Лабораторная работа №4. Классы в C++, использование dll библиотек.

1.1 Задание для работы

Написать класс, который будет внутри генерировать сигнал, рассчитывать спектр (ДПФ) с использованием библиотеки fftw, записывать в комплексный файл .adc.

У класса должно быть:

конструктор с параметрами по умолчанию (размер БПФ, частота дискретизации);

сеттеры и геттеры для параметров из пункта 1;

приватный метод генерации сигнала;

свойства (члены) класса std::vector для сигнала и результата fftw;

публичный метод расчета спектра с использованием fftw;

публичный метод для записи результата fftw в файл.

Проверить работоспособность класса, записать файл и построить в Спектр-2.

1.2 Теоретические сведения

1.2.1 Классы в языке C++

Классы в программировании состоят из свойств и методов. Свойства — это любые данные, которыми можно характеризовать объект класса.

```
//Класс генерации синусоидального сигнала частоты 100 Гц
class SinGen_100Hz {
private:

    float freq = 100.0; // Частота 100 Гц
    float freq_sampling = 48000.0; // Частота дискретизации 48 кГц
    unsigned int size_array; // Размер массива
    std::vector<float> array_sin; // Объект std::vector для отсчетов синусоидального сигнала

public:
    // Конструктор класса SinGen_100Hz
    SinGen_100Hz(unsigned int _size_array)
    {
        size_array = _size_array;
        array_sin.resize(size_array);
    }

    // Функция, для генерации отсчетов sin - Метод класса
    void calculate_sin_array()
    {
        for (int i = 0; i < 5; ++i)
        {
            array_sin[i] = sin(float(2 * M_PI * i * freq / freq_sampling));
        }
```

```
    }

    // Метод класса, получения расчитанных значений
    std::vector<float> get_sin_array()
    {
        return array_sin;
    }

    // Геттер для получения значения свойства size_array
    unsigned int get_size_array()
    {
        return size_array;
    }

    // Сеттер для установки значения свойства size_array
    void set_size_array(unsigned int _n)
    {
        size_array = _n;
        array_sin.resize(size_array);
    }
};
```

В нашем случае, объектом класса является генератор синусоидального сигнала с частотой 100 Гц, а его свойствами — частота, частота дискретизации, количество отсчетов для генерации.

Методы — это функции, которые могут выполнять какие-либо действия над данными (свойствами) класса. Метод `void calculate_sin_array()` рассчитывает отсчеты сигнала по параметрам заданным в свойствах класса.

Методы класса – это его функции.

Свойства класса – это его переменные.

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого. Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа *public*. Все функции и переменные, которые находятся после модификатора *public*, становятся доступными из всех частей программы.

Закрытые данные класса размещаются после модификатора доступа *private*. Если отсутствует модификатор *public*, то все функции и переменные, по умолчанию являются закрытыми (как в первом примере).

Конструктор класса — это специальная функция, которая автоматически вызывается сразу после создания объекта этого класса. Он не имеет типа возвращаемого значения и должен называться также, как класс, в котором он находится.

Деструктор класса вызывается при уничтожении объекта. Имя деструктора аналогично имени конструктора, только в начале ставится знак тильды ~. Деструктор не имеет входных параметров.

Пример использования класса `SinGen_100Hz:`

```
int main()
{
    std::vector<float> data;
    std::vector<float> data1;
    // Создание объекта класса SinGen_100Hz
    SinGen_100Hz generator(256);
    // Вызываем метод генерации
    generator.calculate_sin_array();
    // Вызываем метод получения данных и получаем объект std::vector<float>
    data = generator.get_sin_array();

    // Меняем размер массива для генерации вызывая Сеттер параметра
    generator.set_size_array(512);
    // Вызываем метод генерации
    generator.calculate_sin_array();
    // Вызываем метод получения данных и получаем объект std::vector<float>
    data1 = generator.get_sin_array();
}
```

