МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра телекоммуникационных систем и вычислительных средств (ТС и ВС)

Отчет по производственной практике
по дисциплине
SDR

по теме:

АРХИТЕКТУРА SDR-УСТРОЙСТВ. ПРИМЕРЫ ФОРМИРОВАНИЯ I/Q-СЭМПЛОВ ПРОИЗВОЛЬНОЙ ФОРМЫ. РАБОТА С БУФЕРОМ ПРИЕМА SDR

Студент:
Группа ИА-331 *Я.А Гмыря*

Предподаватели:
Лектор *Калачиков А.А*
Семинарист *Ахпашев А.В*
Семинарист *Попович И.А*

Новосибирск 2025 г.

СОДЕРЖАНИЕ

ЦЕЛЬ И ЗАДАЧИ

Цель:

Более детально рассмотреть принцип работы формирующего фильтра. Программно реализовать логику формирующего фильтра, передать мелодию по радиоканалу.

Задачи:

1. Прослушать и законспектировать лекцию.
2. На основе полученных знаний выполнить программную реализацию формирующего фильтра.
3. Отправить сигнал по радиоканалу, потом принять его и попытаться воспроизвести.

ЛЕКЦИЯ

Введение

На прошлом занятии мы познакомились с двумя типами модуляции: BPSK и QPSK и программно реализовали их. При реализации логики PSF нужно было формировать прямоугольный импульс. Я сделал это самым простым методом: увеличивал кол-во значений I и Q. За счёт этого получали I(t) и Q(t), которые уже длились во времени. Данный подход рабочий и не является ошибкой, но на практике не используется, поскольку подходит только в случае, когда формирующий фильтр имеет прямоугольную импульсную характеристику. Если импульсная характеристика имеет форму приподнятого косинуса, то такой метод не сработает. На этом занятии изучим более грамотный подход.

Почему форма символов так важна?

Форма передаваемых символлов $I_n(t), Q_n(t)$ определяет свойства спектра радиосигнала. Если форма символов прямоугольная, то форма спектра будет иметь вид функции $\frac{sinx}{x}$ и будет занимать большую ширину спектра. В реальных системах чаще всего используется форма приподнятого косинуса.

Upsampling

Итак, мы хотим, чтобы I и Q были не просто числами, а имели длительность, т.е хотим получить I(t) и Q(t). Необходимо установить число семплов, которое будут длиться I и Q. Введем параметр L, который измеряется в $\frac{sample}{symbol}$ и будет отвечать за кол-во семплов, приходящихся на 1 символ, т.е за длительность символа. Если мы хотим сделать символы длящимися, то необходимо поднять частоту дискретизации. Для этого существуют специальные блоки, которые называются Upsampling блоками. Их задача состоит в увеличении частоты дискретизации. Во сколько раз нужно увеличить частоту дискретизации? Если на каждый символ теперь приходится L семплов, то и частота дискретизации должна стать в L раз больше. За частоту дискретизации отве-

чает параметр f_symb (символьная скорость), который показывает скорость, с которой символы поступают из маппера. Соответсвенно после выхода из Upsampling блока получим $f_s = f_{symb} * L$, т.е кол-во семплов на секунду времени (sample_rate). Исходя из этого можно определить расстояние во времени между семплами $T_s = \frac{1}{f_s}$.

