**Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

В.В. Кондратьев

Д.Д. Бурлаков

Э.П. Хотнянский

Д.В. Шпилёв

К.С. Покладов

Документация по теме

**Разработка Frequency Shifter и его реализация на языке С++**

Новосибирск

2022

**СОДЕРЖАНИЕ**

Введение…………………………………………………………………………3

1. Основные теоретические сведения…………………………………………4
   1. Теоретическое определение спектра сигнала и его сдвига…………..4
   2. Библиотеки С++…………………………………………………………7
2. Математическая модель Frequency Shifter………………………………….9
   1. Вводные данные…………………………………………………………9
   2. Дискретное преобразование Фурье…………………………………….11
   3. Смещение частоты………………………………………………………12
   4. Обратное дискретное преобразование Фурье…………………………14
3. Программирование математической модели на языке С++…………….....15
4. Написание Frequency Shifter с использованием звука из микрофона на языке С++……………………………………………………………………..21

Заключение……………………………………………………………………….26

Библиография…………………………………………………………………….27

**ВВЕДЕНИЕ**

**1. Основные теоретические сведения**

**1.1 Теоретическое определение спектра сигнала и его сдвига**

**Общие положения**

Frequency Shifter перемещает частоту входящего звука вверх или вниз на указанное пользователем количество герц. В представленном виде к входящему сигналу добавляется точно такой же сигнал, который регулируется по частоте. Например, если на входе синусоидальный сигнал 440 Гц и частота сдвига установлена в 300 Гц, на выходе будет сигнал с частотой 740 Гц.

**Сигнал**

Сигнал - это изменяющаяся во времени физическая величина, описываемая функцией времени. Один из параметров этой функции содержит информацию о другой физической величине. Такой параметр сигнала называют информативным, а физическую величину, которой представлен сигнал, - носителем сигнала.

**Спектр**

Спектр сигнала – коэффициенты разложения сигнала о базисе ортогональных функций. Само разложение называют спектральным разложением сигнала. В радиотехнике для разложения обычно используются классическое **преобразование Фурье**.

Разложение сигнала в спектр применяется в анализе прохождения сигналов через электрические цепи. Спектр периодического сигнала является и представляет набор гармонических колебаний, в сумме составляющий исходный сигнал. Одним из преимуществ разложения сигнала в спектр является **изменение сигнала** проходя по цепи (усиление, задержка, модулирование, детектирование, изменение фазы, ограничение и т.д.).

**Частота дискретизации**

Дискретизация - представление непрерывной функции дискретной совокупностью её значений при разных наборах аргументов. Для функции переменной f(x) – представление её множеством n её значений на заданном множестве аргумента.

Частота дискретизации в свою очередь, это частота взятия отсчётов непрерывного по времени сигнала при его дискретизации. Чем выше частота дискретизации, тем более широкий спектр сигнала может быть представлен в дискретном сигнале.

**Частота Найквиста**

Частота Найквиста названная в честь Гарри Найквиста, является характеристикой сэмплера (операция, которая извлекает выборки из непрерывного сигнала), который преобразует непрерывную функцию или сигнал в дискретную последовательность. Её значение составляет половину частоты дискретизации. Когда самая высокая частота сигнала меньше частоты Найквиста то результирующая последовательность дискретного времени не содержит искажений.

В применении сэмплирования сначала выбирается самая высокая частота, которая будет сохранена и воссоздана на основе ожидаемого контента и желательной точности воспроизведения. Затем перед сэмплером вставляется фильтр сглаживания. Его работа заключается в ослаблении частот выше этого предела. Наконец, на основе характеристик фильтра выбирается частота дискретизации, которая обеспечит приемлемо малое сглаживание.

**Канал тональной частоты** является основной единицей, по которому судят о количестве передаваемой информации. Обеспечивает передачу электрических сигналов связи в эффективно передаваемой полосе частот 0,3 – 3,4 кГц.

**Преобразования Фурье** – операция, сопоставляющая одной функции вещественной переменной другую функцию вещественной переменной. Эта новая функция описывает коэффициенты прим разложении исходной функции на элементарные составляющие – гармонические колебания с разными частотами.

Прямое преобразование Фурье записывает так:

Обратное преобразование Фурье:

Между сигналом и его спектром существует однозначное соответствие. Для практических приложений важно установить связь между преобразованием сигнала и соответствующим сигнала и соответствующим этому преобразованию изменением спектра. Из многочисленных возможных преобразований сигнала рассмотрим только **сдвиг спектра сигнала по частоте**.

Применим преобразование Фурье к произведению :

Первый интеграл в правой части является спектральной плотностью функции при частоте , а второй интеграл при частоте . Поэтому полученное выше соотношение можно записать в форме:

Где – спектральная плотность сигнала .

Из этого выражения вытекает, что расщепление спектра на две части, смещенные в соответственно на и , эквивалентно умножению функции на гармоническое колебание (при .

**1.2 Библиотеки С++**

**Библиотека алгоритмов для цифровой обработки сигналов dspl-2.0.**

Из данной библиотеки нам понадобятся алгоритмы дискретного и быстрого преобразования Фурье (БПФ).

