МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Кафедра ВС и ТС

Отчет по курсовой работе
по дисциплине
Основы систем мобильной связи

по теме:

ПОСТРОЕНИЕ МОДЕЛИ СИСТЕМЫ СВЯЗИ

Студент:
Группа ИА-331 *Я.А Гмыря*

Предподаватель:
Заведующая кафедрой ТС и ВС *В.Г Дроздова*

Новосибирск 2025 г.

СОДЕРЖАНИЕ

ЦЕЛЬ И ЗАДАЧИ

Цель:

Закрепить и структурировать знания, полученные в рамках изучения дисциплины «Основы систем мобильной связи».

Задачи:

2. Задание и порядок выполнения расчетно-графической работы

1) Введите с клавиатуры ваши имя и фамилию латиницей.

2) Сформируйте битовую последовательность, состоящую из L битов, кодирующих ваши имя и фамилию латинице ASCII-символов. Результат: массив нулей и единиц с данными и разработанный ASCII-кодер. Визуализируйте последовательность на графике.

3) Вычислите CRC длиной M бит для данной последовательности, используя входные данные для своего варианта из работы №5 и добавьте к битовой последовательности. Результат: CRC-генератор и выведенный в терминал CRC.

4) Для того, чтобы приемник смог корректно принимать такой сигнал и находить моменты начала, нужно реализовать синхронизацию. Для этого перед отправкой полученной последовательности добавьте последовательность Голда, которую вы реализовывали в работе №4, длиной G-бит. Результат: функция генерации последовательности Голда и массив с битами данных, CRC и синхронизации. Визуализируйте последовательность на графике.

5) Преобразуйте биты с данными во временные отсчеты сигналов, так чтобы на каждый бит приходилось N-отсчетов. Результат: массив длиной Nx(L+M+G) нулей и единиц – но это уже временные отсчеты сигнала (пример амплитудной модуляции). Визуализируйте последовательность на графике.

6) Создайте нулевой массив длиной 2xNx(L+M+G). Введите с клавиатуры число от 0 до Nx(L+M+G) и в соответствие с введенным значением вставьте в него массив значений из п.5. Результат – массив Signal – визуализируйте на графике.

7) Предположим, что сформированная выше последовательность, промодулировала высокочастотное несущее колебание, передалась через радиоканал и на приемной стороне была оцифрована с заданной частотой дискретизации fs (число отсчетов сигнала в 1 секунде). Проходя через канал отсчеты сигнала исказились (опустим пока историю с затуханием и изменением амплитуды) – к ним добавились значения шумов, присутствовавших в канале, которые можно получить, используя нормальный закон распределения с $\mu=0$ и σ – вводится с клавиатуры (float). То есть нужно сформировать массив с шумом размером 2xNx(L+M+G), реализовав его с помощью нормального распределения, например,

$$noice(\sigma) := rnorm(length(Signal), \mu, \sigma)$$

Затем нужно поэлементно сложить информационный сигнал с полученным шумом. Визуализировать массив отсчетов зашумленного принятого сигнала.

Рисунок 1 — Задание к курсовой работе

8) Реализуйте функцию корреляционного приема и определите, начиная с какого отсчета (семпла) начинается синхросигнал в полученном массиве, удалите лишние биты до этого массива, выведите значение в терминал. Результат: функция корреляционного приемника.
9) Зная длительность в отсчетах N каждого символа, разберите оставшиеся символы. Накапливайте по N отсчетов и сравнивайте их с пороговым значением P (подумайте, какое значение порога следует выбрать, чтобы интерпретировать полученные семплы нулями или единицами). Напишите функцию, которая будет принимать решение по каждым N отсчетам – 0 передавался или 1, на выходе которой должно быть (L+M+G) битов данных. Лишние отсчеты можно отбросить.
10) Удалите из полученного массива G-бит последовательности синхронизации.
11) Проверьте корректность приема бит, посчитав CRC. Выведите в терминал информацию о факте наличия или отсутствия ошибки.
12) Если ошибок в данных нет, то удалит биты CRC и оставшиеся данные подайте на ASCII-декодер, чтобы восстановить посимвольно текст. Выведите результат на экран.
13) Визуализируйте спектр передаваемого и принимаемого (зашумленного) сигналов. Измените длительность символа, уменьшите ее в два раза и увеличьте тоже вдвое. Выведите на одном графике спектры всех трех сигналов (с короткими, средними и длинными символами).
14) Сделайте промежуточные выводы по каждому пункту работы и общее заключение.
15) Оформите работу. Отчет должен содержать титульный лист, содержание, цель и задачи работы, теоретические сведения, исходные данные, этапы выполнения работы, сопровождаемые скриншотами и графиками, демонстрирующими успешность выполнения, и промежуточными выводами, результирующими таблицами и заключение и **ссылка в виде QR-кода на репозиторий с кодом (git)**.

Рисунок 2 — Задание к курсовой работе

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Что такое система связи?

Система связи — это совокупность технических средств, среды передачи и правил, которые позволяют передавать информацию от источника к получателю.