При создании объекта, лучше не копировать память для него, а выделять ее в в куче с помощью указателя. И освобождать ее после того, как мы закончили работу с объектом. Реализуем это в нашей программе, немного изменив содержимое файла **main.cpp**.

```
int main()
{
    std::vector<float> data;
    std::vector<float> data1;
    // Выделение памяти для объекта SinGen_100Hz
    SinGen_100Hz *generator = new SinGen_100Hz(256);
    // Вызываем метод генерации
    generator->calculate_sin_array();
    // Вызываем метод получения данных и получаем объект std::vector<float>
    data = generator->get_sin_array();

    // Меняем размер массива для генерации вызывая Сеттер параметра
    generator->set_size_array(512);
    // Вызываем метод генерации
    generator->calculate_sin_array();
    // Вызываем метод получения данных и получаем объект std::vector<float>
    data1 = generator->get_sin_array();

    // Удаление объекта SinGen_100Hz из памяти
    delete generator;

}
```

При создании статического объекта, для доступа к его методам и свойствам, используют операция прямого обращения — «**.**» (символ точки). Если же память для объекта выделяется посредством указателя, то для доступа к его методам и свойствам используется оператор косвенного обращения — «**->**».

1.2.2 Динамическая библиотека dll в C++

В Windows библиотека динамической компоновки (DLL) является исполняемым файлом, который выступает в качестве общей библиотеки функций и ресурсов. Динамическая компоновка — это возможность операционной системы. Она позволяет исполняемому файлу вызывать функции или использовать ресурсы, хранящиеся в отдельном файле. Эти функции и ресурсы можно компилировать и развертывать отдельно от использующих их исполняемых файлов.

Библиотека DLL не является отдельным исполняемым файлом. Библиотеки DLL выполняются в контексте приложений, которые их вызывают. Операционная система загружает библиотеку DLL в область памяти приложения. Это делается либо при загрузке приложения (*неявная компоновка*), либо по запросу во время выполнения (*явная компоновка*). Библиотеки DLL также упрощают совместное использование функций и ресурсов различными исполняемыми файлами. Несколько приложений могут осуществлять одновременный доступ к содержимому одной копии библиотеки DLL в памяти.

1.2.3 Библиотека численного преобразования Фурье FFTW

Библиотека FFTW - свободно распространяемая библиотека для выполнения быстрого дискретного преобразования Фурье.

Для выполнения преобразования Фурье при помощи библиотеки необходимо создать план – заполнить настройки БПФ.

```
// создание одномерного плана БПФ
// n - размерность выборки
// in  - указатель на исходную выборку комплексных чисел
// out - указатель на результирующую выборку комплексных чисел
//       (в качестве результирующей выборки можно задать ту же выборку, что и исходная)
// direction - направление (FFTW_FORWARD - прямое, FFTW_BACKWARD - обратное)
// flags - флаги, можно задать флаг FFTW_ESTIMATE (используются внутренние настройки по
умолчанию)
fftwf_plan fftwf_plan_dft_1d(int n,
    fftwf_complex* sample_in,
    fftwf_complex* sample_out,
    fftwf_direction direction,
           int flags);
```

Удаление плана после выполнения БПФ:

```
// удаление плана БПФ
// plan - план БПФ
void fftwf_destroy_plan(fftwf_plan plan);
```

Простое (не сдвиговое) преобразование Фурье можно выполнить при помощи

функции

```
// выполнение одномерного простого БПФ в соответствии с планом
// plan - план БПФ
void fftwf_execute(fftwf_plan plan);
```

Пример выполнения одномерного преобразования Фурье:

```
#include <fstream>
#include <complex>
#include <vector>

#include <fftw3.h>

using namespace std;
////////////////////////////////////////////////////////////////////////////
void main()
{
    // создаем одномерную выборку, все значения которой равны 1
    vector<complex<float> > data(64, 1.);

    // создаем план для библиотеки fftw
    fftwf_plan plan = fftwf_plan_dft_1d(data.size(), (fftwf_complex*)&data[0],
(fftwf_complex*)&data[0], FFTW_FORWARD, FFTW_ESTIMATE);

    // преобразование Фурье
    fftwf_execute(plan);
    fftwf_destroy_plan(plan);

    // выводим в файл результат преобразования Фурье (должна получиться Дельта-функция)
    ofstream out("spectr.txt");
    for (size_t i = 0; i < data.size(); ++i)
    {
        out << data[i] << endl;
    }
}
```

