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:
РЕАЛИЗАЦИЯ ПРИЕМА И ПЕРЕДАЧИ BPSK-СИГНАЛОВ

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:

Реализовать на языке C формирование семплов с помощью BPSK модуляции.

Задачи:

1. Сформировать семплы с помощью BPSK модуляции.
2. Отправить семплы в радиоканал, а потом принять их и на их основе построить сигнальное созвездие.

ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Введение

На одном из прошлых занятий мы реализовали программную модель на языке Python, которая формировала сигнал по заданным символам. Мы убедились в правильности построенной модели и теперь построим эту модель на языке C, после чего отправим семплы в радиоканал, примим их и построим сигнальное созвездие.

2.2 Общая структура модели

Рисунок 1 — Модель

В качестве передаваемых данных будет использоваться строка, которую необходимо перевести в набор бит, который поступит на модулятор, на выходе которого получим I,Q символы. Символы поступят на блок upsampling, где после каждого символа добавится L-1 нулей. На выходе блока upsampling получим семплы. Далее необходимо придать нулям значения (на самом деле задается форма сигнала), делается это с помощью формирующего фильтра, в котором при помощи дискретной свертки нулям придается значение в зависимости от импульсной характеристики фильтра. На выходе фильтра получим готовые семплы, которые после масштабирования можно отправлять.

2.3 Перевод строки в биты

Для преобразования строки в нули и единицы на одном из прошлых занятий я написал функцию на языке C:

```
uint8_t* stob(char* str, int* out_bits_count){
    // get string len
```

```
    int len = strlen(str);

    // 1 char = 8 bit
    *out_bits_count = len * sizeof(char) * 8;

    // array for bits
    uint8_t* bits = (uint8_t*)malloc(*out_bits_count *
       sizeof(uint8_t));

    // check pointer
    if (bits == NULL){
        return NULL;
    }

    char c;

    // iterate on string
    for (int i = 0; i < len; i++) {
        // get char
        c = str[i];
        // iterate on bits array
        for (int j = 0; j < 8; j++) {
            // convert char to bits
            bits[i * 8 + j] = (c >> (7 - j)) & 1;
        }
    }

    return bits;
}
```

Пояснения к коду я давал ранее, поэтому сейчас на этом останавливаться не буду.

2.4 Модуляция

BPSK и QPSK я тоже реализовывал ранее, поэтому на подробном объяснении кода останавливаться не буду

```
int* BPSK_modulation(uint8_t* bits, int bits_count, int*
   symbols_count){
    *symbols_count = bits_count * 2;
```

```
    int* IQ_samples = (int*)malloc(sizeof(int) * bits_count * 2);

    int j = 0;
    for(int i = 0; i < bits_count * 2; i+=2){
        if(bits[j]){
            IQ_samples[i] = 1;
            IQ_samples[i + 1] = 0;
        }else{
            IQ_samples[i] = -1;
            IQ_samples[i + 1] = 0;
        }

        ++j;
    }

    *symbols_count = bits_count;

    return IQ_samples;
}
```

```
int* QPSK_modulation(uint8_t* bits, int bits_count, int*
   symbols_count){
    int* IQ_samples = (int*)malloc(sizeof(int) * bits_count);
    *symbols_count = bits_count / 2;
    for(int i = 0; i < bits_count; i+=2){
        if(bits[i]){
            IQ_samples[i] = -1;
            if(bits[i + 1]){
                IQ_samples[i + 1] = -1;
            }else{
                IQ_samples[i + 1] = 1;
            }
        }else{
            IQ_samples[i] = 1;
            if(bits[i + 1]){
                IQ_samples[i + 1] = 1;
            }else{
                IQ_samples[i + 1] = -1;
            }
        }
    }
```

```
    return IQ_samples;
}
```

2.5 upsampling

Функция upsampling должна преобразовать символы I,Q, пришедшие из модулятора, в I(t) и Q(t), т.е растянуть их во времени. Для этого необходимо после каждого символы I,Q добавить L-1 нулей, где L - число семплов, приходящихся на 1 символ. Таким образом мы делаем символы I,Q длящимися во времени.

```
int* upsampling(int* symbols, int symbols_count, int L, int*
   out_size){

    // L - samples on symbols

    //new array size
    *out_size = symbols_count * L;

    //allocate memory
    int* upsampling_symbols = (int*)malloc((symbols_count * L) *
       sizeof(int));

    int cur_pos = 0;

    //iterate on symbols
    for(int i = 0; i < symbols_count; ++i){
        //write original value
        upsampling_symbols[cur_pos++] = symbols[i];

        //insert L-1 zero
        for(int k = 0; k < L-1; ++k){
            upsampling_symbols[cur_pos++] = 0;
        }
    }

    return upsampling_symbols;
}
```