1.2 Схема разрабатываемой модели

В ходе работы необходимо построить модель, имеющую следующую структуру

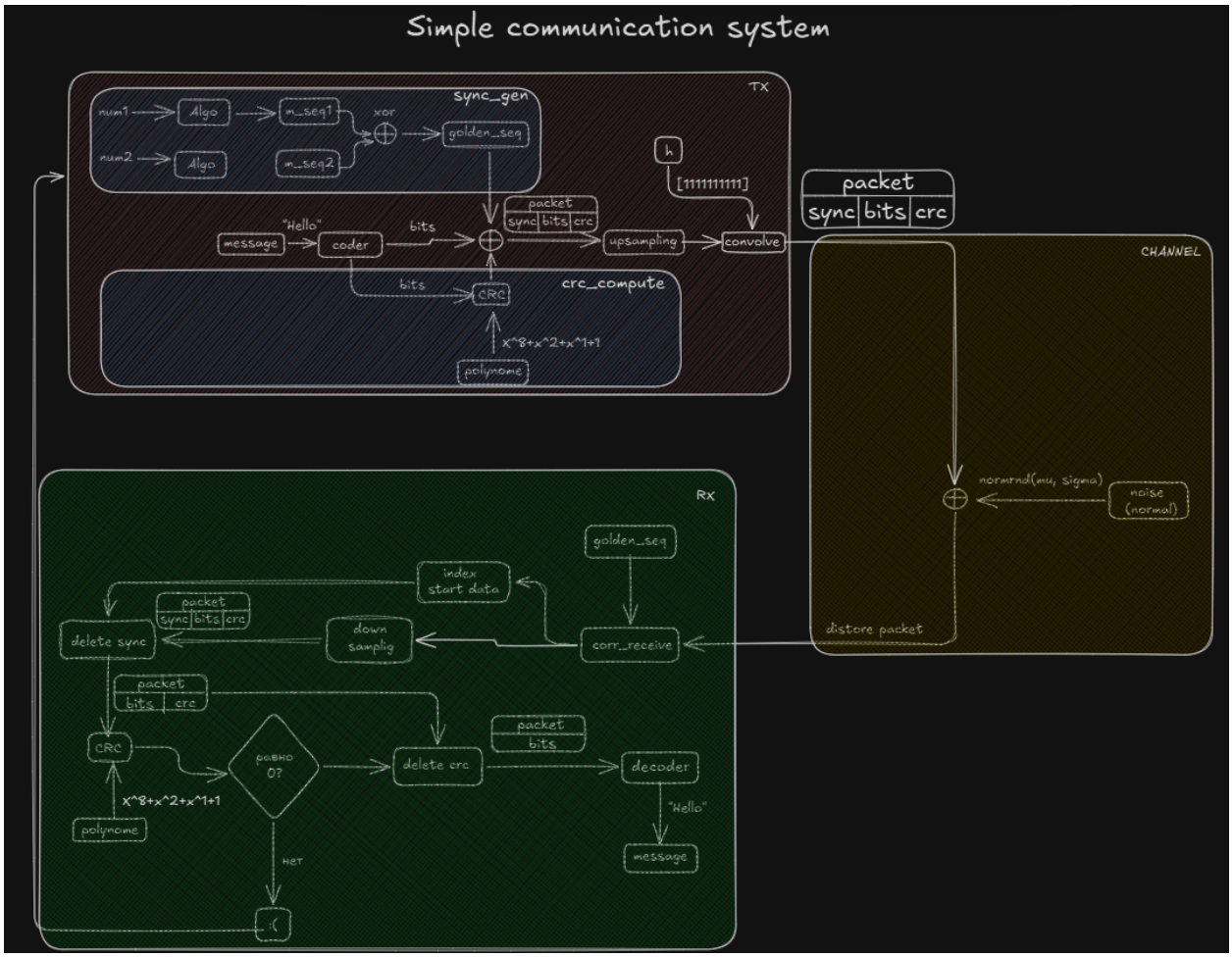

Рисунок 3 — Модель системы связи

1.3 Логика работы системы связи

Модель состоит из 3-ех составных частей: передатчик, канал, приемник. Рассмотрим работу каждого из них подробнее.

1.3.1 Передатчик

1.3.1.1 Блок message

Все начинается с поступления в передатчик инфоормации, которую необходимо передать. Программно это реализовано путем ввода с клавиатуры сообщения.

1.3.1.2 Блок coder

Введенные данные имеют тип данных ”string но нам нужны биты, поэтому нужен блок, который бы переводил символы строки в биты. Такую задачу решает блок coder.

1.3.1.3 Блок sync gen

Для того, чтобы приемная сторона могла определить, с какого момента времени (или с какого семпла) начинаются ”полезные”данные, нужно синхронизировать устройства. Для этого используются специальные битовые последовательности, которые называются синхропоследовательностями. Эти последовательности обладают некоторыми свойствами, позволяющие точно идентифицировать их на приеме и не перепутать с информационной частью пакета или шумами. Одно из главных свойств: свойство корреляции. Оно подразумевает, что автокорреляция такой последовательности со своей сдвинутой копией - очень маленькое число. Имеется только 1 пик, когда сдвиг нулевой.

Примером такой последовательности могут быть м-последовательности. Чтобы сгенерировать такие последовательности, нужно задать начальную любую битовую последовательность длиной M (num1, num2 на схеме) и с помощью специального алгоритма (подробнее о нем в 4 лабораторной работе) из этой последовательности можно получить м-последовательность,

причем размер такой м-последовательности равна $M^2 - 1$.

В нашей системе будут генерироваться две м-последовательности, чтобы получить последовательность Голда. Последовательность тоже относится к синхропоследовательностям, и получить ее можно путем выполнения операции побитового xor между двумя м-последовательностями.

На вход блока подается 2 числа (битовых последовательности длиной M), на выходе получае одну последовательность Голда длиной $2^M - 1$.

1.3.1.4 Блок CRC

На стороне приема важно убедиться, что информационна часть пакета не исказилась при прохождении через канал связи. Для этого используется специальный код, называемый CRC (циклический избыточный код). Данный код рассчитывается по специальному алгоритму, принимающего информационную часть пакета и специальное число, называемое порождающим полиномом (об этом в лабораторной работе №5), и добавляется в конец информационной части пакета. Если для полученной последовательностью снова рассчитать CRC, то получим 0, если информационная часть не повредилась, и число, отличное от нуля, если пакет был поврежден. К изменению CRC при пересчете на приеме может привести любое изменение информационной части пакета.

Расчет CRC находится в блоке CRC. Используется алгоритм CRC-8 с порождающим полиномом $x^8 + x^2 + x^1 + 1$. На выходе получаем CRC код (битовую последовательность).

1.3.1.5 Блок конкатенации

На прошлых трех шагах рассчитывались составные части пакета. Теперь необходимо собрать все это в один пакет. Пакет формируется следующим образом: сначала добавляется синхропоследовательность, потом информационная часть пакета, а затем CRC-код.

Блок принимает 3 битовых последовательности и возвращает одну последовательность, которая является пакетом.

1.3.1.6 Блок upsampling

Для минимизации ошибок на приеме можно каждый элемент пакета продублировать L раз, чтобы на приеме каждый бит представляли 10 значений, из которых опредeнным образом можно сделать вывод о том, была передана единица или передан 0.

Данный блок принимает битовую последовательность и после каждого бита добавляет L-1 нулей, где L-число, на которое мы хотим повторить каждый бит. На выходе получаем последовательность, которая в L раз больше исходной. В работе предполагается, что L = 10.