**Структура fft\_t** – структура данных объекта быстрого преобразования Фурье. Структура хранит указатели на массивы поворотных коэффициентов и массивы промежуточных данных алгоритма быстрого преобразования Фурье.

Структура заполняется функцией **fft\_create** один раз до использования алгоритма БПФ. Указатель на объект данной структуры может быть многократно использован при вызове функций БПФ. Перед выходом из программы выделенную память под повторные коэффициенты и массивы промежуточных данных необходимо очистить функцией **fft\_free**.

**Функция fft\_free() –** Очистить структуру **fft\_t** алгоритма БПФ. Функция производит очищение памяти промежуточных данных и векторов поворотных коэффициентов структуры **fft\_t**.

**Функция fft\_create()**  **–** Заполнение структуры **fft\_t** для алгоритма БПФ. Функция производит выделение памяти и расчёт векторов повторных коэффициентов *n -* точечного БПФ для структуры **fft\_t**.

**Функция complex\_t –** комплексный тип данных определён как массив из двух элементов типа double. При этом первый элемент массива определяет реальную часть комплексного числа, а второй – мнимую.

**Функция fft()** – быстрое преобразование Фурье вещественного сигнала

Функция рассчитывает *n –* точечное быстрое преобразование фурье вещественного сигнала .

где .

**Функция ifft\_cmplx() –** Обратное быстрое преобразование Фурье

Функция рассчитывает *n –* точечное обратное быстрое преобразование Фурье от .

где .

**API DirectSound**

DirectSound – API для ввода и вывода звука. Выбор пал на данное API из-за хорошей реализации атомарной очереди и простом использовании.

**2. Математическая модель Frequency Shifter**

**2.1 Вводные данные**

f1 =500 Гц

f2 = 5500 Гц

f3 = 3400 Гц

f4 = 1000 Гц

f5 = 4000 Гц

N = 2048 – количество отсчётов

fd = 24000 Гц – частота дискретизации

t = 0, 0.00001 … 0.01 с.

Рассчитаем частоту равную расстоянию между отсчётами:

И количество отсчётов на которое произойдёт смещение вверх и вниз:

– используем для смещения вниз

– используем для смещения вверх

Зададим синусоидальный сигнал переменной sin\_r:

И выведем его на график.

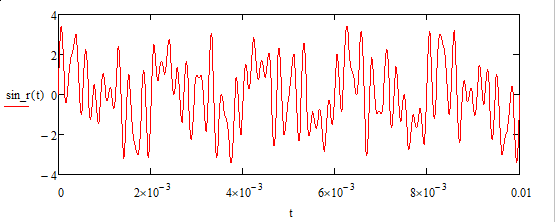


Рисунок 1. График зависимости сигнала sin\_r от времени t.

Используя функцию округления в большую сторону ceil() округлим наши количества отсчётов для смещения:

Найдём период дискретизации:

Напишем программу массива нашего сигнала от произведения периода дискретизации на количество отсчётов:

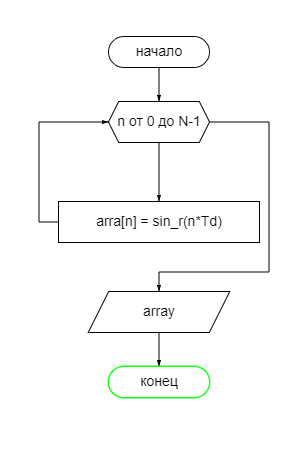


Рисунок 2. Блок-схема массива сигнала.

Выведем его на график

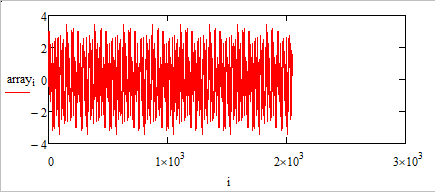


Рисунок 3. График зависимости массива от i, где i = 0, 1…N-1.

**2.2 Дискретное преобразование Фурье**

Наш ранее полученный массив преобразуем с помощью дискретного преобразования Фурье:

Получим такой массив:

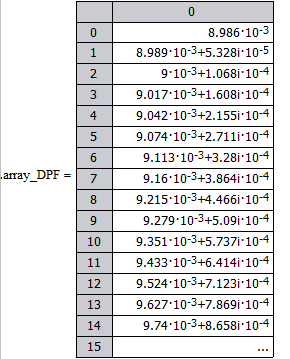


Рисунок 4. Массив дискретного преобразования Фурье.

По заданным значениям построим график.