На вход функции подадим массив symbols, который хранит IQ символы в порядке $I_0, Q_0, I_1, Q_1, ..., I_n, Q_n$, и его размер symbols_count. Также подадим L, чтобы знать, сколько нулей добавлять и переменную out_size, которая вернет размер нового массива. Далее выделим память под семплы, размер нового массива будет в L раз больше, чем размер исходного. Чтобы заполнить массив нулями, будем итерироваться в цикле по исходному массиву, добавлять одно значение, а потом запускать еще один цикл, в котором будем добавлять L-1 нулей.

2.6 Формирующий фильтр

Функция формирующего фильтра заключается в том, чтобы придать сигналу форму, определяемую импульсной характеристикой фильтра. В нашем случае импульсная характеристика имеет прямоугольную форму и задана как массив из L единиц.

```
int16_t* ps_filter(int* samples, int samples_count, int L, int*
   g){
    //allocate memory
    int16_t* new_samples = (int16_t*)malloc(samples_count *
       sizeof(int));

    //iterate on samples
    for(int n = 0; n < samples_count; ++n){
        //var for sum
        int tmp = 0;

        //convolution samples and impulse response
        for(int m = 0; m < L; ++m){
            if (n - m >= 0){
                tmp += samples[n-m]*g[m];
            }
        }

    //write value to new array
    new_samples[n] = tmp;
    }

    return new_samples;
```

```
}
```

В функцию передаются семплы после блока upsampling samples и размер этого массива samples_count. Также передается L - кол-во семплов на символ. g - импульсная характеристика. В нашем случае g - массив из L единиц. Далее выделяем память под новый массив. Размер нового массива не изменится. Далее выполняется дискретная свертка, которая придает нашему сигналу форму или придает значение нулям, которые появились после блока upsampling.

2.7 Масштабирование значения семплов под SDR

На данный момент наши семплы - набор из чисел -1, 0, 1. Pluto SDR имеет 12-ти битный АЦП/ЦАП, т.е максимальное значение отправляемых принимаемых семплов находятся в диапазоне $[-2048; 2047]$ (один бит уходит под знак). Помимо этого Pluto SDR странно интерпритирует семплы в переменной int16_t - она считает за семплы только 12 страших бит переменных. Поэтому необходимо делать битовый сдвиг на 4 знака влево («4). Таким образом, чтобы отмасштабировать семплы под Pluto SDR необходимо каждый семпл умножить на (2047«4).

Напишем отдельную функцию, которая будет масштабировать семплы под Pluto SDR.

```
int16_t* scaler(int* samples, int samples_count){

    for(int i = 0; i < samples; ++i){
        samples[i] *= (2047 << 4);
    }

    return samples;

}
```

2.8 Формирование семплов

Это код главной функции, в которой происходит вызов функций, описанных выше, и формируются семплы.

```
    int main(){
    FILE* bpsk_samples = fopen("bpsk_samples.pcm", "w");

    //impulse response
    int g[] = {1,1,1,1,1,1,1,1,1,1};

    //translate char to bits
    int bits_count = 0;
    uint8_t* bits = stob(MESSAGE, &bits_count);

    //translate bits to symbols (I, Q)
    int symbols_count = 0;
    int* symbols = BPSK_modulation(bits, bits_count,
       &symbols_count);

    //translate IQ to upsampling IQ
    int ups_symbols_count = 0;
    int* ups_symbols = upsampling(symbols, symbols_count,
       SAMPLES_ON_BIT, &ups_symbols_count);

    //transalte upsampling IQ to samples
    int samples_count = 0;
    int16_t* samples = ps_filter(ups_symbols, ups_symbols_count,
       SAMPLES_ON_BIT, g, &samples_count);

    //write samples to file
    fwrite(samples, samples_count * sizeof(int16_t), 1,
       bpsk_samples);

    fclose(bpsk_samples);
    return 0;
}
```

После завершения работы программы в текущей директории появится файл ”bpsk_samples.pcm который будет хранить семплы.

