
```

Обработчик нажатия на Radio Button 6:

```
void CScopeDlg::OnRadio6()
{
  CButton *b;
  b=(CButton*)GetDlgItem(IDC_RADIO2);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO1);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO3);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO4);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO5);
  b->SetCheck(0);
  b=(CButton*)GetDlgItem(IDC_RADIO6);
  b->SetCheck(1);
  int temp;
  RedrawWindow();
  CStatic* st;
  st=(CStatic*)GetDlgItem(IDC_STATIC2);
  CClientDC dc(st);
  dc.SelectObject(HPEN(m_scope_pen));
  CRect r;
  st->GetClientRect(&r);
  dc.MoveTo(0,r.bottom/2);
  temp=0;

  for(int i=0; i<r.right-m_order*2;i=i+2)
  {
  m_pulse_filter[i].x=i;
  m_filter[i].y=(m_noise[i].y+m_noise[i+2].y+m_noise[i+4].y+m_noise[i+6].y)/4;
  for(int j=0; j<m_order*2; j=j+2)
          {
```

```
                temp=temp+(m_pulse_noise[i+j].y)/m_order;
            }
            m_pulse_filter[i].y=temp;
            temp=0;
    }
    for(int k=0; k<r.right-m_order*2;k=k+2)
    {
            dc.LineTo(m_pulse_filter[k]);
    }
}
```

После компиляции программы появится диалоговое окно программы (рис. 1.8), с помощью которого можно исследовать работу цифрового фильтра скользящего среднего.

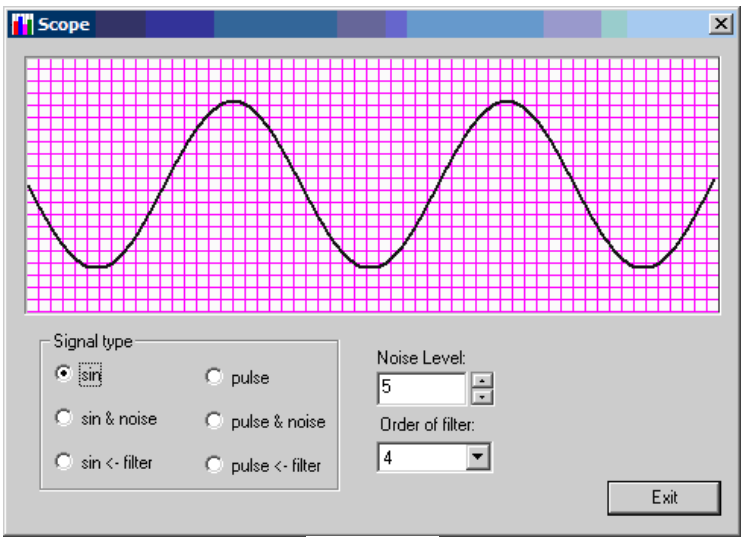
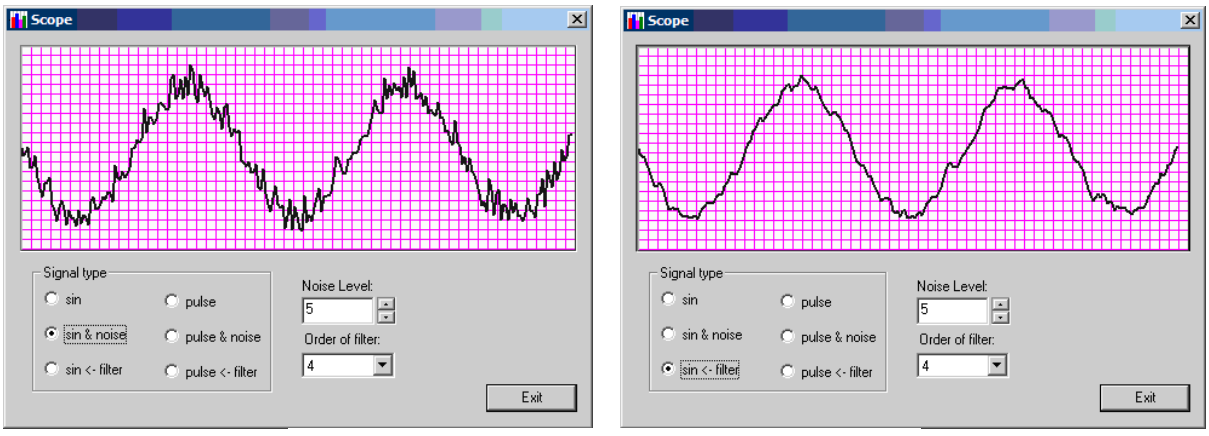


Рис. 1.8.

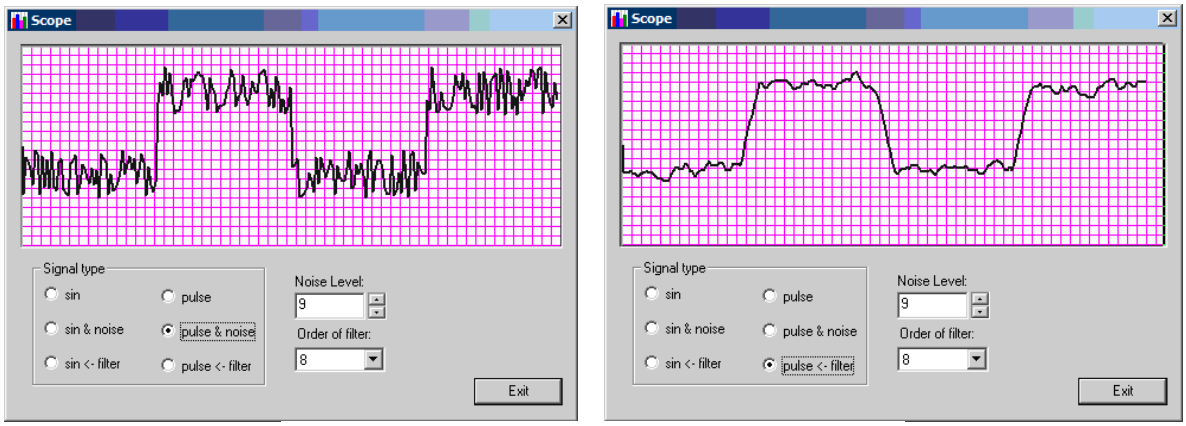
На рис. 1.9, а, представлен вид диалогового окна при выборе пользователем отображения синусоиды с шумом, а на рис. 1.9, б, показан результат обработки этого сигнала фильтром.



а *б*

Рис. 1.9.

На рис. 1.10, а, представлен вид диалогового окна при выборе пользователем отображения импульсного сигнала с шумом, а на рис. 1.10, б, показан результат обработки этого сигнала фильтром. Уровень шума выбран достаточно высоким, при этом используется фильтр 8 порядка.



а *б*

Рис. 1.10.

Нерекурсивные ЦФ обладают рядом положительных качеств, основное из которых состоит в том, что благодаря отсутствию обратных связей они всегда устойчивы. Действительно, передаточная функция нерекурсивного ЦФ

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{N-1} b_k z^{-k} ,$$

имеет N-1 полюсов, которые все находятся в точке z=0. Другой особенностью нерекурсивного фильтра является его конечная импульсная характеристика. Поэтому переходные процессы в фильтре затухают за конечный промежуток времени. Характерная особенность нерекурсивного фильтра состоит в том, что при соответствующем выборе параметров они могут иметь строго линейную фазо-частотную характеристику (ФЧХ). Это особенно важно в тех случаях, когда требуется обеспечить неискаженное преобразование сигналов.

Анализ полученных данных, в том числе и АЧХ, показывает, что рассмотренный цифровой фильтр обеспечивает высокое затухание, при отсутствии колебательности реакции на ступенчатое воздействие, характерное для аналоговых фильтров. Рассмотренный фильтр может быть реализован даже на 8-ми разрядных микроконтроллерах с относительно низким быстродействием.

Данный фильтр рекомендуется применять для приложений, где требуется определение средней величины действующего значения сигнала, выделения его постоянной составляющей.

1.3. Цифровые рекурсивные фильтры 2 порядка

Высококачественные частотные нерекурсивные цифровые фильтры (НЦФ) имеют, как правило, большую ширину окна. Чем меньше допустимая ширина переходной зоны частотной характеристики фильтра между полосами пропускания и подавления, тем больше окно фильтра. Альтернативное решение – применение рекурсивных цифровых фильтров, для которых количество коэффициентов фильтра может быть сокращено на несколько порядков по сравнению с НЦФ.

Рекурсивные фильтры имеют определенную "память" по значениям предыдущих отсчетов, которая, в пределе, может быть бесконечной. С учетом этого фактора рекурсивные фильтры получили название фильтров с бесконечной импульсной характеристикой (БИХ-фильтров), в отличие от нерекурсивных фильтров, всегда имеющих конечную импульсную характеристику (КИХ-фильтры). Реакция рекурсивного фильтра на сигнал с учетом "памяти" исключает возможность создания фильтров с четным импульсным откликом, и частотные характеристики рекурсивных фильтров всегда являются комплексными.

Среди множества рекурсивных фильтров по виду передаточной функции отдельно выделяют наиболее качественные фильтры:

– фильтры Бесселя – обладают наиболее гладкими АЧХ и ФЧХ (особенно в полосе пропускания), однако крутизна спада АЧХ у них наименьшая;

– фильтры Баттерворта – имеют более крутой спад АЧХ (6N дБ/октаву, N – порядок фильтра) и менее линейную ФЧХ;

– фильтры Чебышева – имеет еще более крутой спад АЧХ, однако, их АЧХ не монотонна, а имеет осцилляции заданного уровня в полосе пропускания, либо в полосе подавления. ФЧХ фильтров Чебышева немонотонна и имеет пик вблизи частоты среза. При задании меньших пульсаций фильтра крутизна спада АЧХ уменьшается, и фильтр Чебышева превращается в фильтр Баттерворта;

– эллиптические фильтры – обладают наиболее крутым спад АЧХ, но имеют пульсации АЧХ как в полосе пропускания, так и в полосе подавления. ФЧХ эллиптических фильтров не монотонна. При повышении требований к пульсациям этот фильтр превращается в фильтр Чебышева.

Проектирование рекурсивных частотных фильтров с заданными частотными характеристиками осуществляется с использованием z-преобразований.

Передаточная функция физически реализуемого цифрового фильтра с бесконечной импульсной характеристикой (БИХ-фильтра) может быть представлена в виде:

$$H(z) = \frac{\sum_{k=0}^{K-1} b_k z^{-k}}{1 + \sum_{m=0}^{M-1} a_m z^{-m}}.$$

Разностные уравнения такого фильтра можно записать в виде:

$$y(k)=\frac{1}{a_0}\left(\sum_{m=0}^{k} b_m x(k-m)-\sum_{m=1}^{k} a_m y(k-m)\right),$$

где $y(k)$ – отсчеты на выходе фильтра, $x(k)$ – входные отсчеты, b_m и a_m – коэффициенты числителя и знаменателя передаточной характеристики фильтра соответственно.

При построении БИХ-фильтра это уравнение можно записать в виде:

$$y(k)=\sum_{m=0}^{k}\frac{b_m}{a_0}\cdot x(k-m)-\sum_{m=1}^{k}\frac{a_m}{a_0}\cdot y(k-m)=\sum_{m=0}^{k} d_m\cdot x(k-m)-\sum_{m=1}^{k} c_m\cdot y(k-m),$$

$$d_m=\frac{b_m}{a_0},\ c_m=\frac{a_m}{a_0}.$$

Тогда БИХ-фильтр можно представить как сумму нерекурсивной и рекурсивной составляющих, как это показано на рис. 1.11.

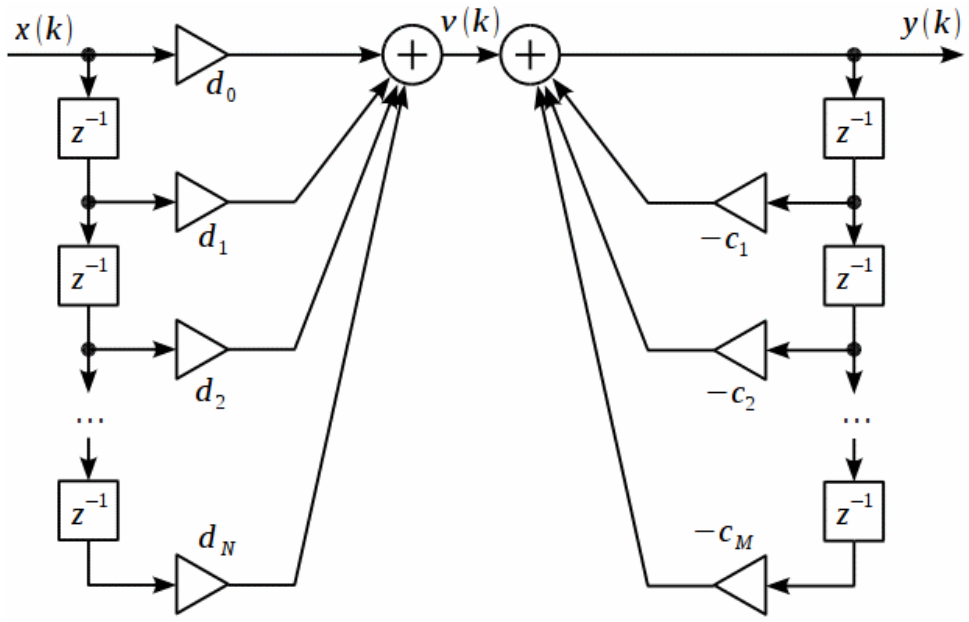


Рис. 1.11.

Вид дискретной передаточной функции БИХ-фильтра аналогичен виду непрерывной передаточной функции аналоговых фильтров. Поэтому можно использовать структуры цифровых БИХ-фильтров для для получения частотных характеристик, соответствующих классическим видам аппроксимации (фильтры Баттерворта, Чебышева, Бесселя и пр.). Однако, реализовать цифровой фильтр с частотной характеристикой, точно совпадающей с частотной характеристикой аналогового фильтра невозможно. Причиной этого является тот факт, что, в отличие от аналогового фильтра в диапазоне частот $0 \le f \le \infty$ цифровой фильтр обладает периодической частотной характеристикой с периодом, равным частоте дискретизации f_s. Однако используемая полоса частот у цифровых фильтров ограничивается

диапазоном $0 \le f \le \frac{1}{2} f_s$. Поэтому поставленную задачу можно видоизменить таким образом, чтобы частотная характеристика цифрового фильтра совпадала с требуемой частотной характеристикой аналогового фильтра лишь в диапазоне частот $0 \le f \le \frac{1}{2} f_s$.

Аналоговый фильтр можно преобразовать в цифровой следующим образом. В выражение для передаточной функции $H(p)$ нормированного аналогового фильтра вместо переменой p подставляется переменная $l\frac{z-1}{z+1}$ и получаем передаточную функцию $H(z)$ цифрового фильтра с аналогичной частотной характеристикой в диапазоне частот $0 \le f' \le \frac{1}{2}$.

Фазо-частотная характеристика при билинейном преобразовании изменяется сильнее. По этой причине не имеет смысла применять билинейное преобразование к фильтрам Бесселя, поскольку линейность ФЧХ в этом случае нарушается.

Стандартные блоки рекурсивных фильтров обычно реализуются биквадратными звеньями в канонической форме (рис. 1.12), которая имеет минимальное количество элементов задержки.

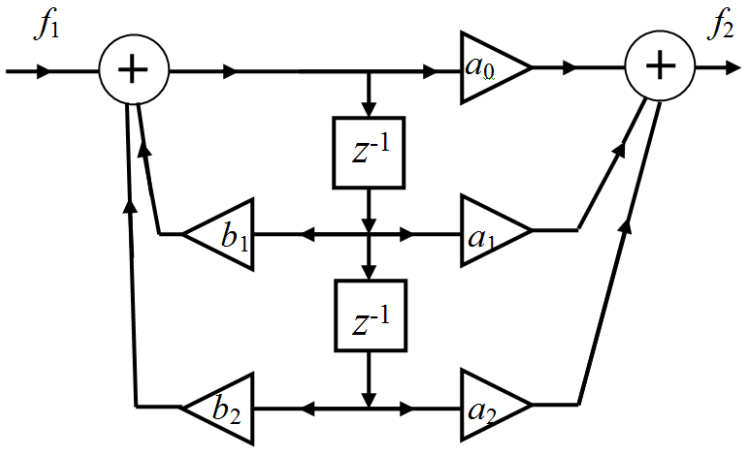


Рис. 1.12.

Рекурсивный цифровой фильтр второго порядка описывается разностным уравнением

$$y[n]=a_0x[n]+a_1x[n-1]+a_2x[n-2]+b_1y[n-1]+b_2y[n-2].$$

По известным значениям коэффициентов разностного уравнения a_k и b_k цифровой фильтр 2-го порядка может быть реализован с помощью умножителей, сумматоров и блоков задержки (рис. 1.12).

Произведем пересчет коэффициентов передаточной функции аналогового фильтра $H(p)$ в коэффициенты передаточной функции цифрового фильтра $H(z)$

для звеньев первого и второго порядка. Используя билинейное преобразование, из выражения для аналоговой передаточной функции:

$$H(p) = \frac{b_0 + b_1 p + b_2 p^2}{a_0 + a_1 p + a_2 p^2},$$

можно найти дискретную передаточную функцию:

$$H(z) = \frac{B_0 + B_1 z + B_2 z^2}{A_0 + A_1 z + A_2 z^2}.$$

Для фильтра первого порядка ($a_2 = b_2 = 0$) имеем:

$$B_0 = \frac{b_0 - b_1 l}{a_0 + a_1 l},\ A_0 = \frac{a_0 - a_1 l}{a_0 + a_1 l},$$

$$B_1 = \frac{b_0 + b_1 l}{a_0 + a_1 l},\ A_1 = 1,$$

$$B_2 = 0,\ A_2 = 0.$$

Для фильтра второго порядка ($c_2 \neq 0$) имеем:

$$B_0 = \frac{b_0 - b_1 l + b_2 l^2}{a_0 + a_1 l + a_2 l^2},\ A_0 = \frac{a_0 - a_1 l + a_2 l^2}{a_0 + a_1 l + a_2 l^2},$$

$$B_1 = \frac{2(b_0 - b_2 l^2)}{a_0 + a_1 l + a_2 l^2},\ A_1 = \frac{2(a_0 - a_2 l^2)}{a_0 + a_1 l + a_2 l^2},$$

$$B_2 = \frac{b_0 + b_1 l + b_2 l^2}{a_0 + a_1 l + a_2 l^2},\ A_2 = 1.$$

Порядок действий при определении передаточной функции $H(z)$ цифрового ФНЧ по заданным требованиям к максимально допустимой неравномерности АЧХ в полосе пропускания A_p, минимально допустимому затуханию в полосе задерживания A_a, а также граничным нормированным частотам полосы пропускания ω_p и полосы задерживания ω_a должен быть следующим.

Вначале определяются нормированные частоты полосы пропускания и полосы задерживания аналогичного аналогового фильтра. После этого, одним из способов синтеза аналоговых фильтров определяется аналоговая передаточная функция $H(p')$ фильтра-прототипа, удовлетворяющего заданным требованиям неравномерности АЧХ в полосе пропускания и затуханию в полосе задерживания. В заключение в определенной передаточной функции аналогового фильтра делается замена переменной p в соответствии с выражением:

$$p' = l\frac{1 - z^{-1}}{1 + z^{-1}} = l\frac{z - 1}{z + 1}.$$

1.4. Программная реализация рекурсивного фильтра 2 порядка

В редакторе ресурсов среды разработки приложений спроектируем диалоговое окно, как показано на рис. 1.13.

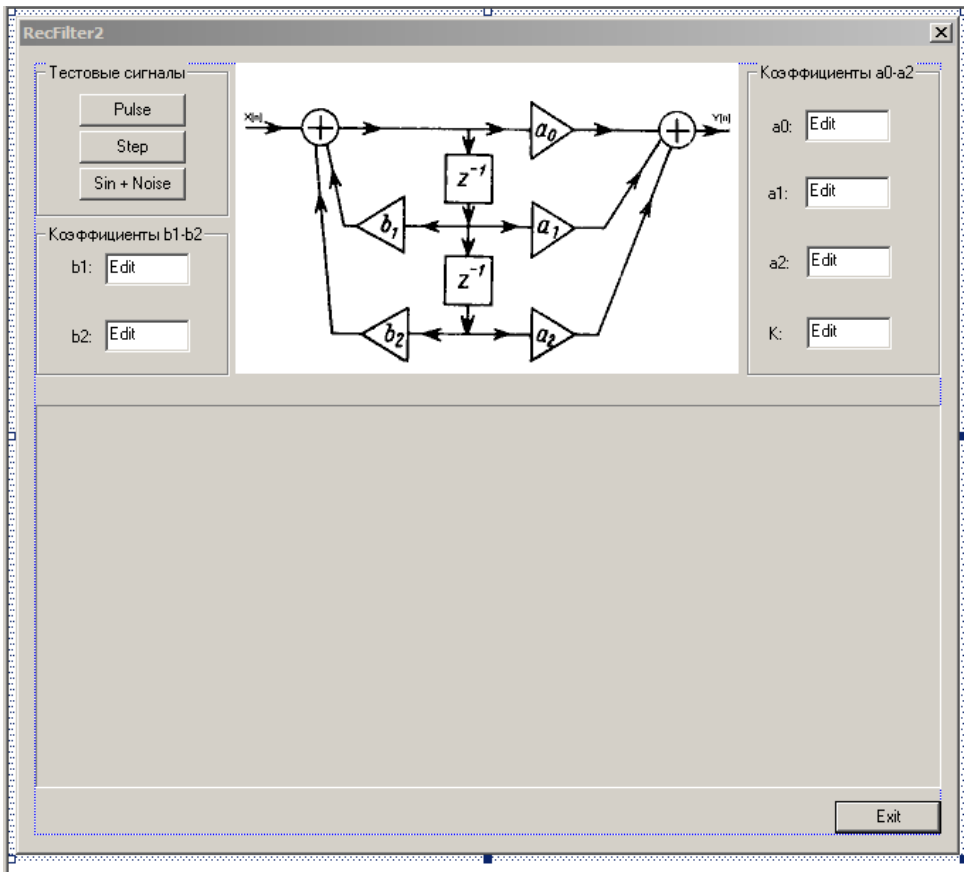


Рис. 1.13.

Изображение структуры цифрового рекурсивного фильтра 2 порядка на диалоговом окне используется исключительно в информационных целях для того, чтобы пользователь понимал назначение вводимых коэффициентов и их место в структуре фильтра.

Для ввода коэффициентов фильтра используется элемент Edit Box. Три кнопки, расположенные в левом верхнем углу позволят сформировать дельта импульс (Pulse), ступенчатое воздействие (Step) и сгенерировать синусоидальный сигнал с шумом (Sin + Noise). В нижней части диалогового окна расположен элемент Stztic Text, который будет использован как стилизированный экран осциллографа. Именно в этой области будут отображаться осциллограммы входных сигналов и выходной сигнал фильтра.

С элементами управления, расположенными на диалоге, связываем переменные. Для этого используется Class Wizard, который можно вызвать нажатием сочетания клавиш Ctrl+W. На закладке Member Variables необходимо для каждого элемента управления добавить переменную, как показано на рис. 1.14.

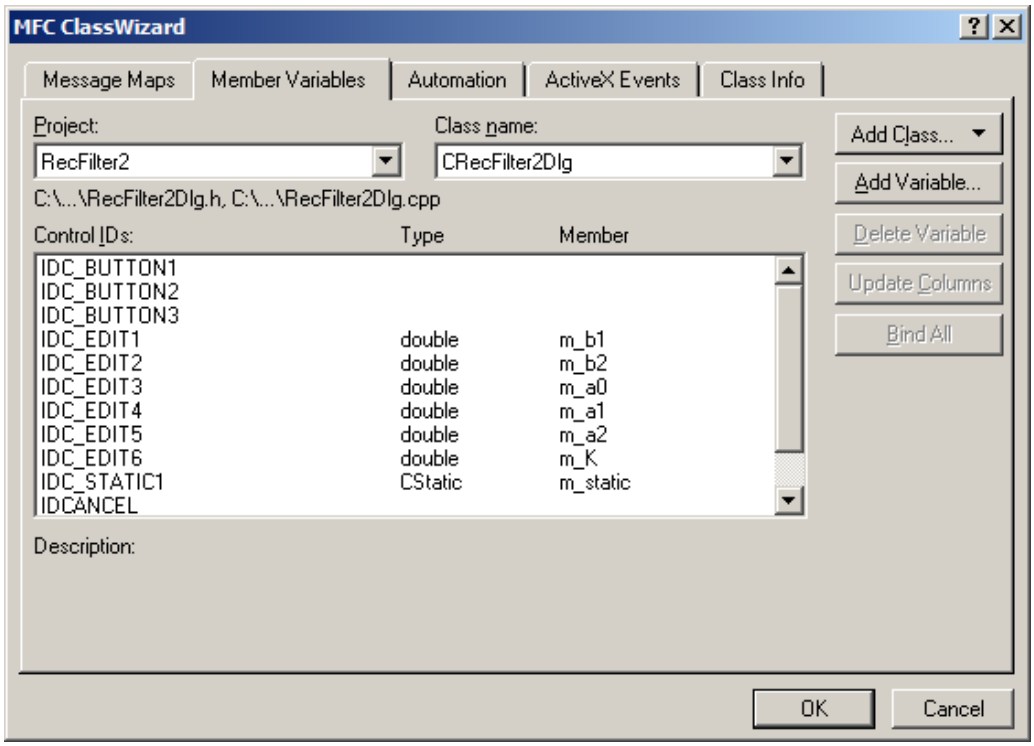


Рис. 1.14.

В класс диалога добавляем переменные:

```
int * X;  // указатель на массив входных значений
int N;            // количество выборок в массиве
```

и функцию:

```
void Filter();     // функция, реализующая цифровой фильтр
```

В функции OnInitDialog выполняется инициализация переменных. Для этого добавляем следующий код:

```
  CRect r;
  m_static.GetClientRect(&r);
  N=r.Width();
  X=new int[N];
  m_a0=0.29955;
  m_a1=0.059909;
  m_a2=0.029955;
  m_b1=1.454240;
  m_b2=-0.574062;
  m_K=0.5;
  UpdateData(false);
```

Добавляем обработчик нажатия на кнопку Pulse:

```
void CRecFilter2Dlg::OnButton1()
{
```

```
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  dc.FillRect(r,&CBrush(RGB(255, 255, 255)));
  CPen gp;
  gp.CreatePen(PS_SOLID,1,RGB(0, 150, 0));
  dc.SelectObject(&gp);
  for(int a=0;a<r.Width();a=a+10)
  {
          dc.MoveTo(a,0);
          dc.LineTo(a,r.Height());
  }
  for(int b=0;b<r.Height();b=b+10)
  {
          dc.MoveTo(0,b);
          dc.LineTo(r.Width(),b);
  }
  for(int i=0;i<N;i++)
  {
          if(i==N/2) X[i]=50;
          else X[i]=0;
  }
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(0, 0, 255));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
          dc.MoveTo(i,r.Height()/3-X[i]);
          dc.LineTo((i-1),r.Height()/3-X[i-1]);
  }
  dc.SelectObject(op);
  Filter();
}
```

Обработчик нажатия на кнопку Step имеет вид:

```
void CRecFilter2Dlg::OnButton2()
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  dc.FillRect(r,&CBrush(RGB(255, 255, 255)));
  CPen gp;
  gp.CreatePen(PS_SOLID,1,RGB(0, 150, 0));
  dc.SelectObject(&gp);
  for(int a=0;a<r.Width();a=a+10)
  {
          dc.MoveTo(a,0);
          dc.LineTo(a,r.Height());
  }
```

```
  for(int b=0;b<r.Height();b=b+10)
  {
        dc.MoveTo(0,b);
        dc.LineTo(r.Width(),b);
  }
  for(int i=0;i<N;i++)
  {
        if(i<N/2) X[i]=0;
        else X[i]=50;
  }
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(0, 0, 255));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
        dc.MoveTo(i,r.Height()/3-X[i]);
        dc.LineTo((i-1),r.Height()/3-X[i-1]);
  }
  dc.SelectObject(op);
  Filter();
}
```

Реализация функции обработки нажатия на кнопку Sin + Noise представлена ниже.

```
void CRecFilter2Dlg::OnButton3()
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  dc.FillRect(r,&CBrush(RGB(255, 255, 255)));
  CPen gp;
  gp.CreatePen(PS_SOLID,1,RGB(0, 150, 0));
  dc.SelectObject(&gp);
  for(int a=0;a<r.Width();a=a+10)
  {
        dc.MoveTo(a,0);
        dc.LineTo(a,r.Height());
  }
  for(int b=0;b<r.Height();b=b+10)
  {
        dc.MoveTo(0,b);
        dc.LineTo(r.Width(),b);
  }
  for(int i=0;i<N;i++)
  {
         X[i]=50*sin(6.28*i/100)+rand()%20;
  }
  CPen p, *op;
```

```
  p.CreatePen(PS_SOLID,2,RGB(0, 0, 255));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
        dc.MoveTo(i,r.Height()/3-X[i]);
        dc.LineTo((i-1),r.Height()/3-X[i-1]);
  }
  dc.SelectObject(op);
  Filter();
}
```

Функция, реализующая алгоритм цифрового рекурсивного фильтра 2 прядка представлена ниже.

```
void CRecFilter2Dlg::Filter()
{
  UpdateData(true);
  int * Y=new int[N];
  memset(Y,0,sizeof(int)*N);

  for(int n=2;n<N;n++)
  {
  Y[n]=m_b1*Y[n-1]+m_b2*Y[n-2]+m_a0*X[n]+m_a1*X[n-1]+m_a2*X[n-2];
  }
  for(n=0;n<N;n++)
  {
        Y[n]=Y[n]*m_K;
  }

  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  CRgn rgn;
  rgn.CreateRectRgn(r.left,r.top,r.right,r.bottom);
  dc.SelectClipRgn(&rgn);
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(255,0,0));
  op=dc.SelectObject(&p);
  for(int i=1;i<N;i++)
  {
        dc.MoveTo(i,3*r.Height()/4-Y[i]);
        dc.LineTo((i-1),3*r.Height()/4-Y[i-1]);
  }
  dc.SelectObject(op);
  delete [] Y;
}
```

После компиляции проекта откроется диалоговое окно программы. Нажатие на кнопку Pulse позволит увидеть реакцию фильтра на дельта-импульс при различных значениях коэффициентов (рис. 1.15).

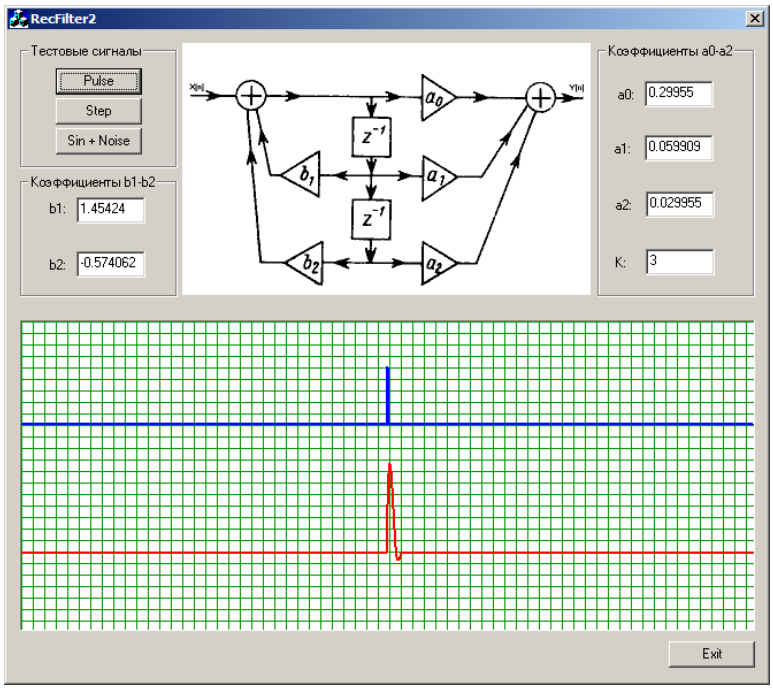


Рис. 1.15.

На рис. 1.16 и рис. 1.17 представлена реакция фильтра на ступенчатое воздействие при различных значениях коэффициентов фильтра. В первом случае переходной процесс носит ярко выраженный колебательный характер. Во втором случае характер переходного процесса приближается к апериодическому закону.

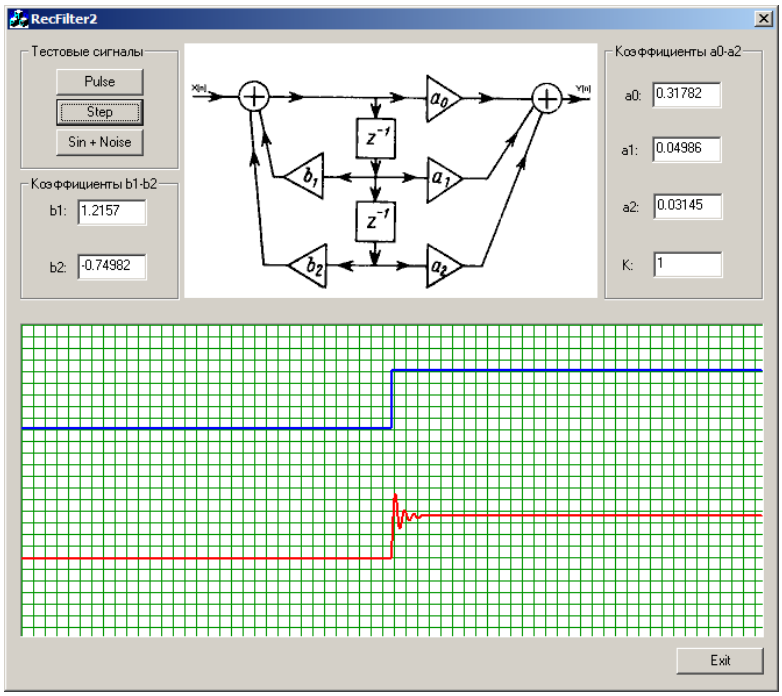


Рис. 1.16.

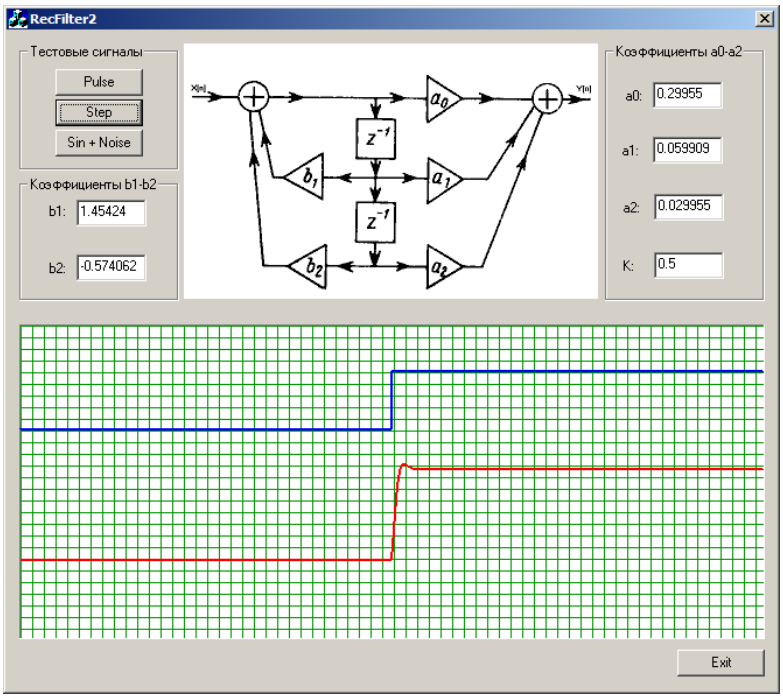


Рис. 1.17.

На рис. 1.18 представлено диалоговое окно программы, которая позволяет автоматизировать процесс проектирования рекурсивных фильтров путем расчета коэффициентов звеньев. В качестве примера с помощью этой программы получены значения коэффициентов для фильтра Чебышева 2 порядка с неравномерностью в полосе пропускания 5%.

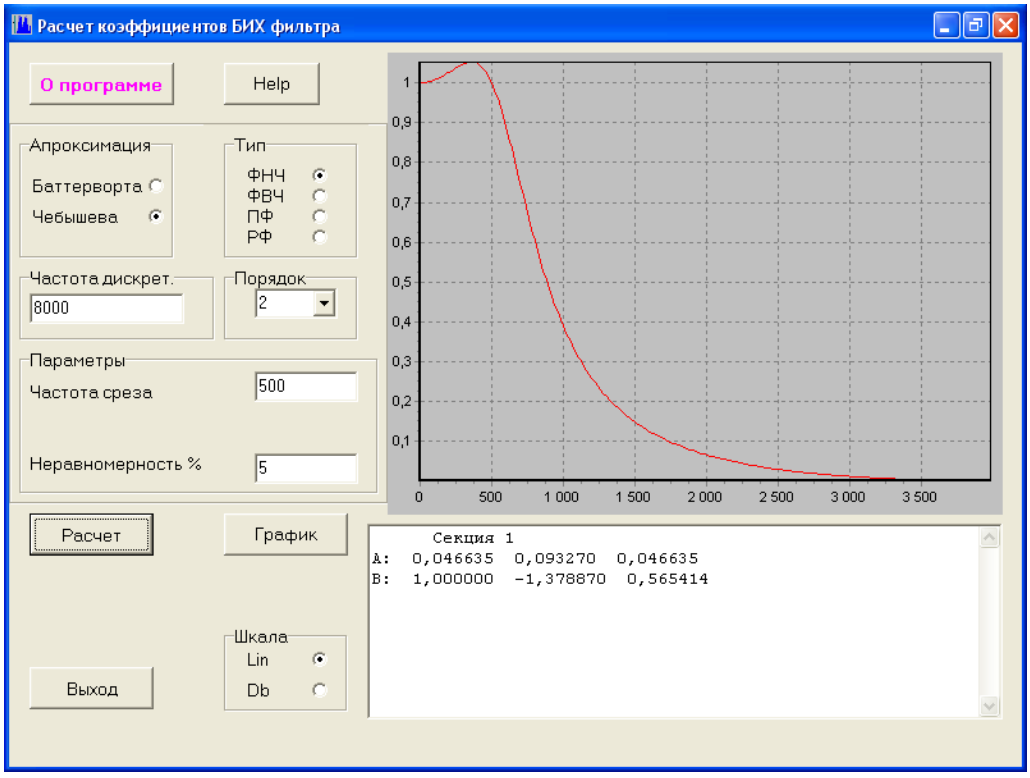


Рис. 1.18.

Полученные коэффициенты фильтра введены в разработанную программу. После подачи на вход фильтра синусоидального сигнала с шумом, можно увидеть результат фильтрации (рис. 1.19).

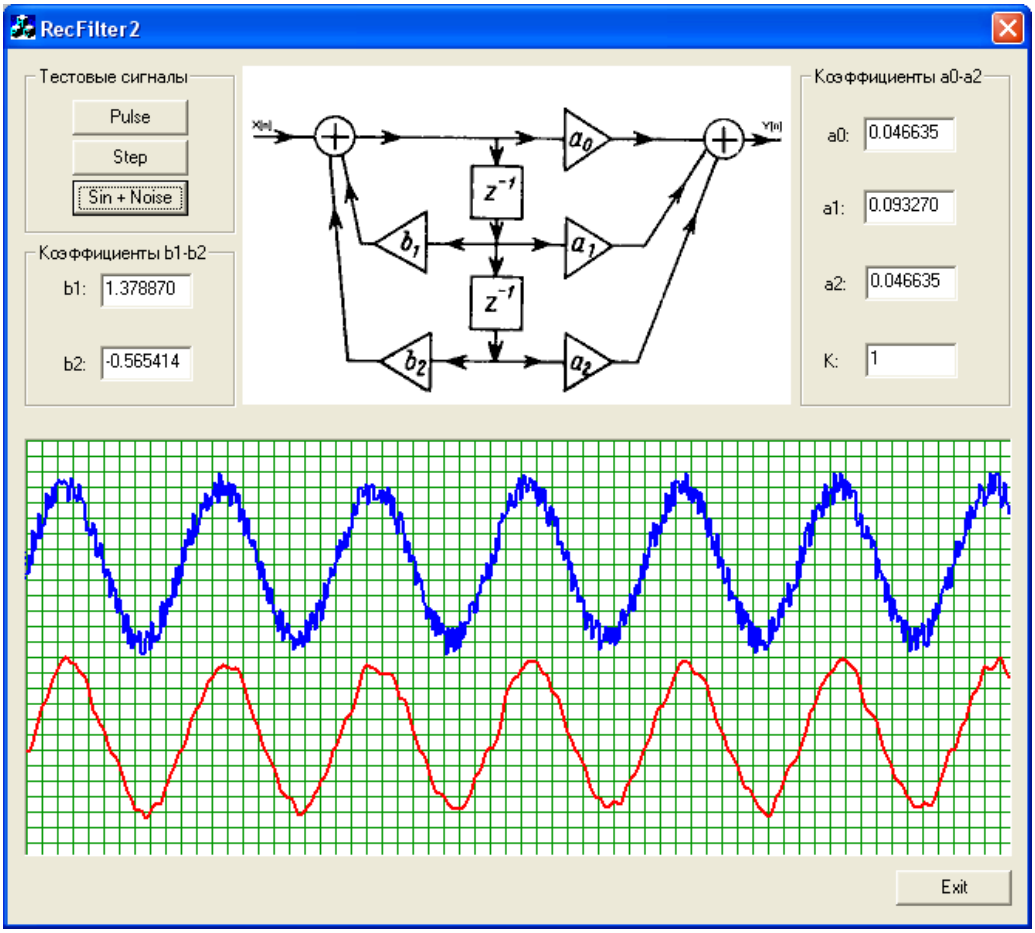


Рис. 1.19.

Примерный вид одного периода АЧХ такого фильтра для различных значений величины

$$r = \frac{b_1}{2} \pm \sqrt{\frac{b_1^2}{4} + b_2}\ ,$$

определяющей положение полюсов функции $K(p)$, показан на рис. 1.20.

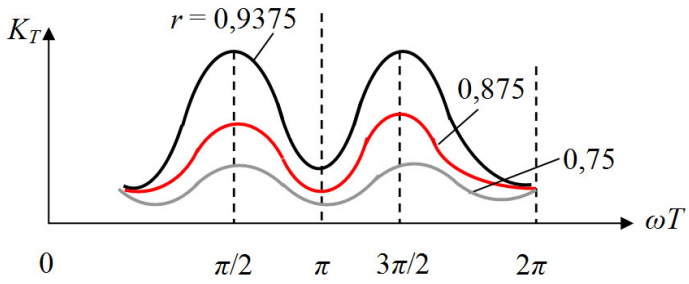


Рис. 1.20.

Варьируя коэффициенты фильтра, можно получать заданную передаточную характеристику. Так для a0=a2=1, a1=–2, при b1=0,21875, b2=0,4375, получается АЧХ, подобная АЧХ двух взаимно-связанных контуров (рис. 1.21) с достаточно хорошей прямоугольностью.

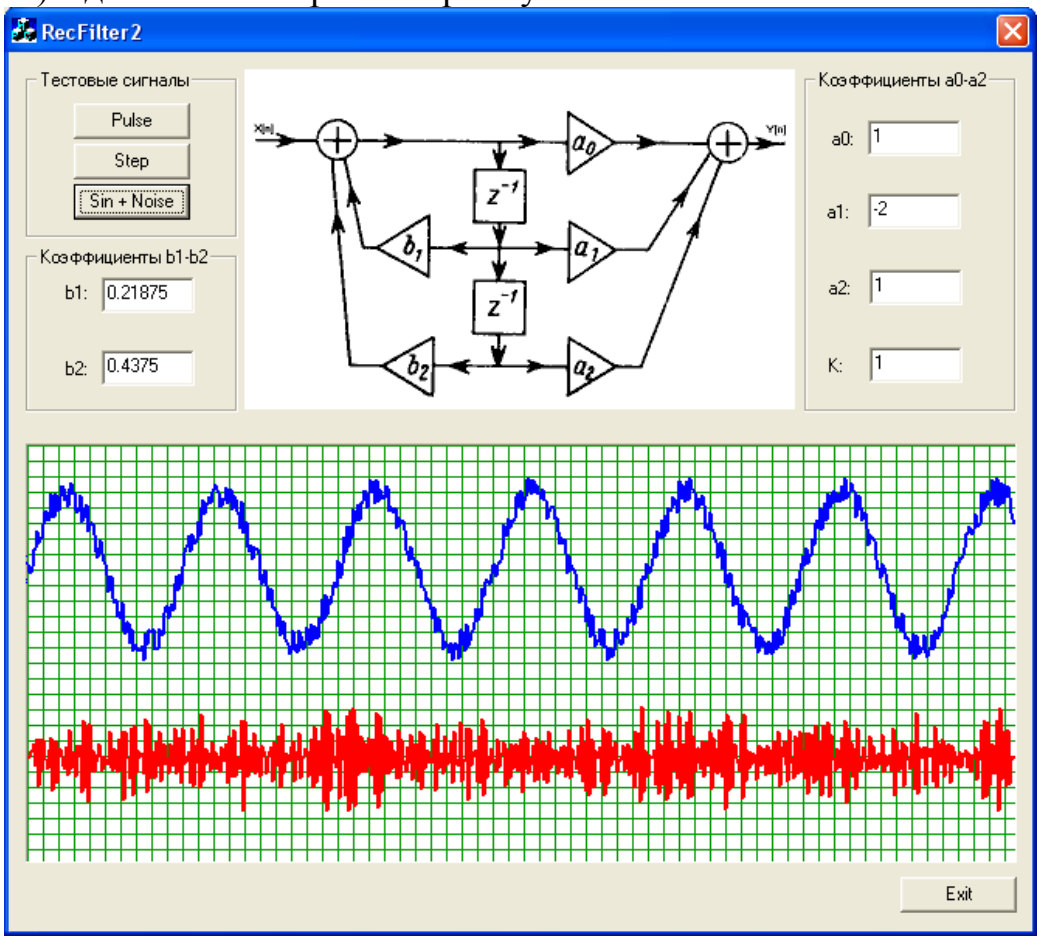


Рис. 1.21.

1.5. Программная реализация цифровых фильтров различных типов

Цифровые фильтры реализуются на основе всего трех элементов: задержка на один такт с передаточной функцией z^{-1}, сумматор и умножитель. При построении фильтров на этих элементах используются прямые и обратные связи, коэффициенты этих связей обозначаются, соответственно, как b_n и a_n.

Математически работа таких фильтров описывается разностным уравнением (уравнение в конечных разностях), как зависимость входного $x(n)$ и выходного $y(n)$ сигналов в функции времени задержки, коэффициентов фильтра и дискретного времени nT, где n – номер выборки, $T=1/f_s$, f_s – частота дискретизации.

Передаточная функция фильтра $H(z)$ определяется как отношение Z-образов выходного $Y(z)$ и входного $X(z)$ сигналов. Модуль передаточной функции $H(z)$ является частотной характеристикой фильтра (АЧХ), фазовая

характеристика определяется аргументом этой функции (ФЧХ). Цифровые фильтры могут работать в режиме усиления (boost), когда $|H(z)|>1$ и в режиме ослабления (cut), когда $|H(z)|<1$.

По тому, какие частоты фильтром пропускаются (задерживаются), фильтры, используемые при частотной коррекции, подразделяются на следующие группы.

– Фильтр низких частот (Lowpass – LP) выделяет нижние частоты до частоты среза и подавляет частоты выше этой частоты.

– Фильтр высоких частот (Highpass – HP) выделяет частоты выше частоты среза и подавляет частоты ниже этой частоты.

– Полосовой пропускающий фильтр (Bandpass – BP) выделяет частоты выше частоты среза $f_{\text{ср1}}$ и ниже частоты среза $f_{\text{ср2}}$. Частоты ниже $f_{\text{ср1}}$ и выше $f_{\text{ср2}}$ подавляются.

– Полосовой режекторный фильтр (Bandreject – BR) выделяет частоты выше частоты среза $f_{\text{ср2}}$ и ниже частоты среза $f_{\text{ср1}}$. Частоты ниже $f_{\text{ср2}}$ и выше $f_{\text{ср1}}$ подавляются.

– Узкополосный пропускающий фильтр (Resonator filter) пропускает частоты в

узкой полосе вблизи частоты среза.

– Узкополосный режекторный фильтр (Notch filter) подавляет частоты в узкой полосе вблизи частоты среза.

– Все пропускающий фильтр – фазовый фильтр (Allpass filter) пропускает все частоты, но изменяет фазу выходного сигнала.

При цифровой фильтрации наиболее широко используются рекурсивные фильтры, работа которых основана использовании частотно-зависимой отрицательной обратной связи. Они очень быстрые и в основном используются в программных продуктах, предназначенных для работы в реальном масштабе времени. Порядок этих фильтров может достигать 30.

Работа нерекурсивных КИХ-фильтров основана на использовании математической операции свертки. Они обеспечивают очень хорошую фильтрацию при отсутствии фазовых искажений, но имеют низкое быстродействие. Применяются в эквалайзерах, когда не требуется большая точность фильтрации.

В редакторе ресурсов среды разработки Visual Studio 6.0 проектируем интерфейс диалогового окна, примерный вид которого показан на рис. 1.22. Осциллограммы сигналов до и после применения фильтра будут отображаться на элементе Satic Text.

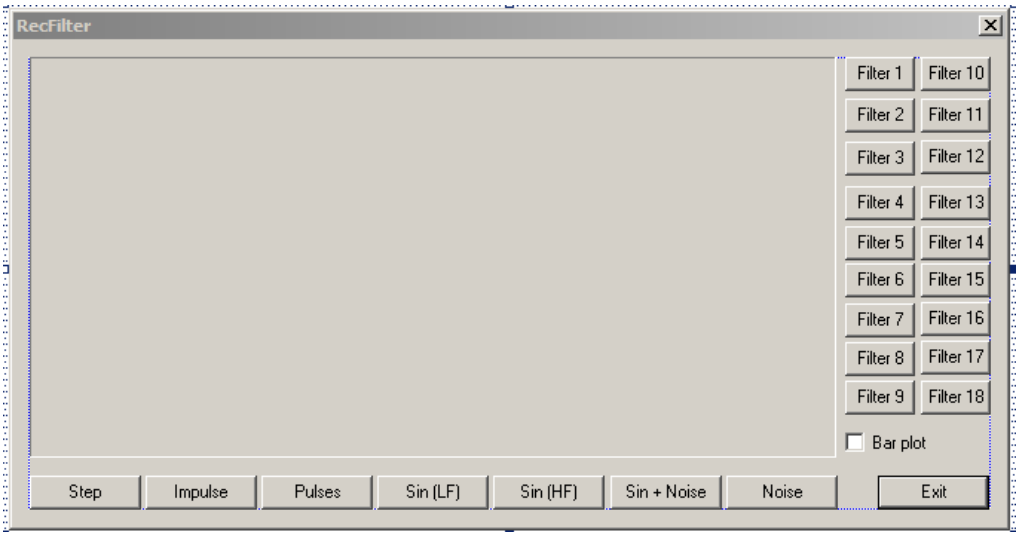


Рис. 1.22.

В класс диалога необходимо добавить 3 переменные и 2 функции:

```
int N;              // размер массива с выборками
int * X;  // указатель на буфер входных данных
int * Y;  // указатель на буфер выходных данных
// функция построения осциллограммы выходного сигнала фильтра
void DrawResult(COLORREF col);
// функция построения осциллограммы входного сигнала
void DrawGraph();
```

В функции OnInitDialog() необходимо инициализировать переменные и динамически выделить память для хранения выборок входного сигнала и результата фильтрации. Для этого добавляется следующий код:

```
  CRect r;
  m_static.GetClientRect(&r);
  N=r.Width();
  X=new int[N];
  Y=new int[N];
```

Для удобства построения осциллограмм размер буфера вхдных данных выбран равной ширине области построения.

Реализация функции DrawGraph() представлена ниже.

```
void CRecFilterDlg::DrawGraph()
{
  CClientDC dc(&m_static);

  CRect r;
  m_static.GetClientRect(&r);
  dc.FillRect(r,&CBrush(RGB(255,255,255)));
```

```
  dc.MoveTo(0,r.Height()/3);
  dc.LineTo(r.Width(),r.Height()/3);

  CPen gp;
  gp.CreatePen(PS_SOLID,1,RGB(0,150,0));
  dc.SelectObject(&gp);
  for(int a=0;a<r.Width();a=a+10)
  {
          dc.MoveTo(a,0);
          dc.LineTo(a,r.Height());
  }
  for(int b=0;b<r.Height();b=b+10)
  {
          dc.MoveTo(0,b);
          dc.LineTo(r.Width(),b);
  }
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(0,0,255));
  op=dc.SelectObject(&p);
  for(int i=1;i<N/4;i++)
  {
          dc.MoveTo(i*4,r.Height()/3-X[i]);
          dc.LineTo((i-1)*4,r.Height()/3-X[i-1]);
  }
  dc.SelectObject(op);
}
```

Реализация функции DrawResult():

```
void CRecFilterDlg::DrawResult(COLORREF col)
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  dc.MoveTo(0,3*r.Height()/4);
  dc.LineTo(r.Width(),3*r.Height()/4);
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,col);
  op=dc.SelectObject(&p);
  if(m_check.GetCheck()==0)
  {
          for(int i=1;i<N/4;i++)
          {
                  dc.MoveTo(i*4,3*r.Height()/4-Y[i]);
                  dc.LineTo((i-1)*4,3*r.Height()/4-Y[i-1]);
          }
  }
  else
  {
```

```
        for(int i=0;i<N/4;i++)
        {
                dc.MoveTo(i*4,3*r.Height()/4);
                dc.LineTo(i*4,3*r.Height()/4-Y[i]);
        }
  }
  dc.SelectObject(op);
}
```

Для исследования работы цифровых фильтров на вход необходимо подавать различные сигналы. На диалоговом окне расположены кнопки, после нажатия на которые массив входных значений сигнала X[i] заполняется выборками. Обработчик нажатия на кнопку Step (ступенчатая функция):

```
void CRecFilterDlg::OnButton1()
{
  for(int i=0;i<N/4;i++)
  {
        if(i<N/8) X[i]=0;
        else X[i]=50;
  }
  DrawGraph();
}
```

Обработчик нажатия на кнопку Impulse (дельта-импульс):

```
void CRecFilterDlg::OnButton3()
{
  for(int i=0;i<N/4;i++)
  {
        if(i==N/8) X[i]=50;
        else X[i]=0;
  }
  DrawGraph();
}
```

Обработчик нажатия на кнопку Pulses (прямоугольные импульсы):

```
void CRecFilterDlg::OnButton5()
{
  bool flag=false;
  for(int i=0;i<N/4;i++)
  {
        if((i%20==0) && (flag==false)) flag=true;
        else if((i%20==0) && (flag==true)) flag=false;
        if(flag) X[i]=50;
        else X[i]=0;
  }
```

```
  DrawGraph();
}
```

Обработчик нажатия на кнопку Sin (LF) (низкочастотный синусоидальный сигнал):

```
void CRecFilterDlg::OnButton6()
{
  for(int i=0;i<N/4;i++)
  {
        X[i]=30*sin(6.28*i/60);
  }
  DrawGraph();
}
```

Обработчик нажатия на кнопку Sin (HF) (высокочастотный синусоидальный сигнал):

```
void CRecFilterDlg::OnButton10()
{
  for(int i=0;i<N/4;i++)
  {
        X[i]=30*sin(6.28*i/10);
  }
  DrawGraph();
}
```

Обработчик нажатия на кнопку Sin + Noise (смесь низкочастотной синусоиды с белым шумом):

```
void CRecFilterDlg::OnButton7()
{
  for(int i=0;i<N/4;i++)
  {
        X[i]=30*sin(6.28*i/40)+(rand()%30-15);
  }
  DrawGraph();
}
```

Обработчик нажатия на кнопку Noise (белый шум):

```
void CRecFilterDlg::OnButton9()
{
  for(int i=0;i<N/4;i++)
  {
        X[i]=rand()%50-25;
  }
  DrawGraph();
}
```

В правой части диалогового окна расположены кнопки, нажатие на которые приведет к запуску процесса цифровой фильтрации. Для удобства исследования различных структур фильтров и изучения особенностей программной реализации, в каждом из обработчиков нажатия на кнопку будет реализована отдельная структура фильтра. Кроме того, программа построена таким образом, что на осциллограмме можно отобразить результат обработки входного сигнала различными фильтрами, что позволит также сравнить их эффективность. Чтобы стереть осциллограммы выходных сигналов необходимо заново сгенерировать входной сигнал путем нажатия на одну из кнопок в нижней части диалогового окна. Также для удобства анализа осциллограммы сигналов, полученные различными фильтрами, изображаются разными цветами. Структуры фильтров, реализованные далее, взяты из книги [Вологдин Э.И. Методы и алгоритмы обработки звуковых сигналов. Курс лекций. – Санкт-Петербург, 2012. – 96 с.: ил.].

Простейшая схема фильтра ФНЧ 1 порядка с одним нулем на основе одного элемента задержки и сумматора приведена на рис. 1.23.

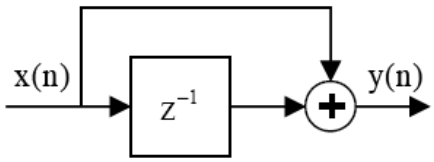


Рис. 1.23.

В этой схеме используется прямая связь, с помощью которой суммируются прямой и задержанный сигналы. Работа такого фильтра нижних частот описывается разностным уравнением:

$$y(n) = x(n) + x(n-1),$$

Порядок этого уравнения определяет порядок фильтра. Передаточная функция фильтра в форме Z-преобразования имеет вид:

$$H(z) = 1 + z^{-1}.$$

Частотная и фазовая характеристики этого фильтра определяются равенствами:

$$H(\eta) = 2 \cdot \cos(\pi \cdot \eta),$$
$$\arg[H(\eta)] = -\pi \cdot \eta,$$

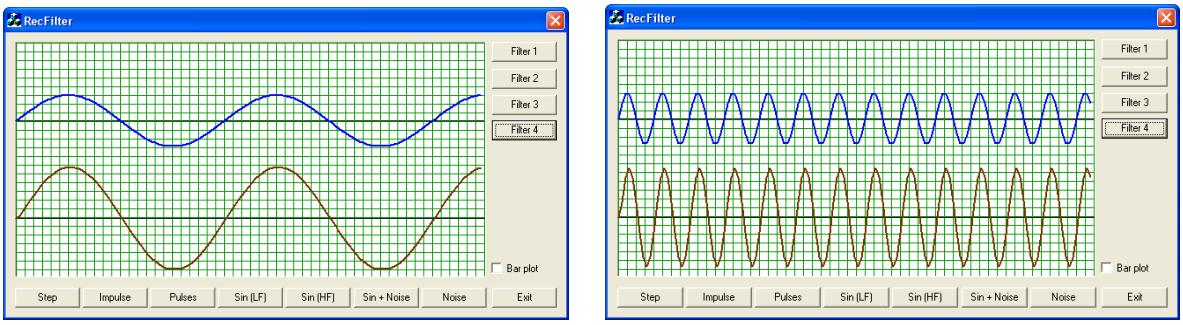
где $\eta = \frac{f}{f_s}$ – относительная частота.

Программная реализация фильтра имеет вид:

```
void CRecFilterDlg::OnButton11()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=1;i<N/4;i++)
  {
```

```
            Y[i]=X[i]+X[i-1];
    }
    DrawResult(RGB(120,50,0));
}
```

На рис. 1.24 показаны осциллограммы сигналов на входе и выходе фильтра. В первом случае на вход фильтра подается низкочастотный гармонический сигнал (рис. 1.24, а), во втором – высокочастотный (рис. 1.24, б). Как видно из рисунков, амплитуда сигналов мало зависит от частоты, и в рассмотренном случае происходит усиление сигнала.



а *б*

Рис. 1.24.

Для расширения функций такого фильтра и возможности изменения передаточной функции в фильтр включаются два умножителя, с помощью которых вводятся коэффициенты b_0 и b_1 (рис. 1.25).

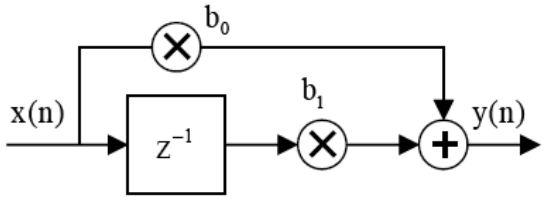


Рис. 1.25.

При этом разностное уравнение, его Z-преобразование и передаточная функция принимают вид:

$$y(n) = b_0 x(n) + b_1 x(n-1),$$

$$Y(z) = b_0 X(z) + b_1 z^{-1} X(z),$$

$$H(z) = b_0 + b_1 z^{-1}.$$

Если коэффициенты b_0 и b_1 равны 1 – это фильтр низких частот. Если b_0=1, а b_1= –1 – это уже фильтр высоких частот (рис. 1.26).

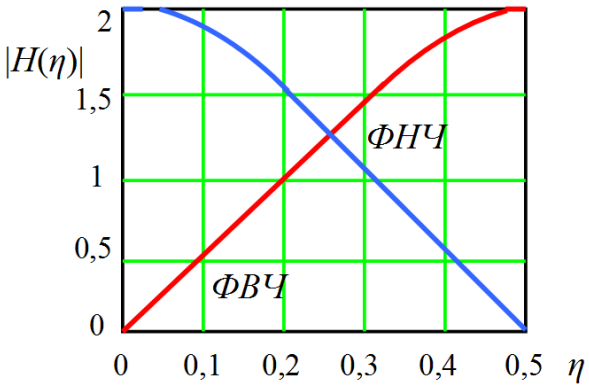


Рис. 1.26.

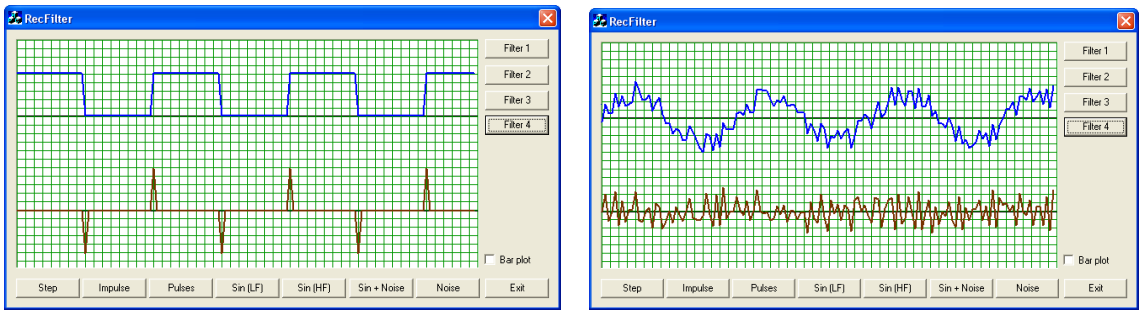
Программная реализация ФНЧ:

```
void CRecFilterDlg::OnButton11()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=1;i<N/4;i++)
  {
          Y[i]=0.5*X[i]+0.5*X[i-1];
  }
  DrawResult(RGB(120,50,0));
}
```

Программная реализация ФВЧ:

```
void CRecFilterDlg::OnButton11()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=1;i<N/4;i++)
  {
          Y[i]=0.5*X[i]-0.5*X[i-1];
  }
  DrawResult(RGB(120,50,0));
}
```

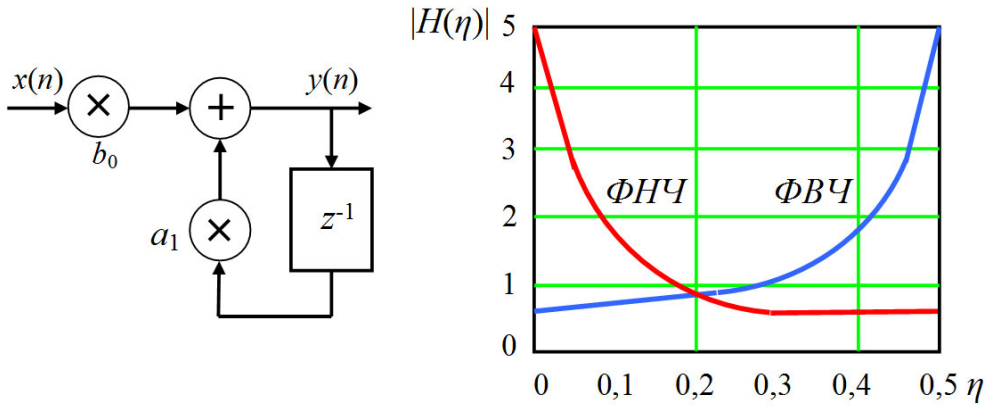
На рис. 1.27 представлены примеры, демонстрирующие работу программной реализации фильтра верхних частот. В первом случае на вход фильтра поступает импульсный сигнал. Анализ осциллограммы показывает, что на выходе фильтра появляется импульс только при изменении входного сигнала, т.е. фильтр представляет собой дифференцирующую цепочку. Второй пример демонстрирует обработку фильтром зашумленного синусоидального сигнала. Низкочастотная составляющая в этом случае ослабляется, а высокочастотный шум пропускается фильтром.



а *б*

Рис. 1.27.

Другой вариант построения фильтра ФНЧ/ФВЧ 1 порядка с одним полюсом приведен на рис. 1.28, а.



а *б*

Рис. 1.28.

В этой схеме используется обратная связь, с помощью которой суммируются прямой и задержанный сигналы, поэтому в разностном уравнении появляется член с отрицательным знаком и коэффициентом a_1:

$$y(n) = b_0 x(n) + a_1 y(n-1)\,.$$

Z – преобразование этого уравнения и передаточная функция этого фильтра определяются равенствами:

$$Y(z) = b_0 X(z) + a_1 z^{-1} Y(z)\,,$$

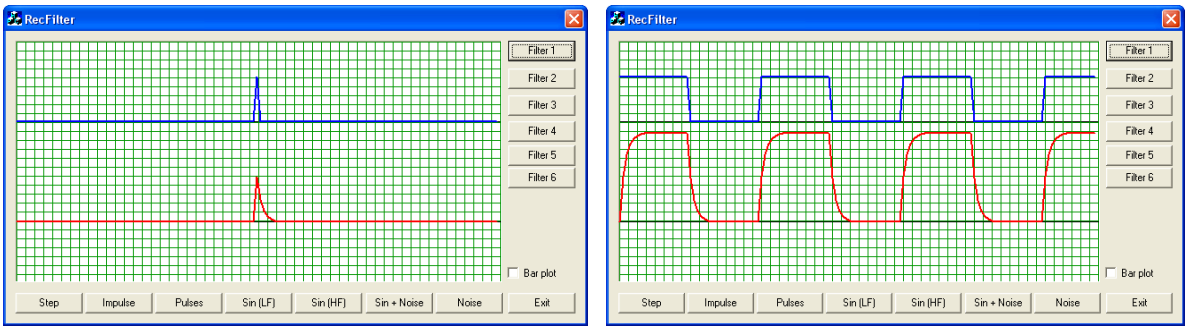
$$H(z) = \frac{b_0}{1 - a_1 z^{-1}}\,.$$

Для этой схемы фильтра, если $a_1>0$ – это ФНЧ, если $a_1<0$ – это ФВЧ. Данный фильтр имеет полюс при $z=-a_1$ как в режиме ФВЧ, так и в режиме ФНЧ, фильтр имеет ноль при $z=0$. На рис. 1.28, б, приведены АЧХ этого фильтра в режимах ФНЧ и ФВЧ. Как видно, они имеют совершенно другой вид по сравнению с графиками на рис. 1.26. Программная реализация фильтра представлена ниже.

```
void CRecFilterDlg::OnButton2()
{
memset(Y,0,sizeof(int)*N);
for(int i=1;i<N/4;i++)
{
  Y[i]=X[i]+0.5*Y[i-1];
}
DrawResult(RGB(255,0,0));
}
```

В программе реализован фильтр, для которого значения коэффициентов a_1=0,5, b_0=1.

На рис. 1.29, а, представлено окно программы с импульсной характеристикой фильтра, а на рис. 1.29, б – результат обработки цифровым фильтром нижних частот прямоугольных импульсов.



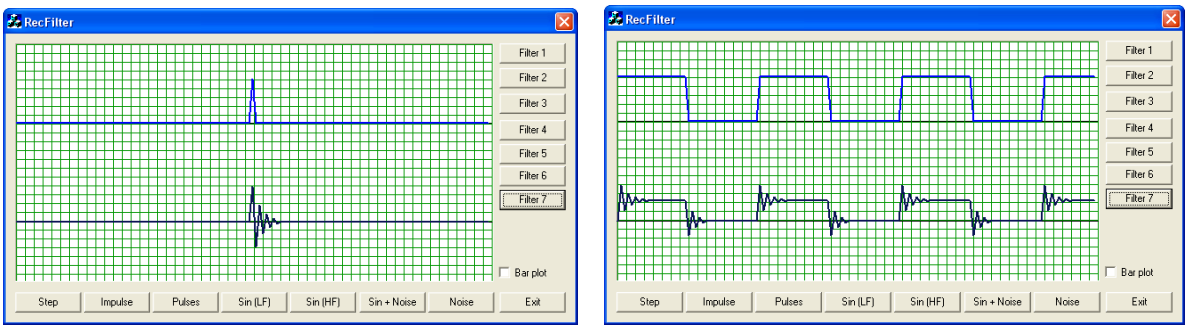
а *б*

Рис. 1.29.

Программа реализации фильтра, для которого значения коэффициентов a_1=–0,7, b_0=0,8, представлена ниже.

```
void CRecFilterDlg::OnButton14()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=1;i<N/4;i++)
  {
          Y[i]=0.8*X[i]-0.7*Y[i-1];
  }
  DrawResult(RGB(10,20,85));
}
```

На рис. 1.30, а, представлено окно программы с импульсной характеристикой этого фильтра, а на рис. 1.30, б – результат обработки цифровым фильтром прямоугольных импульсов.



а *б*

Рис. 1.30.

Фильтры 1 порядка обычно являются составными элементами более сложных фильтров.

Фильтр второго порядка с двумя нулями строится из двух звеньев фильтра (ФНЧ/ФВЧ) 1 порядка с использованием двух прямых связей (рис. 1.31, а). Его разностное уравнение, Z-преобразование и передаточная функция описываются равенствами:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2),$$

$$Y(z) = b_0 X(z) + b_1 z^{-1} X(z) + b_2 z^{-2} X(z),$$

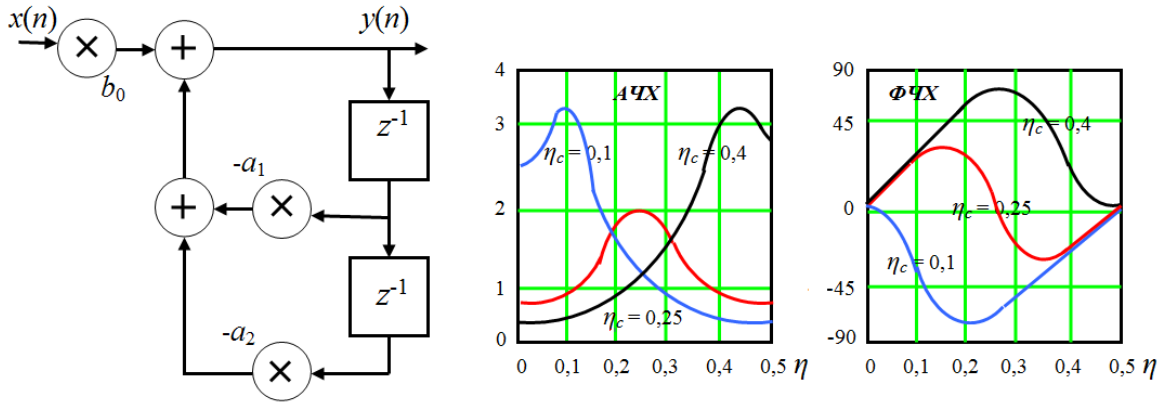
$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2}.$$

Если $(b_1/2)^2 > b_2$ эти равенства можно представить в виде:

$$y(n) = b_0 \{x(n) - [2R\cos(\theta_c)] \cdot x(n-1) + R^2 \cdot x(n-2)\},$$

$$H(z) = b_0 [1 - 2R\cos(\theta_c) \cdot z^{-1} + R^2 \cdot z^{-2}],$$

где $R = \sqrt{b_2 / b_0}$, $R \le 1$, $\theta_c = 2\pi \cdot \eta_c$, $\eta_c = f_c / f_s$, f_c – частота среза фильтра, f_s – частота дискретизации.



а *б* *в*

Рис. 1.31.

Такое преобразование позволяет задавать частоту среза. Если $\eta_c < 0{,}25$ фильтр осуществляет подъем АЧХ в области высоких частот, если $\eta_c < 0{,}25$

фильтр осуществляет подъем АЧХ в области низких частот, при $\eta_c = 0{,}25$ фильтр приобретает свойства заграждающего. Величина R определяет максимальный подъем АЧХ на краях звукового диапазона и максимальное затухание, которое имеет место на частоте близкой или равной частоте среза. Фазовый сдвиг равен нулю на частоте максимального затухания фильтра, изменения фазы с частотой тем сильнее, чем ближе значение R к единице. На рис. 1.31, б, в, приведены АЧХ и ФЧХ при R=0,6 и трех значениях частоты среза $\eta_c = 0{,}1;0{,}25;0{,}4$.

Программная реализация фильтра верхних частот, который образуется при значениях коэффициентов b_0=0,8, b_1=-0,7, b_2=0,1 представлена ниже.

