Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ»

кафедра физики

Задание № 4 по дисциплине

"Физические основы информационных технологий"

Название: Фильтрация звукового сигнала.

|  |  |
| --- | --- |
| Фамилия И.О.: | Ягодаров М. А. |
| Группа: | 1303 |
| Преподаватель: | Альтмарк А.М. |
| Итоговый балл: |  |
|  |  |

Крайний срок сдачи: 05.12.23

Санкт-Петербург 2023

**Условие задания 4**

На входе приемника получен звуковой сигнал в двоичном коде (рис.1.). Необходимо перевести двоичный код в десятичный и затем провести над аналоговым сигналом процедуру фильтрации от высокочастотных помех. Для фильтрации необходимо использовать пассивные фильтры (фильтры без дополнительного источника питания), которые могут в себя включать, резисторы, конденсаторы и катушки индуктивности.

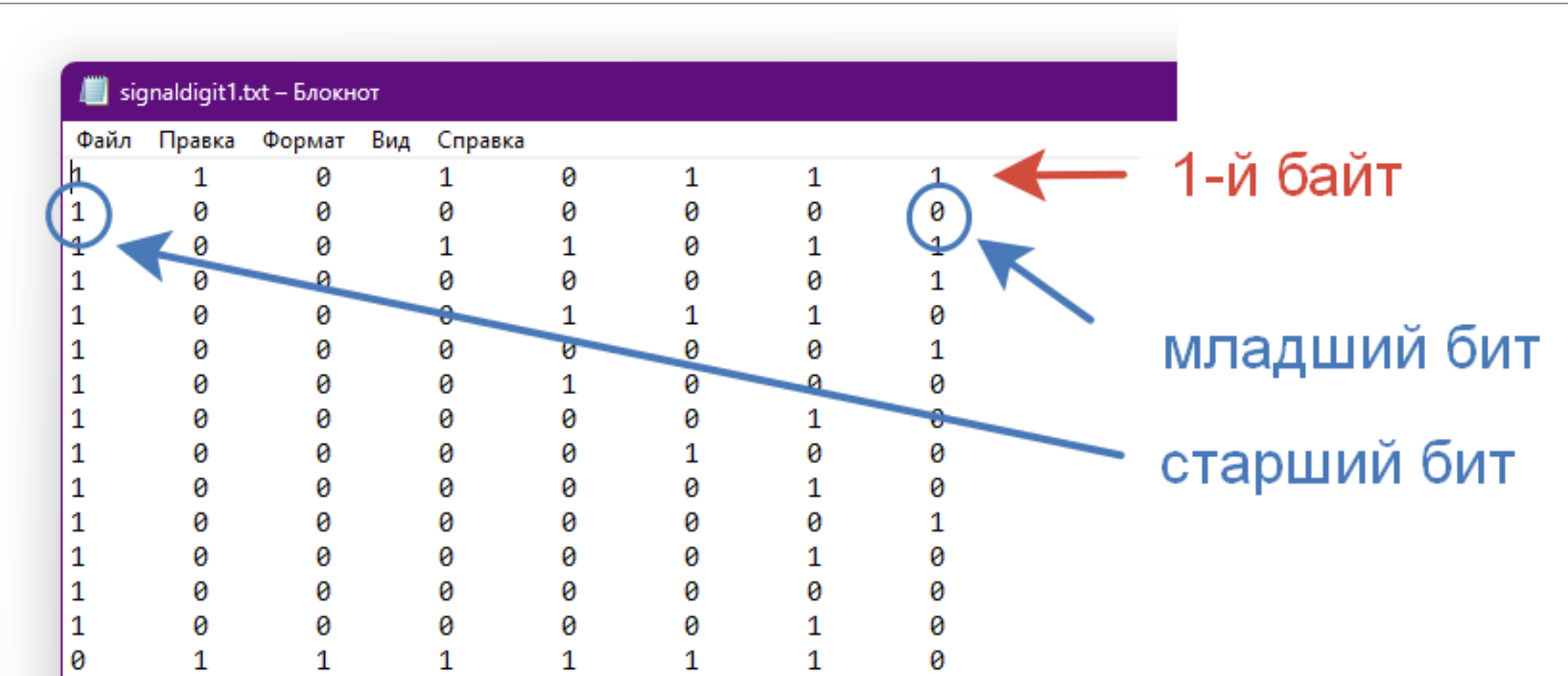