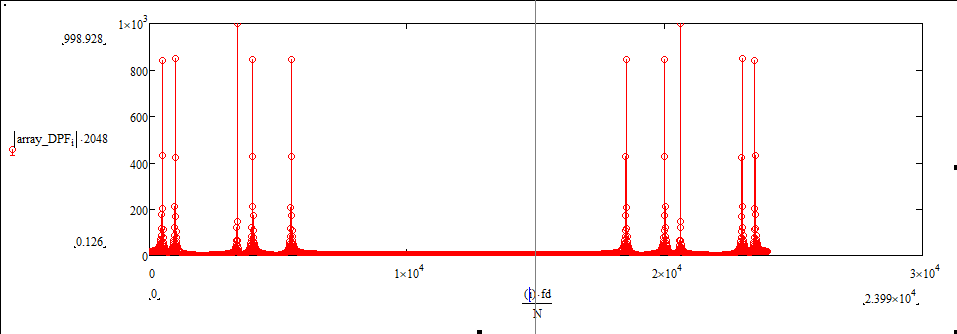


Рисунок 5. График зависимости массива ДПФ от частоты.

**2.3 Смещение частоты**

Создадим массив нулей размерностью 2048 для дальнейшего использования.

Так как мы при преобразовании и обработке сигнала ограничиваемся интервалом частот от 300 до 3400 Гц (КТЧ). То для грамотного использования нашего спектра обрежем его до предела в 4000 Гц. И посчитаем номер отсчёта этой частоты.

С помощью функции ceil() округлим до 342.

Используя функцию submatrix() вырежем из массива array\_DPF нужное количество отсчётов. Это будет наша полоса частот, которую мы будем смещать.

Напишем программы для смещения частоты вверх и вниз:

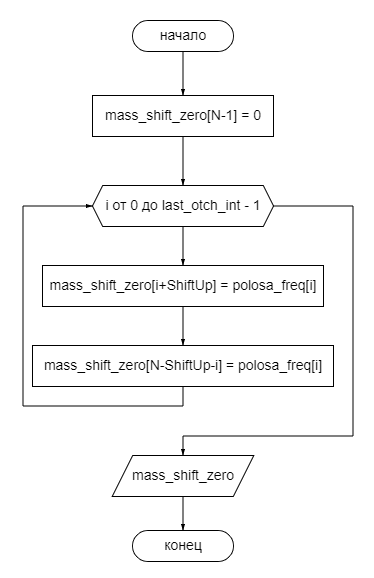


Рисунок 6. Блок-схема программы для смещения частоты вверх.

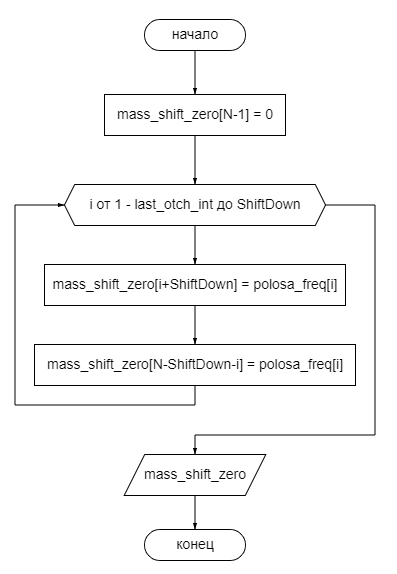


Рисунок 7. Блок схема программы для смещения частоты вниз.

Выведем графики полученных смещений:

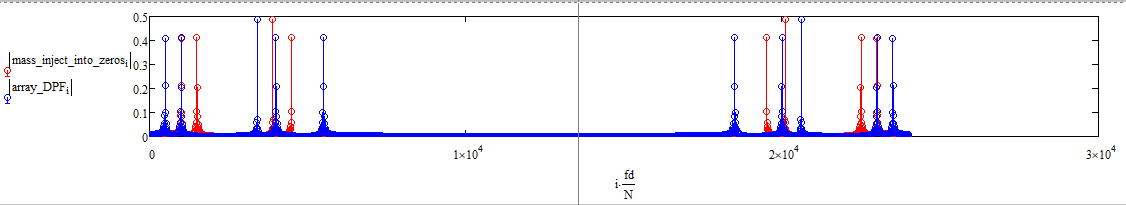


Рисунок 8. График смещения частоты вверх в зависимости от значений array\_DPF.

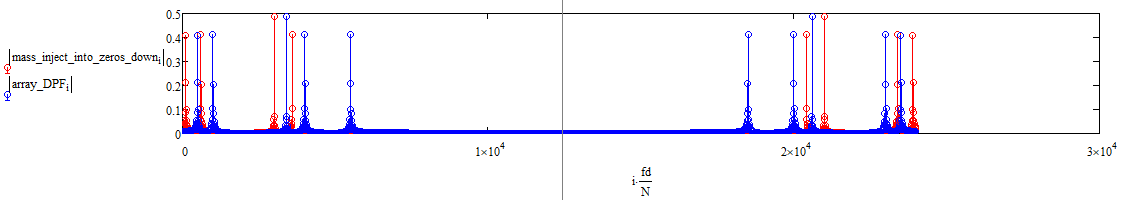


Рисунок 9. График смещения частоты вниз в зависимости от значений array\_DPF.

**2.4 Обратное дискретное преобразование Фурье**

Для того, чтобы из нашего смещённого спектра сделать обратно синусоидальный сигнал нужно использовать функцию ICFFT (ОДПФ) на полученном массиве смещения.