1.3.1.7 Блок convolve

В реальных системах перед передачей сигнала по каналу связи сигналу придается специальная форма. Форма может повлиять на спектр сигнала, на сложность принятия такого сигнала. Форма придается путем пропускания сигнала через фильтр, где выполняется операция свертки сигнала и импульсной характеристики фильтра.

В разрабатываемой модели форма сигнала нас не интересует, но интересует то, как можно дублировать каждый бит L раз. Для данной задачи фильтр вполне подходит. В модели используется прямоугольная импульсная характеристика, которая представляет собой 10 единиц. Свертка с такой импульсной характеристикой позволит заполнить нули, добавленные в блоке upsampling, таким образом, чтобы каждый оригинальный бит пакета (до upsampling) дублировался L раз.

После данного блока сигнал отправляется по каналу связи.

1.3.2 Канал

1.3.2.1 Блок noise

При прохождении пакета через канал связи неизбежны помехи, которые могут исказить пакет данных. Для имитирования помех создадим выборку чисел из нормального распределения с мат.ожиданием ν равным 0 и дисперсией σ, которое вводится с клавиатуры. σ будет мерой того, насколько сильны помехи, чем больше σ, тем сильнее помехи.

1.3.2.2 Блок sum

В этом блоке происходит суммирование пакета с выборкой из прошлого шага. На выходе получаем искаженный пакет.

1.3.3 Приемник

1.3.3.1 Блок corr receive

Из канала искаженный сигнал попадает на приемник. Приемнику важно понять, в какой момент времени (или с какого семпла) ему нужно начать интерпритировать биты, как пакет. Делается это с помощью корреляционного приема. Во время такого приема приемник рассчитывает корреляцию между принятым сигналом и синхропоследовательностью, которая использовалась при отправке. Такие последовательности заранее оговариваются и закрепляются в стандартах, поэтому приемник знает, какую синхропоследовательность ему ожидать. Исходя из свойств синхропоследовательность, в какой-то момент приемник найдет явный пик корреляции. Для приемника это будет значить, что с семпла номер N начинается синхропоследовательность, а поскольку приемник знает эту синхропоследовательность, то ему не составит труда вычислить семпл, с которого начинается информационная часть пакета.

Блок принимает сигнал из канала и синхропоследовательность (заранее вшита в его память), а на выходе выдает номер семпла, с которого начинается синхропоследовательность.

1.3.3.2 Блок downsampling

Принятые биты не совсем отражают реальные биты, которые отправлял приемник, поскольку каждый бит повторяется L раз. Для верной интерпретации битов необходимо выполнить обратную операцию - downsampling. Эта операция заключается в том, чтобы по 10 семплам понять, какое значение бита передавалось. Я сделал это достаточно просто - я делал выборки по 10 бит, высчитывал среднее значение и сравнивал с пороговым значением P = 0.5 (поскольку это среднее значение). Если среднее больше P, то передавалась единица, в ином случае передавался ноль.

На вход блок получает сигнал (последовательность бит), а на выходе выдает последовательность длиной в L раз меньше.

1.3.3.3 Блок delete sync

Поскольку синхропоследовательность, передаваемая в пакете, больше не нужна, то можно от нее избавиться. Приемник знает номер семпла, с которого она начинается и ее длину, поэтому убрать ее из пакета не составит труда.

На вход блок получает битовую последовательность после downsampling и индекс, с которого начинается синхропоследовательность (вычислялся ранее). На выходе блока получаем пакет без синхропоследовательности.

1.3.3.4 Блок CRC

Премник выделил из сигнала полезные биты, но нет гарантии, что биты не исказились во время прохождения через канал. Для проверки целостности данных необходимо выполнить подсчет CRC на основе сигнала и полинома. Полином тоже заранее оговаривается и закрепляется в стандарте, поэтому полином зашит в память приемника.

На вход блок получает сигнал и полином. На выходе выдает какое-то число.

1.3.3.5 Блок с условием

В этом блоке производится проверка CRC. Если CRC != 0, то пакет повредился при передаче и в этом случае необходимо отправить на передатчик сообщение на повторную отправку. Если CRC == 0, то пакет целый и можно переходить к следующему шагу.

1.3.3.6 Блок delete CRC

Поскольку CRC больше не нужен, можем его удалить из пакета. Приемник знает размер порождающего полинома, который совпадает с размером CRC-кода, также приемник знает, что CRC находится в конце пакета (закреплено в абстрактном стандарте), поэтому применику не составит труда вырезать CRC из пакета.

На вход блоку подается пакет. На выходе он выдает пакет без CRC кода в конце.

1.3.3.7 Блок decoder

В пакете осталась только информационная часть, которую с помощью блока decoder можно преобразовать обратно в строку (передаваемое сообщение).

На вход блок получает битовую последовательность. На выходе он выдает передаваемое сообщение.

ПРАКТИКА

2.1 Программная реализация модели

Модель реализовывалась на языке MATLAB с минимальным использованием встреоенных функций.

2.1.1 Реализация передатчика

2.1.1.1 Блок message

```
%% 1) input your name
name = input("Input your name: ", "s");
```

Для генерации данных пользователю предлагается ввести свое имя. В переменной name будет имя пользователя в виде строки.

2.1.1.2 Блок coder

```
%% 2) translate string to bits
bit_seq = coder(name);

disp("Your name in bits: ")
disp(bit_seq');

figure;
plot(0:1:length(bit_seq)-1, bit_seq);
xlabel("time");
ylabel("bit");
title("bits");
grid on;
```

В этом блоке при помощи функции coder переведем строку в битовую последовательность. После этого выведем последовательность и отобразим на графике.

Реализация функции coder

```
function bit_seq = coder(string)
```

```
    seq_len = length(string) * 8;

    bit_seq = zeros(seq_len, 1);

    for i=1:length(string)
        for k=1:8
            byte = int8(string(i));
            bit_seq((i-1)*8+k) = bitand(bitshift(byte), -(8-k)),
               1);
        end
    end
end
```

На вход функция получает строку, далее выделяется память под битовую последовательность, она будет больше размера строки в 8 раз, потому что в ASCII кодировке на 1 символ приходится 8 бит. После этого начинаем заполнять битовую последовательность: приводим символ (char) в байт (int8), а потом с помощью битового сдвига и операции побитового И извлекаем, начиная с младшего (правого) бита, биты и записываем в выходной массив bit_seq.