1.2.4 Комплексные числа std::complex

Для представления комплексных чисел можно использовать класс std::complex. Класс комплексных чисел complex – еще один класс из стандартной библиотеки. Как обычно, для его использования нужно включить заголовочный файл:

```
#include <complex>
```

Комплексное число состоит из двух частей – вещественной и мнимой. Мнимая часть представляет собой квадратный корень из отрицательного числа. Комплексное число принято записывать в виде

2 + 3i

где 2 – действительная часть, а 3i – мнимая. Вот примеры определений объектов типа complex:

```
// чисто мнимое число: 0 + 7-i
comp1ex< double > purei(0, 7);
// мнимая часть равна 0: 3 + Oi
comp1ex< float > rea1_num(3);
// и вещественная, и мнимая часть равны 0: 0 + 0-i
comp1ex< long double > zero;
// инициализация одного комплексного числа другим
comp1ex< double > purei2(purei);
```

Поскольку complex, как и vector, является шаблоном, мы можем конкретизировать его типами float, double и long double, как в приведенных примерах. Можно также определить массив элементов типа complex:

```
complex< double > conjugate[2] = {
  complex< double >(2, 3),
  complex< double >(2, -3)
};
```

Вот как определяются указатель и ссылка на комплексное число:

```
complex< double >* ptr = &conjugate[0];
complex< double >& ref = *ptr;
```

При создании комплексного adc файла необходимо установить флаги `head.flags = ADCF_TYPE_COMPLEX | ADCF_DATA_FLOAT;`.

Так же необходимо учитывать, что количество отсчетов в заголовке указывается, как количество комплексных отсчетов. Комплексный отсчет это 2 float.

1.3 Содержание отчета

Отчет к лабораторной работе должен содержать:

1. Текст задания, вариант.
2. Листинг программы.
3. Скриншот отображения комплексного файла .adc

1.4 Варианты задания

Для определения варианта используется номер студенческого билета (последние 2 цифры) N и номер в списке журнала группы k.

Сигнал состоит из суммы 3х гармонических колебаний:

Каждое колебание описывается $A\sin(2\pi n\frac{f}{Fs})$

Параметры для каждого варианта выбираются исходя из

$f_1 = 100 \cdot N\ Гц;$

$f_2 = 512 \cdot k\ Гц;$

$f_3 = 50 \cdot (N + k)\ Гц;$

$A_1 = 0.01 \cdot N;$

$A_2 = k / 10;$

$A_3 = 0.05 \cdot (N + k);$

$Fs = 48\ кГц;$

Размер БПФ (и количество отсчетов сигнала) выбирается по следующей формуле используя последнюю цифру студенческого билета X

$SizeFFT = 2^{X+7}$

1.5 Приложение А Настройки Visual Studio для работы с библиотекой FFTW

Для того чтобы подключить библиотеку FFTW к Visual Studio, необходимо:

1. разархивировать файлы из fftw.zip в любую папку, например D:\fftw

Для того, чтобы использовать библиотеку FFTW, необходимо проделать следующие настройки (пункт меню Project->[имя проекта]Properties):

1. Выбрать конфигурацию например "Debug"
2. В разделе “C/C++” -> “Общие” -> “Дополнительные каталоги включаемых файлов” указать путь к папке D:\fftw.
3. В разделе “Компоновщик” -> “Общие” -> “Дополнительные каталоги библиотек” указать путь к папке D:\fftw.
4. В разделе “Компоновщик” -> “Ввод” -> “Дополнительные зависимости” указать **libfftw3f-3.lib**;