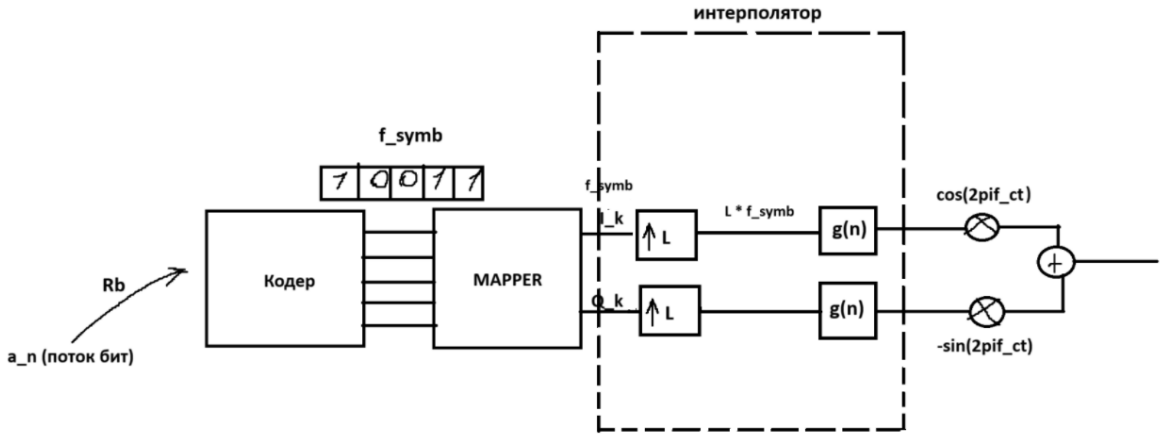

Рисунок 1 — Визуализация архитектуры передатчика

Техническая реализация Upsampling

Техническую реализацию проще будет показать на примере.

Пусть на выходе маппера мы получили I = [1, -1, 1], и хотим, чтобы каждый символ длился L = 4 семпла

Сформируем новую последовательность X(n), в которой между каждым I_n добавим L-1 нулей.

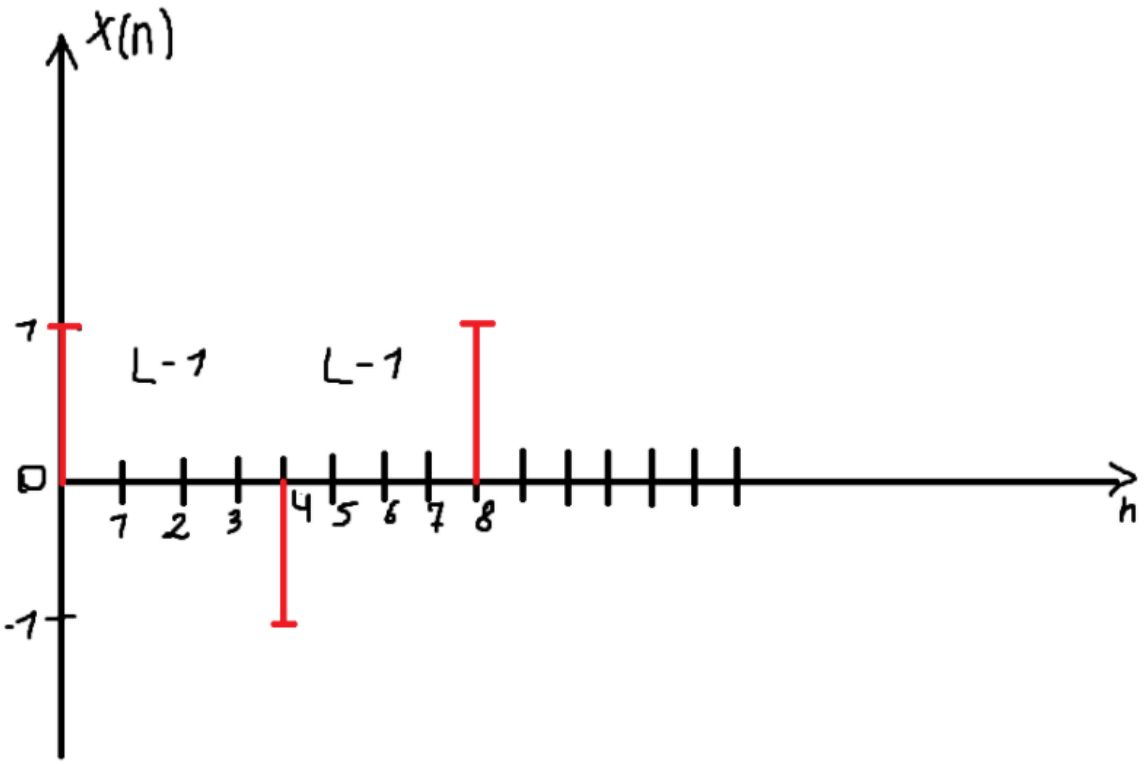

Рисунок 2 — Визуализация работы Upsampling

Получаем последовательность длинной 12 отсчетов (после последнего символа тоже идет 3 нуля).

Формирующий фильтр

На данный момент мы только ”растянули”символы, но не придали им никакой формы. Эти действия выполняет формирующий фильтр (на схеме g(n)).

В блоке g(n) происходят следующие расчеты: S(n) = $\sum_{m=0}^{L-1} X(m)g(n-m)$, где X(n) - отсчеты, g(n) - импульсная характеристика фильтра. Сама формула это дискретная свертка.