2.1.1.3 Блок CRC

```
%% 3) compute CRC
polynome = [1,0,0,0,0,0,1,1,1]; % CRC-8 default polynome
crc = CRC(bit_seq', polynome);

disp("CRC:");
disp(crc');

% add CRC to data
data = [bit_seq',crc'];
disp("data:");
disp(data);
```

В блоке CRC вычисляется CRC-код и выводится на экран, а также добавляется в конец битовой последовательности, полученной на прошлом шаге.

```
function crc = CRC(bits, polynome)
    bits = bits(:);
```

```
    n = length(polynome) - 1;
    bits = [bits; zeros(n,1)];

    for i = 1:length(bits)-n
        if bits(i) == 1
            bits(i:i+n) = xor(bits(i:i+n), polynome(:));
        end
    end

    crc = bits(end-n+1:end);
end
```

Функция принимает битовую последовательность, которую мы хотим передать, и порождающий полином. На выходе получаем CRC-код длиной равной длине полинома.

2.1.1.4 Блок gen_seq

```
%% 4) generate golden seq
seq1 = dec2bin(NUM_IN_JOURNAL, BIT_FORMAT) - '0';
seq2 = dec2bin(NUM_IN_JOURNAL + 7, BIT_FORMAT) - '0';

m_seq1 = m_seq_gen(seq1, 0, 1);
m_seq2 = m_seq_gen(seq2, 0, 3);

golden_seq = xor(m_seq1, m_seq2);

data = [golden_seq',data];

figure;
plot(0:1:length(golden_seq)-1, golden_seq);
xlabel("time");
ylabel("seq");
title("golden seq");
```

Сгенерируем две исходных последовательности seq1 и seq2 на основе чисел, полученных на основе номера в журнале. BIT_FORMAT - битовый формат для последовательности, т.е ее длина. Число 7 в бинарном виде можно записать как 111, а можно как 0000111. BIT_FORMAT в нашем случае равен 8. Далее генерируются две m-последовательности: m_seq1 и m_seq2 с помощью функции m_seq_gen. Далее на основе двух m-последовательностей

генерируем последовательность Голда, выполнив поэлементный xor между ними. Добавим последовательность Голда в начало битовой последовательности, полученной на прошлом шаге. Визуализируем последовательность.

```
function m_seq = m_seq_gen(seq, xor_el1, xor_el2)

    m_seq_len = 2^length(seq) - 1;
    m_seq = zeros(m_seq_len, 1);

    for i=1:m_seq_len
        %get bits for feedback
        xor_bit1 = seq(end-xor_el1);
        xor_bit2 = seq(end-xor_el2);
        %write last bit default seq to result seq
        m_seq(i) = seq(end);
        %shift seq
        seq = circshift(seq, 1);
        %write in head bit from feedback
        seq(1) = xor(xor_bit1, xor_bit2);
    end
end
```

Функция принимает битовую последовательность и индексы элементов, участвующих при формировании бита обратной связи (подробнее об этом в лабораторной работе №4). Далее выделяем память под выходную последовательность, она будет иметь длину $2^M - 1$, где M - длина исходной последовательности. Далее в цикле на каждой итерации выбираем биты для формирования обратной связи, добавляем в выходную последовательность последний бит исходной последовательности, а потом делаем сдвиг исходной последовательности, а в первый элемент исходной последовательности записываем результат операции xor между двумя битами, полученных на прошлых шагах.

2.1.1.5 Блоки upsampling и convolve

```
%% 5) upsampling
samples = upsampling(data, L);

h = [1,1,1,1,1,1,1,1,1,1];
conv = convolve(samples, h);
```

```
figure;
plot(0:1:length(conv)-1, conv);
xlabel("time");
ylabel("sample value");
title("samples");
```

В этом блоке необходимо каждый бит дублировать L раз (L = 10). Для этого используется блок upsampling, который добавляет после каждого бита последовательность из L-1 нулей. Далее симитируем прохождение сигнала через фильтр, выполнив операцию свертки (convolve) между импульсной характеристикой h и семплами после upsampling. Данная операция заполнит нули, добавленные на прошлом шаге, нужным образом. Далее визуализиурем полученную последовательность.

```
function samples = upsampling(bits, N)
    samples = zeros(length(bits) * N, 1);

   for i = 1:length(bits)
        samples((i-1)*N + 1) = bits(i);
   end

end
```

Выделяем память под новую последовательность, она будет в N раз больше. Заполняем последовательность нулями. Далее, начиная с первого элемента начинаем брать каждый 10-ый элемент и записывать значение из оригинального массива.

```
function new_samples = convolve(samples, h)
    new_samples = zeros(length(samples), 1);
    for k=1:length(samples)
        sum = 0;
        for m=1:length(h)
            if(k-m > 0)
                sum = sum + samples(k-m)*h(m);
            end
        end

        new_samples(k) = sum;
    end
```

```
end
```

Выделяем память под новую последовательность и выпираем операцию дискретной свертки между samples и импульсной характеристикой h.

2.1.1.6 Добавление лишних битов

```
%% 6) add zeros
tx_data = zeros(2*length(conv), 1);
enter_index = str2double(input("Enter index for insert packet:
   ", "s"));

while true

    if enter_index + length(conv) > length(tx_data)
        enter_index = str2double(input("Enter index for insert
           packet: ", "s"));
        continue;
    end

    break;
end

for i = enter_index : enter_index + length(conv) - 1
    tx_data(i) = conv(i-enter_index+1);
end
```

Создадим массив, который вдвое больше, чем массив, полученный на прошлом шаге, и заполним его нулями. Длаее попросим пользователя ввести индекс массива, начиная с которого нужно вставить пакет, который формировался все прошлые шаги. Если индекс введен так, что пакет не влезает в новый массив, то предлагается ввести число снова. После получения числа записываем в новый массив наш массив, который до этого сформировали.