Рис.1. Структура данных в текстовом файле с сигналом

Исходные данные нужно взять в файле FOIT\_IDZ4.xlsx. В отчет нужно включить график сигнала во временной области и его спектр, схему фильтра и АЧХ его передаточной функции, спектр фильтрованного сигнала, а также график выходного сигнала во временной области. Файл IDZ4.txt должен содержать ответ на вопрос, который записан в звуком сигнале.

Помимо текстового файла IDZ4.txt в папке IDZ4 должен находиться Word-файл с отчетом, а также файл с кодом (Python, Mathcad, Mathematica). Для лучшего понимания отчетности смотрите папку “Пример организации яндекс-папки студентов”.

Пример содержания файла IDZ4.txt:

25

**Исходные данные (17 вариант)**

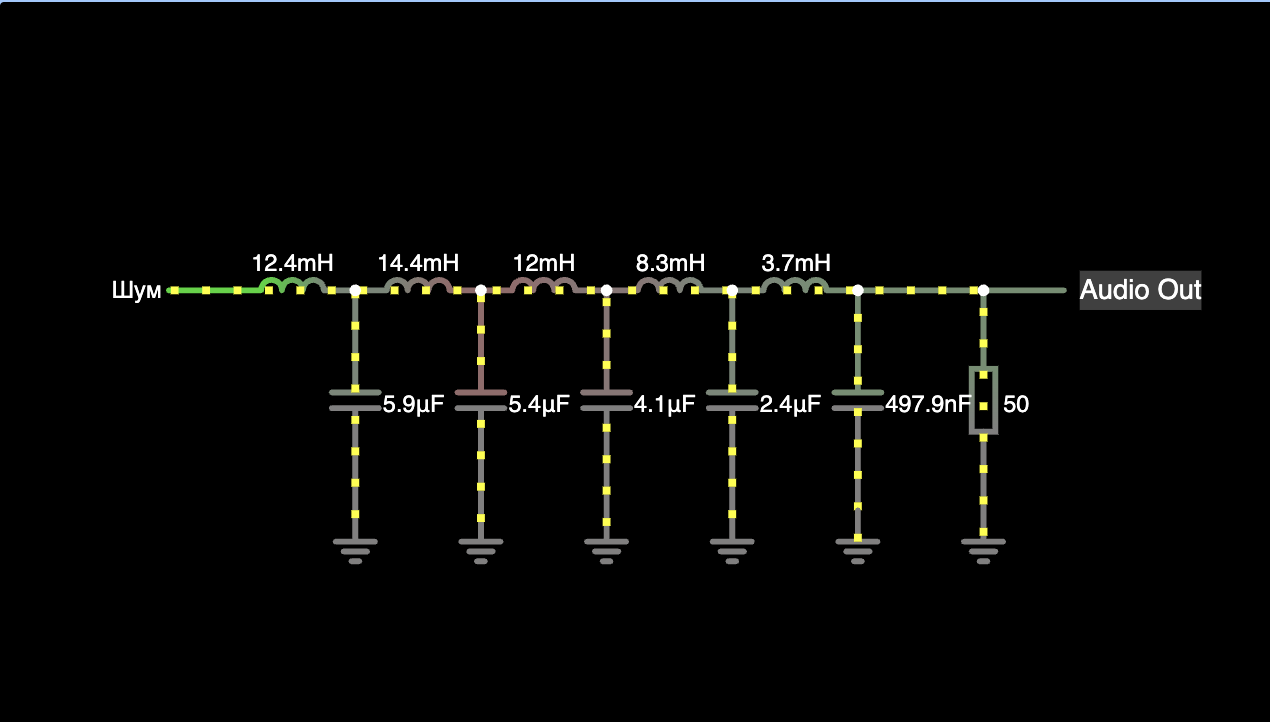
Длительность сигнала 4.25с

**Теоретические сведения**

Для отчистки сигнала от помех используются фильтры.