2.9 Отправка/прием и анализ результатов

Считаем файл, полученный на прошлом шаге, отправим семплы, примим их и проанализируем. Процесс отправки/приема семплов уже рассматривался ранее, поэтому сейчас не буду это описывать. Принятые семплы запишем в .pcm файл и с помощью программы на Python визуализируем сигнал:

2.9.1 BPSK модуляция

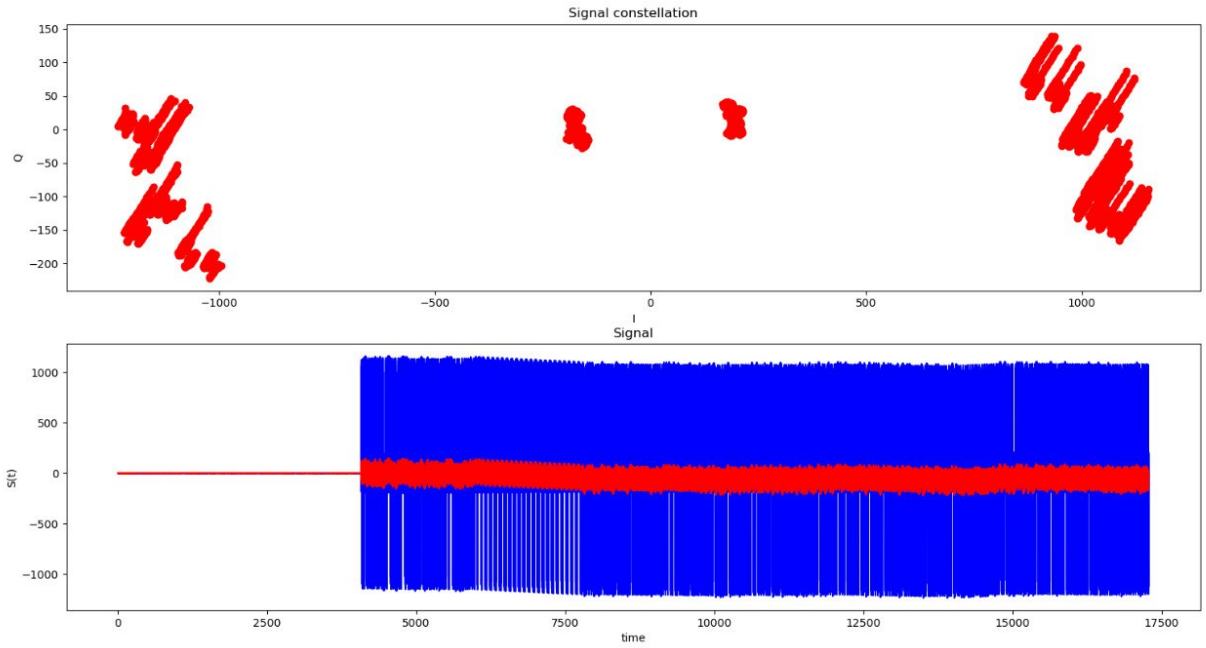

Рисунок 2 — Принятый сигнал и его сигнальное созвездие (BPSK)

На графике сигнала синий цвет I составляющая, красный - Q составляющая. Q в идеале должна быть равна нулю, т.к использовалась BPSK модуляция, но из-за изменения сигнала в процессе передачи Q стало не нулевым. Также можем заметить, что примерно до 4500 семпла вместо сигнала какой-то шум, поэтому при построении сигнальной диаграммы возьмем не все значения, а только те, которые идут после 4500 семплов.

На сигнальном созвездии можем заметить скопление точек по центру. Это скопление напоминает сигнальное созвездие BPSK модуляции, но очень искаженное из-за изменений сигнала в процессе передачи.