```
void CRecFilterDlg::OnButton15()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=2;i<N/4;i++)
  {
          Y[i]=0.8*X[i]-0.7*X[i-1]+0.1*X[i-2];
  }
  DrawResult(RGB(255,135,0));
}
```

На рис. 1.32 представлен результат работы фильтра при подаче на его вход низкочастотного синусоидального сигнала с высокочастотным шумом.

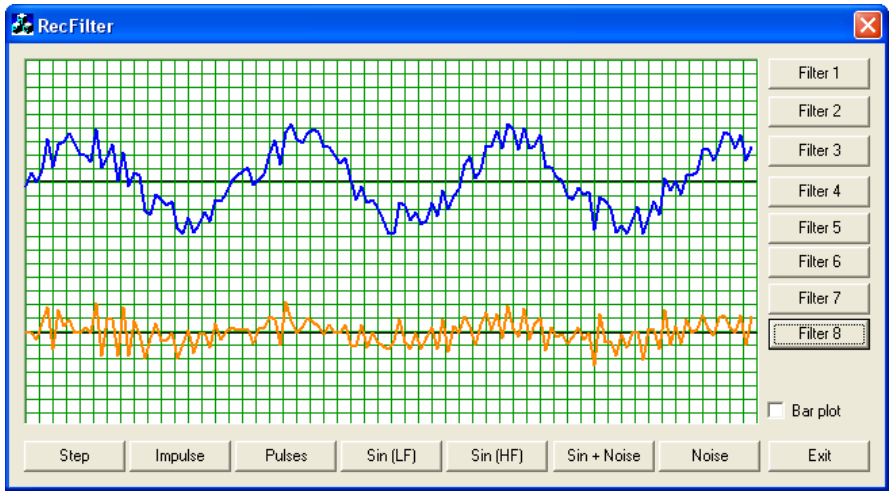
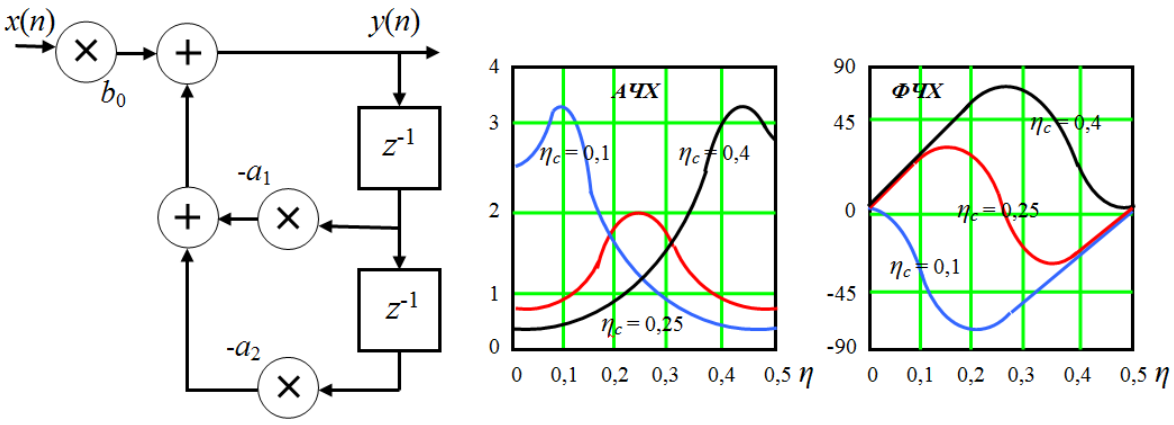


Рис. 1.32.

Фильтр второго порядка с двумя полюсами строится из двух звеньев фильтра (ФНЧ/ФВЧ) 1 порядка с использованием двух обратных связей (рис. 1.33).



а *б* *в*

Рис. 1.33.

Его разностное уравнение, Z-преобразование и передаточная функция описываются равенствами:

$$y(n) = b_0 x(n) - a_1 y(n-1) - a_2 y(n-2),$$

$$Y(z) = b_0 X(z) - a_1 z^{-1} Y(z) - a_2 z^{-2} Y(z),$$

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}}.$$

Если $(a_1/2)^2 > a_2$ эти равенства можно представить в виде:

$$y(n) = b_0 x(n) + [2R\cos(\theta_c)] \cdot y(n-1) - R^2 \cdot y(n-2),$$

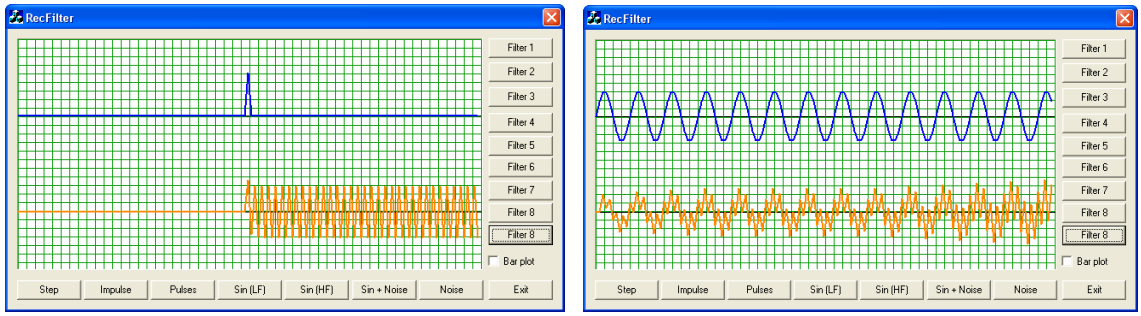
$$H(z) = \frac{b_0}{1 - 2R\cos(\theta_c) \cdot z^{-1} + R^2 \cdot z^{-2}},$$

где $R = \sqrt{a_2}$, $R < 1$, $\cos(\theta_c) = -\frac{a_1}{2R}$. Такое преобразование позволяет задавать частоту среза. Если $\eta_c > 0{,}25$ – это ФВЧ, если $\eta_c < 0{,}25$ – это ФНЧ, при $\eta_c = 0{,}25$ фильтр становится резонансным. Величина R в данном фильтре выполняет функцию демпфера, чем больше ее значение, тем выше и уже всплеск на частоте среза. Изменение фазы с частотой тем сильнее, чем ближе значение R к единице. На рис. 1.33, б, в, приведены АЧХ и ФЧХ при R=0,7 и трех значениях частоты среза: $\eta_c = 0{,}1; 0{,}25; 0{,}4$.

Программная реализация цифрового фильтра, который образуется при значениях коэффициентов b_0=0,75, a_1=0,72, a_2= –0,31 представлена ниже.

```
void CRecFilterDlg::OnButton16()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=2;i<N/4;i++)
  {
        Y[i]=0.75*X[i]-0.72*Y[i-1]+0.31*Y[i-2];
  }
  DrawResult(RGB(255,135,0));
}
```

На рис. 1.34, а, представлена импульсная характеристика фильтра, а на рис. 1.34, б, результат обработки фильтром синусоидального сигнала. Как видно из осциллограммы сигнала после обработки, фильтр является неустойчивым.



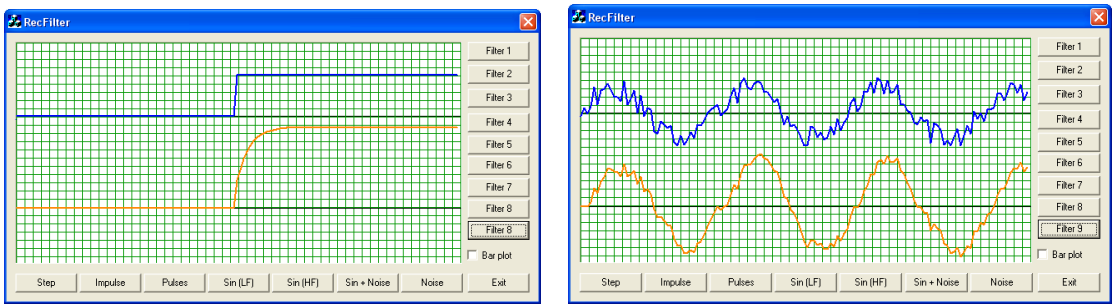
а *б*

Рис. 1.34.

Программная реализация цифрового фильтра, который образуется при значениях коэффициентов b_0=0,68, a_1= –0,37, a_2=0,29 представлена ниже.

```
void CRecFilterDlg::OnButton17()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=2;i<N/4;i++)
  {
          Y[i]=0.68*X[i]+0.37*Y[i-1]+0.29*Y[i-2];
  }
  DrawResult(RGB(255,135,0));
}
```

На рис. 1.35, а, представлена переходная характеристика фильтра, а на рис. 1.35, б, результат обработки фильтром зашумленного синусоидального сигнала. Из осциллограммы выходного сигнала видно, что величина случайной составляющей (высокочастотного шума) уменьшилась, при этом амплитуда низкочастотного синусоидального сигнала увеличилась. Реализованный фильтр представляет собой ФНЧ.



а *б*

Рис. 1.35.

Часто сначала цифровой фильтр проектируется как аналоговый, выбранный за прототип, а затем полученная с помощью преобразований Лапласа передаточная функция переводятся в Z-плоскость с использованием билинейного преобразования. Такой фильтр может быть реализован в цифровом виде в канонической форме, при которой используются и прямые и обратные связи (рис. 1.36).

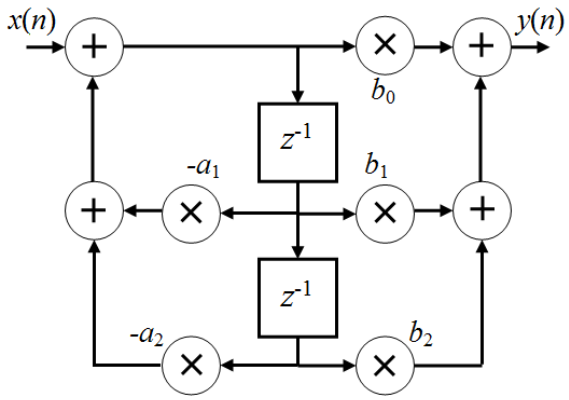


Рис. 1.36.

Разностное уравнение в этом случае имеет вид:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)\,.$$

Передаточная функция этого фильтра записывается в виде:

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

На рис. 1.37 представлен вид АЧХ резонансного фильтра 2 порядка, автоматизация процесса проектирования которого выполнена в программе Calculation of IIR filter (ciirf). Программа находится в свободном доступе в сети Internet.

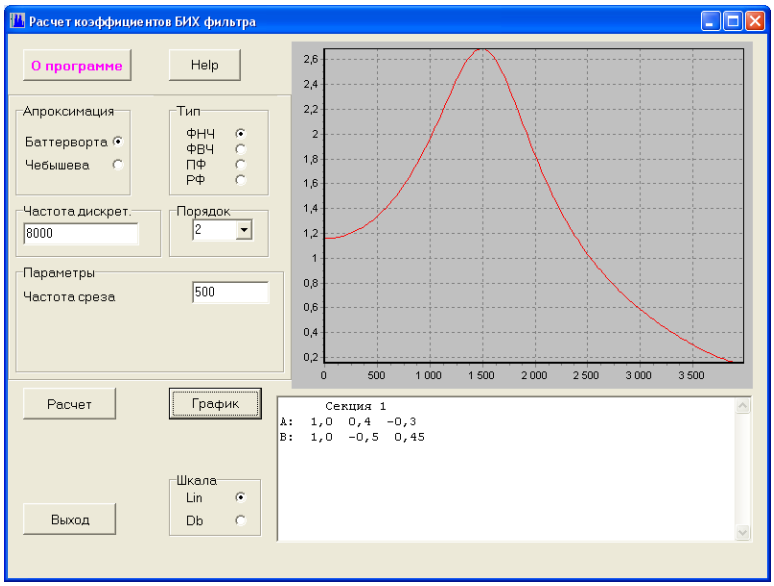


Рис. 1.37.

Программная реализация этого фильтра представлена ниже.

```
void CRecFilterDlg::OnButton4()
{
  memset(Y,0,sizeof(int)*N);

  for(int i=2;i<N/4;i++)
  {
          Y[i]=0.5*Y[i-1]-0.45*Y[i-2]+X[i]+0.4*X[i-1]-0.3*X[i-2];
  }
  DrawResult(RGB(255,0,255));
}
```

На рис. 1.38 приводится окно программы с осциллограммами входного и выходного сигналов фильтра. Видно, что амплитуда сигнала увеличилась, поскольку его частота попадает в область усиления.

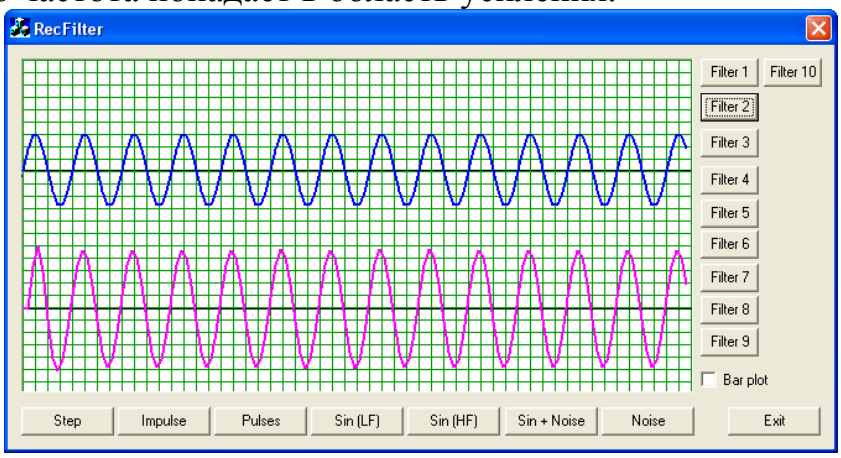


Рис. 1.38.

На рис. 1.39 представлен вид АЧХ ФНЧ 2 порядка.

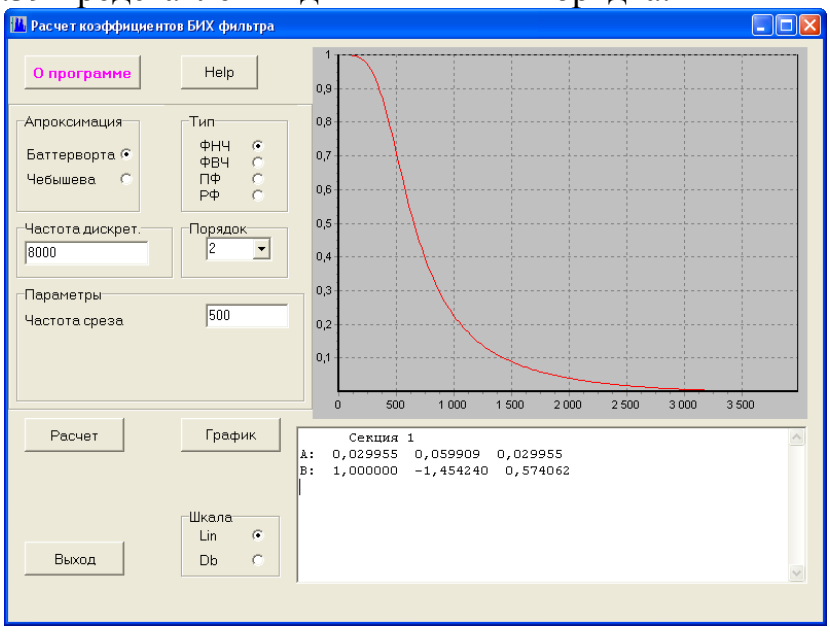


Рис. 1.39.

Программная реализация фильтра нижних частот имеет вид:

```
void CRecFilterDlg::OnButton8()
{
  memset(Y,0,sizeof(int)*N);
  for(int i=2;i<N/4;i++)
  {
  Y[i]=1.454240*Y[i-1]-0.574062*Y[i-2]+0.029955*X[i]+
0.059909*X[i-1]+0.029955*X[i-2];
  }
  DrawResult(RGB(160,170,35));
}
```

Результат обработки фильтром низкочастотного синусоидального сигнала с шумом представлен на рис. 1.40. По осциллограмме видно, что низкочастотный сигнал прошел на выход фильтра, а высокочастотный шум значительно ослабился.

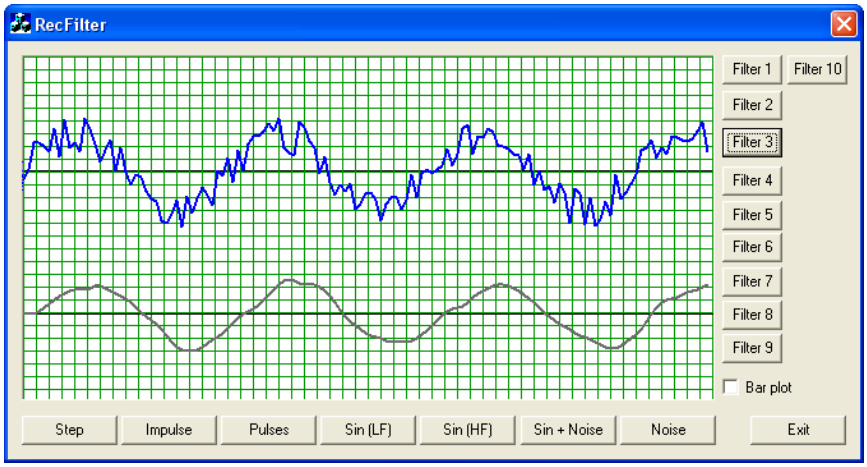
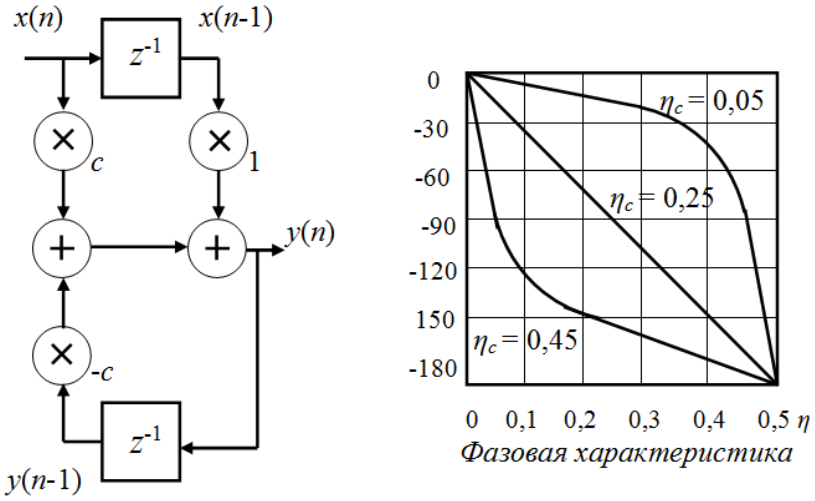


Рис. 1.40.

Все пропускающие фильтры (ФВП) являются базовой основой для построения параметрических фильтров. Такие фильтры наиболее широко используются в аудиотехнике, так как они позволяют осуществлять независимую перестройку частоты среза, добротности или полосы пропускания. ФВП 1 порядка может быть реализован на основе одного элемента задержки, но более широко применяется ФВП 1 порядка на основе двух элементов задержки. Схема такого фильтра приведена на рис. 1.41, а. В ней используется как прямая, так и обратная связи.



а *б*

Рис. 1.41.

Работа фильтра описывается разностным уравнением вида:

$$y(n) = c \cdot x(n) + x(n-1) - c \cdot y(n-1),$$

где

$$c = \frac{\tan(\pi\eta_c) - 1}{\tan(\pi\eta_c) + 1},$$

$\eta_c = f_c / f_s$ – относительная частота среза.

Передаточная функция фильтра ФВП 1 порядка в форме Z-преобразования имеет вид:

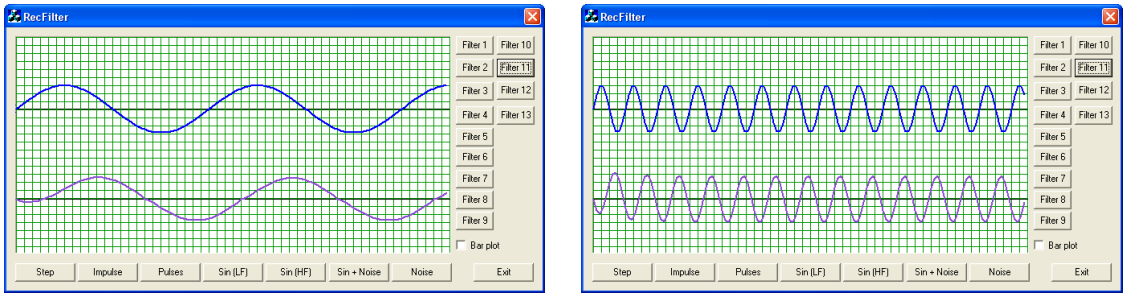
$$A_1(z) = \frac{z^{-1} + c}{1 + c \cdot z^{-1}}.$$

Фазовая характеристика фильтра очень сильно меняется в зависимости от выбора частоты среза. При $\eta_c = 0{,}25$ фазовый сдвиг с увеличением частоты линейно уменьшается от 0 до минус 180^0 на частоте Найквиста. При $\eta_c < 0{,}25$ ФЧХ в большей части частотного диапазона приближается к 0^0, а когда $\eta_c > 0{,}25$ она приближается к минус 180^0 (рис. 1.41, б). Частота среза фильтра плавно изменяется с помощью коэффициента фильтра *c*. Амплитудно-частотная характеристика этого фильтра не зависит от частоты и модуль коэффициента передачи равен единице во всем звуковом диапазоне частот.

```
void CRecFilterDlg::OnButton18()
{
 memset(Y,0,sizeof(int)*N);
 int fs=200;
 double Fc=(double)fs/8000;
 double c=(tan(3.14*Fc)-1)/(tan(3.14*Fc)+1);
 for(int i=1;i<N/4;i++)
 {
```

```
            Y[i]=c*X[i]+X[i-1]-c*Y[i-1];
    }
    DrawResult(RGB(140,75,210));
}
```

На рис. 1.42, а, б, представлены результаты обработки фильтром низкочастотного и высокочастотного синусоидальных сигналов. Видно, что амплитуды сигналов не изменились, однако изменилась фаза.



а *б*

Рис. 1.42.

ФВП 2 порядка строится на основе 4 элементов задержки, и в нем используются две прямые и две обратные связи (рис. 1.43, а). Работа фильтра описывается разностным уравнением:

$$y(n) = -c \cdot x(n) + d(1-c) \cdot x(n-1) + x(n-2) - d(1-c) \cdot y(n-1) + c \cdot y(n-2),$$

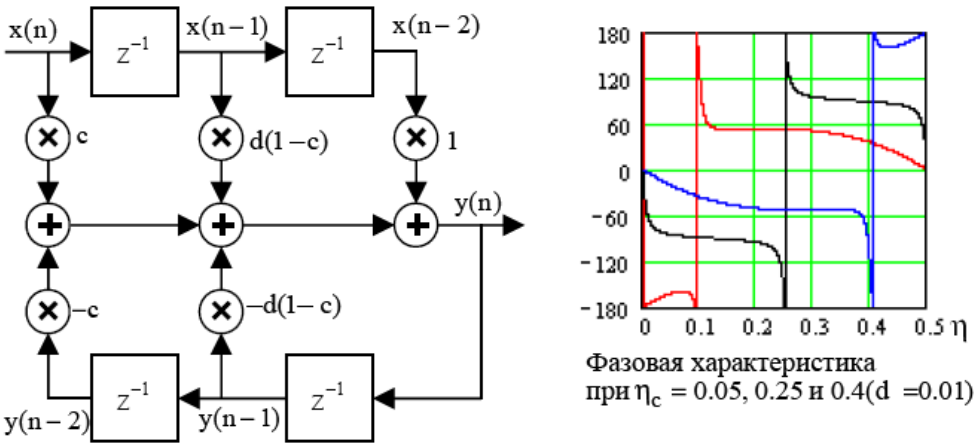
где

$$c = \frac{\tan(\pi\eta_b) - 1}{\tan(\pi\eta_b) + 1},$$

$\eta_b = f_b / f_s$ – относительная полоса (добротность) фильтра,

$$d = -\cos(2\pi \cdot \eta_c),$$

$\eta_c = f_c / f_s$ – относительная частота среза фильтра.



а *б*

Рис. 1.43.

Передаточная функция фильтра ФВП 2 порядка в форме Z-преобразования имеет вид:

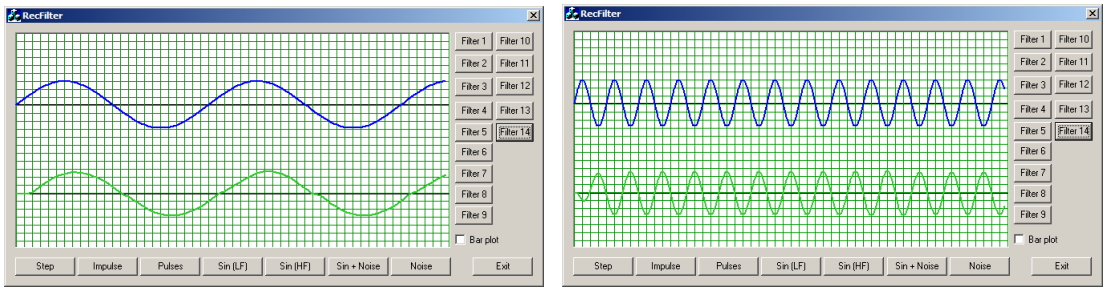
$$A_2(z) = \frac{-c + d(1-c)\cdot z^{-1} + z^{-2}}{1 + d(1-c)\cdot z^{-1} - c\cdot z^{-2}}.$$

Частотная характеристика этого фильтра не зависит от частоты и коэффициент передачи фильтра равен единице во всем звуковом диапазоне частот.

Программная реализация фильтра представлена ниже.

```
void CRecFilterDlg::OnButton21()
{
  memset(Y,0,sizeof(int)*N);
  int fc=800;
  int fs=8000;
  int fb=100;
  double Fc=(double)fc/fs;
  double Fb=(double)fb/fs;
  double c=(tan(3.14*Fc)-1)/(tan(3.14*Fc)+1);
  double d=-cos(6.28*Fc);
  for(int i=2;i<N/4;i++)
  {
          Y[i]=-c*X[i]+d*(1-c)*X[i-1]+X[i-2]-d*(1-c)*Y[i-1]+c*Y[i-2];
  }
  DrawResult(RGB(50,200,50));
}
```

На рис. 1.44, а, б, представлены низкочастотный синусоидальный сигнал и синусоидальный сигнал, частота которого равна частоте среза фильтра, и результат обработки фильтром этих сигналов. В рассмотренном примере частота дискретизации равна 8000 кГц, частота синусоидального сигнала составляет 800 Гц.



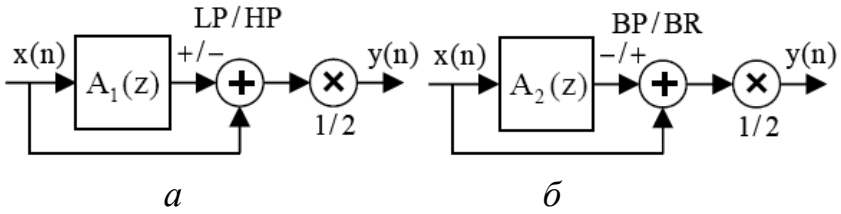
а *б*

Рис. 1.44.

В большей части этого диапазона фазовая характеристика меняется мало, но на частоте среза фазовый сдвиг сигналов скачком изменяется на 360^0.

Параметр *d* задает крутизну изменения фазы вблизи частоты среза, фактически этим параметром задается добротность фильтра.

Параметрические фильтры НЧ, ВЧ, полосовые пропускающие и полосовые режекторные на основе все пропускающих фильтров 1 и 2 строятся путем добавления всего одной дополнительной прямой связи (рис. 1.45).



а *б*

Рис. 1.45.

Для построения фильтров НЧ и ВЧ 1 порядка (рис. 1.45, а) используется фильтр ВП 1 порядка. Если в этой схеме выходной сигнал суммируется с входным – это ФНЧ, если вычитается – то это ФВЧ. Передаточная функция фильтра:

$$H(z) = \frac{1}{2}(1 \pm A_1(z)),$$

где $A_1(z)$ – передаточная функция ФВП 1 порядка. С помощью коэффициента *c* плавно меняется частота среза фильтров.

```
void CRecFilterDlg::OnButton19()
{
  memset(Y,0,sizeof(int)*N);
  int fc=200;
  int fs=8000;
  double Fc=(double)fc/fs;
  double c=(tan(3.14*Fc)-1)/(tan(3.14*Fc)+1);
  for(int i=1;i<N/4;i++)
  {
          Y[i]=(X[i]+c*X[i]+X[i-1]-c*Y[i-1])*0.5;
  }
  DrawResult(RGB(120,50,0));
}
```

На рис. 1.46 представлен результат обработки фильтром входного низкочастотного зашумленного сигнала. В полученном после обработки сигнале величина шума уменьшилась, т.е. фильтр представляет собой ФНЧ.

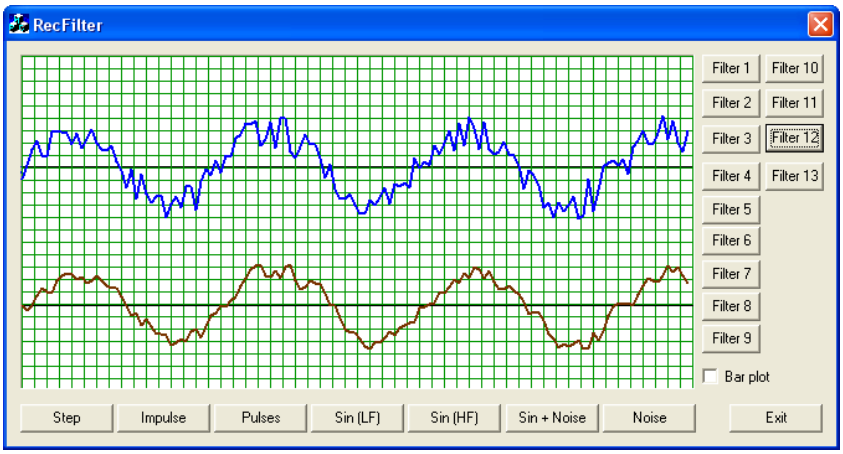


Рис. 1.46.

Программная реализация фильтра верхних частот представлена ниже. Эта реализация цифрового фильтра отличается от предыдущей знаком перед слагаемым $X[i]$ в разностном уравнении.

```
void CRecFilterDlg::OnButton20()
{
  memset(Y,0,sizeof(int)*N);
  int fc=2000;
  int fs=8000;
  double Fc=(double)fc/fs;
  double c=(tan(3.14*Fc)-1)/(tan(3.14*Fc)+1);
  for(int i=1;i<N/4;i++)
  {
          Y[i]=(-1*X[i]+c*X[i]+X[i-1]-c*Y[i-1])*0.5;
  }
  DrawResult(RGB(255,135,150));
}
```

На рис. 1.47 представлен результат обработки фильтром входного низкочастотного зашумленного сигнала. В полученном после обработки сигнале величина низкочастотной составляющей практически равна 0, а высокочастотный шум прошел на выход фильтра.

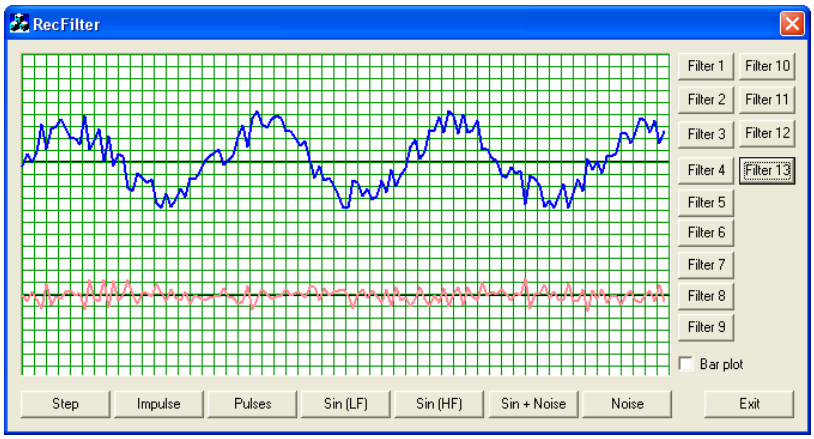


Рис. 1.47.

На рис. 1.48 приведены амплитудно-частотные характеристики параметрических ФНЧ и ФВЧ, построенных на основе все пропускающих фильтров 1 порядка с относительной частотой среза $\eta_c = 0{,}2$. Крутизна спада (подъема) АЧХ равна 6 дБ на октаву. Для перехода из режима ФНЧ в режим ФВЧ достаточно изменить знак в приведенной формуле передаточной функции фильтра с «+» на «–».

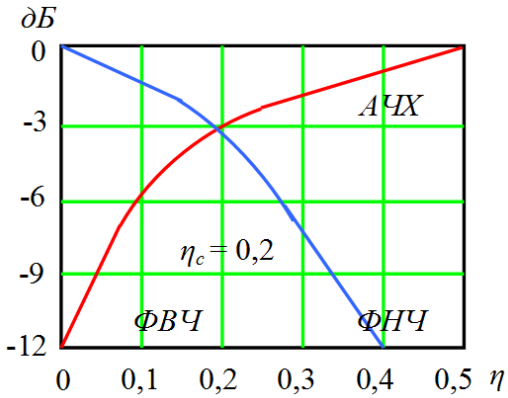


Рис. 1.48.

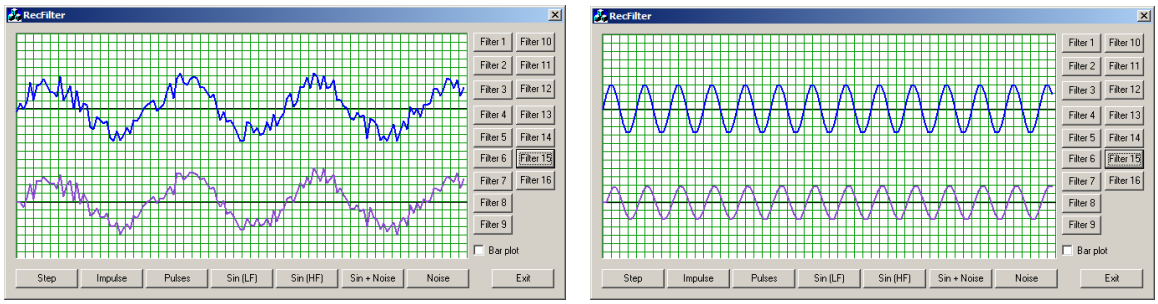
Для построения полосового пропускающего/заграждающего фильтра используется фильтр ВП 2 порядка (рис. 1.45, б). Если в этой схеме выходной сигнал суммируется с входным – это режекторный фильтр, если вычитается – то это пропускающий фильтр. Передаточная функция фильтра:

$$H(z) = \frac{1}{2}(1 \mp A_2(z)),$$

где $A_2(z)$ – передаточная функция ФВП 2 порядка. С помощью коэффициента c плавно меняется частота среза фильтров, а коэффициент d задает их добротность. Ниже представлена программная реализация режекторного фильтра.

```
void CRecFilterDlg::OnButton22()
{
  memset(Y,0,sizeof(int)*N);
  int fc=800;
  int fs=8000;
  int fb=200;
  double Fc=(double)fc/fs;
  double Fb=(double)fb/fs;
  double c=(tan(3.14*Fc)-1)/(tan(3.14*Fc)+1);
  double d=-cos(6.28*Fc);
  for(int i=2;i<N/4;i++)
  {
          Y[i]=(X[i]-c*X[i]+d*(1-c)*X[i-1]+X[i-2]-d*(1-c)*Y[i-1]+c*Y[i-2])/2;
  }
  DrawResult(RGB(140,75,210));
}
```

На рис. 1.49, а, представлен результат обработки фильтром низкочастотного гармонического сигнала в смеси с высокочастотным шумом. Спектр этого сигнала лежит за пределами полосы заграждения, и как видно из рисунка, сигнал проходит на выход фильтра без изменений. На рис. 1.49, б, представлен результат обработки гармонического сигнала, частота которого попадает в полосу заграждения фильтра. Как видно из рисунка сигнал ослабился.



а *б*

Рис. 1.49.

1.6. Программные средства для построения АЧХ цифровых фильтров

Под цифровым фильтром понимают дискретный фильтр, описываемый уравнением

$$y(n) = -\sum_{m=1}^{M-1} a_m y(n-m) + \sum_{k=0}^{K-1} b_k x(n-k),$$

где $x(n)$ и $y(n)$ – соответственно входная и выходная последовательности устройства, и реализованный программным путем с помощью микропроцессора или аппаратным путем в виде специализированного цифрового вычислительного устройства, состоящего из элементов памяти (регистров), сумматоров, умножителей и устройств управления.

Сигналы на входе и на выходе цифрового фильтра являются цифровыми, т.е. последовательностями чисел. Каждое из этих чисел представляется в виде двоичного кода определенной конечной разрядности. В цифровом фильтре выполняются операции пересылки, сложения и умножения кодов. При этом алгоритм функционирования реализуется неточно. Ошибки цифровой фильтрации обусловлены, во-первых, квантованием входных и выходных сигналов, во-вторых, квантованием коэффициентов фильтра и, в-третьих, конечной разрядностью операционных устройств, вследствие чего имеет место округление результатов арифметических операций. Таким образом, выбранная структура цифрового фильтра, разрядность входных и выходных сигналов, разрядность арифметических устройств влияют на точность работы устройства должны выбираться таким образом, чтобы результирующая ошибка цифрового фильтра не превышала допустимой величины.

Частотной характеристикой дискретного фильтра называется отношение:

$$H(e^{j\omega T}) = \frac{Y(e^{j\omega T})}{X(e^{j\omega T})}.$$

Частотная характеристика совпадает с передаточной функцией при

$$z = e^{j\omega T} = e^{j\tilde{\omega}}.$$

Поэтому для рекурсивного фильтра передаточная функция имеет вид:

$$H(e^{j\omega T}) = \frac{\sum_{k=0}^{K-1} a_k e^{-j\omega kT}}{1 + \sum_{m=0}^{M-1} b_m e^{-j\omega kT}},$$

а для нерекурсивного:

$$H(e^{j\omega T}) = \sum_{k=0}^{K-1} a_k e^{-j\omega kT}.$$

В общем случае $H(e^{j\omega T})$ – комплексная функция, которая может быть записана в виде:

$$H(e^{j\omega T}) = A(\omega) e^{j\varphi(\omega)},$$

где $A(\omega)$ – модуль частотной характеристики – амплитудно-частотная характеристика (АЧХ), $\varphi(\omega)$ – аргумент частотной характеристики – фазочастотная характеристика (ФЧХ).

Основные свойства частотных характеристик цифровых фильтров:

1. Частотные характеристики являются непрерывными функциями частоты.

2. При дискретизации данных по интервалам Δt функция $H(\omega)$ является периодической. Период функции $H(\omega)$ равен частоте дискретизации входных данных $F=1/\Delta t$. Первый низкочастотный период (по аргументу ω от $-\pi/\Delta t$ до $\pi/\Delta t$, по f от $-1/2\Delta t$ до $1/2\Delta t$) называется главным частотным диапазоном. Граничные частоты главного частотного диапазона соответствуют частоте Найквиста $\pm\omega_N$, $\omega_N = \pi/\Delta t$. Частота Найквиста определяет предельную частоту данных, которую способен обрабатывать фильтр.

3. Для фильтров с вещественными коэффициентами импульсной реакции $h(n\Delta t)$ функция АЧХ является четной, а функция ФЧХ – нечетной. С учетом этого частотные характеристики фильтров обычно задаются только на интервале положительных частот $0–\omega_N$ главного частотного диапазона. Значения функций на интервале отрицательных частот являются комплексно сопряженными со значениями на интервале положительных частот.

Как правило, при частотном анализе фильтров значение Δt интервала дискретизации принимают за 1, что соответственно определяет задание частотных характеристик на интервале $(0, \pi)$ по частоте ω или $(0, 1/2)$ по f. При использовании быстрых преобразований Фурье (БПФ) вычисления спектров осуществляются в одностороннем варианте положительных частот в

частотном интервале от 0 до 2π (от 0 до 1 Гц), где комплексно сопряженная часть спектра главного диапазона (от -π до 0) занимает интервал от π до 2π (для ускорения вычислений используется принцип периодичности дискретных спектров). При выполнении БПФ количество точек спектра равно количеству точек входной функции, а, следовательно, отсчет на частоте 2π, комплексно сопряженный с отсчетом на частоте 0, отсутствует. При нумерации точек входной функции от 0 до N он принадлежит точке N+1 – начальной точке следующего периода, при этом шаг по частоте равен 2π/(N+1).

Для построения АЧХ цифрового фильтра необходимо в передаточной функции выполнить подстановку $z = e^{j\omega T} = e^{j\tilde{\omega}}$, и вычислить модуль частотной характеристики $A(\omega)$ для диапазона частот от 0 до $f_s/2$, (f_s – частота дискретизации). Однако при такой подстановке появляется необходимость работы с комплексными числами. В языке С++ нет типа данных, позволяющих работать с комплексными числами. Поэтому стоит задача разработки своего собственного класса. Для удобства работы с комплексными числами в классе следует выполнить перегрузку операторов умножения, деления, сложения и вычитания, а также реализовать возможность задания начальных значений в алгебраической или показательной форме.

В среде разработки Visual Studio создаем проект на основе диалогового окна. После этого добавляем в проект заголовочный файл. Название файла указать complex.h. В этом файле выполняется объявление класса для работы с комплексными числами и описание прототипов функций:

```
#include "math.h"
class complex
{
private:
   double re, im;
public:
   complex() { re = 0; im = 0; }
   complex(double r, double i, bool polar)
  {
          if(polar==false)
          {
                  re = r;
                  im = i;
          }
          else
          {
                  re = r*cos(i);
                  im = r*sin(i);
          }
  }
      complex(const complex &ob){ re = ob.re; im = ob.im; };
      complex& operator = (complex);
```

```
        complex operator + (complex);
        complex operator - (complex);
        complex operator * (complex&);
        complex operator / (complex&);
        void ModPhase(double & mod, double & ph);
    };
```

Также в проект необходимо добавить файл с кодом реализации класса (complex.cpp):

```
#include "stdafx.h"
#include "complex.h"

complex complex::operator*(complex &com)
{
   double i, j;
   i = re * com.re - im * com.im;
   j = re * com.im + com.re * im;
   re = i;
   im = j;
   return *this;
}

complex complex::operator/(complex &com)
{
   double i, j, k;
   k = com.re * com.re +com.im * com.im;
   i = (re * com.re + im * com.im) / k;
   j = (com.re * im - re * com.im) / k;
   re = i;
   im = j;
   return *this;
}

complex& complex::operator =(complex com)
{
   this->re = com.re;
   this->im = com.im;
   return *this;
}

complex complex::operator+(complex com)
{
   this->re = this->re + com.re;
   this->im = this->im + com.im;
   return *this;
}

complex complex::operator-(complex com)
{
```

```
    this->re = this->re - com.re;
    this->im = this->im - com.im;
    return *this;
}

void complex::ModPhase(double & mod, double & ph)
{
  ph=atan(im/re);
  if((re<0) && (im>0)) ph=ph+3.14;
  else if((re<0) && (im<0)) ph=ph+3.14;
  mod=sqrt(re*re+im*im);
}
```

Проектируем интерфейс программы, примерный вид которого показан на рис. 1.50. Элементы текстового ввода Edit используются для задания коэффициентов фильтра.

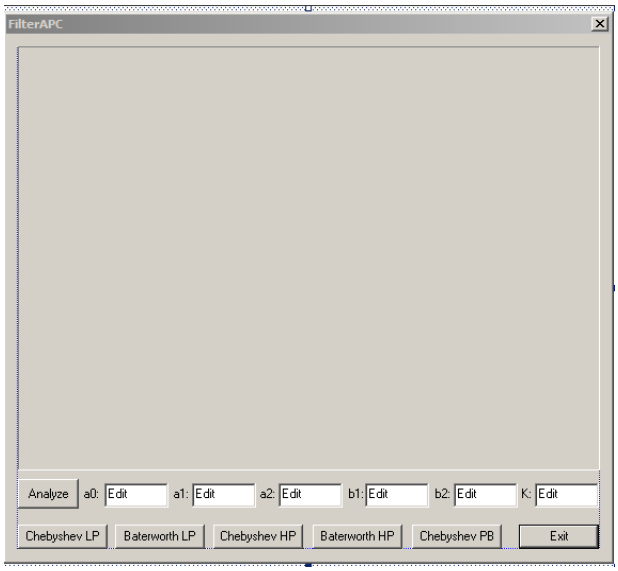


Рис. 1.50.

Программа выполняет построение АЧХ рекурсивного фильтра 2 порядка (биквадратное звено) с использованием значений коэффициентов, которые вводит пользователь с помощью этих элементов управления. В нижней части диалогового окна расположены кнопки, нажатие на которые приведет к заполнению полей заранее рассчитанными коэффициентами фильтров Чебышева и Баттерворта типов ФНЧ (LP) И ФВЧ (HP). Также имеется кнопка Chebyshev PB, нажатие на которую приведет к построению АЧХ полосового фильтра Чебышева.

С каждым из элементов текстового ввода Edit с помощью Class Wizard связывается переменная, как показано на рис. 1.51.

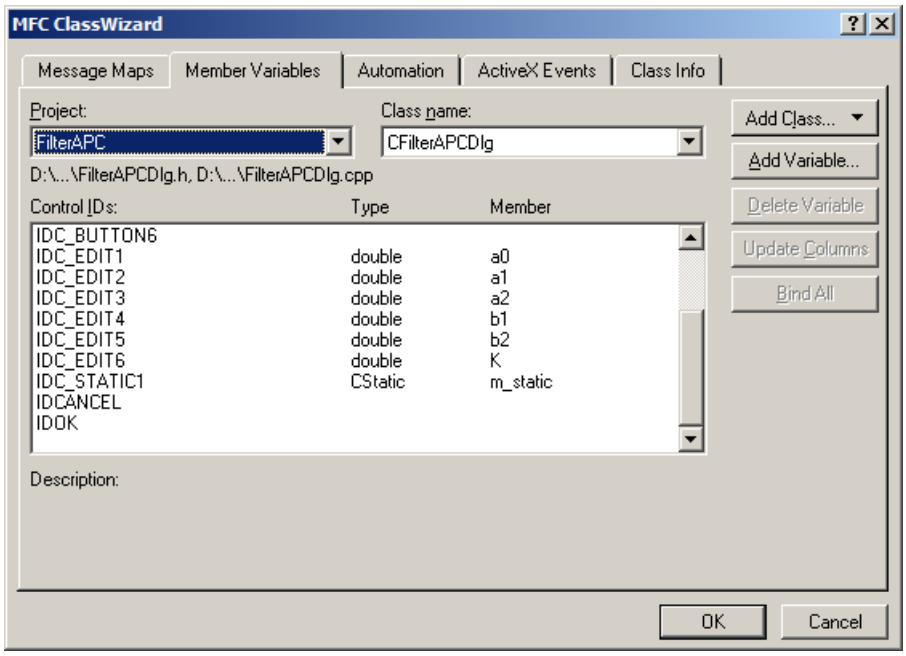


Рис. 1.51.

Реализация функции-обработчика нажатия на кнопку Analyze представлена ниже.

```
void CFilterAPCDlg::OnButton1()
{
  UpdateData(true);
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  int N=r.Width();
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CPen p, *op;
  p.CreatePen(PS_SOLID,1,RGB(0,160,0));
  op=dc.SelectObject(&p);
  double df=0;
  for(int j=0;j<10;j++)
  {
        dc.MoveTo(j*N/10,0);
        dc.LineTo(j*N/10,r.Height());
        if(j%2==0)
        {
              CString s;
              if(j!=0) s.Format("%.1f",df);
              else s="0";
              dc.TextOut(j*N/10+3,r.Height()-17,s);
              df=df+0.1;
        }
  }
  df=0;
```

```
  for(j=0;j<10;j++)
  {
        dc.MoveTo(0,j*r.Height()/10);
        dc.LineTo(N,j*r.Height()/10);
        CString s;
        if(j!=0) s.Format("%.1f",df);
        else s="0";
        dc.TextOut(3,r.Height()*(1-(double)j/10)-17,s);
        df=df+0.2;
  }
  p.DeleteObject();
  p.CreatePen(PS_SOLID,2,RGB(0,0,0));
  dc.SelectObject(&p);
        double M, P;
        double M0=0, P0=0;
        double b0=1;
        complex H, H1, H2;
        for(int i=1;i<N;i++)
        {
              complex cb0(b0,0, true);
              complex cb1(b1,-1.0*(double)3.14*i/N,true);
              complex cb2(b2,-1.0*(double)6.28*i/N,true);
              complex ca0(K*a0,0, true);
              complex ca1(K*a1,-1.0*(double)3.14*i/N,true);
              complex ca2(K*a2,-1.0*(double)6.28*i/N,true);
              H1=cb0+cb1+cb2;
              H2=ca0+ca1+ca2;
              H=H2/H1;
              H.ModPhase(M,P);
              dc.MoveTo(i,r.Height()-(int)M);
              if(i==1) dc.MoveTo(i,r.Height()-(int)M);
              else dc.LineTo(i-1,r.Height()-(int)M0);
              M0=M;
              P0=P;
        }
        dc.SelectObject(op);
}
```

Функция-обработчик нажатия на кнопку Chebyshev LP имеет вид:

```
void CFilterAPCDlg::OnButton2()
{
  a0=0.035690;
  a1=0.071379;
  a2=0.035690;
  b1=-1.501360;
  b2=0.644114;
  K=200;
  UpdateData(false);
}
```

Функция-обработчик нажатия на кнопку Buterworth LP имеет вид:

```
void CFilterAPCDlg::OnButton3()
{
  a0=0.29955;
  a1=0.059909;
  a2=0.029955;
  b1=-1.454240;
  b2=0.574062;
  K=61;
  UpdateData(false);
}
```

Функция-обработчик нажатия на кнопку Chebyshev HP имеет вид:

```
void CFilterAPCDlg::OnButton4()
{
  a0=0.349220;
  a1=-0.698440;
  a2=0.349229;
  b1=-0.089554;
  b2=0.307325;
  K=200;
  UpdateData(false);
}
```

Функция-обработчик нажатия на кнопку Buterworth HP имеет вид:

```
void CFilterAPCDlg::OnButton5()
{
  a0=0.757076;
  a1=-1.514150;
  a2=0.757076;
  b1=-1.454240;
  b2=0.574062;
  K=200;
  UpdateData(false);
}
```

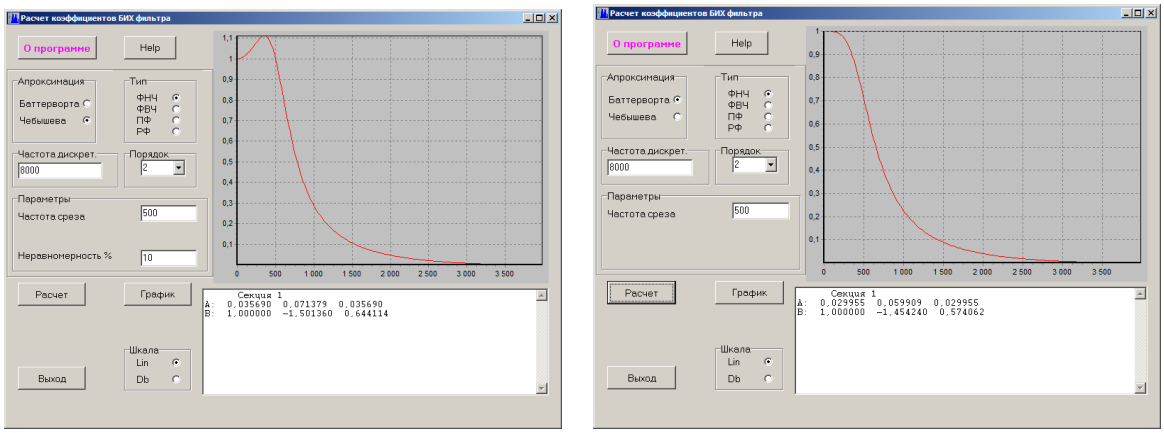
Функция-обработчик нажатия на кнопку Chebyshev PB имеет вид:

```
void CFilterAPCDlg::OnButton6()
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  int N=r.Width();
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CPen p, *op;
```

```
p.CreatePen(PS_SOLID,1,RGB(0,160,0));
op=dc.SelectObject(&p);
double df=0;
for(int j=0;j<10;j++)
{
	dc.MoveTo(j*N/10,0);
	dc.LineTo(j*N/10,r.Height());

	if(j%2==0)
	{
		CString s;
		if(j!=0) s.Format("%.1f",df);
		else s="0";
		dc.TextOut(j*N/10+3,r.Height()-17,s);
		df=df+0.1;
	}
}

df=0;
for(j=0;j<10;j++)
{
	dc.MoveTo(0,j*r.Height()/10);
	dc.LineTo(N,j*r.Height()/10);
	CString s;
	if(j!=0) s.Format("%.1f",df);
	else s="0";
	dc.TextOut(3,r.Height()*(1-(double)j/10)-17,s);
	df=df+0.2;
}
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(0,0,0));
dc.SelectObject(&p);
	double aa0=0.104697*200;
	double aa1=0;
	double aa2=-0.209394*200;
	double aa3=0;
	double aa4=0.104697*200;
	double bb0=1;
	double bb1=-2.354590;
	double bb2=2.582840;
	double bb3=-1.581880;
	double bb4=0.495204;
	double M, P;
	double M0=0, P0=0;
	double b0=1;
	complex H, H1, H2;
	for(int i=1;i<N;i++)
	{
		complex cb0(bb0,0, true);
		complex cb1(bb1,-1.0*(double)3.14*i/(N),true);
```

```
                complex cb2(bb2,-1.0*(double)2*3.14*i/(N),true);
                complex cb3(bb3,-1.0*(double)3*3.14*i/(N),true);
                complex cb4(bb4,-1.0*(double)4*3.14*i/(N),true);
                complex ca0(aa0,0, true);
                complex ca1(aa1,-1.0*(double)3.14*i/(N),true);
                complex ca2(aa2,-1.0*(double)2*3.14*i/(N),true);
                complex ca3(aa3,-1.0*(double)3*3.14*i/(N),true);
                complex ca4(aa4,-1.0*(double)4*3.14*i/(N),true);
                H1=cb0+cb1+cb2+cb3+cb4;
                H2=ca0+ca1+ca2+ca3+ca4;
                H=H2/H1;
                H.ModPhase(M,P);
                dc.MoveTo(i,r.Height()-(int)M);
                if(i==1) dc.MoveTo(i,r.Height()-(int)M);
                else dc.LineTo(i-1,r.Height()-(int)M0);
                M0=M;
                P0=P;
        }
}
```

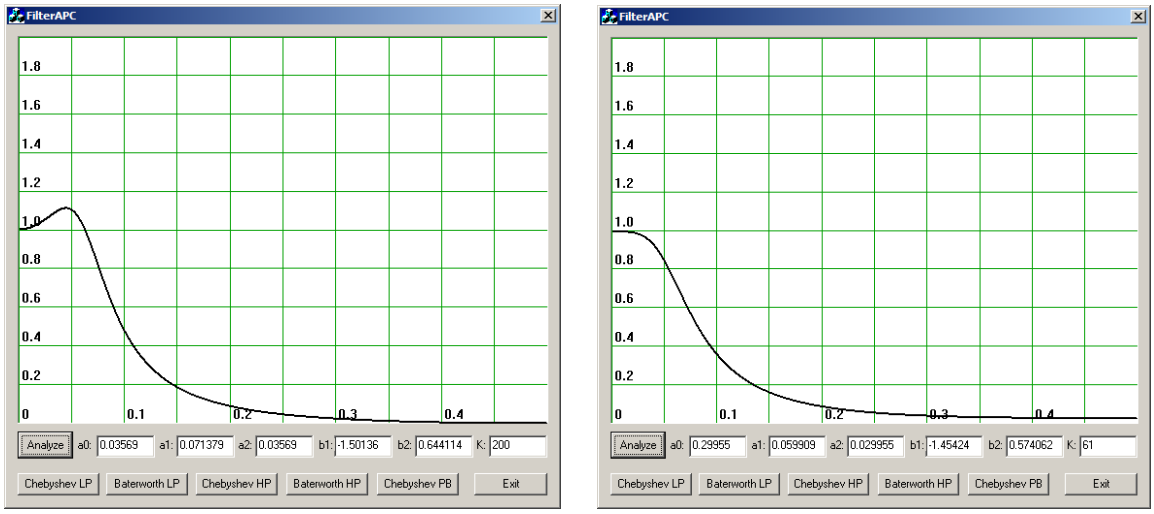
С помощью программы ciirf была выполнена автоматизация процесса проектирования фильтра, в результате которого получены значения коэффициентов, которые использованы в программе для демонстрации и проверки правильности построения АЧХ. На рис. 1.52, а, б, представлен вид окна программы с результатами расчета коэффициентов фильтров.



а *б*

Рис. 1.52.

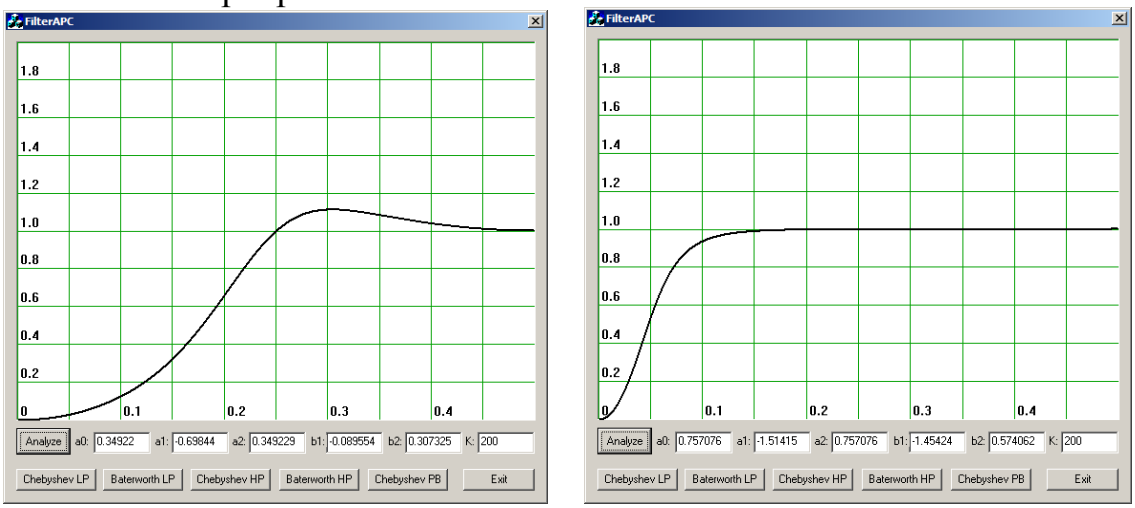
На рис. 1.53, а, б, представлены результаты построения АЧХ ФНЧ Чебышева и Баттерворта.



а *б*

Рис. 1.53.

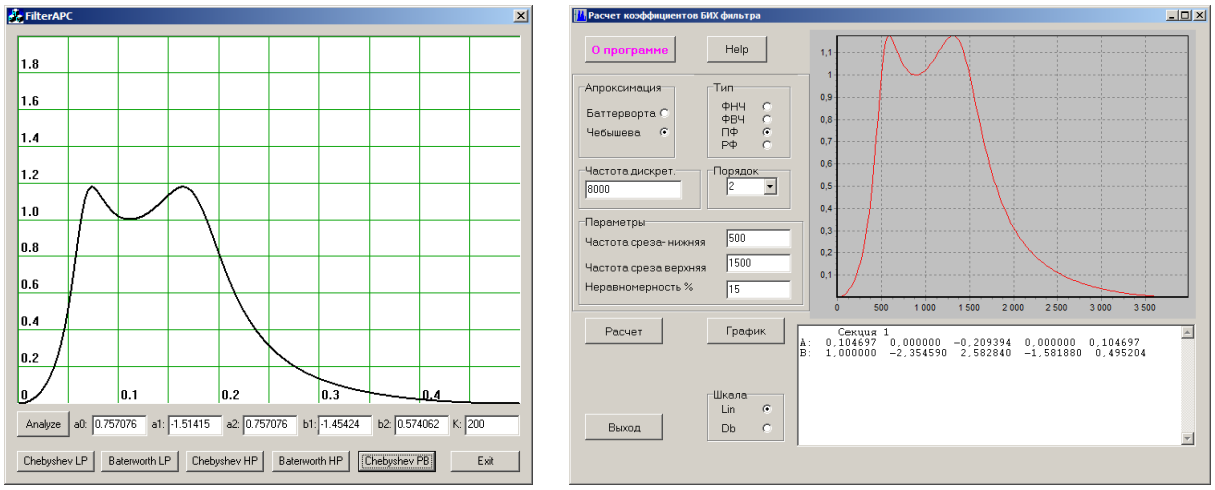
На рис. 1.54, а, б, представлены результаты построения АЧХ ФВЧ Чебышева и Баттерворта.



а *б*

Рис. 1.54.

На рис. 1.55, а, представлены результаты построения АЧХ полосового фильтра Чебышева, а на рис. 1.55, б, окно программы ciirf с результатами расчета коэффициентов этого фильтра.



а *б*

Рис. 1.55.

Анализ АЧХ показывает их полное соответствие, что позволяет сделать вывод о корректности работы реализованных в программе алгоритмов построения АЧХ цифровых фильтров.

1.7. Цифровые фильтры для обработки изображений

Существует множество сигналов, которые по своей природе являются двумерными и многомерными. Это, прежде всего различные изображения, получаемые при аэрофотосъемках и метеосъемках, рентгеновские снимки и сейсмограммы, фототелеграфные и телевизионные изображения, радиолокационные изображения, астрофизические и океанографические карты. Обработкой таких сигналов и изображений занимается направление в науке, называемое цифровой обработкой изображений и многомерных сигналов.

Изображения можно разделить на статические:

– фотография;
– изображение, снятое сканером;
– отдельные кадры телевизионного изображения;

и динамические:

– кинофильмы;
– последовательности кадров ТВ изображения.

Статические изображения имеют две степени свободы (n_1, n_2), а динамические – три степени свободы: n_1, n_2 – определяют пространственные координаты, а n_3 – время.

В связи с этим различают:

– внутрикадровые алгоритмы обработки и анализа изображений (используются для статических изображений);
– межкадровые алгоритмы обработки и анализа (используются для динамических изображений).

Внутрикадровые алгоритмы относят к пространственной обработке изображений, а межкадровые – к временной обработке.

Наиболее ярким примером межкадровой обработки является межкадровая разность, применяемая для обнаружения подвижных объектов.

Отличия цифровой обработки изображений (ЦОИ) от цифровой обработки сигналов (ЦОС)*:*

1. При решении двумерных задач используются гораздо большие объемы данных, чем при решении одномерных задач (требуется высокое быстродействие при обработке сигналов в реальном времени). Так, например, в черно-белых телевизорах в одном кадре содержится порядка 500 тысяч элементов, поэтому, если для передачи 1 элемента использовать 1 байт (256 уровней), то объем переданной информации за одну секунду (25 кадров) составит 500000×25=12,5 Мбайт. Такое количество информации необходимо обрабатывать за доли секунды. Отсюда возникает сложность аппаратной и программной реализации устройств для обработки изображений, поэтому процессы обработки распараллеливают и обрабатывают изображения с помощью систолических процессоров.

2. Математическое описание многомерных систем недостаточно разработано из-за отсутствия адекватных моделей изображений. Поэтому в обработке изображений большую роль играют эвристические алгоритмы, которые зачастую дают лучшие результаты, чем оптимальные линейные алгоритмы, хорошо работающие при нормальном законе распределения помех и шумов.

3. Многомерные сигналы обладают большим числом степеней свободы, что дает большую гибкость при выборе алгоритмов обработки, а именно для буферизированных изображений, но в результате этого реализация и программирование более трудоемко.

Направления ЦОИ:

– улучшение визуального качества изображений (фильтрация помех и шумов);

– восстановление «смазанных» и расфокусированных изображений;

– спектральный анализ;

– оценка параметров изображений (координат, площадей, периметров), выделение признаков, классификация и распознавание изображений.

Схематически алгоритм двумерной линейной фильтрации приведен на рис. 1.56.

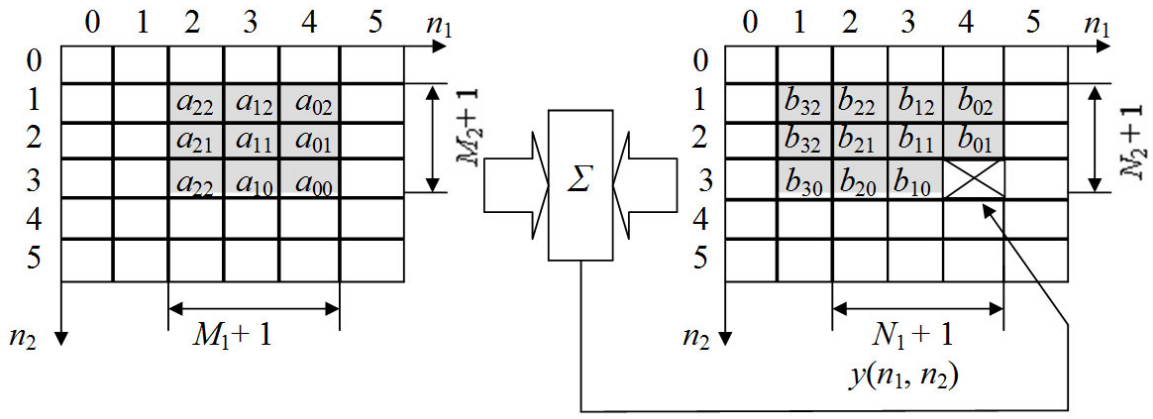


Рис. 1.56.

Здесь $(M_1+1)(M_2+1)$ – размер опорной области по входным данным – $x(n_1, n_2)$, а $(N_1+1)(N_2+1)$ – размер опорной области по выходным данным – $y(n_1, n_2)$.

Разностное уравнение:

$$y(n_1,n_2)=\sum_{i_1=0}^{M_1}\sum_{i_2=0}^{M_2}a_{i_1,i_2}x(n_1-i_1,n_2-i_2)+\sum_{j_1=0}^{N_1}\sum_{j_2=0}^{N_2}b_{j_1,j_2}y(n_1-j_1,n_2-j_2),$$

где $n_1\ge 0, n_2\ge 0$, $(j_1,j_2)\neq 0$.

Системная функция

$$H(z_1,z_2)=\frac{Y(z_1,z_2)}{X(z_1,z_2)},$$

определяется как отношение z-образов входной и выходной последовательностей; здесь z_1^{-1} – задержка на 1 шаг по строке (координата n_1); z_2^{-1} – задержка на 1 шаг по кадру или столбцу (координата n_2).

Если заданы коэффициенты a_{i_1,i_2} и b_{j_1,j_2} в разностном уравнении (РУ), то системная функция (СФ) примет вид:

$$H(z_1,z_2)=\frac{\sum_{i_1=0}^{M_1}\sum_{i_2=0}^{M_2}a_{i_1,i_2}z_1^{-i_1}z_2^{-i_2}}{1-\sum_{j_1=0}^{N_1}\sum_{j_2=0}^{N_2}b_{j_1,j_2}z_1^{-j_1}z_2^{-j_2}}.$$

Отсчет $(j_1, j_2) = 0$ исключается.

Очень широкое применение в обработке изображений находят нерекурсивные фильтры (НРФ):

$$y(n_1,n_2)=\sum_{i_1=0}^{M_1}\sum_{i_2=0}^{M_2}a_{i_1,i_2}x(n_1-i_1,n_2-i_2),$$

$$H(z_1,z_2)=\sum_{i_1=0}^{M_1}\sum_{i_2=0}^{M_2}a_{i_1,i_2}z_1^{-i_1}z_2^{-i_2}.$$

Выходной сигнал двумерного нерекурсивного фильтра представляется в виде линейной комбинации текущего и предыдущих отсчетов входного сигнала. Структурная схема подобного фильтра показана на рис. 1.57.

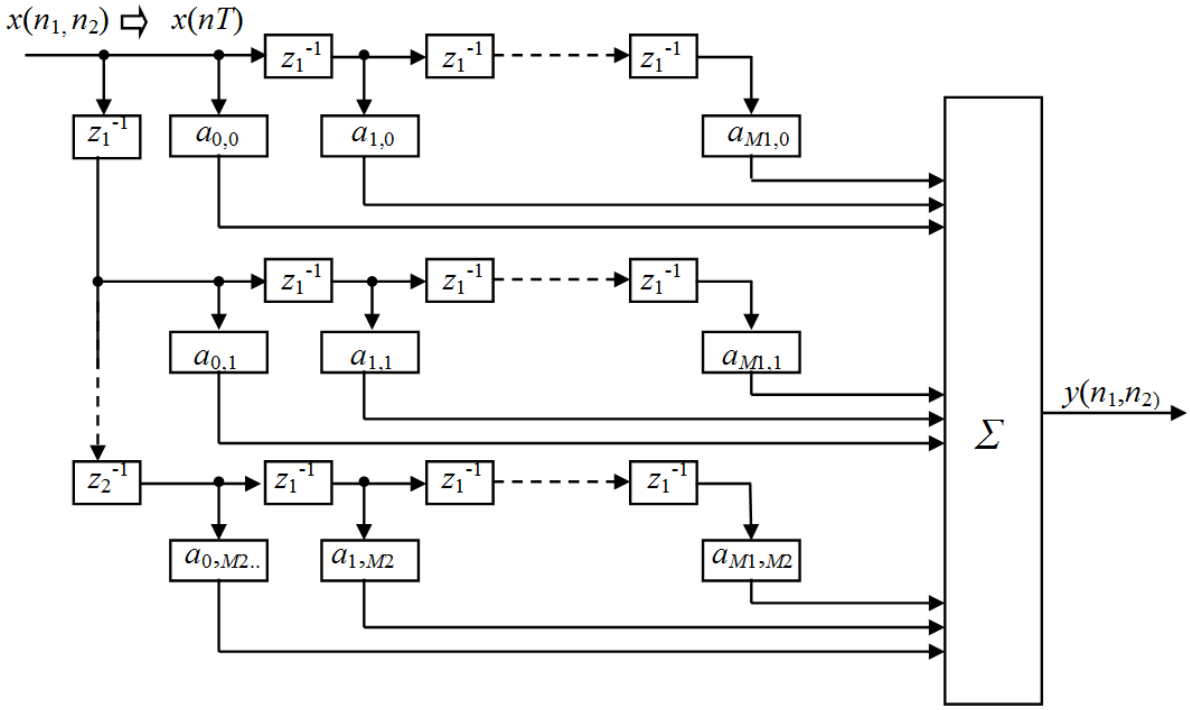


Рис. 1.57.

Коэффициенты фильтра задаются в виде двумерных масок (матриц).

Ядро свёртки – это матрица коэффициентов, которая умножается на значение пикселей изображения для получения требуемого результата. Обычно ядро свертки является квадратной матрицей $n \times n$, где n – нечетное (но может быть и четным). Ниже представлен пример применение матрицы свёртки:

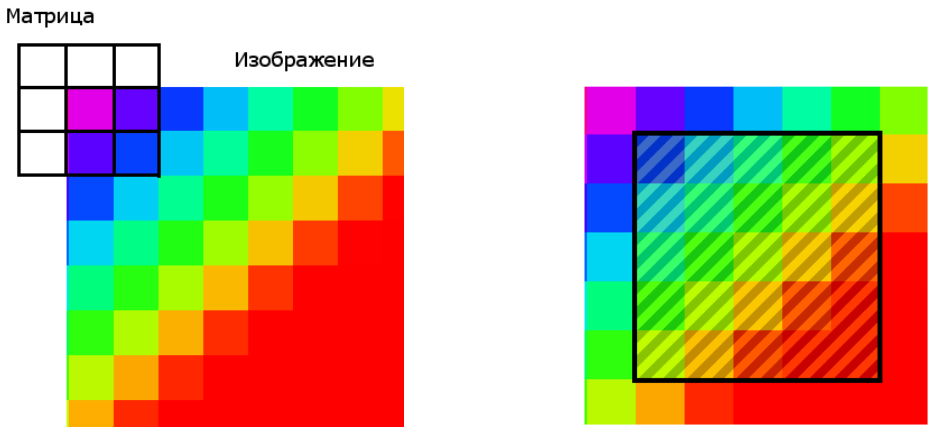
$$\frac{1}{15} \cdot \begin{bmatrix} 12 & 14 & 41 \\ 43 & 84 & 24 \\ 2 & 1 & 43 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix} =$$

$$= \frac{1}{15} \cdot (12 \cdot 1 + 14 \cdot 2 + 41 \cdot 1 + 43 \cdot 2 + 84 \cdot 3 + 24 \cdot 2 + 2 \cdot 1 + 1 \cdot 2 + 43 \cdot 1) = 34{,}26 \approx 34 .$$

Во время вычисления нового значения выбранного пикселя ядро свертки как бы «прикладывается» своим центром (именно тут важна нечетность размера матрицы) к данному пикселю. Окружающие пиксели также «накрываются» ядром. Далее вычисляется сумма, где слагаемыми являются произведения значений яркостей (составляющих цвета) пикселей на значения элементов ядра, накрывшей данный пиксель. Полученная сумма делится на значение суммы всех элементов ядра свертки. Полученное значение как раз и является новым значением выбранного пикселя. В примере использован

коэффициент нормирования, который в данном случае равен 15. Его использование необходимо для того, чтобы средняя яркость изображения оставалась неизменной, а также небыло искажений цвета из-за переполнения разрядной сетки. Таким образом, операция свертки позволяет вычислить новое значение выбранного пикселя, учитывающее значения окружающих его пикселей. Если применить свертку к каждому пикселю изображения, то в результате получится некий эффект, зависящий от выбранного ядра свертки.

Для всех матричных фильтров актуальна проблема с граничными условиями. У верхнего левого пикселя (рис. 1.58, а) не существует «соседа» слева и сверху от него, следовательно, не на что умножать коэффициенты матрицы.



а *б*

Рис. 1.58.

Существует 2 решения этой проблемы:

1. Применение фильтра, только к «окну» изображения (рис. 1.58, б). Это не лучший способ, так как фильтр не применяется ко всему изображению и качество при этом довольно сильно снижается, если размер фильтра большой.

2. Второй метод – дополнение – требует создания промежуточного изображения. Идея в том, чтобы создавать временное изображение с размерами, большими чем исходное. В центр изображения копируется входная картинка, а края заполняются крайними пикселями изображения (рис. 1.59). Размытие применяется к промежуточному буферу, а потом из него извлекается результат. Данный метод не имеет недостатков в качестве, но необходимо производить лишние вычисления.

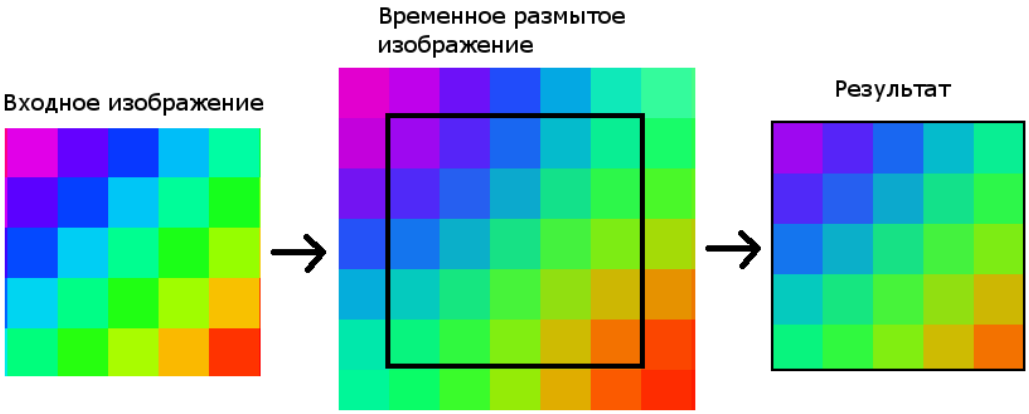


Рис. 1.59.

Ядро свертки – это коэффициенты фильтра с конечным временем отклика (КИХ-фильтр, FIR), применяемого к изображению. Коэффициенты свертки получаются путем применения обратного преобразования Фурье к передаточной функции фильтра. Для изображений, передаточная функция фильтра – это матрица, в которой задаются коэффициенты усиления для разных частот сигналов, составляющих изображение. При этом высокие частоты соответствуют границам объектов, а низкие частоты – фону и плавным переходам. Например, сглаживание – это фильтр, который убирает высокие частоты из изображения. Выделение границ – это фильтр, который убирает низкие частоты из изображения. Повышение резкости – это фильтр, который усиливает высокие частоты, а низкие частоты оставляет без изменений.

В редакторе ресурсов среды разработки проектируем внешний вид пользовательского интерфейса приложения как показано на рис. 5. В качестве «контейнера» для изображений используется элемент Picture. В редакторе ресурсов необходимо загрузить в проект 6 фотографий размером 300×200 пикселей, которые будут использоваться для демонстрации работы цифровых фильтров. Слева расположено исходное изображение, а справа будет отображаться изображение после обработки. Между элементами Picture расположена кнопка с изображением стрелки ⇓, нажатие на которую скопирует обработанное изображение на место исходного. Это позволит применить к изображению последовательно несколько цифровых фильтров.

Под исходным изображением расположены 6 кнопок (Image 1 – Image 6). Нажатие на одну из кнопок приведет к смене обрабатываемого изображения. Это позволит более наглядно показать работу каждого из фильтров. Каждое из анализируемых изображений имеет свои особенности – отсутствие резких перепадов, плавные или резкие линии, мало или много деталей, простые или сложные элементы изображения, наличие или отсутствие градиентов, цветное или полутоновое и др. Элемент Check Box позволит выбрать режим отображения – в прямых или инвертированных цветах.

В нижней левой части диалога расположены кнопки. Нажатие на любую из кнопок приведет к заполнению полей Edit заранее установленными коэффициентами фильтра. Это сделано для удобства работы с программой. В полях Edit пользователь может ввести любые значения коэффициентов. Нажатие на кнопку Filter приведет к запуску алгоритма цифровой обработки изображения и результат обработки отобразится в окне справа от исходного изображения. Поле с подписью 1/N – это масштабный (нормировочный) коэффициент. Обычно выбирается равным сумме всех коэффициентов матрицы ядра фильтра.

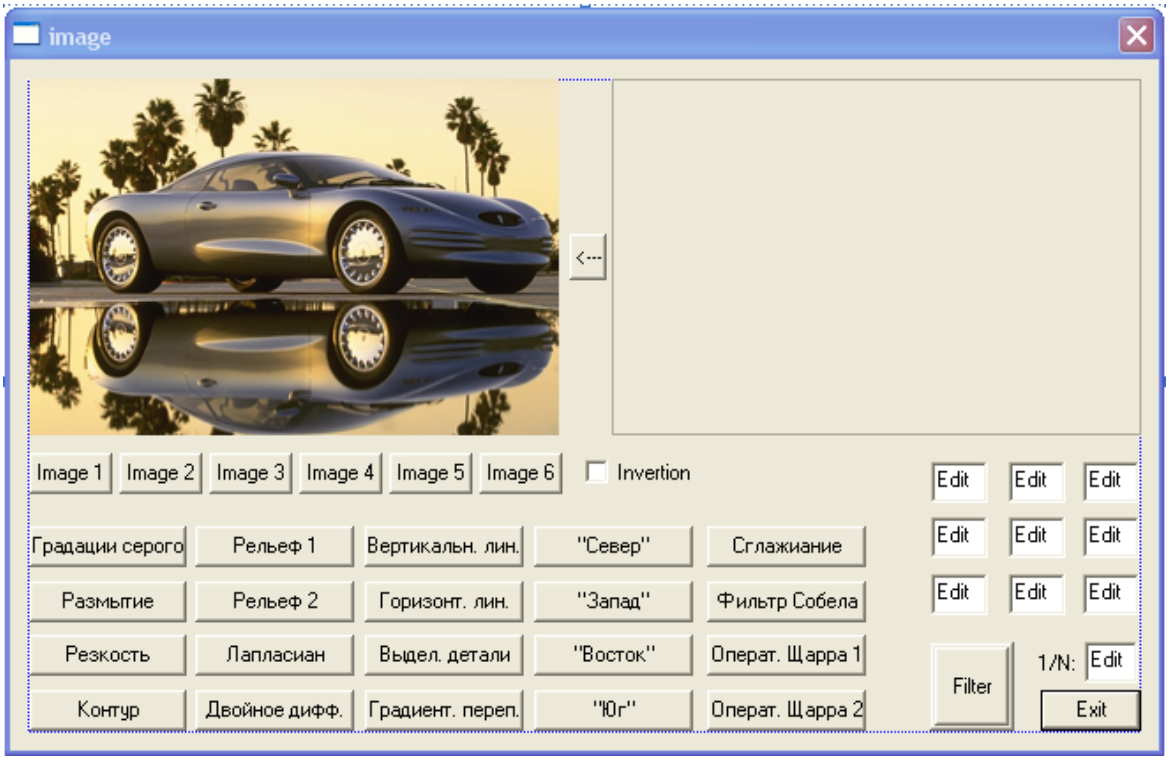


Рис. 1.60.

На рис. 1.61 представлено окно Class Wizard, с помощью которого с элементами управления связаны переменные.

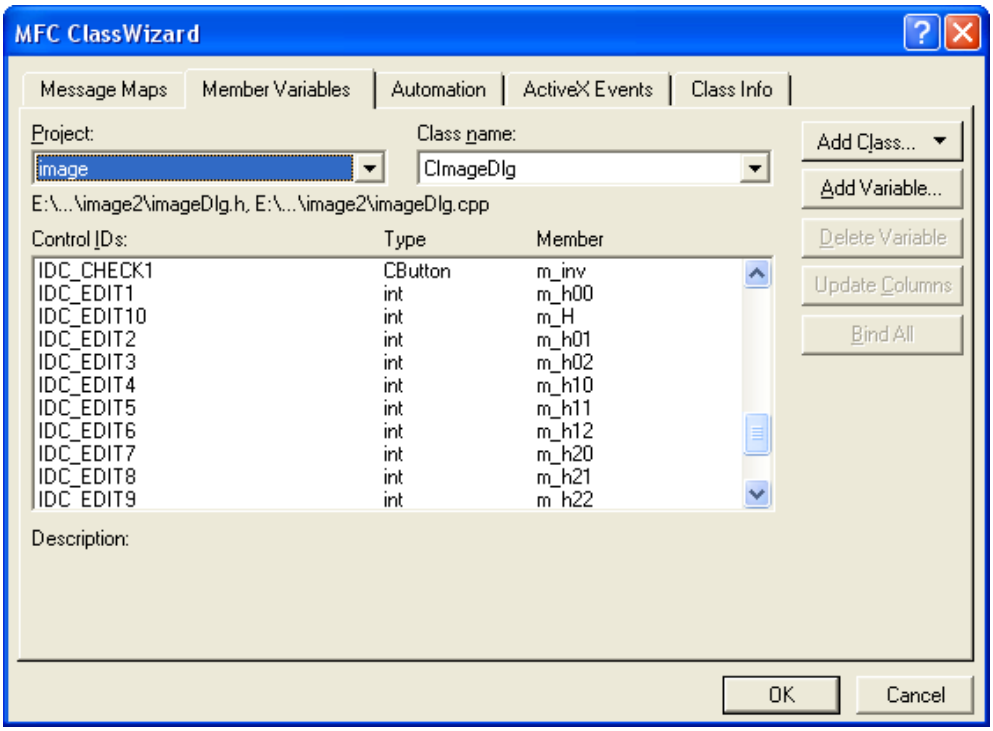


Рис. 1.61.

С элементами управления Picture связаны следующие переменные:
– для исходного изображения: CStatic m_p;
– для обработанного изображения: CStatic m_stat;

В класс диалога необходимо добавить переменную – двумерный массив для хранения коэффициентов фильтра:

```
int h[3][3];
```

В функции OnInitDialog() выполняется инициализация переменных:

```
m_H=1; // масштабный коэффициент равен 1
// коэффициенты фильтра кроме центрального равны 0
m_h00=m_h01=m_h02=m_h10=m_h12=m_h20=m_h21=m_h21=0;        m_h11=1;
for(int y=0;y<=2;y++)
{
  for(int x=0;x<=2;x++) h[y][x]=0;// обнуление матрицы коэффициентов
}
h[1][1]=1;        // центральный элемент матрицы коэффициентов равен 1
UpdateData(false); // обновление состояния экранных элементов управления
```

Функция-обработчик нажатия на кнопку Image 1 имеет вид:

```
void CImageDlg::OnButton9()
{
  HBITMAP bmp;
```

```
bmp=::LoadBitmap(AfxGetInstanceHandle(),
MAKEINTRESOURCE(IDB_BITMAP1));
  m_p.SetBitmap(bmp);
}
```

Функция-обработчик нажатия на кнопку Image 2 имеет вид:

```
void CImageDlg::OnButton9()
{
  HBITMAP bmp;
bmp=::LoadBitmap(AfxGetInstanceHandle(),
MAKEINTRESOURCE(IDB_BITMAP2));
  m_p.SetBitmap(bmp);
}
```

Аналогичные обработчики необходимо добавить для кнопок Image 3 – Image 6. Отличаются они только номером загружаемой картинки: IDB_BITMAP3 – IDB_BITMAP6.

Функция-обработчик нажатия на кнопку ⇓, в которой выполняется копирование обработанного изображения на место исходного, представлена ниже.

