Государственный университет телекоммуникаций им. Бонч-Бруевича
кафедра «Цифровой обработки сигналов»

И. И. Гук,
доцент кафедры ЦОС
gook_igor@mail.ru

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе № 3

«Программная реализация на языке С++ файловой модели КИХ фильтра»

2008 год

Оглавление

1. Создание проекта

Необходимо выполнить действия, описанные в первом разделе методических указаниях к лабораторной работе № 1, а именно:

1. Создать папку для хранения файлов проекта.
2. Запустить программу *wvDev-C++*.
3. Создать проект.
4. Откомпилировать созданный проект.

Для примера, был создан проект с именем *prjKIX*. После выполнения всех действий окно программы *wxDev-C++* имеет вид, показанный на рис.1.

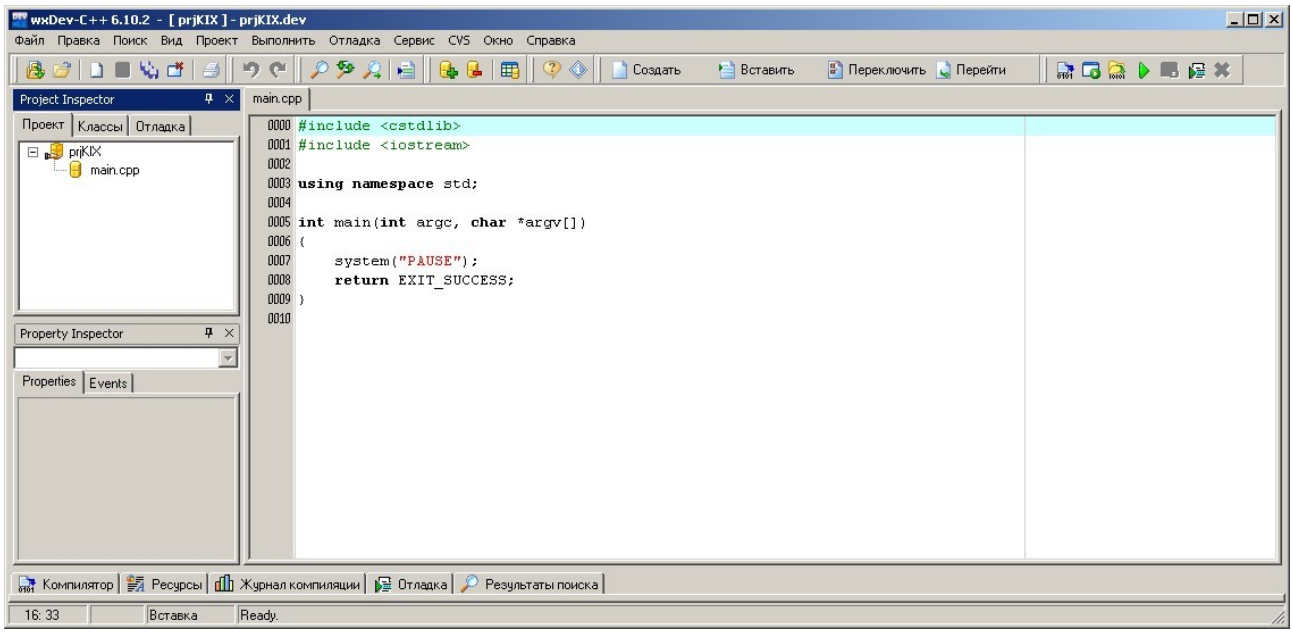

Рис.1. Вид программы wxDev-C++ после создания проекта prjKIX.

Для упрощения задачи, скопируем все файлы, которые были созданы в лабораторной работе № 1 (напомним, это файлы «*main.cpp*», «*runMPY.cpp*», «*initMPY.cpp*», «*constant.cpp*» и «*prjMPY.h*») в папку вновь созданного проекта. При этом, программа выдаст предупреждение об изменении файла «*main.cpp*» (см.рис.2). Подтвердите принятие изменений нажав кнопку «*Yes*».

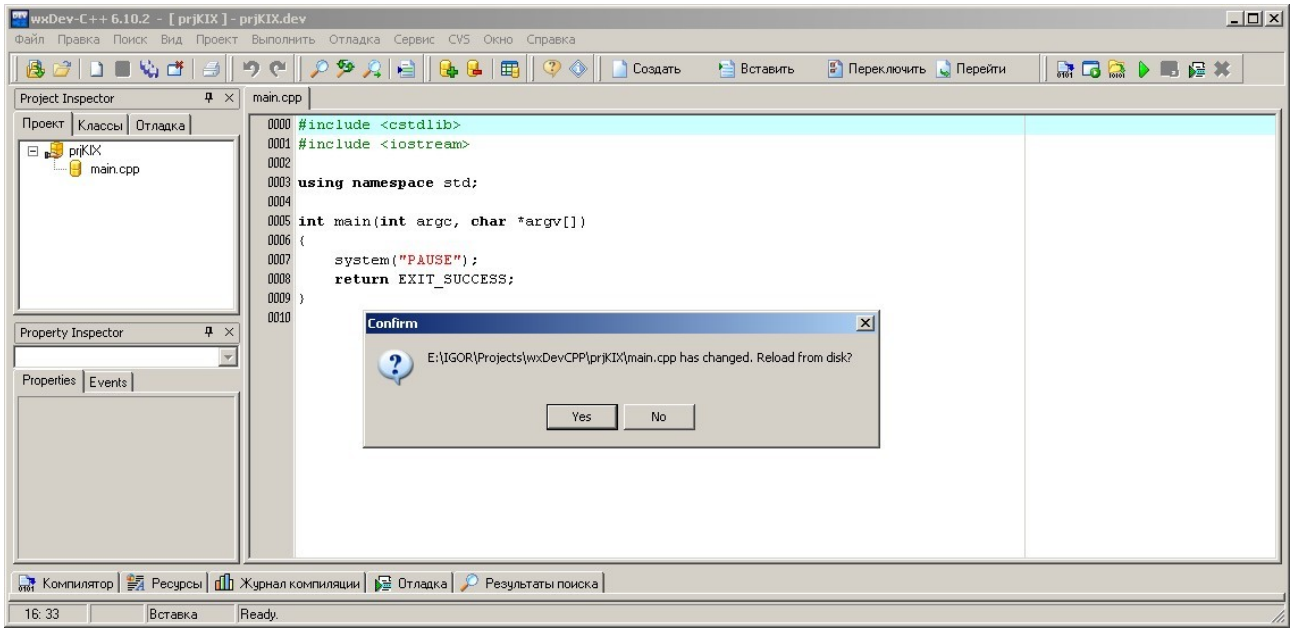

Рис.2. Запрос подтверждения изменений файла «main.cpp».

Вид программы *wxDev-C++* измениться (см.рис.3). Вместо созданного самой программой

файла «*main.cpp*», появиться файл, который мы скопировали в папку проекта. Это файл был создан в лабораторной работе №2.