Импульсная характеристика фильтра имеет сложную форму, которая позволяет сделать сигнал любой формы.

Зададим форму импульсной характеристики. Для упрощения возьмем прямоугольную форму.

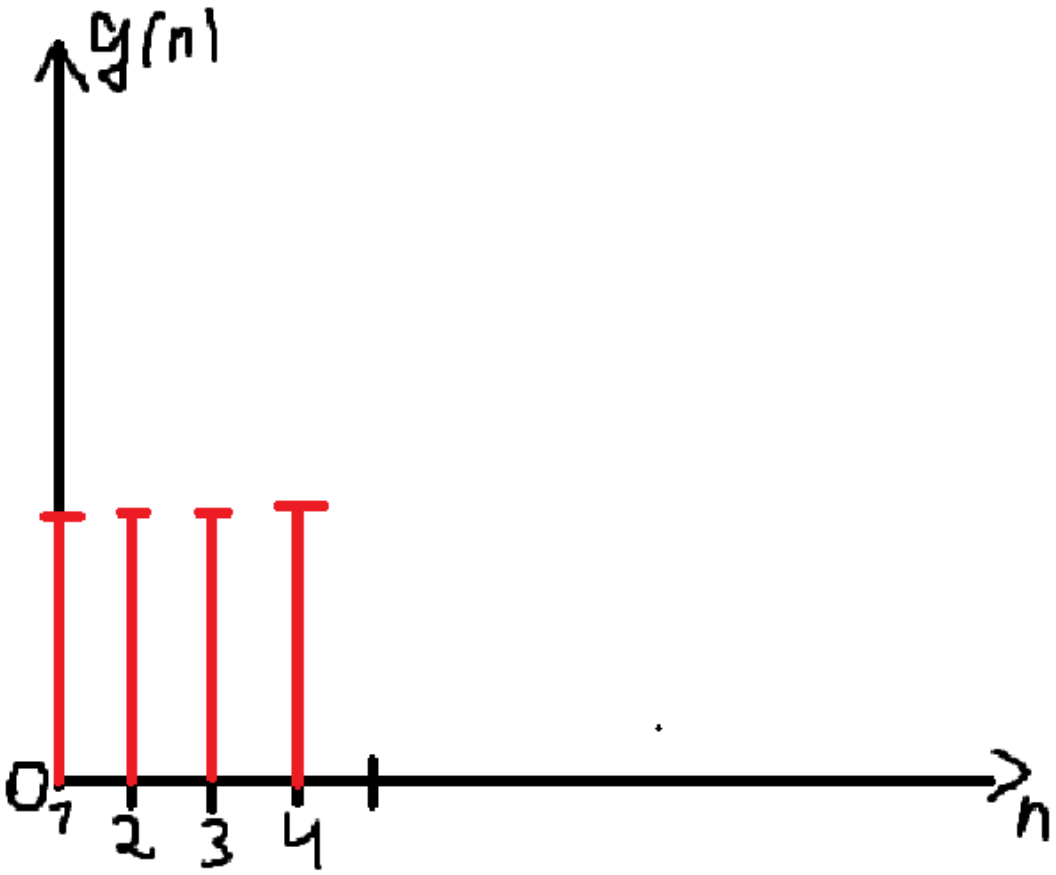

Рисунок 3 — Пример импульсной характеристики

Для иллюстрации работы фильтра произведем вычисления по формуле выше:

$$S(0) = X(0)g(0) = 1*1 = 1$$

$$S(1) = X(0)g(1) + X(1)g(0) = 1*1 + 0*1 = 1$$

$$S(2) = X(0)g(2) + X(1)g(1) + X(2)g(0) = 1*1 + 0*1 + 0*1 = 1$$

$$S(3) = X(0)g(3) + X(1)g(2) + X(2)g(1) + X(3)g(0) = 1*1 + 0*1 + 0*1 + 0*1 = 1$$

$$S(4) = X(0)g(4) + X(1)g(3) + X(2)g(2) + X(3)g(1) + X(4)g(0) = 1*0 + 0*1 + 0*1 + 0*1 + (-1 * 1) = -1$$

S(4) = X(0)g(5) + X(1)g(4) + X(2)g(3) + X(3)g(2) + X(4)g(1) + X(5)g(0) = 1*0 + 0*0 + 0*1 + 0*1 + (-1 * 1) + 0 * 1 = -1

Визуализируем символы после выхода из фильтра:

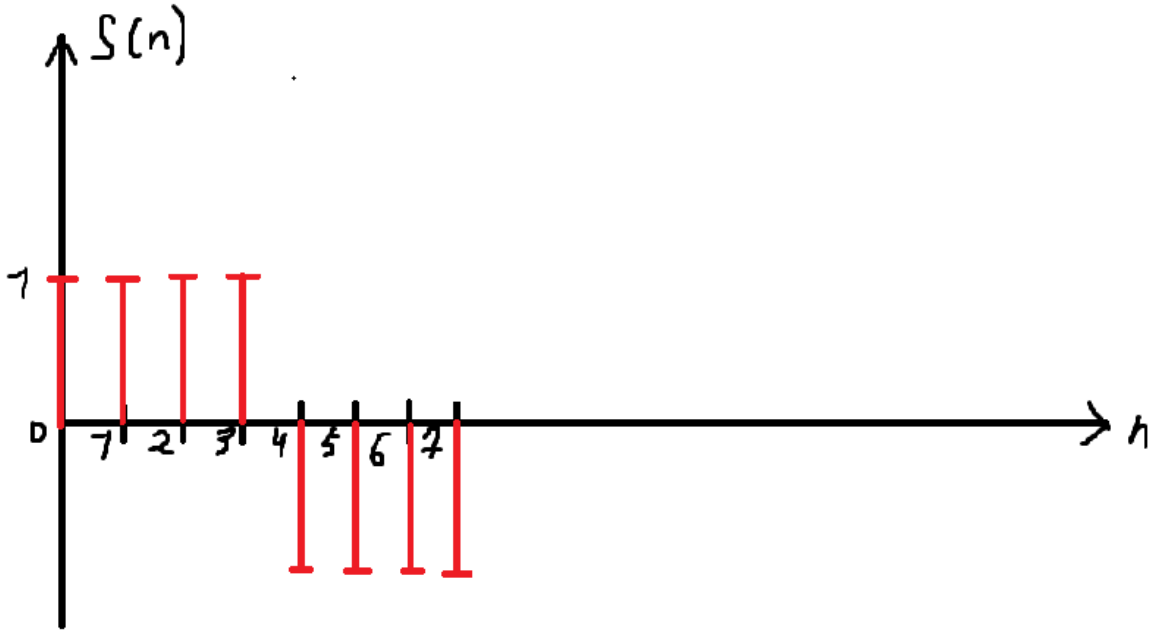

Рисунок 4 — Пример символов после выхода из фильтра

Получили растянутые во времени I и Q с прямоугольной формой. Таким образом можно задать сигналу любую форму.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Введение

На этом занятии отправим в радиоканал мелодию, а потом примем ее и попытаемся воспроизвести.

Конверторы

Для отправки мелодии необходимо конвертировать ее из .mp3 в .pcm формат (поток семплов), а потом после принятия конвертировать обратно из .pcm в .mp3. Для этого нам понадобятся конверторы.

Из .mp3 в .pcm

```
import numpy as np
import librosa
from pydub import AudioSegment

mp3_file = "audio_test.mp3"
pcm_file = "audio_bin.pcm"

# mp3 to pcm
y, sr = librosa.load(mp3_file, sr=44100, mono=True)

pcm_data = (y * 32767).astype(np.int16)

pcm_data.tofile(pcm_file)
```

В переменной y хранится массив отсчетов песни, где каждый отсчет принимает значения [-1;1]. sr - частота дискретизации. Файл .pcm хранит амплитуды как целые числа от -32768 до 32767, поэтому отмасштабируем значения, домножив их на 32767.

Из .pcm в .mp3

```
import numpy as np
```

```
import librosa
from pydub import AudioSegment

pcm_file = "audio_bin.pcm"
mp3_file = "audio_from_pcm.mp3"

pcm_data = np.fromfile(pcm_file, dtype=np.int16)

audio = AudioSegment(
    data=pcm_data.tobytes(),
    sample_width=2,       # 2     = 16
    frame_rate=44100,     #
    channels=1            #
)

audio.export(mp3_file, format="mp3", bitrate="192k")
```

Считываем из .pcm файла отсчеты, с помощью функции AudioSegment формируем аудиофайл, а потом сохраняем файл в текущей директории.