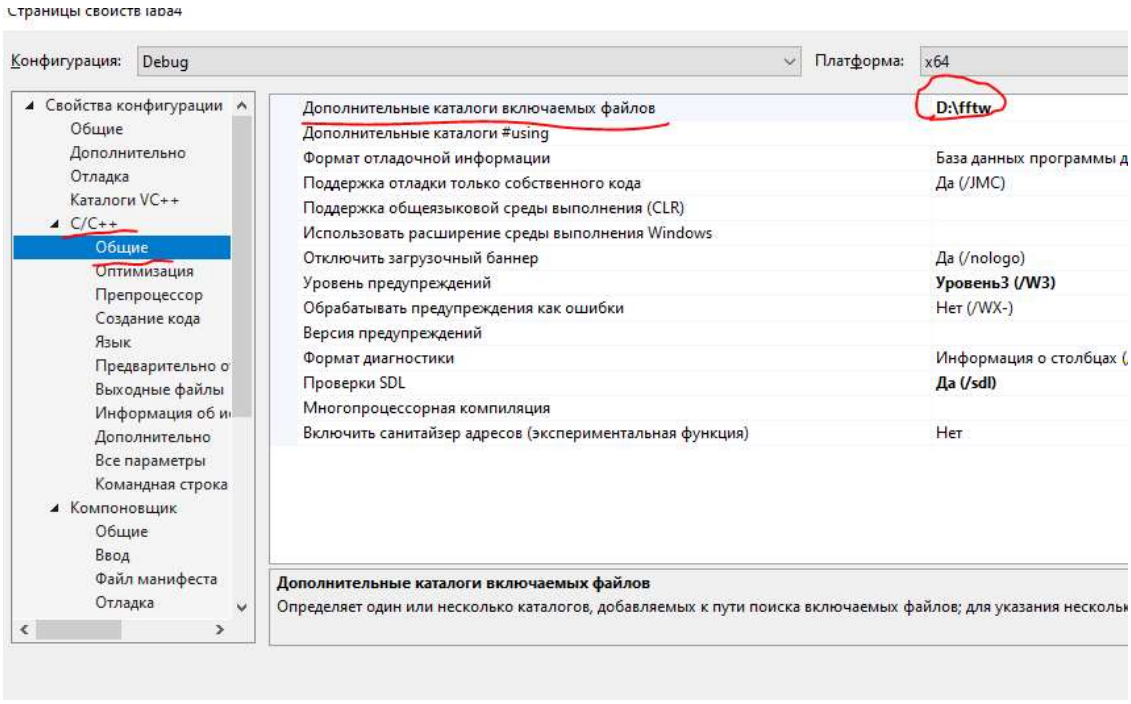

Рисунок 1 Путь к заголовочному файлу fftw и lib

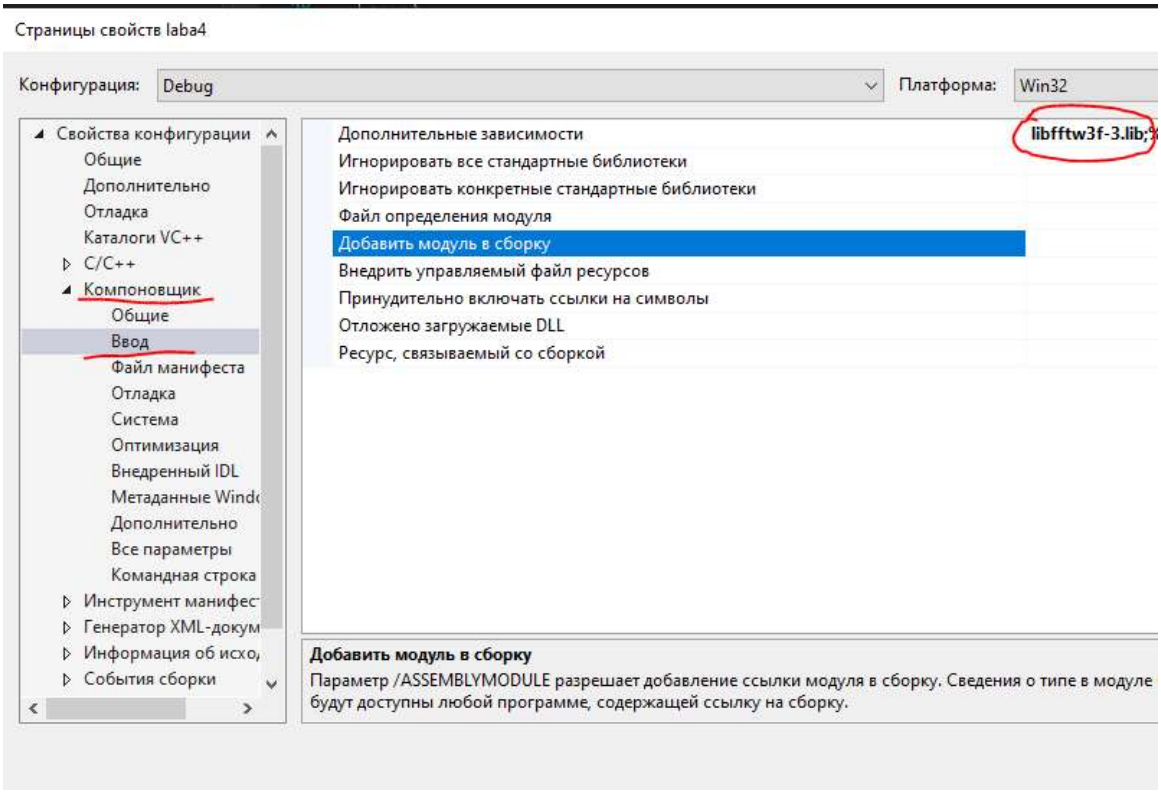

Рисунок 2 Имя подключаемой библиотеки

1.6 Приложение В Просмотр результата в среде Спектр-2

Убедиться, что в папке программы Спектр-2 в подпапке devdll находится библиотека «statistics.dll».

Запустить программу Спектр-2 используя файл Spectr2.exe.