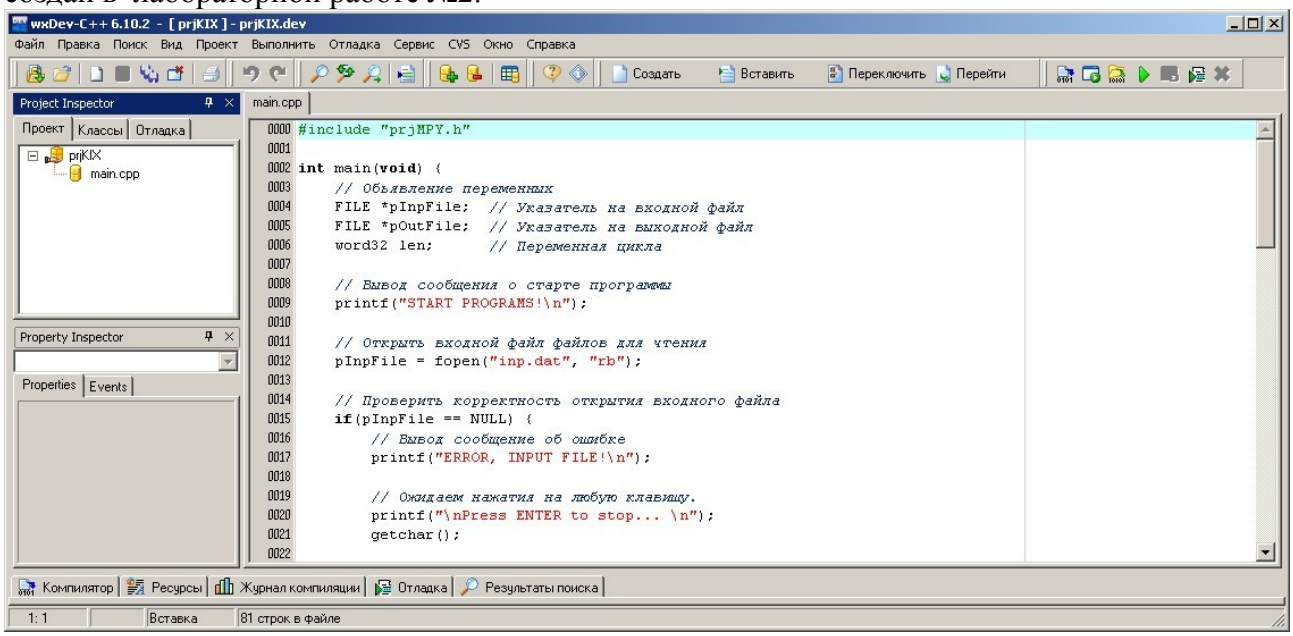

Рис.3. Вид программы wxDev-C++ после замены файла «main.cpp».

Теперь необходимо переименовать скопированные файлы и внести изменения в текст программного кода.

Переименовываем файлы в соответствии со следующими рекомендациями:

- «*main.cpp*» — переименование не производиться.
- «*runMPY.cpp*» — переименовываем в «*runKIX.cpp*».
- «*initMPY.cpp*» — переименовываем в «*initKIX.cpp*».
- «*constant.cpp*» — переименование не производиться.
- «*prjMPY.h*» — переименовываем в «*prjKIX.h*».

Затем подключаем к проекту файлы «*runKIX.cpp*», «*initKIX.cpp*», «*constant.cpp*» и «*prjMPY.h*». Для чего в главном меню выбираем пункт «*Проект->Добавить к проекту*» (см.рис.4) и в появившемся окне указываем подключаемые к проекту файлы.

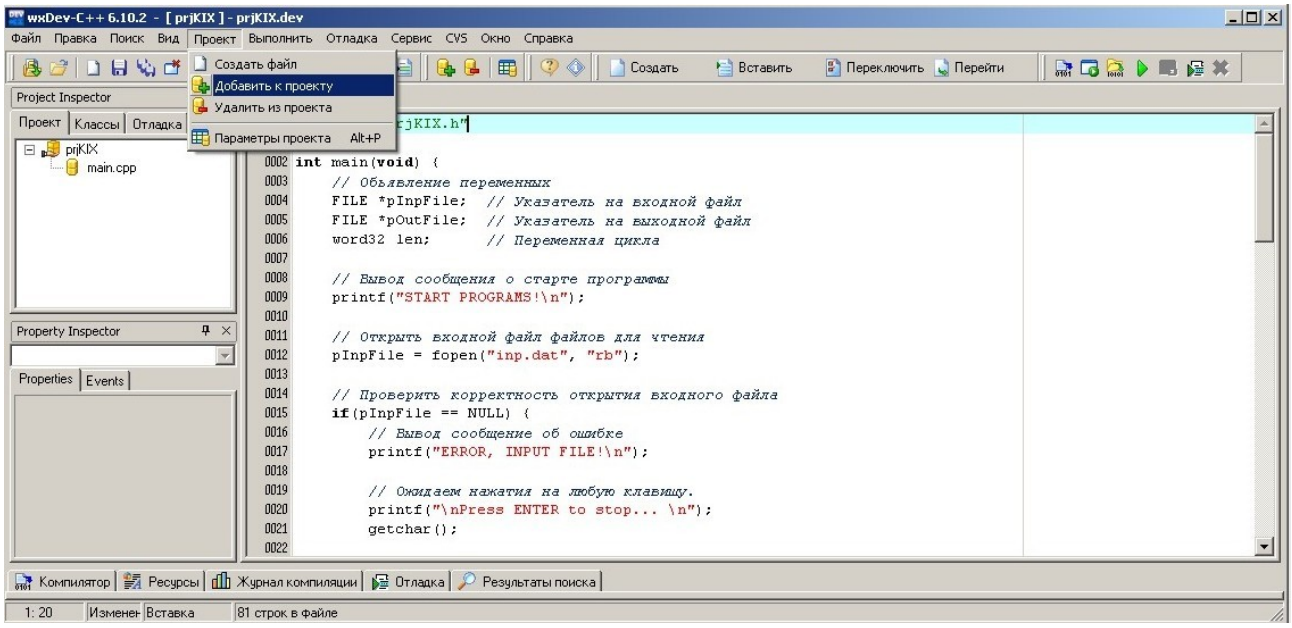

Рис.4. Подключение файлов к проекту.

Можно подключать файлы по одному, а можно указать тип «*ALL files*» и удерживая клавишу «*Ctrl*» мышкой указать необходимые файлы, подключить их за один раз (см.рис.5).