Отправка и прием мелодии

Чтение .pcm файла в C++

Чтобы отправить семплы из .pcm файла, нужно для начала считать файл. Для этого напишем функцию

```
int16_t *read_pcm(const char *filename, size_t *sample_count)
{
    FILE *file = fopen(filename, "rb");

    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);
    printf("file_size = %ld\\n", file_size);
    int16_t *samples = (int16_t *)malloc(file_size);

    *sample_count = file_size / sizeof(int16_t);

    size_t sf = fread(samples, sizeof(int16_t), *sample_count,
       file);
```

```
    if (sf == 0){
        printf("file %s empty!", filename);
    }

    fclose(file);

    return samples;
}
```

Передаем в функцию имя .pcm файла и указатель на переменную, в которой потом вернется кол-во считанных семплов. Далее с помощью fseek(file, 0, SEEK_END) перемещаемся в конец файла, а с помощью ftell(file) узнаем текущую позицию в байтах, т.е фактически узнаем размер файла. Далее выделяем память под массив с семплами, вычисляем кол-во семплов и считываем их из файла.

Отправка и прием

```
size_t sample_count = 0;
int16_t *samples = read_pcm(PATH_TO_AUDIO, &sample_count);

for (size_t offset = 0; offset < sample_count; offset += 1920 *
   2)
{
    if(offset + 1920 * 2 >= sample_count)
        break;

    void *tx_buffs[] = {samples + offset};
    fwrite(samples + offset, 2 * rx_mtu * sizeof(int16_t), 1,
       tx_data);
    printf("offset: %d", offset);
    flags = SOAPY_SDR_HAS_TIME;
    int st = SoapySDRDevice_writeStream(sdr, txStream, (const
       void * const*)tx_buffs, tx_mtu, &flags, tx_time,
       timeoutUs);
    if ((size_t)st != tx_mtu)
    {
        printf("TX Failed: %in", st);
    }
```

```
}
```

Переменная samples содержит семплы, считанные из .pcm файла. Далее запускаем цикл, где будем итерироваться по сдвигам от 0 до sample_count с шагом 1920 * 2 (потому I и Q занимают 2 байта). Сдвиг нужен потому, что за раз отправить мелодию мы не можем, поскольку размер отправляемого буфера должен составлять 1920 семплов. Далее сделаем проверку на выход за границы массива, чтобы не возникало ошибок (часть семплов срежется, но на качество это не влияет). Далее формируем tx_buffs как samples + offset, т.е каждый раз будем перемещаться по массиву и отправлять следующий блок данных. Прием данных остался неизменным. После выполнения программы получим семплы мелодии, принятой из радиоканала. Далее конвертируем .pcm файл в .mp3 и наслаждаемся мелодией.

Эксперимент

SDR всех студентов работают на одной частоте, соотвественно они создают помехи друг для друга. Возьмем разные мелодии, поставим SDR рядом и запустим отправку.