После загрузки программы, по аналогии с лабораторной работой №3, создать систему. В окне «Параметры системы» указать частоту дискретизации 48000 и размер пачки БПФ в зависимости от варианта.

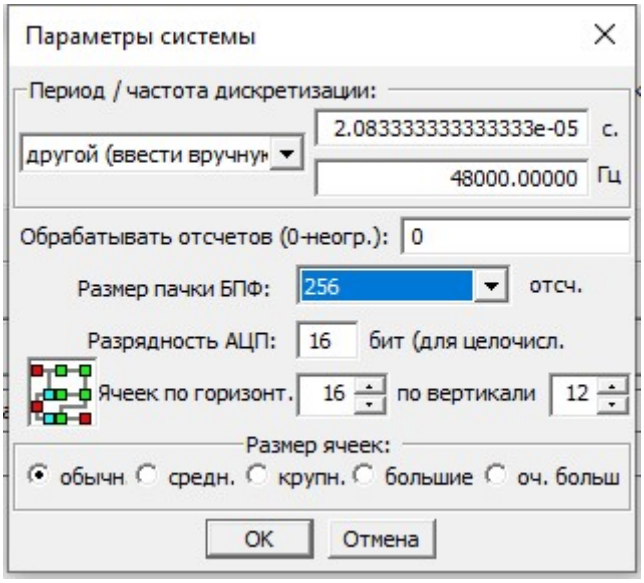

Рисунок 3 Параметры системы

Затем в дереве устройств найти “Входной файл АЦП” и установить на поле.