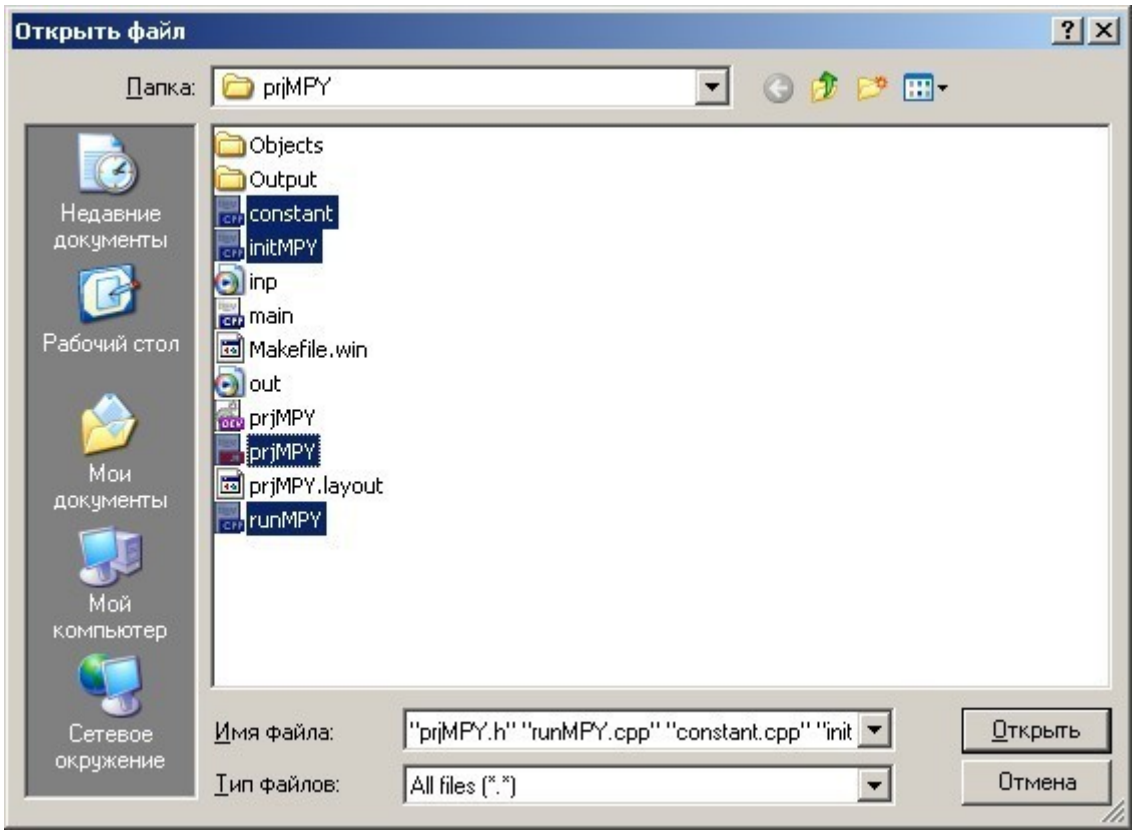

Рис.5. Выбор подключаемых файлов.

Окно программы *wxDev-C++* примет вид показанный на рис.6.

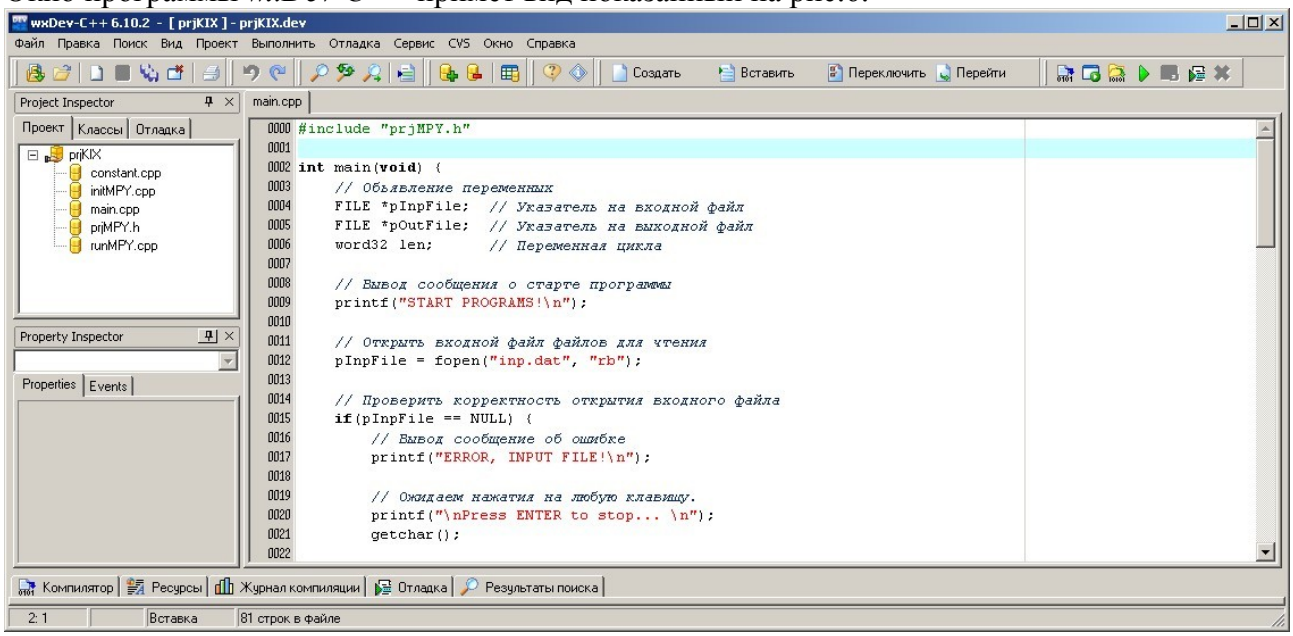

Рис.6. Вид программы wxDev-C++ после подключения файлов.

Следующий шаг — это изменения в программном коде:

- Во всех файлах проекта необходимо заменить строчку «*#include "prjMPY.h"*» на «*#include "prjKIX.h"*».
- Имя типа контекстной структуры «CONTEXMPY» заменить на «CONTEXKIX».
- Имя функции «runMPY» на «runKIX».
- Имя функции «initMPY» на «initKIX».

Откомпилировав и запустив программу на выполнения, нажав кнопку  , можно увидеть результат, показанный на рис.7.

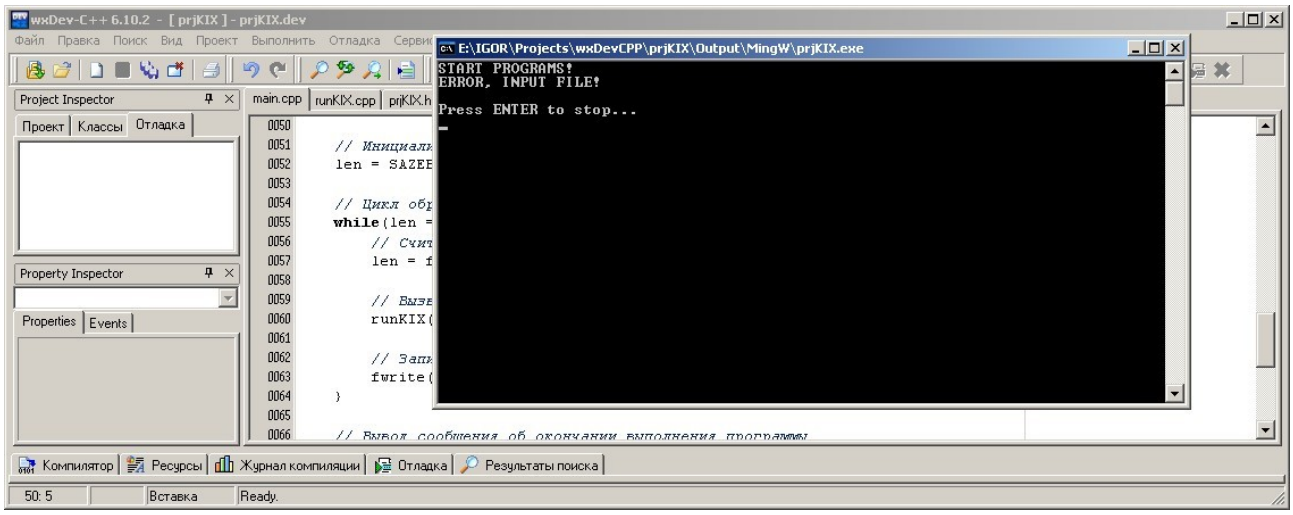

Рис.7. Первый запуск программы.