```
void CImageDlg::OnButton15()
{
  CClientDC dc(&m_p);
  CRect r;
  m_p.GetClientRect(&r);
  CClientDC cdc(&m_stat);
  CBitmap bmp;
  bmp.CreateCompatibleBitmap(&dc,r.Width(),r.Height());
  dc.SelectObject(&bmp);
  dc.BitBlt(0,0,r.Width(),r.Height(),&cdc,0,0,SRCCOPY);
}
```

Функция-обработчик нажатия на кнопку «Градации серого», выполняющая преобразование цветного изображения в полутоновое, имеет вид:

```
void CImageDlg::OnButton1()
{
  CClientDC dc(&m_p);
  CClientDC dc1(&m_stat);
  CRect r;
  m_p.GetClientRect(&r);
          for(int i=0;i<r.Width();i++)
   {
    for(int j=0;j<r.Height();j++)
```

```
    {
            int r=0;
            int g=0;
            int b=0;
            int avg=0;
            COLORREF c=dc.GetPixel(i,j);
            r=GetRValue(c);
            g=GetGValue(c);
            b=GetBValue(c);
            avg=(r+g+b)/3;
            dc1.SetPixel(i,j,RGB(avg,avg,avg));
    }
  }
}
```

Обработчик нажатия на кнопку «Размытие», в котором выполняется заполнение матрицы коэффициентов, представлен ниже.

```
void CImageDlg::OnButton2()
{
  int h1[3][3]={{1,1,1},{1,1,1},{1,1,1}}; m_H=9;
  int index=0;
  CString s;
  for(int y=0;y<=2;y++)
  {
          for(int x=0;x<=2;x++)
          {
                  h[y][x]=h1[y][x];
                  s.Format("%d",h1[y][x]);
                  GetDlgItem(IDC_EDIT1+index)->SetWindowText(s);
                  index++;
          }
  }
  s.Format("%d",m_H);
  GetDlgItem(IDC_EDIT10)->SetWindowText(s);
}
```

Обработчики нажатия на остальные кнопки аналогичны приведенному, за исключением значений элементов матрицы. Ниже приводятся фрагменты кода со значениями коэффициентов каждого из используемых в программе фильтров:

```
Резкость:       int h1[3][3]={{-1,-2,-1},{-2,22,-2},{-1,-2,-1}}; m_H=10;
Контур:         int h1[3][3]={{0,-1,0},{-1,4,-1},{0,-1,0}}; m_H=10;
Рельеф 1:       int h1[3][3]={{0,1,0},{1,0,-1},{0,-1,0}}; m_H=2;
Рельеф 2:       int h1[3][3]={{0,-1,0},{-1,0,1},{0,1,0}}; m_H=2;
Лапласиан:      int h1[3][3]={{-1,-1,-1},{-1,8,-1},{-1,-1,-1}}; m_H=40;
```

Двойное дифф.: int h1[3][3]={{1,-2,1},{-2,4,-2},{1,-2,1}}; m_H=10;
Вертикальн. лин.: int h1[3][3]={{-1,2,-1},{-1,2,-1},{-1,2,-1}}; m_H=15;
Горизонт. лин.: int h1[3][3]={{-1,-1,-1},{2,2,2},{-1,-1,-1}}; m_H=15;
Выдел. детали: int h1[3][3]={{1,2,1},{2,4,2},{1,2,1}}; m_H=3;
Градиент. переп.: int h1[3][3]={{1,1,-1},{1,-2,-1},{1,1,-1}}; m_H=20;
«Север»: int h1[3][3]={{1,1,1},{1,-2,1},{-1,-1,-1}}; m_H=20;
«Запад»: int h1[3][3]={{1,1,-1},{1,-2,-1},{1,1,-1}}; m_H=20;
«Восток»: int h1[3][3]={{-1,1,1},{-1,-2,1},{-1,1,1}}; m_H=20;
«Юг»: int h1[3][3]={{-1,-1,-1},{1,-2,1},{1,1,1}}; m_H=20;
Сглаживание: int h1[3][3]={{2,1,2},{1,2,1},{2,1,2}}; m_H=14;
Фильтр Собела: int h1[3][3]={{-1,-2,-1},{0,0,0},{1,2,1}}; m_H=20;
Операт. Щарра 1: int h1[3][3]={{3,0,-3},{10,0,-10},{3,0,-3}}; m_H=10;
Операт. Щарра 2: int h1[3][3]={{3,10,3},{0,0,0},{-3,-10,-3}}; m_H=10;

Обработчик нажатия на кнопку Filter:

```
void CImageDlg::OnButton16()
{
  UpdateData(true);
  CClientDC dc(&m_p);
  CClientDC dc1(&m_stat);
  int N=3;
  int D=1;
  CRect r;
  m_p.GetClientRect(&r);

  h[0][0]=m_h00;
  h[0][1]=m_h01;
  h[0][2]=m_h02;
  h[1][0]=m_h10;
  h[1][1]=m_h11;
  h[1][2]=m_h12;
  h[2][0]=m_h20;
  h[2][1]=m_h21;
  h[2][2]=m_h22;
  for(int i=D;i<r.Width()-D;i++)
 {
   for(int j=D;j<r.Height()-D;j++)
   {
    int r=0;
    int g=0;
    int b=0;
    int avg=0;
            for(int x=-1;x<2;x++)
            {
                    for(int y=-1;y<2;y++)
                    {
```

```
                        COLORREF c=dc.GetPixel(i+x,j+y);
                        r=r+h[y+1][x+1]*GetRValue(c);
                        g=g+h[y+1][x+1]*GetGValue(c);
                        b=b+h[y+1][x+1]*GetBValue(c);
                    }
                }
  r=r/m_H;
  g=g/m_H;
  b=b/m_H;
  if(m_inv.GetCheck())dc1.SetPixel(i,j,RGB(255-r,255-g,255-b));
  else dc1.SetPixel(i,j,RGB(r,g,b));
  }
}
}
```

На рис. 1.62 представлены типы наиболее используемых масок:
а) скользящее среднее;
б) лапласиан;
в) двойное дифференцирование;
г) оператор выделения вертикальных линий;
д) оператор выделения малоразмерных деталей из шумов;
е) градиентный оператор выделения перепада.

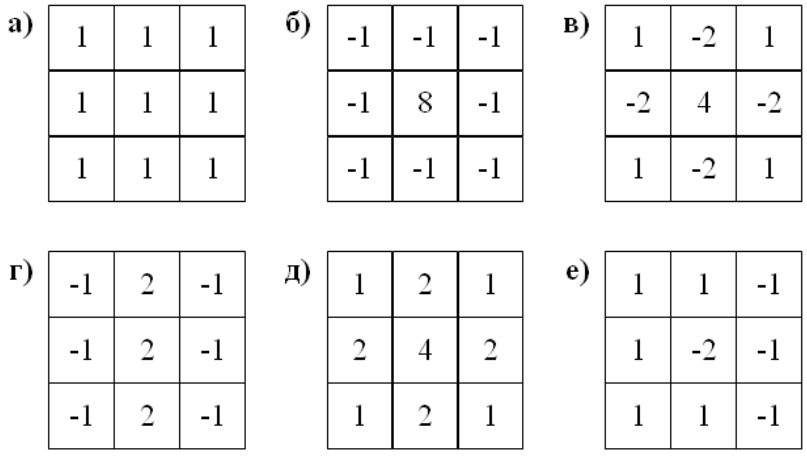


Рис. 1.62.

Структурная схема фильтра «скользящее среднее» приведена на рис. 1.63. Часто для нормирования делят сумму отсчетов на сумму коэффициентов, т.е. на 9. Коэффициенты $a_{i_1,i_2}=1$.

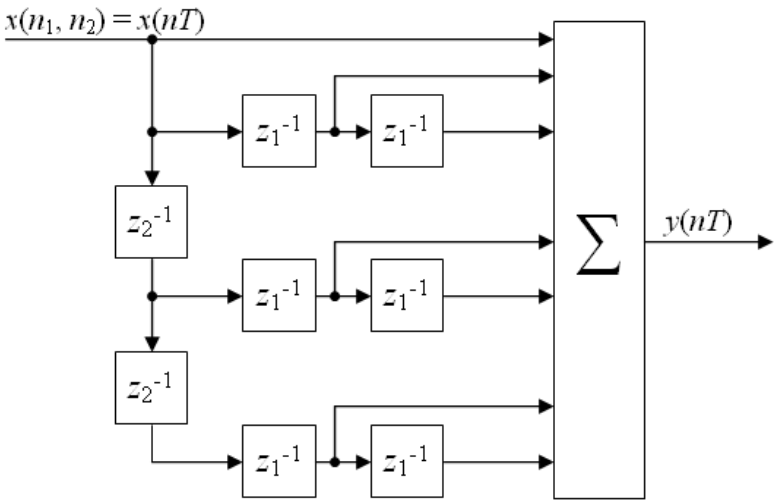


Рис. 1.63.

Разностное уравнение:

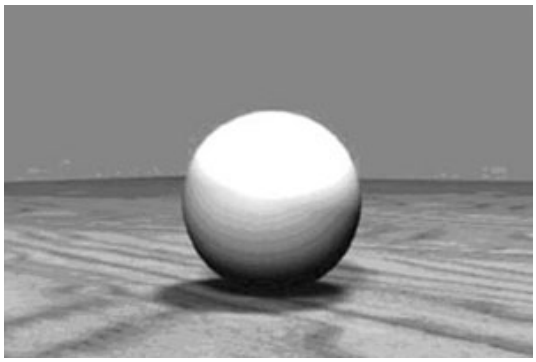
$$y(n_1,n_2) = x(n_1,n_2) + x(n_1-1,n_2) + x(n_1-2,n_2) + x(n_1,n_2-1) + x(n_1-1,n_2-1) + \\ + x(n_1-2,n_2-1) + x(n_1,n_2-2) + x(n_1-1,n_2-2) + x(n_1-2,n_2-2).$$

На рис. 1.64, б, представлен результат обработки исходного изображения (рис. 1.64, а) фильтром «скользящее среднее».

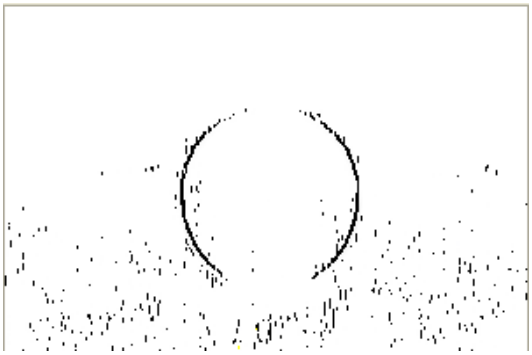
а *б*

Рис. 1.64.

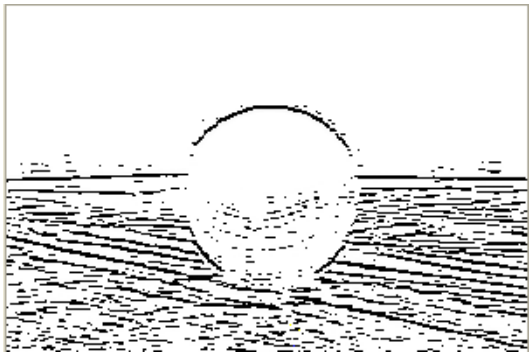
На рис. 1.65, б, представлен результат обработки исходного изображения (рис. 1.65, а) оператором выделения вертикальной границы, а на рис. 1.65, в – оператором выделения горизонтальной границы.



а



б



в

Рис. 1.65.

На рис. 1.66, а, показан дифференцирующий оператор, реализованный на основе маски (3×3). Незаполненные маски имеют дифференцирование не такое «жесткое». Их применяют для выделения малоразмерных объектов, расположенных на гладких фонах. В них размер маски согласован с размером объекта, а именно: размер маски выбирается в 2 раза большим, чем размер объекта. Пример незаполненной маски 3×3 показан на рис. 1.66, б.

-1	-1	-1
-1	8	-1
-1	-1	-1

а

0	-1	0
-1	4	-1
0	-1	0

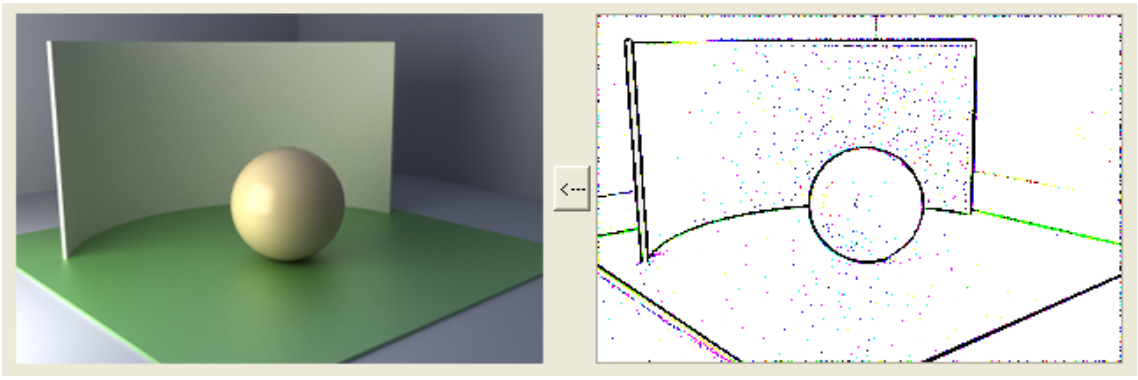
б

Рис. 1.66.

Разностное уравнение лапласиана для «четырех соседей» (рис. 1.66, б) имеет вид:

$$y(n_1, n_2) = -x(n_1 - 1, n_2) - x(n_1, n_2 - 1) + 4x(n_1 - 1, n_2 - 1)$$
$$- x(n_1 - 2, n_2 - 1) - x(n_1 - 1, n_2 - 2).$$

Результат использования фильтра представлен на рис. 1.67.



а *б*

Рис. 1.67.

Оператор двойного дифференцирования (рис. 1.68, а, б) используется для выделения малоразмерных деталей в изображении.

0	-1	0
-1	0	1
0	1	0

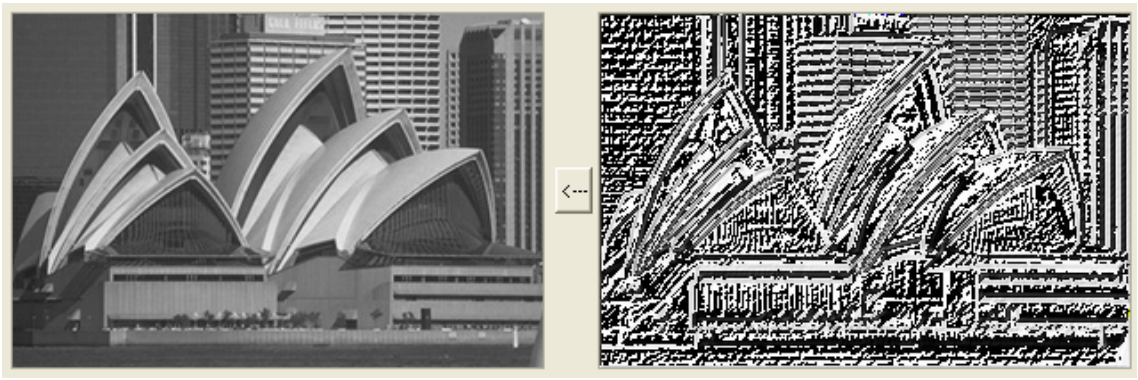
а

0	1	0
1	0	-1
0	-1	0

б

Рис. 1.68.

Результат обработки изображения показан на рис. 1.69.



а *б*

Рис. 1.69.

В лапласианах используются вторые частные производные, а в градиентных операторах – первые частные производные в различных пространственных направлениях (рис. 1.70, а – в).

Оператор «Запад»

1	1	-1
1	-2	-1
1	1	-1

а

Оператор «Север»

1	1	1
1	-2	1
-1	-1	-1

б

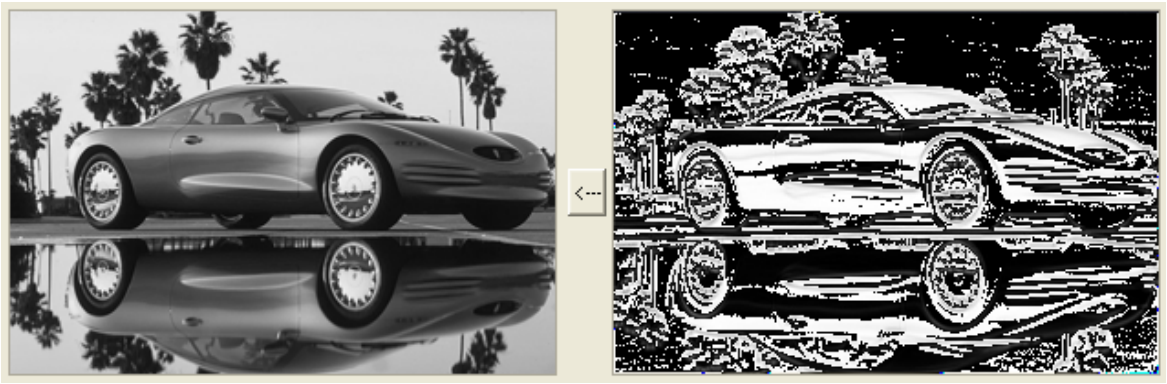
Оператор «Юго-запад»

1	-1	-1
1	-2	-1
1	1	1

в

Рис. 1.70.

Результат обработки изображения (рис. 1.71, а) оператором «Север» представлен на рис. 1.71, б.



а *б*

Рис. 1.71.

Задачи для самостоятельного решения

1. Используя программу проектирования цифровых фильтров, спроектировать фильтр по заданной частотной характеристике. Исходные данные указаны в табл. 1.1. Привести заданную и оптимизированную АЧХ, импульсную характеристику фильтра.

2. Промоделировать работу фильтра. Получить импульсную характеристику. Исследовать реакцию на ступенчатое воздействие и сигнал с шумом.

Табл. 1.1.

№	f_p, кГц	f_c, кГц	K	f_s, кГц	Тип фильтра	Порядок фильтра
1	1	2	1	10	НЧ	3
2	1	3	1	20	ВЧ	4
3	2	3	1,2	20	ВЧ	3
4	2	4	1,2	10	НЧ	4
5	2	3	0,8	10	ВЧ	4
6	1	2	0,8	20	НЧ	3

f_p–частота окончания полосы пропускания;
f_c–частота начла полосы задержки;
f_s–частота дискретизации;
K – коэффициент усиления.

3. Выполнить программную реализацию цифровых фильтров по заданным структурным схемам, представленным на рис. 1.72 и рис. 1.73. Получить передаточную функцию. Исследовать реакцию фильтра на одиночный импульс, на ступенчатое воздействие, синусоидальный сигнал с шумом.

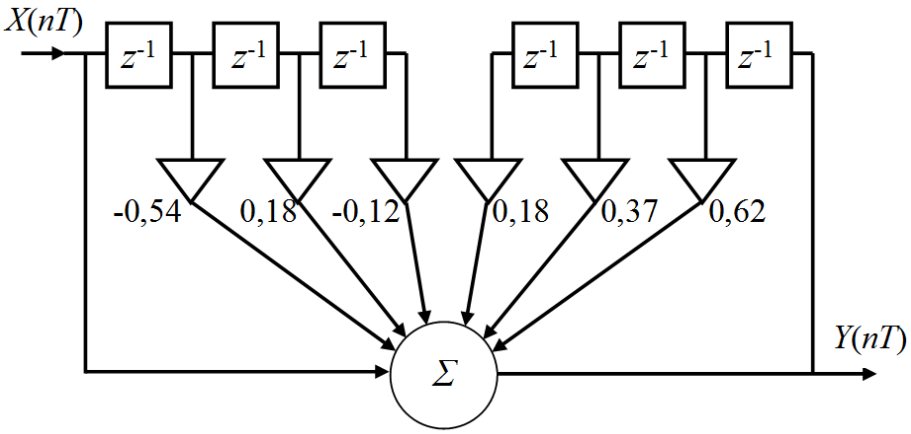


Рис. 1.72.

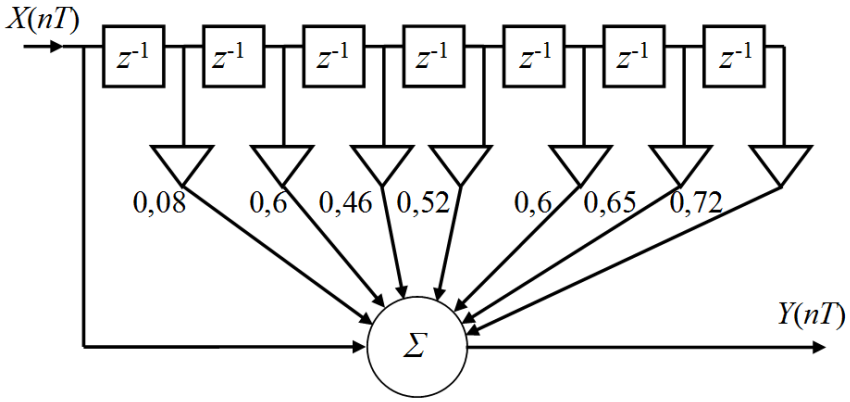


Рис. 1.73.

4. Реализовать программно цифровой фильтр по заданным передаточным функциям последовательно соединенных звеньев:

$$H_1(z) = \frac{1 - 1{,}963z^{-1} + z^{-2}}{1 - 1{,}69z^{-1} + 0{,}799z^{-2}} + 0{,}841,$$

$$H_2(z) = \frac{1 - 1{,}963z^{-1} + z^{-2}}{1 - 1{,}894z^{-1} + 0{,}907z^{-2}}.$$

Исследовать работу фильтра, подав на его вход различные сигналы.

5. Получить непрерывную передаточную функцию аналогового активного фильтра. Исходные данные приводятся в табл. 1.2. Выполнить билинейное Z-

преобразование для получения дискретной передаточной функции. Разработать программу, в которой реализовать спроектированный цифровой фильтр. Исследовать реакцию фильтра на дельта-импульс, ступенчатое воздействие, низкочастотный и высокочастотный гармонический сигнал, сигнал с шумом.

Табл. 1.2.

№	Аппроксимация	Тип фильтра	Порядок фильтра	Неравномерность (для Чебышева)	Частота среза, кГц
1	Баттерворта	НЧ	2		5
2	Чебышева	НЧ	2	1 дБ	10
3	Чебышева	ВЧ	2	3 дБ	15
4	Бесслеля	НЧ	2		8
5	Чебышева	ВЧ	2	3 дБ	9

6. Разработать программу, реализующую обработку изображения цифровыми фильтрами 5 порядка, выполняющие следующие функции:

– преобразование цветного изображения в черно-белое;

– выделение контуров изображения;

– замена определенного цвета в изображении на выбранный пользователем произвольный цвет;

– инвертирование цветов изображения;

– добавление шума в изображение;

– изменение яркости изображения;

– очистка изображения от шума медианным фильтром.

2. ЦИФРОВАЯ ФИЛЬТРАЦИЯ И СТАТИСТИЧЕСКАЯ ОБРАБОТКА СЛУЧАЙНЫХ СИГНАЛОВ И ПОМЕХ

2.1. Гистограмма, функция плотности вероятности и функция вероятностной меры

Основное применение цифровой обработки сигналов (ЦОС) состоит в понижении уровня помех, шумов и других нежелательных составляющих в получаемых данных. Они могут быть частью измеряемого сигнала, возникать из-за недостатка в системе сбора данных или быть побочным эффектом некоторой операции ЦОС. Методы статистики и вероятности позволяют измерить и классифицировать эти вредные составляющие.

Среднее значение (математическое ожидание), обозначаемое через μ, определяет в статистике среднее значение сигнала. Чтобы найти среднее нужно сложить все значения выборок и разделить сумму на количество выборок. В математической форме это выглядит следующим образом:

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} X_i \, .$$

Чтобы сохранить организацию данных после аналого-цифрового преобразования, каждой выборке присвоен номер выборки, или индекс. В ЦОС, как и в программировании, индекс нумеруется начиная с нуля. В рассматриваемом случае сигнал содержится в элементах от X_0 до $X_{N\text{-}1}$, i – индекс, проходящий через эти значения, μ – среднее значение.

В электронике средним значением часто называется величина DC (direct current – постоянный ток). В отличие от него, AC (alternating current – переменный ток) привязывается к тому, как сигнал изменяется около среднего значения. Если сигнал повторяет свою форму во времени, как синусоида или меандр, его отклонение может быть описано в виде амплитуды от пика до пика. Большинство сигналов не обладают определённым значением от пика до пика, а имеют случайный характер. В этом случае должен использоваться обобщённый метод, называемый *стандартным отклонением,* обозначаемым через σ.

Выражение $|x_i - \mu|$ показывает, как i-тое значение отклоняется (отличается) от среднего значения. *Среднее отклонение* сигнала вычисляется через суммирование отклонений всех выборок и делением на их количество N. Отметим, что используется абсолютное значение отклонения, иначе отрицательные и положительные величины дадут среднее значение около нуля. Хотя среднее отклонение является удобным и простым, оно не используется в статистике, поскольку не соответствует физике работы сигналов. В большинстве случаев, важным параметром является не *отклонение* от среднего значения, а *мощность* отклонения. К примеру, когда случайные сигналы объединяются в электрической схеме, результирующий

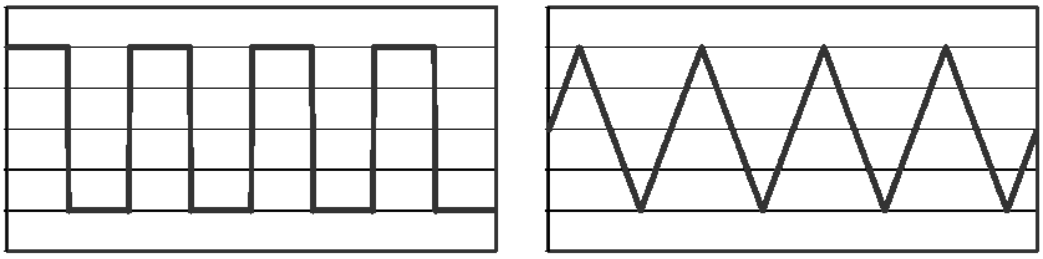
шум равен суммарной *мощности* индивидуальных сигналов, а не их суммарной *амплитуде.*

Стандартное (среднеквадратичное) отклонение похоже на *среднее отклонение*, за исключением того, что усредняется мощность, а не амплитуда. Это достигается возведением в квадрат каждого отклонения перед суммированием. В завершение операции проводится извлечение квадратного корня. В математической форме стандартное отклонение рассчитывается как:

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(X_i - \mu)^2 .$$

Сигнал содержится в элементах X_i, μ – среднее значение, σ – стандартное отклонение. Усреднение проводится делением на N-1, вместо N. Это является особенностью этого выражения.

Величина σ^2 часто встречается в статистике и называется *дисперсия.* Стандартное отклонение является мерой того, как сигнал отличается от среднего значения. Дисперсия представляет мощность этого отклонения. Другим термином, часто используемым в электронике, является среднеквадратичное значение (rms value – root-mean-square value). По определению стандартное отклонение измеряет только AC порцию сигнала, в то время как среднеквадратичное значение измеряет как AC, так DC компоненты. Если сигнал не содержит DC компоненты (рис. 2.1, а, б), его среднеквадратичное значение идентично стандартному отклонению.



а *б*

Рис. 2.1.

Приведённый метод расчета среднего значения и стандартного отклонения удовлетворяет большинству приложений, однако, он имеет два ограничения. Во-первых, среднее значение гораздо больше, чем значение стандартного отклонения. В уравнении производится вычитание двух чисел, которые очень близки по значению, что может привести к ошибкам округления в расчётах. Во-вторых, часто предпочтительно пересчитать среднее значение и стандартное отклонение для новой выборки сигналов. Такой тип вычисления называется *динамическая статистика.* При использовании приведенных уравнений для динамической статистики необходимо для каждого расчёта вводить все выборки сигнала, что приводит к неэффективному использованию процессорного времени и памяти

компьютера. Для решения этой проблемы можно преобразовать приведенные выражения для альтернативного вычисления стандартного отклонения. Расчёт стандартного отклонения для динамической статистики:

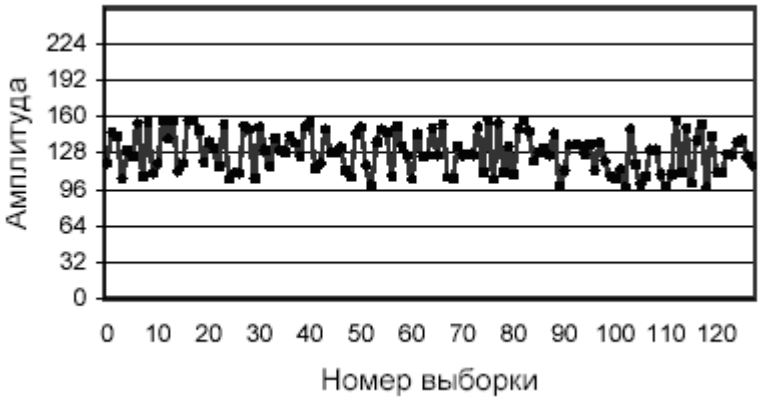
$$\sigma^2 = \frac{1}{N-1}\left[\sum_{i=0}^{N-1} x_i^{\,2} - \frac{1}{N}\left(\sum_{i=0}^{N-1} x_i\right)^2\right],$$

или в упрощённой форме

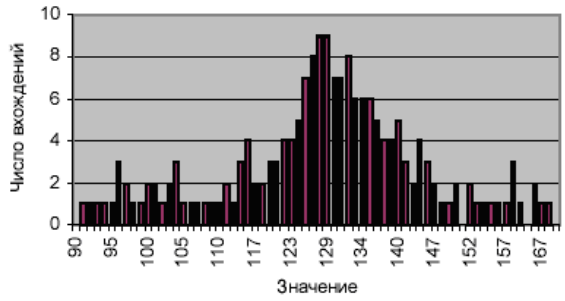
$$\sigma^2 = \frac{1}{N-1}\left[\textit{сумма квадратов} - \frac{\textit{сумма}^2}{N}\right].$$

Предположим, что мы подключили 8-битный аналого-цифровой преобразователь к компьютеру и собрали 256000 выборок некоторого сигнала. Пример на рис. 2.2, а, показывает 128 точек сигнала, которые могли быть частью полученных данных. Значение каждой выборки может изменяться от 0 до 255. Хотя сигнал на рис. 2.2, а, является дискретным, на диаграмме он отображается сплошными линиями. Это поясняется тем, что число выборок велико, чтобы быть различимым в виде отдельных точек. На графиках, которые показывают сигнал с менее чес 100 выборками, обычно показывают отдельные точки. Непрерывная линия может показать, что происходит между выборками, или помочь пользователю проследить тренд в зашумленных данных.

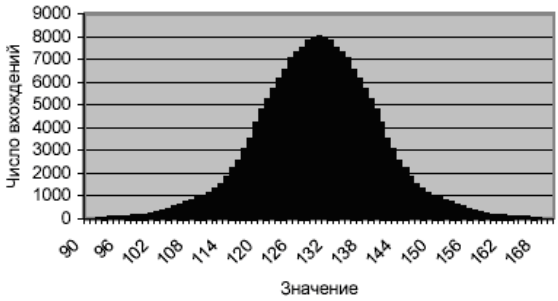
Гистограмма показывает число выборок в сигнале со всеми возможными значениями. Рис. 2.2, б, показывает гистограмму для 128 выборок. Например, 2 точки имеют значение 110, 8 имеют значение 131, 0 точек имеет значение 170 и т.д. Представим гистограмму через H_i, где i является индексом, который лежит в диапазоне от 0 до M-1, где M равняется числу возможных значений. Рис. 2.2, в, показывает гистограмму, для составления которой использовались все 256 тысяч точек сигнала. Как видно из рисунка, чем больше число используемых выборок, тем более сглаженной выглядит гистограмма. Как и в случае со средним значением, статистический шум (грубость) гистограммы обратно пропорционален квадрату числа используемых точек.



а



б



в

Рис. 2.2.

Сумма всех значений гистограммы должна быть равна числу точек сигнала:

$$N = \sum_{i=0}^{M-1} H_i ,$$

где H_i – гистограмма, N – число точек сигнала, M – число точек гистограммы.

Гистограмма может использоваться для эффективного расчёта среднего значения и стандартного отклонения большого набора данных. Это особенно важно для изображений, которые могут содержать миллионы выборок. Гистограмма группирует выборки с одинаковым значением. Это позволяет рассчитать статистику при работе с несколькими группами, а не с огромным числом отдельных выборок. Среднее значение μ и стандартное отклонение σ^2 могут быть вычислены из гистограммы с использованием следующих выражений:

$$\mu = \frac{1}{N} \sum_{i=0}^{M-1} iH_i ,$$

$$\sigma^2 = \frac{1}{N-1} \sum_{i=0}^{M-1} (i-\mu)^2 H_i .$$

Вычисление гистограммы происходит быстро, поскольку требует только индексации и инкремента. В сравнении с этим, расчёт среднего значения и стандартного отклонения требует времени, которое тратится на операции сложения и умножения. Стратегия такого метода заключается в том, что использование медленных операций проводится на нескольких точках гистограммы, а не на множестве отсчётов сигнала. Это делает алгоритм намного быстрее тех, которые были описаны ранее, почти в 10 раз для длинных сигналов, если вычисление осуществляется на стандартных компьютерах.

Важность функции вероятной меры состоит в том, что она описывает вероятность генерации определённого значения. Рассмотрим сигнал, изображенный на рис. 2.3, б. Необходимо ответить на такой вопрос: какова вероятность того, что отсчёт, взятый из такого сигнала, будет иметь величину 120? Рис. 2.3, б, даёт ответ – 0,03, или 1 шанс из 34. Какова вероятность того,

что значение отсчёта будет больше 150? Суммируя величины функции для 151, 152, …, 255, получим ответ – 0,0122, или 1 шанс из 82. Таким образом, каждые 82 отсчёта можно ожидать значение больше 150. Какова вероятность, что значение отсчета будут между 0 и 255? Суммируя все величины PMF, получим вероятность 1,0, что является бесспорным.

Гистограмма и функция вероятностной меры могут использоваться только с дискретными данными, такими как дискретные сигналы, записанные на компьютер. Подобная концепция применима и для непрерывных сигналов, таких как напряжение в аналоговой электронике. Функция плотности вероятности (Probability Density Function – PDF), называется также функцией распределения вероятности, является для непрерывных сигналов тем же самым, чем является функция вероятностной меры для дискретных сигналов.

Чтобы вычислить вероятность, необходимо умножить плотность вероятности на диапазон величин. Например, вероятность того, что сигнал будет находиться между 120,4 и 120,5 мВ равна (120,5–120,4)·0,03=0,003 и т.п. Если PDF в интересующем диапазоне непостоянна, то умножение сводится к интегрированию по этому интервалу. Другими словами, площадь под PDF ограничена определёнными значениями. Поскольку сигнал должен всегда принимать некоторое значение, общая площадь под кривой PDF, интеграл от -∞ до +∞, будет равна единице.

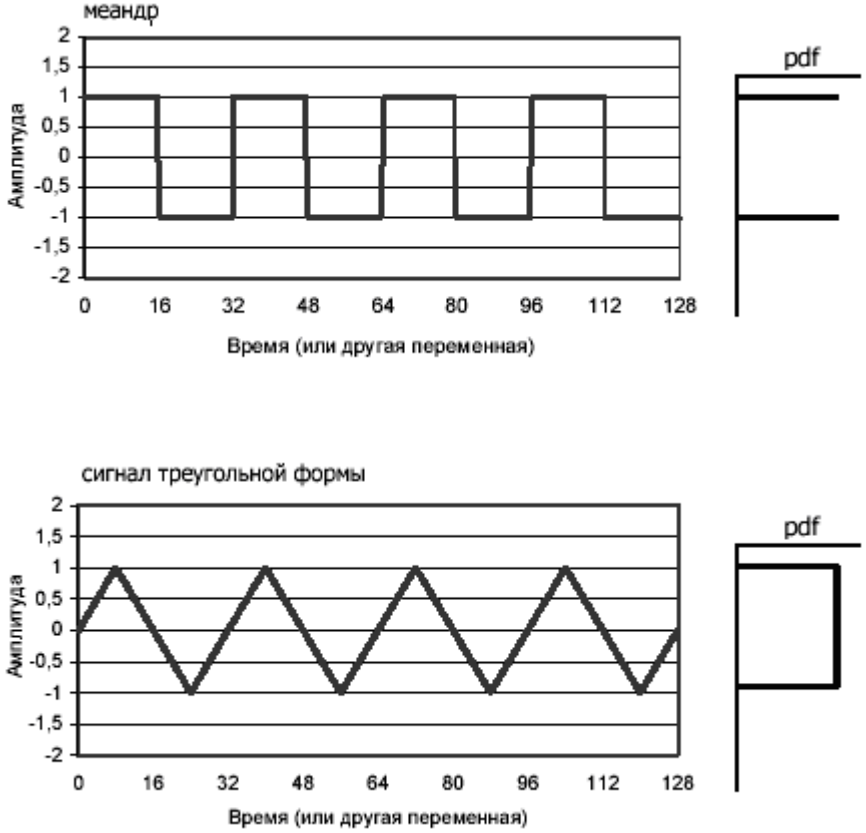


Рис. 2.3.

Сигналы, сформированные от случайных процессов, обычно имеют колоколообразную форму PDF. Такая функция называется нормальным распределением, или распределением Гаусса, по имени немецкого математика Карла Фидриха Гаусса (1777-1885). Основная форма кривой генерируется от экспоненты:

$$y(x) = e^{-x^2}.$$

При добавлении регулируемых среднего значения и стандартного отклонения эта кривая может быть преобразована в законченное распределение Гаусса. Выражение должно быть нормализовано так, чтобы площадь под кривой равнялась единице. Это стандартное требование всех функций распределения вероятности. Общая форма нормального распределения выражается через:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-u)^2}{2\sigma^2}}.$$

$P(x)$ – функция распределения вероятности, u – среднее значение, σ – стандартное отклонение.

2.2. Цифровая генерация помех

Случайная помеха является важной темой, как в электронике, так и в ЦОС. Например, она ограничивает насколько маленький сигнал можно измерить прибором, дистанцию, на которой может работать радиостанция, какое количество радиоизлучения необходимо для выработки рентгеновского изображения. Распространённой потребностью в ЦОС является генерация сигналов, которые похожи на различные типы случайных помех. Это необходимо для проверки эффективности алгоритмов, которые должны работать в условиях наличия помех.

Сердцем цифровой генерации помех является генератор случайных чисел. Большинство языков программирования содержат эту стандартную функцию. Оператор *X*=rand(), загружает в переменную *X* новое случайное число, каждый раз, когда он встречается в программе. Каждое случайное число принимает значение с одинаковой вероятностью. Для того чтобы задать диапазон изменения случайных значений, в языке Си используются выражение *X*=rand()%100. Символ «%» – это оператор, который позволяет получить остаток от деления. В приведенном примере остаток от деления на 100 не может быть больше 99, поэтому случайное значение, записанное в переменную *X*, будет лежать в диапазоне от 0 до 99. Рис. 2.4, а, показывает сигнал, который сформирован из 128 выборок, взятых по принципу такого генератора случайных чисел. Среднее значение основного процесса, который генерирует этот сигнал, составляет 0,5. Стандартное отклонение составляет 0,29 ($1/\sqrt{12}$), распределение равномерно между нулём и единицей.

Алгоритмы должны проверятся с использованием такого типа данных, которые они встречают в реальных условиях. Это создаёт необходимость

цифровой генерации помехи с гауссовой функцией распределения вероятности. Существует два способа для генерации таких сигналов, используя генератор случайных чисел. Рис. 2.4, а, иллюстрирует первый метод. Рис. 2.4, б, показывает сигнал, состоящий из выборок, полученных сложением двух случайных чисел, т.е. X=(rand()%51+rand()%51)/100. Поскольку каждое случайное число может принимать значение от 0 до 50, их сумма будет находиться от 0 до 100. Разделив это значение на 100, получим диапазон изменения X от 0 до 1. Переменная X имеет тип double. Среднее значение будет составлять 0,5, а стандартное отклонение $1/\sqrt{6}$ (когда складываются два дискретных числа, дисперсия также складывается). Как показано, PDF сигнала изменилась от равномерного до треугольного распределения. Поэтому сигнал находится большее своё время вокруг единичного значения, и меньше около нуля и двух.

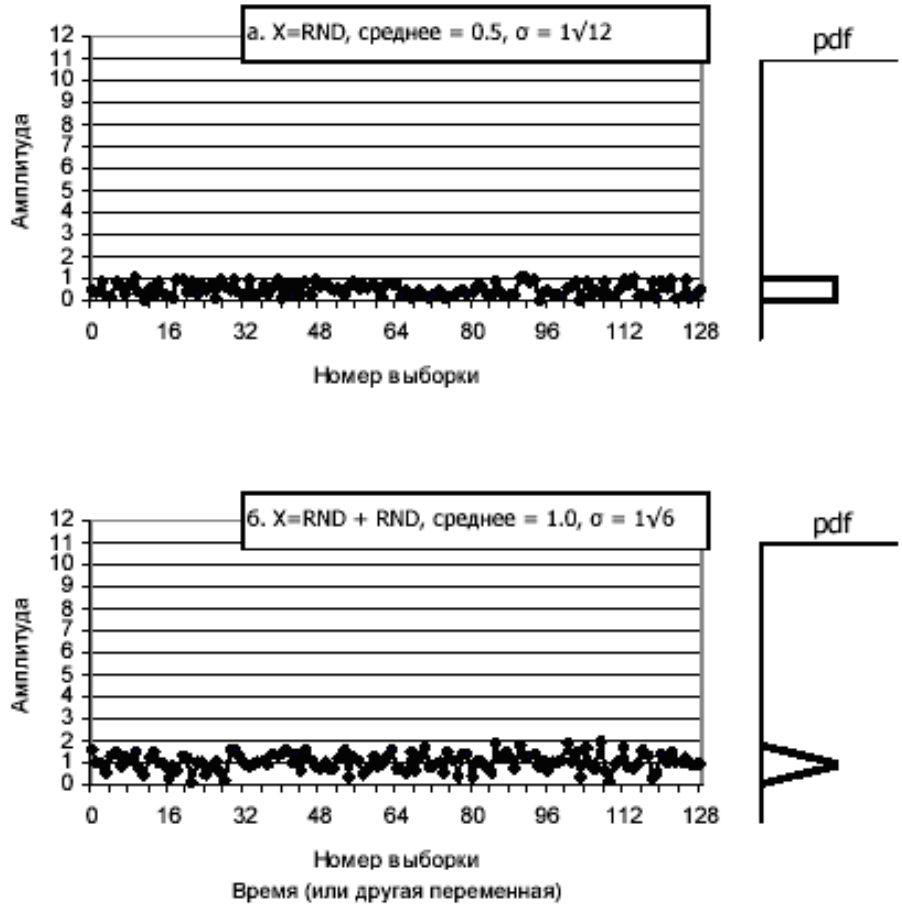


Рис. 2.4.

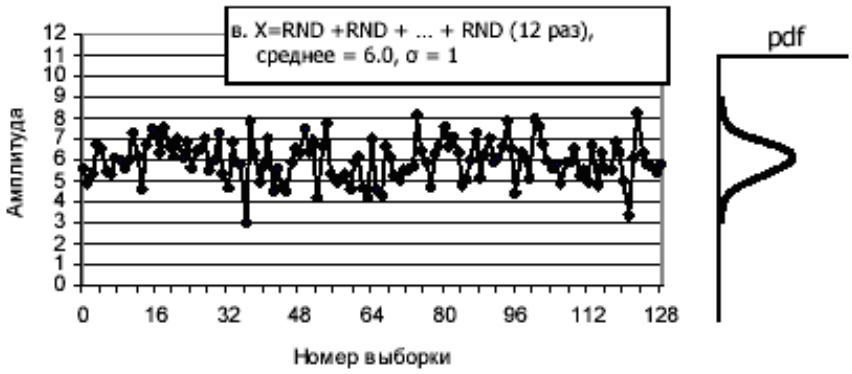


Рис. 2.5.

Математическая основа этого алгоритма содержится в теореме о центральном пределе, одной из важных концепций о вероятности. В простейшей форме теорема о центральном пределе говорит, что сумма случайных чисел становится нормально распределённой (рис. 2.5), чем больше чисел участвует в сложении. Теорема о центральном пределе не требует, чтобы отдельные случайные числа были получены из какого-либо распределения, или были получены из одинакового распределения. Теорема о центральном пределе показывает причину, почему нормально распределённые сигналы настолько часто встречаются в природе. Всякий раз при взаимодействии различных случайных воздействий, результирующая PDF будет нормально распределена.

Второй метод генерации нормально распределённых чисел состоит в том, что реализуются два генератора случайных чисел для получения R_1 и R_2. Нормально распределённое случайное число X может быть тогда найдено как:

$$X = \sqrt{-2\log R_1}\cos(2\pi R_2).$$

R_1 и R_2 являются случайными числами с равномерным распределением между 0 и 1. Результат, X, является нормально распределённым случайным числом со средним значением «0» и стандартным отклонением «1». Логарифм является натуральным, косинус считается в радианах.

Как и в предыдущем случае, этот метод позволяет генерировать нормально распределённое случайное число с произвольными средним значением и стандартным отклонением. Для этого необходимо рассчитать значение по приведенному выражению, умножить на желаемое стандартное отклонение и добавить желаемое среднее значение.

2.3. Программная реализация алгоритмов статистической обработки сигналов

На рис. 2.6 представлен вид интерфейса диалогового приложения для статистической обработки случайных сигналов. На диалоге расположены два элемента Static Text, которые будут использоваться для построения осциллограмм сигналов. В правой части диалогового окна расположены

элементы текстового поля Edit (в свойствах этих элементов установлена метка Read only). Эти элементы будут использоваться для отображения рассчитанных значений математического ожидания (текстовое поле m), среднеквадратического отклонения (текстовое поле s) и дисперсии (текстовое поле D), а также максимального и минимального значений сигнала. Элементы текстовых полей mh, sh, Dh также используются для отображения рассчитанных значений математического ожидания, среднеквадратического отклонения и дисперсии, но с использованием формул динамической статистики. Это позволит сравнить точность методов расчета и проверить правильность реализации алгоритмов.

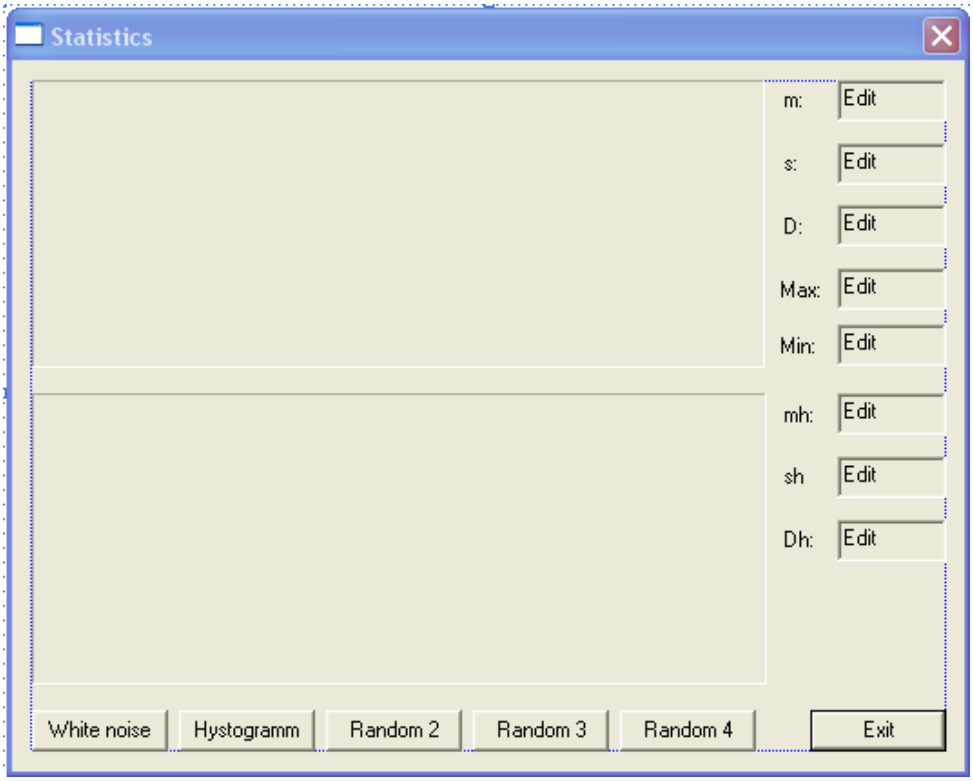


Рис. 2.6.

С элементами управления, расположенными на диалоговом окне, необходимо связать управляющие переменные. На рис. 2.7 представлено окно Class Wizard, в котором указаны названия и тип созданных переменных.

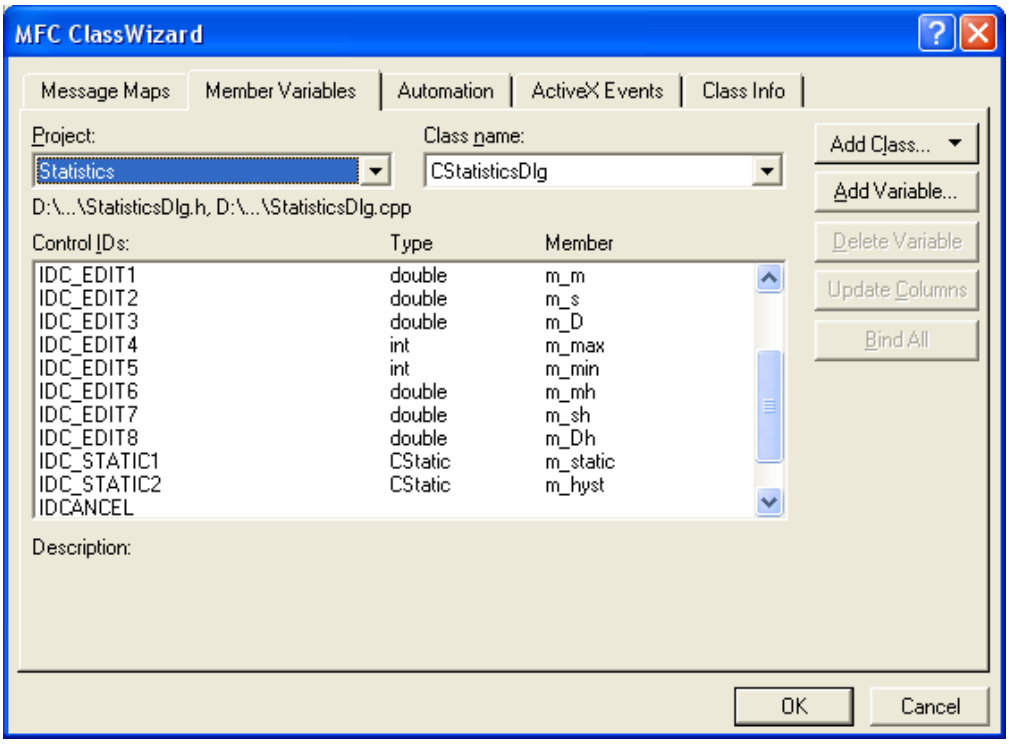


Рис. 2.7.

В класс диалога необходимо добавить две переменные-члены класса и функцию:

```
int N;
int * m_d;
void Calculate();
```

В функцию OnInitDialog() добавляем следующий код:

```
CRect r;
m_static.GetClientRect(&r);
N=r.Width();
m_d=new int[10000];
```

Функция-обработчик нажатия на кнопку White Noise выполняет генерацию 10000 случайных выборок, заполнение массива данных и отображение осциллограммы сгенерированного случайного сигнала в верхней части диалогового окна на элементе Static Text, который используется как стилизированное окно осциллографа, представлена ниже.

```
void CStatisticsDlg::OnButton1()
{
 for(int i=0; i<10000; i++) m_d[i]=rand()%100;
 CRect r;
 m_static.GetClientRect(&r);
```

```
  CClientDC dc(&m_static);
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(255,0,0));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
        dc.MoveTo(i, r.Height()/2-m_d[i]+50);
        dc.LineTo(i-1, r.Height()/2-m_d[i-1]+50);
  }
  dc.SelectObject(op);
  Calculate();
}
```

Функция-обработчик нажатия на кнопку Hystogramm выполняет построение гистограммы случайного сигнала представлена ниже. Гистограмма отображается в нижней части диалогового окна.

```
void CStatisticsDlg::OnButton2()
{
  CRect r;
  m_static.GetClientRect(&r);
  CClientDC dc(&m_hyst);
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));

  int H[100]={0};
  for(int i=0;i<10000;i++) H[m_d[i]]++;
  for(i=0;i<100;i++) H[i]/=2;

  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(0,0,200));
  op=dc.SelectObject(&p);

  for(i=0;i<100;i++)
  {
        dc.MoveTo(i*4,r.Height());
        dc.LineTo(i*4,r.Height()-H[i]);
  }
  dc.SelectObject(op);
}
```

Обработчик нажатия на кнопку Exit, которая закрывает диалоговое окно и очищает выделенную память для хранения элементов массива данных, имеет вид:

```
void CStatisticsDlg::OnOK()
{
  delete [] m_d;
```

```
  CDialog::OnOK();
}
```

Функция-обработчик нажатия на кнопку Random 2 выполняет генерацию массива данных, каждый элемент которого определяется как сумма двух случайных значений, имеет вид:

```
void CStatisticsDlg::OnButton3()
{
  for(int i=0; i<10000; i++) m_d[i]=rand()%50+rand()%50;
  CRect r;
  m_static.GetClientRect(&r);
  CClientDC dc(&m_static);
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(255,0,0));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
        dc.MoveTo(i, r.Height()/2-m_d[i]+50);
        dc.LineTo(i-1, r.Height()/2-m_d[i-1]+50);
  }
  dc.SelectObject(op);
  Calculate();
}
```

Функция-обработчик нажатия на кнопку Random 3 выполняет генерацию массива данных, каждый элемент которого определяется как сумма трех случайных значений, имеет вид:

```
void CStatisticsDlg::OnButton4()
{
  for(int i=0; i<10000; i++) m_d[i]=rand()%33+rand()%34+rand()%33;
  CRect r;
  m_static.GetClientRect(&r);
  CClientDC dc(&m_static);
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(255,0,0));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
        dc.MoveTo(i, r.Height()/2-m_d[i]+50);
        dc.LineTo(i-1, r.Height()/2-m_d[i-1]+50);
  }
  dc.SelectObject(op);
  Calculate();
}
```

Функция-обработчик нажатия на кнопку Random 3 выполняет генерацию массива данных, каждый элемент которого определяется как сумма четырех случайных значений, имеет вид:

```
void CStatisticsDlg::OnButton5()
{
for(int i=0; i<10000; i++) m_d[i]=rand()%25+rand()%25+rand()%25+rand()%25;
  CRect r;
  m_static.GetClientRect(&r);
  CClientDC dc(&m_static);
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(255,0,0));
  op=dc.SelectObject(&p);
  for(i=1;i<N;i++)
  {
        dc.MoveTo(i, r.Height()/2-m_d[i]+50);
        dc.LineTo(i-1, r.Height()/2-m_d[i-1]+50);
  }
  dc.SelectObject(op);
  Calculate();
}
```

Функция Calculate представлена ниже.

```
void CStatisticsDlg::Calculate()
{
  m_m=0;
  for(int i=0;i<10000;i++)
  {
        m_m=m_m+m_d[i];
  }
  m_m=(double)m_m/10000;
  m_D=0;
  m_s=0;
  for(i=0;i<10000;i++)
  {
        m_D=m_D+(m_d[i]-m_m)*(m_d[i]-m_m);
  }
  m_D=(double)m_D/(9999);
  m_s=sqrt(m_D);

  m_max=m_d[0];
  m_min=m_d[0];
  for(i=0;i<10000;i++)
  {
        if(m_max<m_d[i]) m_max=m_d[i];
        if(m_min>m_d[i]) m_min=m_d[i];
  }
  int H[100]={0};
```