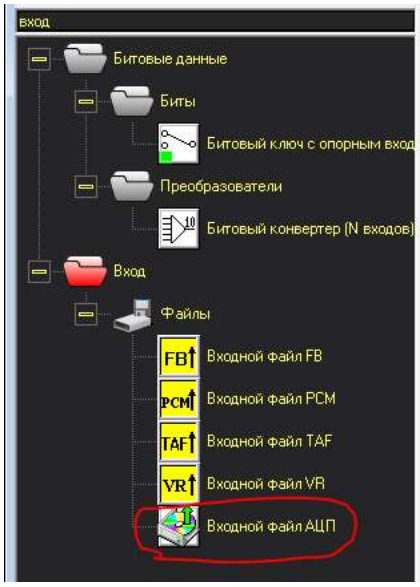

Рисунок 4 Кубик “Входной файл АЦП”

В настройках данного устройства выбрать путь к файлу, полученному во время выполнения лабораторной работы. Проверить что тип файла «отсчеты комплексные с плавающей точкой» и объем файла в отсчетах соответствует вашему варианту. После этого нажать кнопки «Установить в системе» и нажать кнопку «Открыть».

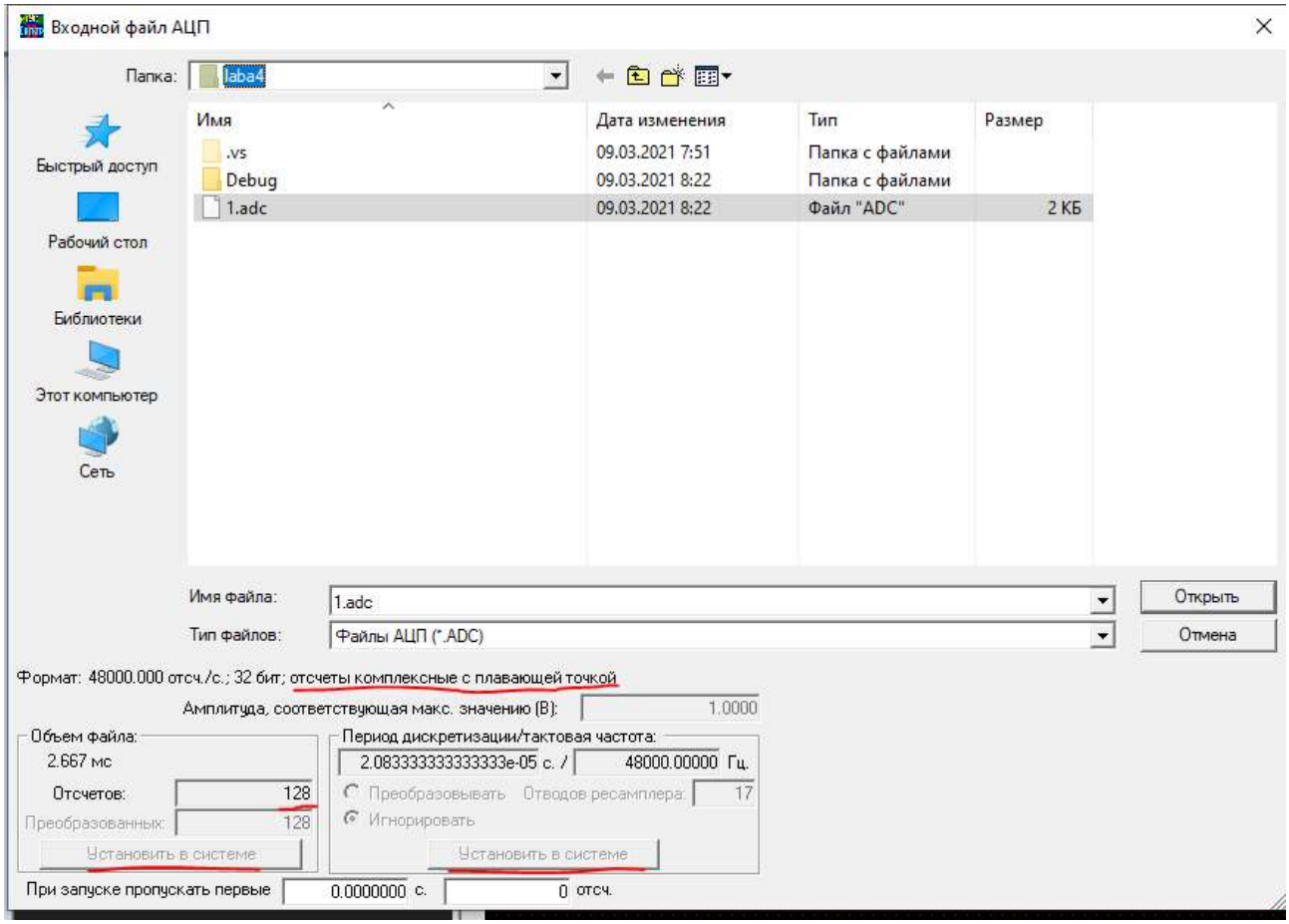

Рисунок 5 Настройки “Входной файл АЦП”