Необходимо добавить входной файл. В этом качестве будем использовать единичный импульс, сформированный в программе EDSW (см.рис.8).

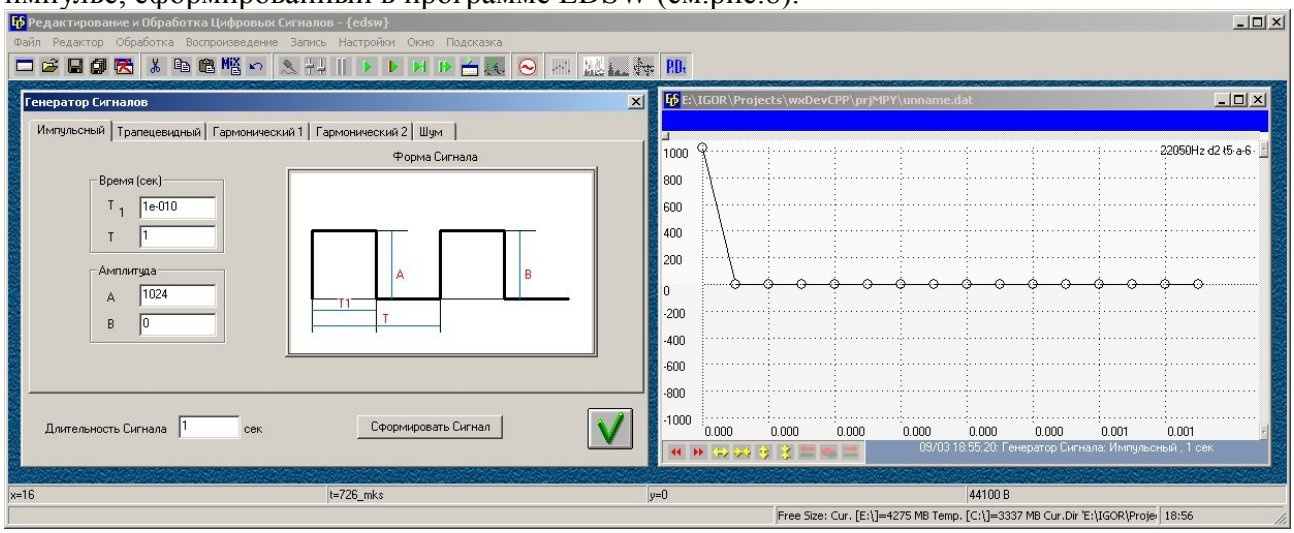

Рис.8. Формирование единичного импульса в программе EDSW.

Запустив программу в режиме отладки можно убедиться в ее корректной работе (см.рис.9).

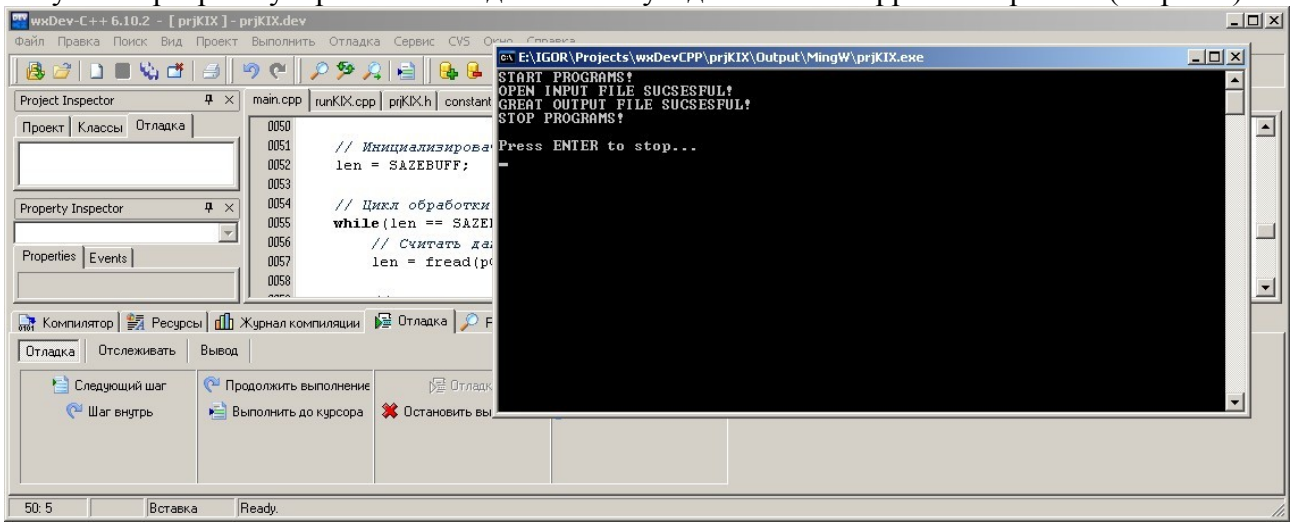

Рис.9. Запуск программы в режиме отладки при наличии входного файла.

При этом, напомним, что запуск программы в режиме отладки осуществляется нажатием кнопки «Отладка» на одноименной вкладке внизу окна программы. Более подробно о режиме отладки написано в «Методических рекомендациях к лабораторной работе №2». Отметим, что программа пока еще работает в соответствии с алгоритмом лаб.раб.№2.

2. Разработка алгоритма

Необходимо разработать алгоритма для вычисления разностного уравнения:

$$y(n)=b_0 x(n)+b_1 x(n-1)+\ldots+b_{N-1} x(n-N+1)\ , \qquad (1)$$

где $y(n)$ — выходной сигнал,

$x(n)$ — входной сигнал,

b_i — коэффициенты фильтра,

N — порядок фильтра (количество коэффициентов).

Для выполнения процедуры, описываемой выражением (1) необходимо выполнить следующие действия:

1. Произвести инициализацию начального состояния алгоритма.
2. Проверить наличие данных во входном файле: если данные есть, то перейти к следующему пункту, в противном случае — перейти к п.6.
3. Считать данные из файла во входной буфер.
4. Обработать считанные данные и записать результат в выходной буфер.
5. Записать выходной буфер в выходной файл.
6. Завершить выполнение программы.

Графическое представление алгоритма функции *main*() представлено на рис.10.