```
  for(i=0;i<10000;i++) H[m_d[i]]++;
  m_mh=0;
  for(i=0;i<100;i++)
  {
          m_mh=m_mh+i*H[i];
  }
  m_mh=(double)m_mh/10000;
  m_sh=0;
  m_Dh=0;
  for(i=0;i<100;i++)
  {
          m_Dh=m_Dh+(i-m_mh)*(i-m_mh)*H[i];
  }
  m_Dh=(double)m_Dh/9999;
  m_sh=sqrt(m_Dh);
  UpdateData(false);
}
```

На рис. 2.8 представлен вид интерфейса программы. После нажатия на кнопку White Noise пользователь увидит осциллограмму случайного сигнала в верхней части диалога. Нажатие на кнопку Hystogramm приведет к построению гистограммы в нижней части диалога. Как видно из рисунка, в сгенерированном случайном сигнале количество появлений каждого значения в диапазоне от 0 до 99 очти одинаковое. С увеличением объема выборки (в примере используется 10000 элементов), форма гистограмма будет приближаться к прямоугольной.

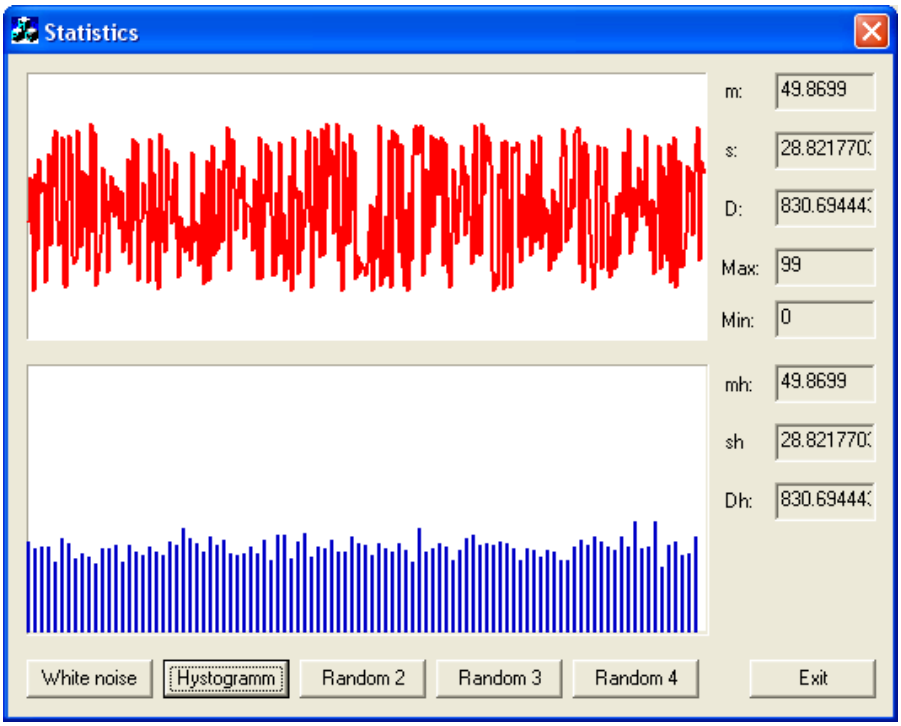


Рис. 2.8.

На рис. 2.9 представлено окно программы при генерации сигнала, состоящего из суммы двух случайных чисел. Полученный в результате сигнал имеет треугольное распределение.

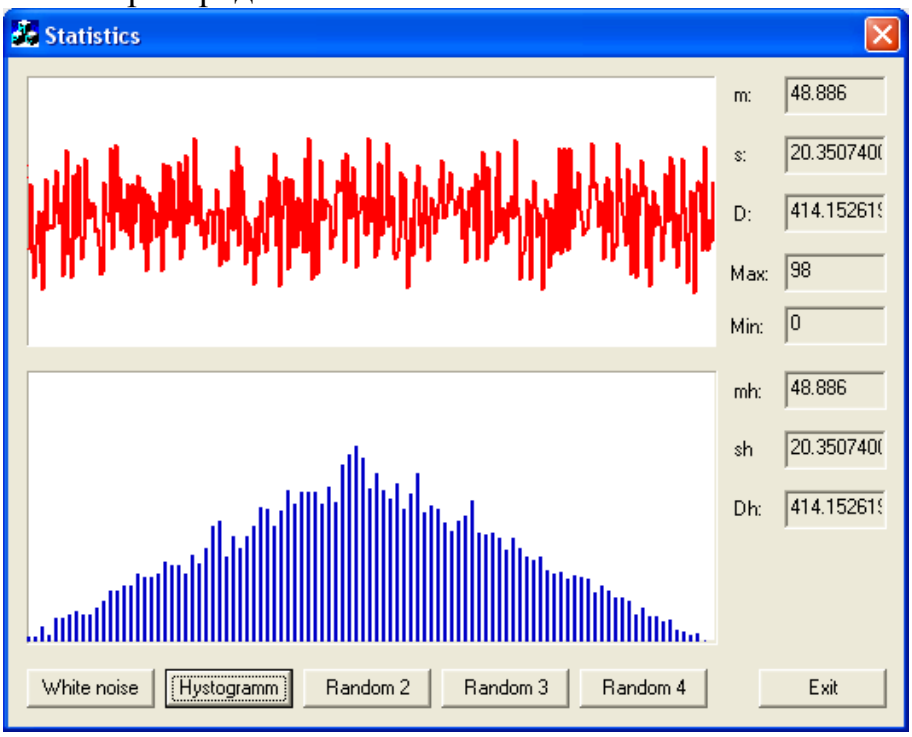


Рис. 2.9.

На рис. 2.10 представлено окно программы при генерации сигнала, состоящего из суммы четырех случайных чисел. Полученный в результате сигнал имеет распределение, приближающееся к нормальному.

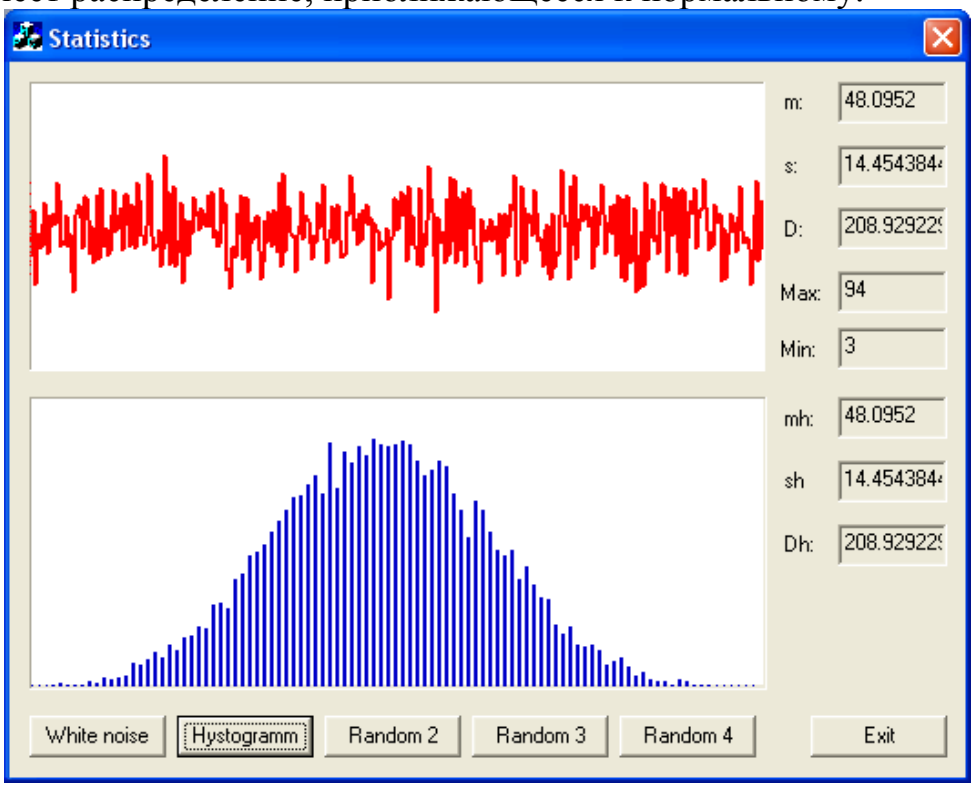


Рис. 2.10.

В правой части диалогового окна можно увидеть рассчитанные статистические характеристики случайного сигнала. Как видно, расчет по классическим формулам и по формулам динамической статистики дает одинаковый результат. Это подтверждает правильность реализованных алгоритмов обработки сигнала.

2.4. Программная реализация алгоритма медианной фильтрации

Медианные фильтры достаточно часто применяются на практике как средство предварительной обработки цифровых данных. Специфической особенностью фильтров является явно выраженная избирательность по отношению к элементам массива, представляющим собой немонотонную составляющую последовательности чисел в пределах окна (апертуры) фильтра, и резко выделяющихся на фоне соседних отсчетов. В то же время на монотонную составляющую последовательности медианный фильтр не действует, оставляя её без изменений. Благодаря этой особенности, медианные фильтры при оптимально выбранной апертуре могут, например, сохранять без искажений резкие границы объектов, эффективно подавляя некоррелированные или слабо коррелированные помехи и малоразмерные детали. Это свойство позволяет применять медианную фильтрацию для устранения аномальных значений в массивах данных, уменьшения выбросов и импульсных помех. Характерной особенностью медианного фильтра является его нелинейность. Во многих случаях применение медианного фильтра оказывается более эффективным по сравнению с линейными фильтрами, поскольку процедуры линейной обработки являются оптимальными при равномерном или гауссовом распределении помех, что в реальных сигналах может быть далеко не так. В случаях, когда перепады значений сигналов велики по сравнению с дисперсией аддитивного белого шума, медианный фильтр дает меньшее значение среднеквадратической ошибки по сравнению с оптимальными линейными фильтрами. Особенно эффективным медианный фильтр оказывается при очистке сигналов от импульсных шумов при обработке изображений, акустических сигналов, передаче кодовых сигналов и т.п.

Алгоритм медианной фильтрации заключается в следующем:

1. Выбирается нечетное количество k отсчетов принятого сигнала, начиная с i-го отсчета.

2. Отсчеты располагаются в порядке возрастания по амплитуде.

3. Центральный отсчет данной выборки, который называется медианой, берется как представитель этой выборки.

4. Выбираются новые k отсчетов принятого сигнала, начиная с (i+1)-го отсчета, и выполняются действия 2-4.

На рис. 2.11 представлена входная последовательность (пунктирная линия) и последовательность на выходе медианного фильтра при k=3 (сплошная линия).

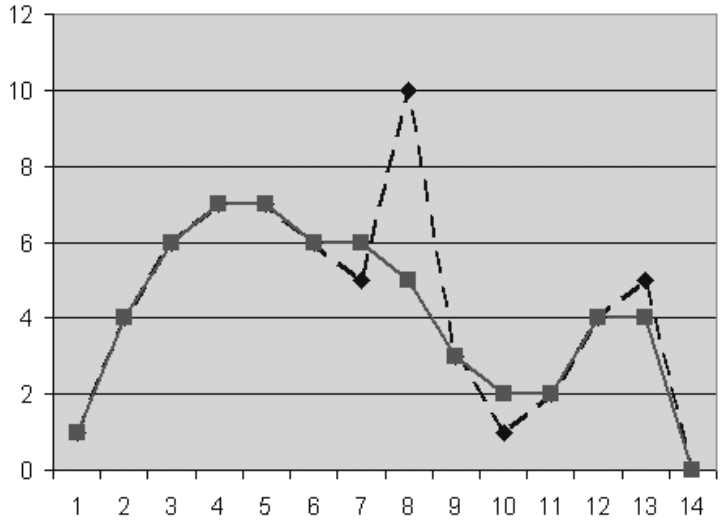


Рис. 2.11.

На рисунке точками обозначены отсчеты передаваемого сигнала, который является гладким. При передаче в канале возникла импульсная помеха, в результате чего седьмой отсчет получил значение 10, и в принятой последовательности образовался выброс. В табл. 2.1 показан пример применения медианного фильтра для одномерного сигнала с окном размером в три отсчёта ко входному массиву. Импульсная помеха в исходном сигнале (ошибочный отсчет) выделена жирным шрифтом в скобках.

Табл. 2.1.

№ отсчета	Входной отсчет *n*	Входной отсчет *n*-1	Входной отсчет *n*-2	Отсортированная последовательность			Выход фильтра
				m-1	медиана *m*	*m*+1	
1	1	0	0	0	0	1	0
2	4	1	0	0	1	4	1
3	6	4	1	1	4	6	4
4	7	6	4	4	6	7	6
5	7	7	6	6	7	7	7
6	6	7	7	6	7	7	7
7	5	6	7	5	6	7	6
8	**(10)**	5	6	5	6	**(10)**	6
9	3	**(10)**	5	3	5	**(10)**	5
10	1	3	**(10)**	1	3	**(10)**	3
11	2	1	3	1	2	3	2
12	4	2	1	1	2	4	2
13	5	4	2	2	4	5	4
14	0	5	4	0	4	5	4
15	0	0	5	0	0	5	0

Медианная фильтрация – эффективная процедура обработки сигналов, подверженных воздействию импульсных помех. Медианный фильтр представляет собой оконный фильтр, последовательно скользящий по массиву сигнала, и возвращающий на каждом шаге один из элементов, попавших в окно (апертуру) фильтра. Значения отсчётов внутри окна фильтра сортируются в порядке возрастания (или убывания), и значение, находящееся в середине упорядоченного списка (медиана m), поступает на выход фильтра. В случае четного числа отсчетов в окне выходное значение фильтра равно среднему значению двух отсчетов в середине упорядоченного списка. Окно перемещается вдоль фильтруемого сигнала и вычисления повторяются. Медианный фильтр относится к числу нелинейных фильтров, заменяющим медианным значением аномальные точки и выбросы независимо от их амплитудных значений, и является устойчивым по определению, способным аннулировать даже бесконечно большие отсчеты.

Таким образом, медианная фильтрация осуществляет замену значений отсчетов в центре апертуры медианным значением исходных отсчетов внутри апертуры фильтра. На практике апертура фильтра для упрощения алгоритмов обработки данных, как правило, устанавливается с нечетным числом отсчетов.

Алгоритм медианной фильтрации обладает явно выраженной избирательностью к элементам массива с немонотонной составляющей последовательности чисел в пределах апертуры и наиболее эффективно исключает из сигналов одиночные выбросы, отрицательные и положительные, попадающие на края ранжированного списка. С учетом ранжирования в списке медианные фильтры хорошо подавляют шумы и помехи, протяженность которых составляет менее половины окна. Стабильной точкой является последовательность (в одномерном случае) или массив (в двумерном случае), которые не изменяются при медианной фильтрации. В одномерном случае стабильными точками медианных фильтров являются "локально-монотонные" последовательности, которые медианный фильтр оставляет без изменений. Исключение составляют некоторые периодические двоичные последовательности.

Благодаря этой особенности, медианные фильтры при оптимально выбранной апертуре могут сохранять без искажений резкие границы объектов, подавляя некоррелированные и слабо коррелированные помехи и малоразмерные детали. При аналогичных условиях алгоритмы линейной фильтрации неизбежно «смазывает» резкие границы и контуры объектов. На рис. 2.12 приведен пример обработки сигнала с импульсными шумами медианным и треугольным фильтрами с одинаковыми размерами окна k=3. Преимущество медианного фильтра очевидно.

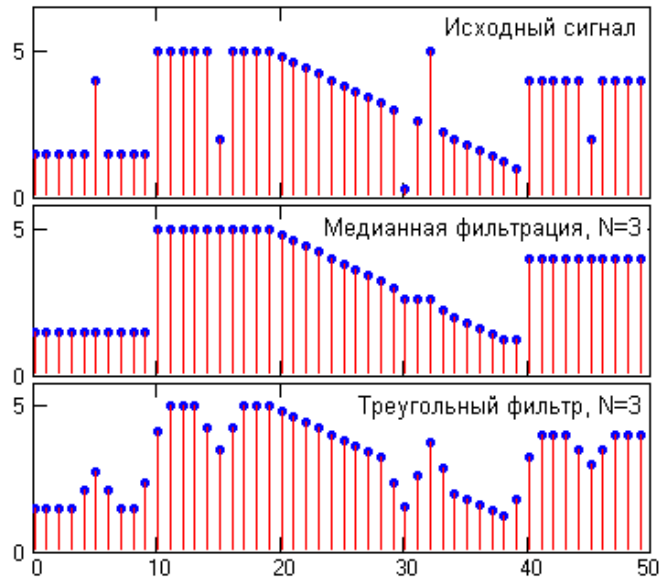


Рис. 2.12.

В качестве начальных и конечных условий фильтрации обычно принимаются концевые значения сигналов, либо медиана находится только для тех точек, которые вписываются в пределы апертуры.

При регистрации, обработке и обмене данными в современных измерительно-вычислительных и информационных системах потоки сигналов кроме полезного сигнала $s(t–t_0)$ и флуктуационных шумов $q(t)$ содержат, как правило, импульсные потоки $g(t)=\sum_k d(t–t_k)$ различной интенсивности с регулярной или хаотической структурой:

$$x(t)=s(t–t_0)+g(t)+q(t).$$

Под импульсным шумом понимается искажение сигналов большими импульсными выбросами произвольной полярности и малой длительности. Причиной появления импульсных потоков могут быть как внешние импульсные электромагнитные помехи, так и наводки, сбои и помехи в работе самих систем. Совокупность статистически распределенного шума и потока квазидетерминированных импульсов представляет собой комбинированную помеху. Радикальный метод борьбы с комбинированной помехой – применение помехоустойчивых кодов. Однако это приводит к снижению скорости и усложнению систем приема и передачи данных. Простым, но достаточно эффективным альтернативным методом очистки сигналов в таких условиях является двухэтапный алгоритм обработки сигналов $x(t)$, где на первом этапе производится устранение из потока $x(t)$ шумовых импульсов, а на втором – очистка сигнала частотными фильтрами от статистических шумов. Для сигналов, искаженных действием импульсных шумов, отсутствует строгая в математическом смысле постановка и решение задачи фильтрации. Известны лишь эвристические алгоритмы, наиболее приемлемым из которых является алгоритм медианной фильтрации.

Итерационные медианные фильтры выполняются последовательным повторением медианной фильтрации. Если апертура единичной медианной

фильтрации сохраняет перепады в сигнале, то они сохраняются при итеративном применении фильтра вплоть до тех пор, пока не прекратятся изменения в фильтруемом сигнале, при этом конечный результат существенно отличается от итеративного применения скользящего среднего, где в пределе получается постоянная числовая последовательность. При использовании итерационных фильтров можно изменять апертуру фильтра при каждом шаге итерации.

Достоинства медианных фильтров.

– Простая структура фильтра, как для аппаратной, так и для программной реализации.

– Фильтр не изменяет ступенчатые и пилообразные функции.

– Фильтр хорошо подавляет одиночные импульсные помехи и случайные шумовые выбросы отсчетов.

Недостатки медианных фильтров.

– Медианная фильтрация нелинейная, так как медиана суммы двух произвольных последовательностей не равна сумме их медиан, что в ряде случаев может усложнять математический анализ сигналов.

– Фильтр вызывает уплощение вершин треугольных функций.

– Подавление белого и гауссового шума менее эффективно, чем у линейных фильтров. Слабая эффективность наблюдается также при фильтрации флуктуационного шума.

– При увеличении размеров окна фильтра происходит размытие крутых изменений сигнала и скачков.

Недостатки метода можно уменьшить, если применять медианную фильтрацию с адаптивным изменением размера окна фильтра в зависимости от динамики сигнала и характера шумов (адаптивная медианная фильтрация).

В редакторе ресурсов проектируем интерфейс диалогового окна программы (рис. 2.13).

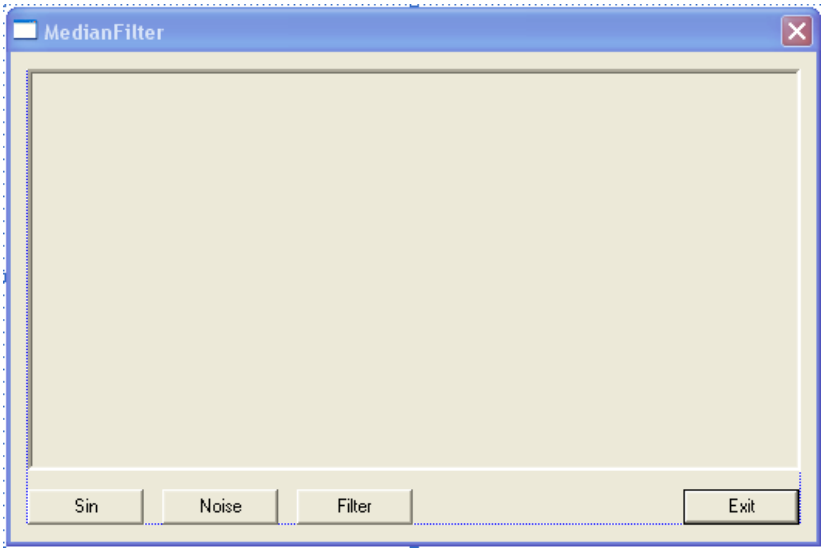


Рис. 2.13.

С элементом Static Text связываем переменную m_static. Для этого используется Class Wizard (рис. 2.14), окно которого можно вызвать нажатием сочетания клавиш Ctrl+W.

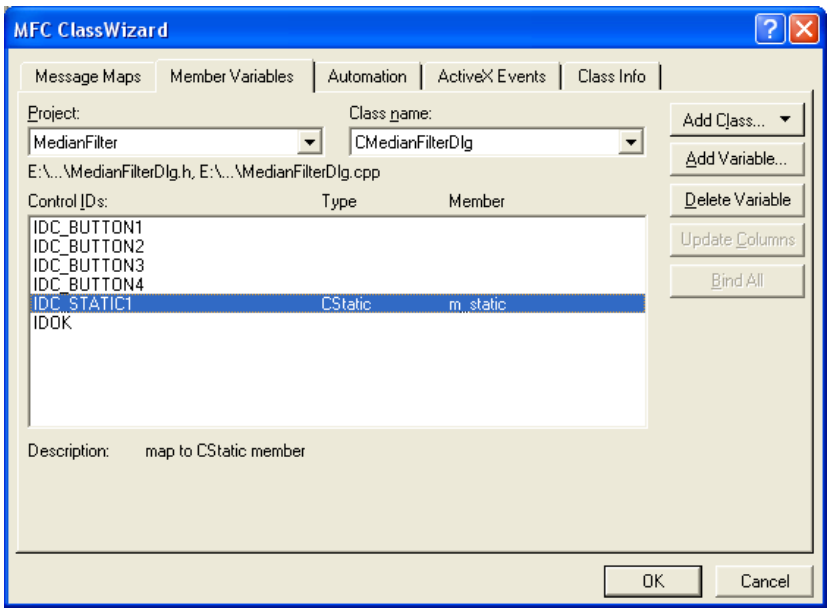


Рис. 2.14.

В класс диалога необходимо добавить две переменные-члены класса и функцию:

```
int m_size;
int * m_d;
void Draw();
```

В функцию OnInitDialog необходимо добавить следующий код:

```
CRect r;
m_static.GetClientRect(&r);
m_size=r.Width();
m_d=new int[m_size];
```

Реализация функции рисования имеет следующий вид:

```
void CMedianFilterDlg::Draw()
{
 CClientDC dc(&m_static);
 CPen p;
 p.CreatePen(PS_SOLID, 2, RGB(0,0,150));
 dc.SelectObject(&p);
 CBrush b;
 b.CreateSolidBrush(RGB(255,255,255));
 CRect r;
 m_static.GetClientRect(&r);
```

```
  dc.FillRect(&r, &b);
  for(int i=1; i<m_size; i++)
 {
        dc.MoveTo(i, r.Height()/2-m_d[i]);
        dc.LineTo(i-1, r.Height()/2-m_d[i-1]);
  }
}
```

Функция-обработчик нажатия на кнопку Sin, в которой выполняется формирование синусоидального сигнала и заполнение массива данных, представлена ниже.

```
void CMedianFilterDlg::OnButton1()
{
  for(int i=0; i<m_size; i++)
{
        m_d[i]=(int)50*sin(6.28*i/(m_size/4));
 }
  Draw();
}
```

Функция-обработчик нажатия на кнопку Noise, в которой выполняется генерация случайных импульсных помех и добавление этих помех в полезный сигнал имеет вид:

```
void CMedianFilterDlg::OnButton2()
{
  for(int i=0; i<10; i++)
{
        int d=rand()%(m_size-1);
        m_d[d]=m_d[d]+(rand()%80)*pow(-1, rand()%2);
  }
  Draw();
}
```

Функция-обработчик нажатия на кнопку Filter, в которой реализуется медианный фильтр, представлена ниже.

```
void CMedianFilterDlg::OnButton3()
{
  int t[3];
  for(int i=0; i<(m_size-3); i++)
{
        t[0]=m_d[i];
        t[1]=m_d[i+1];
        t[2]=m_d[i+2];
        if(t[0]>t[1]){
              int a=t[0];
```

```
                t[0]=t[1];
                t[1]=a;
        }
        if(t[1]>t[2]){
                int a=t[1];
                t[1]=t[2];
                t[2]=a;
        }
        if(t[0]>t[1]){
                int a=t[0];
                t[0]=t[1];
                t[1]=a;
        }
        m_d[i]=t[1];
  }
  Draw();
}
```

Функция-обработчик нажатия на кнопку Exit, выполняющая очистку памяти и закрытие диалогового окна программы, имеет вид:

```
void CMedianFilterDlg::OnOK()
{
  delete[]m_d;
  CDialog::OnOK();
}
```

На рис. 2.15 представлено окно программы. Нажатие на кнопку Sin приведет к генерированию синусоидального сигнала, осциллограмма которого отобразится на экране. Нажатие на кнопку Noise приведет к добавлению к сигналу импульсных помех в случайные моменты времени и случайной амплитудой.

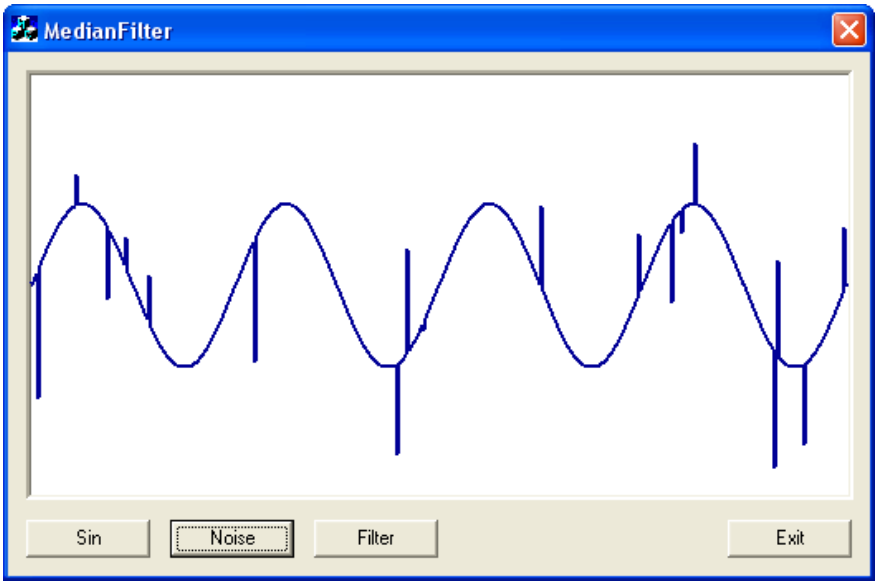


Рис. 2.15.

Нажатие на кнопку Filter приводит к вызову соответствующей функции-обработчика, в которой реализован алгоритм медианной фильтрации. На рис. 2.16 представлен вид диалогового окна с осциллограммой сигнала после фильтрации.

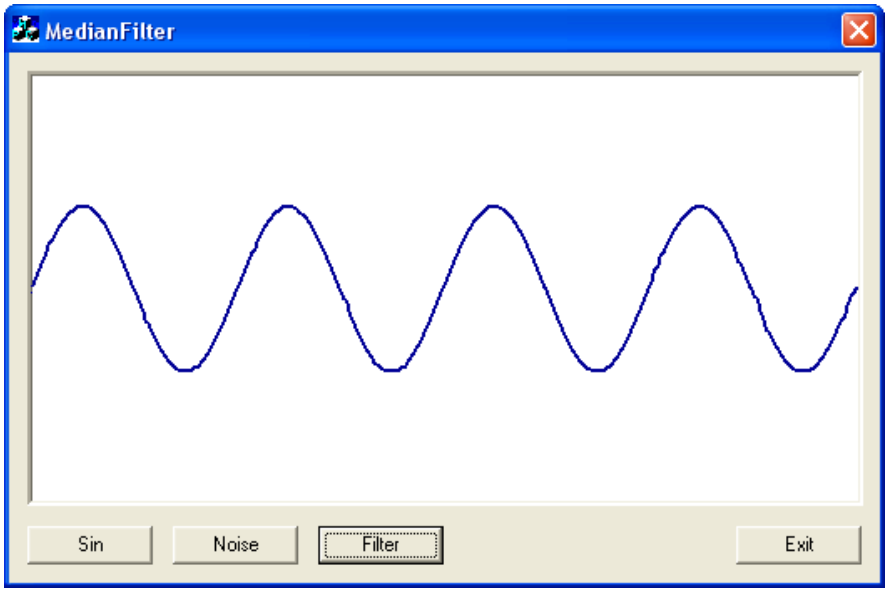


Рис. 2.16.

2.5. Программная реализация алгоритмов вычисления среднего и среднеквадратического значений периодических сигналов

Синтез современных микропроцессорных систем управления предполагает разработку алгоритмов обработки сигналов, снимаемых с первичных измерительных преобразователей (датчики напряжения, тока). При этом прямой перенос методов, отработанных в аналоговых системах управления, на цифровые системы порой оказывается неэффективным. В связи с этим актуальным является задача оптимизация процесса вычислений в дискретной системе.

В рамках указанного подхода может быть поставлена задача синтеза цифровой структуры для вычисления среднего и среднеквадратического значения детерминированного периодического сигнала при оптимальном быстродействии дискретной системы.

Арифметическое среднее значение переменного сигнала с периодом T рассчитывается по формуле:

$$u_{cp} = \frac{1}{T}\int_0^T |u(t)|dt\,.$$

Если кривая $u(t)$ симметрична относительно оси абсцисс, то:

$$u_{cp} = \frac{2}{T}\int_0^{T/2} |u(t)|dt\,.$$

Непрерывный входной сигнал может быть преобразован в последовательность дискретных значений, если с помощью элемента выборки-

хранения через равные интервалы времени $t_n = n \cdot T_s$ брать значения входного сигнала. Здесь, $f_s = \dfrac{1}{T_s}$ – частота выборки (дискретизации).

Тогда выражение для расчета среднего значения может быть представлено в дискретной форме следующим образом:

$$u_{cp}(t) = \frac{1}{N} \sum_{n=0}^{N-1} \left| u(t - nT_s) \right| ,$$

где $N = \dfrac{T}{2 \cdot T_s}$.

Введём в рассмотрение обозначения: $Y = u_{cp}$, $X = |u|$. Перепишем это уравнение, внеся константу $\dfrac{1}{N} = h$ под знак суммы и переходя к дискретному времени:

$$Y(kT_s) = \sum_{n=0}^{N-1} h \cdot X\left[(k-n)T_s\right].$$

Это уравнение описывает линейный фильтр с импульсной характеристикой конечной длительности $N-1$ порядка (рис. 2.17).

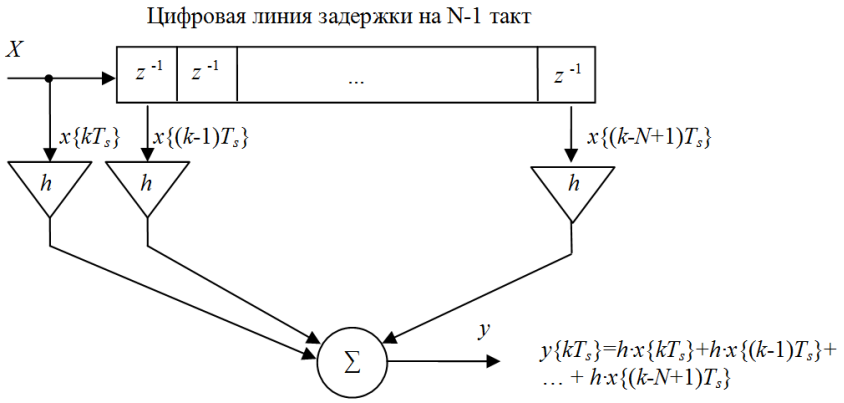


Рис. 2.17.

Создаем проект на основе диалогового приложения и в редакторе ресурсов проектируем внешний вид интерфейса пользователя, примерный вид которого показан на рис. 2.18.

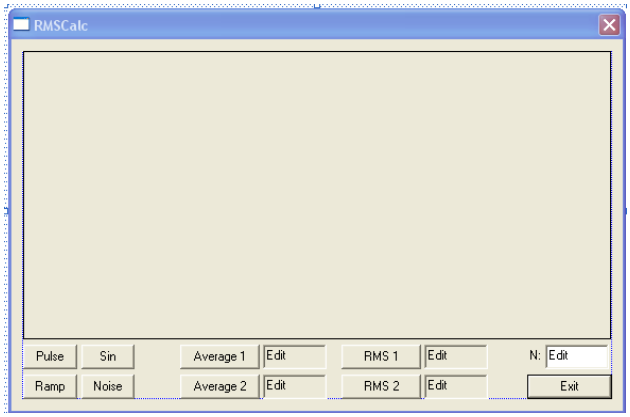


Рис. 2.18.

Назначение кнопок на диалоговом окне:

Pulse – формирование сигнала в виде прямоугольных импульсов;

Sin – формирование синусоидального сигнала;

Ramp – формирование сигнала в пилообразной формы (треугольные импульсы);

Noise – формирование белого шума;

Average 1 – расчет среднего значения по неоптимизированному алгоритму (рис. 2.17);

RMS 1 – расчет среднеквадратического значения по неоптимизированному алгоритму;

Average 2 – расчет среднего значения по оптимизированному алгоритму (рис. 2.21);

RMS 2 – расчет среднеквадратического значения по оптимизированному алгоритму (рис. 2.22).

Элементы Edit используются для отображения числовых значений рассчитаных величин. Элемент Edit напротив текстовой надписи N: используется для ввода длины линии задержки. Также на диалоге расположен элемент Static Text (выделен рамкой), на котором выполняется построение осциллограмм сигналов. С помощью Class Wizard, окно которого можно вызвать нажатием сочетания клавиш Ctrl+W, с элементами управления необходимо связать переменные, как показано на рис. 2.19.

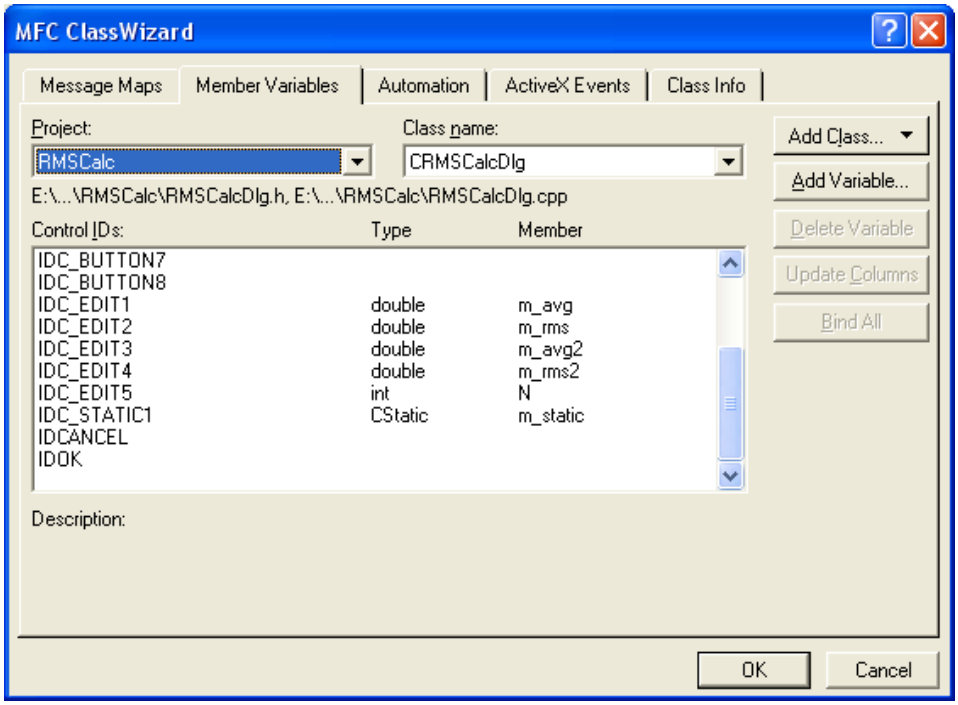


Рис. 2.19.

В класс диалога добавляем 3 переменные и 2 функции:

```
int * m_res;      // указатель на массив для хранения результатов расчета
int m_N;          // количество элементов в массиве выборок
```

```
int * m_d;          // указатель на массив с исходными данными
void DrawSignal();      // функция рисования осциллограммы сигнала
void DrawResult(COLORREF col); // функция отображения рассчитанных значений
```

В функции OnInitDialog() выполняется динамическое выделение памяти для хранения выборок сигнала и рассчитанных значений, а также инициализация пременных:

```
CRect r; // переменная для хранения координат прямоугольной области
m_static.GetClientRect(&r); // получение координат области построения
осциллограммы
m_N=r.Width();          // количество элементов в массиве равно ширине области
построения осциллограммы
m_d=new int [m_N];    // выделение памяти для хранения масисва выборок сигнала
m_res=new int[m_N];  // выделение памяти для хранения рассчитанных значений
N=100;                  // начальное значение длины линии задержки
UpdateData(false);
memset(m_d, 0, sizeof(int)*m_N);      // обнуление всех элементов массива m_d
memset(m_res, 0, sizeof(int)*m_N);   // обнуление всех элементов массива m_res
```

В функции нажатия на кнопку Exit следует освободить ранее выделенную память:

```
void CRMSCalcDlg::OnOK()
{
  delete [] m_d;
  delete [] m_res;
  CDialog::OnOK();
}
```

Реализация функции DrawSignal() имеет вид:

```
void CRMSCalcDlg::DrawSignal()
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  dc.MoveTo(0, r.Height()/2);
  dc.LineTo(m_N, r.Height()/2);
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(200,0,0));
  op=dc.SelectObject(&p);
  dc.MoveTo(0,r.Height()/2);
  for(int i=0;i<(m_N-1);i++)
  {
          dc.LineTo(i,r.Height()/2-m_d[i]);
  }
  dc.SelectObject(op);
}
```

Реализация функции DrawResult(COLORREF col) представлена ниже.

```
void CRMSCalcDlg::DrawResult(COLORREF col)
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  dc.MoveTo(0, r.Height()/2);
  dc.LineTo(m_N, r.Height()/2);
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,col);
  op=dc.SelectObject(&p);
  dc.MoveTo(0,r.Height()/2);
  for(int i=0;i<(m_N-1);i++)
  {
          dc.LineTo(i,r.Height()/2-m_res[i]);
  }
  dc.SelectObject(op);
}
```

Обработчик нажатия на кнопку Pulse:

```
void CRMSCalcDlg::OnButton2()
{
  bool t=false;
  for(int i=0;i<m_N;i++)
  {
          if(t==false) m_d[i]=0;
          else m_d[i]=100;
          t=(i%50==0)?!t:t;
  }
  DrawSignal();
}
```

Обработчик нажатия на кнопку Sin:

```
void CRMSCalcDlg::OnButton3()
{
  for(int i=0;i<m_N;i++)
  {
          m_d[i]=100*sin(6.28*i/100);
  }
  DrawSignal();
}
```

Обработчик нажатия на кнопку Ramp:

```
void CRMSCalcDlg::OnButton6()
```

```
{
  for(int i=0;i<m_N;i++)
  {
        m_d[i]=i%101;
  }
  DrawSignal();
}
```

Обработчик нажатия на кнопку Noise:

```
void CRMSCalcDlg::OnButton7()
{
  for(int i=0;i<m_N;i++)
  {
        m_d[i]=rand()%101-50;
  }
  DrawSignal();
}
```

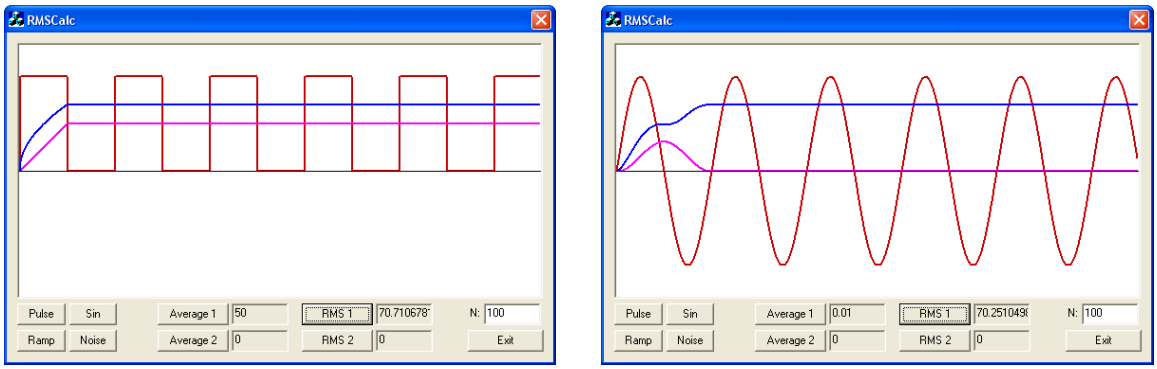
Обработчик нажатия на кнопку Average 1:

```
void CRMSCalcDlg::OnButton1()
{
  UpdateData(true);
  int S=0;
  for(int i=0;i<m_N;i++)
  {
        if(i<N)
        {
              for(int k=0;k<i;k++)
              {
                    S=S+m_d[i-k];
              }
        }
        else
        {
              for(int k=0;k<N;k++)
              {
                    S=S+m_d[i-k];
              }

        }
              m_res[i]=(int)S/N;
              m_avg=(double)S/N;
              S=0;
  }
  UpdateData(false);
  DrawResult(RGB(255,0,255));
}
```

Обработчик нажатия на кнопку RMS 1:

```
void CRMSCalcDlg::OnButton4()
{
  UpdateData(true);
  int S=0;
  for(int i=0;i<m_N;i++)
  {
          if(i<N)
          {
                  for(int k=0;k<i;k++)
                  {
                          S=S+m_d[i-k]*m_d[i-k];
                  }
          }
          else
          {
                  for(int k=0;k<N;k++)
                  {
                          S=S+m_d[i-k]*m_d[i-k];
                  }

          }
                  m_res[i]=(int)sqrt((double)S/N);
                  m_rms=(double)sqrt((double)S/N);
                  S=0;
  }
  UpdateData(false);
  DrawResult(RGB(0,0,255));
}
```

После компилаяции проекта появится диалоговое окно приложения. Нажатие на одну из кнопок (Pulse, Sin, Ramp, Noise) приведет к заполнению масива данных и отображению осциллограммы. Для выбранного сигнала можно рассчитать среднее и действующее значения. При этом на графике имеется возможность отобразить сразу обе величины (верхняя линия – среднеквадратическое значение, нижняя линия – среднее значение). На рис. 2.20, а, представлено окно программы с рассчитанными значениями для прямоугольных импульсов, а на рис. 2.20, б – для синусоидального сигнала. Период сигнала в обоих случаях составляет 100 выборок. Длина линии задержки также равна 100.



(а) (б)

Рис. 2.20.

Запишем выражение расчета среднего значения в развёрнутом виде:

$$y(kT_S) = h\left[x\{kT_S\} + x\{(k-1)T_S\} + \ldots + x\{(k-N+1)T_S\}\right].$$

Для следующего отсчета времени $kT_S + 1$ справедливым будет выражение:

$$y(kT_S + 1) = h\left[x\{kT_S + 1\} + x\{kT_S\} + \ldots + x\{(k-N)T_S\}\right].$$

Объединяя оба выражения, получим уравнение для выходного отсчета $y(kT_S + 1)$, выраженное через его предыдущее значение $y(kT_S)$:

$$y(kT_S + 1) = y(kT_S) + h \cdot x\{kT_S + 1\} - h \cdot x\{(k-N+1)T_S\}.$$

Используя приведенное выражение можно предложить оптимизированную структуру трансверсального цифрового фильтра с равными весовыми коэффициентами (рис. 2.21).

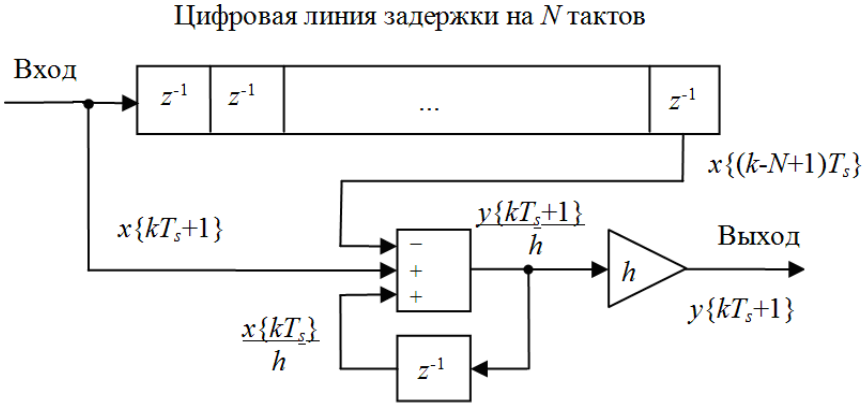


Рис. 2.21.

Итак, для вычисления очередного выходного отсчёта оптимизированного цифрового фильтра необходимо выполнить 3 операции сложения и одну операцию умножения, против N операций сложения и N операций умножения. Платой за снижение арифметических вычислений является увеличение длины цифровой линии задержки на один элемент памяти и выделение дополнительного элемента памяти для хранения промежуточного результата.

Введя дополнительные функциональные блоки A^2 и $\sqrt{A}$ (рис. 2.22) оптимизированную структуру цифрового фильтра с равными весовыми

коэффициентами можно использовать для вычисления действующего (среднеквадратического) значения входного сигнала.

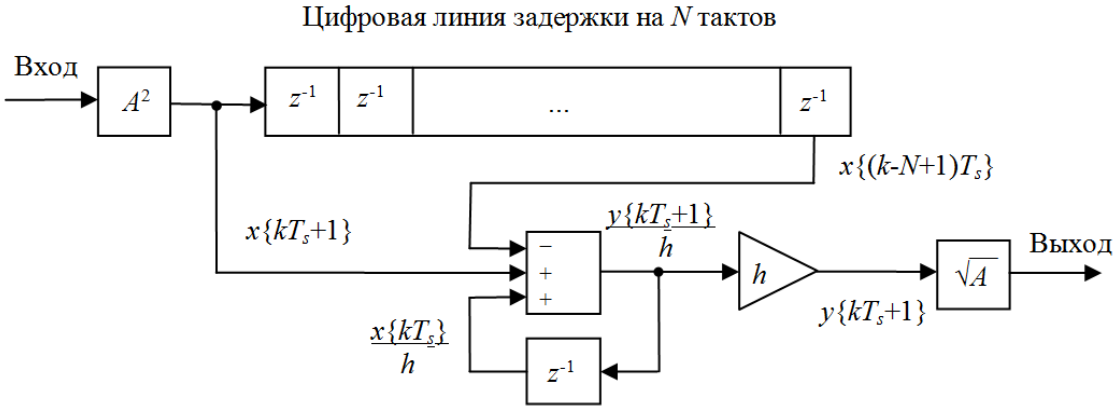


Рис. 2.22.

Функция-обработчик на кнопку Average 2, в котором реализован оптимизированный алгоритм расчета среднего значения, представлена ниже.

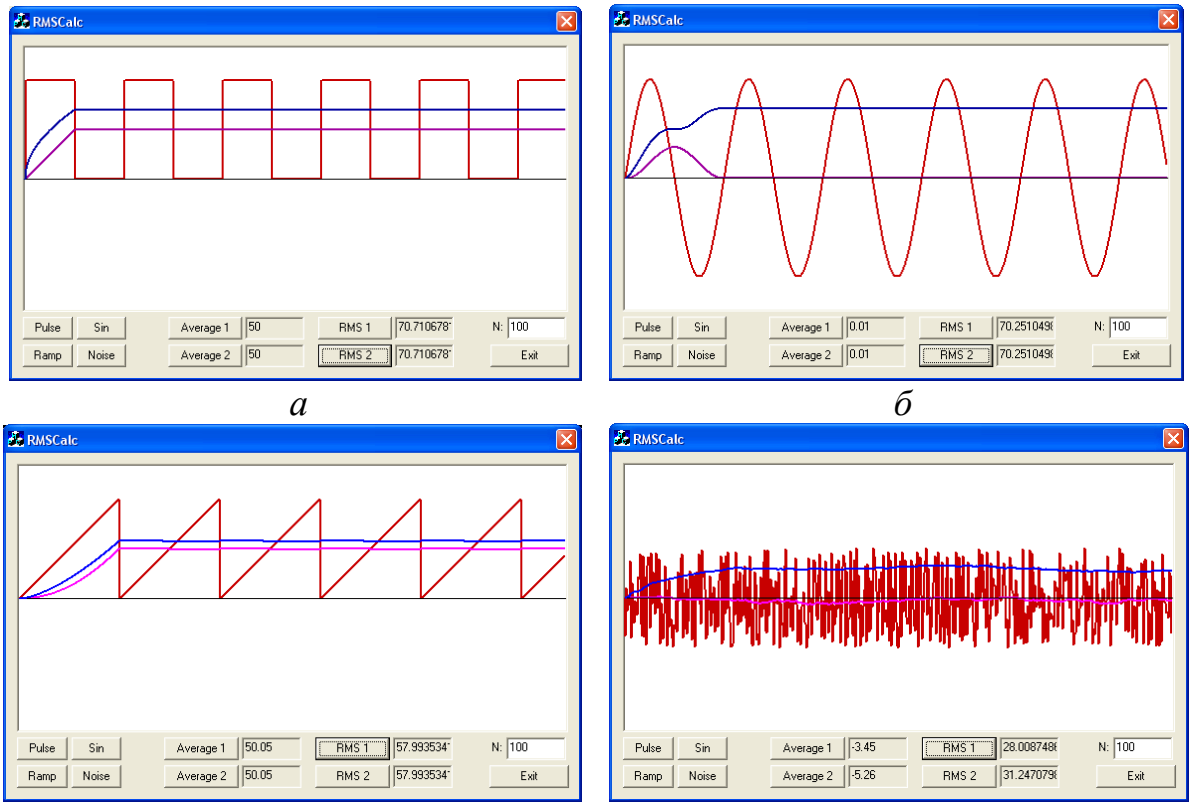
```
void CRMSCalcDlg::OnButton5()
{
  UpdateData(true);
  int S=0;
  for(int i=0;i<m_N;i++)
  {
          if(i<N)
          {
                  S=S+m_d[i];
          }
          else
          {
                  S=S+m_d[i]-m_d[i-N];
          }
                  m_res[i]=(int)S/N;
                  m_avg2=(double)S/N;
  }
  UpdateData(false);
  DrawResult(RGB(160,0,160));
}
```

Функция-обработчик на кнопку RMS 2, в котором реализован оптимизированный алгоритм расчета среднеквадратического значения, имеет вид:

```
void CRMSCalcDlg::OnButton8()
{
  UpdateData(true);
```

```
int S=0;
int t;
for(int i=0;i<m_N;i++)
{
        t=m_d[i]*m_d[i];
        if(i<N)
        {
                S=S+t;
        }
        else
        {
                S=S+t-m_d[i-N]*m_d[i-N];
        }
                m_res[i]=(int)sqrt((double)S/N);
                m_rms2=(double)sqrt((double)S/N);
}
UpdateData(false);
DrawResult(RGB(0,0,160));
}
```

На рис. 2.23, а-г, представлены результаты выполнения программы для различных сигналов. Анализ результатов позволяет сделать вывод о правильности работы оптимизированных алгоритмов.



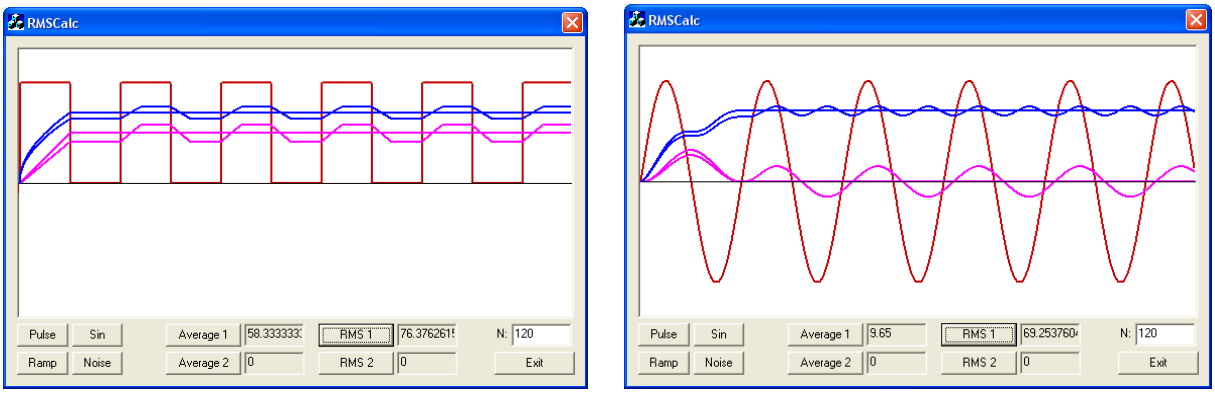
а *б*

в *г*

Рис. 2.23.

Время реакции предложенных алгоритмов составляет T_s (период дискретизации), при этом увеличение частоты дискретизации в оптимизированных структурах фильтров не приводит к возрастанию количества арифметических операций сложения и умножения.

Если длина линии задержки не совпадает с количеством выборок, приходящихся на один период периодических сигналов, рассчитанные значения среднего и среднеквадратического будут меняться в течение времени. Рис. 2.24 демонстрирует это.



а *б*

Рис. 2.24.

Задачи для самостоятельного решения

1. Написать программу для компьютера, которая выполняет генерацию 128 случайных чисел. Определить значения статистических характеристик сигналов (мат. ожидания, дисперсии, среднеквадратичного отклонения). Построить гистограмму (на графическом ЖКИ или компьютере). Сделать выводы о форме гистограммы.

2. Написать программу, которая выполняет генерацию 128 случайных чисел. В табл. 2.2 приводятся значения математического ожидания и среднеквадратичного отклонения для сгенерированного сигнала (по вариантам).

Табл. 2.2.

№	μ	σ
1	4	2
2	8	5
3	12	4
4	16	5

3. Построить гистограмму для сгенерированного случайного сигнала. Какой вид имеет функция распределения вероятности? Вычислить вероятность появления в числовой последовательности значения, равного математическому ожиданию.

4. Разработать структуру и выполнить программную реализацию цифрового фильтра, выполняющего расчет взвешенного скользящего среднего (weighted moving average – WMA) – скользящее среднее, при вычислении которого вес каждого члена исходной функции, начиная с меньшего, равен соответствующему члену арифметической прогрессии. То есть, при вычислении WMA для временного ряда, мы считаем последние значения исходной функции более значимы чем предыдущие, причём функция значимости линейно убывающая. Например, для арифметической прогрессии с начальным значением и шагом, равным 1, формула вычисления скользящей средней примет вид:

$$WMA_t = \frac{n \cdot p_t + (n-1) \cdot p_{t-1} + \cdots + (n-i) \cdot p_{t-i} + \cdots + 2 \cdot p_{t-n} + 1 \cdot p_{t-n+1}}{n + (n-1) + \cdots + (n-i) + \cdots + 2 + 1} = \frac{2}{n \cdot (n+1)} \sum_{i=0}^{n-1} (n-i) \cdot p_{t-i},$$

где WMA_t – значение взвешенного скользящего среднего в точке t, n – количество значений исходной функции для расчёта скользящего среднего, p_{t-i} – значение исходной функции в момент времени, отдалённый от текущего на i интервалов. На рис. 2.25 представлены веса значений исходной функции при вычислении WMA с n=15.

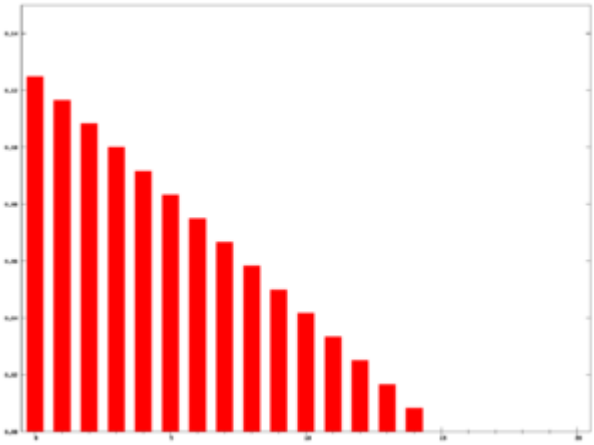


Рис. 2.25.

5. Разработать структуру и выполнить программную реализацию цифрового фильтра, выполняющего расчет экспоненциального скользящего среднего (exponential moving average – EMA) – разновидность взвешенной скользящей средней, веса которой убывают экспоненциально и никогда не равны нулю (рис. 2.26). Определяется следующей формулой:

$$EMA_t = \alpha p_t + (1-\alpha) EMA_{t-1},$$

$$\alpha = \frac{2}{n+1},$$

$$EMA_0 = p_0,$$

где EMA_t – значение экспоненциального скользящего среднего в точке t (последнее значение, в случае временного ряда), EMA_{t-1} – значение экспоненциального скользящего среднего в точке t–1 (предыдущее значение в

случае временного ряда), p_t – значение исходной функции в момент времени t (последнее значение, в случае временного ряда), α (сглаживающая константа) – коэффициент характеризующий скорость уменьшения весов, принимает значение от 0 и до 1, чем меньше его значение тем больше влияние предыдущих значений на текущую величину среднего.

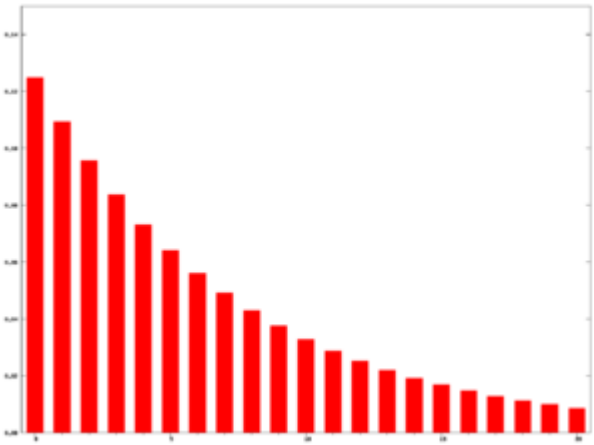


Рис. 2.26.

6. Разработать структуру и выполнить программную реализацию цифрового фильтра, выполняющего расчет среднего геометрического. Средним геометрическим n чисел является корень n-ой степени из произведения этих чисел:

$$A_0(x_1, \ldots, x_n) = g = \sqrt[n]{x_1 x_2 \cdots x_n}.$$

7. Разработать структуру и выполнить программную реализацию цифрового фильтра, выполняющего расчет среднего гармонического. Средним гармоническим чисел является обратная величина к среднему арифметическому их обратных:

$$A_{-1}(x_1, \ldots, x_n) = h = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}.$$

8. Исследовать работу медианного фильтра при наличии в сигнале аддитивной помехи в виде белого шума.

9. Разработать программу, реализующую медианную фильтрацию цифровым фильтром с апертурой окна, равной 5. Исследовать работу фильтра при наличии импульсной помехи в одной выборе и в дух выборках.

10. Разработать программу, в которой предусмотреть возможность выбора пользователем апертуры окна. Смоделировать импульсную помеху и проверить работу разработанного фильтра.

3. СПЕКТРАЛЬНЫЙ И КОРРЕЛЯЦИОННЫЙ АНАЛИЗ СИГНАЛОВ

3.1. Исследование спектральных характеристик сигналов

Современную технику связи невозможно представить без спектрального анализа. Представление сигналов в частотной области необходимо как для анализа их характеристик, так и для анализа блоков и узлов приемопередатчиков систем радиосвязи. Цифровой спектральный анализ используется:

– в анализаторах спектра;
– при обработке речи;
– при обработке изображений;
– при распознавании образов.
– при проектировании цифровых фильтров;
– для вычисления импульсной характеристики по частотной;
– для вычисления частотной характеристики по импульсной.

Для преобразования сигналов в частотную область применяется прямое преобразование Фурье. Обобщенная формула прямого преобразования Фурье записывается следующим образом:

$$S(f) = \int_{-\infty}^{\infty} e^{-i2\pi ft} x(t)dt .$$

Как видно из этой формулы для частотного анализа производится вычисление корреляционной зависимости между сигналом, представленным во временной области и комплексной экспонентой с заданной частотой. При этом по формуле Эйлера комплексная экспонента разлагается на реальную и мнимую часть:

$$e^{-i2\pi f \cdot t} = \cos(2\pi \cdot f \cdot t) - i\sin(2\pi \cdot f \cdot t).$$

В формуле прямого преобразования Фурье используется интегрирование по времени от минус бесконечности до бесконечности. Естественно это является математической абстракцией. В реальных условиях мы можем провести интегрирование от данного момента времени, который мы можем обозначить за 0, до момента времени T. Формула прямого преобразования Фурье при этом будет преобразована к следующему виду:

$$S(f) = \int_{0}^{T} e^{-i2\pi ft} x(t)dt .$$

В результате существенно меняются свойства преобразования Фурье. Спектр сигнала вместо непрерывной функции становится дискретным рядом значений. Теперь минимальной частотой и одновременно шагом частотных значений спектра сигнала становится:

$$f_{\min} = \Delta f = \frac{1}{T},$$

$$\Omega = \frac{2\pi}{T}.$$

Только функции sin и cos с частотами k/T будут взаимно ортогональны, а это является непременным условием преобразования Фурье.

Спектр сигнала $x(t)$ при анализе на ограниченном интервале времени будет выглядеть так, как это показано на рис. 3.1.

Рис. 3.1.

В данном случае формула вычисления прямого преобразования Фурье преобразуется к следующему виду:

$$X(k) = \int_0^T e^{-i\frac{2\pi k}{T}t} x(t)dt$$

Подобным образом можно определить формулу прямого преобразования Фурье для цифровых отсчетов сигнала. Учитывая, что вместо непрерывного сигнала используются его цифровые отсчеты, а интеграл заменяется на сумму. В данном случае длительность анализируемого сигнала определяется количеством цифровых отсчетов N. Преобразование Фурье для цифровых отсчетов сигнала называется дискретным преобразованием Фурье (ДПФ) и записывается следующим образом:

$$X(k) = \sum_{n=0}^{N-1} e^{-i\frac{2\pi}{T}kn} x(n) = \sum_{n=0}^{N-1} \left[\cos\left(\frac{2\pi}{T} kn\right) - i \sin\left(\frac{2\pi}{T} kn\right) \right] x(n)$$

По отношению к этому уравнению следует сделать некоторые терминологические разъяснения. $X(k)$ 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.

Значение $x(n)$ 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.

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).

Дискретное преобразование Фурье 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, как показано на рис. 3.2.

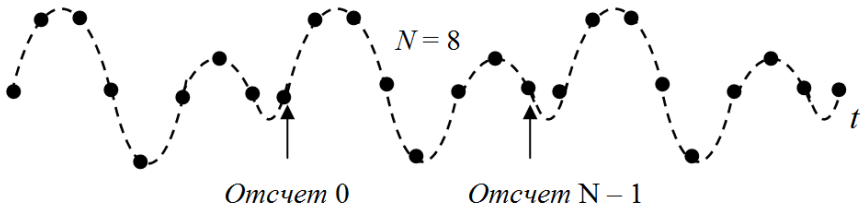


Рис. 3.2.

Существует два основных типа ДПФ: вещественное ДПФ и комплексное ДПФ.

На рис. 3.3 представлены уравнения комплексного и вещественного ДПФ.

Комплексное преобразование	Вещественное преобразование
$X(k)=\frac{1}{N}\sum_{n=0}^{N-1}x(n)e^{\frac{-j2\pi nk}{N}}$ $x(n)=\sum_{k=0}^{N-1}X(k)e^{\frac{j2\pi nk}{N}}$	$\mathrm{Re}\,X(k)=\frac{2}{N}\sum_{n=0}^{N-1}x(n)\cos\left(\frac{2\pi nk}{N}\right)$ $\mathrm{Im}\,X(k)=-\frac{2}{N}\sum_{n=0}^{N-1}x(n)\sin\left(\frac{2\pi nk}{N}\right)$ $x(n)=\sum_{k=0}^{N/2}\left[\mathrm{Re}\,X(k)\cos\left(\frac{2\pi nk}{N}\right)-\mathrm{Im}\,X(k)\sin\left(\frac{2\pi nk}{N}\right)\right]$
Временная область: $x(n)$ является комплексной, дискретной и периодической величиной. Частотная область: $X(k)$ является комплексной, дискретной и периодической величиной. k изменяется в диапазоне от 0 до N-1. Для k от 0 до $N/2$ – положительные частоты. Для k от $N/2$ до N-1 – отрицательные частоты.	Временная область: $x(n)$ является вещественной, дискретной и периодической величиной. n изменяется в диапазоне от 0 до N-1. Частотная область: $ReX(k)$, $ImX(k)$ являются вещественными, дискретными и периодическими величинами. k изменяется в диапазоне от 0 до $N/2$. Перед использованием уравнения x для вычисления (n) значения $ReX(0)$ и $ReX(N/2)$ должны быть поделены на два.

Рис. 3.3.

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 ($ReX(k)$) и мнимую ($ImX(k)$) компоненты, которые могут быть преобразованы в амплитуду и фазу.

Вещественное ДПФ выглядит несколько проще и, в основном, является упрощением 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.

Выходной спектр ДПФ может быть представлен либо в полярной системе координат (амплитудой и фазой), либо в алгебраической форме (вещественной и мнимой частями), как показано на рис. 3.4. Обе указанных формы находятся во взаимно однозначном соответствии.

$$X(k) = ReX(k) + j\, ImX(k)$$

$$MAG\,[X(k)] = \sqrt{ReX(k)^2 + ImX(k)^2}$$

$$\varphi\,[X(k)] = tan^{-1} \cdot ImX(k)/ReX(k)$$

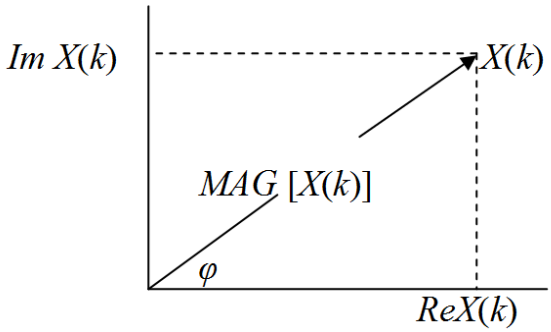


Рис. 3.4.

3.2. Реализация алгоритма дискретного преобразования Фурье

В редакторе ресурсов среды Visual Studio 6.0 проектируем внешний вид диалогового окна (рис. 3.5). На диалоговом окне расположен элемент управления Static Text, на котором выполняется построение осциллограммы сигнала и спектра, кнопка Signal, нажатие на которую приведет к запуску алгоритма генерирования сигнала, кнопка Spectrum, нажатие на которую приведет к запуску алгоритма ДПФ, и элемент Slider, перемещение ползунка которого изменит скважность прямоугольных импульсов и перерасчету спектра.

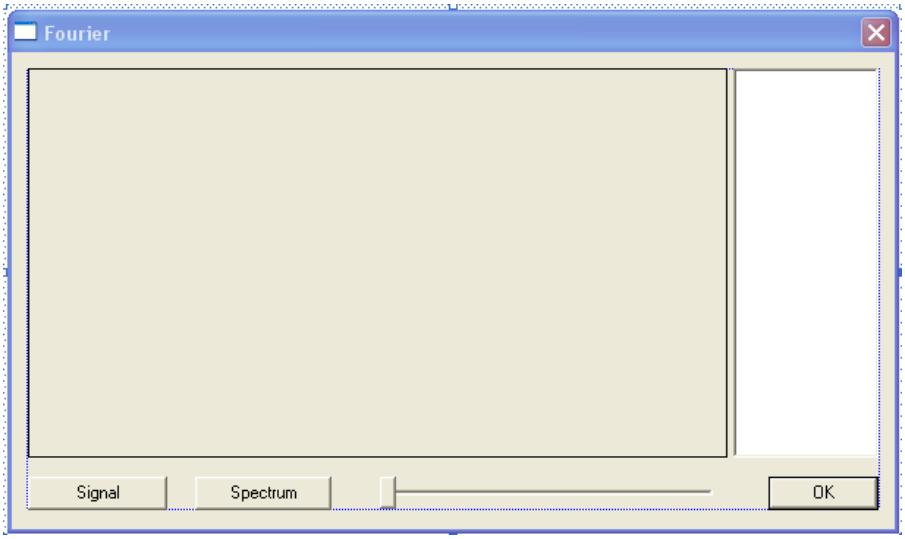


Рис. 3.5.

С элементом управления Static Text связываем переменную m_static, воспользовавшись окном ClassWizard (закладка Member Variables). С остальными переменными также необходимо связать переменные, как показано на рис. 3.6.

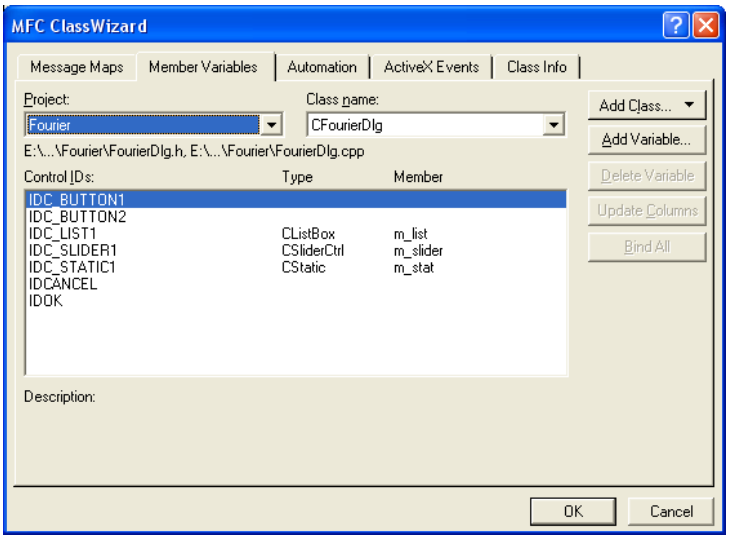


Рис. 3.6.

В класс диалога добавляем переменные-члены класса:

```
int m_H;
int m_W;
int * m_d;
```

В функции OnInitDialog() выполняем инициализацию переменных, выделение памяти для хранения данных, настройку диапазона изменения элемента управления Slider (ползунок или линейный регулятор):

```
  CRect rc;
  m_stat.GetClientRect(&rc);
  m_W=rc.Width();
  m_H=rc.Height();
  m_d=new int[m_W];
  m_slider.SetRange(0,64);
```

Обработчик нажатия на кнопку Signal имеет вид:

```
void CFourierDlg::OnButton2()
{
  int i=0;
  int y=0;
   for(i=0;i<64;i=i+1)
   {
    if(i<m_slider.GetPos()) m_d[i]=0;
    else m_d[i]=100;
   }
  CClientDC cdc(&m_stat);
  CPen p;
  p.CreatePen(PS_SOLID,2,RGB(200,0,0));
  cdc.SelectObject(&p);
  while(y<m_W)
  {
  for(i=1;i<64;i++)
  {
            cdc.MoveTo(y,m_H/2-m_d[i]);
            cdc.LineTo(y-1,m_H/2-m_d[i-1]);
            y++;
            if(y>(m_W)) break;
  }
            if(y>(m_W)) break;
            cdc.LineTo(y,m_H/2-m_d[0]);
  }
}
```

Обработчик нажатия на кнопку Spectrum представлен ниже.

```
void CFourierDlg::OnButton1()
{
   int k, n;
   float a1, a2;
   int amp=0;
  int spectrum[32];
for(k=0;k<32;k=k+1)
  {
  for(n=0;n<64;n=n+1)
  {
   a1=a1+m_d[n]*cos(6.28*n*k/64);
```

```
   a2=a2+m_d[n]*sin(6.28*n*k/64);
  }
  a1=a1/32;
  a2=a2/32;
  amp=sqrt(a1*a1+a2*a2);
  spectrum[k]=amp;
  a1=0;
  a2=0;
  }
  m_list.ResetContent();
  CString str;
  CClientDC cdc(&m_stat);
  for(k=1;k<32;k++)
  {
cdc.FillRect(CRect(10*k,m_H,10*k+3,m_H-spectrum[k]),&CBrush(RGB(0,0,200)));
          str.Format("U[%d]=%d B",k, spectrum[k]);
          m_list.AddString(str);
  }
}
```

В обработчике нажатия на кнопку OK необходимо очистить выделенную ранее память.

```
void CFourierDlg::OnOK()
{
  delete [] m_d;
  CDialog::OnOK();
}
```

Для удобства исследования спектра сигнала в виде прямоугольных импульсов добавим обработчик изменения положения ползунка регулятора, в котором симитируем нажатия сначала на кнопку Signal, а затем на кнопку Spectrum:

```
void CFourierDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  m_stat.RedrawWindow();
  OnButton2();
  OnButton1();
  CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

После компиляции проекта можно проверить работу программы. На рис. 3.7 представлено окно программы с вычисленным спектром прямоугольных импульсов. Для генерации импульсов использован следующий код:

```
if(i<m_slider.GetPos()) m_d[i]=0;
else m_d[i]=100;
```

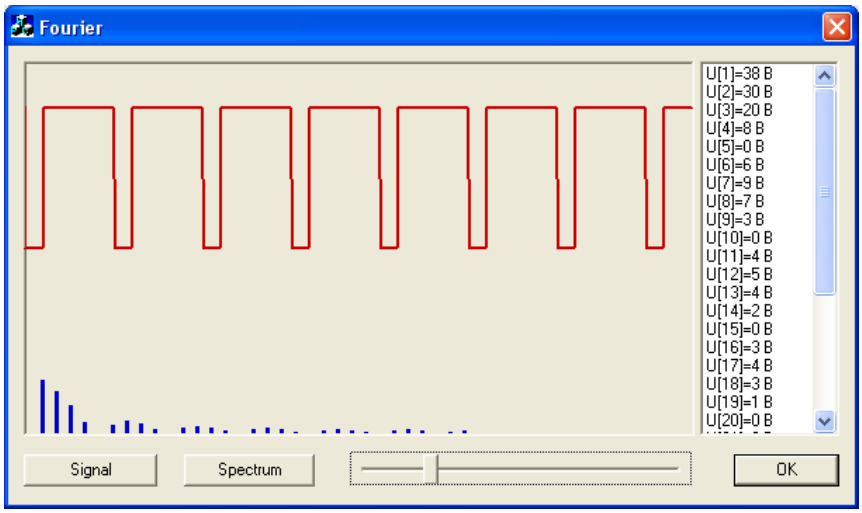


Рис. 3.7.

На рис. 3.8 представлена осциллограмма импульсов и спектр сигнала при скважности равной 0,5.

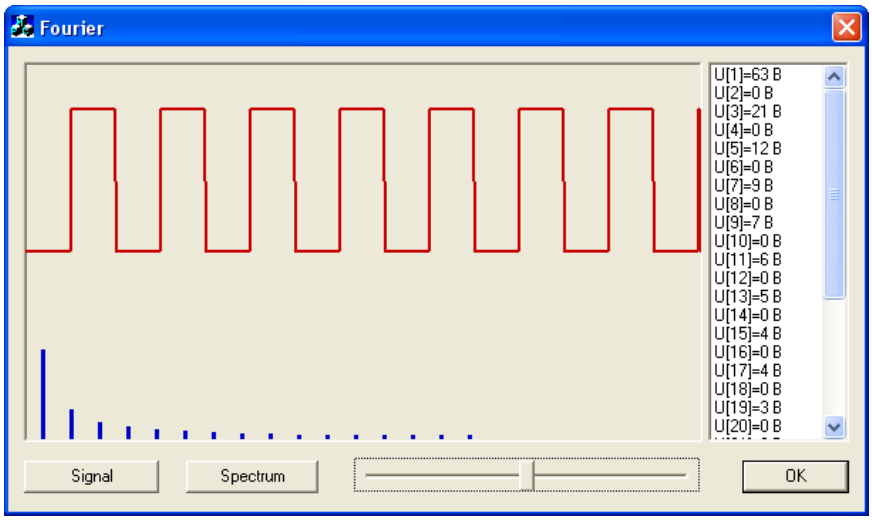


Рис. 3.8.

На рис. 3.9 представлена осциллограмма пилообразного сигнала и его спектр. Для генерирования пилообразного сигнала использован следующий код:

```
if(i<m_slider.GetPos()) m_d[i]=2*i;
else m_d[i]=2*(64-i);
```

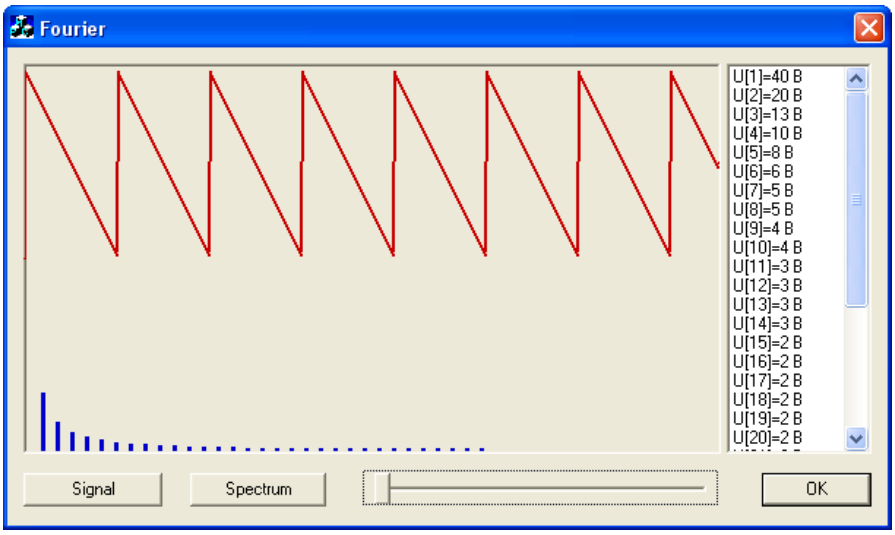


Рис. 3.9.

На рис. 3.10 представлена осциллограмма случайного сигнала и его спектр. Для генерирования случайных значений использован следующий код:

```
m_d[i]=rand()%100;
```

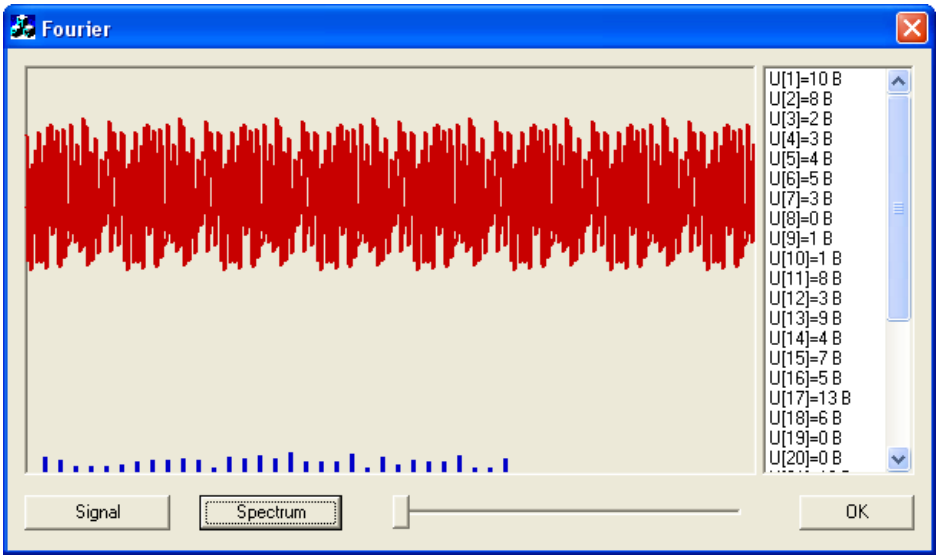


Рис. 3.10.

На рис. 3.11 представлена осциллограмма синусоидального сигнала и его спектр. Для генерирования пилообразного сигнала использован следующий код:

```
m_d[i]=50*sin(6.28*i/16);
```

На интервале наблюдения, который составляет 64 точки, укладывается целое число периодов синусоиды (4 периода). Поэтому на границах интервала наблюдения отсутствуют разрывы, и в спектре сигнала присутствует только

одна (четвертая) гармоника, рассчитанная амплитуда которой равна 49 условных единиц. В исходном сигнале амплитуда составляла 50 единиц. Разница значений обусловлена наличием ошибок округления. При использовании тригонометрических функций результат имеет тип данных с плавающей запятой float (double). Полученные в реальных системах значения с АЦП представляют собой целые числа. В разработанной программе массив для хранения спектральных составляющих имеет целочисленный тип int, поэтому при расчетах произошло округление данных, что вызвало небольшие различия в исходной амплитуде сигнала и полученной в результате ДПФ.

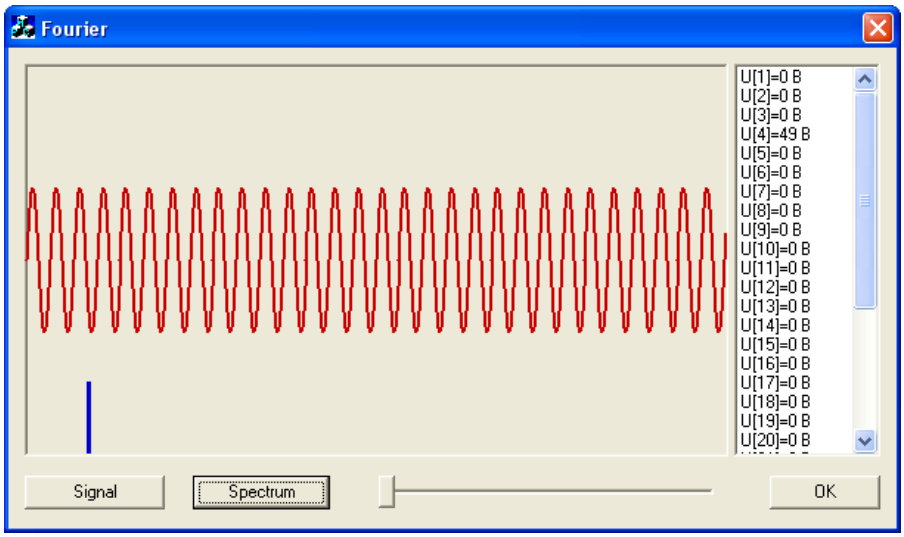


Рис. 3.11.

3.3. Использование цифровых окон для уменьшения растекания спектра

Чрезвычайно важным обстоятельством является то, что ДПФ позволяет вычислить спектр сигнала лишь на определенных частотах, кратных $\frac{2\pi}{NT}$, где T – интервал дискретизации, N – число отсчетов сигнала. Выясним, как меняется ДПФ, если в сигнале присутствует составляющая с частотой, не кратной $\frac{2\pi}{NT}$.

Пусть непрерывный сигнал задан следующим образом:

$$x(t) = \cos(\omega t).$$

Дискретизируем его с интервалом T, взяв N отсчетов сигнала, так что интервал наблюдения $T_н = NT$. При этом зададим частоту сигнала $\omega = \frac{2\pi}{NT}(k_0 + \alpha)$, где k_0 – целое число, $0 \le \alpha < 1$, то есть α задает отклонение от частоты, кратной $\frac{2\pi}{NT}$. Тогда

$$x(n) = x(nT) = \cos(\frac{2\pi}{NT}(k_0 + \alpha)nT) = \cos(\frac{2\pi}{N}k_0 n + \frac{2\pi}{N}\alpha n)$$

Вычислим теперь ДПФ этого сигнала, заменив для удобства расчетов косинус на сумму экспонент по формуле Эйлера:

$$\cos(\frac{2\pi}{N}k_0 n+\frac{2\pi}{N}\alpha n)=\frac{e^{i(\frac{2\pi}{N}k_0 n+\frac{2\pi}{N}\alpha n)}+e^{-i(\frac{2\pi}{N}k_0 n+\frac{2\pi}{N}\alpha n)}}{2}.$$

Тогда

$$\widetilde{X}(k)=\frac{1}{2}\sum_{n=0}^{N-1}e^{i(\frac{2\pi}{N}k_0 n+\frac{2\pi}{N}\alpha n)-i\frac{2\pi}{N}kn}+\frac{1}{2}\sum_{n=0}^{N-1}e^{-i(\frac{2\pi}{N}k_0 n+\frac{2\pi}{N}\alpha n)-i\frac{2\pi}{N}kn}.$$

Рассмотрим сначала случай, когда $\alpha=0$.

$$\widetilde{X}(k)=\frac{1}{2}\sum_{n=0}^{N-1}e^{i\frac{2\pi}{N}(k_0-k)n}+\frac{1}{2}\sum_{n=0}^{N-1}e^{-i\frac{2\pi}{N}(k_0+k)n}.$$

Первая сумма из этого равенства имеет ненулевое значение когда $k=k_0$, а вторая когда $k=N-k_0$. Следовательно, $\widetilde{X}(k_0)=\frac{N}{2}$ и $\widetilde{X}(N-k_0)=\frac{N}{2}$. То есть единичной амплитуде исходного сигнала соответствует значение ДПФ $\frac{N}{2}$.

Таким образом, если частота дискретного сигнала кратна частоте $\frac{2\pi}{NT}$, то ДПФ точно определяет спектр сигнала: среди составляющих спектра только одна ненулевая, как и должно быть для гармонического сигнала.

Если же $\alpha\neq 0$, то даже при $k\neq k_0$ значения ДПФ будут отличны от нуля, то есть оценка спектра сигнала будет искаженной – в исходном сигнале одна составляющая, а в спектре дискретного сигнала – много, как показано на рис. 3.12. Исходный сигнал представляет собой чистую синусоиду, однако ее частота не кратна значению $\frac{2\pi}{NT}$. На границе условного периода наблюдения возникает разрыв, которого в реальном сигнале нет (рис. 3.12).