2.1.2 Канал

2.1.2.1 Блок noise

```
%% 7) generate noise
var = str2double(input("Enter normal distribution var: ", "s"));
```

```
mu = 0;
noise = normrnd(mu, var, length(tx_data), 1);

% add noise to signal
rx_samples = tx_data + noise;

% disp(rx_samples);

figure;
plot(0:1:length(rx_samples)-1, rx_samples);
xlabel("time");
ylabel("sample value");
title("samples");
```

Генерируем выборку из нормального распределения с $\nu = 0$ и σ, введенной с клавиатуры. Далее суммируем массив, полученный на прошлом шаге, и выборку. Выведем полученный сигнал на график.

```
%% 8) corr receive
ups_golden_seq = convolve(upsampling(golden_seq, L), h);

corr_func = rx_corr(rx_samples, ups_golden_seq);

figure;
plot(0:1:length(corr_func)-1, corr_func);
xlabel("time");
ylabel("corr");
title("corr function");

start_sync_seq = start_sync(corr_func);
disp("start sync_seq");
disp(start_sync_seq);
```

Выполним корреляционный прием сигнала. Выведем корреляционную функцию между принятым сигналом и синхропоследовательностью на график. Далее найдем номер семпла, с которого начинается синхропоследовательность.

```
function func = rx_corr(signal, sync_seq)
    signal = signal(:);

    signal_len = length(signal);
    seq_len    = length(sync_seq);
```

```
    func = zeros(signal_len, 1);

    for i = 1:signal_len
        index = mod((i-1 : i+seq_len-2), signal_len) + 1;
        shift_signal = signal(index);
        func(i) = norm_corr(shift_signal, sync_seq);
    end
end
```

Выделяем память под корреляционную функцию. Далее в цикле берем блоки размером, равным размеру синхропоследовательности, и вычисляем корреляцию между таким блоком и синхропоследовательностью.

```
function index = start_sync(signal)
    max_value = -100;
    for i=1:length(signal)
        if signal(i) > max_value
            max_value = signal(i);
            index = i;
        end
    end
end
```

Эта функция перебирает массив и находит индекс максимального элемента.

2.1.2.2 Блок downsampling

```
%% 9) translate samples to bit
bits =
   samples_to_bits(rx_samples(start_sync_seq:start_sync_seq+length(sampl
   L);
```

С помощью функции samples_to_bits делаем downsampling, т.е принимаем решение, какой бит передавался на каждые 10 семплов.

```
function bits = samples_to_bits(samples, sample_per_bit)

    num_bits = floor(length(samples) / sample_per_bit);
    bits = zeros(1, num_bits);

    for k = 1:num_bits
```

```
        start_i = (k-1) * sample_per_bit + 1;
        end_i = k * sample_per_bit;

        bits(k) = mean(samples(start_i:end_i)) > 0.5;
    end
end
```

Выделяем размер память под выходной массив, который будет иметь размер в L раз меньше, чем входной. Далее в цикле берем блоки по 10 элементов из оригинального массива, вычисляем среднее значение и сравниваем с пороговым значением 0.5. Результат сравнения - передаваемый бит.

2.1.2.3 Блок delete sync

```
%% 10) delete sync_seq from packet
bits = bits(length(golden_seq)+1:end);

disp(bits);
```

Чтобы удалить синхропоследовательность из пакета, сделаем срез массива, начиная с length(golden_seq)+1 и до конца. Элемент с индексом length(golden_seq)+1 - элемент, идущий за синхропоследовательностью, и получается, что синхропоследовательность просто обрезается.

2.1.2.4 Блок CRC

```
%% 11) check errors

rx_crc = CRC(bits, polynome);

disp("RX CRC:")
disp(rx_crc);
```

Для проверки ошибок посчитаем CRC для полученного на прошлом шаге пакета и выведем CRC на экран.

2.1.2.5 Блок delete CRC

```
%% 12) delete crc
```

```
bits = bits(1:end-length(rx_crc));

disp("RX data:")
disp(bits);
```

Удаляем CRC с поммощью среза массива. Для этого с конца срезаем M элементов, где M = length(rx_crc). Выведем новый пакет на экран.

2.1.2.6 Блок decoder

```
%% 13) bits to char

rx_str = bin2str(bits);

disp(rx_str);
```

С помощью функции bit2str преобразуем полученные биты в строку и выведем строку на экран.

```
function str = bin2str(bits)
    if mod(length(bits), 8) ~= 0
        disp("Length must % 8");
    end

    str = "";
    for i=1:length(bits)/8
        dec = bin2dec(bits((i-1)*8+1 : i*8));
        str = str + char(dec);
    end
end
```

Проверяем переданную в функцию последовательность на кратность 8, чтобы убедиться в том, что нет лишних или недостающих бит. Далее в цикле берем блоки по 8 бит и переводим биты в десятичную СС с помощью функцию bin2dec. Далее приводим полученное десятичное число к char и добавляем в строку.

```
function dec = bin2dec(bits)
    if length(bits) ~= 8
        disp("Length must be 8");
    end
```

```
    dec = 0;

    for i=1:length(bits)
        dec = dec + bits(9-i)*2^(i-1);
    end
end
```

Проверяем, что битовая последовательность равана 8, чтобы исключить ошибки. Далее в цикле берем справа налево бит и возводим его значение в степень, которая равна индексу элемента (отсчет от нуля).