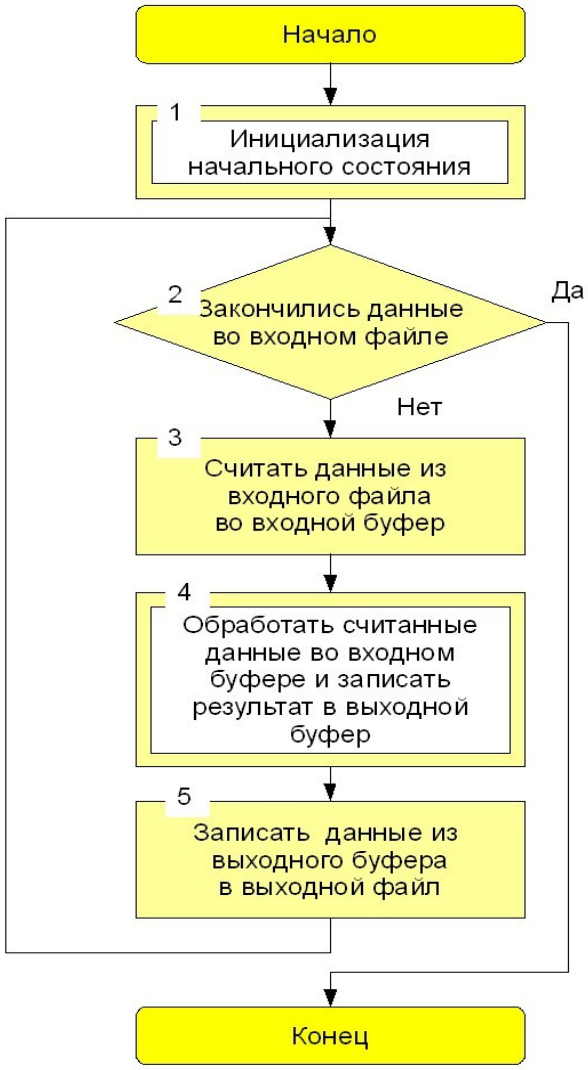

Рис.10. Алгоритм функции main().

Отметим, что п.п.1 и 4 должны быть реализованы в виде подпрограмм (то есть в виде отдельных функций).

Алгоритм программы для реализации инициализации (п.1. на рис.10) показан на рис.11. Он

достаточно прост и не нуждается в особых комментариях.

Рис.11. Алгоритм функции initKIX().

П.4 на рис.10 реализуется двумя вложенными функциями — *runKIX()* и *cnvKIX()*.
Функция *runKIX()* выполняет считывание данных из входного буфера, вызов функции обработки этого буфера *cnvKIX()* и запись результата в выходной буфер. Алгоритм работы этой функции представлен на рис.12.

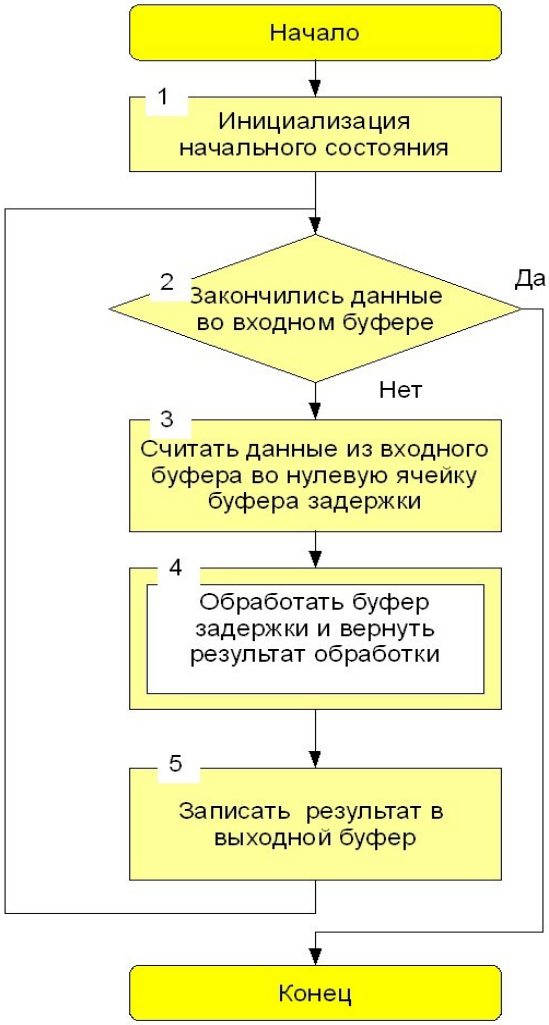

Рис.12. Алгоритм функции runKIX().

Функция *cnvKIX()* выполняет вычисления согласно выражения (1). Что обеспечивается выполнение следующих действий:

- поэлементное умножение буфера коэффициентов и буфера линии задержки,
- накопление результата,
- сдвиг буфера задержки.

Алгоритм функции представлен на рис.13.

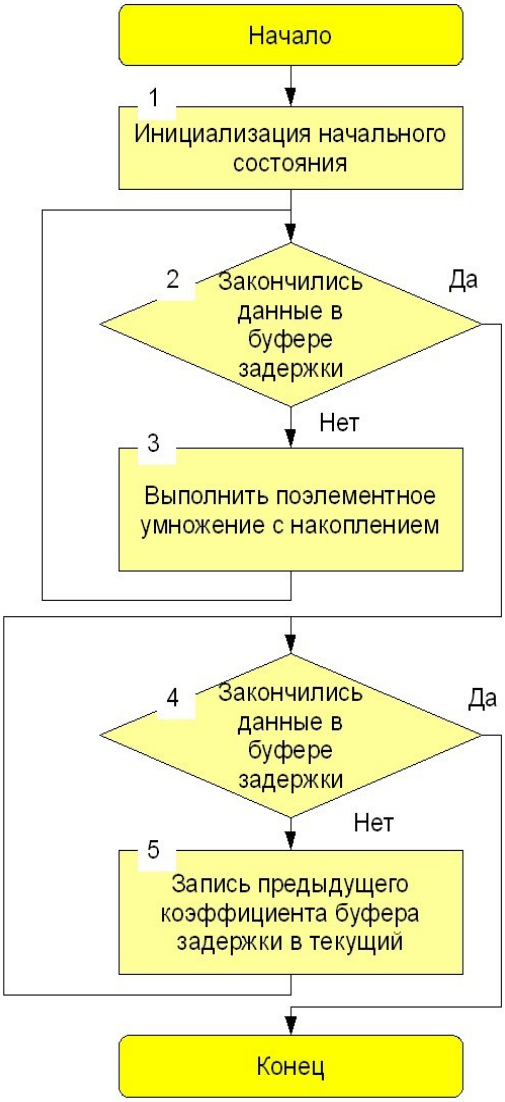

Рис.13. Алгоритм функции cnvKIX().

3. Написание программного кода КИХ фильтра

Прежде чем приступить непосредственно к написанию программного кода КИХ фильтра необходимо создать функцию *cnvKIX()*, файл для ее хранения и подключить ее к проекту. Как это сделать подробно описано в «Методических рекомендациях к лабораторной работе №2».
Основные моменты:

- создать файл для хранения функции,
- сохранить файл с нужным именем,
- подключить его к проекту,
- создать в файле требуемую функцию,
- объявить функцию в заголовочном файле.