В данной работе используется фильтр Баттерворта.

АЧХ его передаточной функции имеет вид ступеньки, что позволяет достаточно точно обрезать высокие частоты шума, почти не ослабив основной сигнал. Схема фильтра представлена на рисунке.



**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл main.py

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, ifft

from scipy.io.wavfile import write

coef = 1.2

t = 4.25

file\_path = 'signaldigit17.txt'

R = 50

L1 = 12.4E-3 \* coef

L2 = 14.4E-3 \* coef

L3 = 12E-3 \* coef

L4 = 8.3E-3 \* coef

L5 = 3.7E-3 \* coef

C1 = 5.9E-6 \* coef

C2 = 5.4E-6 \* coef

C3 = 4.1E-6 \* coef

C4 = 2.4E-6 \* coef

C5 = 497.9E-9 \* coef

sample\_rate = 41400

def zl(l, omega):

return 1j \* omega \* l

def zc(c, omega):

return 1 / (1j \* omega \* c)

def butterworth\_filter(omega):

z5 = 1 / (1 / R + 1 / zc(C5, omega))

z4 = 1 / (1 / zc(C4, omega) + 1 / (z5 + zl(L5, omega)))

z3 = 1 / (1 / zc(C3, omega) + 1 / (z4 + zl(L4, omega)))

z2 = 1 / (1 / zc(C2, omega) + 1 / (z3 + zl(L3, omega)))

z1 = 1 / (1 / zc(C1, omega) + 1 / (z2 + zl(L2, omega)))

zl1 = zl(L1, omega)

z = zl1 + z1

i\_in = 1 / z

u1 = i\_in \* z1

i1 = u1 / (z2 + zl(L2, omega))

u2 = i1 \* z2

i2 = u2 / (z3 + zl(L3, omega))

u3 = i2 \* z3

i3 = u3 / (z4 + zl(L4, omega))

u4 = i3 \* z4

i4 = u4 / (z5 + zl(L5, omega))

u5 = i4 \* z5

return u5

def save\_audio(signal, filename):

write(filename, 44100, np.array(signal).astype(np.float32))

with open(file\_path, 'r') as file:

discrete\_signal = [line.strip().split('\t') for line in file]

analog\_signal = [int(''.join(b), 2) for b in discrete\_signal]

dt = t / len(discrete\_signal)

# Input signal

time\_axis = np.linspace(0, t, len(analog\_signal))

plt.plot(time\_axis, analog\_signal)

plt.title('Analog Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.show()

# Spectre calculation

fsig = fft(analog\_signal)

out\_n = len(fsig)

df = 1 / dt

freq\_axis = np.fft.fftfreq(out\_n, dt)[1:]

plt.plot(freq\_axis, np.abs(fsig[1:]))

plt.title('Spectrum')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.show()

# Filter's frequency response

freq\_range = np.linspace(1, out\_n / 10, out\_n - 1)

filter\_response = np.abs(butterworth\_filter(freq\_range))

plt.plot(freq\_range, filter\_response)

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.show()

# Filter's application

H\_list = filter\_response

F\_new = fsig[1:] \* H\_list

plt.plot(freq\_axis, np.abs(F\_new))

plt.title('Filtered Spectrum')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.show()

# Inverse Fourier transform

filtered\_signal = np.real(ifft(np.concatenate(([0], F\_new))))

filtered\_time\_axis = np.linspace(0, t, len(filtered\_signal))

# Filtered signal plot

plt.plot(filtered\_time\_axis, filtered\_signal)

plt.title('Filtered Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.show()

# Play audio file

save\_audio(filtered\_signal, 'filtered\_signal.wav')