2.2 Демонстрация работы

2.2.1 Передатчик

2.2.1.1 Блоки message и coder

Пользователь вводит сообщение, а кодер переводит сообщение в биты

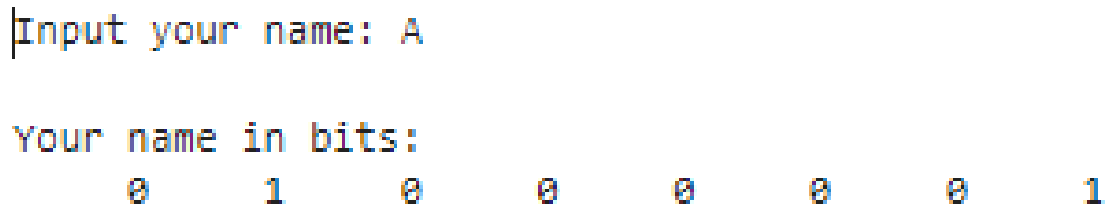

Рисунок 4 — Блоки message и coder

Визуализируем полученную битовую последовательность

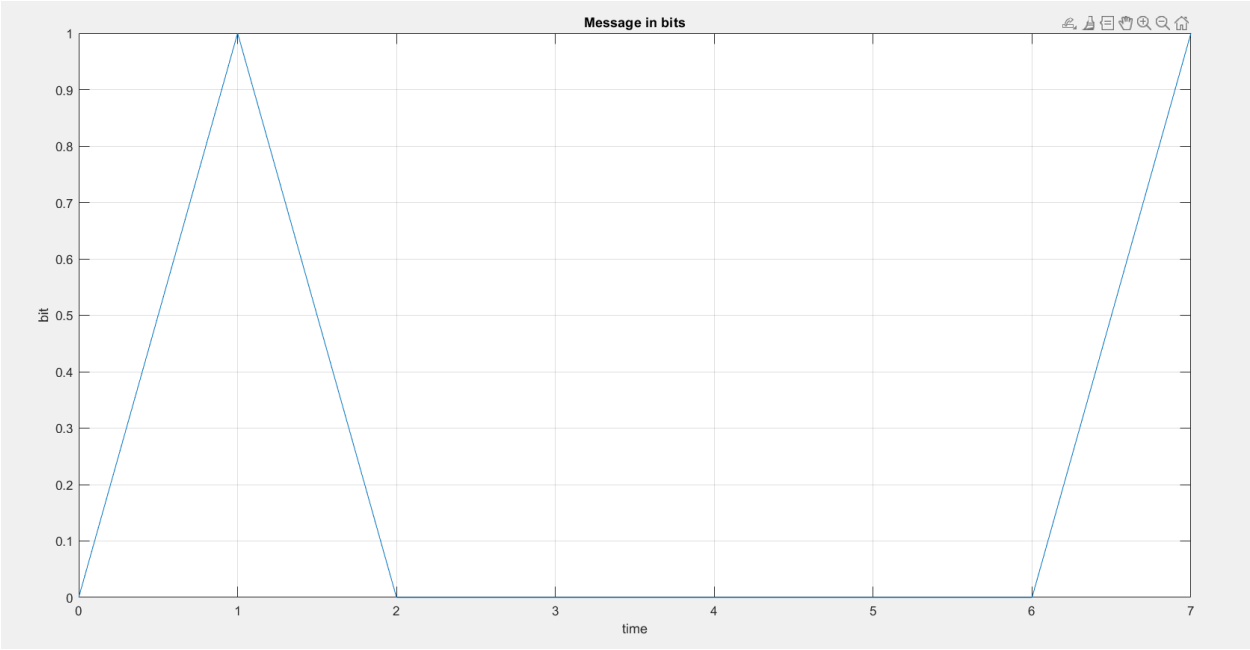

Рисунок 5 — График битовой последовательности

2.2.1.2 Блоки CRC и sync_gen

Генерируем CRC-код и синхропоследовательность. Выводим их на экран

```
CRC:
     1     1     0     0     0     0     0     0

sync-seq:

    1  1  0  1  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0  1  1  1  1  0  0  1  1  1  1  0  0  0  1  1  0  0  0  0  0  1  0  1  0  0  1

    0  1  1  1  0  0  1  0  1  1  0  1  0  1  0  0  0  0  1  0  0  1  0  1  1  1  1  1  1  0  1  1  0  0  1  1  0  0  0  1  1  0  0  0  1

    0  0  0  1  0  0  1  0  0  1  1  1  1  1  1  0  1  1  0  0  1  0  1  0  1  0  0  1  1  0  0  0  0  1  0  1  0  0  0  0  1  1  0  0  1  0

    0  0  1  0  1  0  1  1  1  1  1  1  0  1  1  0  1  0  0  1  0  1  1  1  1  0  1  0  1  0  0  1  0  0  1  0  0  0  0  0  0  1  1  0  0  0  1

    1  1  0  1  1  1  1  1  1  1  0  0  0  0  0  1  0  0  1  1  1  0  0  1  1  1  1  1  0  0  1  0  1  1  1  1  1  1  1  1  1  0  1  0  1  1  0

    1  0  0  1  1  0  1  0  0  1  0  0  0  0  0  0  0  1  0  1  1  0  1  0  1  0  1  0  0  1
```