Листинг функции (всего лишь шаблон для дальнейшей модификации) *cnvKIX()* будет выглядеть следующим образом:

```
// Функция вычисление свертки
#include "prjKIX.h"

word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
    // Объявление локальных переменных
    word32 rezult;
    ;
    // Выход из функции
    return rezult;
}
```

3.1. Модифицировать заголовочный файл *prjKIX.h*

В заголовочном файле необходимо модифицировать контекстную структуру, добавив поля для указателей массивов коэффициентов и линии задержки, а так же для хранения их размера. Обратите внимание, что вводиться макрос NAMBERCOEFF, соответствующий порядку фильтра (его значение определяется номером варианта и получено в лабораторной работе №1), а так же меняется значение макроса MASHTAB (его величина должна соответствовать значению масштабирующего множителя при переводи числа к виду с фиксированной точкой). Заголовочный файл примет вид:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF    120 // Размер входного и выходного буферов
#define MASHTAB     10  // Маштабирующий коэффициент
#define NAMBERCOEFF 12  // Порядок фильтра

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
    word32 constA;      // Маштабирующий коэффициент
    word16* pCoeff;     // Указатель на буфер коэффициентов фильтра
    word16* pShift;     // Указатель на буфер задержки
    word16  namberCoeff; // Порядок фильтра
} CONTEXKIX;

// Объявление функций проекта
void initKIX(CONTEXKIX* pCntx);
void runKIX(CONTEXKIX* pCntx);
word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA);

// Объявление переменных, массивов и т.д.
extern CONTEXKIX* pCntx;        // Указатель на контекстную структуру
extern CONTEXKIX cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
extern word16 coeffBuff[];      // Массив для входных данных;
extern word16 shiftBuff[];      // Массив для выходных данных;
```

3.2. Создать массивы для линии задержки и хранения коэффициентов

Объявленные буферы для линии задержки и коэффициентов необходимо создать в файле *constant.cpp*. Листинг данного файла будет выглядеть примерно так:

```
#include "prjKIX.h"

// Создание указателя на контекстную структуру
CONTEXKIX* pCntx;

// Создание контекстной структуры
CONTEXKIX cntx;

// Массив для входных данных
word16 inpBuff[SAZEBUFF];

// Массив для выходных данных
word16 outBuff[SAZEBUFF];

// Массив для входных данных;
word16 coeffBuff[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

// Массив для выходных данных;
word16 shiftBuff[NAMBERCOEFF];
```

Обратите внимание на инициализацию буфера коэффициентов. Данный массив не только создается но и инициализируется. В него записываются значения коэффициентов фильтра. Необходимо записать значения, полученные в лабораторной работе №1 и приведенные к целочисленному виду.

3.3. Модифицировать функцию *initKIX()*

В функции *initKIX()* необходимо провести инициализацию вновь созданных полей контекстной структуры, а так же обнуление массивов входных коэффициентов и линии задержки. Листинг функции будет выглядеть следующим образом:

```
// Функция инициализации начального состояния
// разрабатываемого проекта

#include "prjKIX.h"

void initKIX(CONTEXKIX* pCntx) {
    // Локальные переменные
    word32  count;      // Переменная цикла

    // Инициализация полей контекстной структуры
    pCntx->pInpBuff = inpBuff;     // Указатель на входной буфер
    pCntx->pOutBuff = outBuff;     // Указатель на выходной буфер
    pCntx->lenBuff  = SAZEBUFF;    // Размером входного и выходного буферов
    pCntx->constA   = MASHTAB;     // Маштабирующий коэффициент
    pCntx->pCoeff   = coeffBuff;   // Указатель на буфер коэффициентов фильтра
    pCntx->pShift   = shiftBuff;   // Указатель на буфер задержки
    pCntx->namberCoeff = NAMBERCOEFF; // Порядок фильтра

    // Обнуление массива входных коэффициентов
    for(count = 0; count < SAZEBUFF; count++) pCntx->pInpBuff[count] = 0;

    // Обнуление линии задержки
    for(count = 0; count < NAMBERCOEFF; count++) pCntx->pShift[count] = 0;
}
```

3.4. Модифицировать функцию *runKIX()*

Функция *runKIX()* должна выполнить считывание одного отсчета из входного буфера, запись его в нулевую ячейку буфера задержки, запуск функции обработки буфера задержки (*cnvKIX()*) и запись результата в выходной буфер. Строить функцию будем на базе созданной в предыдущей лабораторной работе функции *runMPY()*. После модификации функция будет выглядеть следующим образом:

```
// Функция обработки входного буфера и
// запись результата в выходной буфер

#include "prjKIX.h"

void runKIX(CONTEXKIX* pCntx) {
    // Объявление локальных переменных
    word16* pInpBuff;   // Указатель на входной буфер
    word16* pOutBuff;   // Указатель на выходной буфер
    word16  lenBuff;    // Размером входного и выходного буферов
    word32  constA;     // Маштабирующий коэффициент
    word32  count;      // Переменная цикла
    word16  coeff;      // Переменная для хранение промежуточных результатов
    word16* pCoeff;     // Указатель на буфер коэффициентов фильтра
    word16* pShift;     // Указатель на буфер задержки
    word16  namberCoeff; // Порядок фильтра

    // Инициализация локальных переменных
    pInpBuff    = pCntx->pInpBuff;
    pOutBuff    = pCntx->pOutBuff;
    lenBuff     = pCntx->lenBuff;
    constA      = pCntx->constA;
    pCoeff      = pCntx->pCoeff;
    pShift      = pCntx->pShift;
    namberCoeff = pCntx->namberCoeff;

    // Цикл обработки и записи
    for(count = 0; count < lenBuff; count++){

        // Чтение текущего отсчета
        coeff = *pInpBuff++;

        // Запись текущего отсчета в нулевую ячейку линии задержки
        *pShift = coeff;

        // Вычисление выходного коэффициента фильтра
        coeff = cnvKIX(pCoeff, pShift, namberCoeff, constA);

        // Запсь текущего отсчета
        *pOutBuff++ = coeff;
    }
}
```

Создаются локальные переменные, соответствующие полям контекстной структуры. Для удобства они могут носить те же имена, что и поля структуры, но это не обязательно. Происходит инициализация этих переменных соответствующими значениями полей контекстной структуры через указатель на эту структуру.

3.5. Создать функцию *cnvKIX()*

Шаблон функции был уже создан в начале раздела 3. Полный листинг функции выглядит следующим образом:

```
// Функция вычисление свертки
#include "prjKIX.h"

word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
    // Объявление локальных переменных
    word16 coeffX;
    word16 coeffB;
    word32 count;
    word32 coeff;
    word32 rezult;

    // Инициализация локальных переменных
    rezult = 0;

    // Цикл обработки линии задержки
    for(count = 0; count < namberCoeff; count++){

        // Чтение текущего отсчета
        coeffX = *pShift++;

        // Чтение соответствующего коэффициента фильтра
        coeffB = *pCoeff++;

        // Умножение отсчета на коэффициент
        coeff = coeffX * coeffB;

        // Накопление результата
        rezult += coeff;
    }

    // Корректировка значения указателя
    pShift--;

    // Сдвиг линии задержки
    for(count = namberCoeff - 1; count > 0; count--){

        // Чтение предыдущего отсчета
        coeffX = pShift[-1];

        // Запись предыдущего отсчета в текущую ячейку
        *pShift-- = coeffX;
    }

    // Нормирование результата суммирования
    rezult >>= constA;