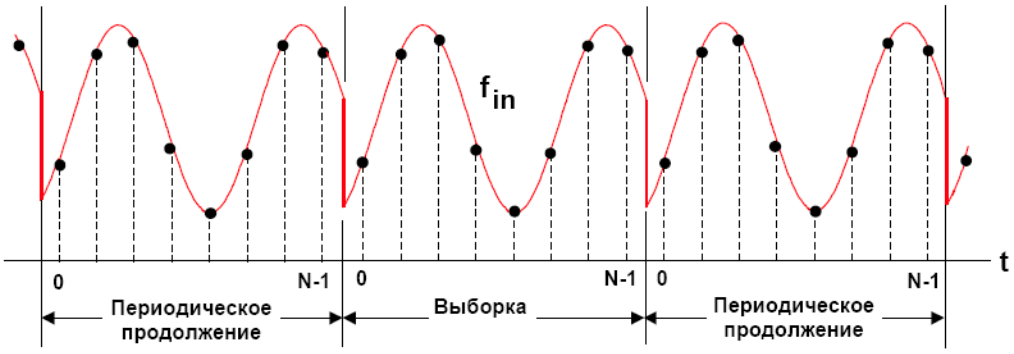


Рис. 3.12.

Конечную запись данных $x(n)$ из N отсчетов можно представить как некоторую часть исходной бесконечной последовательности $x_u(n)$, видимую через прямоугольное окно $w(n)$, в форме произведения:

$$x[n] = x_u[n]w[n],$$
$$w[n] = \begin{cases} 1, & 0 \le n \le N-1, \\ 0, & \text{при других } n. \end{cases}$$

При этом мы полагаем, что все ненаблюдаемые отсчеты равны нулю независимо от того, так ли это на самом деле или нет.

ДПФ взвешенной окном последовательности, выраженное через преобразования последовательности $x_u[n]$ и прямоугольного окна $w[n]$, равно свертке этих преобразований:

$$X(f) = X_u(f) * D_N(f),$$

Влияние прямоугольного окна на дискретно-временную синусоиду с частотой f_0 проявляется в том, что острые спектральные пики ДПФ исходной синусоидальной последовательности расширились из-за воздействия ДПФ окна. При этом минимальная ширина спектральных пиков конечной последовательности ограничена шириной, определяемой главным лепестком ДПФ окна, и не зависит от исходных данных. Боковые лепестки ДПФ окна, называемые растеканием [просачиванием (spectral leakage)], будут изменять амплитуды соседних спектральных пиков, приводя к смещению спектральных оценок. Аналогичные искажения будут наблюдаться и в случае несинусоидальных сигналов. Просачивание приводит не только к появлению амплитудных ошибок, но может также маскировать присутствие слабых сигналов и, следовательно, затруднять их обнаружение. Предложен ряд других функций окна, имеющих меньший уровень боковых лепестков их спектра, чем в случае прямоугольного окна. Однако, как правило, это достигается ценой расширения главного лепестка спектра окна. Следовательно, окно должно выбираться с учетом компромисса между шириной главного лепестка и уровнем подавления боковых лепестков.

Разрывы, которые образуются в конечных точках выборки, приводят к расширению спектра анализируемого сигнала вследствие появления дополнительных гармоник (рис. 3.13). В дополнение к появлению боковых лепестков, происходит расширение основного лепестка, что приводит к снижению разрешающей способности по частоте.

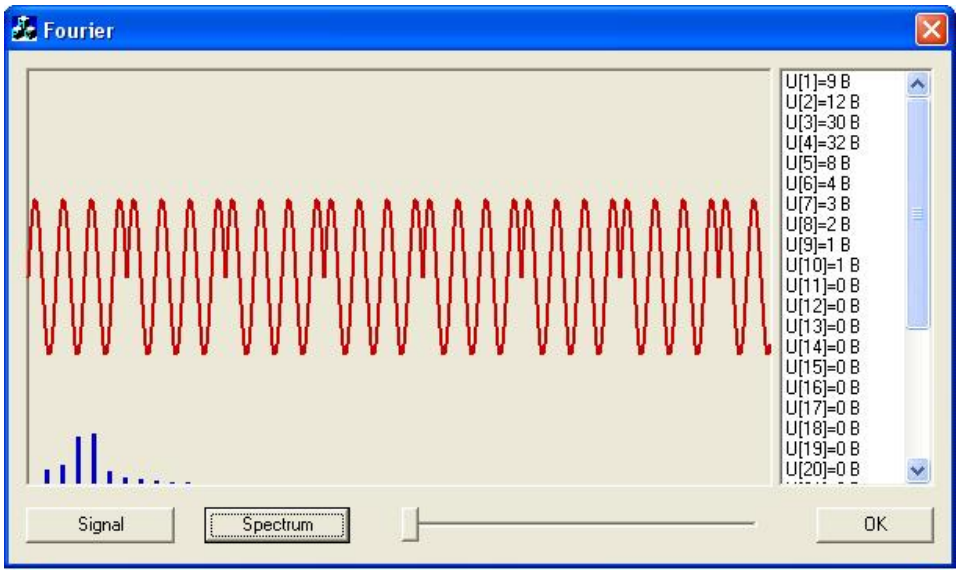


Рис. 3.13.

Следовательно, неискаженную оценку спектра исходного сигнала с помощью ДПФ можно получить лишь в том случае, когда на интервале наблюдения укладывается целое число периодов сигнала, то есть все частоты всех составляющих кратны $\frac{2\pi}{NT}$.

Такая ситуация неприемлема для большинства задач анализа спектра. Поскольку в практических приложениях ДПФ для спектрального анализа точные входные частоты неизвестны, следует предпринять определенные шаги к уменьшению боковых лепестков. Оно достигается выбором оконной функции с более сложной формой, чем прямоугольная. Входные отсчеты по времени умножаются на соответствующую функцию окна, что влечет за собой обнуление сигнала на краях выборки, как показано на рис. 3.14.

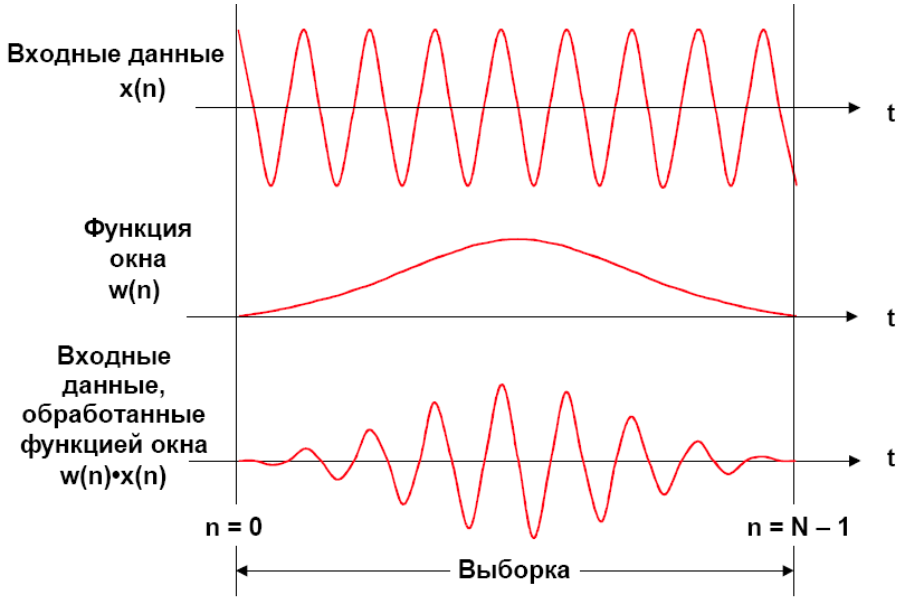


Рис. 3.14.

В табл. 3.1 приведены определения и сравнительные характеристики некоторых используемых N-точечных функций окна, симметричных относительно n=0.

Табл. 3.1.

Окно	Дискретно-временная функция $w(n)$	Максимальный уровень боковых лепестков (дБ)	Ширина полосы по уровню 0,5 мощности относительно $1/NT$ Гц
Прямоугольное	1	–13,3	0,89
Треугольное (Бартлета)	$1-\frac{2\lvert n\rvert}{N}$	–26,5	1,28
Ханна	$0{,}5-0{,}5\cos\left(\frac{2\pi n}{N}\right)$	–31,5	1,44
Хэмминга	$0{,}54-0{,}46\cos\left(\frac{2\pi n}{N}\right)$	–43	1,3

На рис. 3.15 представлена осциллограмма сигнала после применения цифрового окна и результат спектрального анализа. Для генерирования синусоидального сигнала без применения цифрового окна использован код:

```
m_d[i]=50*sin(6.28*i/16);
```

Для генерирования синусоидального сигнала с применения цифрового окна Хэмминга использован код:

```
m_d[i]=50*sin(6.28*i/18)*(0.54-0.46*cos(6.28*i/64));
```

Как видно из рисунка, уровень ложных боковых лепестков после применения цифрового окна значительно снизился.

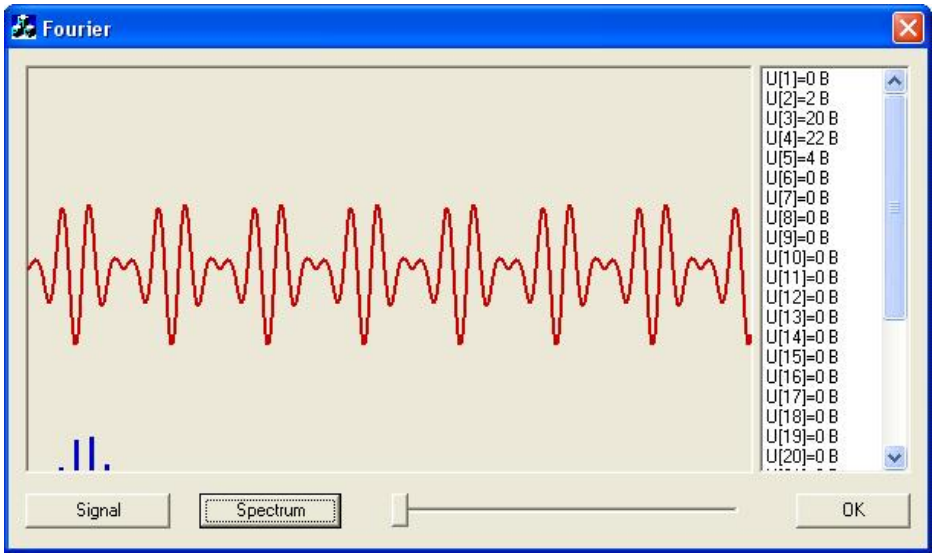


Рис. 3.15.

В табл. 3.2 представлены характеристики различных окон.

Табл. 3.2.

Окно	Разрешение по частоте	Разрешение по амплитуде	Подавление утечки	Применение
Bartlett	достаточное	достаточное	умеренное	
Blackman	достаточное	хорошее	отличное	Измерение искажений
Flattop	плохое	отличное	умеренное	Точные измерения амплитуды
Hamming	достаточное	достаточное	достаточное	Измерение искажений, шума
Hanning	достаточное	отличное	отличное	
Kaiser	достаточное	достаточное	плохое	
Parzen	достаточное	достаточное	плохое	
Triangular	достаточное	достаточное	плохое	
Uniform	отличное	плохое	плохое	Измерение частоты с высоким разрешением, измерение импульсного отклика

Существует два основных способа обработки сигналов в реальном масштабе времени: обработка одного отсчета в каждый момент времени (непрерывная обработка) и обработка одного пакета данных в каждый момент времени (пакетная обработка). Системы, основанные на непрерывной обработке, такие как цифровой фильтр, получают данные в виде одного отсчета в каждый момент времени. В каждом такте новый отсчет поступает в систему, а обработанный отсчет передается на выход. Системы, основанные на пакетной обработке, такие как построенный на ДПФ или БПФ (быстрое преобразование Фурье – алгоритм быстрого вычисления дискретного преобразования Фурье) цифровой анализатор спектра, получают данные в виде целого пакета отсчетов. Происходит обработка всего пакета исходных данных, результатом которой является пакет преобразованных выходных данных.

Для обеспечения функционирования в реальном масштабе времени полный расчет ДПФ или БПФ должен выполняться в промежутке, соответствующем времени накопления одного пакета данных. Предполагается, что, пока производится вычисление преобразования Фурье текущего пакета данных, DSP-процессор накапливает данные для следующего пакета. Накопление данных является одной из сфер, где важную роль играют специальные архитектурные особенности DSP. Непрерывное получение данных облегчается, благодаря возможностям гибкой адресации данных в DSP в сочетании с использованием различных каналов прямого доступа к памяти (DMA).

3.4. Исследование корреляционных характеристик сигналов

Корреляция (correlation) является методом анализа сигналов. Допустим, что имеется сигнал $s(t)$, в котором может быть (а может и не быть) некоторая последовательность $x(t)$ конечной длины T, временное положение которой нас интересует. Для поиска этой последовательности в скользящем по сигналу $s(t)$ временном окне длиной T вычисляются скалярные произведения сигналов $s(t)$ и $x(t)$. Тем самым мы "прикладываем" искомый сигнал $x(t)$ к сигналу $s(t)$, скользя по его аргументу, и по величине скалярного произведения оцениваем степень сходства сигналов в точках сравнения.

Процесс корреляции занимает значительное место в обработке сигналов. Этот математический аппарат нашел применение в обработке изображений в сфере компьютерного зрения или дистанционного зондирования со спутников, в которых сравниваются данные из любых изображений, в радарных или гидроакустических установках для дальнометрии и местоопределения (пеленгации), в которых сравниваются переданные и отраженные сигналы. Он используется в детектировании и идентификации сигналов в шуме, в организации технического контроля для наблюдения за влиянием входа на выход, в идентификации двоичных кодовых слов в системе с импульсно-кодовой модуляцией, в обычных схемах оценки методом наименьших квадратов и других областях, например, в климатологии. Корреляция также является неотъемлемой частью процесса свертки, который, по сути, та же корреляция двух последовательностей данных, при вычислении которой одна из последовательностей обращена во времени. Это означает, что для вычисления корреляции и свертки могут использоваться одни и те же алгоритмы, только в случае свертки одна из последовательностей обращается.

Корреляционный анализ дает возможность установить в сигналах (или в рядах цифровых данных сигналов) наличие определенной связи изменения значений сигналов по независимой переменной, то есть, когда большие значения одного сигнала (относительно средних значений сигнала) связаны с большими значениями другого сигнала (положительная корреляция), или, наоборот, малые значения одного сигнала связаны с большими значениями другого (отрицательная корреляция), или данные двух сигналов никак не связаны (нулевая корреляция).

В функциональном пространстве сигналов эта степень связи может выражаться в нормированных единицах коэффициента корреляции, т.е. в косинусе угла между векторами сигналов, и, соответственно, будет принимать значения от 1 (полное совпадение сигналов) до –1 (полная противоположность) и не зависит от значения (масштаба) единиц измерений.

В варианте автокорреляции (autocorrelation) по аналогичной методике производится определение скалярного произведения сигнала $s(t)$ с собственной копией, скользящей по аргументу. Автокорреляция позволяет оценить среднестатистическую зависимость текущих отсчетов сигнала от своих предыдущих и последующих значений (так называемый радиус

корреляции значений сигнала), а также выявить в сигнале наличие периодически повторяющихся элементов.

Особое значение методы корреляции имеют при анализе случайных процессов для выявления неслучайных составляющих и оценки неслучайных параметров этих процессов.

Автокорреляционная функция (АКФ, CF – correlation function) сигнала $s(t)$, конечного по энергии, является количественной интегральной характеристикой формы сигнала, выявления в сигнале характера и параметров взаимной временной связи отсчетов, что всегда имеет место для периодических сигналов, а также интервала и степени зависимости значений отсчетов в текущие моменты времени от предыстории текущего момента. АКФ определяется интегралом от произведения двух копий сигнала s(t), сдвинутых относительно друг друга на время τ:

$$B_s(\tau) = \int_{-\infty}^{\infty} s(t)s(t+\tau)dt .$$

Как следует из этого выражения, АКФ является скалярным произведением сигнала и его копии в функциональной зависимости от переменной величины значения сдвига τ. Соответственно, АКФ имеет физическую размерность энергии, а при τ=0 значение АКФ непосредственно равно энергии сигнала и является максимально возможным:

$$B_s(0) = \int_{-\infty}^{\infty} s^2(t)dt = E_s .$$

Максимум АКФ, равный энергии сигнала при τ=0, всегда положителен, а модуль АКФ при любом значении временного сдвига не превосходит энергии сигнала.

Взаимная корреляционная функция (ВКФ) разных сигналов (cross-correlation function, CCF) описывает как степень сходства формы двух сигналов, так и их взаимное расположение друг относительно друга по координате (независимой переменной). Обобщая формулу автокорреляционной функции на два различных сигнала $s(t)$ и $u(t)$, получаем следующее скалярное произведение сигналов:

$$B_{su}(\tau) = \int_{-\infty}^{\infty} s(t)u(t+\tau)dt .$$

Взаимная корреляция сигналов характеризует определенную корреляцию явлений и физических процессов, отображаемых данными сигналами, и может служить мерой “устойчивости” данной взаимосвязи при раздельной обработке сигналов в различных устройствах.

Рис. 3.16 демонстрирует разницу между сверткой, АКФ и ВКФ.

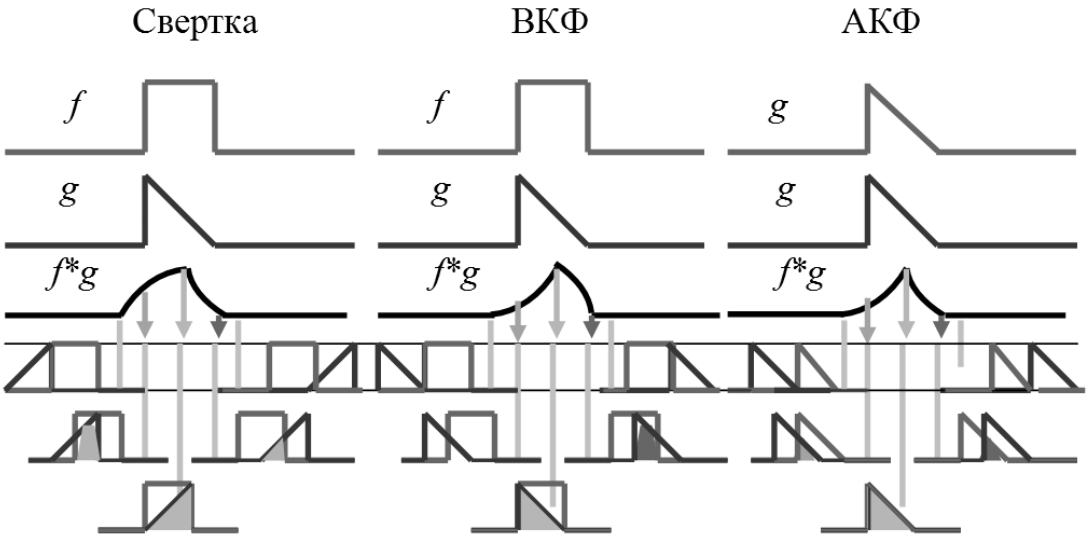


Рис. 3.16.

Если два похоже меняются при переходе от точки к точке, то меру их корреляции можно вычислить, взяв сумму произведений соответствующих пар точек. Данное предложение становится более аргументированным, если рассмотреть две независимые и случайные последовательности данных.

Таким образом, взаимную корреляцию $r_{12}(n)$ двух последовательностей данных $x_1(n)$ и $x_2(n)$, содержащих по N элементов, можно записать как:

$$r_{12} = \sum_{0}^{N-1} x_1(n)x_2(n) .$$

Такое определение взаимной корреляции дает результат, который зависит от числа взятых точек. Чтобы это исправить, результат нормируется на число точек (делится на N). Данную операцию можно также рассматривать как усреднение суммы произведений:

$$r_{12} = \frac{1}{N}\sum_{0}^{N-1} x_1(n)x_2(n) .$$

В некоторых случаях корреляция, определенная указанным выше способом, может быть нулевой, хотя две последовательности коррелируют на 100%. Это может произойти, например, когда два сигнала идут не в фазе (как часто и бывает).

Чтобы преодолеть подобный сдвиг фаз, необходимо сдвинуть (или задержать) один из сигналов относительно другого. Обычно, чтобы выровнять сигналы перед определением корреляции, x_2 смещается влево. Это эквивалентно замене $x_2(n)$ на $x_2(n+j)$, где j представляет величину задержки – число точек выборки, на которое x_2 смещается влево. Альтернативной и эквивалентной процедурой является смещение x_1 вправо. В результате получаем такую формулу для взаимной корреляции:

$$r_{12}(j) = \frac{1}{N}\sum_{0}^{N-1} x_1(n)x_2(n+j) = r_{12}(-j) = \frac{1}{N}\sum_{0}^{N-1} x_2(n)x_1(n-j) .$$

3.5. Реализация алгоритма расчета корреляционной функции

В редакторе ресурсов среды Visual Studio 6.0 проектируем внешний вид диалогового окна (рис. 3.17). На диалоговом окне расположен элемент управления Static Text, на котором выполняется построение осциллограмм сигналов и функции корреляции, кнопка Button, нажатие на которую приведет к запуску алгоритма расчета функции корреляции, и элементы Radio Button, с помощью которых выбирается тип сигнала для выполнения корреляционного анализа.

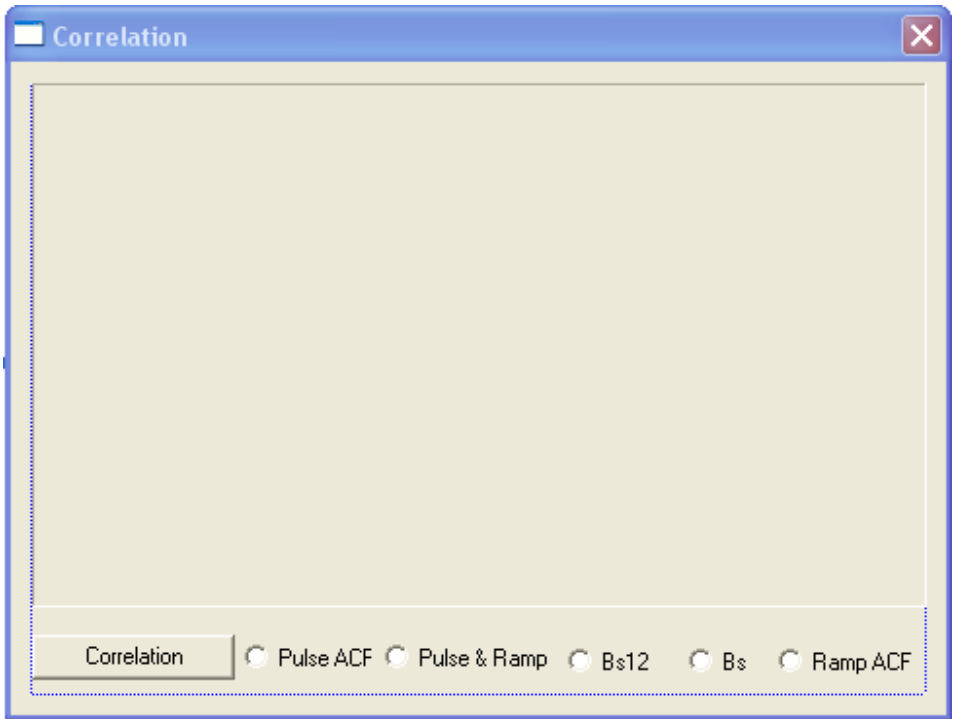


Рис. 3.17.

С элементом управления Static Text связываем переменную m_static, а с Radio Button переменную m_mode, воспользовавшись окном ClassWizard (закладка Member Variables), как показано на рис. 3.18.

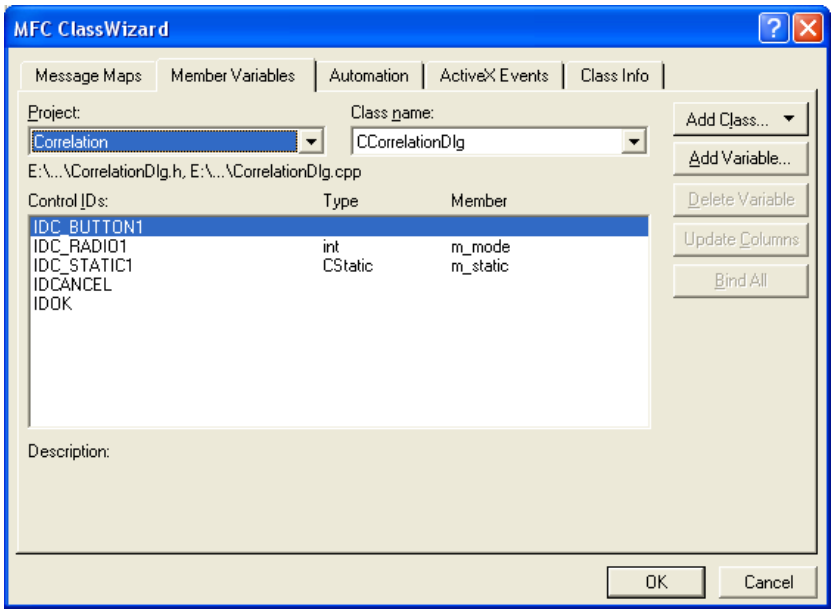


Рис. 3.18.

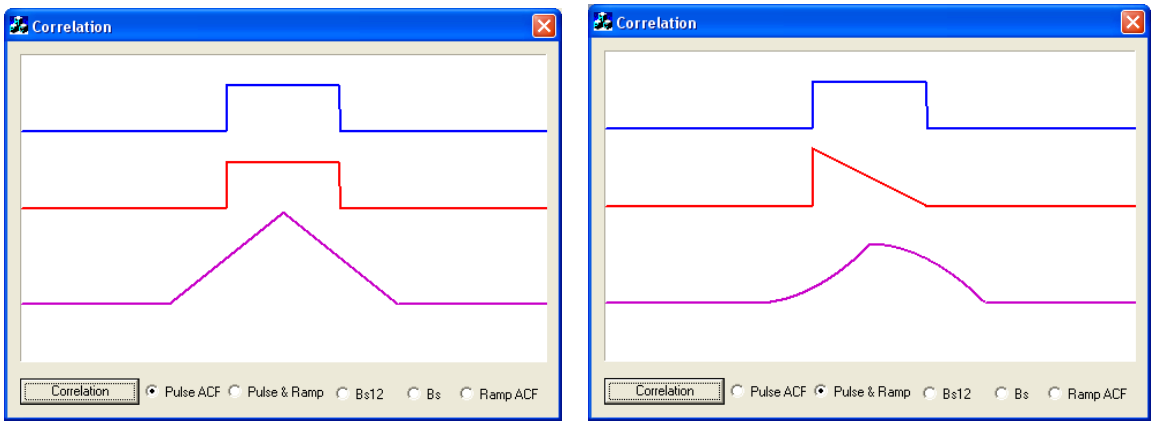
Добавляем обработчик нажатия на кнопку Correlation:

```
void CCorrelationDlg::OnButton1()
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  int W=r.Width();
  int H=r.Height();
  int *s1=new int[100];
  int *s2=new int[100];
  memset(s1,0,sizeof(int)*100);
  memset(s2,0,sizeof(int)*100);
  UpdateData(true);
  for(int i=0;i<100;i++)
  {
          if(m_mode==0)
          {
                  s1[i]=40;
                  s2[i]=40;
          }
          else if(m_mode==1)
          {
                  s1[i]=40;
                  s2[i]=50-i/2;
          }
          else if(m_mode==2)
          {
                  s1[i]=rand()%61-30;
                  s2[i]=rand()%61-30;
          }
```

```
        else if(m_mode==3)
        {
                s1[i]=rand()%61-30;
                s2[i]=s1[i];
        }
        else if(m_mode==4)
        {
                s1[i]=i/2;
                s2[i]=i/2;
        }
}
dc.FillRect(r,&CBrush(RGB(255,255,255)));
CPen p, *op;
p.CreatePen(PS_SOLID,2,RGB(0,0,255));
op=dc.SelectObject(&p);
dc.MoveTo(0,H/4);
dc.LineTo(W/2-50,H/4);
dc.LineTo(W/2-50+1,H/4-s1[0]);
for(i=1;i<100;i++)
{
        dc.MoveTo(i+W/2-50,H/4-s1[i]);
        dc.LineTo(i-1+W/2-50,H/4-s1[i-1]);
}
dc.LineTo(W/2+50,H/4);
dc.MoveTo(W/2+50,H/4);
dc.LineTo(W,H/4);
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(255,0,0));
op=dc.SelectObject(&p);
dc.MoveTo(0,H/2);
dc.LineTo(W/2-50,H/2);
dc.LineTo(W/2-50+1,H/2-s2[0]);
for(i=1;i<100;i++)
{
        dc.MoveTo(i+W/2-50,H/2-s2[i]);
        dc.LineTo(i-1+W/2-50,H/2-s2[i-1]);
}
dc.LineTo(W/2+50,H/2);
dc.MoveTo(W/2+50,H/2);
dc.LineTo(W,H/2);
int *corr=new int[200];
memset(corr,0,sizeof(int)*200);
int n,m;
for(n=0;n<100;n++)
{
        for(m=0;m<n;m++)
        {
                corr[n]=corr[n]+s1[m]*s2[100-n+m];
        }
        corr[n]=corr[n]/2000;
```

```
    }
    for(n=0;n<100;n++)
    {
            for(m=n;m<100;m++)
            {
                    corr[n+100]=corr[n+100]+s1[m]*s2[m-n];
            }
            corr[n+100]=corr[n+100]/2000;
    }
    p.DeleteObject();
    p.CreatePen(PS_SOLID,2,RGB(200,0,200));
    op=dc.SelectObject(&p);

    dc.MoveTo(0,H-50);
    dc.LineTo(W/2-100,H-50);
    for(i=1;i<200;i++)
    {
            dc.MoveTo(W/2-100+i,H-50-corr[i]);
            dc.LineTo(W/2-100+i-1,H-50-corr[i-1]);
    }
    dc.MoveTo(W/2+100,H-50);
    dc.LineTo(W,H-50);
    dc.SelectObject(op);
    delete [] s1;
    delete [] s2;
    delete [] corr;
}
```

На рис. 3.19, а, представлено окно программы с осциллограммами сигналов и вычисленной автокорреляционной функцией прямоугольного импульса, а на рис. 3.19, б, представлено окно программы с осциллограммами сигналов и вычисленной функцией взаимной корреляции прямоугольного и треугольного импульсов.

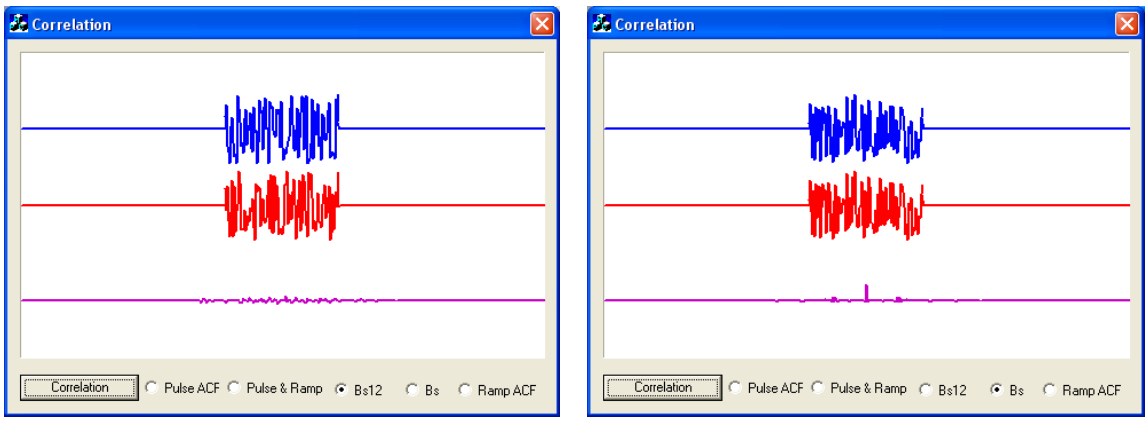


(а) (б)

Рис. 3.19.

На рис. 3.19, б, сигналы $s(t)$ (прямоугольный импульс) и $u(t)$ (треугольный импульс) одинаковы по временному расположению и площадь "перекрытия" сигналов максимальна при $\tau=0$, что и фиксируется функцией B_{su}. Вместе с тем функция B_{su} резко асимметрична, так как при асимметричной форме сигнала $u(t)$ для симметричной формы $s(t)$ (относительно центра сигналов) площадь "перекрытия" сигналов изменяется по разному в зависимости от направления сдвига (знака τ при увеличения значения τ от нуля). При смещении исходного положения сигнала $u(t)$ влево по оси ординат (на опережение сигнала $s(t)$ – сигнал $v(t)$) форма ВКФ остается без изменения и сдвигается вправо на такое же значение величины.

На рис. 3.20, а, представлено окно программы с осциллограммами сигналов и вычисленной функцией взаимной корреляции двух независимых случайных сигналов, а на рис. 3.20, б, представлено окно программы с осциллограммами сигналов и вычисленной автокорреляционной функцией случайного сигнала (белый шум).



а *б*

Рис. 3.20.

На рис. 3.21 представлено окно программы с осциллограммами сигналов и вычисленной автокорреляционной функцией треугольного импульса.

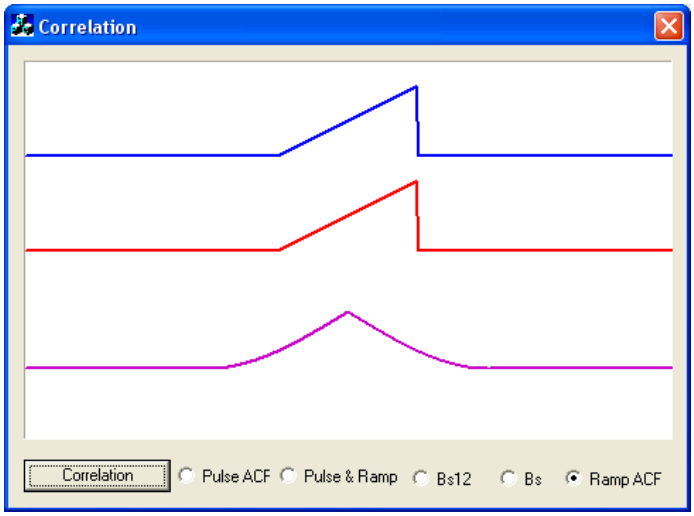


Рис. 3.21.

3.6. Использование согласованной фильтрации в цифровых системах передачи данных

Одной из задач, решаемых при цифровой обработке сигналов, является их различение. В качестве примера рассмотрим различение детерминированных сигналов.

На известном временном интервале [0, T_c] на входе устройства различения сигналов (различителя) действует сигнал из ансамбля детерминированных сигналов $\{s_i(t)\}$, i=0, …, N. Сигналы равны нулю вне интервала [0, T_c]. От различителя сигналов требуется определить, какой из известных сигналов поступил на его вход. Структурная схема различителя показана на рис. 3.22.

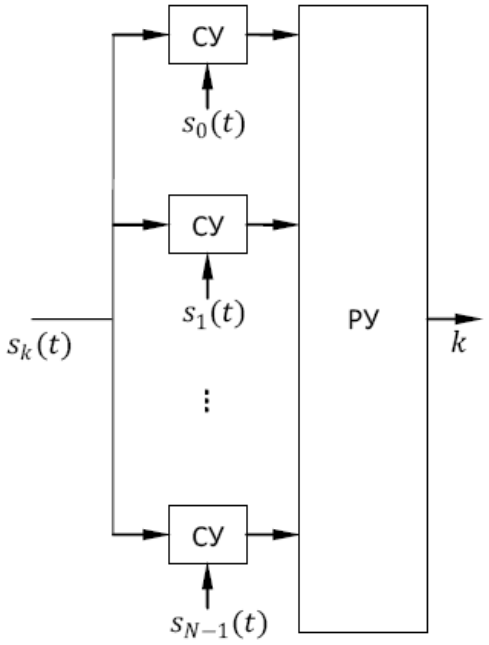


Рис. 3.22.

Различитель располагает эталонами (образцами) сигналов и сравнивает сигнал, поступивший на вход, с каждым из эталонов. По результатам сравнения, осуществляемом в сравнивающих устройствах (СУ), в решающем устройстве (РУ) принимается решение о том, какой из сигналов действует на входе. В основу различения сигналов, таким образом, положено их сравнение. При сравнении сигналов устанавливается степень их взаимного соответствия по форме. При совпадении сравниваемых сигналов коэффициент корреляции максимален и равен энергии. Таким образом, сравнение сигналов может осуществляться и на основе анализа значения коэффициента корреляции.

Максимальное значение действительной части коэффициента корреляции будет получено только в том канале устройства различения сигналов где произошло совпадение по форме обрабатываемого и эталонного сигналов. Номер этого канала будет установлен решающим устройством различителя по максимальному отклику.

Свойства модуля коэффициента корреляции аналогичны свойствам его действительной части, поэтому при сравнении сигналов может использоваться и модуль коэффициента корреляции.

Структурная схема устройства для получения корреляционного интеграла (коррелятора) в показана на рис. 3.23.

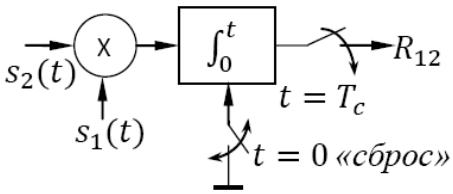


Рис. 3.23.

В момент, когда начинают действовать сигналы, осуществляется сброс интегратора. Значение коэффициента корреляции снимается с выхода интегратора по окончании действия сигналов в момент времени. Один из входов коррелятора можно условно считать опорным. На второй вход поступает тестируемый сигнал. При формировании отклика коррелятора вычисляется произведение опорного и тестируемого сигналов, и интегратор определяет площадь под графиком этого произведения.

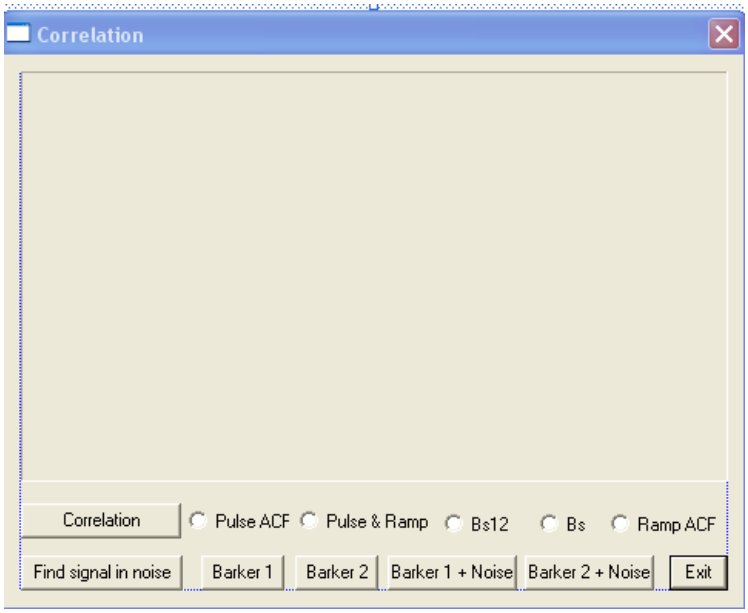


Рис. 3.24.

Функция-обработчик нажатия на кнопку Find signal in noise представлена далее.

```
void CCorrelationDlg::OnButton2()
{
  CClientDC dc(&m_static);
```

```
CRect r;
m_static.GetClientRect(&r);
int W=r.Width();
int H=r.Height();
int *s1=new int[W];
int *s2=new int[W];
const int T=25;
int ramp[T];
for(int k=0;k<T;k++) ramp[k]=k;
int MaxAutoCorr=0;
for(k=0;k<T;k++) MaxAutoCorr=MaxAutoCorr+ramp[k]*ramp[k];
memset(s1,0,sizeof(int)*W);
memset(s2,0,sizeof(int)*W);

int pos1=rand()%100;
int pos2=rand()%100+110+T/2;
int pos3=rand()%100+230+T;
for(int i=0;i<W;i++)
{
	s1[i]=rand()%50-25;

	if((i>=pos1) && (i<(pos1+T)))
	{
		s2[i]=i-pos1;
	}
	else if((i>=pos2) && (i<(pos2+T)))
	{
		s2[i]=10;
	}
	else if((i>=pos3) && (i<(pos3+T))) s2[i]=i-pos3;
	else s2[i]=0;
	s1[i]=s1[i]+s2[i];
}
dc.FillRect(r,&CBrush(RGB(255,255,255)));
CPen p, *op;
p.CreatePen(PS_SOLID,2,RGB(0,0,255));
op=dc.SelectObject(&p);
for(i=1;i<W;i++)
{
	dc.MoveTo(i,H/4-s1[i]);
	dc.LineTo(i-1,H/4-s1[i-1]);
}
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(255,0,0));
op=dc.SelectObject(&p);
for(i=1;i<W;i++)
{
	dc.MoveTo(i,H/2-s2[i]);
	dc.LineTo(i-1,H/2-s2[i-1]);
}
```

```
int *corr=new int[W];
memset(corr,0,sizeof(int)*W);
int n;
int tmp=0;
for(n=0;n<(W-T);n++)
{
	for(int m=0;m<T;m++)
	{
		tmp=tmp+s1[n+m]*ramp[m];
	}
	corr[n]=tmp;
	tmp=0;
}
p.DeleteObject();
p.CreatePen(PS_DASH,1,RGB(0,0,0));
op=dc.SelectObject(&p);
int Delta=500;
bool is_find=false;
for(i=1;i<W;i++)
{
	if((abs(corr[i]-MaxAutoCorr)<Delta) && (is_find==false))
	{
		dc.MoveTo(i,0);
		dc.LineTo(i,H);
		is_find=true;
	}
if((corr[i]<(MaxAutoCorr-2*Delta)) && (is_find==true)) is_find=false;
}
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(200,0,200));
op=dc.SelectObject(&p);
for(i=1;i<W;i++)
{
	dc.MoveTo(i,H-50-corr[i]/200);
	dc.LineTo(i-1,H-50-corr[i-1]/200);
}
p.DeleteObject();
p.CreatePen(PS_SOLID,1,RGB(0,0,0));
op=dc.SelectObject(&p);
dc.MoveTo(0,H/4);
dc.LineTo(W,H/4);
dc.MoveTo(0,H/2);
dc.LineTo(W,H/2);
dc.MoveTo(0,H-50);
dc.LineTo(W,H-50);
dc.SelectObject(op);
delete [] s1;
delete [] s2;
delete [] corr;
}
```

На рис. 3.25 представлен результат работы программы. Информационный сигнал сильно зашумлен. Известно, что в сигнале имеются прямоугольные и треугольные импульсы. Программа путем сигнасованной фильтрации обнаруживает положение треугольных импульсов и обозначает их положение пунктирной линией.

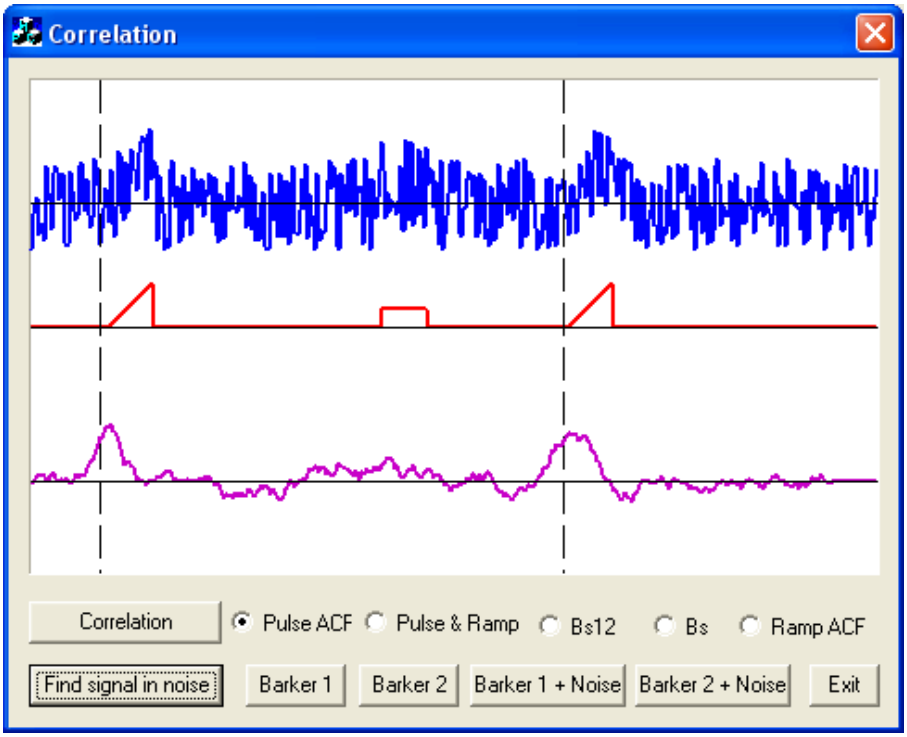


Рис. 3.25.

При потенциальном кодировании информационные биты – логические нули и единицы – передаются прямоугольными импульсами напряжений. Прямоугольный импульс длительности T имеет спектр, ширина которого обратно пропорциональна длительности импульса. Поэтому чем меньше длительность информационного бита, тем больший спектр занимает такой сигнал.

Для преднамеренного расширения спектра первоначально узкополосного сигнала в каждый передаваемый информационный бит (логический 0 или 1) в буквальном смысле встраивается последовательность так называемых чипов. Если информационные биты – логические нули или единицы – при потенциальном кодировании информации можно представить в виде последовательности прямоугольных импульсов, то каждый отдельный чип – это тоже прямоугольный импульс, но его длительность в несколько раз меньше длительности информационного бита. Последовательность чипов представляет собой последовательность прямоугольных импульсов, то есть нулей и единиц, однако эти нули и единицы не являются информационными. Поскольку длительность одного чипа в n раз меньше длительности информационного бита, то и ширина спектра преобразованного сигнала будет

в *n*-раз больше ширины спектра первоначального сигнала. При этом и амплитуда передаваемого сигнала уменьшится в *n* раз.

Чиповые последовательности, встраиваемые в информационные биты, называют шумоподобными кодами (PN-последовательности), что подчеркивает то обстоятельство, что результирующий сигнал становится шумоподобным и его трудно отличить от естественного шума.

Кодовые сигналы являются разновидностью дискретных сигналов. На определенном интервале кодового слова Δt они могут иметь только два амплитудных значения: 0 и 1 или 1 и –1. При выделении кодов на существенном уровне шумов форма АКФ кодового слова имеет особое значение. Используемые для расширения спектра сигнала чиповые последовательности должны удовлетворять определенным требованиям автокорреляции. С этой позиции наилучшими считаются такие коды, значения боковых лепестков АКФ которых минимальны по всей длине интервала кодового слова при максимальном значении центрального пика. К числу таких кодов относится код Баркера (табл. 3.3).

Табл. 3.3.

M	Сигнал Баркера	АКФ сигнала
2	1, -1	2, -1
3	1, 1, -1	3, 0, -1
4	1, 1, 1, -1	4, 1, 0, -1
	1, 1, -1, 1	4, -1, 0, 1
5	1, 1, 1, -1, 1	5, 0, 1, 0, 1
7	1, 1, 1, -1, -1, 1, -1	7, 0, -1, 0, -1, 0, -1
11	1,1,1,-1,-1,-1,1,-1,-1,1,-1	11,0,-1,0,-1,0,-1,0,-1,0,-1
13	1,1,1,1,1,-1,-1,1,1-1,1,-1,1	13,0,1,0,1,0,1,0,1,0,1,0,1

Если подобрать такую чиповую последовательность, для которой функция автокорреляции будет иметь резко выраженный пик лишь для одного момента времени, то такой информационный сигнал возможно будет выделить на уровне шума. Для этого в приемнике полученный сигнал умножается на ту же чиповую последовательность, то есть вычисляется автокорреляционная функция сигнала. В результате сигнал становится опять узкополосным, поэтому его фильтруют в узкой полосе частот и любая помеха, попадающая в полосу исходного широкополосного сигнала, после умножения на чиповую последовательность, наоборот, становится широкополосной и обрезается фильтрами, а в узкую информационную полосу попадает лишь часть помехи, по мощности значительно меньшая, чем помеха, действующая на входе приемника.

Чиповых последовательностей, отвечающих указанным требованиям автокорреляции, существует достаточно много, но особый интерес

представляют коды Баркера, поскольку именно они используются в протоколе 802.11. Графическое изображение АКФ последовательности Баркера показано на рис. 3.26.

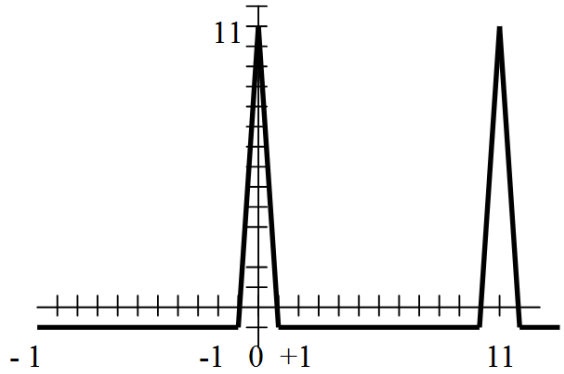


Рис. 3.26.

Такую АКФ можно назвать идеальной, поскольку на ней отсутствуют боковые пики, которые могли бы способствовать ложному обнаружению сигнала.

В протоколах семейства 802.11 используется код Баркера длиной в 11 чипов (11100010010). Для того чтобы передать сигнал логическая единица передается прямой последовательностью Баркера, а логический нуль – инверсной последовательностью. Таким образом имеется возможность каждый бит информации кодировать 11 битами последовательности Баркера – прямой для единиц и инверсной для нулей. На практике кодирование происходит примерно так, как показано на рис. 3.27.

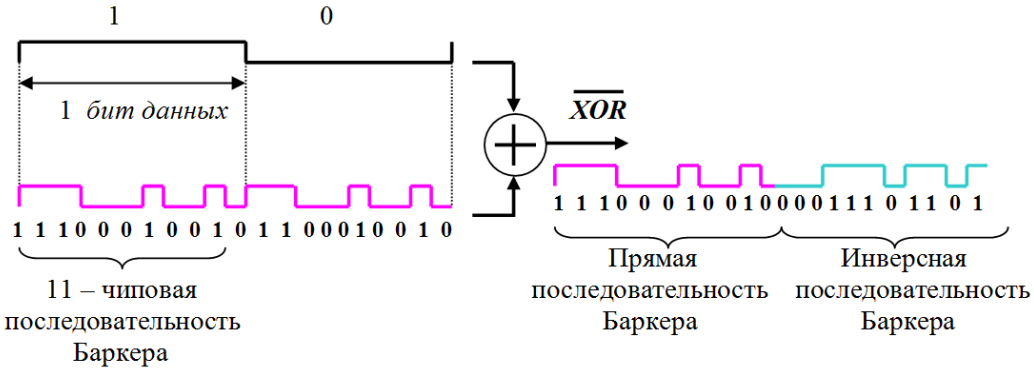


Рис. 3.27.

Приемник может рассчитывать корреляцию последовательностей Баркера (прямой и инверсной) и входного сигнала, и по пикам корреляционной функции определять – где во входном сигнале закодированы нули, а где – единицы. Корреляционный метод является самым оптимальным методом определения сигнала известной формы на фоне белого шума – метод имеет наилучшее отношение сигнал/шум. Результат осреднения или интегрирования случайного шума без постоянной составляющей в течении долгого периода времни будет равен нулю.

Задача формулируется следующим образом: найти в длинной последовательности входных данных, полученных в результате передачи по сильно зашумленному каналу заранее известную короткую последовательность.

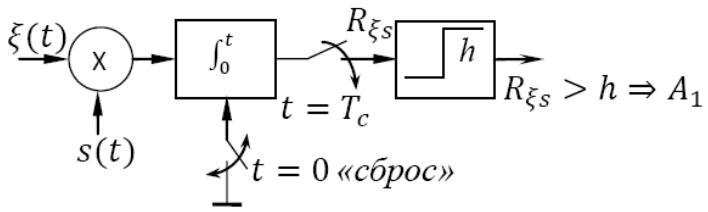


Рис. 3.28.

В состав обнаружителя входит коррелятор или согласованный фильтр и пороговое устройство. Принцип оптимального обнаружения основан на сравнении обрабатываемого колебания с опорным сигналом. Когда коэффициент корреляции превышает пороговое значение, обрабатываемое колебание считается подобным опорному сигналу и принимается решение о его наличии. В схеме с коррелятором опорный сигнал генерируется непосредственно в месте обработки.

Функция-обработчик нажатия на кнопку Barker 1 представлена ниже.

```
void CCorrelationDlg::OnButton3()
{
  CClientDC dc(&m_static);
  CRect r;
  m_static.GetClientRect(&r);
  int W=r.Width();
  int H=r.Height();
  int *s1=new int[100];
  int *s2=new int[100];
  memset(s1,0,sizeof(int)*100);
  memset(s2,0,sizeof(int)*100);
  int index=0;
  int bark1[11]={-1,1,1,1,-1,-1,-1,1,-1,-1,1};
  int bark2[11]={-1,1,1,1,-1,-1,-1,1,-1,-1,1};
  for(int i=1;i<100;i++)
  {
          s1[i]=20*bark1[index];
          s2[i]=20*bark2[index];
          if(i%8==0) index++;
          if(index==11) break;
  }
  dc.FillRect(r,&CBrush(RGB(255,255,255)));
  CPen p, *op;
  p.CreatePen(PS_SOLID,2,RGB(0,0,255));
  op=dc.SelectObject(&p);
```

```
dc.MoveTo(0,H/4);
dc.LineTo(W/2-50,H/4);
dc.LineTo(W/2-50+1,H/4-s1[0]);
for(i=1;i<100;i++)
{
	dc.MoveTo(i+W/2-50,H/4-s1[i]);
	dc.LineTo(i-1+W/2-50,H/4-s1[i-1]);
}
dc.LineTo(W/2+50,H/4);
dc.MoveTo(W/2+50,H/4);
dc.LineTo(W,H/4);
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(255,0,0));
op=dc.SelectObject(&p);
dc.MoveTo(0,H/2);
dc.LineTo(W/2-50,H/2);
dc.LineTo(W/2-50+1,H/2-s2[0]);
for(i=1;i<100;i++)
{
	dc.MoveTo(i+W/2-50,H/2-s2[i]);
	dc.LineTo(i-1+W/2-50,H/2-s2[i-1]);
}
dc.LineTo(W/2+50,H/2);
dc.MoveTo(W/2+50,H/2);
dc.LineTo(W,H/2);
int *corr=new int[200];
memset(corr,0,sizeof(int)*200);
int n,m;
for(n=0;n<100;n++)
{
	for(m=0;m<n;m++)
	{
		corr[n]=corr[n]+s1[m]*s2[100-n+m];
	}
	corr[n]=corr[n]/800;
}
for(n=0;n<100;n++)
{
	for(m=n;m<100;m++)
	{
		corr[n+100]=corr[n+100]+s1[m]*s2[m-n];
	}
	corr[n+100]=corr[n+100]/800;
}
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(200,0,200));
op=dc.SelectObject(&p);
dc.MoveTo(0,H-50);
dc.LineTo(W/2-100,H-50);
for(i=1;i<200;i++)
```

```
  {
        dc.MoveTo(W/2-100+i,H-50-corr[i]);
        dc.LineTo(W/2-100+i-1,H-50-corr[i-1]);
  }
  dc.MoveTo(W/2+100,H-50);
  dc.LineTo(W,H-50);
  p.DeleteObject();
  p.CreatePen(PS_SOLID,1,RGB(0,0,0));
  op=dc.SelectObject(&p);
  dc.MoveTo(0,H/4);
  dc.LineTo(W,H/4);
  dc.MoveTo(0,H/2);
  dc.LineTo(W,H/2);
  dc.MoveTo(0,H-50);
  dc.LineTo(W,H-50);
  dc.SelectObject(op);
  delete [] s1;
  delete [] s2;
  delete [] corr;
}
```

Результат работу программы представлен на рис. 3.29. Из рисунка видно, что максимальным значение функции корреляции является при отсутствии сдвига между сигналами.

а *б*

Рис. 3.29.

Функция-обработчик нажатия на кнопку Barker 2 отличается от предыдущего тем, что один из сигналов инвертирован:

```
int bark1[11]={-1,1,1,1,-1,-1,-1,1,-1,-1,1};
int bark2[11]={1,-1,-1,-1,1,1,1,-1,1,1,-1};
```

Для поиска сигнала в смеси с шумом (реализация корреляционного проиемника), в обработчик нажатия на кнопку Barker 1 + Noise необходимо добавить следующий код:

```
void CCorrelationDlg::OnButton5()
{
 CClientDC dc(&m_static);
 CRect r;
 m_static.GetClientRect(&r);
 int W=r.Width();
 int H=r.Height();
 int *s1=new int[W];
 int *s2=new int[W];
 const int T=110;
 memset(s1,0,sizeof(int)*W);
 memset(s2,0,sizeof(int)*W);
 int pos1=rand()%100;
 int pos2=rand()%100+200;
 int index=0;
 int bark1[11]={1,1,1,-1,-1,-1,1,-1,-1,1,-1};
 int bark2[11]={1,1,1,-1,-1,-1,1,-1,-1,1,-1};
 int sb1[110]={0};
 int sb2[110]={0};
 for(int k=0;k<11;k++)
 {
        for(int i=0;i<10;i++)
        {
                sb1[8*k+i]=10*bark1[k];
                sb2[8*k+i]=10*bark2[k];
        }
 }
 int MaxAutoCorr=0;
 for(k=0;k<110;k++) MaxAutoCorr=MaxAutoCorr+sb1[k]*sb1[k];
 for(int i=0;i<W;i++)
 {
        s1[i]=rand()%50-25;

        if((i>=pos1) && (i<(pos1+110)))
        {
                s2[i]=sb1[i-pos1];
        }
        else if((i>=pos2) && (i<(pos2+110)))
        {
                s2[i]=sb2[i-pos2];
        }
        else s2[i]=0;
        s1[i]=s1[i]+s2[i];
 }
 dc.FillRect(r,&CBrush(RGB(255,255,255)));
```

```
CPen p, *op;
p.CreatePen(PS_SOLID,2,RGB(0,0,255));
op=dc.SelectObject(&p);
for(i=1;i<W;i++)
{
	dc.MoveTo(i,H/4-s1[i]);
	dc.LineTo(i-1,H/4-s1[i-1]);
}
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(255,0,0));
op=dc.SelectObject(&p);
for(i=1;i<W;i++)
{
	dc.MoveTo(i,H/2-s2[i]);
	dc.LineTo(i-1,H/2-s2[i-1]);
}
int *corr=new int[W];
memset(corr,0,sizeof(int)*W);
int n;
int tmp=0;
for(n=0;n<(W-T);n++)
{
	for(int m=0;m<T;m++)
	{
		tmp=tmp+s1[n+m]*sb1[m];
	}
	corr[n]=tmp;
	tmp=0;
}
p.DeleteObject();
p.CreatePen(PS_DASH,1,RGB(0,0,0));
op=dc.SelectObject(&p);
int Delta=3000;
bool is_find=false;
for(i=1;i<W;i++)
{
	if((abs(corr[i]-MaxAutoCorr)<Delta) && (is_find==false))
	{
		dc.MoveTo(i,0);
		dc.LineTo(i,H);
		is_find=true;
	}
if((corr[i]<(MaxAutoCorr-2*Delta)) && (is_find==true)) is_find=false;
}
p.DeleteObject();
p.CreatePen(PS_SOLID,2,RGB(200,0,200));
op=dc.SelectObject(&p);
for(i=1;i<W;i++)
{
	dc.MoveTo(i,H-50-corr[i]/200);
```

```
            dc.LineTo(i-1,H-50-corr[i-1]/200);
    }
    p.DeleteObject();
    p.CreatePen(PS_SOLID,1,RGB(0,0,0));
    op=dc.SelectObject(&p);
    dc.MoveTo(0,H/4);
    dc.LineTo(W,H/4);
    dc.MoveTo(0,H/2);
    dc.LineTo(W,H/2);
    dc.MoveTo(0,H-50);
    dc.LineTo(W,H-50);
    dc.SelectObject(op);
    delete [] s1;
    delete [] s2;
    delete [] corr;
}
```

Функция-обработчик нажатия на кнопку Barker 2 + Noise отличается от предыдущей тем, что один из сигналов Баркера инвертирован:

```
int bark1[11]={-1,1,1,1,-1,-1,-1,1,-1,-1,1};
int bark2[11]={1,-1,-1,-1,1,1,1,-1,1,1,-1};
```

На рис. 3.30 представлен результат работы программы. По сильно зашумленному каналу пердается цифровая последовательность. Ставится задача обнаружения этой последовательности. В результате работы алгоритма определено начало цфровой последовательноти (обозначено пунктирной линией).

а *б*

Рис. 3.30

При смещении x_2 влево сигналы уже не перекрываются, и данные в конце последовательностей не формируют парные произведения – возникает так называемый краевой эффект. Одно из возможных решений возникшей

проблемы заключается в том, чтобы длину одной последовательности сделать в два раза больше длины, необходимой для нахождения корреляции. Для этого можно записать больше данных или, если одна из последовательностей периодична, повторить последовательность (особое внимание следует обратить на согласование краев).

Задачи для самостоятельного решения

1. Модифицировать приведенную программу так, чтобы появилась возможность построения зависимости фазы гармонических составляющих сигнала от частоты.

2. Написать программу для микроконтроллера или компьютера, которая выполняет ДПФ и выводит результат на экран. Размер массива данных равен 128 (максимальная амплитуда сигнала равна 64 усл. ед.). количество спектральных составляющих равно 64.

3. Исследовать зависимость спектра прямоугольного импульса от его скважности.

4. Построить спектры (амплитудный и фазовый) треугольного симметричного сигнала та пилообразного сигнала.

5. Разработать программу построения функции автокорреляции для сигнала, указанного в табл. 3.4:

Табл. 3.4.

№	Форма сигналу
1	Полуволна синусоиды
2	Пилообразный импульс
3	Прямоугольный импульс
4	Трапециевидный импульс
5	Случайный сигнал

[*] *Количество выборок сигнала равно 100.*

Чему равна энергия сигнала? Какое максимальное значение функции автокорреляции?

6. Разработать программу вычисления функции корреляции двух сигналов из табл. 3.4, которые находятся в соседних ячейках. Определить максимальное значение функции взаимной корреляции.

7. Разработать программу, выполняющую поиск в зашумленном сигнале случайной цифровой последовательности. Исследовать и сравнить эффективность обнаружения сигналов Баркера, М-последовательности, произвольного цифрового 8-разрядного кода.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ РАЗЛИЧНЫХ ПРИНЦИПОВ МОДУЛЯЦИИ СИГНАЛОВ

4.1. Амплитудная модуляция

Амплитудная модуляция (amplitude modulation, AM) исторически была первым видом модуляции, освоенным на практике. В настоящее время AM применяется в основном только для радиовещания на сравнительно низких частотах (не выше коротких волн) и для передачи изображения в телевизионном вещании. Это обусловлено низким КПД использования энергии модулированных сигналов.

При амплитудной модуляции мгновенная амплитуда несущего колебания:

$$A(t) = A_0 + as_c(t), \qquad (4.1)$$

где A_0 – амплитуда несущей; a – коэффициент пропорциональности, выбираемый так, чтобы амплитуда $A(t)$ всегда была положительной. Частота и фаза несущего гармонического колебания при AM остаются неизменными.

Для математического описания AM сигнала в (4.1) вместо коэффициента a, зависящего от конкретной схемы модулятора, вводится индекс модуляции:

$$m_{AM} = \frac{A_{\max} - A_{\min}}{A_{\max} + A_{\min}},$$

т.е. отношение разности между максимальным и минимальным значениями амплитуд AM сигнала к сумме этих значений. Для симметричного модулирующего сигнала $s_c(t)$ AM сигнал также симметричный, т.е. $A_{\max} = A_{\min} = 2\Delta A$. Тогда индекс модуляции равен отношению максимального приращения амплитуды, к амплитуде несущей.

$$m_{AM} = \frac{\Delta A}{A_0}.$$

Физически индекс модуляции характеризует собой глубину амплитудной модуляции и может изменяться в пределах $0 \le m_{AM} \le 1$.

Таким образом для любого AM сигнала справедливо:

$$S_{AM}(s_c, t) = A_0\left[1 + m_{AM} s_c(t)\right]\cos(\omega_0 t + \varphi_0).$$

Амплитудная модуляция гармоническим колебанием. В простейшем случае модулирующий сигнал является гармоническим колебанием с частотой $\Omega << \omega_0$. При этом выражение

$$S_{AM}(s_c, t) = A_0\left[1 + m_{AM}\cos\Omega t\right]\cos(\omega_0 t + \varphi_0),$$

соответствует однотональному AM сигналу, представленному на рис. 4.1.

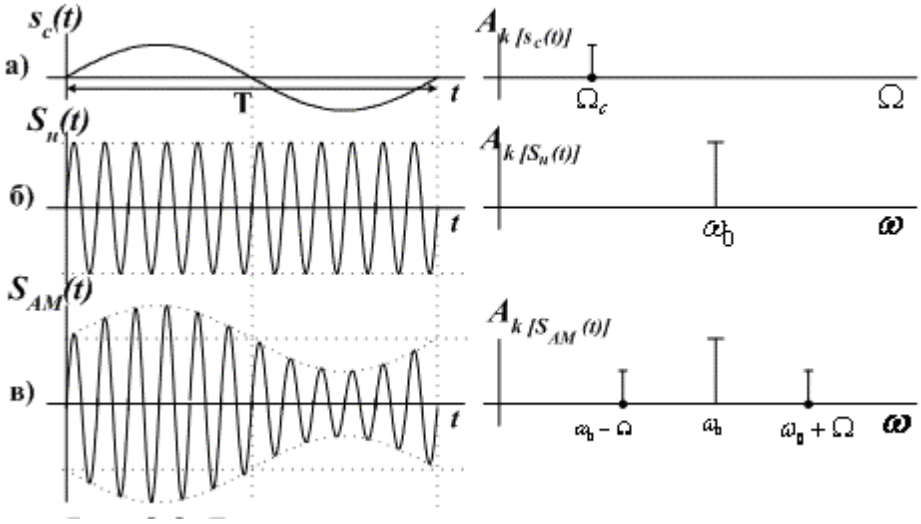


Рис. 4.1.

Однотональный АМ сигнал можно представить в виде суммы трех гармонических составляющих с частотами: ω_0 – несущей; $\omega_0 + \Omega$ – верхней боковой и $\omega_0 - \Omega$ – нижней боковой:

$$S_{AM}(s_c, t) = A_0 \cos(\omega_0 t + \Phi_0) + \frac{A_0 m_{AM}}{2} \cos[(\omega_0 + \Omega)t + \Phi_0 + \varphi_0] + \frac{A_0 m_{AM}}{2} \cos[(\omega_0 - \Omega)t - \Phi_0 - \varphi_0].$$

Спектральная диаграмма однотонального АМ сигнала симметрична относительно несущей частоты ω_0 (рис. 4.1, в). Амплитуды боковых колебаний с частотами $\omega_0 - \Omega$ и $\omega_0 + \Omega$ одинаковы, и даже при $m_{AM} = 1$ не превышают половины амплитуды несущего колебания A_0.

Гармонические модулирующие сигналы и соответственно однотональный АМ сигнал на практике встречаются редко. В большинстве случаев модулирующие первичные сигналы $s_c(t)$ являются сложными функциями времени (рис. 4.2, а). Любой сложный сигнал $s_c(t)$ можно представить в виде конечной или бесконечной суммы гармонических составляющих, воспользовавшись рядом или интегралом Фурье. Каждая гармоническая составляющая сигнала $s_c(t)$ с частотой Ω_i приведет к появлению в АМ сигнале двух боковых составляющих с частотами $\omega_0 \pm \Omega_i$.

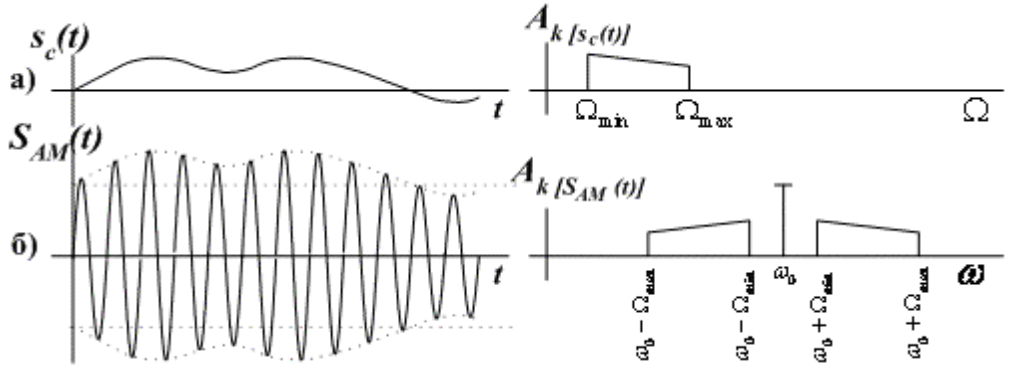


Рис. 4.2.

Множеству гармонических составляющих в модулирующем сигнале с частотами $\Omega_i, i = 1,2,..., N$ будет соответствовать множество боковых составляющих с частотами $\omega_0 \pm \Omega_i, i = 1,2,..., N$. Для наглядности такое преобразование спектра при АМ показано на рис. 4.2, б. Спектр сложномодулированного АМ сигнала, помимо несущего колебания с частотой ω_0, содержит группы верхних и нижних боковых колебаний, образующих соответственно верхнюю боковую полосу и нижнюю боковую полосу АМ сигнала.

При этом верхняя боковая полоса частот является масштабной копией спектра информационного сигнала, сдвинутого в область высоких частот на величину ω_0. Нижняя боковая полоса частот также повторяет спектральную диаграмму сигнала $s_c(t)$, но частоты в ней располагаются в зеркальном порядке относительно несущей частоты ω_0.

Ширина спектра АМ сигнала Δm_{AM} равна удвоенному значению наиболее высокой частоты $\Omega_{\max}$ спектра модулирующего низкочастотного сигнала, т.е. $\Delta\omega_{AM} = 2\Omega_{\max}$.

Наличие двух боковых полос обусловливает расширение занимаемой полосы частот примерно в два раза, по сравнению со спектром информационного сигнала. Мощность, приходящаяся на колебание несущей частоты, постоянна. Мощность, заключенная в боковых полосах, зависит от индекса модуляции и увеличивается с увеличением глубины модуляции. Однако, даже в крайнем случае, когда $m_{AM} = 1$, только 1/3 всей мощности колебания приходится на две боковые полосы.

На рис. 4.3 представлена структурная схема амплитудного модулятора с перемножением.

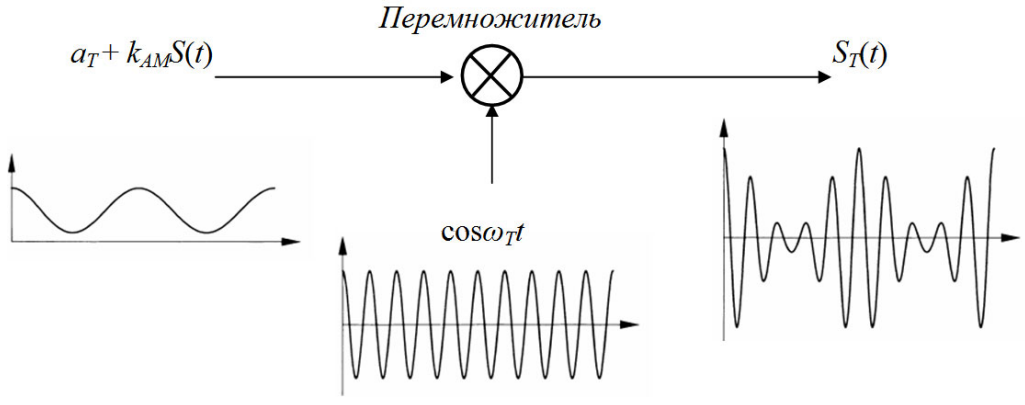


Рис. 4.3.

В среде Visual Studio создаем проект на основе диалогового приложения. В редакторе ресурсов проектируем интерфейс приложения, как показано на рис. 4.4.

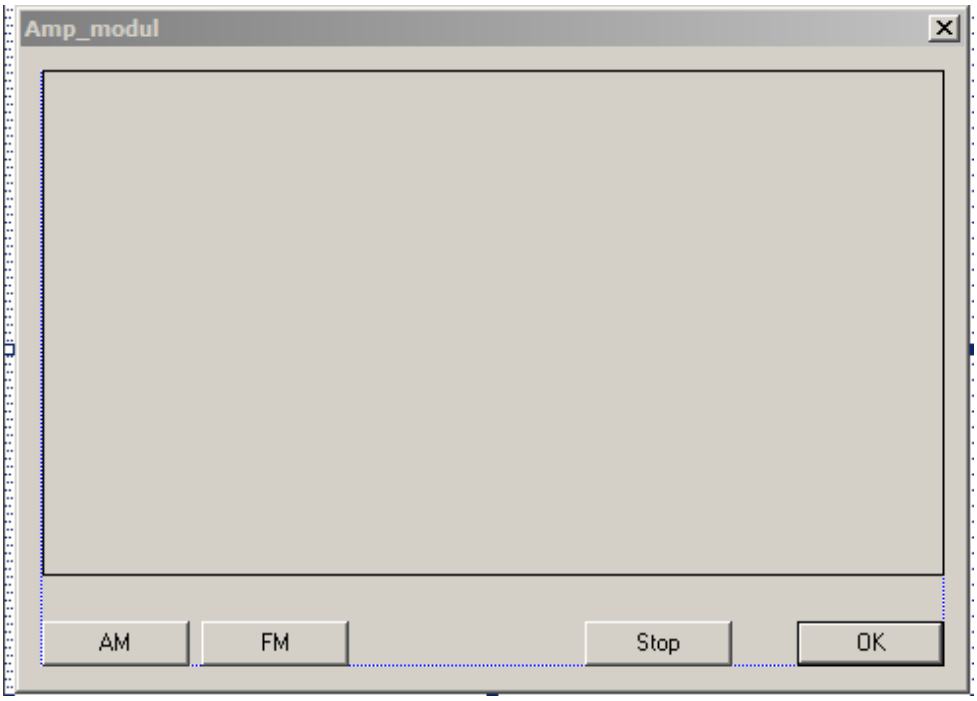


Рис. 4.4.

С элементом управления Static Text связываем управляющую переменную. Для этого используется Class Wizard, окно которого показано на рис. 4.5.