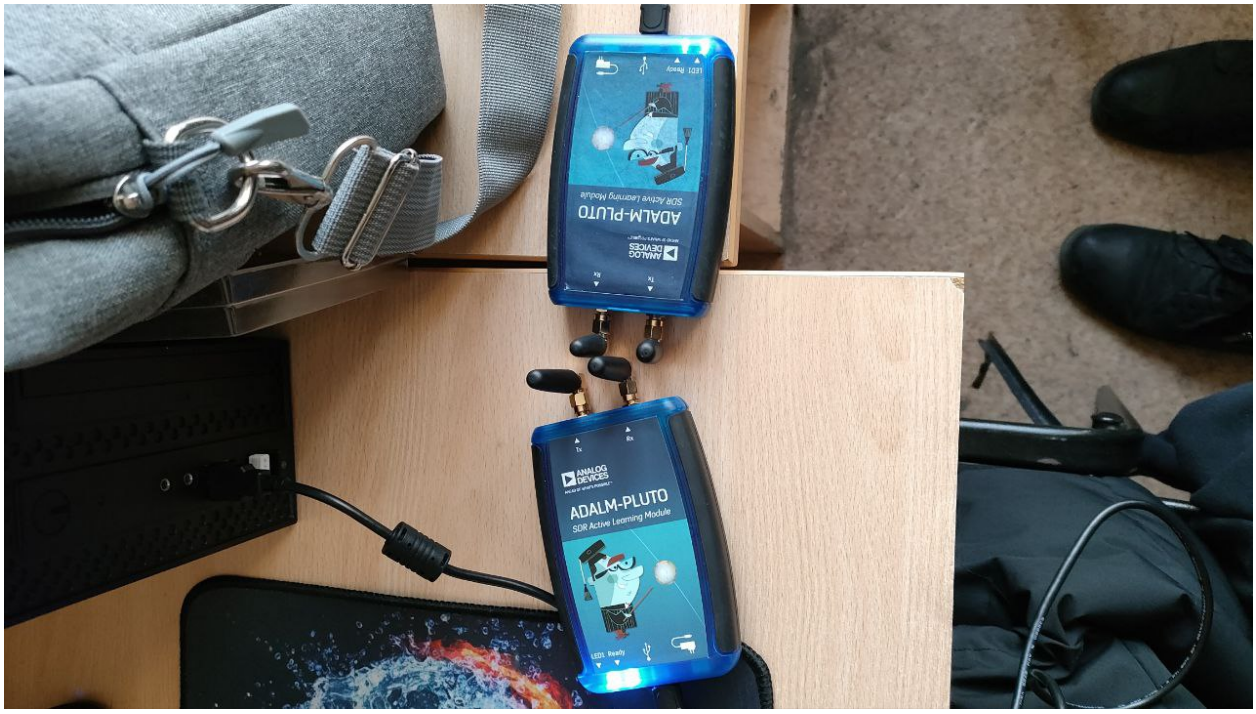

Рисунок 5 — Эксперимент

Итоговая мелодия одновременно похожа на две мелодии с некоторыми помехами. Этот пример иллюстриурет недостаток систем аналоговой связи - она подвержена интерференции от других сигналов.

Реализация логики формирующего фильтра

В теории был указан корректный метод для формирования длящихся символов I(t) и Q(t) из символов I и Q. Реализуем эту логику на Python:

```
# samples on symbol
L = 1000

# samples
I_upsampling = []
Q_upsampling = []

# upsampling for In-phase component
for x in I_symbols:
    I_upsampling.append(x)
    I_upsampling.extend([0] * (L-1))

# upsampling for Quadrature component
for x in Q_symbols:
    Q_upsampling.append(x)
    Q_upsampling.extend([0] * (L-1))

# set impulse response (rect)
g = [1] * L

s_I = []
s_Q = []

# compute convolution
for n in range(len(I_upsampling)):
    tmp_I = 0
    tmp_Q = 0
    for m in range(L):
        if n - m >= 0:
            tmp_I += I_upsampling[n-m]*g[m]
            tmp_Q+= Q_upsampling[n-m]*g[m]
    s_I.append(tmp_I)
```

```
        s_Q.append(tmp_Q)
```

Считывем и парсим из файла символы I и Q. После этого повышаем их частоу дискретизации путем формирования нового списка символов с добавлением L-1 нулей после каждого. Далее устанавливаем импульсную характеристику сигнала g. Далее вычисляем свертку. В переменных s_I и s_Q находятся символы I(t) и Q(t), растянутые во времени.

Для проверки корректности работы визуализируем полученные результаты. Если всё верно, то графики должны быть идентичны графикам из предыдущего занятия.