    // Выход из функции
    return rezult;
}
```

Первый цикл производит операцию умножения с накоплением: считываются коэффициент фильтра и отсчет входного сигнала из буфера коэффициентов и линии задержки, соответственно, при этом происходит инкрементация (увеличение на одну позицию) указателей на буферы коэффициентов и отсчетов. Далее считанные коэффициент и отсчет перемножаются, а полученное значение добавляется к результату - накапливается (таким образом и происходит умножение с накопление). Затем цикл повторяется.

Обратите внимание на второй цикл, где происходит сдвиг линии задержки. Во-первых, до цикла происходит корректировка указателя на буфер отсчетов (линию задержки). Это необходимо потому, что происходит увеличение этого указателя на единицу каждый раз после считывания очередного отсчета и после считывания последнего увеличение приведет к тому, что он будет указывать на элемент за пределами буфера отсчетов. Во-вторых, сдвиг

происходит в направлении из конца в начало. Если сделать наоборот, то произойдет не сдвиг отсчетов, а размножение значения нулевой ячейки на весь массив.
Если все выполнено правильно, то запуск программы в режиме отладки выполнится без ошибки и сформируется выходной файл *out.dat*. Файл можно просмотреть в программе EDSW. Для рассматриваемого примера, результат будет иметь вид показанный на рис.14.

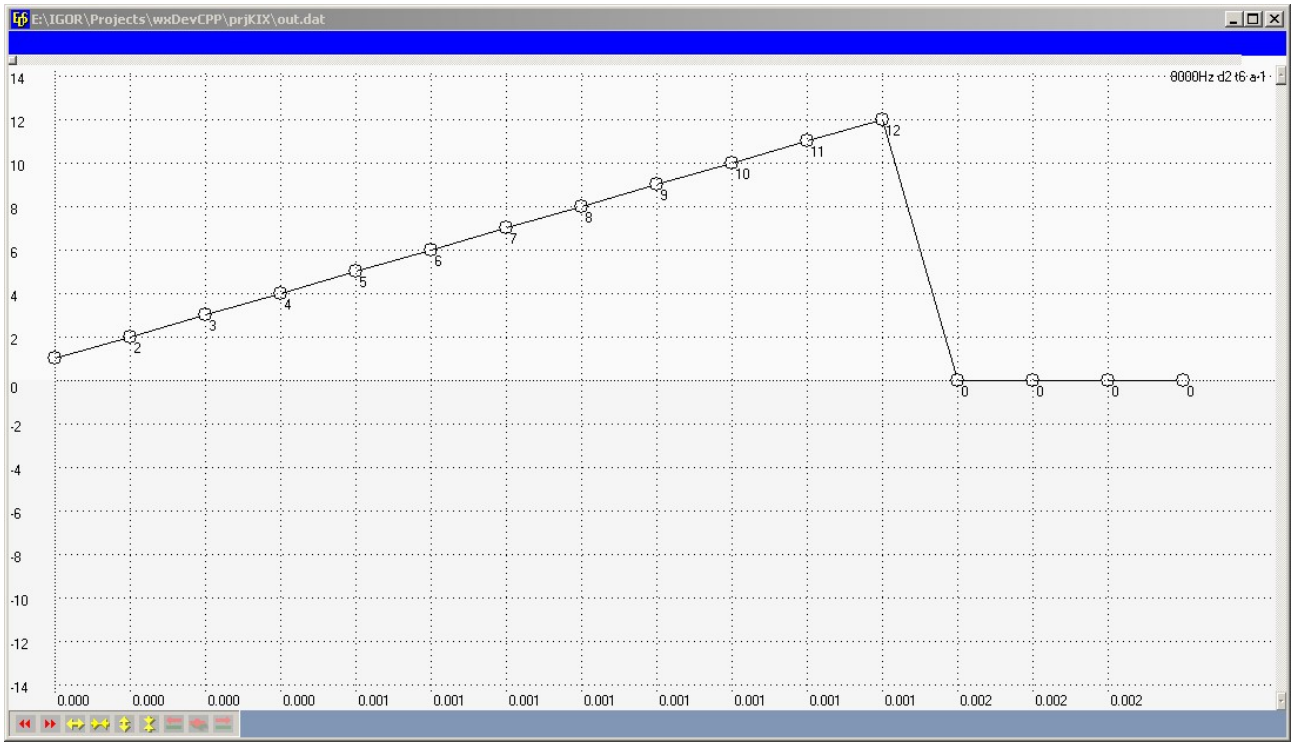

Рис.14. Результат работы программы КИХ фильтра.

Отсчеты импульсной характеристики тождественно равны коэффициентам фильтра.
Кроме этого, в программе EDSW строиться и реальная АЧХ. Для рассматриваемого примера вид АЧХ представлен на рис.15

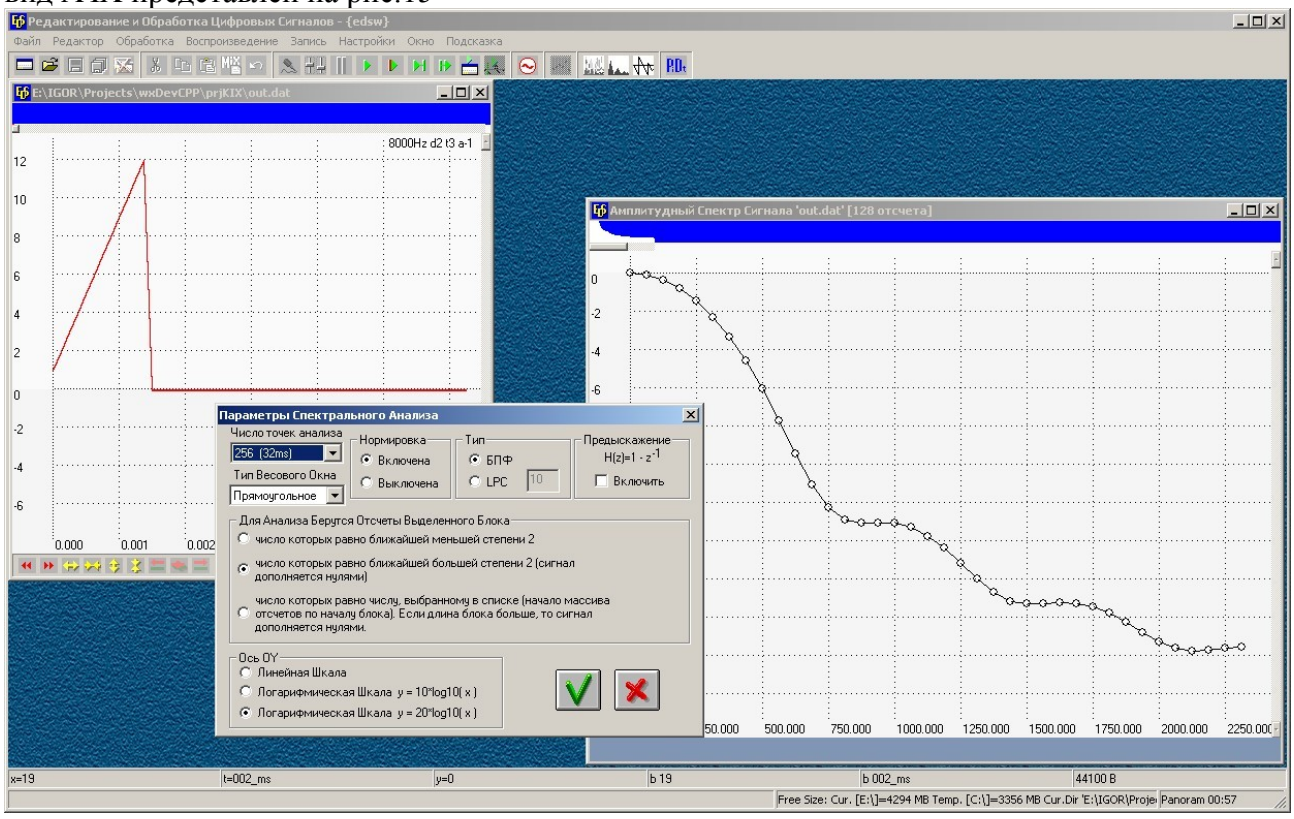

Рис.15. Расчет АЧХ.

4. Приложения

Приложение 1.

Листинг файла *prjKIX.h*

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF    120 // Размер входного и выходного буферов
#define MASHTAB     10  // Маштабирующий коэффициент
#define NAMBERCOEFF 12  // Порядок фильтра

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
    word32 constA;      // Маштабирующий коэффициент
    word16* pCoeff;     // Указатель на буфер коэффициентов фильтра
    word16* pShift;     // Указатель на буфер задержки
    word16  namberCoeff; // Порядок фильтра
} CONTEXKIX;

// Объявление функций проекта
void initKIX(CONTEXKIX* pCntx);
void runKIX(CONTEXKIX* pCntx);
word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA);

// Объявление переменных, массивов и т.д.
extern CONTEXKIX* pCntx;        // Указатель на контекстную структуру
extern CONTEXKIX cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
extern word16 coeffBuff[];      // Массив для входных данных;
extern word16 shiftBuff[];      // Массив для выходных данных;
```