Выведем наши сигналы на график и посмотрим, что с ними стало

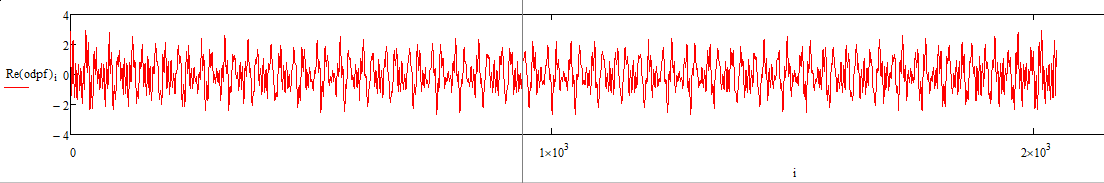


Рисунок 10. Действительная часть сигнала после смещения по частоте вверх.

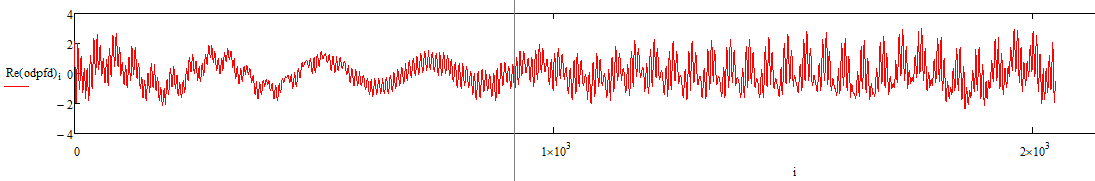


Рисунок 11. Действительная часть сигнала после смещения по частоте вниз.

**3. Программирование математической модели на языке С++**

Для программирования используем те же исходные данные, что были в математической модели.

Подключаем все стандартные библиотеки:

iostream

fstream

cmath math.h – математические функции

dspl.hpp – описывалась раньше для дискретного преобразования Фурье

Задаём количество отсчётов 2048 и частоту дискретизации 24000Гц. Далее нужно задать комплексный массив (БПФ) complexSignal[N].

Создадим функцию для добавления в ранее созданный файл мнимой и действительной части для построения графиков в математической модели.

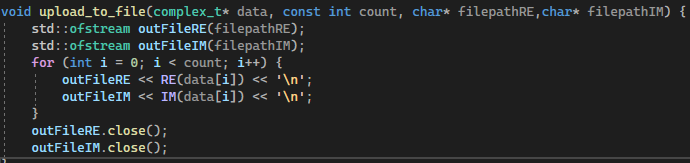


Рисунок 1. Функция вывода мнимой и действительной части спектра в файл.

где data – комплексный массив, count – его размерность, filepathRe/Im(outFileRe/Im) – путь к файлу и соответственно поток файла.

Так же нужно будет записать в файл значения синусоидального сигнала для этого создаём функцию.

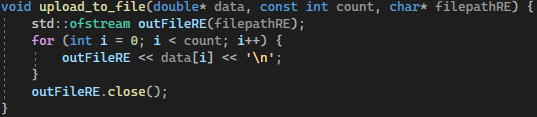


Рисунок 2. Функция вывода действительной части сигнала в файл.

Теперь можем написать функцию для смещения **частоты вверх**. Создадим БПФ объект, заполненный нулями, комплексный массив, также заполненный нулями далее используем всё те же формулы что и в математической модели для нахождения частоты одного отсчёта, количество отсчётов на которое смещаем, последний отсчёт в нашей полосе частот (Канал тональной частоты) и с помощью цикла производим смещение. С помощью обратного преобразования Фурье делаем из спектра синусоидальный сигнал, далее выводим этот сигнал в массив и записываем значения действительной и мнимой частей спектра, смещенного спектра и смещенного сигнала в отдельные файлы.

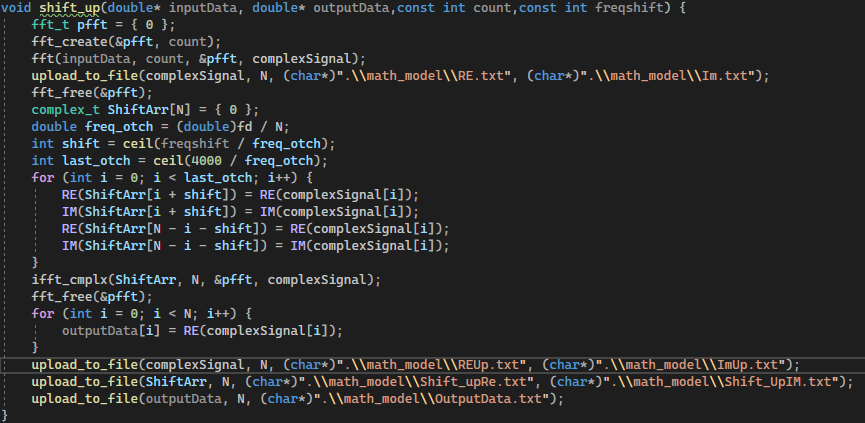


Рисунок 3. Функция для смещения частоты вверх.