2.9.2 QPSK модуляция

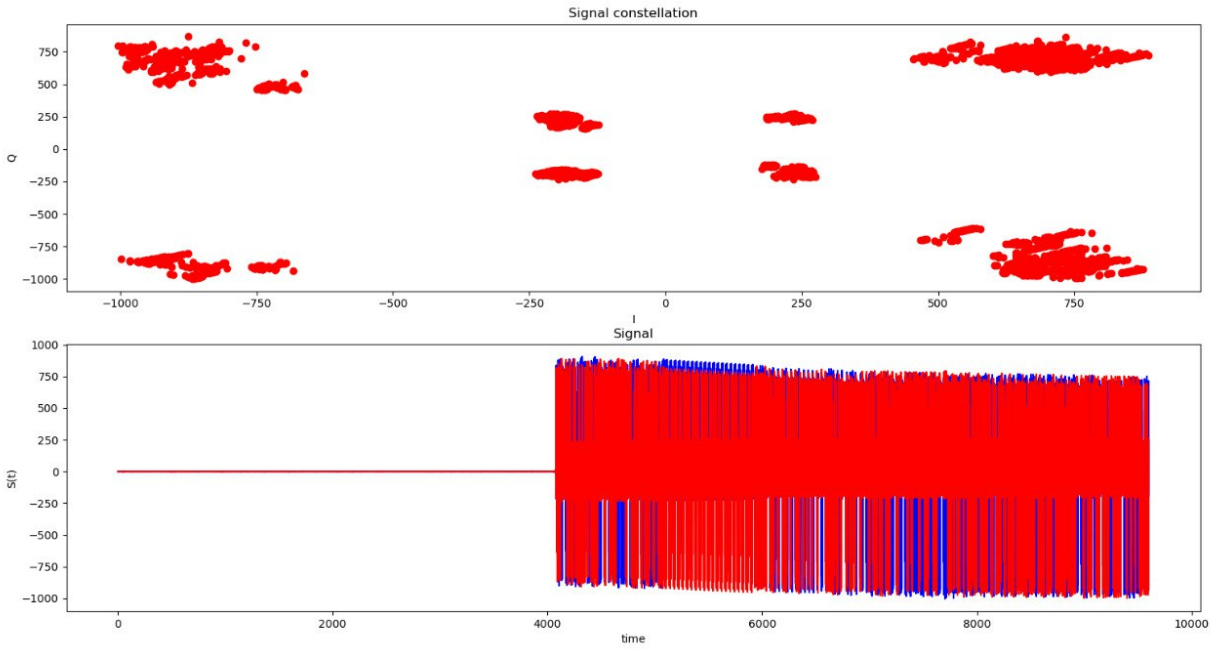

Рисунок 3 — Принятый сигнал и его сигнальное созвездие (QPSK)

На графике сигнала заметно, что Q составляющая примерно равна I, составляющей, что логично, ведь в QPSK модуляции Q составляющая не зануляется, как в BPSK. На сигнальной диаграмме видим 4 скопления точек, что очень похоже на сигнальную диаграмму QPSK модуляции, но очень искаженную.

Python скрипт для визуализации:

```
import numpy as np
import matplotlib.pyplot as plt
import sys

def main():
    #check command line arguments
    if len(sys.argv) != 2:
        print(f"Programm wait 2 command-line argument, but have
           {len(sys.argv)}")

    # get filename
    file = sys.argv[1]
```

```
#samples
imag = []
real = []

count = []
counter = 0

#samples on symbol
L = 10

#open file
with open(file, "rb") as f:
    index = 0
    #read 2L byte (2L bytes = 10 samples = 1 symbol)
    while (bytes := f.read(2 * L)):
        #if byte position is even, then it is I
        if(index % 2 == 0):
            for i in range(0,len(bytes), 2):
                real.append(int.from_bytes(bytes[i:i+2],
                   byteorder='little', signed=True))
                counter += 1
                count.append(counter)
        #if byte position is odd, then it is Q
        else:
            for i in range(0,len(bytes), 2):
                imag.append(int.from_bytes(bytes[i:i+2],
                   byteorder='little', signed=True))
        index += 1


plt.subplot(2, 1, 1)
plt.scatter((real)[5000:],(imag)[5000:],color='red')
plt.xlabel("I")
plt.ylabel("Q")
plt.title("Signal constellation")

plt.subplot(2, 1, 2)
plt.plot(count, real, color='blue')
plt.plot(count, imag, color='red')
plt.xlabel("time")
plt.ylabel("S(t)")
plt.title("Signal")
```

```
    plt.show()

if __name__ == '__main__':
    main()
```

ВЫВОД

В ходе работы я реализовал модель формирования семплов на языке C, произвел отправку/прием семплов, по принятым семплам визуализировал сигнальное созвездие.