Приложение 2.

Листинг файла *constantrj.cpp*

```
#include "prjKIX.h"

// Создание указателя на контекстную структуру
CONTEXKIX* pCntx;

// Создание контекстной структуры
CONTEXKIX cntx;

// Массив для входных данных
word16 inpBuff[SAZEBUFF];

// Массив для выходных данных
word16 outBuff[SAZEBUFF];

// Массив для входных данных;
word16 coeffBuff[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

// Массив для выходных данных;
word16 shiftBuff[NAMBERCOEFF];
```

Приложение 3.

Листинг функции *initKIX()*

```
// Функция инициализации начального состояния
// разрабатываемого проекта

#include "prjKIX.h"

void initKIX(CONTEXKIX* pCntx) {
    // Локальные переменные
    word32  count;      // Переменная цикла

    // Инициализация полей контекстной структуры
    pCntx->pInpBuff = inpBuff;      // Указатель на входной буфер
    pCntx->pOutBuff = outBuff;      // Указатель на выходной буфер
    pCntx->lenBuff  = SAZEBUFF;     // Размером входного и выходного буферов
    pCntx->constA   = MASHTAB;      // Маштабирующий коэффициент
    pCntx->pCoeff   = coeffBuff;    // Указатель на буфер коэффициентов фильтра
    pCntx->pShift   = shiftBuff;    // Указатель на буфер задержки
    pCntx->namberCoeff = NAMBERCOEFF; // Порядок фильтра

    // Обнуление массива входных коэффициентов
    for(count = 0; count < SAZEBUFF; count++) pCntx->pInpBuff[count] = 0;

    // Обнуление линии задержки
    for(count = 0; count < NAMBERCOEFF; count++) pCntx->pShift[count] = 0;
}
```

Приложение 4.

Листинг функции *runKIX()*

```
// Функция обработки входного буфера и
// запись результата в выходной буфер

#include "prjKIX.h"

void runKIX(CONTEXKIX* pCntx) {
    // Объявление локальных переменных
    word16* pInpBuff;   // Указатель на входной буфер
    word16* pOutBuff;   // Указатель на выходной буфер
    word16  lenBuff;    // Размером входного и выходного буферов
    word32  constA;     // Маштабирующий коэффициент
    word32  count;      // Переменная цикла
    word16  coeff;      // Переменная для хранение промежуточных результатов
    word16* pCoeff;     // Указатель на буфер коэффициентов фильтра
    word16* pShift;     // Указатель на буфер задержки
    word16  namberCoeff; // Порядок фильтра

    // Инициализация локальных переменных
    pInpBuff    = pCntx->pInpBuff;
    pOutBuff    = pCntx->pOutBuff;
    lenBuff     = pCntx->lenBuff;
    constA      = pCntx->constA;
    pCoeff      = pCntx->pCoeff;
    pShift      = pCntx->pShift;
    namberCoeff = pCntx->namberCoeff;

    // Цикл обработки и записи
    for(count = 0; count < lenBuff; count++){

        // Чтение текущего отсчета
        coeff = *pInpBuff++;

        // Запись текущего отсчета в нулевую ячейку линии задержки
        *pShift = coeff;

        // Вычисление выходного коэффициента фильтра
        coeff = cnvKIX(pCoeff, pShift, namberCoeff, constA);

        // Запсь текущего отсчета
        *pOutBuff++ = coeff;
    }
}
```

Листинг функции *main()*

```
#include "prjMPY.h"

int main(void) {
    // Объявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл
    word32 len;      // Переменная цикла

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        // Вывод сообщение об ошибке
        printf("ERROR, INPUT FILE!\n");

        // Ожидаем нажатия на любую клавишу.
        printf("\nPress ENTER to stop... \n");
        getchar();

        // Выход из программы
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        // Вывод сообщение об ошибке
        printf("ERROR, OUT FILE\n");

        // Ожидаем нажатия на любую клавишу.
        printf("\nPress ENTER to stop... \n");
        getchar();

        // Выход из программы
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Инициализировать переменную цикла
    len = SAZEBUFF;

    // Цикл обработки входного буфера
    while(len == SAZEBUFF) {
        // Считать данные из файла
        len = fread(pCntx->pInpBuff, sizeof(word16), SAZEBUFF, pInpFile);

        // Вызвать функцию обработки входного буфера
        runMPY(pCntx);

        // Записать данные в файл
        fwrite(pCntx->pOutBuff, sizeof(word16), len, pOutFile);
    }

    // Вывод сообщения об окончании выполнения программы
    printf("STOP PROGRAMS!\n");

    // Ожидаем нажатия на любую клавишу.
    printf("\nPress ENTER to stop... \n");
    getchar();

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```

Листинг функции *cnvKIX()*

```
// Функция вычисление свертки
#include "prjKIX.h"

word16 cnvKIX(word16* pCoeff, word16* pShift, word16 namberCoeff, word32 constA){
    // Объявление локальных переменных
    word16 coeffX;
    word16 coeffB;
    word32 count;
    word32 coeff;
    word32 rezult;

    // Инициализация локальных переменных
    rezult = 0;

    // Цикл обработки линии задержки
    for(count = 0; count < namberCoeff; count++){

        // Чтение текущего отсчета
        coeffX = *pShift++;

        // Чтение соответствующего коэффициента фильтра
        coeffB = *pCoeff++;

        // Умножение отсчета на коэффициент
        coeff = coeffX * coeffB;

        // Накопление результата
        rezult += coeff;
    }

    // Корректировка значения указателя
    pShift--;

    // Сдвиг линии задержки
    for(count = namberCoeff - 1; count > 0; count--){

        // Чтение предыдущего отсчета
        coeffX = pShift[-1];

        // Запись предыдущего отсчета в текущую ячейку
        *pShift-- = coeffX;
    }

    // Нормирование результата суммирования
    rezult >>= constA;

    // Выход из функции
    return rezult;
}
```