Смещение **частоты вниз** описывается точно также как смещение частоты вверх.



Рисунок 4. Функция для смещения частоты вниз.

В конце кода записываем функцию main() с исходными данными и вызовом функций описанными выше.

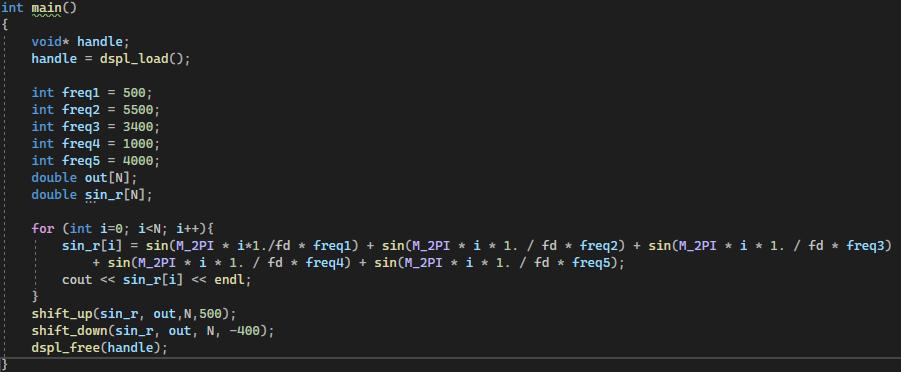


Рисунок 5. Функция main().

Чтобы проверить правильность расчётов нашей программы вернёмся в нашу математическую модель и присвоим нашим файлам с записями значения и сравним с графиками, которые были до этого.

FileRe = RE.txt

FileIm = Im.txt

И с помощью цикла соберём все значения в один комплексный массив:

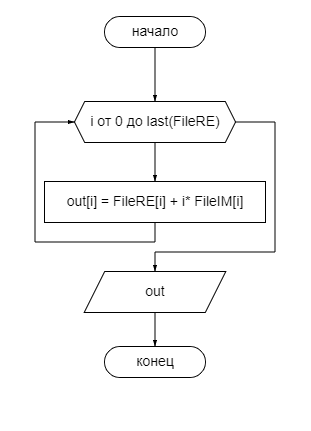


Рисунок 6. Блок-схема цикла расчёта значений из ДПФ спектра сигнала из расчётов программы.

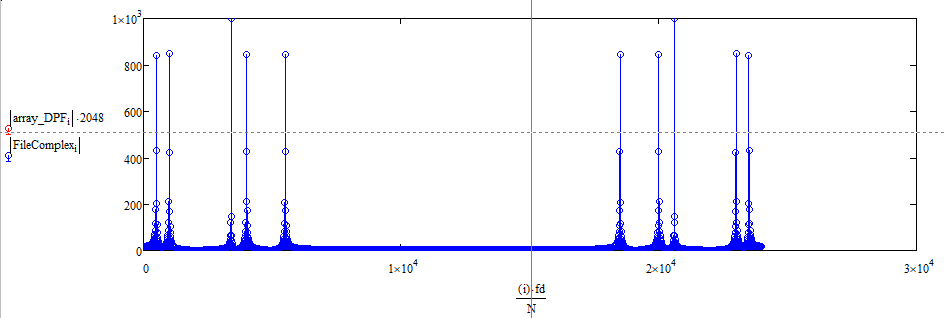


Рисунок 7. График спектра сигнала из расчётов программы.

Из графика видно, что спектр рассчитанный с помощью данных программы совпадает со спектром, который был получен ранее.

Также мы можем проверить правильность смещения частоты. Используя файлы Shift\_UpIm.txt, ShiftUpRe.txt, Shift\_DownIm.txt и Shidt\_DownRe.txt напишем циклы для расчёта значений смещения спектра вверх и вниз.

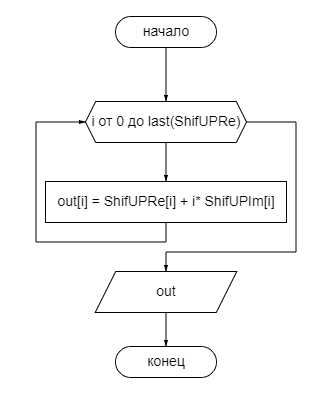


Рисунок 8. Блок-схема цикла расчёта значений смещения частоты вверх.

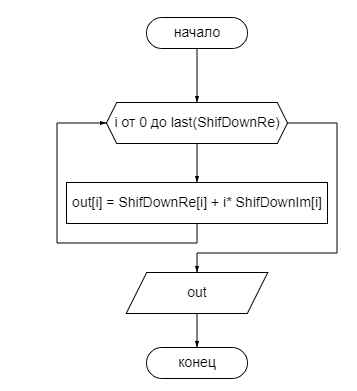


Рисунок 9. Блок-схема цикла расчёта значений смещения частоты вниз.