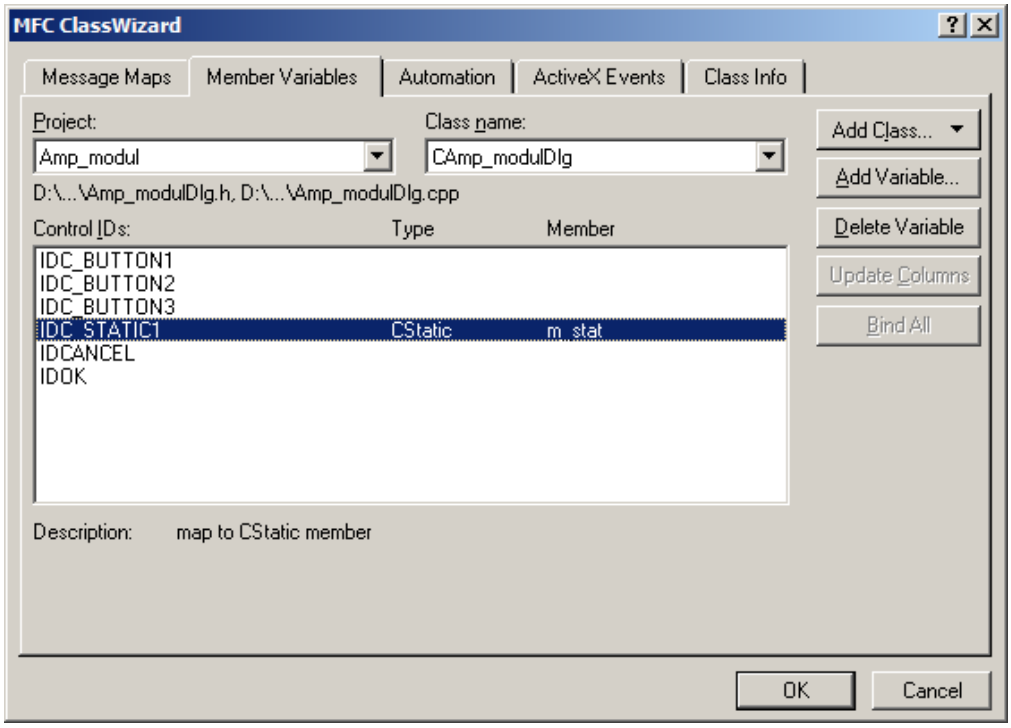


Рис. 4.5.

В класс диалога добавляем следующие переменные:

```
int m_x;
int m_t;
int F;
```

Реализация функции-обработчика нажатия на кнопку AM, в которой выполняется запуск таймера для формирования однотональной амплитудной модуляции и отображение модулированного сигнала, представлена ниже.

```
void CAmp_modulDlg::OnButton1()
{
  m_stat.RedrawWindow();
  SetTimer(1,50,NULL);
  m_x=0;
}
```

Реализация функция-обработчик таймера представлена ниже.

```
void CAmp_modulDlg::OnTimer(UINT nIDEvent)
{
  if(nIDEvent==1)
  {
  CRect r;
  m_stat.GetClientRect(&r);
  if(m_x>=r.Width())
  {
          m_x=1;
          KillTimer(1);
  }
  else m_x++;

  CClientDC dc(&m_stat);
  int y1=r.Height()/2-r.Height()/5*(sin(6.28*m_x/10))*(1+0.9*sin(6.28*m_x/100));
  int y0;
y0=r.Height()/2-r.Height()/5*(sin(6.28*(m_x-1)/10))*
(1+0.9*sin(6.28*(m_x-1)/100));
  dc.MoveTo(m_x,y1);
  dc.LineTo(m_x-1,y0);
  CPen p;
  p.CreatePen(PS_SOLID,2,RGB(0,0,100));
  dc.SelectObject(&p);
  dc.MoveTo(m_x,r.Height()/4-r.Height()/6*sin(6.28*m_x/100));
  dc.LineTo(m_x-1,r.Height()/4-r.Height()/6*sin(6.28*(m_x-1)/100));
  }
}
```

Результат выполнения программы представлен на рис. 4.6.

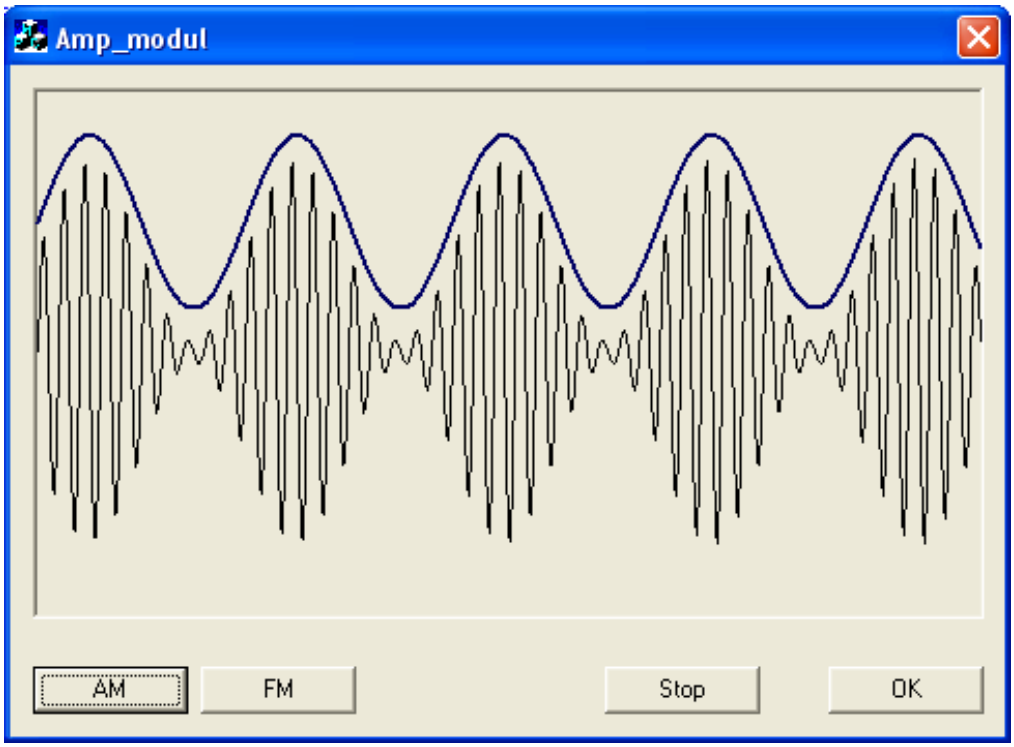


Рис. 4.6.

4.2. Фазовая и частотная модуляция

Фазовая модуляция – процесс изменения мгновенной фазы несущего колебания пропорционально изменению непрерывного информационного сигнала:

$$\varphi(t) = \omega_0 t + \Delta\varphi(t) = \omega_0 t + a s_c(t).$$

Таким образом

$$S_{ФМ}(t) = A_0 \cos\left[\omega_0 t + a s_c(t)\right].$$

Максимальное отклонение фазы называется индексом модуляции:

$$a\left|s_c(t)\right|_{\max} = m_{ФМ}.$$

Если модуляция осуществляется гармоническим колебанием (тональная модуляция) $s_c(t) = A_{0\Omega}\cos\Omega t$ с частотой Ω, то

$$S_{ФМ}(t) = A_0 \cos(\omega_0 t + a A_{0\Omega}\cos\Omega t) = A_0 \cos(\omega_0 t + m_{ФМ}\cos\Omega t).$$

Заметим, что индекс модуляции $m_{ФМ} = aA_{0\Omega}$ пропорционален амплитуде модулирующего колебания.

На рис. 4.7 показано, как изменяются мгновенная частота и фаза при тональной фазовой модуляции. Информационный однотональный сигнал $s_c(t) = A_{0\Omega}\cos\Omega t$ (рис. 4.7, а) модулирует несущее колебание $s_н(t)$ (рис. 4.7, б), при этом закон изменения мгновенной фазы несущего колебания $\varphi(t) = \omega_0 t + a s_c(t)$ повторяет закон изменения $s_c(t)$ «косинус» (рис. 4.7, в), т.е. на линейное изменение фазы (пунктир на рисунке) накладывается переменное

приращение $\Delta\varphi(t) = as_c(t)$, а закон изменения мгновенной частоты несущего колебания $\omega(t)$ (рис. 4.7, г) определяется производной:

$$\omega(t) = \frac{d\varphi(t)}{dt} = \frac{d}{dt}(\omega_0 t + aA_{0\Omega}\cos\Omega t) = \omega_0 - aA_{0\Omega}\sin\Omega t\,.$$

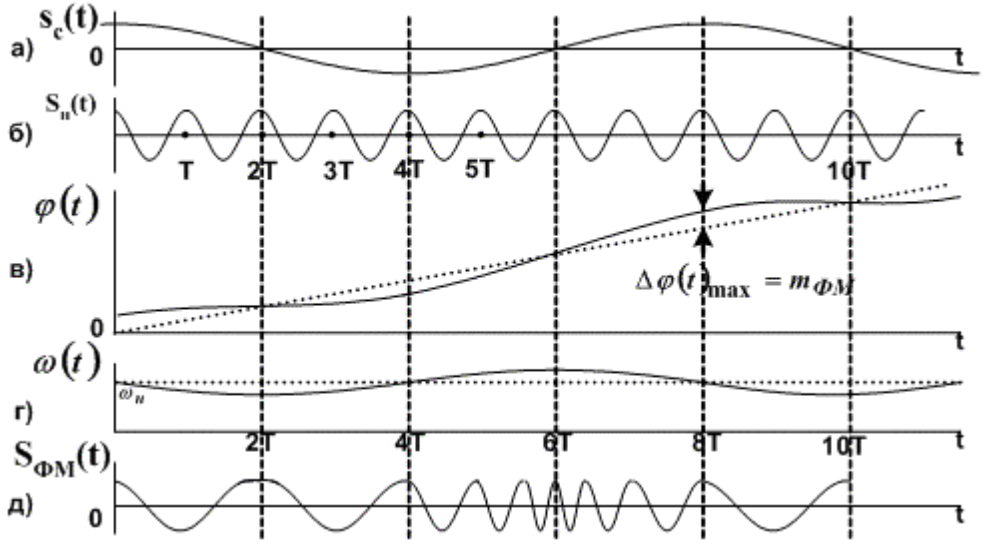


Рис. 4.7.

Фазомодулированное колебание (рис. 4.7, д) построено на основании графика $\omega(t)$; в моменты времени $t = 2T$ и $t = 10T$ сигнал $S_{ФМ}(t)$ имеет минимальную, а в момент $t = 6T$ максимальную мгновенную частоту.

Частотная модуляция – процесс изменения мгновенной частоты несущего колебания в соответствии с изменением информационного сигнала:

$$\omega(t) = \omega_0 + aS_c(t)\,.$$

Рассмотрим наиболее простой способ однотональной частотной модуляции. На рис. 4.8 изображены временные диаграммы изменения мгновенной частоты и фазы для однотональной частотной модуляции.

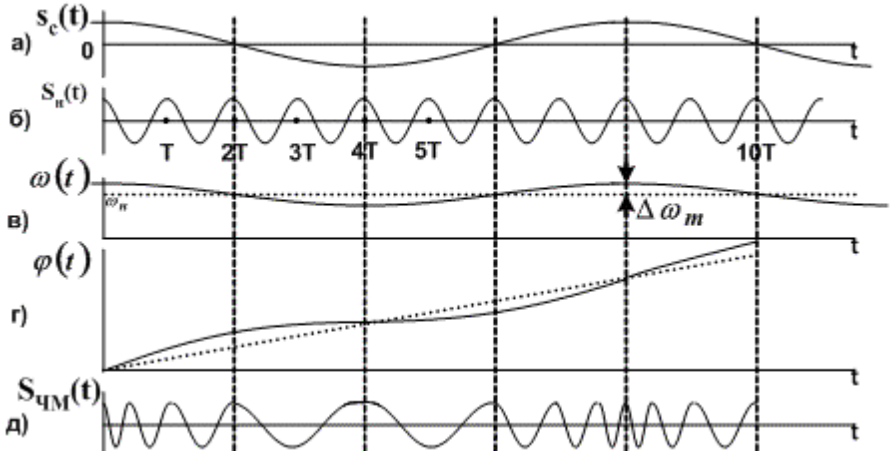


Рис. 4.8.

Информационный однотональный сигнал $s_c(t) = A_0 \cos \Omega t$ (рис. 4.8, а) модулирует несущее колебание $s_н(t)$ (рис. 4.8, б), при этом закон изменения мгновенной частоты несущего колебания $\omega(t) = \omega_0 + aA\cos\Omega t$ повторяет закон изменения $s_c(t)$ (рис. 4.8, в). Здесь $aA_{0\Omega} = \Delta\omega_m$ – девиация частоты, пропорциональная амплитуде модулирующего колебания $A_{0\Omega}$. Девиацией частоты называется максимальное отклонение частоты от среднего значения ω_0:

$$a\left|s_c(t)\right|_{\max} = \Delta\omega_m$$

Отношение девиации частоты $\Delta\omega_m$ к частоте модулирующего колебания Ω называется индексом частотной модуляции:

$$m_{ЧМ} = \frac{\Delta\omega_m}{\Omega}$$

В моменты времени $t = 0$, $t = 8T$ мгновенная частота максимальна, в момент $t = 4T$ – минимальна. Закон изменения мгновенной фазы несущего колебания $\varphi(t)$ (рис. 4.8, г) определяется интегрированием:

$$\varphi(t) = \int_0^t \omega(t)dt\,,$$

$$\varphi(t) = \omega_0 t = m_{ЧМ}\sin\Omega t\,.$$

Учитывая связь частоты и фазы, выражение для частотно-модулированного сигнала запишется следующим образом:

$$S_{ЧМ}(t) = A_0\cos\left[\int_0^t \omega(t)dt\right] = A_0\cos\left[\omega_0 t + a\int_0^t s_c(t)dt\right]$$

Для тональной частотной модуляции эта формула принимает вид:

$$S_{ЧМ}(t) = A_0\cos(\omega_0 t + m_{ЧМ}\sin\Omega t)\,.$$

При ФМ приращение фазы пропорционально модулирующему колебанию $s_c(t)$, а при ЧМ – интегралу от $s_c(t)$. Если сначала проинтегрировать $s_c(t)$, а затем этим колебанием модулировать несущую по фазе, то получится ЧМ сигнал. Такой способ формирования ЧМ сигнала применяется практически. Подобным же образом, если продифференцировать $s_c(t)$ и это колебание использовать для модуляции частоты, то получим ФМ сигнал.

Сигнал с тональной угловой модуляцией (УМ) состоит из бесконечного числа боковых составляющих $\omega_0 \pm m\Omega$, расположенных симметрично относительно несущей частоты ω_0. Амплитуды всех составляющих, в том числе и несущей, определяются величиной индекса угловой модуляции m.

При малых значениях m результирующий вектор УМ-сигнала почти не меняется по величине (в отличие от АМ), но изменяет направление, причем угол отклонения зависит от времени.

При увеличении m увеличиваются амплитуды боковых составляющих высших порядков. Так для m=2 амплитудный спектр сигнала с УМ будет иметь вид, представленный на рис. 4.9.

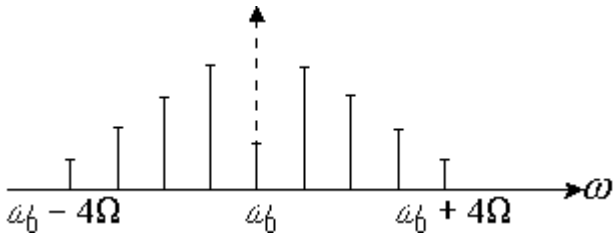


Рис. 4.9.

т.е. полоса частот, занимаемая сигналом, составляет примерно 8Ω (при АМ – 2Ω), а при $m>>1$ она равна уже $2m\Omega = 2\Delta\omega_m$,т.е. при больших индексах угловой модуляции ширина спектра сигнала близка к удвоенной девиации частоты.

Модули спектров ЧМ сигнала при разных индексах модуляции представлены на рис. 4.10.

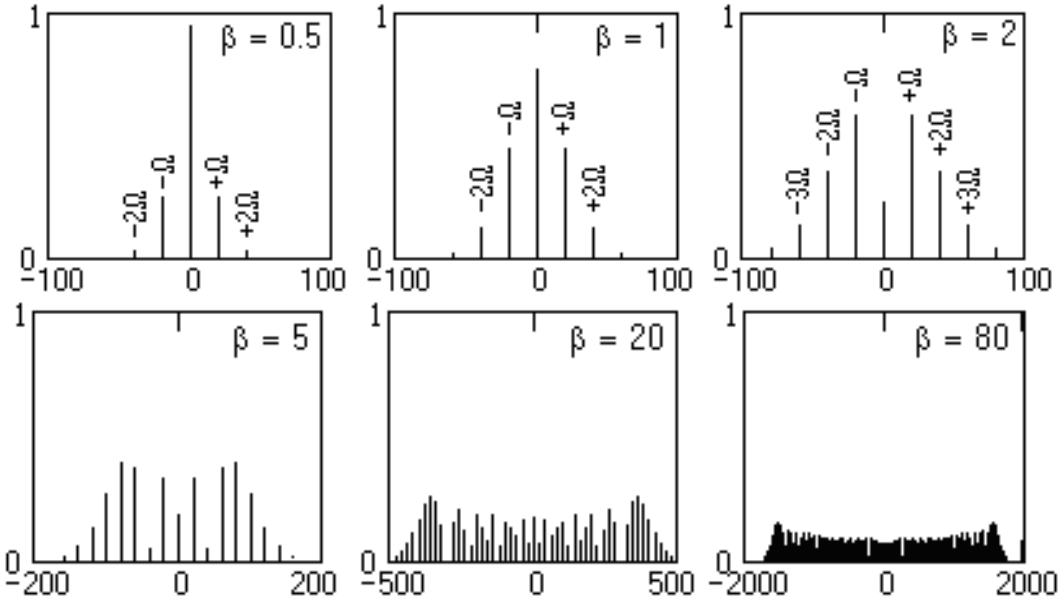


Рис. 4.10.

На рис. 4.11 представлена структурная схема ЧМ-модулятора. На вход подается модулирующий сигнал, который нормируется по амплитуде, так чтобы амплитуда не превышала единицы. Нормированный модулирующий сигнал интегрируется, и усилитель задает, а девиацию частоты ω_∂. Затем формируется комплексная огибающая согласно выражению:

$$I(t) = A_0 \cos(\omega_\partial \cdot \int s_m(t)dt),$$

$$Q(t) = A_0 \sin(\omega_\partial \cdot \int s_m(t)dt),$$

и квадратурный модулятор формирует радиосигнал.

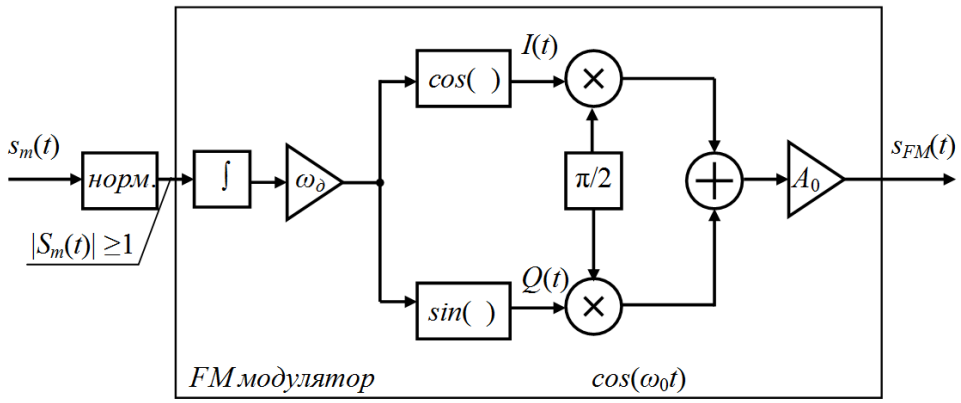


Рис. 4.11.

Если модулирующий сигнал нормирован по амплитуде $|s_m(t)| \leq 1$ тогда формировать ФМ (PM – phase modulation) сигнал можно при помощи ЧМ (FM – frequency modulation) модулятора, а ЧМ (FM) сигнал при помощи ФМ (PM) модулятора, как это показано на рис. 4.12.

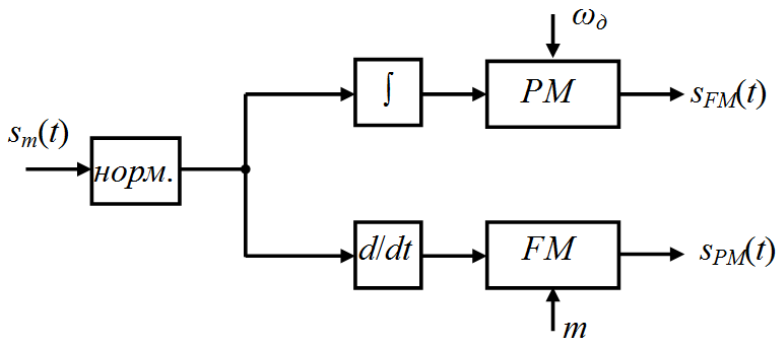


Рис. 4.12.

На рис. 4.13 приведены осциллограммы ФМ и ЧМ сигналов.

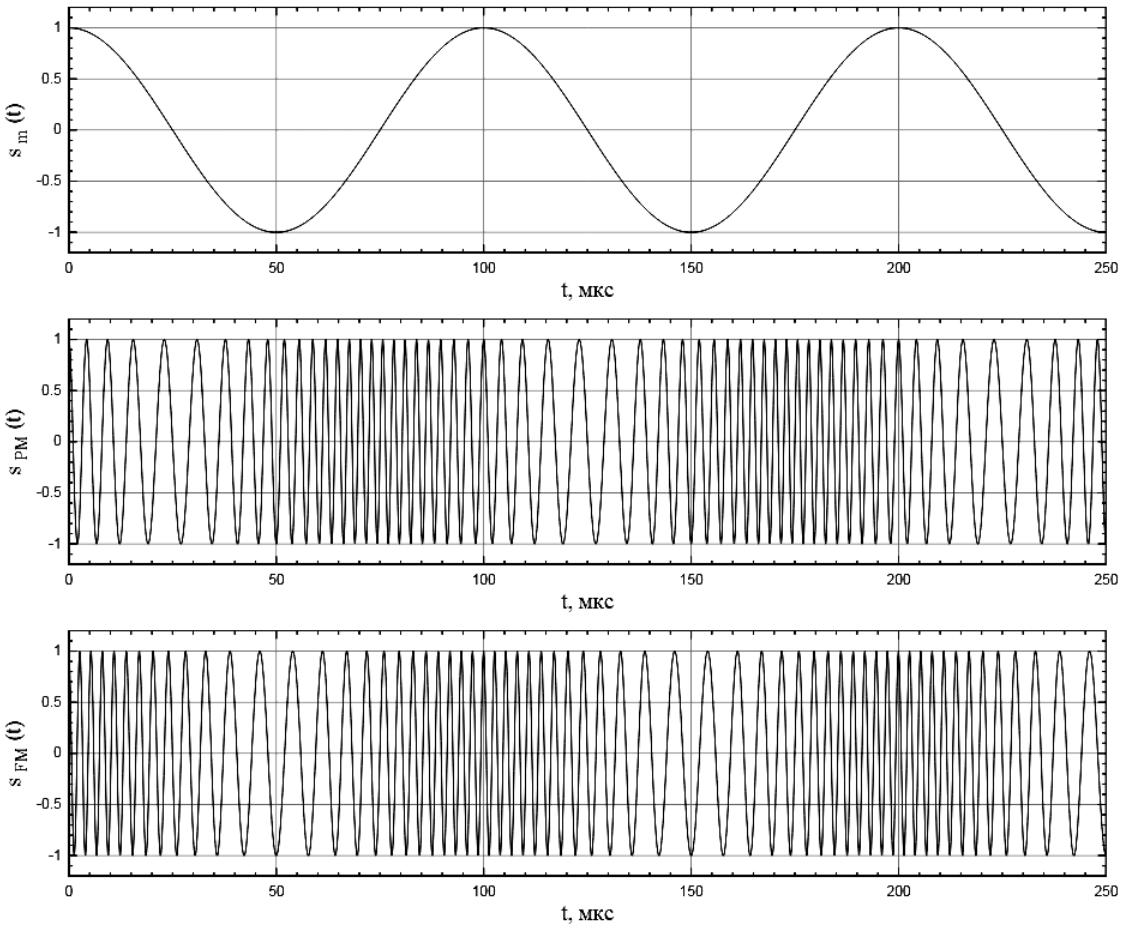


Рис. 4.13.

Из рис. 4.13 следует, что максимальная частота несущего колебания при ФМ будет при максимальной производной модулирующего сигнала (в районе 75 и 175 мкс), а минимальная частота сигнала с ФМ будет при минимальной отрицательной производной модулирующего сигнала (в районе 25, 125 и 225 мкс). При ЧМ максимальная частота сигнала соответствует максимальному значению модулирующего сигнала (в районе 100 и 200 мкс), а минимальная частота будет при минимальном отрицательном значении модулирующего сигнала (в районе 50 и 150 мкс).

Частотная модуляция применяется для высококачественной передачи звукового (низкочастотного) сигнала в радиовещании (в диапазоне УКВ), для звукового сопровождения телевизионных программ, передачи сигналов цветности в телевизионном стандарте SECAM, музыкальных синтезаторах.

Высокое качество кодирования аудиосигнала обусловлено тем, что при ЧМ применяется большая (по сравнению с шириной спектра сигнала АМ) девиация несущего сигнала, а в приёмной аппаратуре используют ограничитель амплитуды радиосигнала для устранения импульсных помех.

Реализация обработчика нажатия на кнопку FM, в котором выполняется запуск таймера для построения ЧМ-сигнала, имеет вид:

```
void CAmp_modulDlg::OnButton3()
{
  m_stat.RedrawWindow();
  SetTimer(2,50,NULL);
  m_x=0;
}
```

В функцию-обработчик таймера для построения ЧМ-сигнала необходимо добавить следующий код:

```
  if(nIDEvent==2)
  {
  CRect r;
  m_stat.GetClientRect(&r);
  if(m_x>=r.Width())
  {
          m_x=1;
          KillTimer(2);
  }
  else m_x++;
  CClientDC dc(&m_stat);
  int T=150;
  double f=40;
  double fmax=100;
  double fmin=20;
  double b=(double)(fmax-fmin)/4;
  if(m_x==0)
  {
  F=f;
  m_t=0;
  }
  if((m_t%F)==0)
  {
  F=f+b*sin(6.28*m_x/T);
  m_t=0;
  }
  int y1=r.Height()/2-r.Height()/3*sin(6.28*m_t/F);
  int y0=r.Height()/2-r.Height()/3*sin(6.28*(m_t+1)/F);
  dc.MoveTo(m_x,y1);
  dc.LineTo(m_x+1,y0);
  CPen p;
  p.CreatePen(PS_SOLID,2,RGB(0,0,100));
  dc.SelectObject(&p);
  dc.MoveTo(m_x, r.Height()/2-b*sin(6.28*m_x/T));
  dc.LineTo(m_x+1, r.Height()/2-b*sin(6.28*(m_x+1)/T));
  m_t++;
  }
```

Функция-обработчик нажатия на кнопку Stop представлена ниже.

```
void CAmp_modulDlg::OnButton2()
{
  KillTimer(1);
  KillTimer(2);
}
```

На рис. 4.14 представлено диалоговое окно программы после запуска в режиме формирования ЧМ-сигнала.

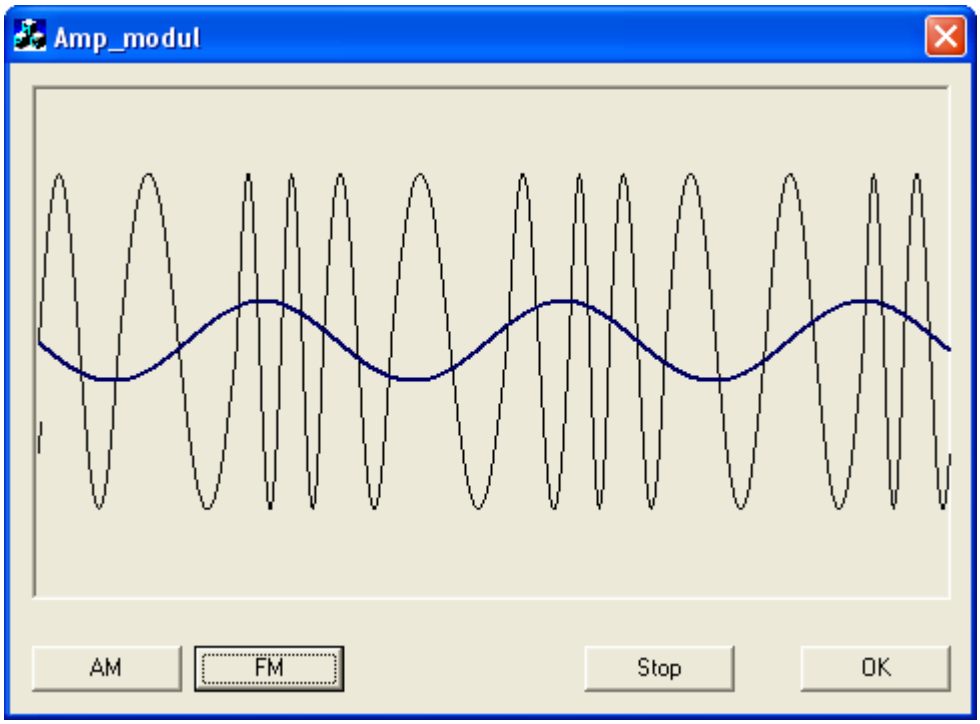


Рис. 4.14.

4.3. Анализ спектров модулированных сигналов

Для исследования спектров сигналов удобно использовать программу SpectraLab. Программа SpectraLab – одна из лучших программ спектрального анализа сигналов. Эта программа разработана для исследований, инженерных, технических и мультимедийных измерений с частотами дискретизации от 24 до 192 кГц, способна производить комплексный анализ сигналов и расширенную пост-обработку, БПФ выполняется с размерами окон до 65535 точек, имеется возможность увеличения и детального просмотра графиков, многоцветных диаграмм в перспективе, карт интенсивности, имеется большой набор инструментов анализа и многие другие возможности.

Программа имеет три различных режима работы и пять различных способов (окон) представления данных.

– В режиме реального времени (Real Time) сигнал берется непосредственно со звуковой карты, программа обрабатывает его и

отображает результаты. Исходный оцифрованный звук не сохраняется в памяти и на диске.

– Режим регистратора (Recorder) позволяет сохранять исходный цифровой звук на жестком диске в формате WAV-файла. Можно также воспроизвести звук с помощью акустической системы, подключенной к звуковой карте.

– Режим пост-обработки (Post-Processing) позволяет обрабатывать звуковые данные, которые были ранее записаны и сохранены на диске в WAV-файле. Этот режим позволяет более гибко проводить анализ, чем предыдущие режимы. В частности, этот режим допускает использование обработки с перекрытием, для того чтобы эффективно растянуть временное разрешение спектрограммы и изображения трехмерной поверхности. Именно в этом режиме выполняется дальнейшая работа с программой.

В программе используются различные окна представления данных:

– Временной ряд (Time Series) показывает оцифрованный звуковой сигнал. Изображение похоже на экран осциллографа (величина сигнала в зависимости от времени).

– Спектр (Spectrum) показывает зависимость величины сигнала от частоты.

– Фаза (Phase) отображает зависимость фазы сигнала от частоты (на рисунке не представлено).

– Спектрограмма (Spectrogram) показывает зависимость спектра от времени. Амплитуда показывается в цвете или полутоновой шкале.

– Трехмерная поверхность (3-D Surface) дает пространственное изображение спектра во времени.

Текст программы, в которой реализованы различные способы модуляции с записью модулированных данных в WAV-файл для анализа в программе SpectraLab, представлен ниже.

```
#include "stdafx.h"
#include "WaveSample.h"
#pragma comment(lib,"winmm.lib")

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CWinApp theApp;
using namespace std;

struct WAVESTRUCT
{
  char formatId[4];
```

```
  int blockLength;
  char soundId[4];
  char subId[4];
  int headerLength;
  short formatType;
  short channels;
  int frequency;
  int dataRate;
  short sampleBytes;
  short sampleWidth;
  char dataId[4];
  int dataLength;
};

int main(int argc, TCHAR* argv[], TCHAR* envp[])
{
  int nRetCode = 0;
  if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
  {
          cerr << _T("Fatal Error: MFC initialization failed") << endl;
          nRetCode = 1;
  }
  else
  {
  // Заполнение массива выборок числовыми данными
  char d=255;
  char data[88400];

  for(int i=0;i<88400;i++)
  {
  //AM
  //        data[i]=128+(64+64*cos(3.14*i/22))*sin(3.14*i/2);
  //        data[i]=128+(32+64*cos(3.14*i/22))*sin(3.14*i/2);
  //PM
  //        data[i]=128+127*cos(3.14*i/2+5*sin(3.14*i/40));
  //FM
  //        data[i]=128+127*cos((3.14+0.0001*cos(3.14*i/40))*i/2);
  //APM
  //        data[i]=128+(64+64*cos(3.14*i/18))*cos(3.14*i/2+0.5*sin(3.14*i/40));
  //        data[i]=128+(64+64*cos(3.14*i/18))*cos(3.14*i/2+8*sin(3.14*i/40));
  // FM b=1
  //        data[i]=128+127*cos((3.14+0.0005*cos(3.14*i/80))*i/2);

  data[i]=128+(3.1*dt)*cos(6.28*i/15)+(16+15*cos(3.14*i/25))*sin(3*3.14*i/10)+
(4+16*cos(3.14*i/50))*sin(6*3.14*i/10);
  }

  WAVESTRUCT sws;
  strcpy(sws.formatId,"RIFF");
  sws.blockLength=sizeof(data)+36;
```

```
  strcpy(sws.soundId,"WAVE");
  strcpy(sws.subId,"fmt ");
  sws.headerLength=16;
  sws.formatType=1;
  sws.channels=1;
  sws.frequency=22100;
  sws.dataRate=22100;
  sws.sampleBytes=1;
  sws.sampleWidth=8;
  strcpy(sws.dataId,"data");
  sws.dataLength=88400;

  CFile nf;
  nf.Open("e:\\test.wav",CFile::modeCreate | CFile::modeWrite);
  nf.Write(&sws,sizeof(sws));
  nf.Write(data,sizeof(data));
  nf.Close();
  //воспроизведение звука
  sndPlaySound((LPCSTR)&sws, SND_MEMORY | SND_ASYNC);
  cout<<"Press any key to exit.\n";
  while(getch()==13);
  }
  return nRetCode;
}
```

Если синусоидальный сигнал проходит через нуль в начале и конце временного ряда, результирующий спектр состоит из одной линии с правильными значениями амплитуды и частоты. С другой стороны, если значение сигнала не равно нулю на одном или обоих концах временного ряда, концы синусоиды будут отсекаться, и оцифрованный сигнал станет разрывным на концах ряда. Этот разрыв приводит к тому, что в результате ДПФ (БПФ) происходит размазывание одной спектральной линии на смежные линии. Этот эффект называется «утечкой», или «растеканием»: энергия сигнала «просачивается» из правильной по частоте позиции в смежные линии спектра.

Утечки можно избежать, если пересечения нулевого значения сигнала синхронизировать с моментами отсчета сигнала, но этого невозможно добиться на практике. Форма спектра при наличии утечки зависит от величины разрыва сигнала, и для реальных сигналов обычно непредсказуема. На рис. 4.15 представлено окно программы SpectraLab при анализе сигнала, представляющего собой сумму двух синусоид близких частот. Результирующий сигнал представляет собой биения, которые внешне напоминают сигнал с однотональной амплитудной модуляцией. Однако анализ спектрального состава показывает, что сигнал состоит из гармоник, расположенных в диапазоне от 400 до 620 Гц. Причиной такого результата является эффект растекания спектра. Исходные 2 гармоники замаскировали

одна другую, и это не позволяет их выделить. При анализе использовано прямоугольное окно:

$$\omega(n) = \begin{cases} 1, n \in [0, N-1] \\ 0, n \notin [0, N-1] \end{cases}.$$

Количество выборок N для анализа равно 1024.

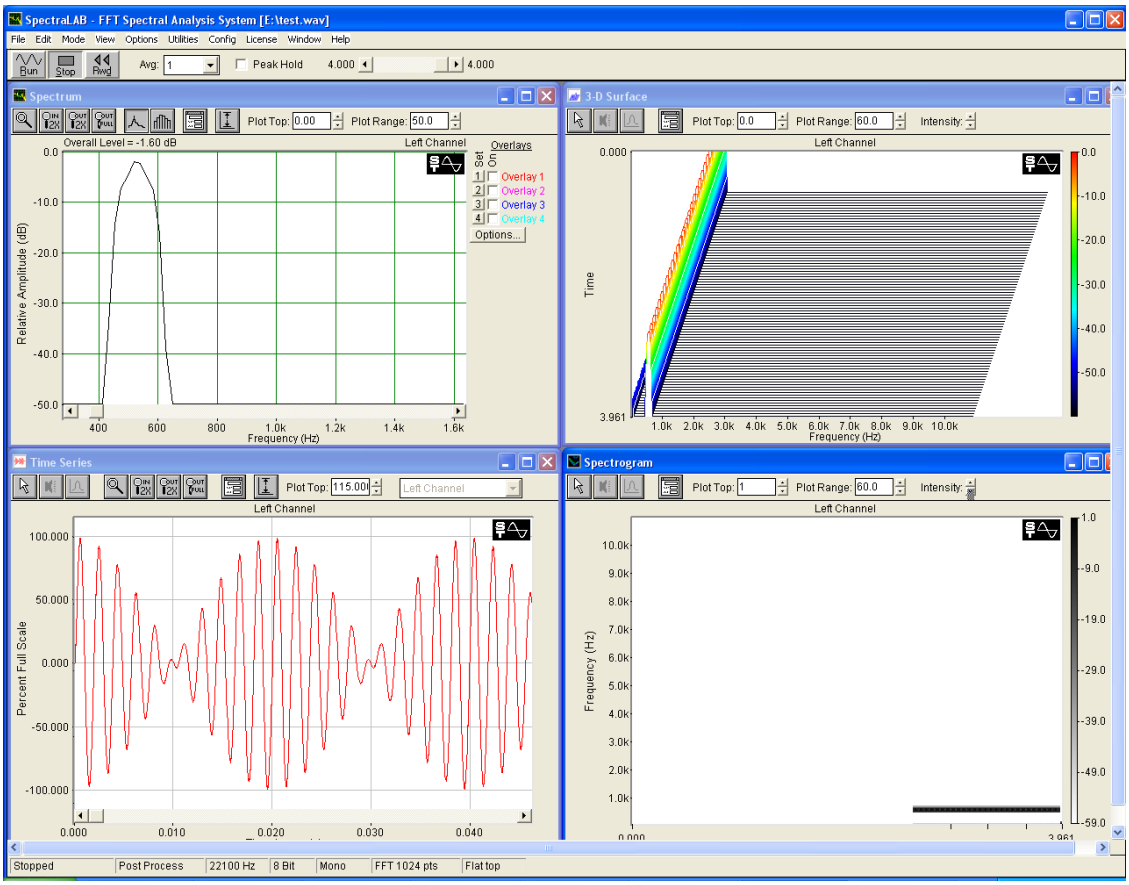


Рис. 4.15.

Для того, чтобы уменьшить эффект утечки, необходимо принудительно установить нулевое значение в начале и в конце временного ряда. Это делается путем умножения временного ряда на весовую функцию, называемую «сглаживающим окном», которая может иметь различную форму. Различие между сглаживающими окнами определяется характером перехода от низких весовых значений на краях к более высоким значениям в середине ряда. Если оконная функция не используется, говорят о «прямоугольном» окне. В табл. 4.1 представлены характеристики различных окон.

Табл. 4.1.

Окно	Разрешение по частоте	Разрешение по амплитуде	Подавление утечки	Применение
Bartlett	достаточное	достаточное	умеренное	
Blackman	достаточное	хорошее	отличное	Измерение искажений
Flattop	плохое	отличное	умеренное	Точные измерения амплитуды
Hamming	достаточное	достаточное	достаточное	Измерение искажений, шума
Hanning	достаточное	отличное	отличное	
Kaiser	достаточное	достаточное	плохое	
Parzen	достаточное	достаточное	плохое	
Triangular	достаточное	достаточное	плохое	
Uniform	отличное	плохое	плохое	Измерение частоты с высоким разрешением, измерение импульсного отклика

В программе SpectraLab настройку процесса анализа можно выполнить в диалоговом окне, внешний вид которого показан на рис. 4.16. Вызвать это окно можно, выбрав в меню Options команду Settings, или нажав на клавишу F4.

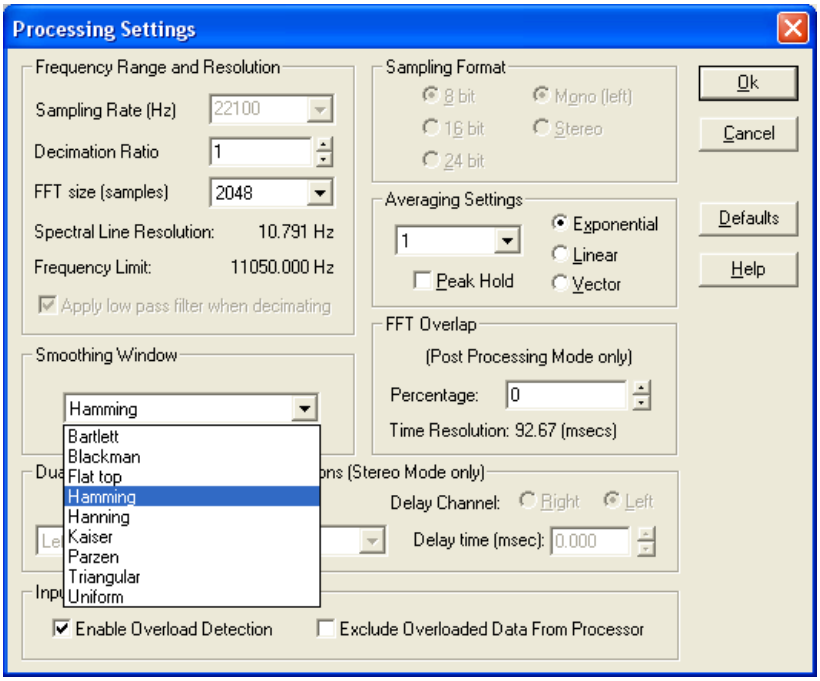


Рис. 4.16.

На рис. 4.17 представлен результат анализа сигнала с настройками параметров, аналогичными приведенным на рис. 4.16.

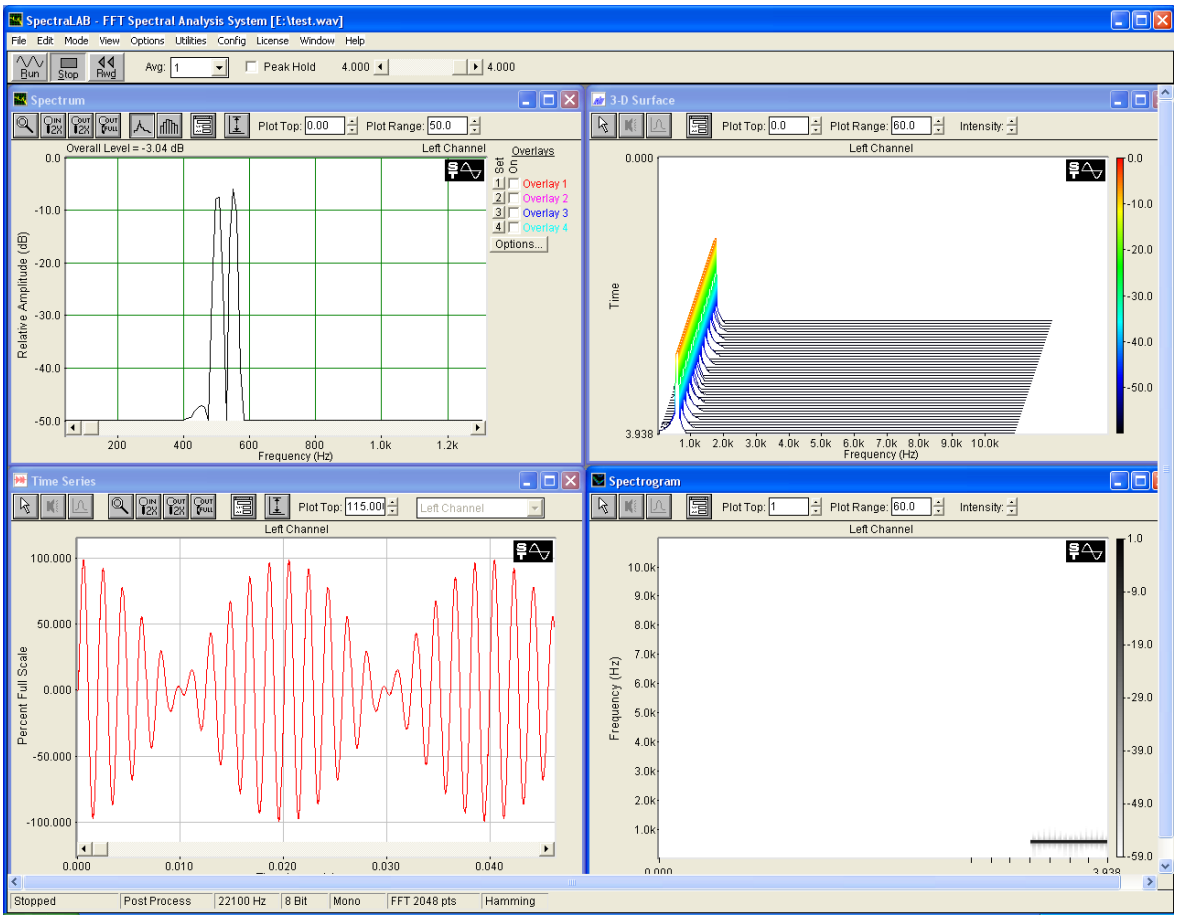


Рис. 4.17.

Выбор окна Хэмминга и изменение размера выборки для спектрального анализа (2048 значений) увеличило разрешающую способность по частоте и уменьшило эффект растекания спектра. По спектру сигнала видно, что он состоит из двух гармоник с близкими частотами. Текущие настройки анализа (количество анализируемых выборок и тип окна) отображаются в нижней части окна программы, в строке состояния.

На рис. 4.18 представлено окно программы SpectraLab с результатами анализа сигнала с однотональной амплитудной модуляцией. При открытии WAV-файла, частота и формат дискретизации устанавливаются в соответствие со значениями, при которых файл был записан. Частота дискретизации составляет 22100 Гц, количество выборок равно 1024, использована оконная функция Хэннинга. Выбранный размер блока данных для БПФ (FFT) непосредственно влияет на разрешение получаемого спектра. Число спектральных линий всегда равно 1/2 от выбранного размера блока данных. Так, 1024-точечное БПФ на выходе дает 512 спектральных линий.

Разрешение по частоте для каждой спектральной линии равно частоте дискретизации, деленной на размер блока данных. Например, если размер блока данных составляет 1024 точек и частота выборки равна 22100 Гц, разрешение каждой спектральной линии будет равно: 22100/1024=21,5 Гц.

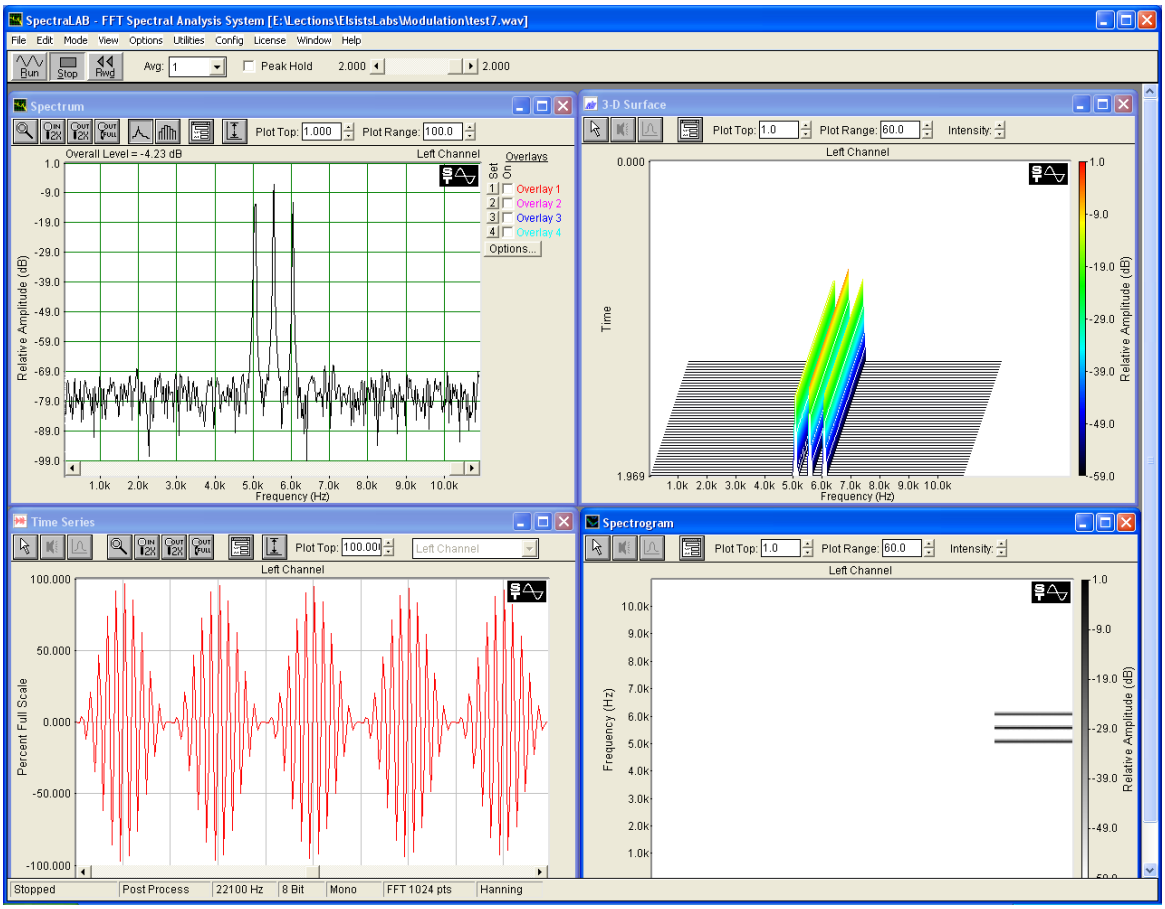


Рис. 4.18.

На рис. 4.19 представлено окно программы SpectraLab с результатами анализа сигнала с однотональной частотной модуляцией. Индекс модуляции равен 0,0001.

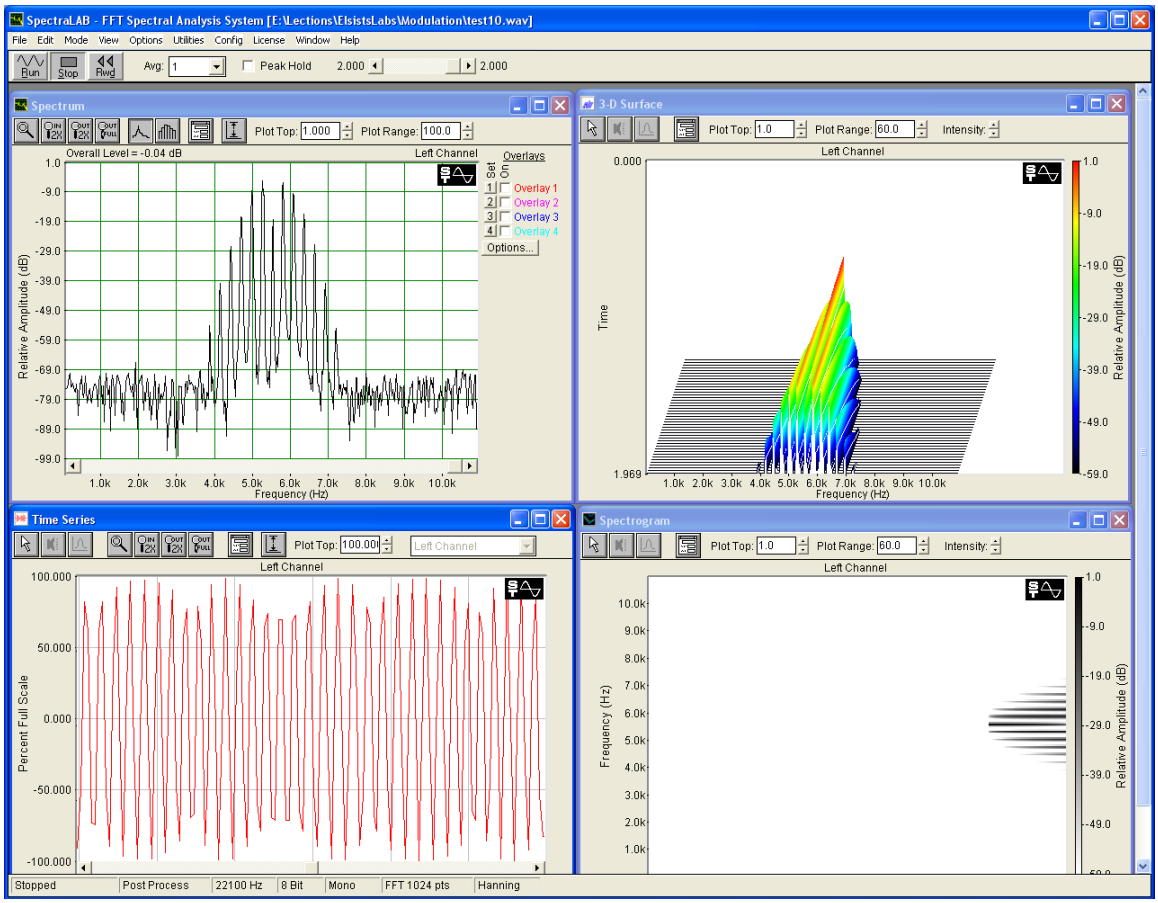


Рис. 4.19.

На рис. 4.20 представлено окно программы SpectraLab с результатами анализа сигнала с однотональной фазовой модуляцией. Индекс модуляции равен 5.

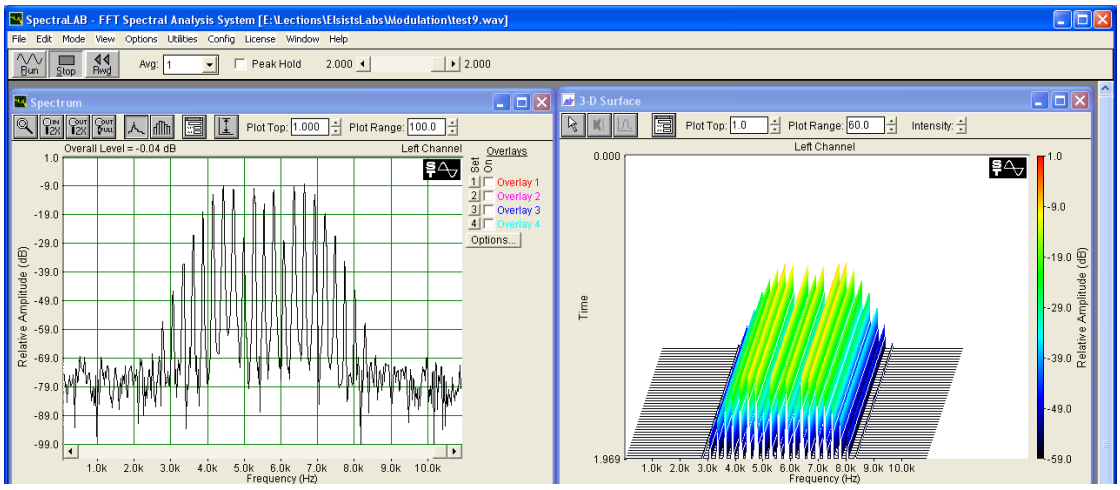


Рис. 4.20.

На рис. 4.21 представлено окно программы SpectraLab с результатами анализа сигнала с квадратурной: $m_{AM} = 1$, $m_{ЧМ} = 0{,}5$.

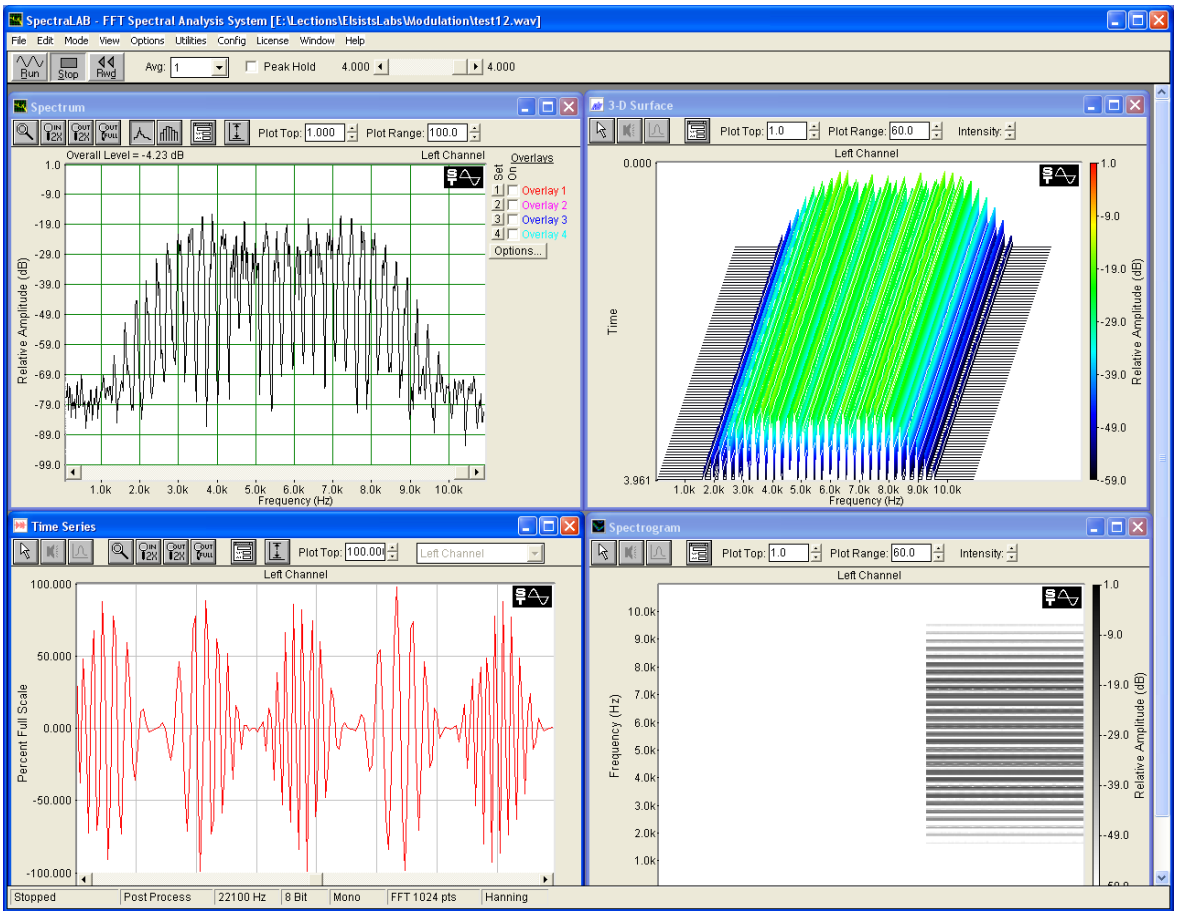


Рис. 4.21.

Анализ результатов работы программы позволил сделать следующие выводы. АМ сигнал формируется путем управления амплитудой несущего колебания по закону модулирующего сигнала. При слишком больших значениях глубины АМ может возникнуть перемодуляция, искажающая модулирующий сигнал. При отсутствии перемодуляции на излучение информации приходится не более 33% мощности сигнала, остальное – излучение несущей. Спектр АМ всегда симметричен относительно несущей при вещественном модулирующем сигнале, и имеет ширину равную удвоенной верхней частоте модулирующего сигнала. Рассмотрены параметры угловой модуляции, девиация частоты и фазы, и показана их взаимосвязь.

4.4. Цифровая модуляция для передачи дискретных сообщений

Цифровой сигнал обладает большим числом преимуществ. Однако при передаче на дальние расстояния (более 100 метров) он начинает терять одно из своих самых важных свойств: помехозащищенность. Это связано с тем, что в качестве среды, как правило, используется воздушное пространство в случае радиопередачи и проводные каналы связи, а цифровой сигнал в этих средах очень быстро затухает. Использовать ретрансляторы через каждые несколько сотен метров при передаче на дальние расстояния экономически неэффективно. Кроме того, это не всегда технически реализуемо, в частности в

сотовых системах связи максимальная удаленность мобильной станции (MS) от базовой станции (BTS) может достигать 35 км. Также есть еще одно важное свойство, требуемое для цифрового канала связи – широкополосность. Цифровой сигнал с резкими переходами между уровнями требует широкой полосы для его передачи. В противном случае переходы между уровнями будут "заламываться" и сигнал будет "смазанным", что может привести к высокому проценту ошибок. Для решения вышеуказанных проблем используют различные методы модуляции цифровых сигналов.

Модуляция – это процесс изменения каких-либо параметров несущего сигнала под действием информационного потока. Данный термин обычно применяют для аналоговых сигналов. Применительно к цифровым сигналам существует другой термин "манипуляция", однако его часто заменяют все тем же словом "модуляция" подразумевая, что речь идет о цифровых сигналах.

Существует 3 основных вида манипуляции сигналов (рис. 4.22): амплитудная (Amplitude-shift keying (ASK)), частотная (Frequency-shift keying (FSK)) и фазовая (Phase-shift keying (PSK)). Этот набор манипуляций определяется основными характеристиками, которыми обладает любой сигнал.

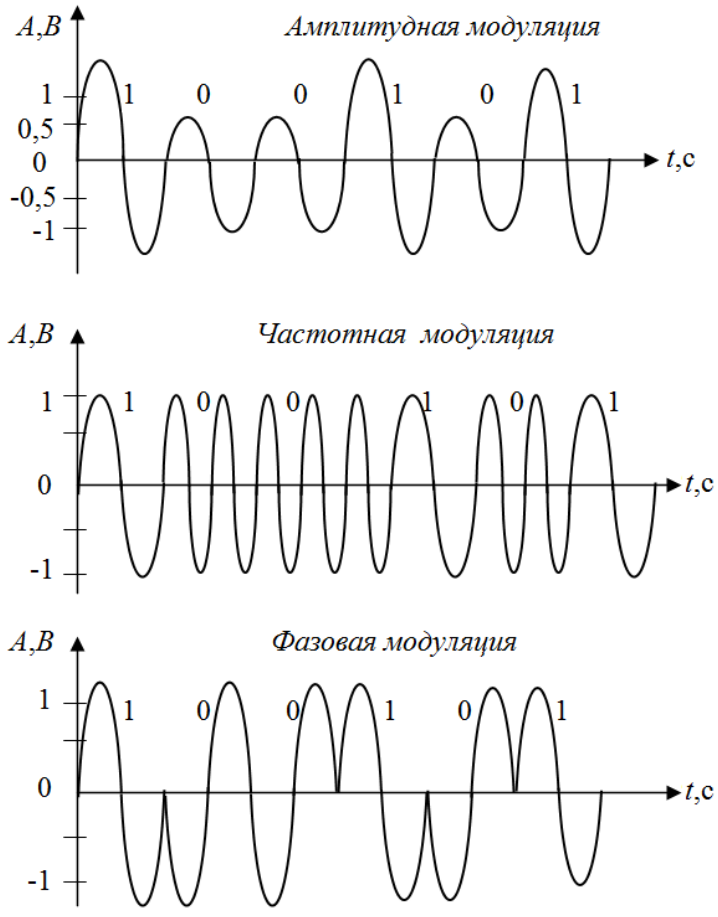


Рис. 4.22.

В среде разработки Visual Studio проектируем диалоговое окно, как показано на рис. 4.23.

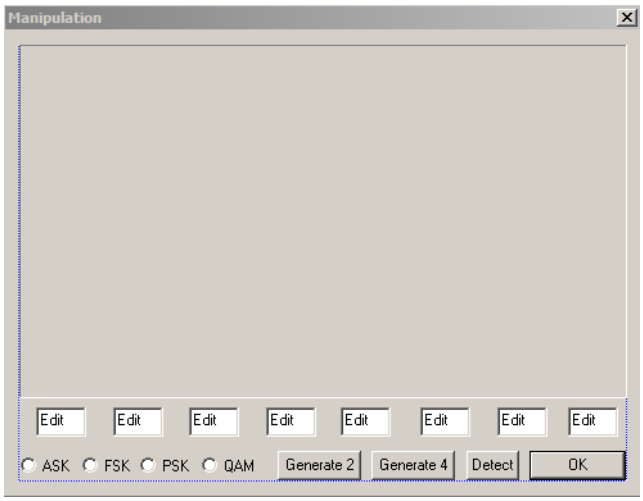


Рис. 4.23.

С элементами управления, расположенными на диалоговом окне, необходимо связать переменные. Имена переменных, а также их тип, показаны на рис. 4.24.

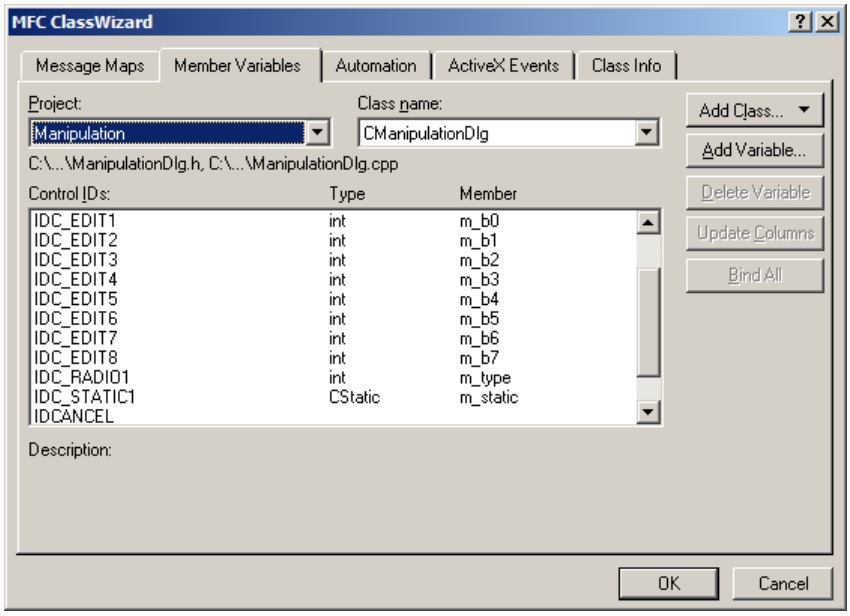


Рис. 4.24.

В класс диалога необходимо добавить 2 переменные:

```
int m_N;        // для хранения размера массива выборок
int * m_d;      // указатель на массив мгновенных значений сигнала
```

В функции OnInitDialog() выполняется инициализация переменных и выделение памяти для хранения мгновенных значений сигнала.

```
CRect r;
m_static.GetClientRect(&r);
```

```
  m_N=r.Width();
  m_d=new int[m_N];
  int s[8]={1,1,0,1,0,1,1,0};
  m_b0=s[0];
  m_b1=s[1];
  m_b2=s[2];
  m_b3=s[3];
  m_b4=s[4];
  m_b5=s[5];
  m_b6=s[6];
  m_b7=s[7];
  m_type=0;
  UpdateData(false);
```

При закрытии окна приложения обязательно следует освободить выделенную память:

```
void CManipulationDlg::OnOK()
{
  delete [] m_d;
  CDialog::OnOK();
}
```

Функция-обработчик нажатия на кнопку Generate 2 представлена ниже.

```
void CManipulationDlg::OnButton1()
{
  UpdateData(true);
  int s[8];
  s[0]=m_b0;
  s[1]=m_b1;
  s[2]=m_b2;
  s[3]=m_b3;
  s[4]=m_b4;
  s[5]=m_b5;
  s[6]=m_b6;
  s[7]=m_b7;
  int k=0;
  for(int i=0;i<m_N;i++)
  {
  if(m_type==0) m_d[i]=50*(s[k]%2+1)*sin(6.28*i/30);
  else if(m_type==1) m_d[i]=100*sin(6.28*i/(15+15*s[k]));
  else if(m_type==2) m_d[i]=100*sin(6.28*i/30+s[k]%2*3.14);
  else if (m_type==3) m_d[i]=50*(s[k]%2+1)*sin(6.28*i/30+s[k]%2*3.14);
  if((i>0) && (i%60==0)) k++;
  }
  CRect r;
  m_static.GetClientRect(&r);
  CClientDC dc(&m_static);
```

```
    int H=r.Height();
    dc.FillRect(&r, &CBrush(RGB(255,255,255)));
    CString str;
    for(i=0;i<=m_N/60;i++)
    {
            dc.MoveTo(i*60,0);
            dc.LineTo(i*60,H);
            str.Format("%d",s[i]);
            dc.TextOut(i*60+25,10,str);
    }
    CPen p, *op;
    p.CreatePen(PS_SOLID,1,RGB(255,0,0));
    op=dc.SelectObject(&p);

    for(i=1;i<m_N;i++)
    {
            dc.MoveTo(i,H/2-m_d[i]);
            dc.LineTo(i-1,H/2-m_d[i-1]);
    }
    dc.SelectObject(op);
}
```

Амплитудная манипуляция – изменение сигнала, при котором скачкообразно меняется амплитуда несущего колебания. В передатчике этот метод реализуется наиболее просто по сравнению с другими видами манипуляции (в простейшем случае – просто включается и выключается питание передатчика). На рис. 4.26, а, представлено окно программы с осциллограммой сигнала с амплитудной манипуляцией. Телеграфные сигналы – азбуку Морзе – чаще всего передают при помощи амплитудной манипуляции. На рис. 4.25 представлена схема формирования АМн сигнала.

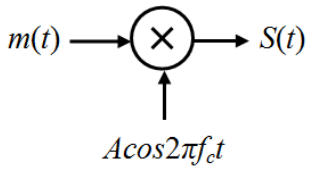


Рис. 4.25.

При частотной манипуляции значениям «0» и «1» информационной последовательности соответствуют определённые частоты синусоидального сигнала при неизменной амплитуде (рис. 4.26, б). Частотная манипуляция весьма помехоустойчива, поскольку помехи телефонного канала искажают в основном амплитуду, а не частоту сигнала. Однако при частотной манипуляции неэкономно расходуется ресурс полосы частот телефонного канала. Поэтому этот вид модуляции применяется в низкоскоростных протоколах, позволяющих осуществлять связь по каналам с низким отношением сигнал/шум.

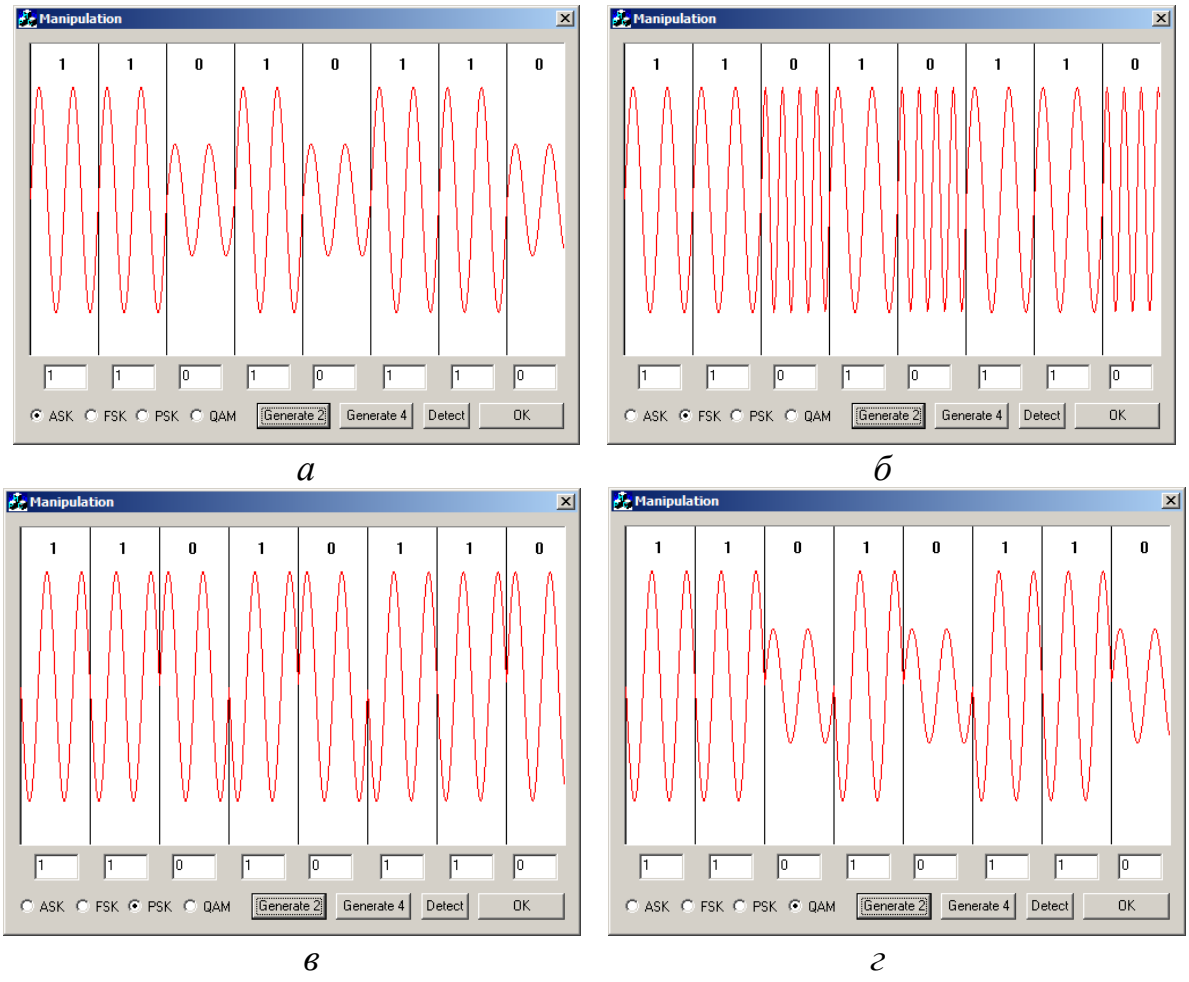


Рис. 4.26.

На рис. 4.27 представлена схема формирования сигнала с двоичной частотной манипуляцией.

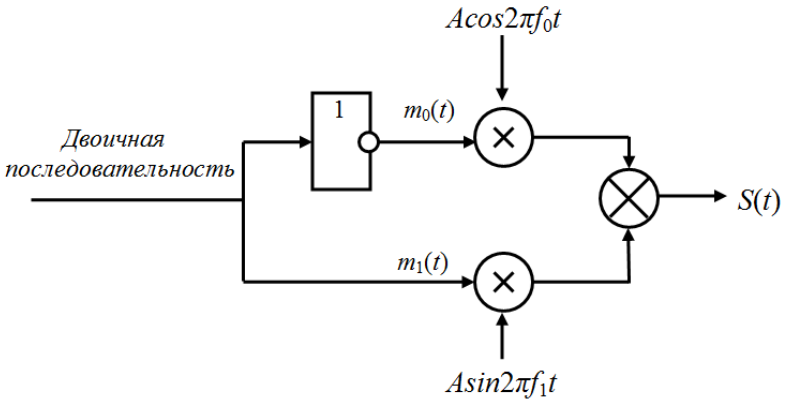


Рис. 4.27.