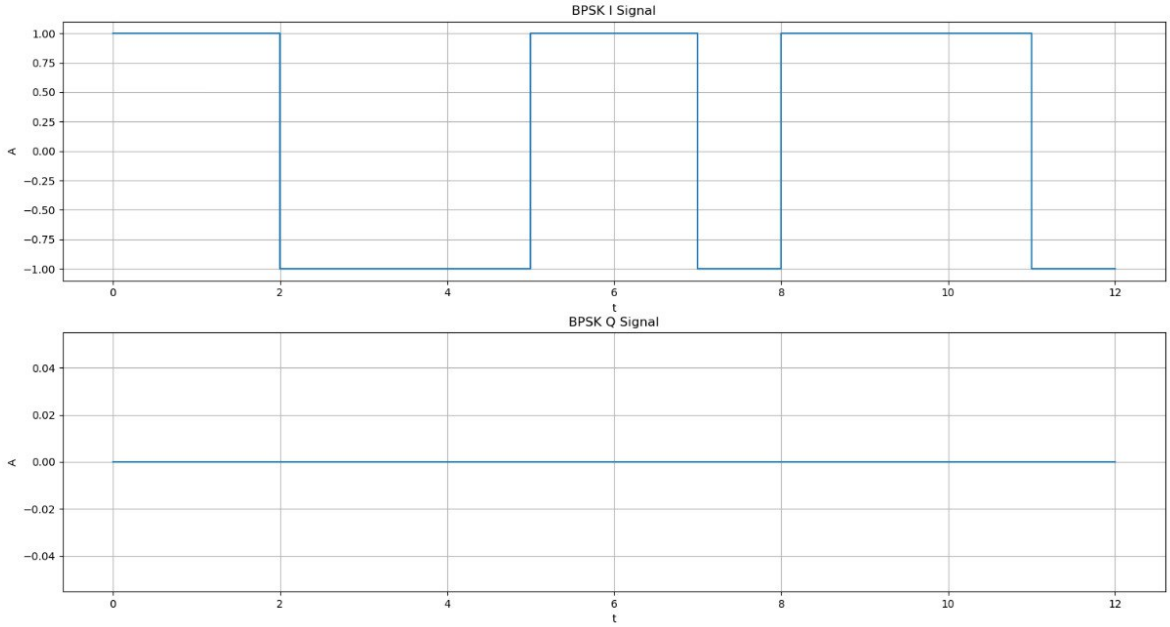

Рисунок 6 — Символы во времени (BPSK)

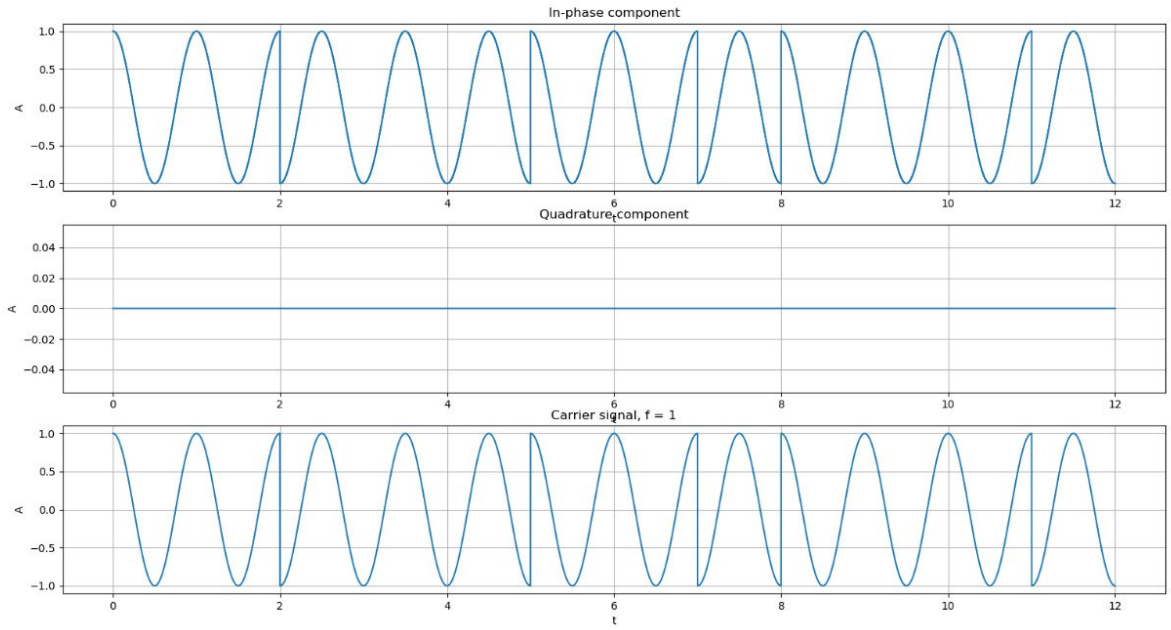

Рисунок 7 — Несущее колебание, перемноженное на символы (BPSK)