При построении графиков получим

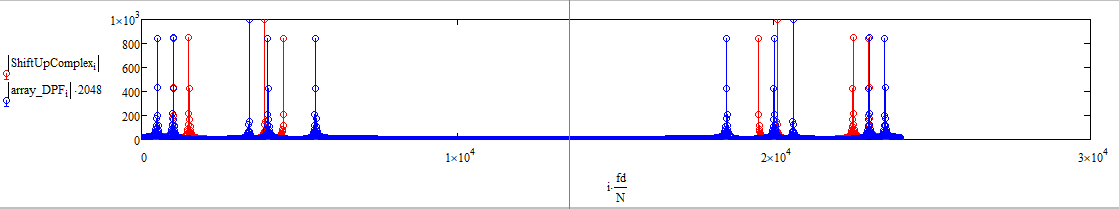


Рисунок 10. График смещения частоты вверх из расчётов программы

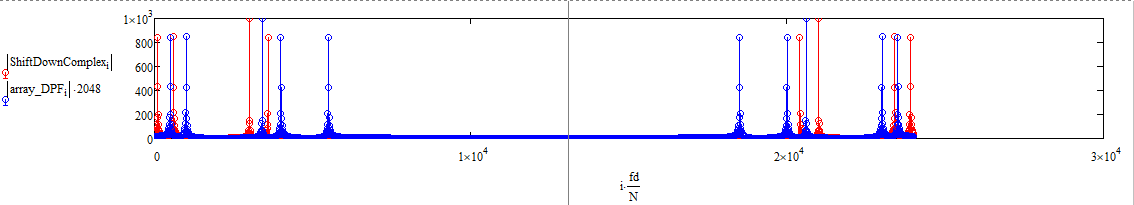


Рисунок 11. График смещения частоты вниз из расчётов программы

Сравнив данные спектры сигналов с рисунком 8 (вверх) и рисунком 9 (вниз) из второго пункта увидим, что они одинаковые, значит программа правильно производит смещение частоты.

Проверим финальный сигнал, который мы будем слышать при полной реализации программы. Построим графики сигналов со смещением с помощью файлов OutputData.txt и OutputDataDown.txt и сравним с переменной odpf заданной в 2 пункте.

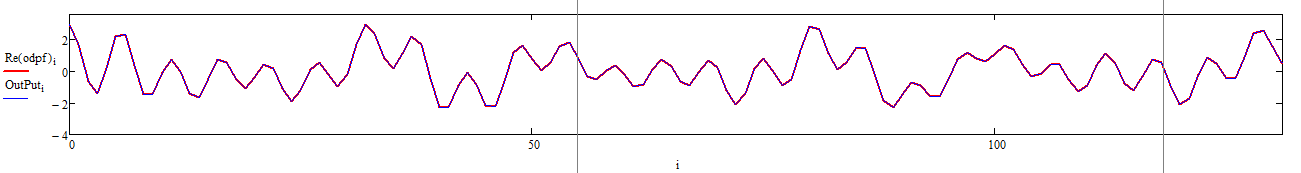


Рисунок 12. Сравнение выходного сигнала программы и в математической модели со смещением частоты вверх

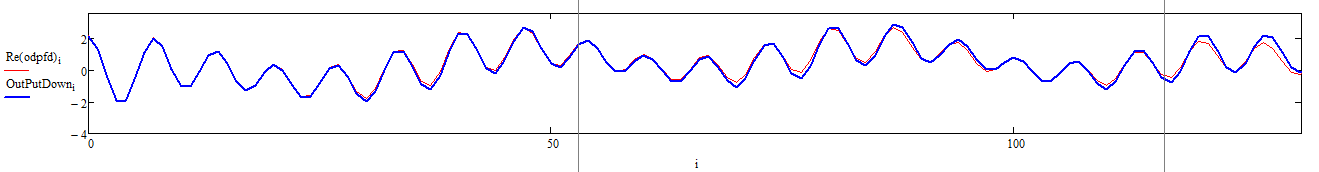


Рисунок 13. Сравнение выходного сигнала программы и в математической модели со смещением частоты вниз

Из всех сравнений можно сделать вывод, что программа правильно работает и смещение фаз происходит как в математической модели. Можем перейти к финальной реализации проекта и написать программу для работы со звуком.

**4. Написание Frequency Shifter с использованием звука из микрофона на языке С++**

Для написания программы нам понадобится всё то, что мы сделали раньше, но с некоторыми изменениями.

Подключим дополнительные библиотеки:

csignal

AtomicQueue.hpp

Audio\_player\_v2.hpp

Audio\_recorder\_v2.hpp

Чтобы каждый раз не открывать файлы в функциях откроем их потоки в самом начале

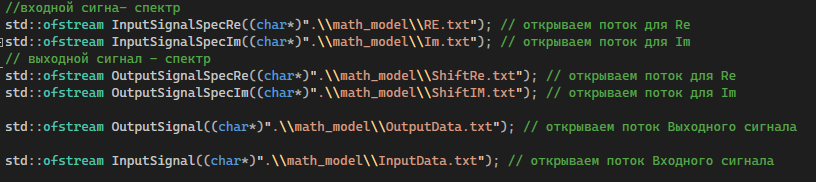


Рисунок 1. Открытие потоков файлов.