Рисунок 6 — Блоки CRC и sync_gen

Визуализируем синхропоследовательность

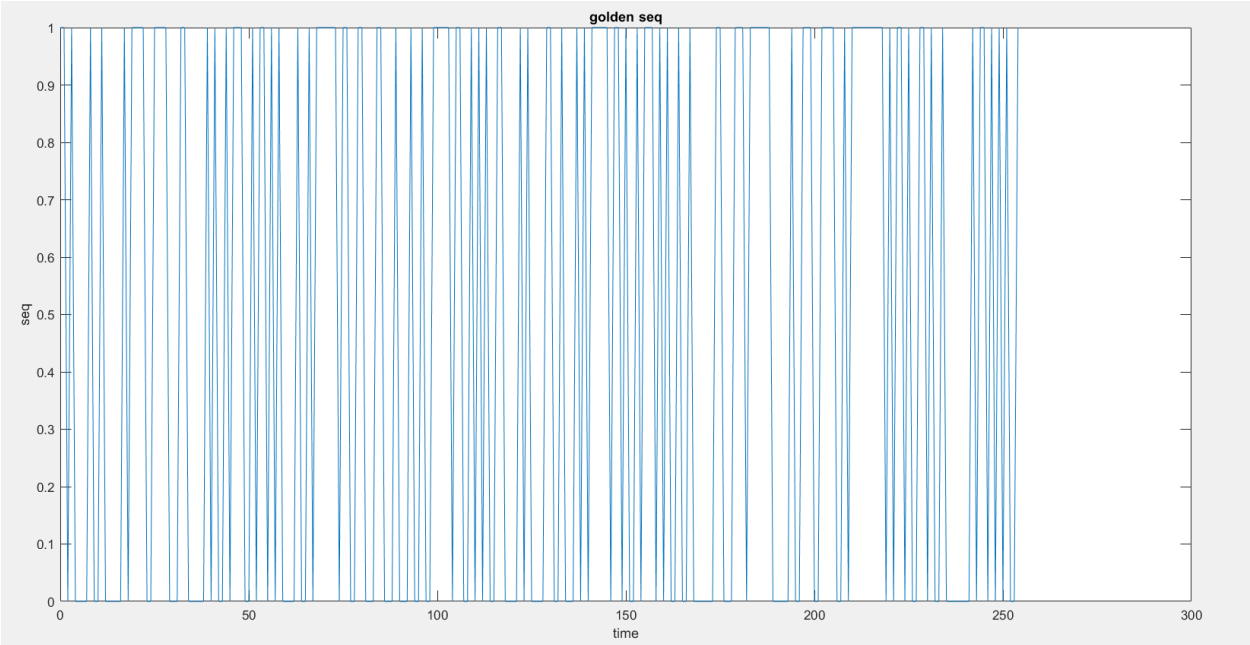

Рисунок 7 — График синхропоследовательности

2.2.1.3 Блоки upsampling и convolve

Дублируем каждый бит L раз и выведем на график

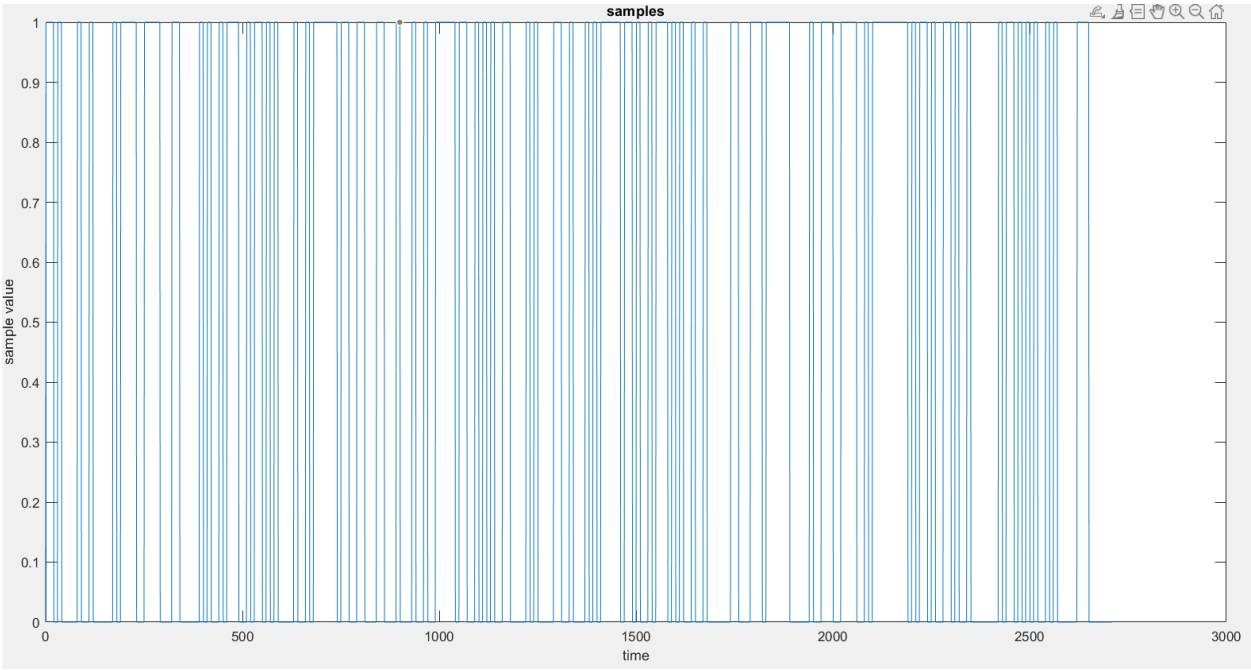

Рисунок 8 — График семплов

Наш пакет изначально состоял из 8 бит информации, 8 бит CRC-кода и 255 бит синхропоследовательности. Итого 271 бит. После upsampling получили 2710 бит, значит, все верно и теперь на кажждый бит приходится L семплов.