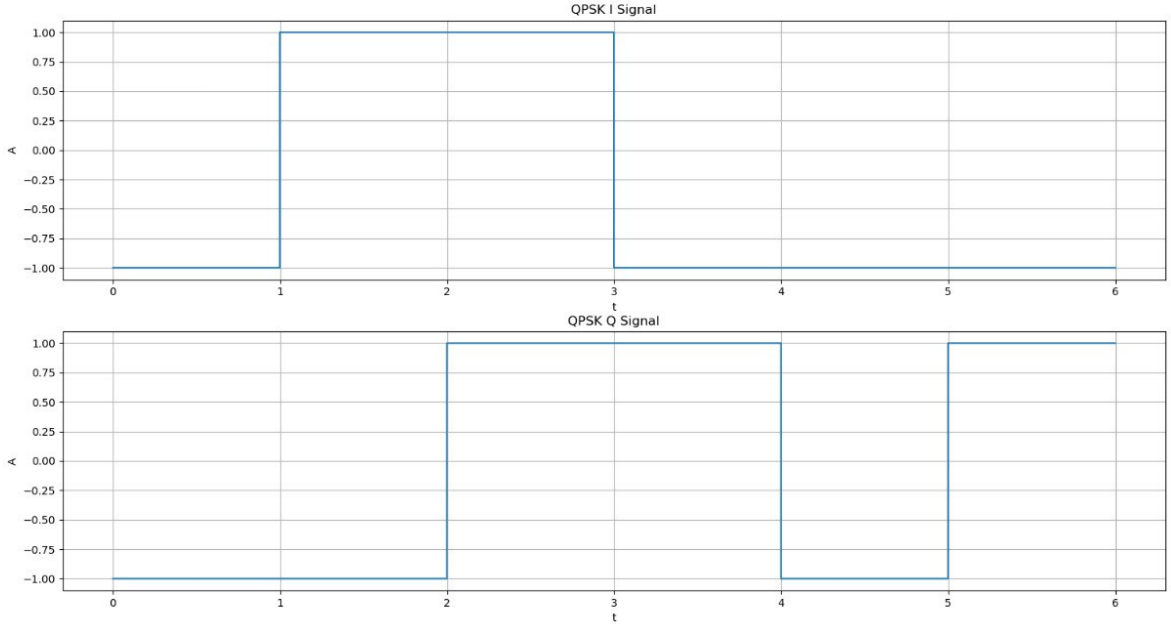

Рисунок 8 — Символы во времени (QPSK)

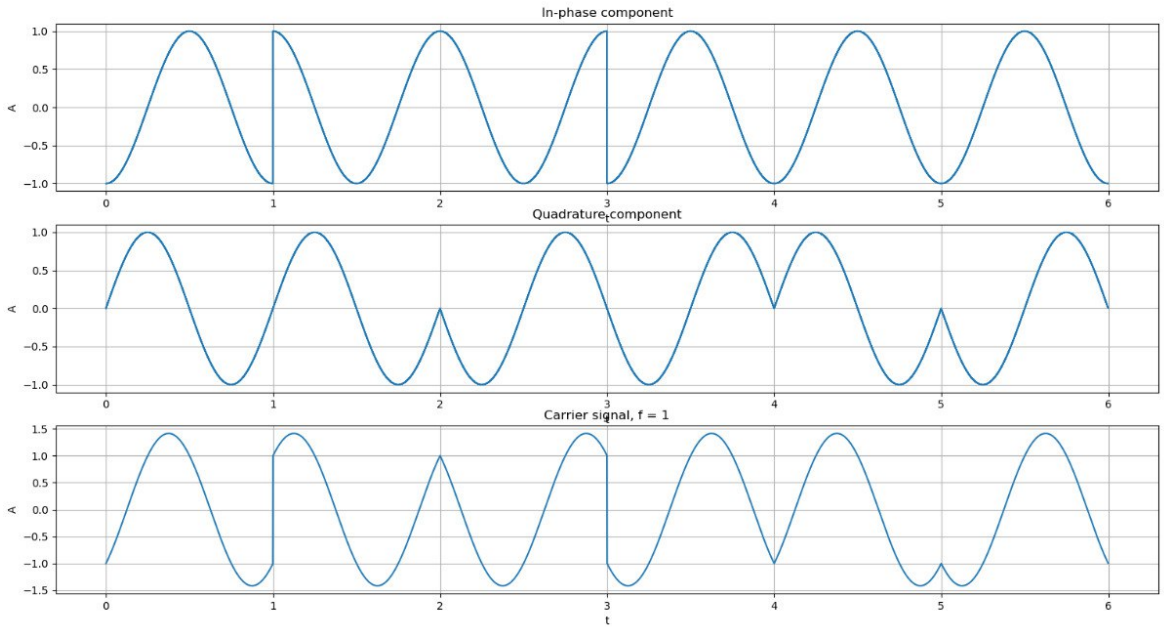

Рисунок 9 — Несущее колебание, перемноженное на символы (QPSK)

Графики идентичны графикам из прошлого занятия, значит, все работает верно. Таким образом теперь мы можем задать сигналу любую форму, изменяя импульсную характеристику g.

ВЫВОД

В ходе проделанной работы я углубился в работу формирующего фильтра и программно реализовал его логику, а также отправил мелодию в радиоканал, принял ее и воспроизвел.