Функции upload\_to\_file, shift\_up, shift\_down оставляем без изменений.

Для того, чтобы использовать звуковой файл создадим функцию void loadsignalbuffer

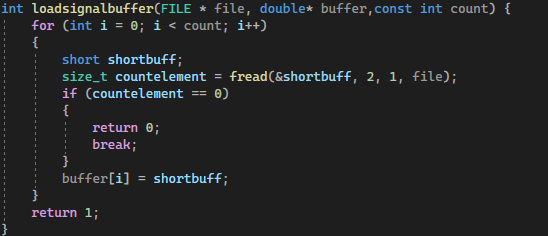


Рисунок 2. Функция взятия отсчётов из звукового файла.

Аналогичная функция для записи изменённого сигнала в файл

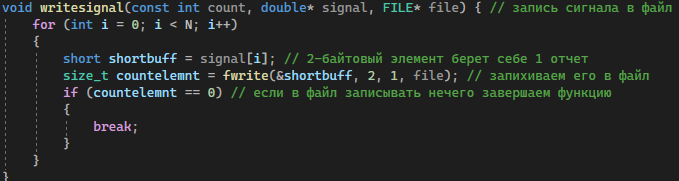


Рисунок 3. Функция записи изменённого сигнала в файл.

Далее чтобы слышать наш изменённый звук сразу из динамиков не включая файл, добавим функцию void secmain(), которую опишем дальше. И функцию вывода звука на динамики из файла void secmainfile().

В конце создадим функцию int main() с вызовом всех функций и закрытием используемых файлов

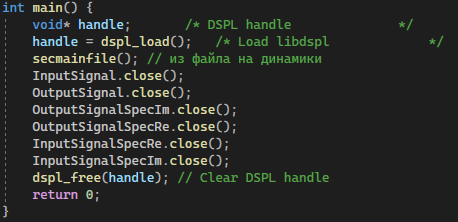


Рисунок 4. Функция main().

Создадим новый .CPP файл назовём его Potok в котором будем описывать ранние сказанные функции void secmain() и void secmainfile().

Перед началом добавим все прошлые библиотеки, а также объявим глобальные переменные с помощью #define, такие как частоту дискретизации, бит на семпл и типа атомик очереди (short).

Из библиотеки атомарных очередей добавим буфер-очередь записи с микрофона, для воспроизведения на динамик, а также классы звукозаписи и воспроизведения.

Зададим переменную для количества элементов очереди, которые будут воспроизводится в течении секунды, флаг остановки и число отсчётов.

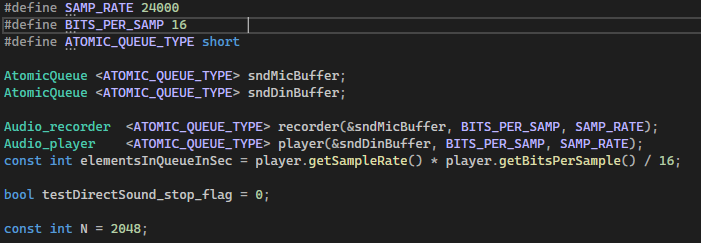


Рисунок 5. Первоначальные настройки.

Чтобы мы не слышали изменённый звук сразу после нашего голоса добавим задержку воспроизведения.

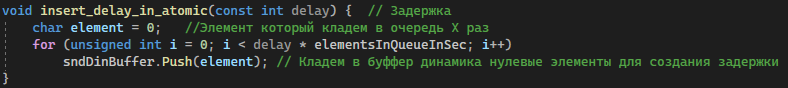


Рисунок 6. Функция задержки.

Чтобы наша обработка сигнала не шла всё время с помощью этой функции (Рисунок 7) при нажатии сочетания клавиш ctrl+c мы сможешь остановить обработку, и программа завершит своё выполнение

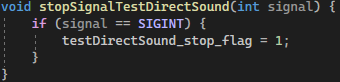


Рисунок 7. Функция stopsignalTestDirectSound.

Чтобы показать всю информацию пользователю выведем её на экран

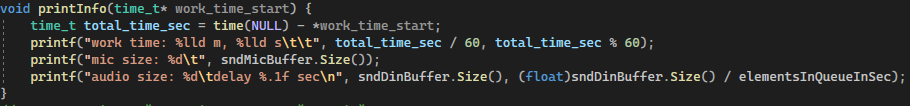


Рисунок 8. Функция printInfo.

Так как функции смещения находятся у нас в файле main.cpp произведём их объявление, чтобы наш файл смог их использовать.



Рисунок 9. Объявление функций из файла main.