Фазовая манипуляция (ФМн) – один из видов фазовой модуляции, при которой фаза несущего колебания меняется скачкообразно в зависимости от информационного сообщения и может принимать M значений. Если M=2, то фазовая манипуляция называется двоичной фазовой манипуляцией (BPSK, B-

Binary – 1 бит на 1 смену фазы), если *M*=4 – квадратурной фазовой манипуляцией (QPSK, Q-Quadro – 2 бита на 1 смену фазы). Осциллограмма сигнала с QPSK представлена на рис. 28, в. Здесь: переход 1→0 – скачек фазы, 1→1 – нет скачка фазы 0→0, – нет скачка фазы 0→1 – скачек фазы.

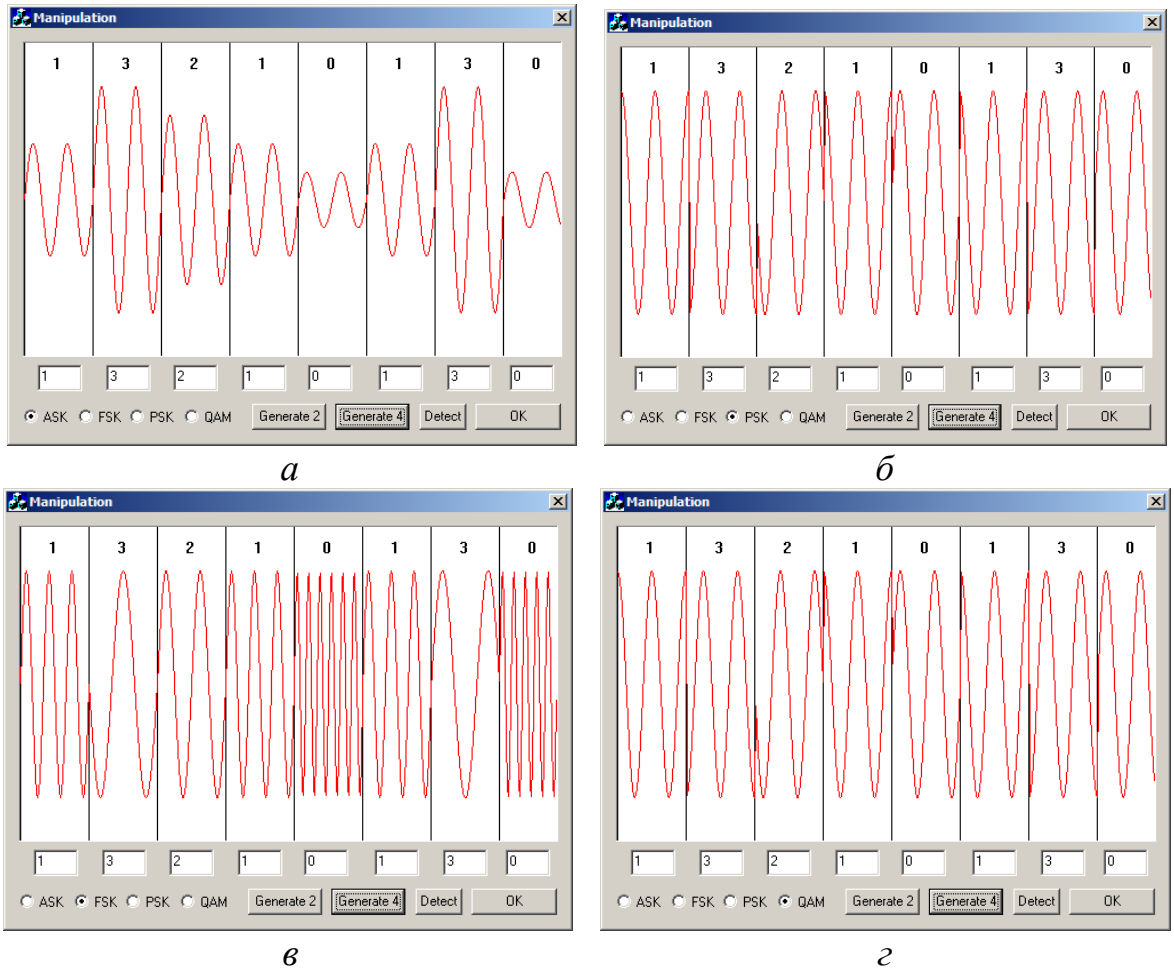
Двоичная фазовая манипуляция – самая простая форма фазовой манипуляции. Работа схемы двоичной ФМн заключается в смещении фазы несущего колебания на одно из двух значений, нуль или 180° (рис. 4.26, в).

Реализация функции-обработчика нажатия на кнопку Generate 4 имеет вид:

```
void CManipulationDlg::OnButton2()
{
  UpdateData(true);
  int s[8];
  s[0]=m_b0;
  s[1]=m_b1;
  s[2]=m_b2;
  s[3]=m_b3;
  s[4]=m_b4;
  s[5]=m_b5;
  s[6]=m_b6;
  s[7]=m_b7;
  int k=0;
  for(int i=0;i<m_N;i++)
  {
          if(m_type==0) m_d[i]=25*(s[k]+1)*sin(6.28*i/30);
          else if(m_type==1) m_d[i]=100*sin(6.28*i/(10+10*s[k]));
          else if(m_type==2) m_d[i]=100*sin(6.28*i/30+s[k]*1.57);
          else if (m_type==3) m_d[i]=100*sin(6.28*i/30+s[k]*1.57);
          if((i>0) && (i%60==0)) k++;
  }
  CRect r;
  m_static.GetClientRect(&r);
  CClientDC dc(&m_static);
  int H=r.Height();
  dc.FillRect(&r, &CBrush(RGB(255,255,255)));
  CString str;
  for(i=0;i<=m_N/60;i++)
  {
          dc.MoveTo(i*60,0);
          dc.LineTo(i*60,H);
          str.Format("%d",s[i]);
          dc.TextOut(i*60+25,10,str);
  }
  CPen p, *op;
  p.CreatePen(PS_SOLID,1,RGB(255,0,0));
  op=dc.SelectObject(&p);
  for(i=1;i<m_N;i++)
```

```
  {
        dc.MoveTo(i,H/2-m_d[i]);
        dc.LineTo(i-1,H/2-m_d[i-1]);
  }
  dc.SelectObject(op);
}
```

При квадратурной фазовой манипуляции используется созвездие из четырёх точек, размещённых на равных расстояниях на окружности. Используя 4 фазы, в QPSK на символ приходится два бита, как показано на рис. 4.28, г. Анализ показывает, что скорость может быть увеличена в два раза относительно BPSK при той же полосе сигнала, либо оставить скорость прежней, но уменьшить полосу вдвое.



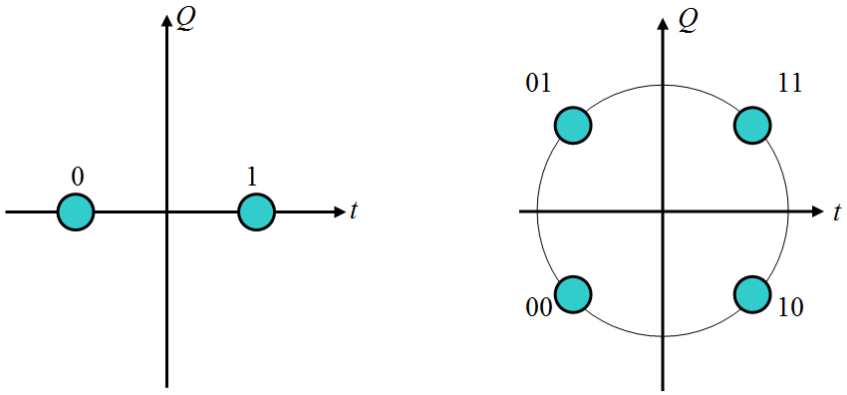
а *б*

в *г*

Рис. 4.28.

Радиосигнал представляется в виде двухмерной точечной диаграммы на комплексной плоскости, точками на которой являются все возможные символы, представленные в геометрической форме. Более абстрактно, на диаграмме отмечены все значения, которые могут быть выбраны данной схемой манипуляции, как точки на комплексной плоскости. Сигнальные

созвездия, полученные в результате измерения радиосигнала, могут использоваться для определения типа манипуляции, рода интерференции и уровня искажений. На рис. 4.29, а, представлено сигнальное созвездие двоичной фазовой манипуляции (BPSK), а на рис. 4.29, б, сигнальное созвездие квадратурной фазовой манипуляции (QPSK).



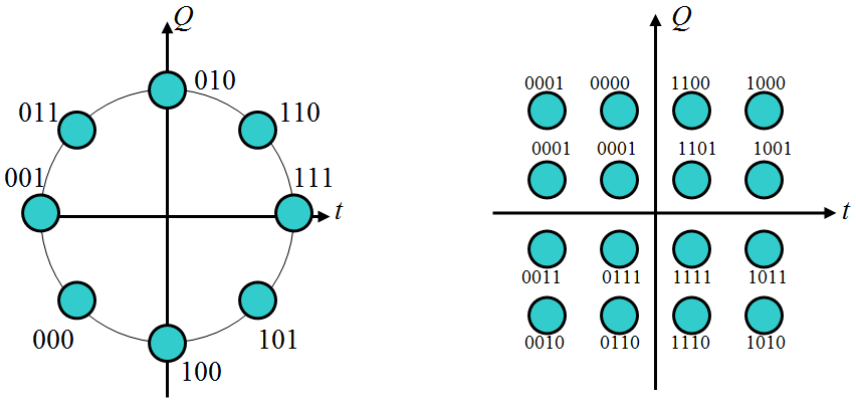
а *б*

Рис. 4.29.

Точки на диаграмме называют сигнальными точками (или точками созвездия). Они представляют множество модулирующих символов, то есть модулирующий алфавит.

Задачи для самостоятельного решения

1. Разработать программу, выполняющую формирование сигнала, состоящего из суммы следующих сигналов:
   – сигнал с однотональной амплитудной модуляцией с частотой несущей 3 кГц, частота модулирующего сигнала 300 Гц;
   – сигнал с однотональной амплитудной модуляцией и подавленой несущей, с частотой несущей 6 кГц, частота модулирующего сигнала 600 Гц;
   – сигнал с однотональной частотной модуляцией, индекс модуляции 0,0001, частота модулирующего сигнала 12 кГц.
2. Модифицировать программу, выполняющую формирование сигнало с ASK, FSK, PSK, QPSK таким образом, чтобы была возможность записи полученного сигнала в WAV-файл. Получить и выполнить анализ спектров этих сигналов в программе SpectraLab.
3. Разработать программу, выполняющую формирование сигнала с квадратурной манипуляцией. Сигнальные созвездия представлены на рис. 4.30.


Q
010
011
110
001
111
t
000
101
100
Q
0001
0000
1100
1000
0001
0001
1101
1001
t
0011
0111
1111
1011
0010
0110
1110
1010

а *б*

Рис. 4.30.

5. АЛГОРИТМЫ ОБРАБОТКИ ЗВУКОВЫХ СИГНАЛОВ

5.1. Частотная коррекция сигналов

Под обработкой звукового сигнала понимается изменение его частотной или фазовой характеристики, сужение или расширение динамического диапазона, применение амплитудной, частотной или фазовой модуляции, удаление шумов, а также создание задержанных по времени затухающих копий этого сигнала. Целью обработки могут быть как чисто технические задачи, такие как согласование параметров сигнала с характеристиками электроакустического тракта, так и художественные, определяемые звукорежиссером, в частности, это могут быть различные звуковые эффекты (тремоло, вибрато, хор, эхо, реверберация и другие).

Обработка звуковых сигналов производится преимущественно в цифровом виде с помощью звуковых процессоров. Программное обеспечение позволяет осуществлять сколь угодно сложные преобразования звуковых сигналов и создавать самые невероятные звуковые эффекты. Очень важно, что для различных манипуляций со звуком не требуется постоянная смена оборудования, достаточно изменить программное обеспечение.

Чтобы собрать все пространственно-временные события и сделать из них музыку, в которой каждая часть идеально подходит к другой, нужно быть немного художником, немного ученым. Необходимо знать физические основы осуществляемых преобразований и уметь грамотно пользоваться оборудованием. Научный аспект работы состоит в том, чтобы знать, как соединить все в единую систему и как управлять параметрами, влияющими на обработку звука. Художественный аспект включается, когда вы принимаете решение, какие эффекты и звуки использовать, каким должен быть баланс и как разместить различные партии в окончательном миксе.

Под частотной коррекцией понимается повышение или понижение уровня спектральных составляющих звуковых сигналов в выбранных полосах с помощью фильтров без внесения новых составляющих спектра. Необходимость серьезной частотной коррекции звуковоспроизводящей аппаратуры наиболее часто обусловлена плохими акустическими характеристиками помещений, где проводится концерт или прослушивается звукозапись. Если, например, в зале имеются ровные твердые поверхности сцены и пола, бетонные или кирпичные стены, жестяная крыша, то все это может начать греметь и дребезжать, а в лучшем случае вокалист и слушатели перестанут понимать слова из-за снижения разборчивости. Серьезные проблемы с качественным восприятием звука возникают и в салоне автомобиля. Аппаратура частотной коррекции звуковых сигналов является связывающим звеном между звучанием звуковоспроизводящей системы и откликом помещения, и она в значительной мере может такие проблемы решить.

Данные, имеющие отношение к мультимедиа (звук, видео) хранятся в файлах в так называемом RIFF-формате (Resource Interchange File Format – формат файла для обмена ресурсами). WAV-файлы, содержащие звук имеют формат RIFF.

В табл. 5.1 представлена структура WAV файла.

Табл. 5.1.

Смещение	Название поля	Описание	Размер
0	formatId	Содержит символы "RIFF" в ASCII кодировке. Является началом RIFF-цепочки.	4
4	blockLength	Это оставшийся размер цепочки, начиная с этой позиции. Иначе говоря, это размер файла - 8, то есть, исключены поля chunkId и chunkSize.	4
8	soundId	Содержит символы "WAVE".	4
12	subId	Содержит символы "fmt ".	4
16	headerLength	16 для формата PCM. Это оставшийся размер подцепочки, начиная с этой позиции.	4
20	formatType	Аудио формат, полный список можно получить здесь. Для PCM = 1 (то есть, Линейное квантование). Значения, отличающиеся от 1, обозначают некоторый формат сжатия.	2
22	channels	Количество каналов. Моно = 1, Стерео = 2.	2
24	frequency	Частота дискретизации: 8000 Гц, 44100 Гц и т.д.	4
28	dataRate	Количество байт, переданных за секунду воспроизведения.	4
32	sampleBytes	Количество байт для одной выборки, включая все каналы.	2
34	sampleWidth	Количество бит в выборке. Так называемая "глубина" или точность звучания. 8 бит, 16 бит и т.д.	2
36	dataId	Содержит символы "data"	4
40	dataLength	Количество байт в области данных.	4
44	data	Непосредственно WAV-данные.	data Length

В моно варианте значения амплитуды расположены последовательно. В стерео сначала располагается значение амплитуды для левого канала, затем для правого. Для монофонического сигнала с дискретностью 8 бит звуковые данные представляют собой массив однобайтовых значений, каждое из которых является выборкой сигнала.

Для стереофонического сигнала с дискретностью 8 бит звуковые данных имеют формат массива двухбайтовых слов, причем младший байт слова соответствует левому каналу, а старший – правому.

Формат звуковых данных с дискретностью 16 бит выглядит аналогично. Для монофонического сигнала данные хранятся в массиве 16-битовых слов.

Для стереофонического используется массив двойных слов, причем младшему слову соответствует левый канал, а старшему – правый.

Диапазон изменения значений выборок сигнала определяется дискретизацией. Для 8-битовых данных он составляет от 0 до 255 (0xFF), причем отсутствию сигнала (полной тишине) соответствует значение 128 (0x80). Для 16-битовых данных диапазон изменения составляет от –32768 (–0x8000) до 32767, (0x7FFF), отсутствию сигнала соответствует значение 0. [Теоретические сведения взяты из книги Фролов А. В., Фролов Г. В. "Мультимедиа для Windows". – Москва: "Диалог-МИФИ", 1995 г.]

Фильтр нижних частот скользящего среднего – разновидность цифрового фильтра с конечной импульсной характеристикой. На рис. 5.1 представлены АЧХ фильтра скользящего среднего различных порядков.

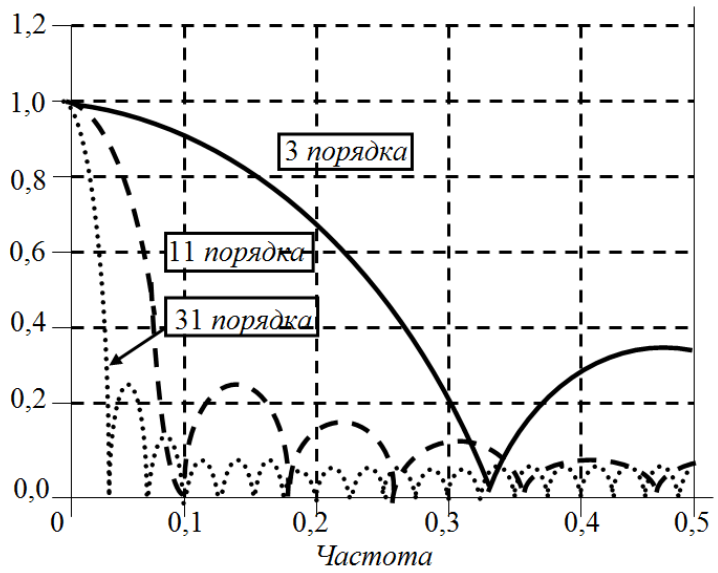


Рис. 5.1.

Создаем проект Win32 Console Application с поддержкой MFC. Текст программы, в которой реализован 8-точечный фильтр скользящего среднего, представлен ниже.

```
#include "stdafx.h"
#include "WaveSample.h"
#pragma comment(lib,"winmm.lib")

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CWinApp theApp;
using namespace std;
```

```
struct WAVESTRUCT
{
  char formatId[4];
  int blockLength;
  char soundId[4];
  char subId[4];
  int headerLength;
  short formatType;
  short channels;
  int frequency;
  int dataRate;
  short sampleBytes;
  short sampleWidth;
  char dataId[4];
  int dataLength;
};

int main(int argc, TCHAR* argv[], TCHAR* envp[])
{
  int nRetCode = 0;
  if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
  {
          cerr << _T("Fatal Error: MFC initialization failed") << endl;
          nRetCode = 1;
  }
  else
  {
  // Заполнение массива выборок числовыми данными
  char data[88400];
  for(int i=0;i<88400;i++) data[i]=128+rand()%127;
  //Реализация КИХ-фильтра скользящего среднего
  int tmp=0;
  int N=8;
  int Y[88400];
  memset(Y,0,88400*sizeof(int));
  for(i=N;i<88400;i++)
  {
          for(int k=0;k<N;k++)
          {
                  tmp=tmp+data[i-k];
          }
          Y[i]=tmp;
          tmp=0;
  }
  for(i=0;i<88400;i++) data[i]=Y[i]/N;

  WAVESTRUCT sws;
  strcpy(sws.formatId,"RIFF");
  sws.blockLength=sizeof(data)+36;
  strcpy(sws.soundId,"WAVE");
```

```
  strcpy(sws.subId,"fmt ");
  sws.headerLength=16;
  sws.formatType=1;
  sws.channels=1;
  sws.frequency=22100;
  sws.dataRate=22100;
  sws.sampleBytes=1;
  sws.sampleWidth=8;
  strcpy(sws.dataId,"data");
  sws.dataLength=88400;
  CFile nf;
  nf.Open("e:\\test.wav",CFile::modeCreate | CFile::modeWrite);
  nf.Write(&sws,sizeof(sws));
  nf.Write(data,sizeof(data));
  nf.Close();
  // проигрывание файла
  sndPlaySound((LPCSTR)&sws, SND_MEMORY | SND_ASYNC);
  cout<<"Press Enter key for play sound or any key to exit.\n";
  while(getch()==13);
  }
  return nRetCode;
}
```

Для демонстрации работы фильтра в программе реализован генератор, который формирует белый шум с равномерным спектром во всем диапазоне частот. После обработки такого сигнала фильтром, выходной сигнал записывается в WAV-файл. Спектральный анализ выполняется в программе SpectraLab. На рис. 5.2 показано окно с настойками программы.

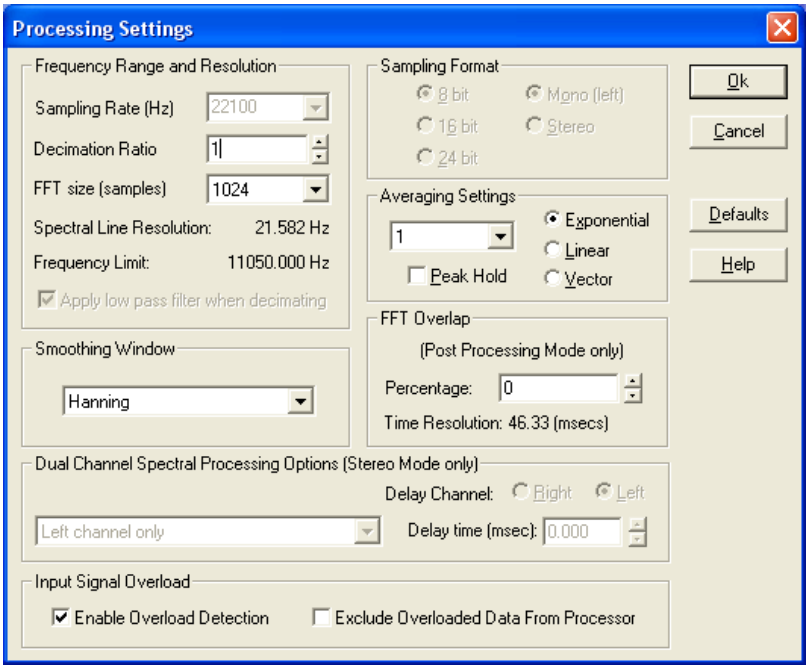


Рис. 5.2.

На рис. 5.3 представлено окно программы SpectraLab с результатами анализа WAV-файла, в который записан белый шум. Из рисунка видно, что спектральные составляющие распределены равномерно в анализируемом диапазоне частот.

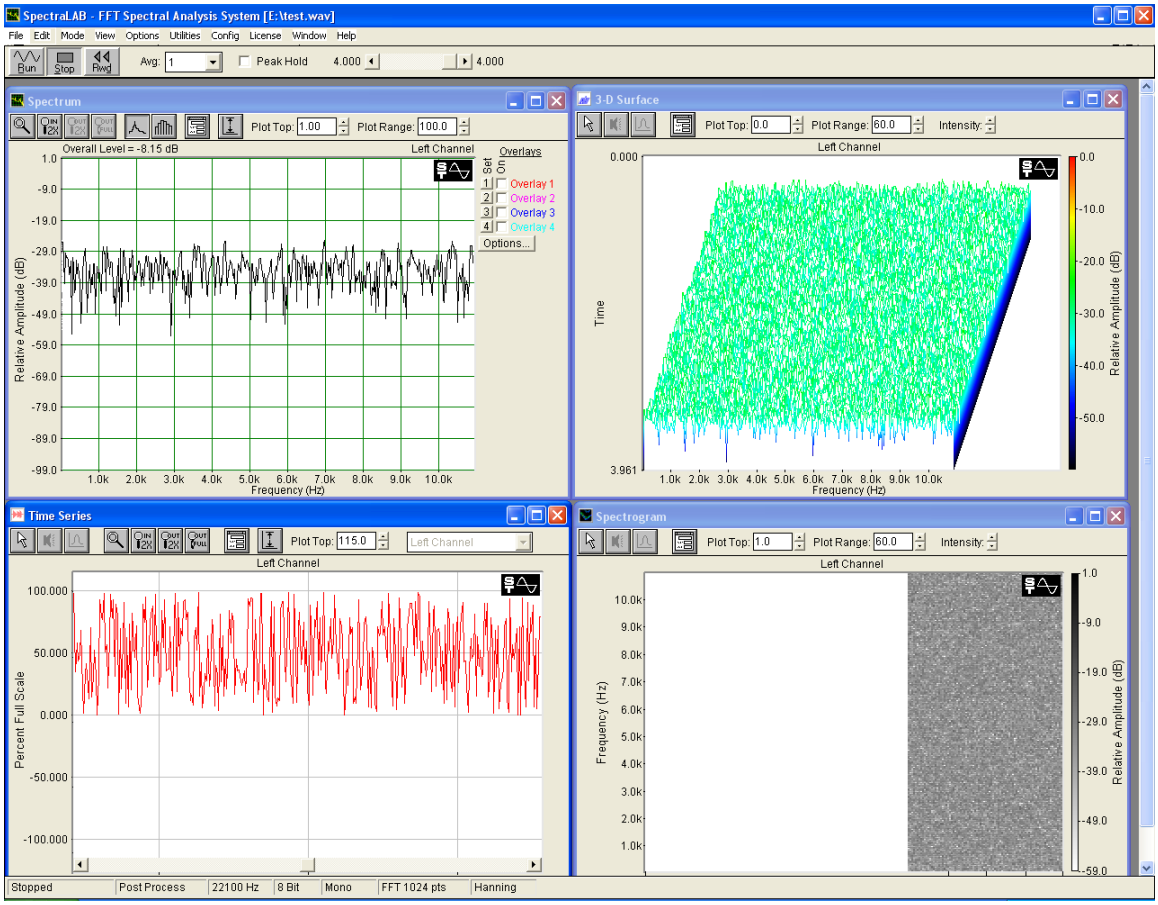


Рис. 5.3.

Программная реализация фильтра скользящего среднего имеет вид:

```
int tmp=0;
int N=8;
int Y[88400];
memset(Y,0,88400*sizeof(int));
for(i=N;i<88400;i++)
{
        for(int k=0;k<N;k++)
        {
                tmp=tmp+data[i-k];
        }
        Y[i]=tmp;
        tmp=0;
}
for(i=0;i<88400;i++) data[i]=Y[i]/N;
```

На рис. 5.4 представлено окно программы SpectraLab с результатами спектрального анализа сигнала, представляющего собой белый шум, пропущенный через 8-точечный фильтр скользящего среднего.

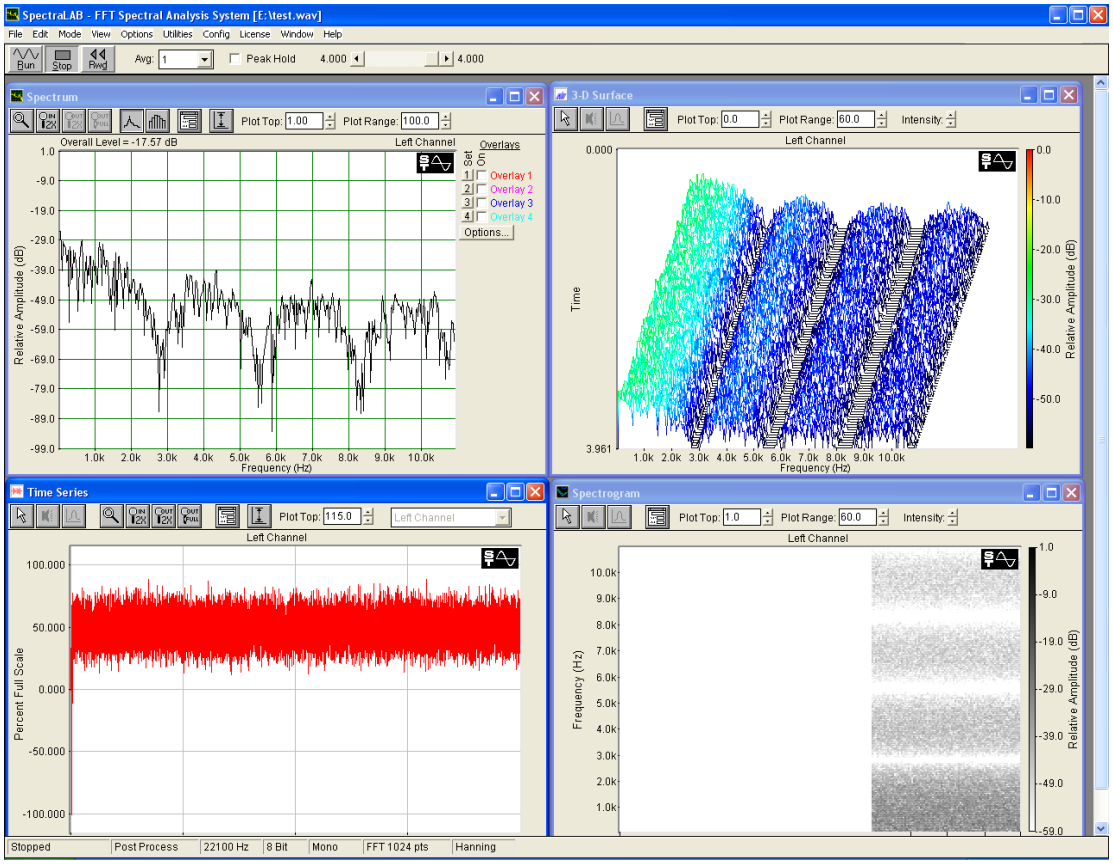


Рис. 5.4.

Реализация фильтра верхних частот. С помощью программы ciirf, вид интерфейса которой представлен на рис. 5.5, можно рассчитать коэффициенты фильтра с бесконечной импульсной характеристикой (рекурсивного), корректировать эти коэффициенты одновременно наблюдая за изменением амплитудно-частотной характеристики. Фильтр реализован в виде последовательного соединения секций второго порядка для ФНЧ и ФВЧ, и четвертого порядка для полосового и режекторного фильтров.

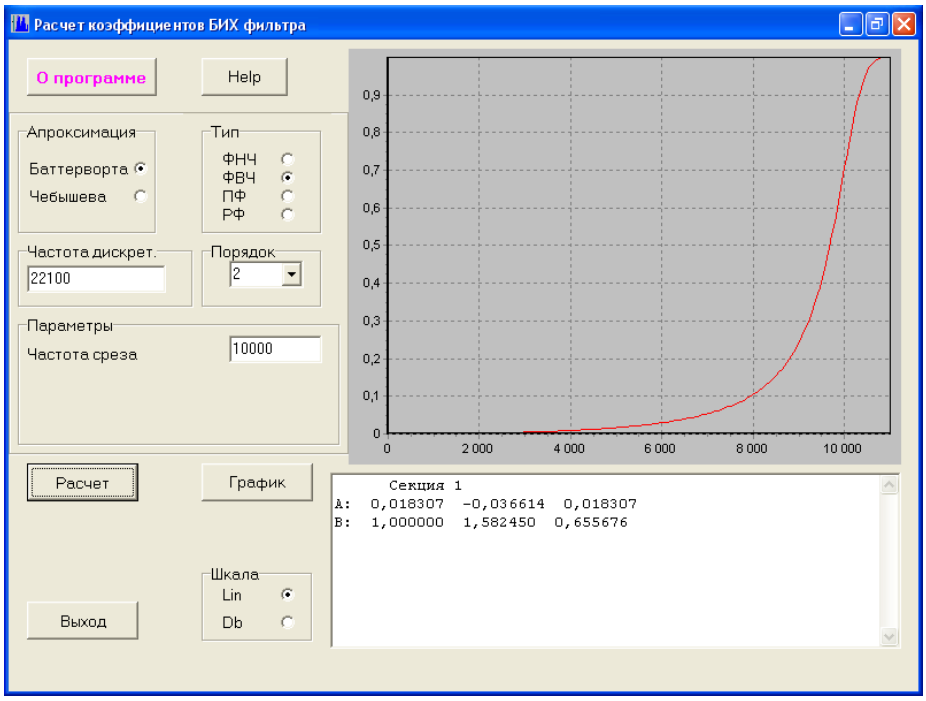


Рис. 5.5.

Программная реализация такого фильтра представлена ниже.

```
double a0=0.018307;
double a1=-0.036614;
double a2=0.018307;
double b1=-1.582450;
double b2=-0.655667;
unsigned int temp[88400];
memset(temp,0,88400*sizeof(int));
for(int n=2;n<88400;n++)
{
temp[n]=b1*temp[n-1]+b2*temp[n-2]+a0*data[n]+a1*data[n-1]+a2*data[n-2];
}
for(n=0;n<88400;n++) data[n]=temp[n];
```

Результат спектрального анализа сигнала, полученного с выхода фильтра, представлен на рис. 5.6.

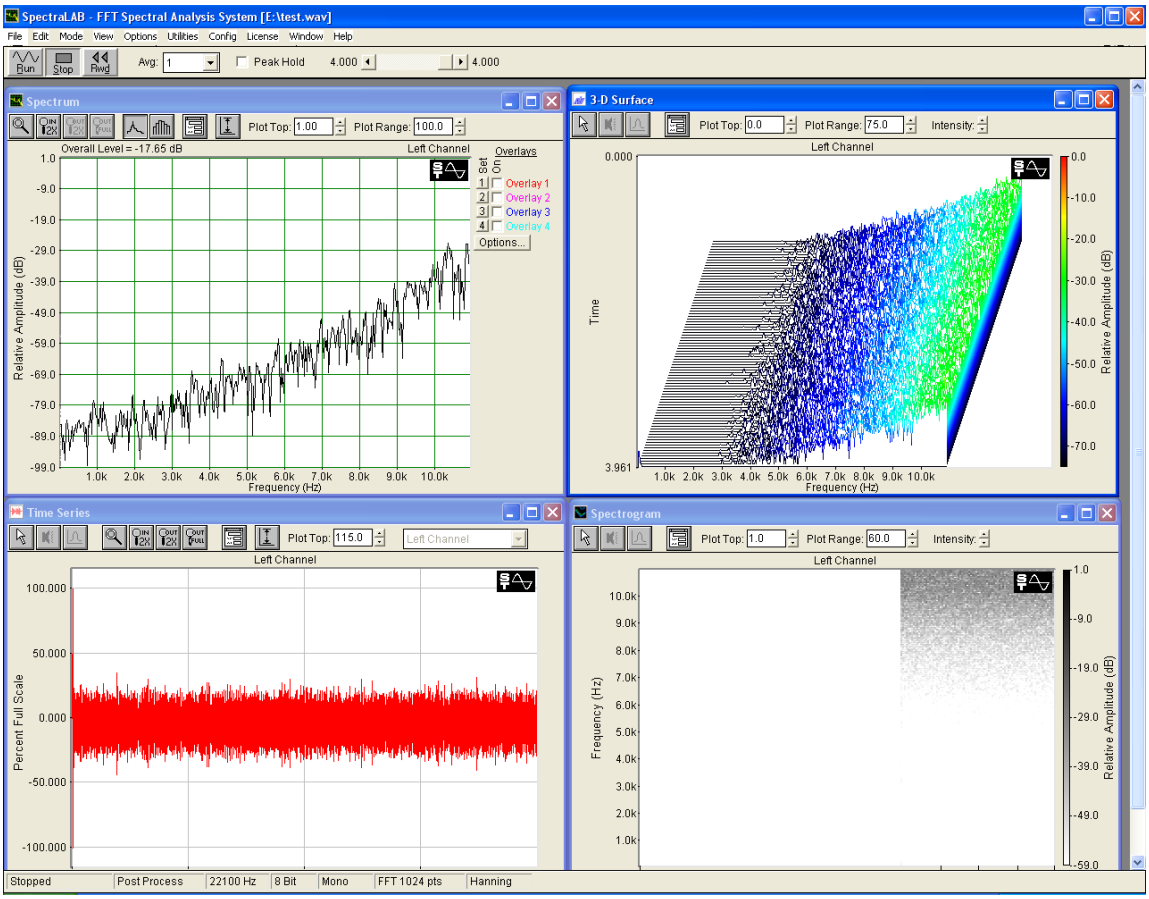


Рис. 5.6.

На рис. 5.7 представлена АЧХ полосового фильтра 4 порядка и значения коэффициентов секций.

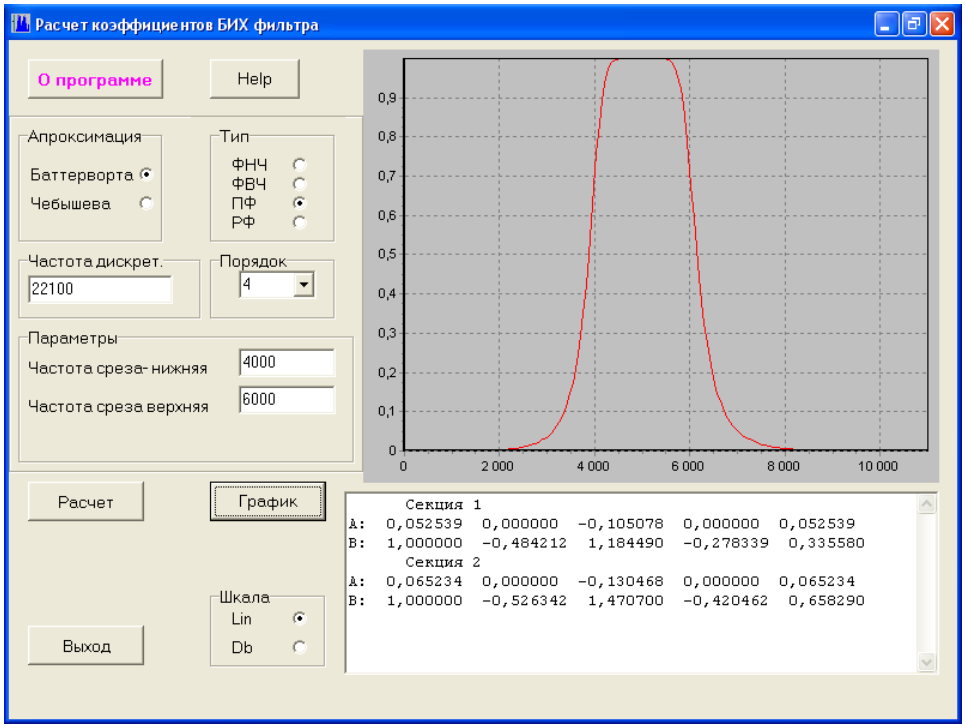


Рис. 5.7.

Программная реализация полосового фильтра 4 порядка представлена далее.

```
unsigned int Y[88400];
memset(Y,0,88400*sizeof(int));
double a0=0.052539;
double a1=0;
double a2=-0.105078;
double a3=0;
double a4=0.052539;
double b1=0.484212;
double b2=-1.184490;
double b3=0.278339;
double b4=-0.335580;
for(int n=4;n<88400;n++)
{
        Y[n]=b1*Y[n-1]+b2*Y[n-2]+b3*Y[n-3]+b4*Y[n-4]+a0*data[n]+
a1*data[n-1]+a2*data[n-2]+a3*data[n-3]+a4*data[n-4];
}
for(n=0;n<88400;n++) data[n]=Y[n]/2;

a0=0.065234;
a1=0;
a2=-0.130468;
a3=0;
a4=0.065234;
b1=0.526342;
b2=-1.470700;
b3=0.420462;
b4=-0.658290;
memset(Y,0,88400*sizeof(int));
for(n=4;n<88400;n++)
{
Y[n]=b1*Y[n-1]+b2*Y[n-2]+b3*Y[n-3]+b4*Y[n-4]+a0*data[n]+
a1*data[n-1]+a2*data[n-2]+a3*data[n-3]+a4*data[n-4];
}
for(n=0;n<88400;n++)
{
        data[n]=Y[n]/2;
}
```

После записи результирующего сигнала в файл и выполнения спектрального анализа были получены результаты представленные на рис. 5.8.

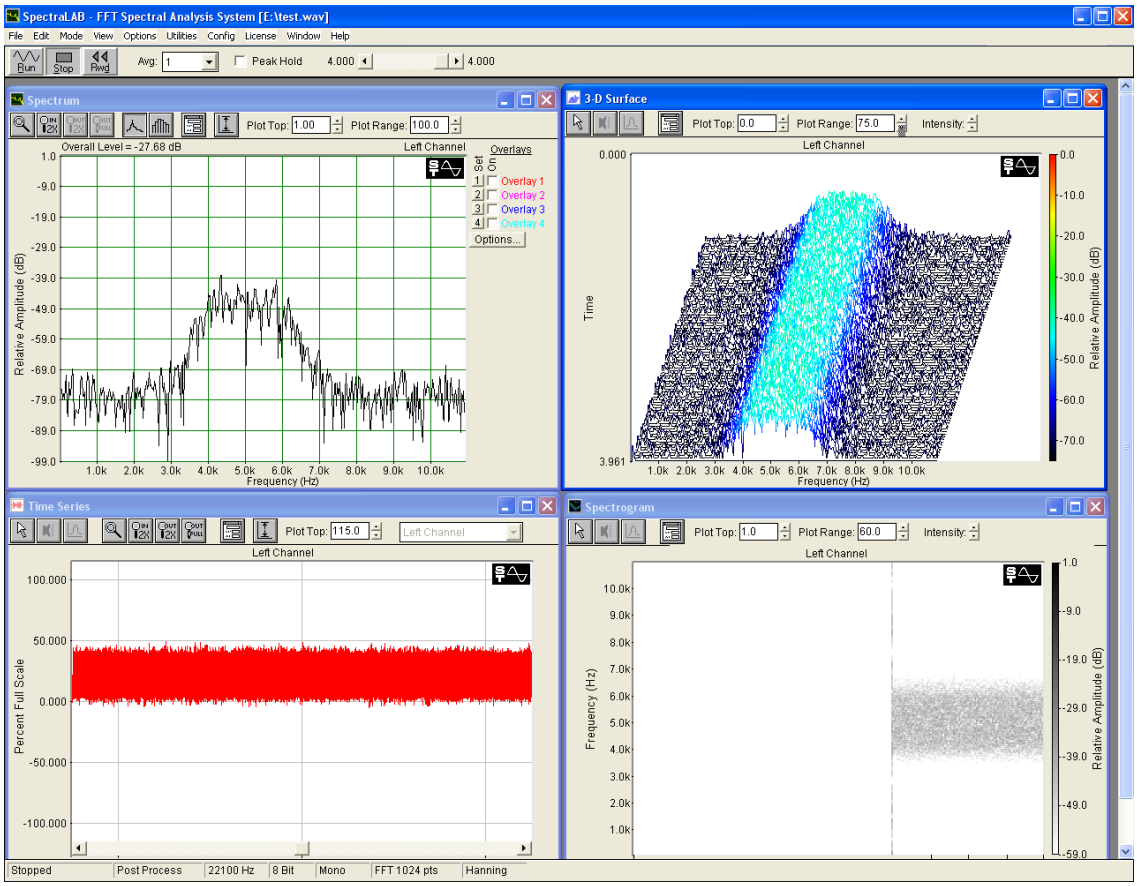


Рис. 5.8.

На рис. 5.9 представлена АЧХ режекторного фильтра 4 порядка и значения коэффициентов секций.

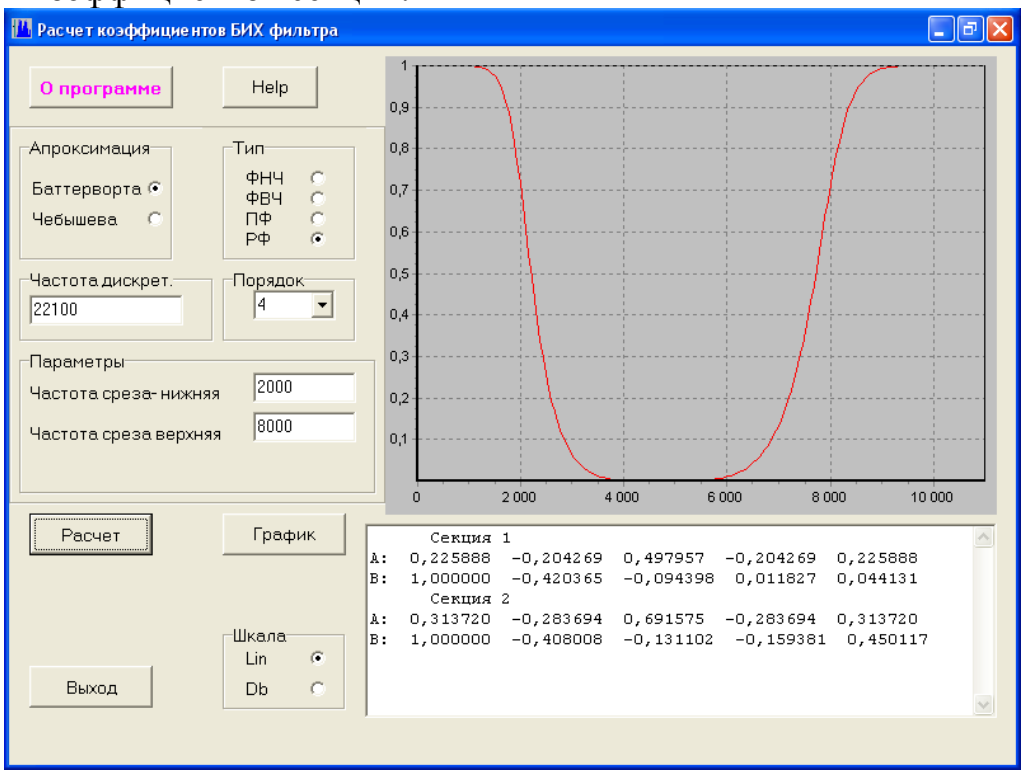


Рис. 5.9.

Программная реализация режекторного фильтра 4 порядка представлена ниже.

```
  int Y[88400];
  memset(Y,0,88400*sizeof(int));

  double a0=0.225888;
  double a1=-0.204269;
  double a2=0.497957;
  double a3=-0.204269;
  double a4=0.225888;

  double b1=0.420365;
  double b2=0.094398;
  double b3=-0.011827;
  double b4=-0.044131;

  for(int n=4;n<88400;n++)
  {
  Y[n]=b1*Y[n-1]+b2*Y[n-2]+b3*Y[n-3]+b4*Y[n-4]+a0*data[n]+
a1*data[n-1]+a2*data[n-2]+a3*data[n-3]+a4*data[n-4];
  }
  for(n=0;n<88400;n++) data[n]=(unsigned char)Y[n]/2;

  a0=0.313720;
  a1=-0.283594;
  a2=0.691575;
  a3=-0.283694;
  a4=0.313720;

  b1=0.408008;
  b2=0.131102;
  b3=0.159381;
  b4=-0.450117;

  memset(Y,0,88400*sizeof(int));
  for(n=4;n<88400;n++)
  {
        Y[n]=b1*Y[n-1]+b2*Y[n-2]+b3*Y[n-3]+b4*Y[n-4]+a0*data[n]+
a1*data[n-1]+a2*data[n-2]+a3*data[n-3]+a4*data[n-4];
  }
  for(n=0;n<88400;n++) data[n]=(unsigned char)Y[n];
```

В результате спектрального анализа сигнала с выхода фильтра получены результаты, представленные на рис. 5.10.

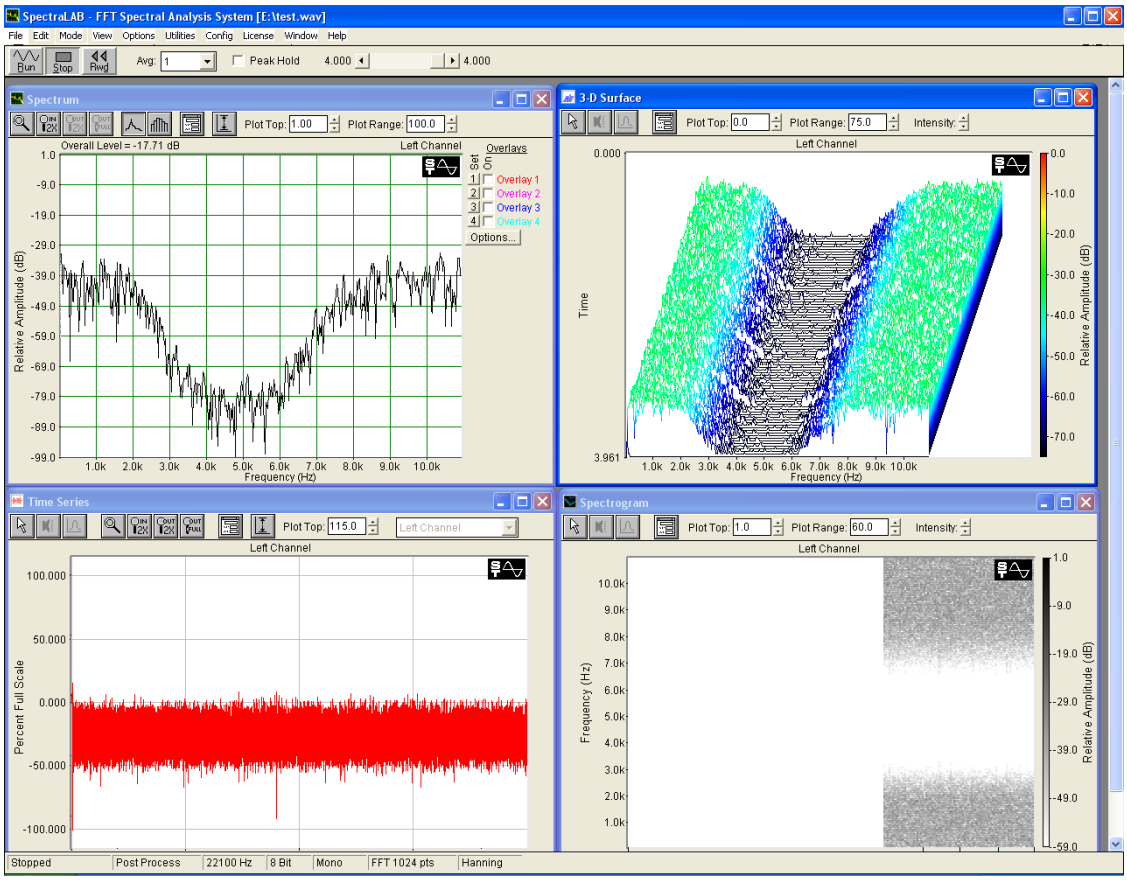


Рис. 5.10.

На рис. 5.11 представлена АЧХ КИХ-фильтра нижних частот.

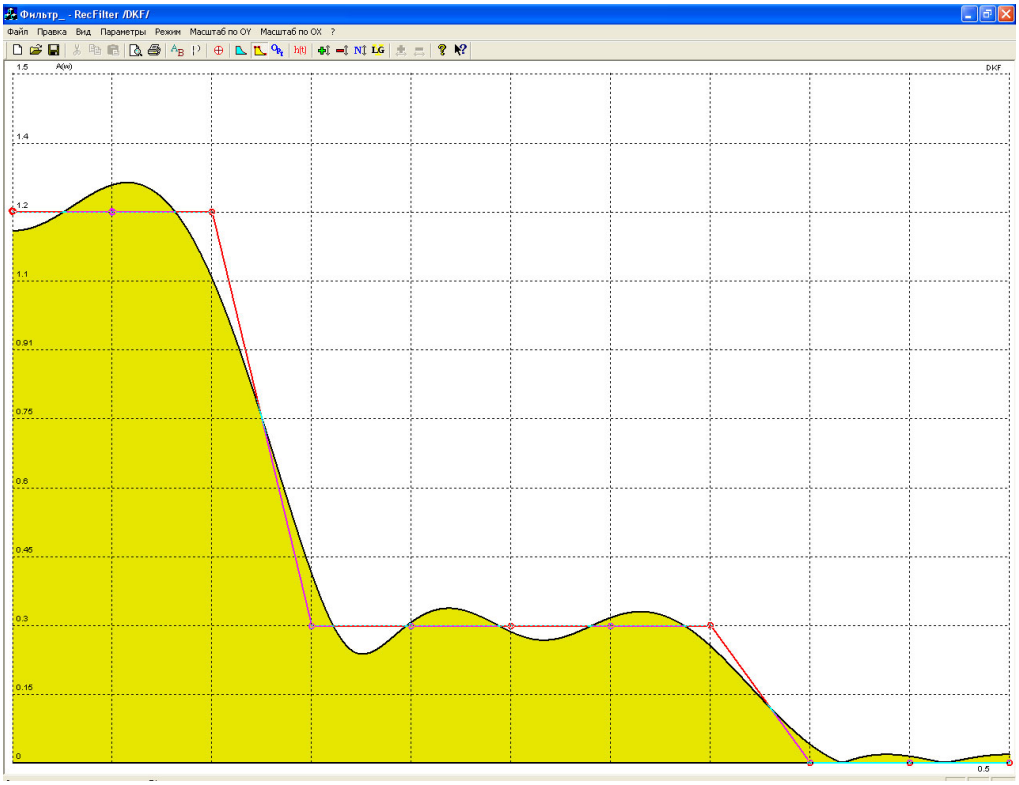


Рис. 5.11.

Структура фильтра с указной формой АЧХ и значения коэффициентов импульсной характеристики представлены на рис. 5.12.

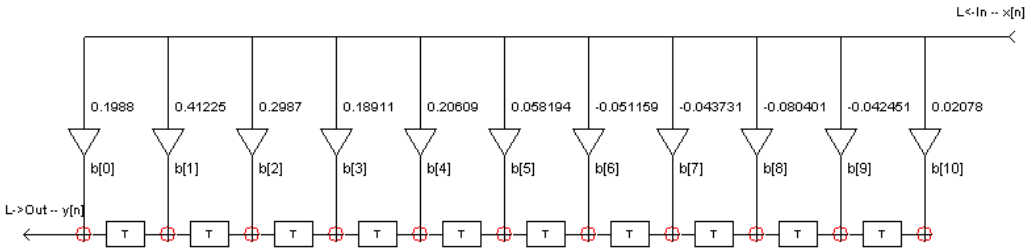


Рис. 5.12.

Текст программы, в которой выполняется реализация фильтра, представлен ниже.

```
#include "stdafx.h"
#include "WaveSample.h"
#pragma comment(lib,"winmm.lib")

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CWinApp theApp;

using namespace std;

struct WAVESTRUCT
{
  char formatId[4];
  int blockLength;
  char soundId[4];
  char subId[4];
  int headerLength;
  short formatType;
  short channels;
  int frequency;
  int dataRate;
  short sampleBytes;
  short sampleWidth;
  char dataId[4];
  int dataLength;
};

int main(int argc, TCHAR* argv[], TCHAR* envp[])
{
```

```
  int nRetCode = 0;
if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
  {
          cerr << _T("Fatal Error: MFC initialization failed") << endl;
          nRetCode = 1;
  }
  else
  {
  // Заполнение массива выборок числовыми данными
  char data[88400];

  for(int i=0;i<88400;i++) data[i]=142+rand()%106;
  //Реализация КИХ-фильтра нижних частот
  int tmp=0;
  int N=11;
  int Y[88400];
  memset(Y,0,88400*sizeof(int));
  double h[11]={0.1988, 0.41225, 0.2987, 0.18911, 0.20609, 0.058194, -0.051159,
-0.043731, -0.080401, -0.042451, 0.02078};

  for(i=N;i<88400;i++)
  {
          for(int k=0;k<N;k++)
          {
                  tmp=tmp+(int)(data[i-k]*h[k]);
          }
          Y[i]=tmp;
          tmp=0;
  }

  for(i=0;i<88400;i++) data[i]=Y[i];

  WAVESTRUCT sws;
  strcpy(sws.formatId,"RIFF");
  sws.blockLength=sizeof(data)+36;
  strcpy(sws.soundId,"WAVE");
  strcpy(sws.subId,"fmt ");
  sws.headerLength=16;
  sws.formatType=1;
  sws.channels=1;
  sws.frequency=22100;
  sws.dataRate=22100;
  sws.sampleBytes=1;
  sws.sampleWidth=8;
  strcpy(sws.dataId,"data");
  sws.dataLength=88400;

  CFile nf;
  nf.Open("e:\\test.wav",CFile::modeCreate | CFile::modeWrite);
  nf.Write(&sws,sizeof(sws));
```

```
    nf.Write(data,sizeof(data));
    nf.Close();
    // проигрывание файла
    sndPlaySound((LPCSTR)&sws, SND_MEMORY | SND_ASYNC);
    cout<<"Press Enter key for play sound or any key to exit.\n";
    while(getch()==13);
    }
    return nRetCode;
}
```

Результат спектрального анализа сигнала, полученного с выхода фильтра, представлен на рис. 5.13.

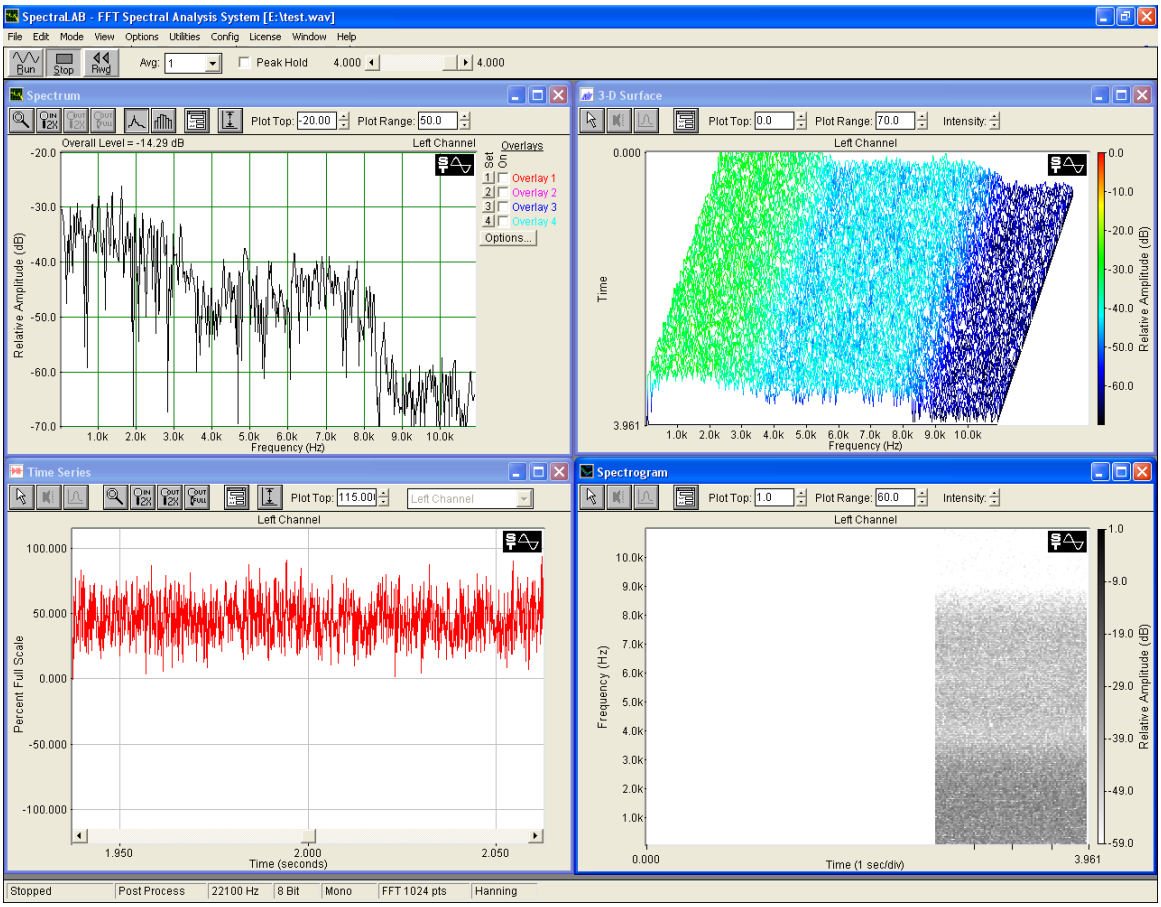


Рис. 5.13.

5.2. Динамическая и спектральная обработка звуковых сигналов

Динамическая обработка предназначена для сокращения динамического диапазона звуковых сигналов. Звуковые сигналы в радиовещании, телевидении и звукозаписи всегда подвергаются такой обработке, независимо аналоговые они или цифровые. Это связано с тем, что часто динамический диапазон природных звуков, звуков музыки и речи значительно шире динамического диапазона электроакустических трактов современной аппаратуры. Динамическая обработка звуковых сигналов производиться с

помощью лимитеров, максимайзеров, компрессоров, экспандеров и гейтов. Это все пороговые устройства, в которых при достижении сигнала установленного уровня их коэффициент передачи меняется скачком. Таким способом можно как сжать, так и расширить динамический диапазон сигнала. В системах передачи звуковых сигналов по линиям связи сокращение и обратное расширение динамического диапазона производится с помощью компандерной системы компрессирования. На входе линии устанавливается компрессор, а на выходе – экспандер, поэтому в такой системе сигнал компрессирован только в линии связи. Названия приборов такие же, как и при динамической обработке, но принцип их работы совершенно иной, на это нужно обратить внимание.

Спектральные процессоры звуковых сигналов предназначены для изменения тембра звука музыкальных инструментов и вокальных исполнителей путем преднамеренного изменения его спектрального состава. При спектральном преобразовании искажения могут создаваться в виде гармоник и субгармоник, которые отсутствовали в исходном звуковом материале. Искажения могут вноситься в звуковых полосах и для отдельных музыкальных инструментов или солистов. Они могут создаваться и во всем звуковом диапазоне. При этом может меняться частотный и амплитудный состав обертонов музыкальных звуков. В отличие от частотной коррекции тембра с помощью эквалайзеров при спектральной обработке звуковых сигналов создаются обертона (гармоники), которых не было. Таким образом, спектральное преобразование – это принципиально нелинейный процесс. Такие преобразования осуществляются для создания художественных эффектов звучания, а также при реставрации старых записей, в которых безвозвратно утеряны высшие гармоники. Нередко приходиться использовать спектральную коррекцию для восстановления динамики музыки в области высоких частот, нарушенной при компрессировании. К спектральным процессорам звука относятся эксайтеры, энхансеры, виталайзеры, спектралайзеры и многие другие. Их названия большей частью являются брендом фирмы-изготовителя.

Следующая программа демонстрирует реализации фильтра, который выполняет децимацию сигнала.

```
#include "stdafx.h"
#include "WaveSample.h"
#pragma comment(lib,"winmm.lib")

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
CWinApp theApp;
```

```
using namespace std;
struct WAVESTRUCT
{
  char formatId[4];
  int blockLength;
  char soundId[4];
  char subId[4];
  int headerLength;
  short formatType;
  short channels;
  int frequency;
  int dataRate;
  short sampleBytes;
  short sampleWidth;
  char dataId[4];
  int dataLength;
};
int main(int argc, TCHAR* argv[], TCHAR* envp[])
{
  int nRetCode = 0;
   if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
  {
          cerr << _T("Fatal Error: MFC initialization failed") << endl;
          nRetCode = 1;
  }
  else
  {
  byte * data;
  WAVESTRUCT sws;
  CFile nf;
  CStdioFile tf;
  char path[255]="e:\\s1.wav";
  if(!nf.Open(path, CFile::modeRead | CFile::typeBinary))
  {
          cout<<"File error!";
          return 0;
  };
  nf.Read(&sws,sizeof(sws));
  int size=sws.dataLength;
  data=new byte[size];
  nf.Read(data, size);
  nf.Close();

  CFile ff("e:\\nw.wav", CFile::modeWrite | CFile::typeBinary | CFile::modeCreate);
  byte * nd=new byte[size];
  memset(nd,0,size);

  sws.frequency/=2;
  sws.dataRate/=2;
  sws.dataLength=size/2;
```

```
  // реализация децимации
  for(int i=0;i<size/2;i++)
  {
          nd[i]=data[2*i];
  }

ff.Write(&sws,sizeof(sws));
  ff.Write(nd,sws.dataLength);
  ff.Close();
  delete [] data;
  delete [] nd;
  }
  PlaySound("e:\\nw.wav",NULL,SND_FILENAME | SND_ASYNC);
  return nRetCode;
}
```

На рис. 5.14 представлен результат спектрального анализа исходного сигнала, а на рис. 5.15 – сигнала после децимации.

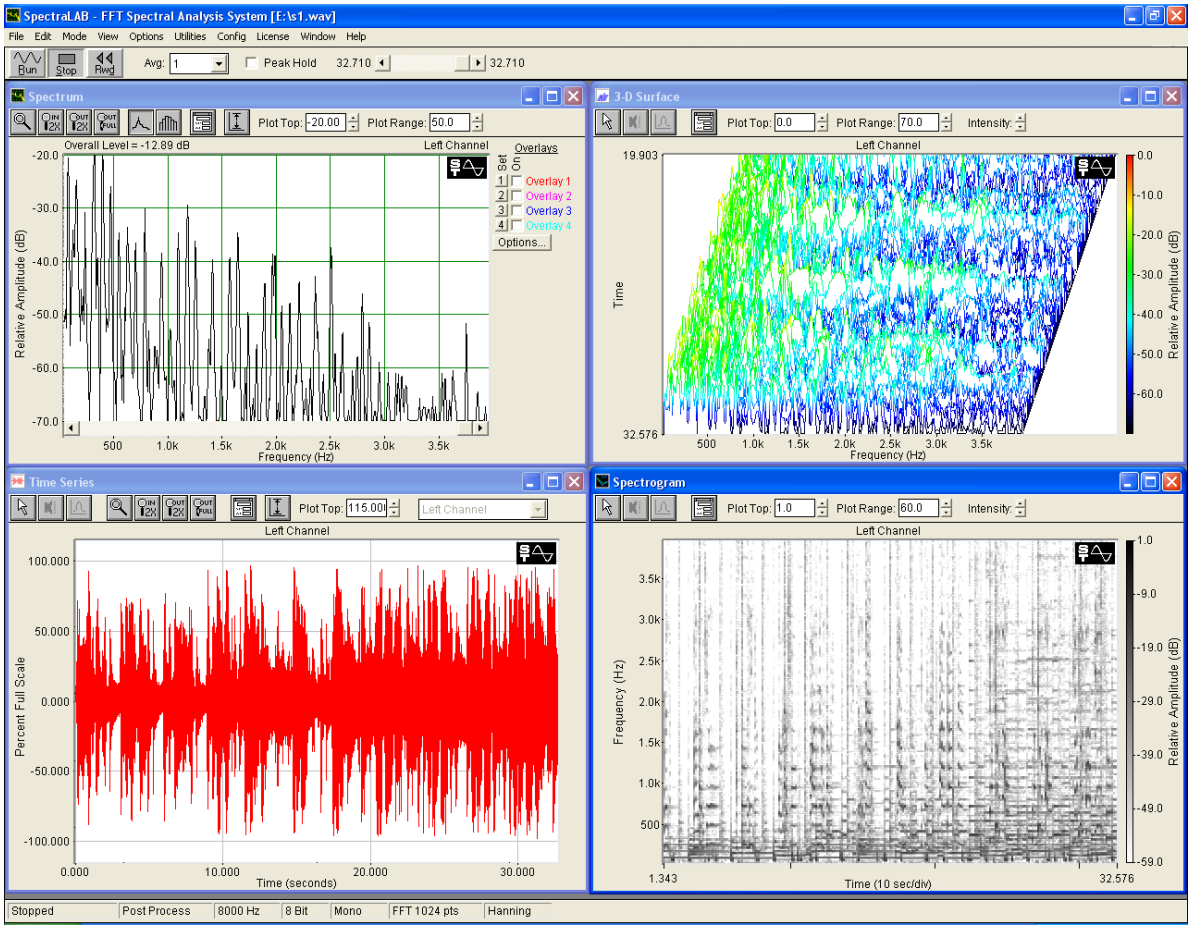


Рис. 5.14.

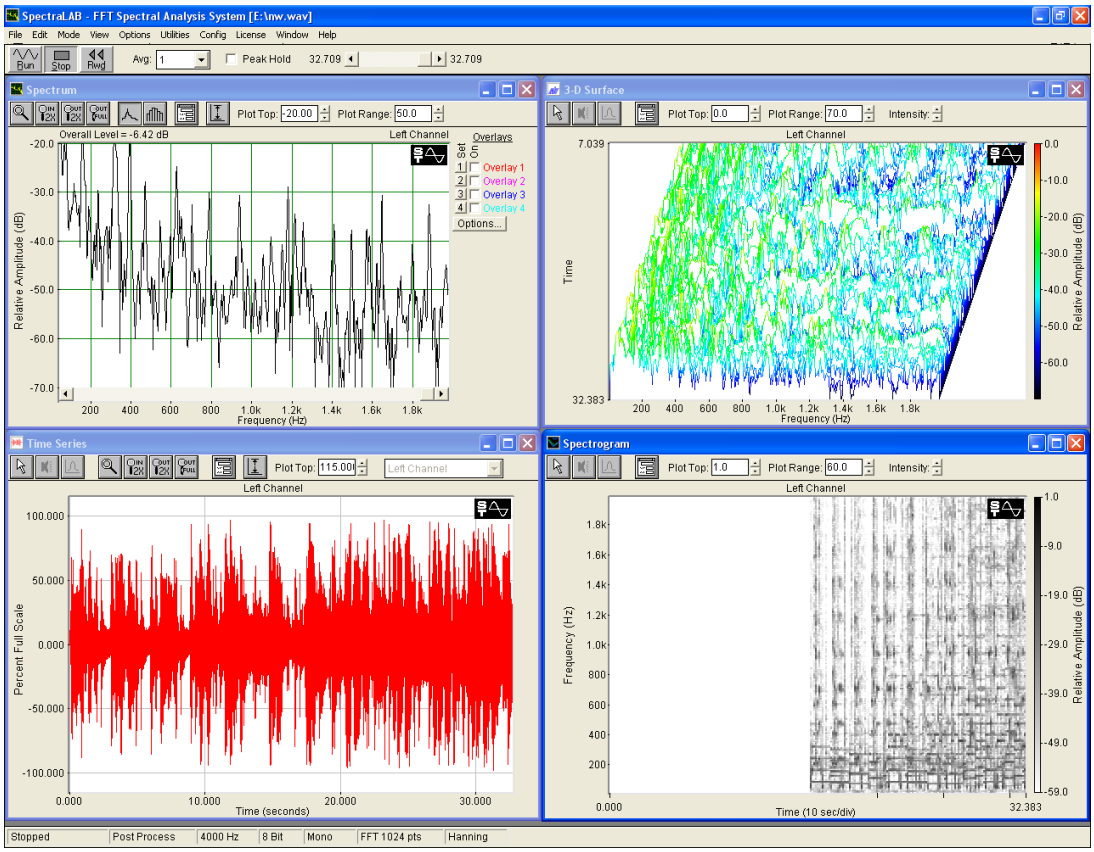


Рис. 5.15.

При прослушивании звукового файла после децимации можно услышать неприятные щелчки и треск. Уменьшить этот эффект можно применением фильтрации. Ниже представлен фрагмент программы для реализации децимации с цифровой фильтрацией. Используется КИХ-фильтр скользящего среднего 4 порядка.

```
byte * tmp=new byte[size/2];
for(int i=0;i<size/2;i++)
{
        tmp[i]=data[2*i];
}
for(i=3;i<size/2;i++)
{
        nd[i]=((int)(tmp[i]+tmp[i-1]+tmp[i-2]+tmp[i-3]))/4;
}
delete [] tmp;
```

Следующий фрагмент кода показывает, как можно изменить частоту дискретизации и получить интересный звуковой эффект.

```
sws.frequency=12000;
sws.dataRate=12000;
for(int i=0;i<size;i++)
```

```
{
        nd[i]=data[i];
}
```

На рис. 5.16 представлено окно программы SpectraLab после анализа сигнала с измененной частотой дискретизации.

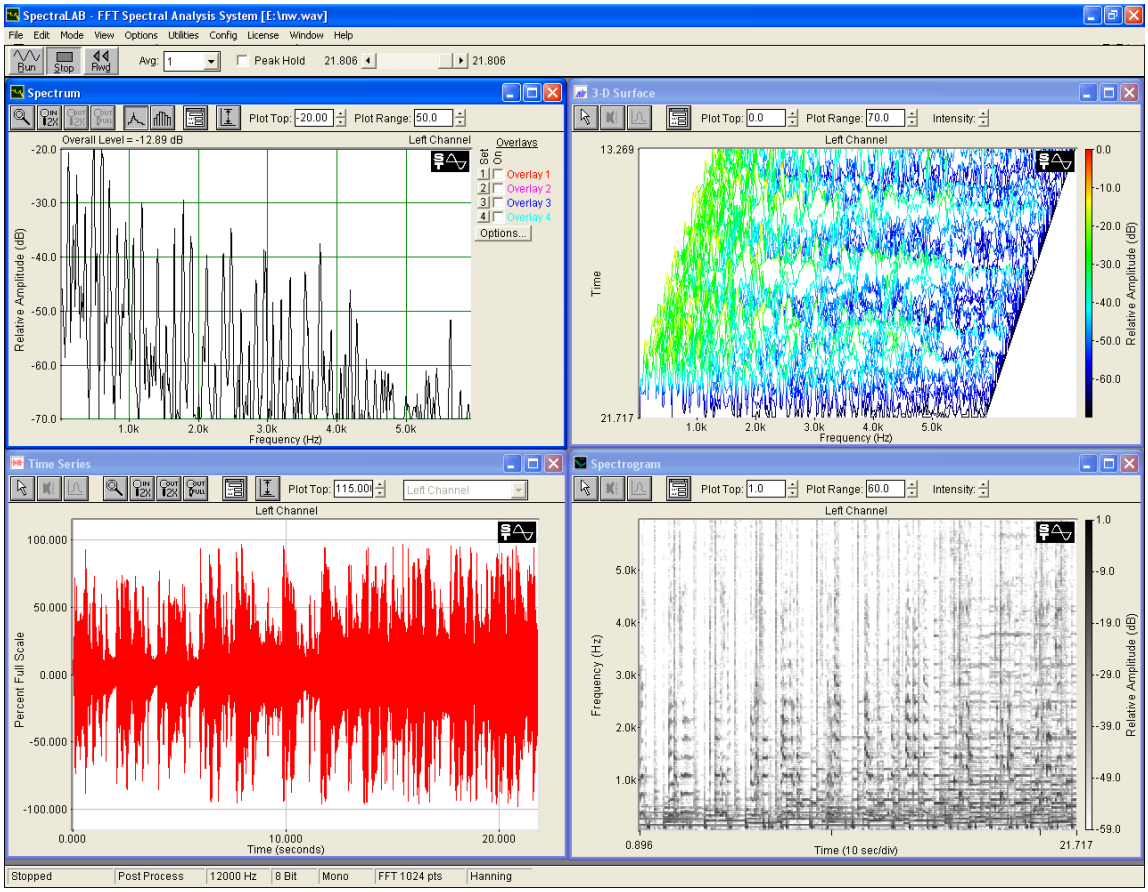


Рис. 5.16.

5.3. Программная реализация различных звуковых эффектов

При подготовке аудиоматериала к воспроизведению в него могут быть внесены некоторые звуковые эффекты, добавляющие в него дополнительные краски. Эффекты могут использоваться не только при обработке музыкальных произведений, но и при воспроизведении речевого материала для снижения монотонности при прослушивании длительных фрагментов. Однако в этом случае следует быть особенно осторожным, поскольку подобные обработки, как правило, приводят к снижению разборчивости речевого сигнала. Поэтому необходимо постоянно следить за тем, чтобы достигаемый положительный эффект не был бы сведен на нет снижением разборчивости речевого материала, приводящей, в свою очередь, к повышению нагрузки слушателя.

Звуковой эффект «эхо» и «реверберация». Один из самых распространенных классов являются звуковые эффекты, основанные на

задержке сигнала. В реальной ситуации задержка звукового сигнала может возникнуть при отражении звуковой волны от твердой преграды, например от стены (рис. 5.17).