2.2.2 Канал

2.2.2.1 Блок noise

Визуализируем шум и сигнал, на который воздействовал этот шум

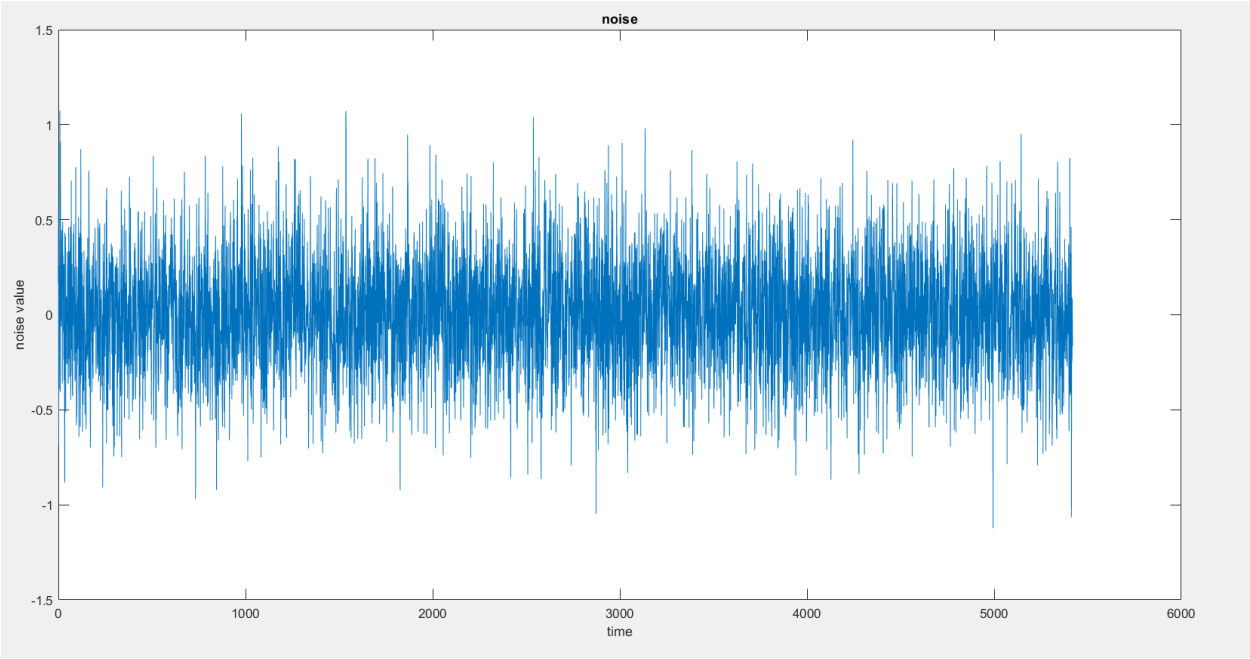

Рисунок 9 — График шума

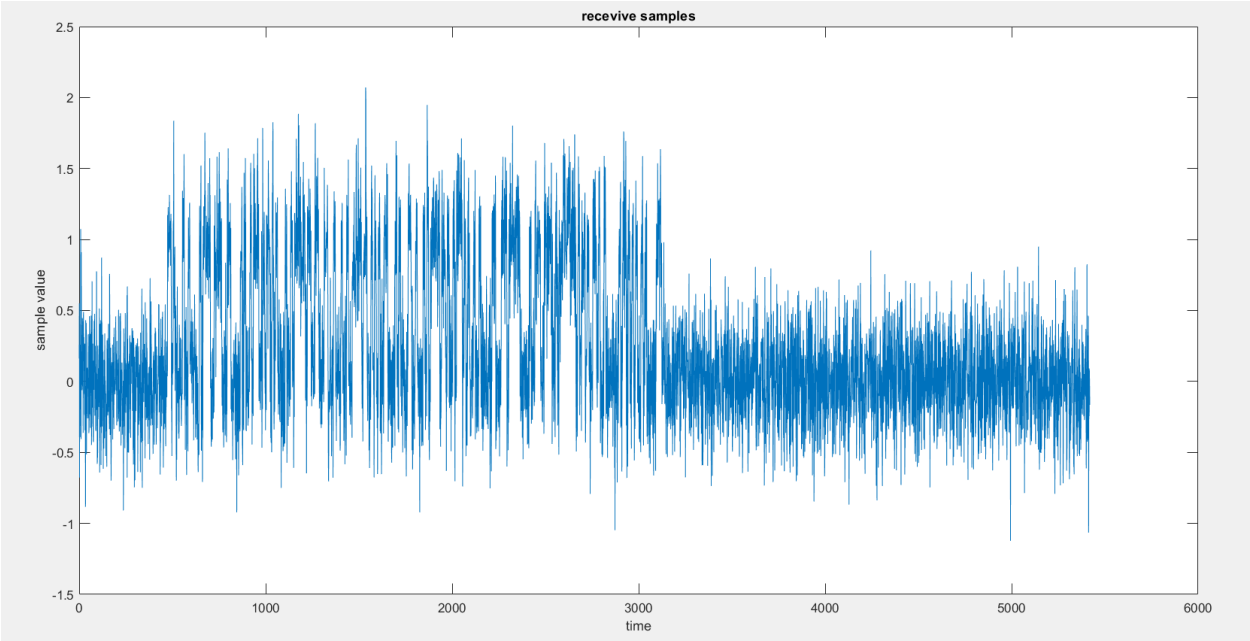

Рисунок 10 — Искаженный сигнал на приеме

Можем заметить, что в какой-то отрезок времени сигнал как бы приподнят. Этот и есть наш пакет, все остальное - нулевой массив, в который мы записали наш пакет.

2.2.3 Приемник

2.2.3.1 Блок corr_receive

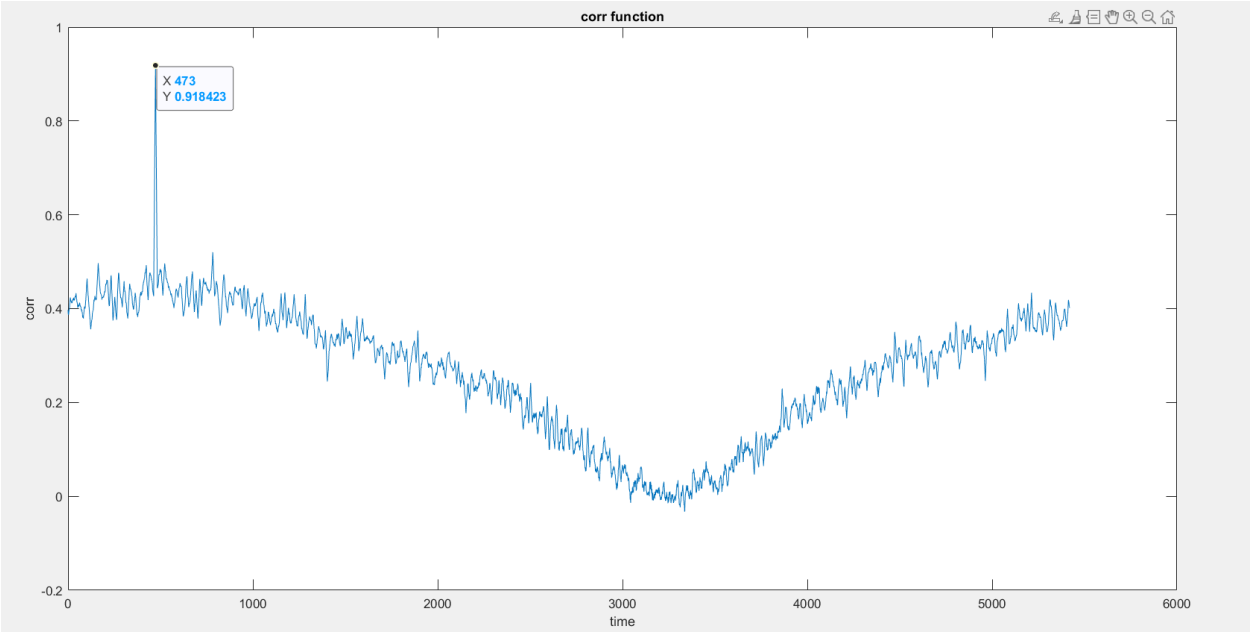

Рисунок 11 — Корреляционная функция

Видим один явный корреляционный пик на 473 сампле, это именно тот номер сампла, с которого мы вставили наш пакет в нулевой массив. Теперь приемник знает, с какого сампла начинается синхропоследовательность и сам пакет.

2.2.3.2 Блок decoder

Выведем на экран информационную часть пакета (подразумевается, что CRC и синхропоследовательность уже удалили) и переведем данные с помощью кодера в строку:

```
RX data:
     0     1     0     0     0     0     0     1

A

```

Рисунок 12 — Декодированная информация

ВЫВОД

В ходе работы я закрепил и структурировал знания, полученные в рамках изучения дисциплины «Основы систем мобильной связи» путем построения программной модели системы связи.