Теперь приступим к написанию функции вывода звука на динамики из файла (secmainfile()). Нам нужно установить обработчик сигнала на обработку SIGIN, добавить задержку между записью звука и воспроизведением. Запустим одновременно запись и плеер. Далее объявим 2 массива длиной 2048, в один будем записывать входной сигнал, в другой выходной. Добавим переменную с типом clock\_t для того, чтобы засечь время вывода информации в цикле.

Откроем файл с входным сигналом, выставим задержку (в примере 0 с.). Добавим условие для проверки существования файла, если его нет, закрываем программу. В противном случае запускаем цикл while до момента пока пользователь не нажмёт ctrl+c. В цикле производится смещение частоты вверх или вниз. Вставляем поток динамика наш смещенный сигнал. И создаём условие для вывода информации раз в Х секунд. Если оно выполняется выводи на экран отладочную информацию и сбрасываем таймер.

По окончании (пользователь нажал ctrl+c) цикла закрываем файл, останавливаем запись и воспроизведение.

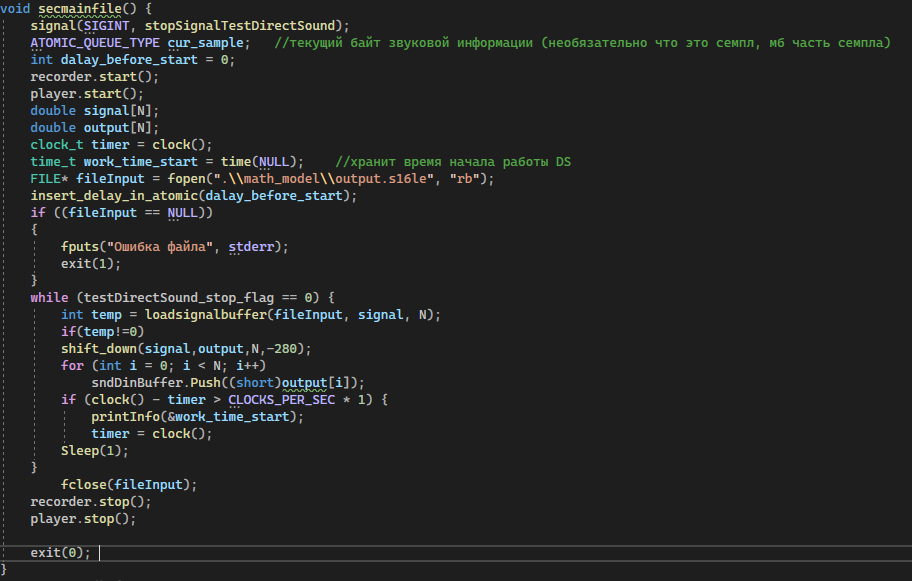


Рисунок 10. Функция secmainfile().

Запишем финальную функцию secmain(), которая позволяет выводить звук на динамик с микрофона.

Данная функция отличается от secmainfile() только изменением содержимого цикла while, так что всё что до него оставляем прежним.

В цикле создаётся условие, что только при наличии 2048 семплов в очереди микрофона, программа начинает работу с данными. Далее создаётся массив, в который добавляем эти семплы. Производим смещение частоты вверх или вниз и также как в прошлой функции выводим наш сигнал в динамики. Также выводим информацию раз в Х секунд. И при нажатии сочетании клавиш ctrl+c завершаем работы программы.

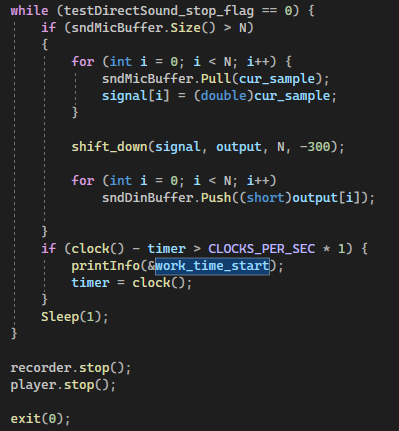


Рисунок 11. Цикл while в функции secmain().

На этом написание Frequency Shifter на языке С++ завершено, и мы можем прослушивать изменённый голос со сдвигами частот вверх и вниз.

**ЗАКЛЮЧЕНИЕ**

**БИБЛИОГРАФИЯ**

1. Некоторые свойства преобразования Фурье [Электронный ресурс]// scask.ru URL: https://scask.ru/q\_book\_g\_rts.php?id=14 (дата обращения 11.12.2022)
2. DSPL-2.0 – библиотека алгоритмов цифровой обработки сигналов [Электронный ресурс]// ru.dsplib.org URL: https://ru.dsplib.org/dspl/ (дата обращения 11.11.2022)
3. Гармонический анализ непериодических сигналов [Электронный ресурс]// scask.ru URL: https://scask.ru/q\_book\_g\_rts.php?id=12 (дата обращения 11.11.2022)
4. Программирование звука в Directsound [Электронный ресурс]// codenet.ru URL: http://codenet.ru/progr/directx/dxsound.php (дата обращения 15.11.2022)
5. Документация по Directsound [Электронный ресурс]// github.com URL: https://github.com/ExplosiveE137/Shifter-CHO (дата обращения 15.11.2022)