Найти в дереве устройств «Qt График БПФ» и установить на поле. Соединить 2 устройства.

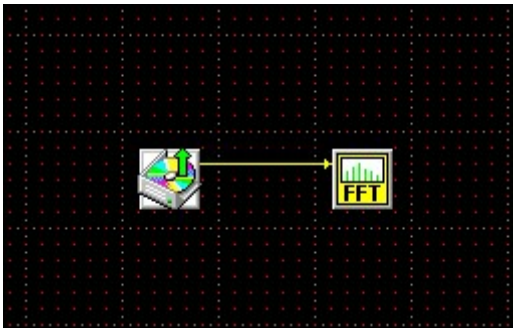

Рисунок 6 Схема для проверки результата

Двойным нажатием на устройство «Qt График БПФ» расположенное на схеме открыть окно. После этого на главном тулбаре нажать кнопку «Обработка очередной пачки» или F7 на клавиатуре. В окне устройства «Qt График БПФ» отобразятся данные из файла.

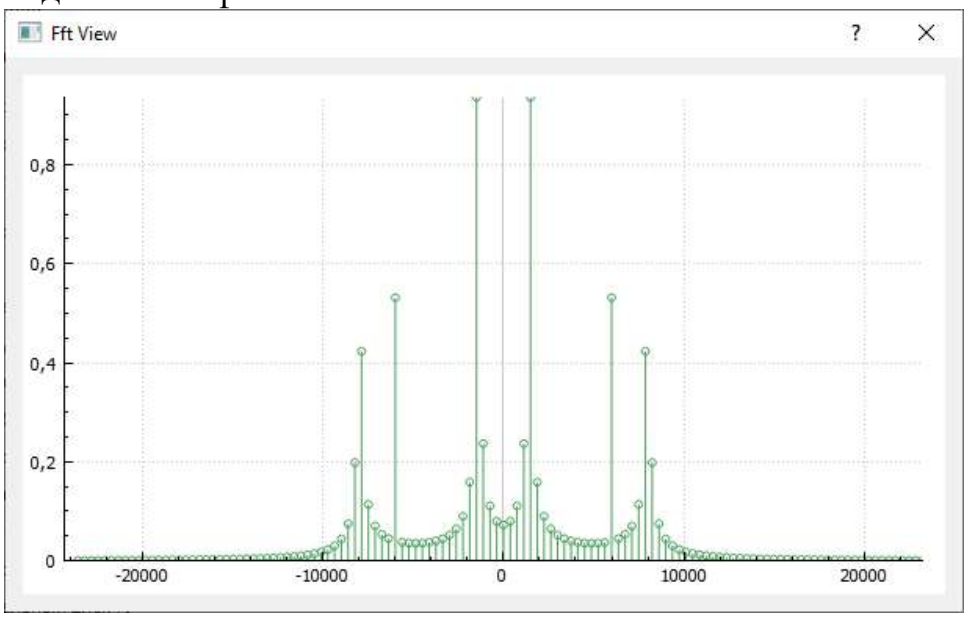

Рисунок 7 График БПФ в окне устройства «Qt График БПФ»