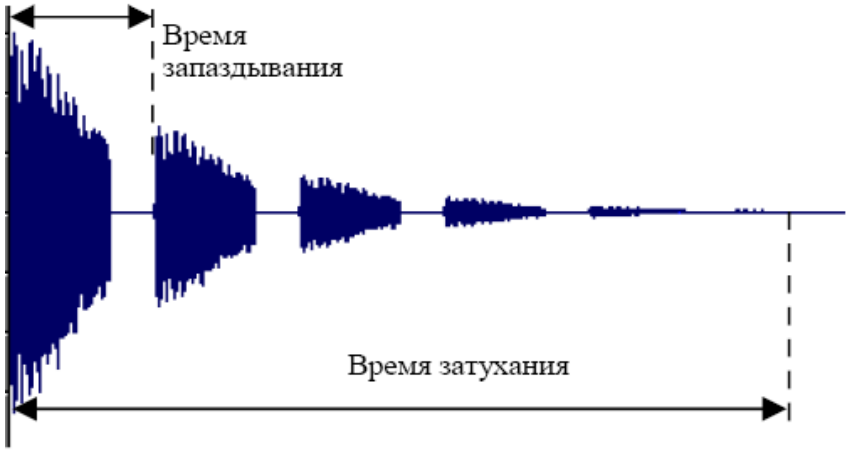


Рис. 5.17.

Величину возникающей при этом задержки легко определить, зная пройденный звуковой волной путь и учитывая, что скорость звука можно считать равной 330 м/с. Таким образом, за одну миллисекунду звуковая волна проходит 33 сантиметра. Для расчетов удобнее помнить, что один метр она проходит за 3 мс. Зная частоту дискретизации аналогового сигнала, легко вычислить количество выборок, которые будут получены за указанный промежуток времени.

Эхо возникает в тоннелях, на стадионах, в горных ущельях пещерах и других местах из-за отражения звуковых волн от любых препятствий, мешающих их распространению, но только в случае запаздывания отражений на время больше 50 мс. Это время равно постоянной времени слуха человека. Пока запаздывания отраженных сигналов меньше этого времени все они воспринимаются как единый звук, эхо не возникает.

Звуковой эффект «эхо» получается, если один или несколько задержанных сигналов, добавляются к оригинальному. Чтобы эффект воспринимался как эхо, задержка должна быть порядка 50 мс или более. Эффект может быть достигнут с помощью как аналоговой, так и цифровой обработки. Если значительное число задержанных сигналов воспроизводится в течение нескольких секунд, результирующий сигнал создает эффект присутствия в большом помещении и воспринимается как эффект реверберации.

Рассмотрим реализацию однократного отражения звуковой волны. Этот эффект является базовым для реализации более сложных эффектов, таких как имитация реверберации и хора.

Блок-схема реализации однократной задержки приведена на рис. 5.18.

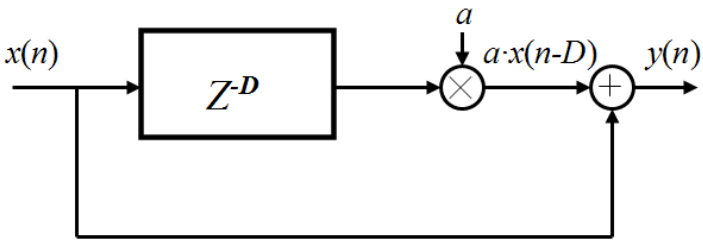


Рис. 5.18.

Входной сигнал $x(n)$ одновременно поступает на линию задержки и на сумматор, формирующий выходной сигнал. Задержанный сигнал $x(n)$ умножается на масштабный коэффициент a и также поступает на сумматор. Преобразования, осуществляемые данной схемой, могут быть выражены следующей формулой:

$$y(n) = x(n) + ax(n - D).$$

Данная схема является линейным фильтром с передаточной функцией:

$$H(z) = 1 + az^{-D}.$$

Такая передаточная характеристика является характеристикой гребенчатого фильтра. Отличительной особенностью данного класса фильтров является выполнение для них равенства:

$$H(j\omega) = H(j\omega + \frac{2\pi}{D}).$$

Поскольку при любом отражении звуковой сигнал ослабляется, то $-1 < a < 1$. Отрицательные значения соответствуют отражению в противофазе. Эта модель является достаточно грубой, поскольку как распространение звуковой волны, так и ее отражение являются частотно-зависимыми процессами. Однако в большинстве случаев эта модель является адекватной. Величина D определяется разностью времени между приходом прямой и отраженной звуковой волны. Программная реализация фильтра представлена ниже.

```
int D=200;
for(int i=0;i<size;i++)
{
        if(i<D) nd[i]=data[i];
        else
        {
                nd[i]=0.5*data[i]+0.5*data[i-D]/2;
        }
}
```

Частота дискретизации в обрабатываемом файле равна 8 кГц. Величина D=200 соответствует задержке 25 мс. При задержке более 50 мс (D>400) слушатель услышит эхо. Рассмотренный пример моделирует однократное

отражение звуковой волны, расположенной на расстоянии 4,3 метра от слушателя (звуковая волна преодолевает расстояние до стены за 12,5 мс, и возвращается обратно за такое же время). Для того, чтобы при выполнении этой операции не возникало переполнения разрядной сетки, оба сигнала складываются с коэффициентом 0,5. При уменьшении задержки ниже 10 мс в сигнале могут возникнуть нежелательные биения, поскольку сумма двух гармонических сигналов с одинаковой амплитудой может быть представлена как амплитудная модуляция сигнала с частотой, равной разности частот исходных сигналов.

Введение задержки в один из каналов позволяет создать эффект псевдостереофонического прослушивания. В этом случае один из стереоканалов подается исходный монофонический сигнал, а в другой – с задержкой его копия. Временная задержка между каналами не должна превышать 15–20 мс. В противном случае стереосигнал субъективно будет разбиваться на два задержанных относительно друг друга монофонических сигнала. Иногда для создания псевдостереофонического эффекта в оба канала подаются суммы исходного и задержанного сигналов, различающиеся величиной задержки и весовыми коэффициентами, используемыми при создании выходного сигнала в соответствующем канале.

Однократная задержка часто используется для оживления «сухой» музыки. В этом случае как основной, так и задержанный аудиосигналы складываются с одинаковой амплитудой.

Область использования однократных задержек сигнала очень ограничена, поскольку в реальной жизни редко встречается ситуация, когда в акустическом поле присутствует только одна отражающая поверхность. Кроме того, даже в этом случае для повышения адекватности модели необходимо использовать формирующий фильтр, учитывающий частотную характеристику отражающей поверхности, и, возможно, частотную характеристику среды распространения сигнала. Для моделирования акустических полей с несколькими отражающими поверхностями и учета частотных характеристик процессов отражения и распространения звуковой волны используются многократные задержки.

Многократная задержка может быть достигнута двумя путями: доступом к различным элементам, хранящимся в одной линии задержки, или созданием нескольких отдельных линий, имеющих различное время задержки.

Многократная задержка сигнала может быть реализована как с использованием трансверсальных (КИХ) фильтров, так и с использованием рекурсивных фильтров.

Блок-схема реализации многократной задержки сигнала с использованием КИХ-фильтра приведена на рис. 5.19.

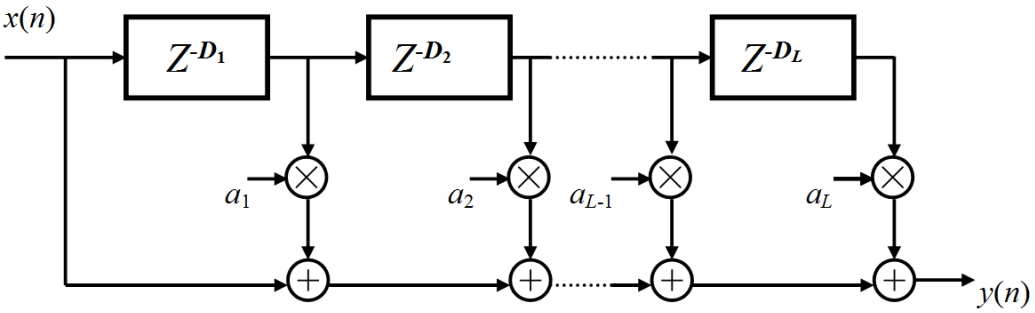


Рис. 5.19.

Разностное уравнение такого фильтра может быть записано в виде:

$$y(n) = x(n) + \sum_{i=1}^{L} a_i x(n - \sum_{k=1}^{i} D_k) .$$

В данном фильтре используются различные величины задержек входного сигнала, что не позволяет считать этот фильтр гребенчатым. Использование конечного числа линий задержки дает возможность учесть влияние ограниченного числа отражающих поверхностей.

Для реализации многократной задержки и моделирования переотраженной звуковой волны может быть использован рекурсивный фильтр, структура которого представлена на рис. 5.20.

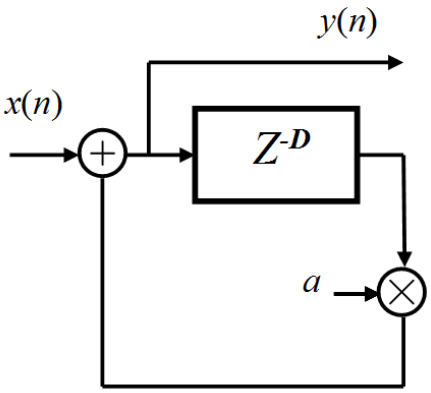


Рис. 5.20.

Этот фильтр может быть использован для описания переотраженной звуковой волны между двумя бесконечными параллельными стенами (рис. 5.21), на одной из которых расположен излучатель звуковой волны.

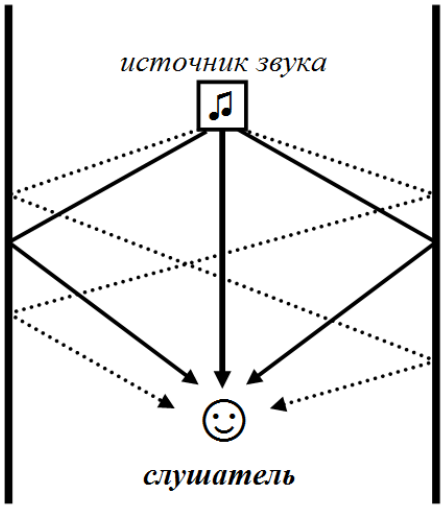


Рис. 5.21.

Приемник может располагаться на любой из стен. При отражении от стены энергия звуковой волны падает в a раз. Импульсная характеристика данного фильтра представляет собой последовательность одиночных импульсов, расположенных на расстоянии D отсчетов друг от друга, причем импульс с номером n имеет амплитуду a_n. Программная реализация такого фильтра представлена ниже.

```
int D=800;
for(int i=0;i<size;i++)
{
        if(i<D) nd[i]=data[i];
        else
        {
                nd[i]=0.3*data[i]+0.7*nd[i-D];
        }
}
```

Данный фильтр описывается разностным уравнением

$$y(n) = x(n) + ay(n - D),$$

и имеет передаточную функцию

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

Такой фильтр создает эффект реверберации. Реверберация – это процесс постепенного уменьшения интенсивности звука при его многократных отражениях.

Главным назначением реверберaторов является имитация объемного звучания путем моделирования акустических условий распространения звука в различных условиях – концертный зал, жилая комната, заглушенная камера, лес или чистое поле. Эхосигнал представляет отражённую от препятствия звуковую волну. Явление реверберации состоит в суперпозиции различных

эхосигналов от одного источника звука. Эффект реверберации можно наблюдать в закрытых помещениях после выключения источника звука. Художественно-эстетическое впечатление, создаваемое реверберацией, зависит от контекста звукового. Обычно избыточная длительность реверберации приводит к неприятной гулкости, «пустоте» помещения, а недостаточная – к резкому отрывистому звучанию, лишённому музыкальной «сочности». Искусственно создаваемая реверберация в определённых пределах способствует улучшению качества звучания, создавая ощущение приятного «резонанса» помещения.

При записи речи, пения, музыки, а также создания различных шумовых эффектов использование искусственной реверберации является составной частью общей обработки аудиосигнала. Такой вид обработки определяется как техническими условиями проведения записи, так и художественно-эстетическими задачами. Реверберацию используют для улучшения и подчёркивания художественной выразительности речи, пения, звучания отдельных музыкальных инструментов. Так, например, при записи музыкальных программ в помещении с неудовлетворительной акустикой или малого для данного состава исполнителей объёма обычно не удаётся получить необходимое соотношение между гулкостью и чёткостью звучания. В этом случае применение искусственной реверберации позволяет добиться улучшения качества звучания музыкальной программы. Аналогично, реверберация помогает создать необходимую акустическую окраску голоса или инструмента при записи вокалиста или солирующего инструмента, когда он «тонет» в звучании сопровождающего ансамбля.

В результате реверберации изменяется акустический частотный спектр звука. Высокие частоты ослабляются быстрее, чем низкие, вот почему тембр отраженного звука мягче и более заглушен по сравнению с исходным звуком. Величина затухания высоких частот зависит от расстояния, преодолеваемого звуковой волной, и свойств материала отражающих поверхностей. Этот сплошной звук с экспоненциально уменьшающейся интенсивностью и есть собственно реверберация. Часто его называют реверберационный хвост.

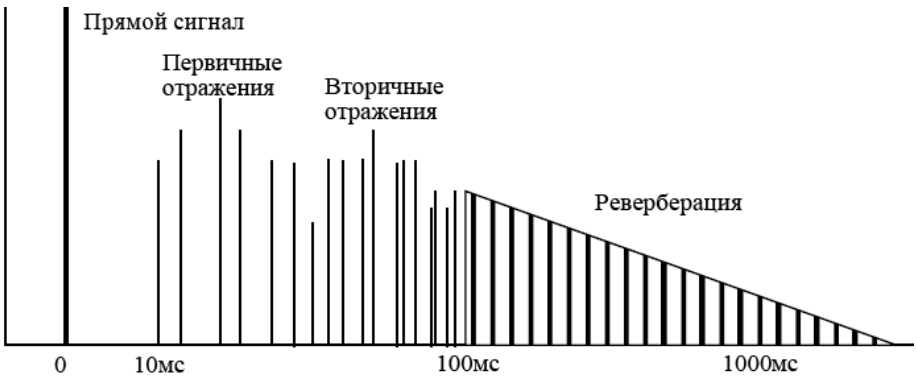


Рис. 5.22.

Чем больше число отражений и меньше временные интервалы между ними, тем выше плотность (диффузность) звуков в этом хвосте. Именно

диффузность звука создает ощущение объема помещения. Для ее увеличения необходимо, чтобы в создании реверберации принимало участие как можно большее число отражений сигналов от отражающих поверхностей. Поэтому в хороших залах вы не встретите простых плоских стен – для улучшения диффузности получаемого звукового поля их всегда делают изломанными, чтобы как можно больше увеличить число отражений.

Прямой сигнал несет информацию только о расположении источника звука справа или слева от слушателя. Какой-либо иной пространственной информации в нем не содержится. Ранние отражения несут в себе информацию о месте расположения исполнителя и о размерах помещения. Именно данные отражения вносят наибольший вклад в пространственное ощущение акустики зала. Завершающий участок реверберационного процесса определяет так называемую "гулкость" звучания, свойственную пустому помещению.

Звук реверберации в аудитории затухает со временем из-за потери энергии от поглощающего материала элементов отражения комнаты. Чем больше коэффициент отражения комнаты, тем дольше происходит затухание, и о комнате говорят «живая». Время затухания зависит и от громкости тестового звука. Это связано с тем, что затухающее колебание может оказаться ниже уровня шума помещения. Поэтому стандартное время реверберации определяется как интервал времени, в течение которого уровень звука уменьшается на 60 дБ относительно исходного значения. За исходное значение принято звуковое давление 100 дБ SPL, а уровень прекращения реверберации – 40 дБ SPL, что соответствует уровню шума в хороших концертных залах. Время реверберации может быть рассчитано по формуле

$$T_{60} = 163\frac{V}{\alpha S},$$

где – V объем комнаты (м^3), α – коэффициент поглощения поверхностей, S – общая площадь поверхностей (м^2). Для концертных залов время реверберации должно быть в пределах от 1,5 до 2,5 с.

Рассмотренные алгоритмы не учитывают, что звук, распространяющийся в зале, отражается не только от стен, но также и от кресел, пола и прочих поверхностей, которые порождают потоки дополнительных звуковых волн. Кроме того, каждая поверхность обладает свойством поглощения, в результате чего отраженный от этой поверхности сигнал может иметь несколько отличный от пришедшего сигнала спектр. По этой причине, для создания реалистичной реверберации пользуются значительно более сложными алгоритмами.

Наиболее полной характеристикой акустических свойств помещения является импульсный (реверберационный) отклик на единичное импульсное воздействие (рис. 5.23). Технически процедура получения импульсного отклика отработана для любых помещений. Она очень важна тем, что позволяет с помощью операции свертки импульсного отклика помещения и

звукового сигнала создать искусственно реверберацию такую же, как в любом концертном зале. Такая возможность широко используется в цифровых реверберaторах.

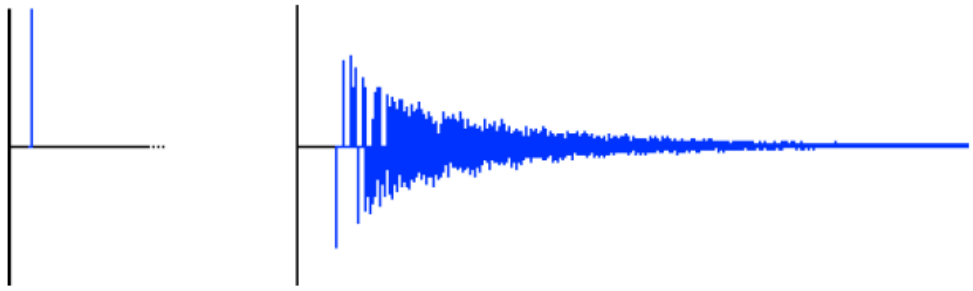


Рис. 5.23.

С помощью реверберации можно создать эффект приближения и удаления источника звука. Для этого постепенно изменяют уровень реверберации, создавая иллюзию изменения акустического отношения, а значит, и впечатление изменения звукового плана. Ниже представлена программная реализация такого эффекта.

```
int M=200;
int N;
for(int i=0;i<size;i++)
{
  if(i<2*M) nd[i]=data[i]/2;
  else
  {
          N=M*(1+sin(6.28*i/2000));
          nd[i]=0.5*data[i]+0.5*data[i-N];
  }
}
```

Базовой основой имитации поздних отражений в цифровых реверберaторах являются гребенчатый и все пропускающий фильтры. В первом используется только обратная связь, а втором – прямая и обратная связи. Их функциональные схемы приведены на рис. 5.24.

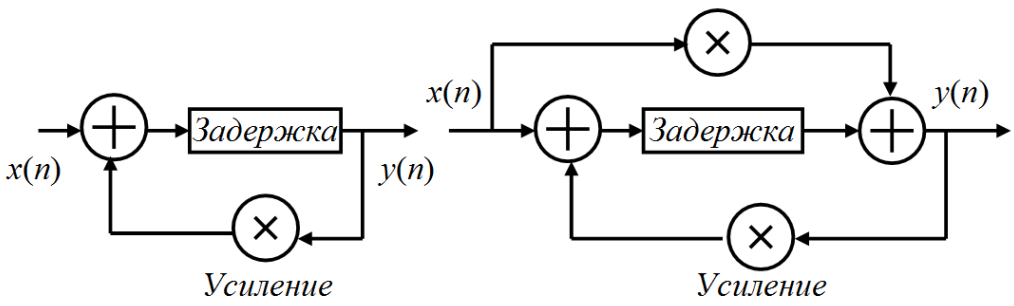


Рис. 5.24.

Передаточная функция гребенчатого фильтра:

$$H(z) = \frac{z^{-M}}{1 - gz^{-M}},$$

передаточная функция все пропускающего фильтра:

$$H(z) = \frac{z^{-M} - g}{1 - gz^{-M}},$$

где M – длина линии задержки, g – коэффициент усиления.

К классу эффектов, основанных на изменении формы, относятся все эффекты, тем или иным способом изменяющие форму и уровень звукового сигнала. Их диапазон простирается от простой регулировки уровня воспроизведения до таких сложных эффектов, как изменение динамического диапазона сигнала. Например, ниже представлен фрагмент программы, реализующей эффект модуляции амплитуды звукового сигнала. Изменяя величину периода модулирующего сигнала T и глубину модуляции, можно добиться эффекта «металлического голоса». Если значение T достаточно большое, создается эффект периодического приближения и отдаления источника звуковой волны.

```
int T=500;
for(int i=0;i<size;i++)
{
  nd[i]=(1+0.8*sin(6.28*i/T))*(data[i]/2);
}
```

Эффект детонации был характерен для кассетных магнитофонов и появлялся вследствие неравномерных (гармонических) изменений скорости ленты относительно считывающей головки. В некоторых случаях использование данного эффекта позволяет обогатить звуковую палитру произведения.

Поскольку в цифровых устройствах отсутствуют механические компоненты, в них отсутствует и эффект детонации. Для его реализации может быть использован цифровой фильтр, блок-схема которого приведена на рис. 5.25.

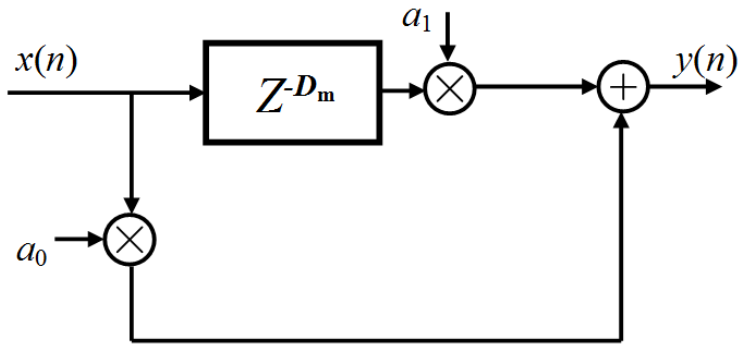


Рис. 5.25.

Данной блок-схеме соответствует следующее разностное уравнение:

$$y(n) = a_0 x(n) + a_1 x(n + f(n)),$$

$$f(n) = \frac{D}{2}\left[1 + \cos\left(\frac{2\pi n}{N} n\right)\right].$$

Программная реализация фильтра представлена ниже.

```
int N=1000;
int D=100;
int fn;
for(int n=0;n<size;n++)
{
        if(n<D) nd[n]=data[n];
        else
        {
                fn=(D/2)*(1+cos(6.28*n/N));
                nd[n]=0.5*data[n]+0.5*data[n-fn];
        }
}
```

Для предотвращения возникновения переполнения разрядной сетки при вычислении выходного значения коэффициент a_0 выбирается равным 0,5, коэффициент a_1 выбирается от –0,5 до 0,5. Значение параметра D, определяющего диапазон изменения задержки, обычно выбирается от 0,25 до 25 мс. Значение параметра N, определяющего частоту детонации, выбирается исходя из требований, предъявляемых к исходному сигналу. Уменьшение длины линии задержки приводит к пропорциональному подъему частот спектральных составляющих исходного сигнала, а ее уменьшение – к понижению этих частот.

Другим эффектом, обогащающим звуковую палитру произведения, является эффект хора. Он позволяет создать у слушателя иллюзию, что данное произведение одновременно исполняется на нескольких одинаковых музыкальных инструментах. Хотя музыканты стараются играть синхронно, они не могут избежать небольших отклонений от общего темпа. Кроме того, неизбежны различия в настройке отдельных инструментов и в громкости исполнения.

Использование данного эффекта позволяет. Например, приблизить исполнение произведения на шестиструнной гитаре к исполнению того же самого произведения на двенадцатиструнной гитаре.

Для создания эффекта одновременного исполнения произведения на двух инструментах может быть использован цифровой фильтр, блок-схема которого приведена на рис. 5.26.

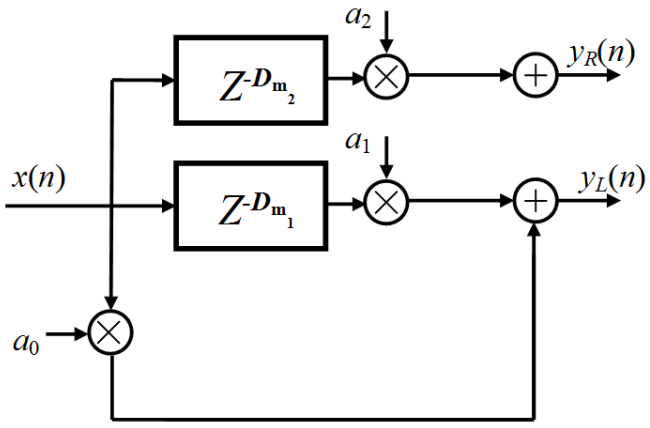


Рис. 5.26.

Ниже представлена программная реализация данного фильтра.

```
unsigned int D1, D2;
for(int n=0;n<size;n++)
{
  if(n<3000) nd[n]=data[n];
  else
  {
          if((n%3000)==0)
          {
                  D1=rand()%200;
                  D2=rand()%200;
          }
  nd[n]=0.4*data[n]+0.3*data[n-D1]+0.3*data[n-D2];
  }
}
```

Основным отличием этого фильтра от фильтра, реализующего эффект детонации, является закон изменения величины задержки сигнала. Если при создании эффекта детонации задержка изменяется по гармоническому закону, то при создании эффекта хора задержка изменяется случайным образом. Для получения качественного эффекта хора спектр модулирующего сигнала должен быть низкочастотным. Частота среза формирующего фильтра нижних частот должна составлять порядка 3 Гц. Задержка должна быть малой для избегания эффекта эхо, но больше 5 мс, в противном случае интерференция волны приведет к эффекту флэнжер. Часто задержанные сигналы чуть сдвигают по высоте для достижения более реалистичного эффекта ансамблевого звучания. Флэнжер (англ. flanger) – задержанный сигнал добавляется к оригинальному с переменной задержкой до 10 мс. Этот эффект достигается воспроизведением той же самой записи на двух синхронизированных проигрывателях и последующим микшированием. Для достижения эффекта оператор помещает свой палец на кромку (англ. flange) одного из дисков, несколько замедляя его движение, а значит и его звучание.

Когда оператор убирает свой палец, механизм ускоряет движение диска, синхронизируя его с другим. Программа, реализующая эффект, похожий на эффект флэнжер, представлена ниже.

```
unsigned int D1=0, D2=0;
for(int n=0;n<size;n++)
{
  if(n<2000) nd[n]=data[n];
  else
  {
          if((n%2000)==0)
          {
                  D1=(D1+rand()%80)/2;
                  D2=(D2+rand()%80)/2;
          }
  nd[n]=0.4*data[n]+0.3*data[n-D1]+0.3*data[n-D2];
  }
}
```

Для создания эффекта одновременного исполнения произведения на M инструментах может быть использован фильтр, блок-схема которого приведена на рис. 5.27.

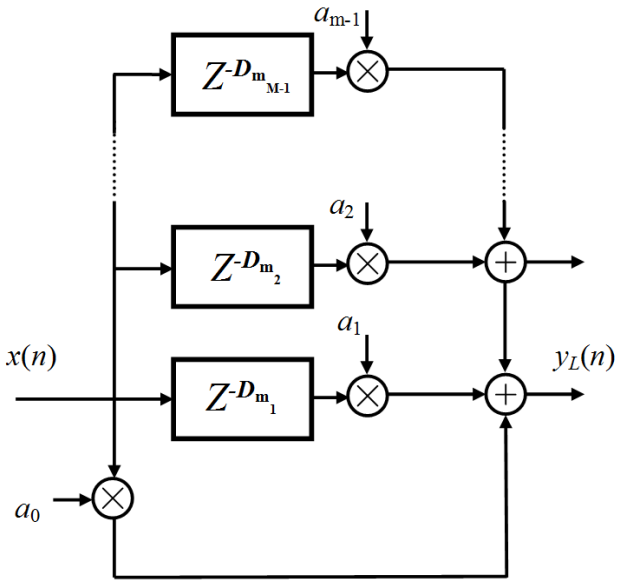


Рис. 5.27.

Как правило, все коэффициенты при реализации эффекта задержки выбираются равными $1/M$, а сами величины задержки не превышают 15–25 мс. Каждая линия задержки должна изменять свои параметры независимо от остальных. При таком выборе коэффициентов моделируются только различиия в темпе исполнения произведения и в настройке инструментов.

```
  unsigned int D1=950, D2=720, D3=550, D4=600;
  for(int n=0;n<size;n++)
  {
          if(n<1000) nd[n]=data[n];
          else
          {
                  if((n%8000)==0)
                          {
                                  D1=10+rand()%200;
                                  D2=10+rand()%200;
                                  D3=10+rand()%200;
                                  D4=10+rand()%200;
                          }
nd[n]=0.2*data[n]+0.2*data[n-D1]+0.2*data[n-D2]+
0.2*data[n-D3]+0.2*data[n-D4];
          }
  }
```

Для моделирования различного уровня воспроизведения достаточно сделать переменными коэффициенты фильтра, добавив к их постоянному значению некоторый случайный низкочастотный сигнал малого уровня, поскольку исполнители стараются сохранять средний уровень громкости при воспроизведении. Для простоты в качестве случайного низкочастотного сигнала может быть использован тот же самый сигнал, что и в линиях задержки. Для схемы на рис. 5.27 программа выглядит следующим образом:

```
unsigned int D1, D2;
double a1, a2;
for(int n=0;n<size;n++)
{
  if(n<3000) nd[n]=data[n];
  else
  {
          if((n%3000)==0)
          {
                  D1=rand()%200;
                  D2=rand()%200;
          }
  a1=(double)D1/400;
  a2=(double)D2/400;
  nd[n]=0.4*data[n]+a1*data[n-D1]+a2*data[n-D2];
  }
}
```

Эффект хора может быть применен для генерации псевдостереофонического сигнала. При использовании данного эффекта у слушателя будет возникать иллюзия перемещения источника сигнала в пространстве. Блок-схема такого фильтра приведена на рис. 5.28.

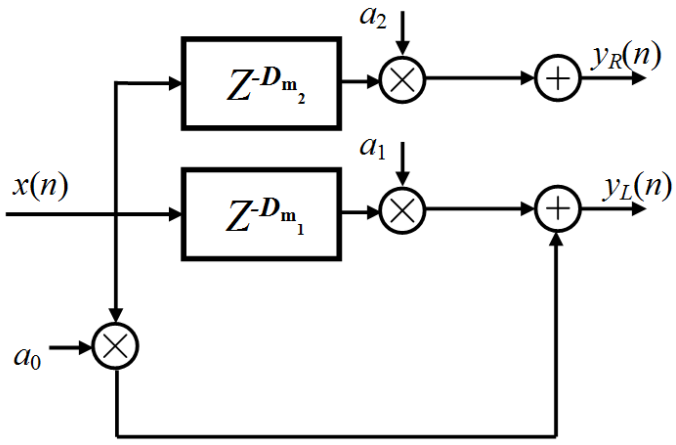


Рис. 5.28.

Наличие задержки в правом канале не обязательно, но позволяет усилить эффект пространственного перемещения источника звука. Наиболее впечатляющий эффект получается при использовании для модуляции гармонических сигналов одной и той же частоты. Сдвинутых друг относительно друга на π/4 радиан.

На рис. 5.29 представлено окно программы SpectraLab с результатами анализа сигнала, представляющего жужжание роя пчел. После обработки сигнала фильтром у слушателя создается впечатление движения вокруг него роя пчел. Период модулирующего гармонического колебания составляет 10 секунд, что придает большую реалистичность звучанию.

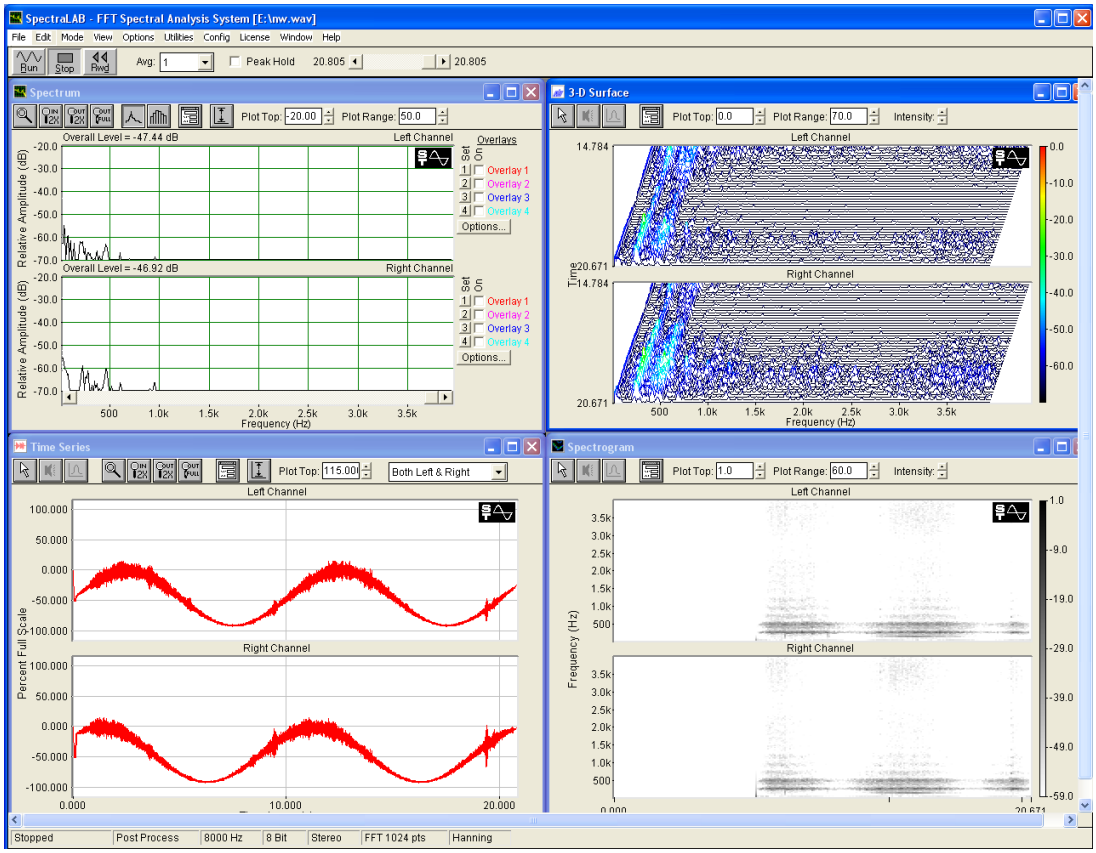


Рис. 5.29.

Программная реализация такого звукового эффекта представлена ниже.

```
#include "stdafx.h"
#include "WaveSample.h"
#pragma comment(lib,"winmm.lib")

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CWinApp theApp;
using namespace std;

struct WAVESTRUCT
{
  char formatId[4];
  int blockLength;
  char soundId[4];
  char subId[4];
  int headerLength;
  short formatType;
  short channels;
  int frequency;
  int dataRate;
  short sampleBytes;
  short sampleWidth;
  char dataId[4];
  int dataLength;
};

int main(int argc, TCHAR* argv[], TCHAR* envp[])
{
  int nRetCode = 0;
          if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
  {
          cerr << _T("Fatal Error: MFC initialization failed") << endl;
          nRetCode = 1;
  }
  else
  {
  byte * data;
  WAVESTRUCT sws;
  CFile nf;
  CStdioFile tf;
  char path[255]="e:\\s3.wav";
  if(!nf.Open(path, CFile::modeRead | CFile::typeBinary))
  {
          cout<<"File error!";
```

```
            return 0;
    };
    nf.Read(&sws,sizeof(sws));
    int size=sws.dataLength;
    data=new byte[size];
    nf.Read(data, size);
    nf.Close();

    CFile ff("e:\\nw.wav", CFile::modeWrite | CFile::typeBinary | CFile::modeCreate);
    byte * nd=new byte[size];
    memset(nd,0,size);

    sws.channels=2;
    sws.channels=2;
    sws.frequency=8000;
    sws.dataRate=16000;
    sws.sampleBytes=2;
    sws.sampleWidth=8;
    sws.dataLength=2*size;

    byte * nd1=new byte[size];
    byte * res=new byte[2*size];

    for(int i=0;i<size;i++)
    {
            nd[i]=data[i]*(0.55+0.45*sin(6.28*i/80000));
            nd1[i]=data[i]*(0.55+0.45*sin(6.28*i/80000+3.14/4));
    }
    for(i=0;i<size;i++)     res[2*i]=nd[i];
    for(i=0;i<size;i++)     res[2*i+1]=nd1[i];

    ff.Write(&sws,sizeof(sws));
    ff.Write(res,2*size);
    ff.Close();

  delete [] nd1;
    delete [] res;
    delete [] data;
    delete [] nd;
    }
    PlaySound("e:\\nw.wav",NULL,SND_FILENAME | SND_ASYNC);
    return nRetCode;
  }
```

5.4. Использование звуковой системы персонального компьютера

Обращение к звуковым системам в ОС Windows происходит по средствам специальных объектов, являющихся интерфейсами звукового оборудования (рис. 5.30). Использование таких объектов позволяет создавать вторичные звуковые буферы, данные из которых поступают и суммируются в первичном

звуковом буфере. Затем выбирается формат аудио-потока, в котором указываются частота дискретизации сигнала, кол-во выборок, количество вторичных буферов, кол-во звуковых каналов, разрядность потока и прочее.

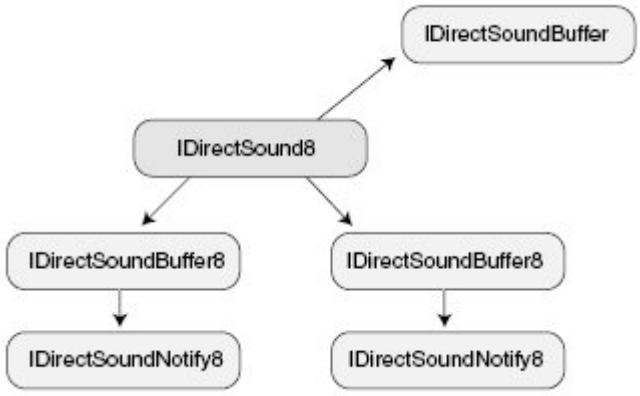


Рис. 5.30.

Для работы с объектами существуют стандартные библиотеки DirectSound, в которых реализованы основные методы и функции звуковых интерфейсов. Для работы с ними необходимо использовать Microsoft DirectX SDK, путь, к которому указывается в разрабатываемом проекте. Так же необходимо подключить ряд библиотек и заголовочных файлов, в зависимости от набора необходимых методов обработки аудио-потока.

В работе используется функции для работы с мультимедиа, которые имеются в стандартной библиотеке mmsystem.h.

Первичный звуковой буфер представляет собой кольцевой буфер (рис. 5.31).

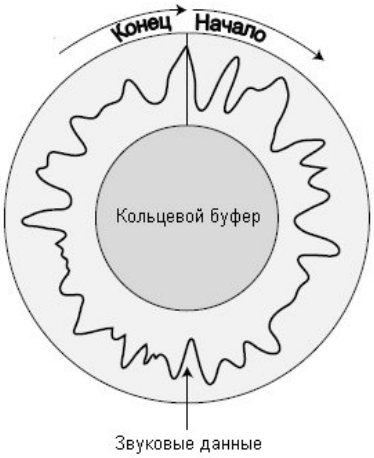


Рис. 5.31.

Для формирования непрерывного звукового потока используются вторичные буферы. На рис. 5.32 представлена схема, поясняющая принцип использования вторичных звуковых буферов.

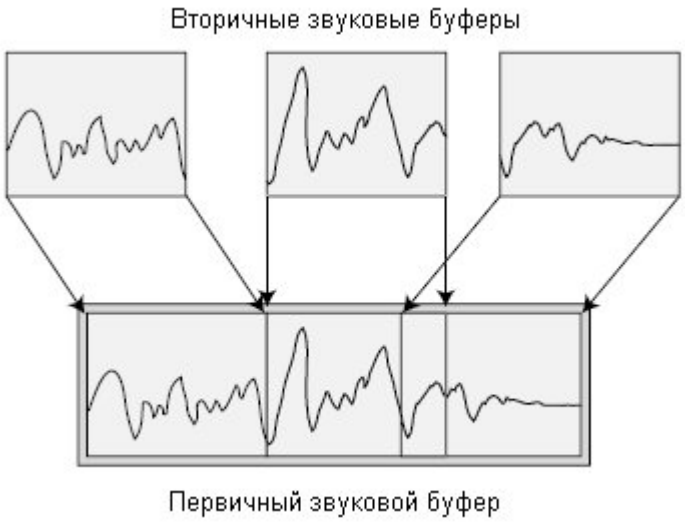


Рис. 5.32.

Для исследования алгоритммов обработки звуковых сигналов необходимо создать проект на основе диалогового приложения и добавить на диалоговое окно элемент «Button» (кнопка). В свойствах кнопки изменить текст в поле «Caption» (название) на «Record» (запись).

Подключаем заголовочные файлы и библиотеку для работы со звуком путем добавления в .cpp файл класса диалога следующего фрагмента кода:

```
#include "mmsystem.h"
#pragma comment(lib,"winmm.lib")
```

Описываем структуру, в которой содержится информация о звуковом фрагменте:

```
struct WAVESTRUCT
{
char formatId[4];        // Содержит символы "RIFF" в ASCII кодировке (0x52494646 в big-endian
// представлении). Является началом RIFF-цепочки.
int blockLength;         // Это оставшийся размер цепочки, начиная с этой позиции. Иначе говоря,
// это размер файла минус 8 байт.
char soundId[4];         // Содержит символы "WAVE" (0x57415645 в big-endian представлении)
char subId[4];           // Содержит символы "fmt " (0x666d7420 в big-endian представлении)
int headerLength;        // 16 для формата PCM. Это оставшийся размер подцепочки, начиная с этой
// позиции.
short formatType;        // Аудио формат, полный список можно получить здесь. Для PCM = 1 (то
// есть, Линейное квантование). Значения, отличающиеся от 1, обозначают
```

```
    // некоторый формат сжатия.
    short channels;  // Количество каналов. Моно = 1, Стерео = 2
    int frequency;          // Частота дискретизации. 8000 Гц, 44100 Гц и т.д.
    int dataRate;           // Количество байт, переданных за секунду воспроизведения.
    short sampleBytes;      // Количество байт для одного сэмпла, включая все каналы.
    short sampleWidth;      // Количество бит в сэмпле. Так называемая "глубина" или
точность
    // звучания. 8 бит, 16 бит и т.д.
    char dataId[4];  // Содержит символы "data" (0x64617461 в big-endian представлении)
    int dataLength;  // Количество байт в области данных.
    };
```

WAV файл (Windows PCM) представляет собой две, четко делящиеся, области. Одна из них – заголовок файла, другая – область данных. В заголовке файла хранится информация о:

– Размере файла.

– Количестве каналов.

– Частоте дискретизации.

– Количестве бит в сэмпле (эту величину еще называют глубиной звучания).

Для монофонического сигнала с дискретностью 8 бит звуковые данные пред ставляют собой массив однобайтовых значений, каждое из которых является выбор-кой сигнала.

Для стереофонического сигнала с дискретностью 8 бит звуковые данных имеют формат массива двухбайтовых слов, причем младший байт слова соответствует левому каналу, а старший – правому.

Формат звуковых данных с дискретностью 16 бит выглядит аналогично. Для монофонического сигнала данные хранятся в массиве 16-битовых слов. Для стерео-фонического используется массив двойных слов, причем младшему слову соот-ветствует левый канал, а старшему – правый.

Диапазон изменения значений выборок сигнала определяется дискретизацией. Для 8-битовых данных он составляет от 0 до 255 (0xff), причем отсутствию сигнала (полной тишине) соответствует значение 128 (0x80). Для 16-битовых данных диапа-зон изменения составляет от -32768 (-0x8000) до 32767, (0x7fff), отсутствию сигнала соответствует значение 0.

В обработчике нажатия на кнопку «Record» выполняется реализация алгоритмов записи, частотной и динамической коррекции, а также воспроизведения звука. Текст программы представлен далее.

В обработчике нажатия на кнопку запись добавляем код:

```
void CSound3391Dlg::OnButton14()
{
HWAVEIN microHandle;// идентификатор устройства, полученный функцией waveInOpen;
WAVEHDR waveHeader;     // структура содержит описание буфера и ссылку на массив
    // передаваемых данных
```

```
const int t=5;

const int NUMPTS = 22050 * t;       // количество выборок за t секунд
int sampleRate = 22050;             // частота дискретизации
short int waveIn[NUMPTS];           // 'short int' это 16-разрядный тип данных
                                    // для 8-разрядных используется 'unsigned char' or 'BYTE'
MMRESULT result = 0;

WAVEFORMATEX format;
format.wFormatTag=WAVE_FORMAT_PCM;      // простой, не сжатый формат
format.wBitsPerSample=16;                   //  16 для высого качества, 8 для телефонного
format.nChannels=1;                             //  1=моно, 2=стерео
format.nSamplesPerSec=sampleRate;               //  22050
format.nAvgBytesPerSec=format.nSamplesPerSec*format.nChannels*format.wBitsPerSample/8;
        // = nSamplesPerSec * n.Channels * wBitsPerSample/8
format.nBlockAlign=format.nChannels*format.wBitsPerSample/8;
                                    // = n.Channels * wBitsPerSample/8
format.cbSize=0;

result = waveInOpen(&microHandle, WAVE_MAPPER, &format, 0L, 0L,
WAVE_FORMAT_DIRECT);

/* Комментарий
Функция
UINT waveInOpen(
LPHWAVEIN lphWaveIn,            // указатель на идентификатор устройства
UINT wDeviceID,                 // номер открываемого устройства
LPWAVEFORMAT lpFormat,          // указатель на структуру WAVEFORMAT
DWORD dwCallback;               // адрес функции обратного вызова
// или идентификатор окна
DWORD dwCallbackInstance,        // данные для функции обратного вызова
DWORD dwFlags);                 // режим открытия устройства
```

Открывает устройство записи (ввода).

В параметр lphWaveIn в результате работы возвращается указатель на переменную типа HWAVEIN, который необходим для выполнения всех операций с данным устройством ввода.

Через параметр wDeviceID приложение передает функции waveInOpen номер устройства ввода,которое оно собирается открыть или константу WAVE_MAPPER, если нужно чтобы Было выбрано устройство ввода по умолчанию, определенное в операционной системе.

Через параметр lpFormat приложение должно передать функции указатель на предварительно заполненную структуру WAVEFORMAT, которая имеет следующую структуру:

```
*/

if (result)
{
   Sleep(1000);
   return;
```

```
}

// Настройка и подготовка структуры заголовка данных для записи
waveHeader.lpData = (LPSTR)waveIn;
waveHeader.dwBufferLength = NUMPTS*2;
waveHeader.dwBytesRecorded=0;
waveHeader.dwUser = 0L;
waveHeader.dwFlags = 0L;
waveHeader.dwLoops = 0L;
waveInPrepareHeader(microHandle, &waveHeader, sizeof(WAVEHDR));

/* Комментарий
Функция
UINT waveInPrepareHeader(
HWAVEIN hWaveIn,                // идентификатор устройства
LPWAVEHDR lpWaveInHdr,          // указатель на структуру WAVEHDR
UINT wSize);                    // размер структуры WAVEHDR
```

Подготавливает буфер к передаче устройству ввода.
Параметры:
hWaveIn - идентификатор устройства, полученный функцией waveInOpen;
В параметре lpWaveInHdr программа передает в функцию указатель на структуру типа WAVEHDR, содержащую описание буфера и ссылку на массив передаваемых данных. Структура WAVEHDR выглядит так:

```
typedef struct wavehdr_tag {
 LPSTR lpData;          // адрес буфера данных - ссылка на символьный массив
 DWORD dwBufferLength;  // размер буфера данных
 DWORD dwBytesRecorded; // количество записанных байт (только при записи)
 DWORD dwUser;          // пользовательские данные
 DWORD dwFlags;         // флаги состояния буфера данных
 DWORD dwLoops;  // количество повторений (только при воспроизведении)
 struct wavehdr_tag far * lpNext; // зарезервировано
 DWORD reserved;        // зарезервировано
 } WAVEHDR;
*/

// Добавление первичного буфера
result = waveInAddBuffer(microHandle, &waveHeader, sizeof(WAVEHDR));

/* Комментарий
Функция
UINT waveInAddBuffer(
HWAVEIN hWaveIn,          // идентификатор устройства
LPWAVEHDR lpWaveInHdr, // указатель на структуру WAVEHDR
UINT wSize);              // размер структуры WAVEHDR
```

Передает подготовленный буфер памяти драйверу устройства ввода. Параметры аналогичны функции waveInPrepareHeader.

```
*/
```

```
        if (result)
        {
           Sleep(1000);
           return;
        }

        result = waveInStart(microHandle);

        /* Комментарий
        Функция
        UINT waveInStart (HWAVEIN hWaveIn) // идентификатор устройства, полученный
waveInOpen.
        Запускает процесс ввода данных.
        */

        if (result)
        {
           Sleep(1000);
           return;
        }

        // Ожидание пока выполняется запись данных в буфер
        do {} while (waveInUnprepareHeader(microHandle, &waveHeader,
sizeof(WAVEHDR))==WAVERR_STILLPLAYING);

        /* Комментарий
          Функция
        UINT waveInUnprepareHeader(
        HWAVEIN hWaveIn,              // идентификатор устройства ввода
        LPWAVEHDR lpWaveInHdr,  // указатель на структуру WAVEHDR
        UINT wSize);                      // размер структуры WAVEHDR

        Освобождает буфер ввода.
        */

        waveInClose(microHandle);

        /* Комментарий
        Функция
        UINT waveInClose(HWAVEIN hWaveIn);   // идентификатор устройства ввода
        Закрывает устройство ввода.
        */

        // воспроизведение звука
        int Fd=22050;

        WAVESTRUCT sws;
        strcpy(sws.formatId,"RIFF");
        sws.blockLength=sizeof(data)+36;
        strcpy(sws.soundId,"WAVE");
```

```
strcpy(sws.subId,"fmt ");
sws.headerLength=16;
sws.formatType=1;
sws.channels=1;
sws.frequency=Fd;
sws.dataRate=Fd;
sws.sampleBytes=2;
sws.sampleWidth=16;
strcpy(sws.dataId,"data");
sws.dataLength=Fd*sizeof(short)*t;

// динамическая коррекция сигнала
for(int i=0;i<NUMPTS;i++)
{
if(waveIn[i]<8096) waveIn[i]= waveIn[i]*2;  // усиление слабого сигнала в 2 раза
}

// запись данных в файл для анализа в программе SpectraLAB
CFile nf;
nf.Open("c:\\test.wav",CFile::modeCreate | CFile::modeWrite);
nf.Write(&sws, sizeof(sws));
nf.Write(waveIn, sizeof(waveIn));
nf.Close();

// воспроизведение записанного звукового фрагмента
sndPlaySound((LPCSTR)&sws, SND_MEMORY | SND_ASYNC);
}
```

В программе используется структура WAVEFORMATEX, которая является расширенным вариантом структуры WAVEFORMAT. Описание структуры WAVEFORMAT представлено ниже.

```
typedef struct waveformat_tag {
 WORD wFormatTag; // тип формата, всегда – WAVE_FORMAT_PCM
 WORD nChanels; // количество каналов (1 – моно, 2 – стерео)
 DWORD nSamplesPerSec; // частота дискретизации
 DWORD nAvgBytesPerSec; // скорость потока данных
 WORD nBlockAlign; // выравнивание блока данных
} WAVEFORMAT;
```

nAvgBytesPerSec вычисляется как произведение числа каналов, частоты дискретизации и количества байт в одном отсчете. Количество байт в отсчете, естественно, равно размерности отсчета в битах, разделенному на 8 (именно столько бит находится в одном байте).

nBlockAlign вычисляется как произведение количества каналов на количество байт в одном отсчете.

Четвертый параметр функции waveInOpen – dwCallback, – объект, которому будут передаваться уведомления о выполнении запрошенных операций, его смысл зависит от значения реквизита dwFlags.

Если dwFlags содержит CALLBACK_NULL, реквизит dwCallback игнорируется и сообщения от драйвера не обрабатываются, если dwFlags содержит CALLBACK_WINDOW – dwCallback должен содержать дескриптор (типа HWND) окна, оконная процедура котрого должна обработать сообщения от драйвера (главным образом нас интересует MM_WIM_DATA – сообщение, посылаемое в момент заполнения переданного драйверу буфера данными из входного потока). Если dwFlags содержит CALLBACK_EVENT, реквизит dwCallback является дескриптором события, если dwFlags содержит CALLBACK_THREAD, реквизит dwCallback является дескриптором потока, если dwFlags содержит CALLBACK_FUNCTION, реквизит dwCallback является указателем функции обратного вызова.

Для открытия синхронного устройства (такого, которое останавливает работу приложения до завершения записи или воспроизведения) dwFlags должно содержать также (через логическую операцию ИЛИ) флаг WAVE_ALLOWSYNC.

Функция waveInOpen возвращает значения:

MMSYSERR_NOERROR – успешное выполнение,

MMSYSERR_BADDEVICED – неправильный номер устройства,

MMSYSERR_ALLOCATED – устройство уже открыто,

MMSYSERR_NOMEM – недостаточно памяти,

WAVERR_BADFORMAT – указанный формат не поддерживается драйвером устройства вывода,

WAVERR_SYNC – была выполнена попытка открыть синхронное устройство

Описание остальных функций библиотеки можно найти по ссылке: http://speech-text.narod.ru/chap1_3_3.html

5.5. Частотная коррекция и динамическая обработка звуковых сигналов, полученных с микрофона

На рис. 5.33 представлен 3D-спектр исходного записанного звукового фрагмента. Линиями отмечены наиболее ярко выраженные спектральные составляющие записанного звукового фрагмента, по которым удобно анализировать изменение спектра сигнала после частотной коррекции.

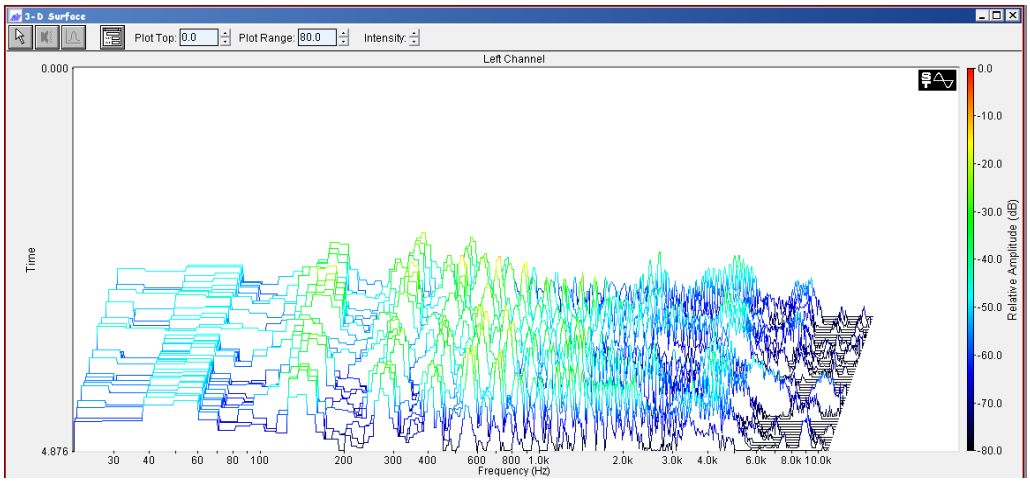


Рис. 5.33.

После частотной коррекции спектр сигнала сместился в область высоких частот (рис. 5.34) и в область низких частот (рис. 5.35).

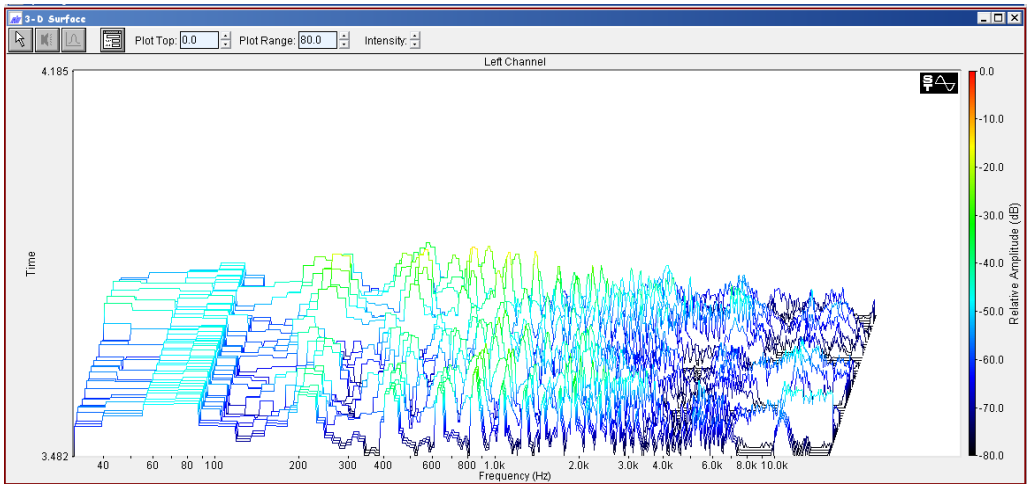


Рис. 5.34.

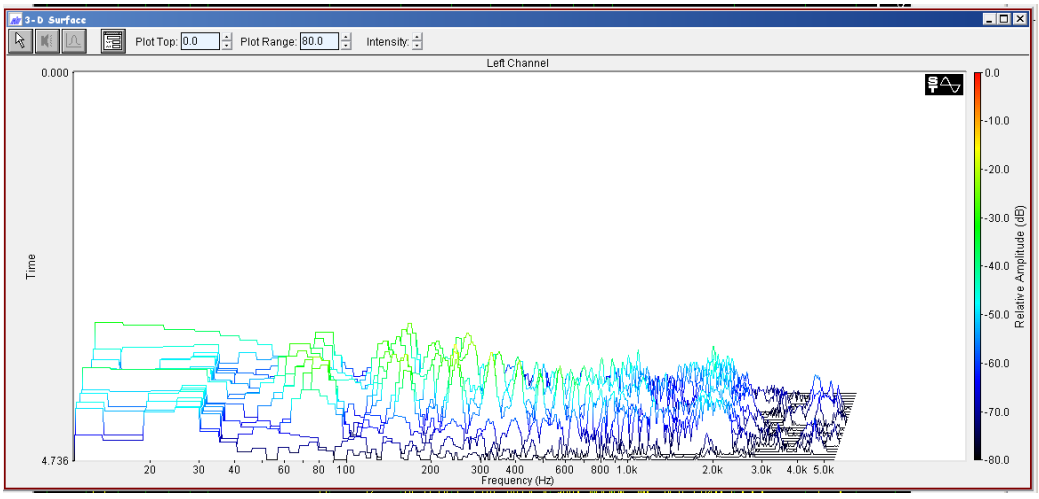


Рис. 5.35.

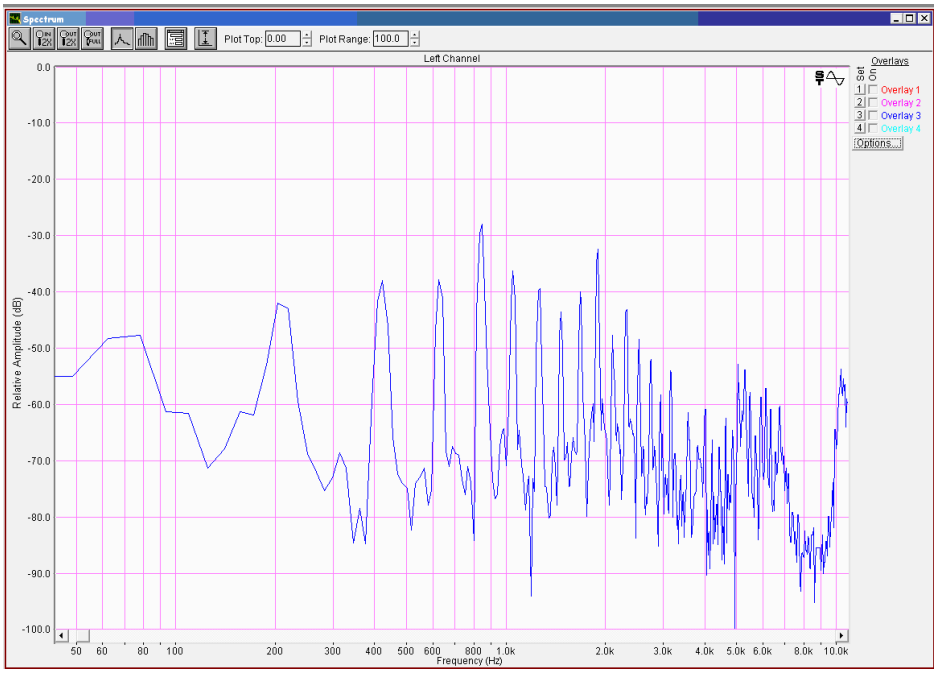


Рис. 5.36.

Использование методов и структур библиотеки мультимедиа (mmsystem.h) позволяет использовать звуковую карту в качестве простейшего осциллографа с полосой пропускания до 22050 Гц, наличием функций захвата и цифровой обработки сигнала, функций формирования звукового потока и его конфигурации. Так же возможно использовать звуковую карту в качестве анализатора спектра аудио-потока, определения амплитудно-частотных характеристик, системы определения звуковых образов и других технических задач.

Задачи для самостоятельного решения

1. Разработать программу, выполняющую имитацию множественных переотражений. Для имитации более сложных переотражений может быть использован фильтр, блок-схема которого приведена на рис. 5.37.

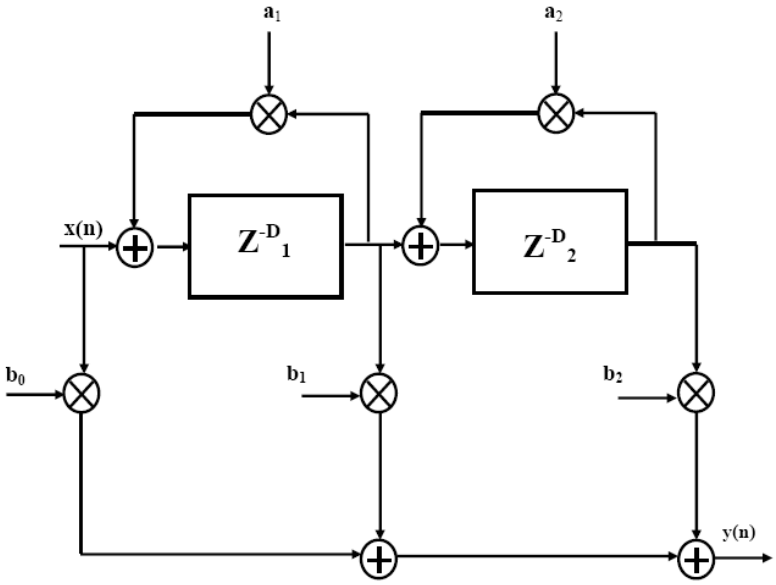


Рис. 5.37.

Передаточная функция этого фильтра имеет вид:

$$H(z) = b_0 + b_1\left[\frac{z^{-D_1}}{1 - a_1 z^{-D_1}}\right] + b_2\left[\frac{z^{-D_1}}{1 - a_1 z^{-D_1}}\right]\left[\frac{z^{-D_2}}{1 - a_2 z^{-D_2}}\right].$$

2. Разработать программу, выполняющую изменение динамического диапазона сигнала, т. е. для уменьшения различия уровней самого тихого и самого громкого сигналов. Для этого при превышении текущим значением сигнала некоторого порога устанавливается коэффициент передачи, меньший, чем используемый до превышения порога. Отношение старого коэффициента передачи к новому называется ослаблением. Все коэффициенты передачи задаются в логарифмической шкале. То есть при ослаблении 2:1 приращению входного сигнала на 20 дБ будет соответствовать приращение выходного сигнала на 10 дБ. Подобный компрессор, изменяющий свою передаточную характеристику в точке a, может быть реализован по следующей формуле:

$$y = \begin{cases} x, |x| \le \alpha \\ \operatorname{sgn}(x)\sqrt{\dfrac{|x|}{\alpha}}, |x| > \alpha \end{cases}$$

Применение компрессоров позволяет передавать по линиям связи сигналы, имеющие больший динамический диапазон, чем передающий канал. На приемном конце этой линии устанавливается экспандер, имеющий передаточную характеристику, обратную характеристике компрессора.

3. Разработать конвертер частоты дискретизации. Наиболее просты алгоритмы изменения частоты дискретизации в целое число раз. При уменьшении частоты дискретизации в N раз частота Найквиста (половина частоты дискретизации) становится в N раз ниже, т.е. частотный диапазон сужается. Поэтому для предотвращения наложения спектра (алиасинга) применяют НЧ-фильтр, подавляющий все частотные составляющие выше будущей частоты Найквиста. После фильтрации отсчеты сигнала прореживаются в N раз. При этой операции спектр сигнала ниже новой частоты Найквиста остается неискаженным. Для увеличения частоты дискретизации в M раз сигнал сначала интерполируется («разбавляется») нулями. Это сохраняет неизменным спектр сигнала ниже частоты Найквиста, но создает копии спектра выше частоты Найквиста. После этого возникшие копии спектра отфильтровываются НЧ-фильтром.

Для передискретизации сигнала в нецелое число раз (например, из 96 кГц в 44,1 кГц) можно скомбинировать повышение и понижение частоты дискретизации в целое число раз (например, 44100 =96000×M/N = 96000×147/320). Поскольку НЧ-фильтрация выполняется после повышения частоты дискретизации в M раз, но до понижения ее в N раз, то две фильтрации можно совместить в одну, установив частоту среза фильтра на минимум из двух необходимых частот среза. Фильтр в данном случае работает над сигналом с повышенной в M раз частотой дискретизации.

4. Выполнить динамическую обработку звукового сигнала. Принцип цифровой динамической обработки основан на почти мгновенном изменении коэффициента передачи сигналов, когда уровень огибающей звукового сигнала становится выше (ниже) установленного порога.

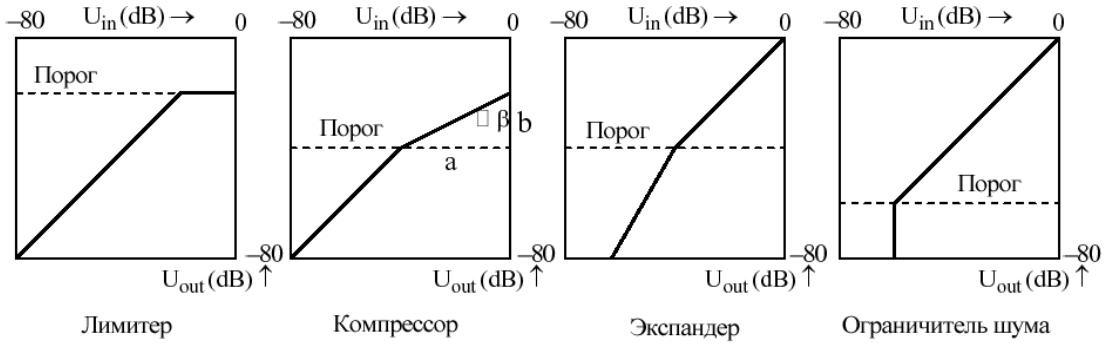


Рис. 5.38.

Порог может задаваться как пиковым значением уровня, так и среднеквадратическим. В зависимости от вида передаточной характеристики динамический процессор может выполнять функции лимитера, компрессора, экспандера или ограничителя шума. Все эти функции могут быть реализованы с помощью одного программного модуля.

СПИСОК ЛИТЕРАТУРЫ

1. Агеев Д. В. Основы теории линейной селекции / Д. В. Агеев // Научно-технический сборник ЛЭИС. — 1935. — № 10. — С. 29-37.
2. Антонью А. Цифровые фильтры: анализ и проектирование: Пер. с англ. / А. Антонью. — М. : Радио и связь, 1983. — 320 с. : ил.
3. Атаев Д. И. Аналоговые микросхемы для бытовой аппаратуры. Справочник. / Д. И. Атаев, В. А. Болотников. — М. : МЭИ, ПКФ «Печатное дело», 1992. — 240 с. : ил.
4. Баева Н. Н. Многоканальная электросвязь и РРЛ : Учебник для вузов связи. / Н. Н. Баева, И. К. Бобровская и др. — М. : Радио и связь, 1984. — 216 с. : ил.
5. Баскаков С. И. Радиотехнические цепи и сигналы: Учебник. / С. И. Баскаков. — М. : Высшая школа, 1983. — 536 с. : ил.
6. Бонч-Бруевич А. М. Системы спутниковой связи / А. М. Бонч-Бруевич, В. Л. Быков, Л. Я. Кантор и др.; Под ред. Л. Я. Кантора : Учеб. пособие для вузов. — М. : Радио и связь, 1992. — 224 с. : ил.
7. Варакин Л. Е. Системы связи с шумоподобными сигналами / Л. Е. Варакин. – М. : Радио связь, 1985. — 384 с. : ил.
8. Голышко А. Торжество цифровой сотовой связи / А. Голышко // Радио. — 2001. — №2. — С.70-71.
9. Голышко А. Основа будущей сотовой связи / А. Голышко // Радио. — 2001. — №3. — С. 68-70.
10. Гольденберг Л. М. Цифровая обработка сигналов: Учебное пособие для вузов / Л. М. Гольденберг, Б. Д. Матюшкин, М. Н. Поляк. — М. : Радио и связь, 1990. — 256 с. : ил.
11. Гоноровский И. С. Радиотехнические цепи и сигналы: Учебное пособие для вузов. — 5-е изд., перераб. и доп. / И. С. Гоноровский, М. Н. Демин — М. : Радио и связь, 1994. — 480 с. : ил.
12. Даджион Д. Цифровая обработка многомерных сигналов: Пер. с англ. / Д. Даджион, Р. Мерсеро. — М. : Мир, 1988. — 488 с. : ил.
13. Дворецкий И. М. Цифровая передача сигналов звукового вещания / И. М. Дворецкий, И. Н. Дриацкий. — М. : Радио и связь, 1987. — 192 с.
14. Карташев В. Г. Основы теории дискретных сигналов и цифровых фильтров: Учебное пособие для вузов / В. Г. Карташев. — М. : Высшая школа, 1982. — 109 с. : ил.
15. Казанцев Г. Д. Измерительное телевидение: Учебное пособие для вузов / Г. Д. Казанцев, М. И. Курячий, И. Н. Пустынский. — М. : Высшая школа, 1994. — 288 с. : ил.
16. Каппелини В. Цифровые фильтры и их применение / В. Каппелини, А. Константинидис, П. Эмилиани. — М.: Энергоатомиздат, 1983. — 360 с. : ил.

17. Куприянов М. С. Цифровая обработка сигналов: процессоры, алгоритмы, средства проектирования. — 2-е изд. / М. С. Куприянов, Б. Д. Матюшкин. — СПб. : Политехника, 1999. — 592 с. : ил.
18. Мелихов С. В., Титов А. А. Радиовещание, связь и электроакустика : Учебно-методическое пособие по практическим занятиям для студентов радиотехнических специальностей / С. В. Мелихов, А. А. Титов. — Томск: Том. гос. ун-т систем управления и радиоэлектроники, 2000. — 49 с. : ил.
19. Оппенгейм А. В. Цифровая обработка сигналов: Пер. с англ. / А. В. Оппенгейм, Р. В. Шафер. — М. : Связь, 1979. — 416 с. : ил.
20. Оппенгейм А. Применение цифровой обработки сигналов: Пер. с англ. / А. Оппенгейм. — М. : Мир, 1980. — 552 с. : ил.
21. Остапенко А. Г. Рекурсивные цифровые фильтры на микропроцессорах / А. Г. Остапенко, А. Б. Сушков, В. В. Бутенко. — М. : Радио и связь, 1988. — 128 с. : ил.
22. Поляков А. К. Языки VHDL и Verilog в проектировании цифровой аппаратуры. — М. : СОЛОН-Пресс, 2003. — 320 с. : ил.
23. Ратынский М. В. Основы сотовой связи / М. В. Ратынский, Д. Б. Зимин. Под. ред. Д. Б. Зимина. — М. : Радио и связь, 2000. — 248 с. : ил.
24. Радиовещание и электроакустика : Учебник для вузов / Под ред. Ю. А. Ковалгина. — М.: Радио и связь, 2000. — 792 с. : ил.
25. Рабинер Л. Теория и применение цифровой обработки сигналов: Пер. с англ. / Л. Рабинер, Б. Гоулд. — М. : Мир, 1978. — 848 с. : ил.
26. Рябенький В. М., Ушкаренко О. О. Verilog. Практика проектування цифрових пристроїв на ПЛІС. Навчальний посібник / В.М. Рябенький, О. О. Ушкаренко. — Миколаїв : Вид-во «Іліон», 2007. — 324 с. : ил.
27. Рябенький В.М., Ходаков В.Е., Ушкаренко А.О. Компьютерное управление внешними устройствами через стандартные интерфейсы. Учебное пособие / В. М. Рябенький, В. Е. Ходаков. А. О. Ушкаренко. — Херсон : Олди-плюс, 2008. — 380 с. : ил.
28. Системы радиосвязи / Под ред. Н. И. Калашникова. — М. : Радио и связь, 1988. — 352 с. : ил.
29. Соловьев В. В. Проектирование цифрових систем на основе программируемых логических интегральных схем / В. В. Соловьев. — М.: Горячая линия — Телеком, 2001. — 636 с. : ил.
30. Угрюмов Е. П. Цифровая схемотехника / Е. П. Угрюмов. — СПб. : БХВ-Петербург, 2000. — 528 с. : ил.
31. Уоткинсон Д. Пособие для инженеров по цифровому сжатию. / Д. Уоткинсон. — М.: «Снелл и Уилкокс», 1994. — 64 с. : ил.
32. Феер К. Беспроводная цифровая связь. Методы модуляции и расширения спектра: Пер. с англ. / К. Феер. Под ред. В. И. Журавлева. — М. : Радио и связь, 2000. — 520 с. : ил.

СОДЕРЖАНИЕ

УДК 681.3.06(075.8)
ББК 32.884.1
Р 98

Рябенький В. М.

Р 98 Програмна реалізація алгоритмів цифрової обробки сигналів : монографія / В. М. Рябенький, О. О. Ушкаренко. – Миколаїв : НУК, 2016. – 244 с.

У монографії розглянуті базові питання дискретизації сигналів за часом, їх цифрового уявлення, спектрально-кореляційного аналізу, принципів модуляції сигналів, цифрової обробки зображень. Основний акцент робиться на програмну реалізацію алгоритмів цифрової обробки сигналів, зокрема цифрових фільтрів, які можуть використовуватися в системах мультимедіа, цифрового зв'язку, телекомунікації, обробці зображень і цифрове телебачення. Наводиться велика кількість прикладів програм, розглянуто теоретичні основи цифрової обробки сигналів і пропонуються завдання для самостійного рішення.

Монографія рекомендується фахівцям в області цифрової обробки сигналів, студентам, які навчаються за напрямом «Телекомунікації», а також студентам інших напрямів і спеціальностей, в освітніх програмах яких передбачено вивчення дисципліни «Цифрова обробка сигналів» або споріднених дисциплін.

Наукове видання

РЯБЕНЬКИЙ Володимир Михайлович
УШКАРЕНКО Олександр Олегович

ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ

Монографія

(*російською мовою*)

Комп'ютерне складання та верстання *В. В. Москаленко*

Підписано до друку 08.07.2016 р. Формат 60×84/16. Ум. друк. арк. 14,3. Тираж 300 прим. Зам. № 129.
Видавець і виготівник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв Сталінграда, 9, м. Миколаїв, 54025
E-mail : publishing@nuos.edu.ua
Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК