



Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации  
федеральное государственное бюджетное образовательное учреждение высшего образования  
«Сибирский государственный университет телекоммуникаций и информатики»  
(СибГУТИ)

09.03.01 Информатика и вычислительная техника  
(направление подготовки/специальность)

Программное обеспечение мобильных систем  
(профиль/специализация)

Очная  
(форма обучения)

## ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

(вид практики)

### Тип практики

Технологическая (проектно-технологическая) практика на предприятии ООО «Бюро 1440»  
(наименование профильной организации/ структурного подразделения СибГУТИ)

### ТЕМА ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

TODO

#### Выполнил:

студент института информатики и вычислительной техники  
Гмыря Ярослав Александрович  
группа ИА-331

«\_\_\_\_\_» \_\_\_\_\_ 202\_\_\_\_ г.

\_\_\_\_\_ / Гмыря Я.А. /

(подпись) (ФИО)

#### Проверил<sup>1</sup>:

Руководитель практики от профильной организации

«\_\_\_\_\_» \_\_\_\_\_ 202\_\_\_\_ г.

\_\_\_\_\_ / Андреев А.В. /

(подпись) (ФИО)

#### Проверил:

Руководитель практики от СибГУТИ

«\_\_\_\_\_» \_\_\_\_\_ 202\_\_\_\_ г.

\_\_\_\_\_ / Брагин К.И. /

(подпись) (ФИО)

отметка<sup>2</sup> \_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_ 202\_\_\_\_ г.

Новосибирск 2025

<sup>1</sup> В случае прохождения практики в профильной организации

<sup>2</sup> Заполняется во время промежуточной аттестации

**План-график проведения  
Производственной практики**  
(вид практики)

Гмыря Ярослав Александрович  
Фамилия Имя Отчество студента

Институт ИВТ, курс 3, гр. ИА-331  
Направление: 09.03.01 Информатика и вычислительная техника  
Код – Наименование направления (специальности)

Направленность (профиль): Программное обеспечение мобильных систем  
Место прохождения: ООО «Бюро 1440», Новосибирск  
Адрес практики: 314  
Объем практики: 360 / 10  
Срок практики: **16.09.2025**

Содержание практики:  
Тема индивидуального задания практики: TODO

<b>Наименование видов деятельности практики</b>	<b>Дата (начало–окончание)</b>
Архитектура Adalm Pluto SDR. GNU Radio. Построение радио-приёмника	16.09.2025
Введение в архитектуру SDR-устройств. Знакомство с библиотеками Soapy SDR, Libio для работы с Adalm Pluto SDR. Инициализация SDR-устройства. Работа с буфером: получение цифровых IQ-отсчетов.	23.09.2025
Принципы работы библиотеки Soapy SDR и работы с Adalm Pluto. Работа с библиотеками Soapy SDR, Libio. Формирование и передача с SDR сигналов произвольной формы	30.09.2025
Архитектура SDR-устройств. Примеры формирования I/Q-сэмплов произвольной формы. Работа с буфером приема SDR	7.10.2025
Прием сигналов с фазовой модуляцией BPSK/QPSK. Имитация аналоговой передачи звука и его прием с использованием SDR. Анализ влияния чувствительности приемника и усиления передатчика на качество принятых отсчетов сигнала (сэмплов)	14.10.2025
Имитация аналоговой передачи звука и его прием с использованием SDR. Анализ влияния чувствительности приемника и усиления передатчика на качество принятых отсчетов сигнала (сэмплов)	21.10.2025
Моделирование формирования и приема QPSK-сигналов. Реализация приема и передачи BPSK-сигналов	28.10.2025
Алгоритм дискретной свертки. Дискретная свертка. Реализация приема и передачи BPSK-символов	11.11.2025
Прием QPSK BPSK, прием на согласованный фильтр, глазковая диаграмма, поиск оптимального отсчетного значения и необходимость символьной синхронизации. Прием и фильтрация сигнала. Прямоугольный и приподнятый косинус	18.11.2025
Символьная синхронизация, детектор временной ошибки, схема Гарднера. Программная реализация детектора временной ошибки (синхронизация приемника и передатчика) на SDR	25.11.2025

В соответствии с рабочей программой практики  
Руководитель практики от профильной организации  
«\_\_\_\_\_» \_\_\_\_\_ 2025 г.

\_\_\_\_\_  
(подпись) \_\_\_\_\_ / Андреев А.В. /  
(ФИО)

Руководитель практики от СибГУТИ  
«\_\_\_\_\_» \_\_\_\_\_ 2025 г.  
\_\_\_\_\_  
(подпись) \_\_\_\_\_ / Брагин К.И. /  
(ФИО)

\* В случае прохождения практики в профильной организации.

## Оглавление

Занятие 1 .....	5
Занятие 2 .....	27
Занятие 3 .....	40
Занятие 4 .....	52
Занятие 5 .....	72
Занятие 6 .....	85
Занятие 7 .....	94
Занятие 8 .....	107
Занятие 9 .....	115
Занятие 10 .....	131

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  
Занятие №1

по теме:  
АРХИТЕКТУРА ADALM PLUTO SDR. GNU RADIO. ПОСТРОЕНИЕ  
РАДИО-ПРИЁМНИКА

Студент:  
*Группа ИА-331*

*Я.А Гмыря*

Предподаватели:  
*Лектор*  
*Семинарист*  
*Семинарист*

*Калачиков А.А*  
*Ахпашев А.В*  
*Попович И.А*

Новосибирск 2025 г.

## СОДЕРЖАНИЕ

1	ВВЕДЕНИЕ .....	4
1.1	Введение .....	4
1.2	Цель .....	4
2	ЛЕКЦИЯ .....	5
2.1	Что такое SDR? .....	5
2.2	Базовая архитектура системы радиосвязи .....	5
2.3	Описание компонентов архитектуры .....	5
2.4	Описание процесса обмена данными .....	5
2.5	Внутренняя архитектура TX .....	6
2.6	Coder .....	7
2.7	Mapper .....	7
2.8	Pulse shaping filter .....	9
3	ПРАКТИКА .....	11
3.1	Adalm Pluto SDR .....	11
3.2	Архитектура Adalm Pluto SDR .....	12
3.3	Описание основных блоков Adalm Pluto .....	13
3.3.1	PA (Power Amplifier) .....	13
3.3.2	LNA (Low Noise Amplifier) .....	13
3.3.3	ADC / DAC .....	13
3.3.4	FIR (Finite Impulse Response) .....	13
3.3.5	Mixer .....	13
3.3.6	Filter .....	13
3.3.7	libiio .....	13
3.3.8	Linux Kernel (ядро Linux) .....	14
3.3.9	Drivers .....	14
3.3.10	Xilinx Zynq .....	14
3.4	GNU Radio .....	14
3.5	Построение схемы в GNU Radio .....	15
3.5.1	Блок options .....	15
3.5.2	Блок variable .....	15

3.5.3	QT GUI Range .....	16
3.5.4	PlutoSDR Source .....	17
3.5.5	Low Pass Filter.....	18
3.5.6	QT GUI Frequency Sink.....	18
3.5.7	QT GUI Time Sink.....	19
3.5.8	WBFM Receive .....	20
3.5.9	Audio Sink .....	20
4	ВЫВОД .....	22

## ВВЕДЕНИЕ

### 1.1 Введение

Эта и дальнейшие работы будут направлены на изучение и применение на практике алгоритмов цифровой обработки сигналов и построению на их основе простой цифровой системы радиосвязи. В роли принимающего и передающего устройства будет использоваться устройство Adalm Pluto SDR, поэтому первое занятие будет посвящено знакомству с Adalm Pluto SDR.

### 1.2 Цель

Познакомиться с общей архитектурой системы радиосвязи, более подробно рассмотреть архитектуру передатчика. Познакомиться с таким классом устройств как SDR. Получить знания о внутренней архитектуре Adalm Pluto SDR. Познакомиться с программой GNU Radio и с помощью ее графического интерфейса построить FM-приемник.

# ЛЕКЦИЯ

## 2.1 Что такое SDR?

**Software-Defined Radio (SDR)** - радиосистема, в которой часть аппаратных компонентов (фильтры, модуляторы и т.п.) реализованы на программном уровне.

## 2.2 Базовая архитектура системы радиосвязи

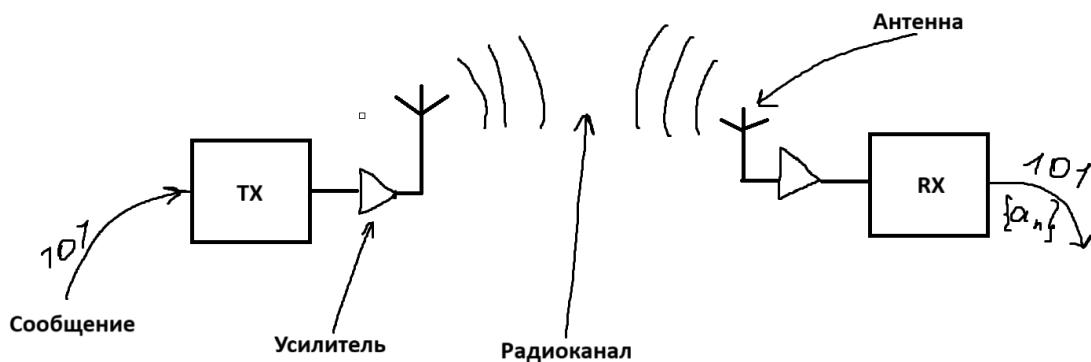


Рисунок 1 — Архитектура системы радиосвязи

## 2.3 Описание компонентов архитектуры

Базовая архитектура состоит из **передатчика (TX)** и **приемника (RX)**. Между ними находится **радиоканал - среда**, в которой распространяется сигнал. У **TX** и **RX** есть **антенна - устройство**, которое излучает или принимает электромагнитные волны, и преобразует их в электрический ток и обратно, в самом простом случае это просто кусок проволоки. Также и у **TX** и у **RX** есть **усилитель**, который усиливает отправляемый/принимаемый сигнал.

## 2.4 Описание процесса обмена данными

На стороне **TX** формируется **сообщение**, которое необходимо передать. Это сообщение поступает в передатчик в виде набора **нулей и единиц**.

**TX** преобразует нули и единицы определенным образом в электрические колебания, которые через **антенну** излучаются в виде электромагнитных колебаний в радиоканал.

В этом же радиоканале находится **приемник**, **антенна** которого принимает эти электромагнитные колебания и преобразует в электрический ток. После этого электрические колебания определенным образом преобразуются в набор нулей и единиц (сообщение, которое отправлял **TX**). Стоит отметить, что **прием сообщения** намного сложнее, чем отправка. Это связано с изменениями, которым подвергается сигнал во время прохождения через **радиоканал**. Сигнал изменяется случайным образом, поэтому точно сказать, как изменится сигнал, мы не можем, мы можем это только предположить с какой-то точностью. Эта проблема решается путем добавления в исходный сигнал **избыточности**, которая позволяет с более высокой точностью принять сигнал на стороне приемника. Такой избыточностью может быть **контрольная сумма - число**, которое вычисляется по определенному алгоритму, который учитывает позицию бита и его значение, т.е если хоть в какой нибудь позиции изменится значение бита, то **контрольная сумма** будет уже другой, что сигнализирует об искажении сигнала.

## 2.5 Внутренняя архитектура TX

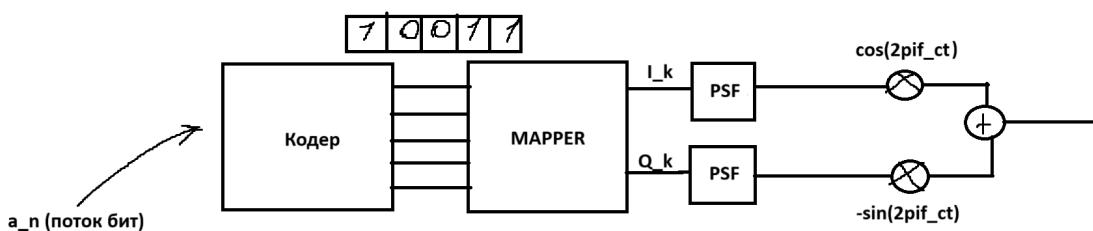


Рисунок 2 — TX архитектура

## 2.6 Coder

В нашем случае кодер будет выполнять единственную задачу - формировать из потока бит блоки (допустим по 8 бит) и направлять их в Mapper.

## 2.7 Mapper

Устройство, выполняющее **отображение исходных данных на множество символов или сигналов** в соответствии с выбранной схемой модуляции.

Символ - элемент сигнального множества.

Сигнальное множество - набор состояний радиосигнала.

Иными словами: mapper берет блок битов (у нас это 8 бит) и сопоставляет его сигналу с определенными характеристиками, для этого в mapper хранится таблица с комбинациями битов и соответствующие им символы. Если в блоке 8 бит, то всего должно быть 256 символов (на каждую возможную комбинацию).

b0	b1	state
0	0	state1
0	1	state2
1	0	state3
1	1	state4

Рисунок 3 — Пример таблицы в маппере

Также для визуализации данного процесса используется созвездие символов

**Созвездие символов** - это **графическое представление множества возможных символов модуляции** в комплексной плоскости. Каждый символ соответствует определённой комбинации параметров сигнала и изображается в виде точки на диаграмме.

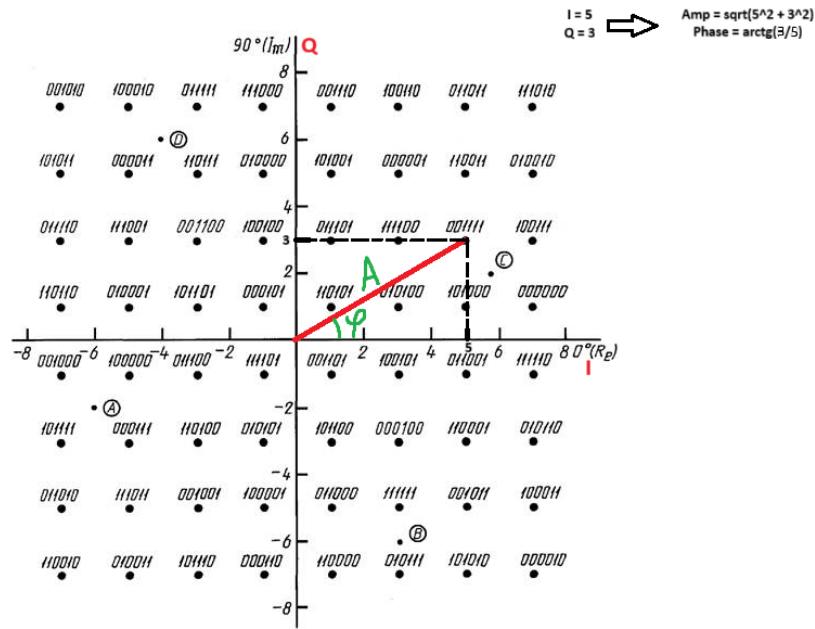


Рисунок 4 — Пример созвездия символов

Каждой точке соответствуют координаты (I, Q), где I - действительная составляющая, Q - мнимая. Зная координаты, можем вычислить длину радиус-вектора до этой точки, это будет амплитудой этого сигнала, угол между действительной осью и радиус-вектором - фаза сигнала.

От mapper идет 2 выхода, один для I составляющей, другой для Q составляющей, которые поступают на вход формирующего фильтра.

## 2.8 Pulse shaping filter

Устройство, преобразующее последовательности символов в непрерывный сигнал с заданной формой. Этот фильтр должен превратить символы (I и Q) в длительные (передаваемые) символы (символы, растянутые во времени с длиной  $T_s$ )

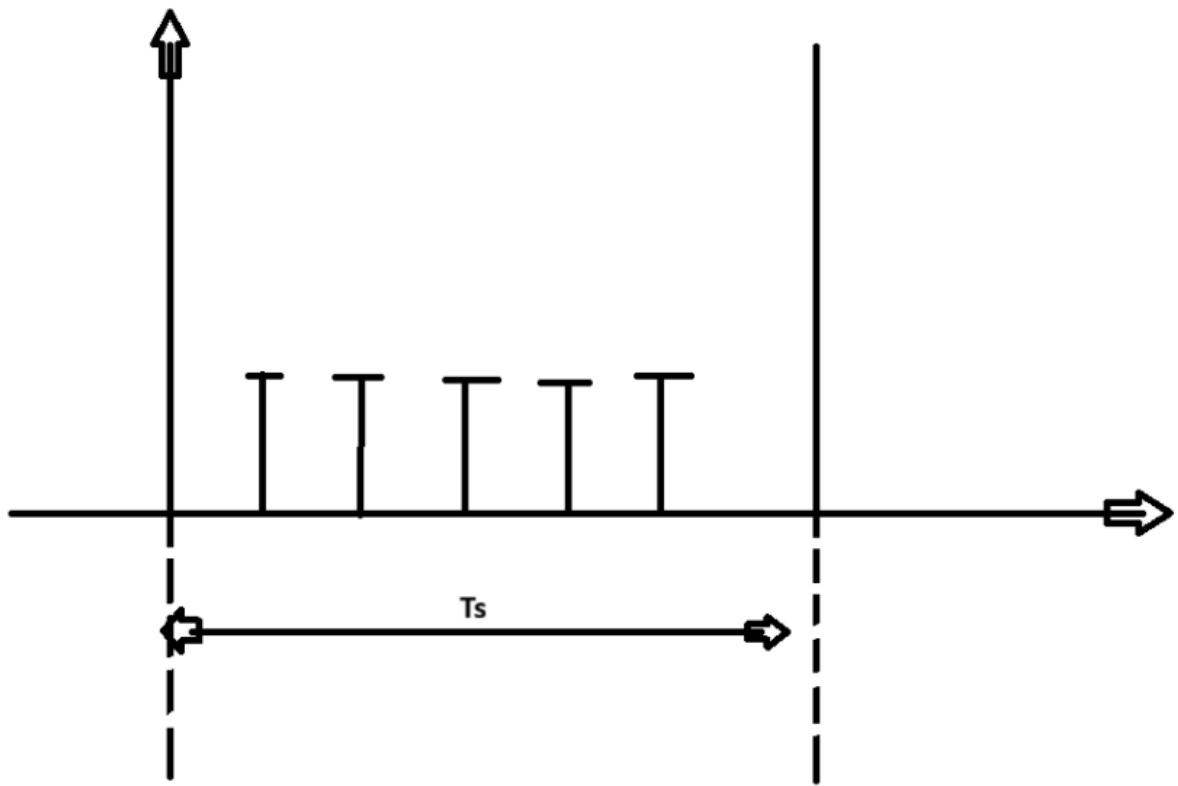


Рисунок 5 — Пример длительного символа

До этого момента все выполнялось программно. Всё, что будет дальше - работа самой SDR.

Далее происходит генерация непрерывного сигнала. Математически этот процесс можно записать следующим образом:

$$s(t) = I(t) \cos(2\pi f_c t) - Q(t) \sin(2\pi f_c t)$$

$f_c$  здесь - несущая частота (высокочастотное колебание).

## ПРАКТИКА

### 3.1 Adalm Pluto SDR

**Adalm Pluto SDR** - модель SDR, разработанная компанией **Analog Devices** для обучения основам SDR.

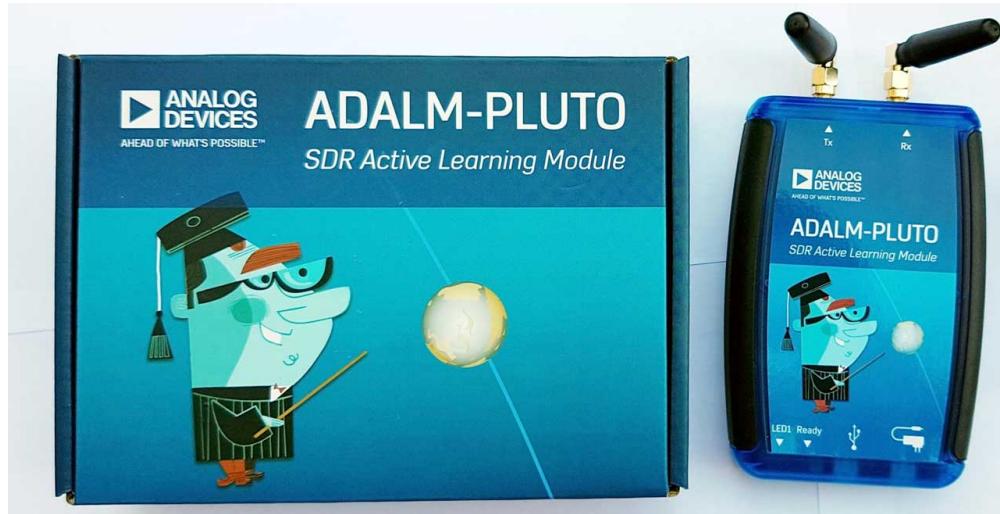


Рисунок 6 — Внешний вид Adalm Pluto

### 3.2 Архитектура Adalm Pluto SDR

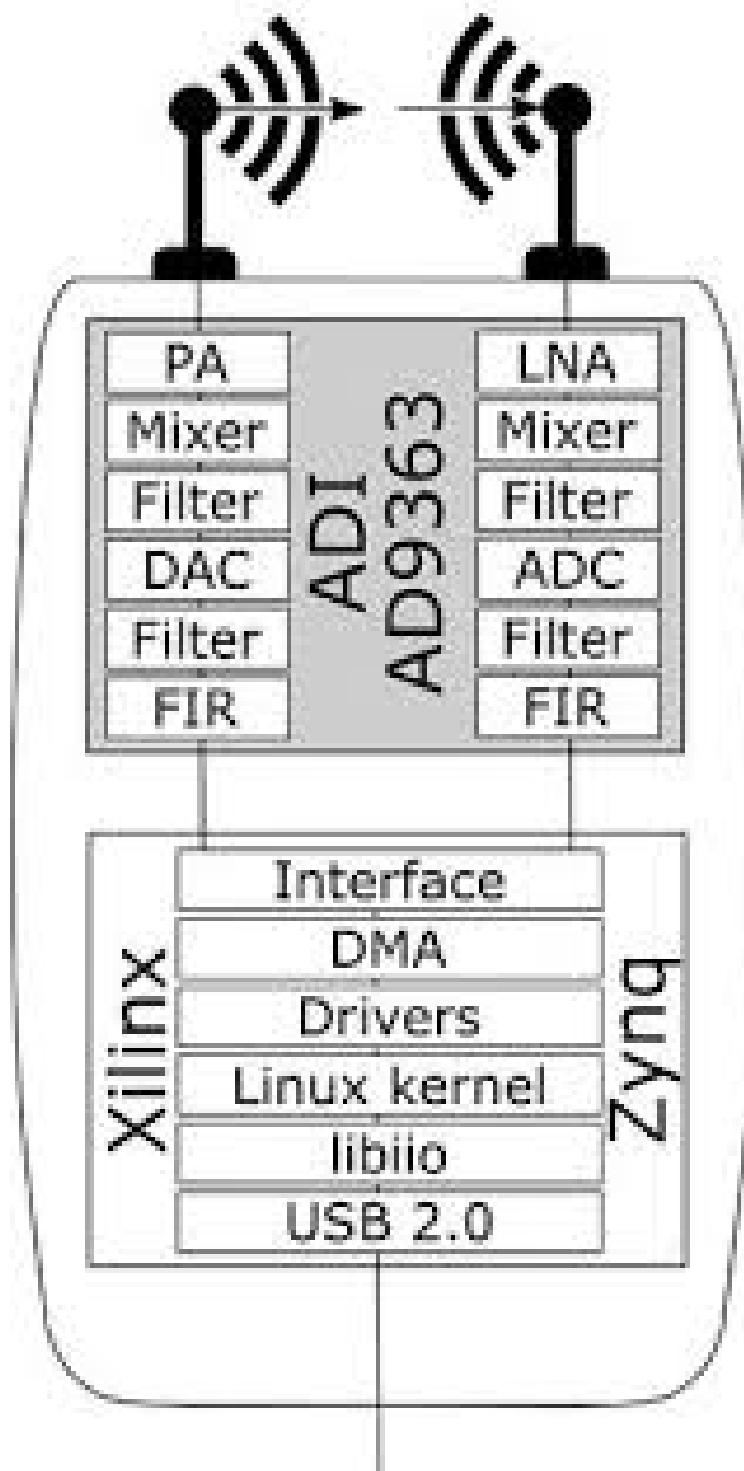


Рисунок 7 — Архитектура Adalm Pluto

### 3.3 Описание основных блоков Adalm Pluto

#### 3.3.1 PA (Power Amplifier)

Назначение: усиление сигнала.

#### 3.3.2 LNA (Low Noise Amplifier)

Назначение: усиление слабого приёмного сигнала с минимальным добавлением шума.

#### 3.3.3 ADC / DAC

ADC (Analog-to-Digital Conversion): оцифровка аналогового сигнала. Получение отсчетов сигнала. DAC (Digital-to-Analog Conversion): преобразование цифровых семплов в аналоговый сигнал.

#### 3.3.4 FIR (Finite Impulse Response)

Назначение: детальная фильтрация, коррекция формы спектра.

#### 3.3.5 Mixer

Перенос низкочастотного сигнала на несущую частоту или высокочастотного на низкочастотный.

#### 3.3.6 Filter

Фильтрация выходного/входного сигнала.

#### 3.3.7 libiio

Библиотека, которая облегчает работу с устройствами ввода/вывода в Linux. Она даёт API для обмена данными и управления устройствами.

### 3.3.8 Linux Kernel (ядро Linux)

Назначение: управляет железом.

### 3.3.9 Drivers

Назначение: обеспечение корректной работы Linux с устройствами.

### 3.3.10 Xilinx Zynq

Семейство микросхем от компании Xilinx. На одном кристалле объединены ARM-процессор, на котором работает Linux, и ПЛИС для более скоростных вычислений.

## 3.4 GNU Radio



Рисунок 8 — GNU Radio

**GNU Radio** - это инструмент с открытым исходным кодом для разработки программного обеспечения в сфере программно-определенного радио.

Он позволяет при помощи «строительных блоков» создавать конфигурации радиоустройств, не написав ни одной строчки кода, и запускать программы непосредственно с использованием SDR-модулей.

В библиотеке имеется широкий спектр функций для цифровой обработки сигналов. Модули написаны на **C++**, а их взаимодействие реализовано на **Python**. Приложения можно строить как через **API GNU Radio**, так и посредством графического интерфейса **GNU Radio Companion (GRC)**.

## 3.5 Построение схемы в GNU Radio

### 3.5.1 Блок options

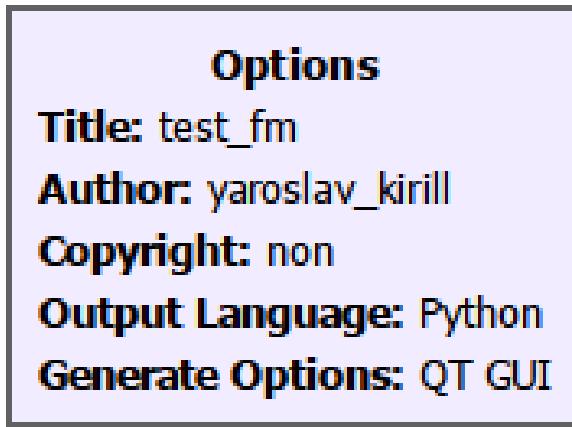


Рисунок 9 — Блок options

Этот блок задает настройки проекта. Самое важное здесь: **Output Language** и **Generate Options**.

**Output Language** — язык, на котором будет сгенерирован код программы.  
**Generate Options** — используемый графический интерфейс.

### 3.5.2 Блок variable

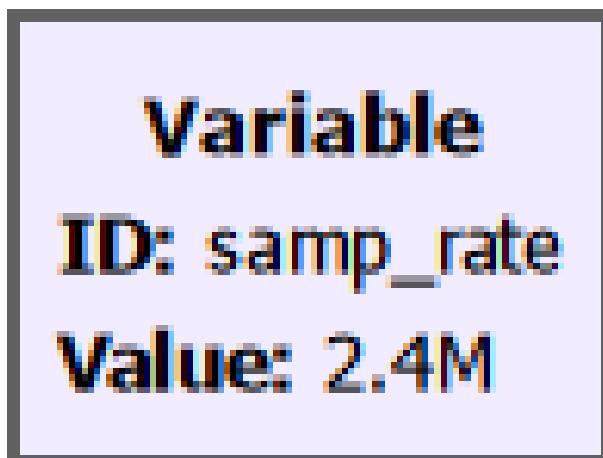


Рисунок 10 — Блок variable

В этом блоке можно задать переменную (почти как в языке программирования). Переменная имеет ID (имя) и значение. Здесь задается samp\_rate (частота дискретизации), равная  $2.4 \times 10^6$  Hz.

### 3.5.3 QT GUI Range

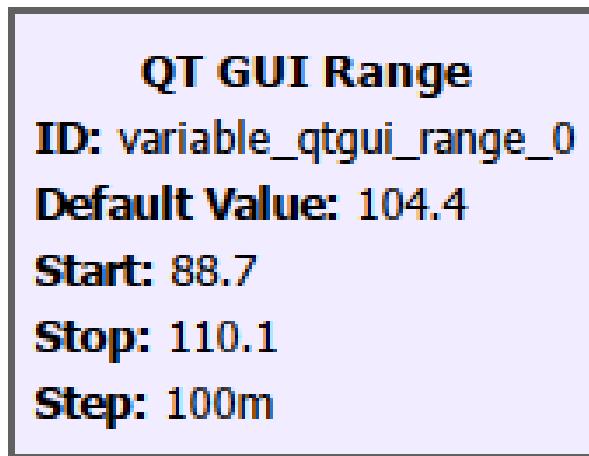


Рисунок 11 — Блок QT GUI Range

Этот блок задает ползунок из QT, позволяющий удобно менять значение переменной во время работы программы. Это позволяет не перезапускать программу, когда нам требуется поменять какое-либо значение. Здесь задается ползунок для настройки частоты приема FM волны.

Основные параметры блока:

- **Default Value** — значение, которое будет устанавливаться при запуске программы;
- **Start** — минимальное значение;
- **Stop** — максимальное значение;
- **Step** — шаг изменения при сдвиге ползунка.

### 3.5.4 PlutoSDR Source

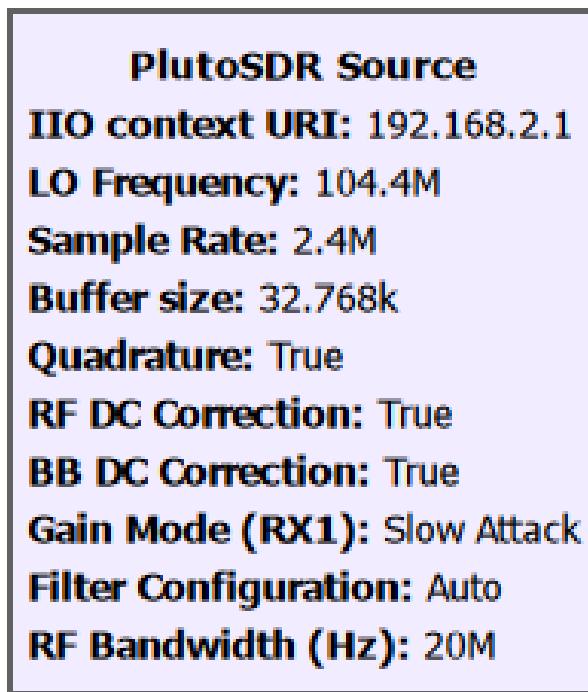


Рисунок 12 — Блок PlutoSDR Source

Этот блок отвечает за приём данных от устройства ADALM-Pluto (PlutoSDR). Он подключается к SDR, управляет настройками, получает поток отсчётов.

#### Параметры:

- **IIO context URI** — IP адрес Adalm Pluto, нужен, потому что PlutoSDR может подключаться по USB или сети (Ethernet/USB-Ethernet);
- **Sample Rate** — частота дискретизации АЦП внутри PlutoSDR. Определяет, с какой частотой будут делаться отсчеты при оцифровке;
- **Buffer Size** — встроенный буфер для временного хранения данных перед их передачей в компьютер;
- **Quadrature** — задаём представление сигнала в виде I/Q семплов;

### 3.5.5 Low Pass Filter



Рисунок 13 — Блок Low Pass Filter

Ограничивает полосу сигнала, выделяя только FM-станцию.

**Параметры:**

- **Decimation** — снижение частоты дискретизации в n раз;
- **Gain** — усиление амплитуды после фильтрации;
- **Sample Rate** — дефолтная частота дискретизации;
- **Cutoff Freq** — полоса 100 кГц (ширина FM сигнала).

### 3.5.6 QT GUI Frequency Sink

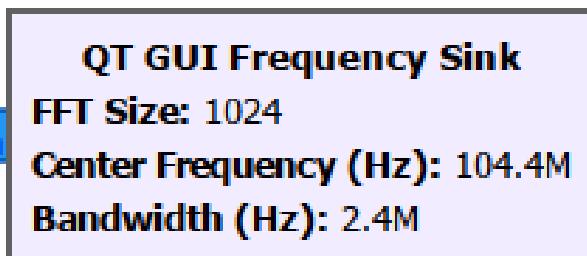


Рисунок 14 — Блок QT GUI Frequency Sink

Этот блок при помощи QT задает спектральное представление сигнала, которое меняется в реальном времени. Таких блоков 2: до фильтра

(напрямую из блока source) и после фильтра. Первый показывает весь эфир, а второй — захваченный сигнал (именно FM частоту).

### Параметры:

- **FFT Size** — кол-во точек для спектра;
- **Center Frequency** — центральная частота захвата;
- **Bandwidth** — полоса частот захвата.

### 3.5.7 QT GUI Time Sink

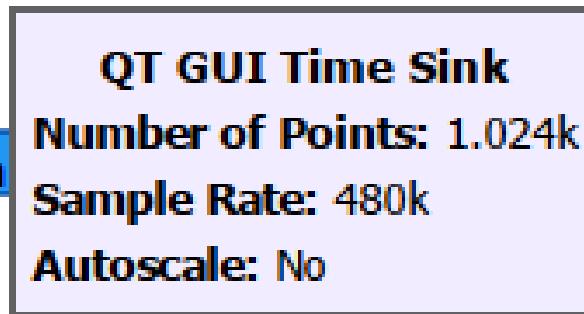


Рисунок 15 — Блок QT GUI Time Sink

Этот блок при помощи QT задает временное представление сигнала, которое меняется в реальном времени.

### Параметры:

- **Number of Points** — кол-во точек, отображаемых в каждый момент времени;
- **Sample Rate** — частота дискретизации при отрисовке;
- **Autoscale** — нужно ли масштабировать сигнал по вертикали.

### 3.5.8 WBFM Receive



Рисунок 16 — Блок WBFM Receive

Блок демодуляции FM-сигнала.

#### Параметры:

- **Quadrature Rate** — входная частота дискретизации (после фильтра и децимации);
- **Audio Decimation** — уменьшение дискретизации для звука (в моем случае до 48к, чего вполне достаточно для звука).

На выходе — звуковой сигнал.

### 3.5.9 Audio Sink



Рисунок 17 — Блок Audio Sink

От **WBFM Receive** звук идет на блок **Audio Sink** — блок, который выводит звуковой поток на аудиокарту хоста.

#### Параметры:

- **Sample Rate** — стандартная частота звука.

Соединить блоки нужно следующим образом, тогда у нас получится работающая радиосистема, которая будет принимать FM радио.

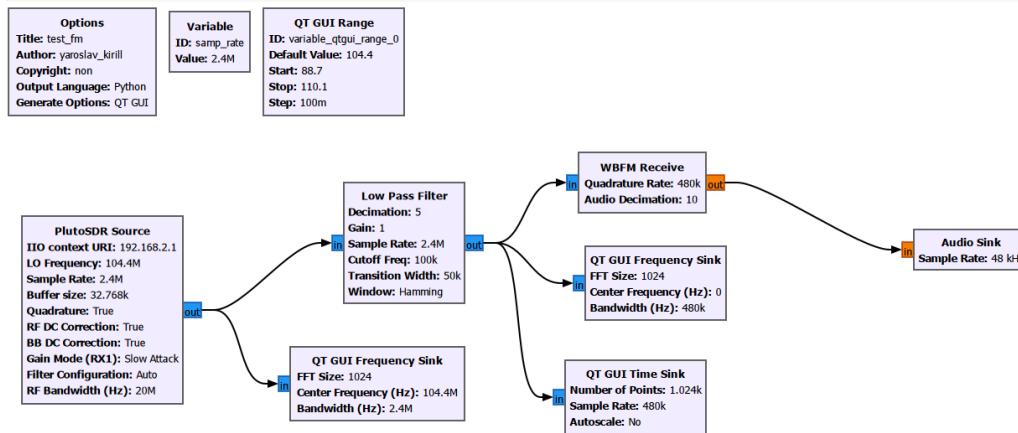


Рисунок 18 — Пример простой радиосистемы в GNURadio

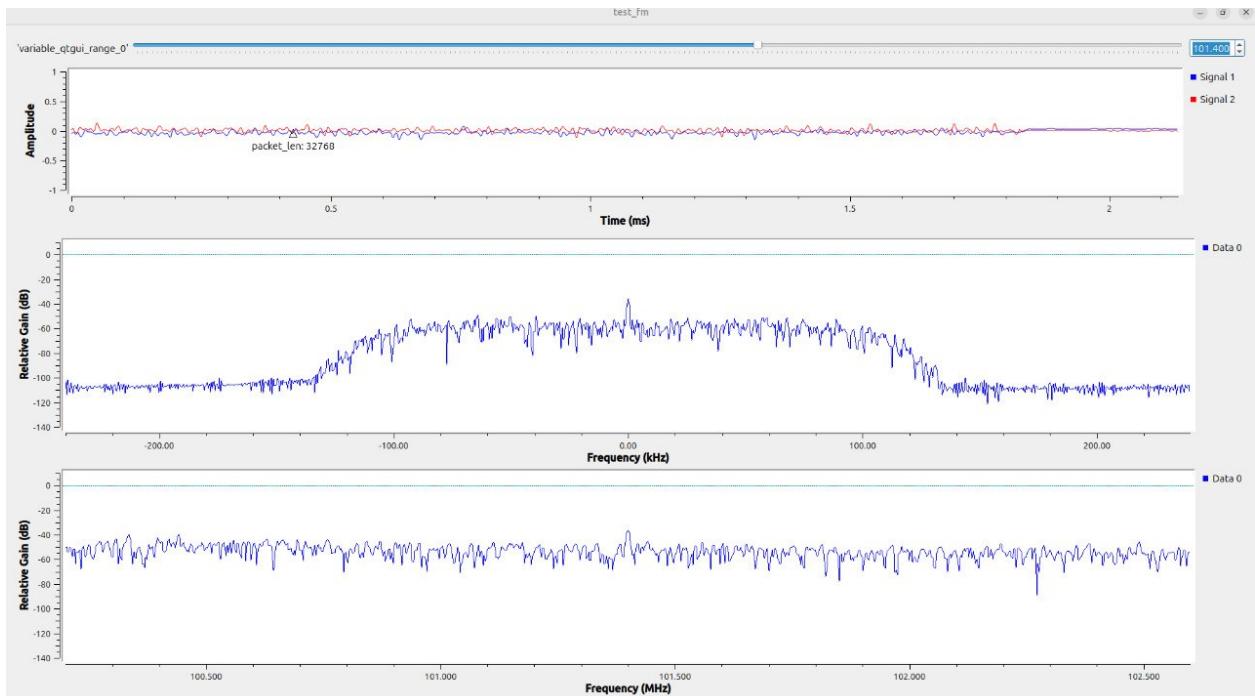


Рисунок 19 — Пример работы программы

## **ВЫВОД**

В ходе проделанной работы я узнал, что такое SDR, изучил принципы его работы и внутреннюю архитектуру на базовом уровне. Познакомился с инструментом GNU Radio и создал с его помощью программу для SDR, позволяющую принимать FM радио.

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  
Занятие №2

по теме:

ВВЕДЕНИЕ В АРХИТЕКТУРУ SDR-УСТРОЙСТВ. ЗНАКОМСТВО С  
БИБЛИОТЕКАМИ SOapy SDR, LIBIO ДЛЯ РАБОТЫ С ADALM PLUTO  
SDR. ИНИЦИАЛИЗАЦИЯ SDR-УСТРОЙСТВА. РАБОТА С БУФЕРОМ:  
ПОЛУЧЕНИЕ ЦИФРОВЫХ IQ-ОТСЧЕТОВ.

Студент:

*Группа ИА-331*

*Я.А Гмыря*

Предподаватели:

*Лектор*

*Калачиков А.А*

*Семинарист*

*Ахпашев А.В*

*Семинарист*

*Попович И.А*

Новосибирск 2025 г.

## СОДЕРЖАНИЕ

1	ВВЕДЕНИЕ .....	3
1.1	Введение .....	3
1.2	Цель .....	3
2	ЛЕКЦИЯ .....	4
2.1	Архитектура простого приемника .....	4
2.2	Принцип работы демодулятора .....	4
3	ПРАКТИЧЕСКАЯ ЧАСТЬ .....	7
3.1	Передача данных между Adalm Pluto и хостом .....	7
3.2	Timestamping .....	7
3.3	Установка необходимых библиотек и зависимостей .....	7
3.3.1	SoapySDR .....	7
3.3.2	Libiio .....	8
3.3.3	LibAD9361 .....	9
3.3.4	SoapyPlutoSDR .....	9
3.4	Основные моменты работы с Adalm Pluto напрямую из C++ .....	10
3.4.1	Подключение библиотек .....	10
3.4.2	Инициализация устройства .....	10
3.4.3	Формирование потоков и буферов .....	10
3.4.4	Получение I/Q семплов .....	11
3.4.5	Освобождение памяти .....	11
3.5	Результат работы .....	12
4	ВЫВОД .....	13

## ВВЕДЕНИЕ

### 1.1 Введение

На прошлом занятии мы работали с SDR с помощью программы GNU Radio, но для более гибкой работы она не очень подходит, поэтому с этого занятие мы начнем работать с SDR с помощью C/C++ и библиотеки SoapySDR. Также на прошлом занятии мы рассмотрели архитектуру передатчика, поэтому на этом занятии познакомимся с архитектурой приемника.

### 1.2 Цель

Познакомиться с упрощенной архитектурой приемника. Научиться работать с SDR напрямую из C/C++ с помощью библиотеки SoapySDR. Отправить в эфир семплы, а потом принять их и проанализировать.

# ЛЕКЦИЯ

## 2.1 Архитектура простого приемника

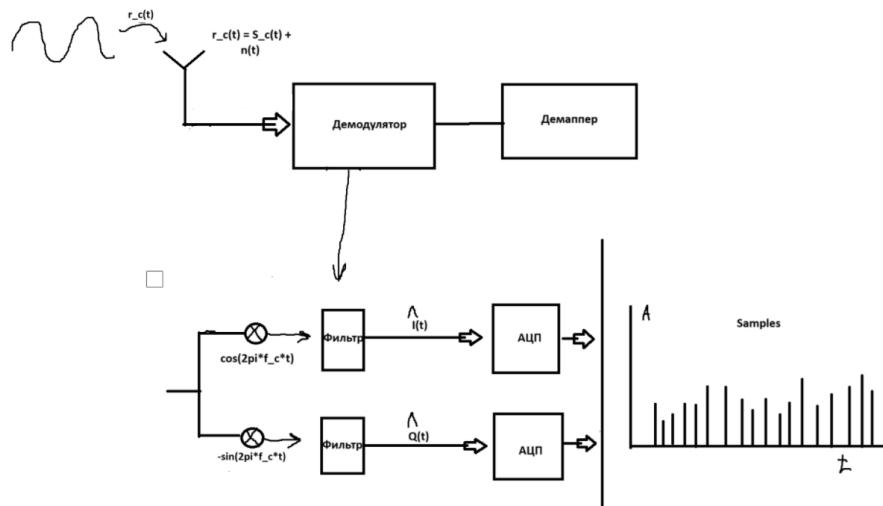


Рисунок 1 — Архитектура простого приемника

Сигнал поступает на антенну приемника и преобразуется в колебание электрического тока. Первым делом нам нужно выделить из этого высокочастотного сигнала сигнал низкой частоты, т.к с низкочастотным сигналом работать проще и вся информацией содержится именно в нем. Для этого подаем сигнал на демодулятор

**Демодуляция** - процесс, обратный модуляции колебаний, выделение информационного (модулирующего) сигнала из модулированного колебания высокой (несущей) частоты. При передаче цифровых сигналов в результате демодуляции получается последовательность символов, передающих исходную информацию ( $I(t)$  и  $Q(t)$ ).

## 2.2 Принцип работы демодулятора

На схеме видим, что поступивший в демодулятор сигнал проходит через цепь, в которой он перемножается на несущие  $\cos(\omega_c t)$  и  $-\sin(\omega_c t)$ . Но как это помогает нам узнать низкочастотный сигнал?

Вспомним из прошлого занятия, какой сигнал по итогу мы отправляли в радиоканал:

$$s(t) = I(t) \cos(2\pi f_c t) - Q(t) \sin(2\pi f_c t)$$

Рисунок 2 — Отправляемый сигнал

Вспомним тригонометрические формулы

### Тема: Произведение синусов и косинусов.

$$\sin \alpha \cdot \sin \beta = \frac{1}{2} (\cos(\alpha - \beta) - \cos(\alpha + \beta))$$

$$\cos \alpha \cdot \cos \beta = \frac{1}{2} (\cos(\alpha - \beta) + \cos(\alpha + \beta))$$

$$\sin \alpha \cdot \cos \beta = \frac{1}{2} (\sin(\alpha - \beta) + \sin(\alpha + \beta))$$

Рисунок 3 — Тригонометрические формулы

Теперь произведем перемножения входного сигнала на несущие:

$$(I(t) \cos(\omega_c t) - Q(t) \sin(\omega_c t)) * \cos(\omega_c t)$$

$$= I(t) \cos(\omega_c t) \cos(\omega_c t) - Q(t) \sin(\omega_c t) \cos(\omega_c t)$$

$$= \frac{I(t)}{2} (\cos(2\omega_c t) + \cos(0)) - \frac{Q(t)}{2} (\sin(2\omega_c t) + \sin(0))$$

$$= \frac{I(t)}{2} \cos(2\omega_c t) - \frac{Q(t)}{2} \sin(2\omega_c t) + \boxed{\frac{I(t)}{2}}$$

У нас получилось выделить синфазную компоненту  $I(t)$ .

Проделаем те же действия с умножением на  $\sin$

$$(I(t) \cos(\omega_c t) - Q(t) \sin(\omega_c t)) \cdot \sin(\omega_c t)$$

$$= I(t) \cos(\omega_c t) \sin(\omega_c t) - Q(t) \sin^2(\omega_c t)$$

$$= \frac{I(t)}{2} \sin(2\omega_c t) - \frac{Q(t)}{2}(1 - \cos(2\omega_c t))$$

$$= \frac{I(t)}{2} \sin(2\omega_c t) + \frac{Q(t)}{2} \cos(2\omega_c t) - \boxed{\frac{Q(t)}{2}}$$

Получили квадратурную компоненту  $Q(t)$ .

Помимо самих компонент остались и другие сигналы, которые нам не нужны, поэтому с помощью фильтра уберем их. На выходе получим чистые  $\frac{I}{2}$  и  $-\frac{Q}{2}$ . Можно заметить, что после извлечения символов их амплитуда упала вдвое. Эта проблема решается путем усиления (на схеме не отображено)

Далее компоненты поступают на АЦП, где будут "нарезаны" на семплаы. В этих семплах нужно произвести символьную синхронизацию, чтобы правильно выделить переданную информацию, но это тема следующих занятий.

## ПРАКТИЧЕСКАЯ ЧАСТЬ

### 3.1 Передача данных между Adalm Pluto и хостом

Передача данных (IQ-сэмплов) между Adalm Pluto и хост-компьютером осуществляется посредством USB 2.0. Важно отметить, что в случае с SDR, данные необходимо передавать непрерывно в обе стороны (с хост-компьютера на SDR и обратно) одновременно. Хоть и теоретическая пропускная способность USB 2.0 равна 480 Mb/s, работа в полудуплексном режиме с передачей данных в обе стороны одновременно (с точки зрения пользователя) разительно снижается. Целевое значение частоты дискретизации желательно задавать в пределах 6 Msps.

### 3.2 Timestamping

В библиотеке SoapySDR реализованы функции получения временных меток (timestamp) с FPGA (Xilinx Zynq). Временные метки (timestamp) привязаны к каждому запросу данных с буфера ПЛИС, что, в свою очередь, позволяет синхронно получать/передавать данные в потоках RX/TX. Более того, из-за проблем с пропускной способностью USB 2.0 возникает проблема увеличения частоты десквантации, при больших значениях которой, USB 2.0 не может обеспечить полноценную передачу и прием (одновременных) сэмплов из Adalm Pluto на хост-компьютер. Выявить данную проблему можно благодаря реализации функции временных меток с Xilinx Zynq.

### 3.3 Установка необходимых библиотек и зависимостей

#### 3.3.1 SoapySDR

SoapySDR — открытая обобщённая API и библиотека времени выполнения для взаимодействия с SDR-устройствами. С помощью SoapySDR можно создавать экземпляры, настраивать и вести потоковую передачу данных с SDR-устройством в различных средах. Большинство готовых SDR-платформ поддерживаются SoapySDR, и многие открытые приложения используют

SoapySDR для интеграции с оборудованием. Кроме того, SoapySDR имеет привязки к средам разработки, таким как GNU Radio и Pothos.

```
sudo apt-get install python3-pip python3-setuptools
sudo apt-get install cmake g++ libpython3-dev python3-numpy swig
    python3-matplotlib

git clone --branch soapy-sdr-0.8.1
    https://github.com/TelecomDep/SoapySDR.git

cd SoapySDR
mkdir build && cd build

cmake ../

make -j 16
sudo make install
sudo ldconfig
```

### 3.3.2 Libiio

libiio — библиотека, разработанная компанией Analog Devices, которая предназначена для упрощения работы с устройствами ввода-вывода данных (I/O), особенно с программируемыми аналогово-цифровыми и цифроаналоговыми преобразователями (ADC/DAC), а также с радиооборудованием на базе платформы ADI (например, ADALM-PLUTO). Позволяет читать и записывать данные в реальном времени.

```
sudo apt-get install libxml2 libxml2-dev bison flex libcdk5-dev
    cmake

sudo apt-get install libusb-1.0-0-dev libaio-dev pkg-config
sudo apt install libavahi-common-dev libavahi-client-dev

git clone --branch v0.24 https://github.com/TelecomDep/libiio.git

cd libiio
mkdir build && cd build
cmake ../
make -j 16
sudo make install
```

### 3.3.3 LibAD9361

LibAD9361 - библиотека для работы с радиочипами семейства AD9361 от Analog Devices. В сочетании с libiio позволяет организовать потоковое чтение/запись данных в реальном времени.

```
git clone --branch v0.3
https://github.com/TelecomDep/libad9361-iio.git
cd libad9361-iio

mkdir build && cd build

cmake ../

make -j 16
sudo make install
sudo ldconfig
```

### 3.3.4 SoapyPlutoSDR

SoapyPlutoSDR - библиотека, которая является расширением библиотеки SoapySDR, предназначенная для работы конкретно с Adalm Pluto.

```
git clone --branch sdr_gadget_timestamping
https://github.com/TelecomDep/SoapyPlutoSDR.git
cd SoapyPlutoSDR

mkdir build && cd build

cmake ../

make -j 16
sudo make install
sudo ldconfig
```

## 3.4 Основне моменты работы с Adalm Pluto напрямую из C++

### 3.4.1 Подключение библиотек

```
// Init device
#include <SoapySDR/Device.h>
// Data types for writing samples
#include <SoapySDR/Formats.h>
```

### 3.4.2 Инициализация устройства

```
//create struct for init
SoapySDRKwargs args = {};

//Select device type
SoapySDRKwargs_set(&args, "driver", "plutosdr");
if (1) {
    // Sample transmission method (usb)
    SoapySDRKwargs_set(&args, "uri", "usb:");
} else {
    // Or IP
    SoapySDRKwargs_set(&args, "uri", "ip:192.168.2.1");
}
SoapySDRKwargs_set(&args, "direct", "1");
// Buffer size and timestamps
SoapySDRKwargs_set(&args, "timestamp_every", "1920");
SoapySDRKwargs_set(&args, "loopback", "0");
// Init
SoapySDRDevice *sdr = SoapySDRDevice_make(&args);
// Free memory
SoapySDRKwargs_clear(&args);
```

### 3.4.3 Формирование потоков и буферов

```
// create streams
SoapySDRStream *rxStream = SoapySDRDevice_setupStream(sdr,
    SOAPY_SDR_RX, SOAPY_SDR_CS16, channels, channel_count, NULL);
SoapySDRStream *txStream = SoapySDRDevice_setupStream(sdr,
    SOAPY_SDR_TX, SOAPY_SDR_CS16, channels, channel_count, NULL);
```

```

//start streaming
SoapySDRDevice_activateStream(sdr, rxStream, 0, 0, 0);
SoapySDRDevice_activateStream(sdr, txStream, 0, 0, 0);

// Get RX/TX MTU sizes

size_t rx_mtu = SoapySDRDevice_getStreamMTU(sdr, rxStream);
size_t tx_mtu = SoapySDRDevice_getStreamMTU(sdr, txStream);

// allocate memory for buffers (for RX/TX samples)
int16_t tx_buff[2 *tx_mtu];
int16_t rx_buffer[2 *rx_mtu];

```

### 3.4.4 Получение I/Q сэмплов

```

// start receive samples
for (size_t buffers_read = 0; buffers_read < iteration_count;
     buffers_read++)
{
    void *rx_buffs[] = {rx_buffer};
    // flags set by receive operation
    int flags;
    //timestamp for receive buffer
    long long timeNs;

    // Read samples from stream and write I/Q samples in file
    int sr = SoapySDRDeviceReadStream(sdr, rxStream, rx_buffs,
                                       rx_mtu, &flags, &timeNs, timeoutUs);
    // write in file
    for(int i = 0; i < rx_mtu * 2; i++){
        fprintf(file, "%d %d\n", rx_buffer[i], rx_buffer[i+1]);
    }
}

```

### 3.4.5 Освобождение памяти

```
//stop streaming
```

```

SoapySDRDevice_deactivateStream(sdr, rxStream, 0, 0);
SoapySDRDevice_deactivateStream(sdr, txStream, 0, 0);

//shutdown the stream
SoapySDRDevice_closeStream(sdr, rxStream);
SoapySDRDevice_closeStream(sdr, txStream);

//cleanup device handle
SoapySDRDevice_unmake(sdr);

```

### 3.5 Результат работы

После работы программы создается файл samples.txt, в котором будут храниться полученные семплы в формате: (I,Q). Для визуализации I(t) и Q(t) воспользуемся Python.

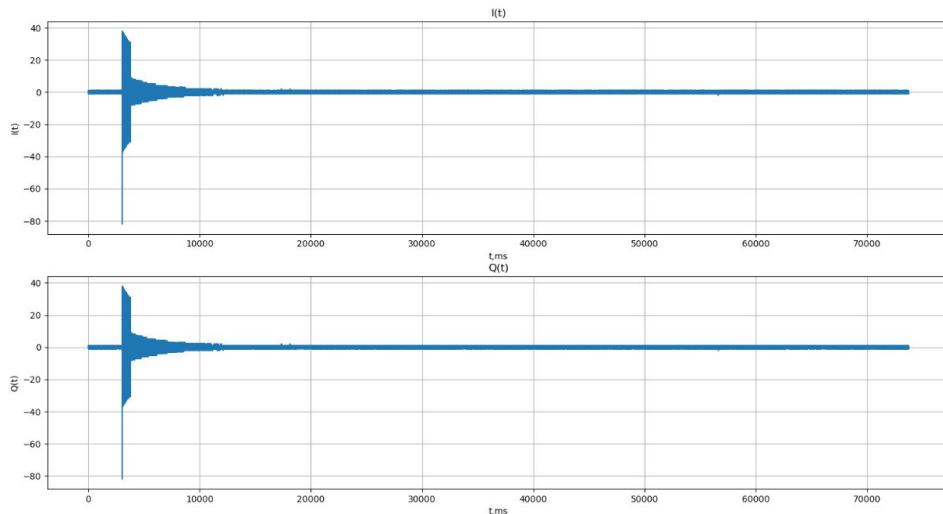


Рисунок 4 — Графики I(t) и Q(t)

Можем заметить, что графики напоминают прямоугольный сигнал.

## ВЫВОД

В ходе проделанной работы я познакомился с архитектурой простого приемника. Научился работать с Adalm Pluto SDR напрямую из C/C++ с помощью библиотеки SoapySDR. Путем визуализации семплов узнал, как выглядит сигнал в виде дискретных отсчетов.

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  
Занятие №3

по теме:

ПРИНЦИПЫ РАБОТЫ БИБЛИОТЕКИ SOAPY SDR И РАБОТЫ С ADALM  
PLUTO. РАБОТА С БИБЛИОТЕКАМИ SOAPY SDR, LIBIO  
ФОРМИРОВАНИЕ И ПЕРЕДАЧА С SDR СИГНАЛОВ ПРОИЗВОЛЬНОЙ  
ФОРМЫ

Студент:

*Группа ИА-331*

*Я.А Гмыря*

Предподаватели:

*Лектор*

*Ахпашев А.В*

*Семинарист*

*Ахпашев А.В*

*Семинарист*

*Попович И.А*

Новосибирск 2025 г.

## СОДЕРЖАНИЕ

1	ВВЕДЕНИЕ .....	3
1.1	Введение .....	3
1.2	Цель .....	3
2	ЛЕКЦИЯ .....	4
2.1	Структура семплов в Pluto SDR .....	4
2.2	Структура буфера семплов в Pluto SDR .....	5
2.3	Запись RX семплов в буффер с Pluto SDR .....	5
2.4	Создание своих семплов и их отправка в Pluto SDR .....	6
3	ПРАКТИКА .....	7
3.1	Формирование семплов .....	7
3.1.1	Перевод строки в бинарный вид .....	7
3.1.2	Способ кодирования .....	8
3.1.3	Заполнение буфера семплами .....	8
3.2	Результат работы .....	10
3.2.1	Семплы до отправки .....	10
3.2.2	Требование к длине сообщения .....	10
3.3	Семплы на приеме .....	10
4	ВЫВОД .....	12

# ВВЕДЕНИЕ

## 1.1 Введение

На прошлом занятии мы отправляли, а потом принимали семплы и визуализировали их, работая с SDR с помощью C/C++ и библиотеки SoapySDR. На этом занятии сформируем семплы произвольной формы, отправим их и визуализируем.

## 1.2 Цель

Лучше освоить работу с SDR с помощью C/C++ и библиотеки SoapySDR, сформировать собственные семплы произвольной формы, отправить их и визуализировать после приема.

# ЛЕКЦИЯ

## 2.1 Структура семплов в Pluto SDR

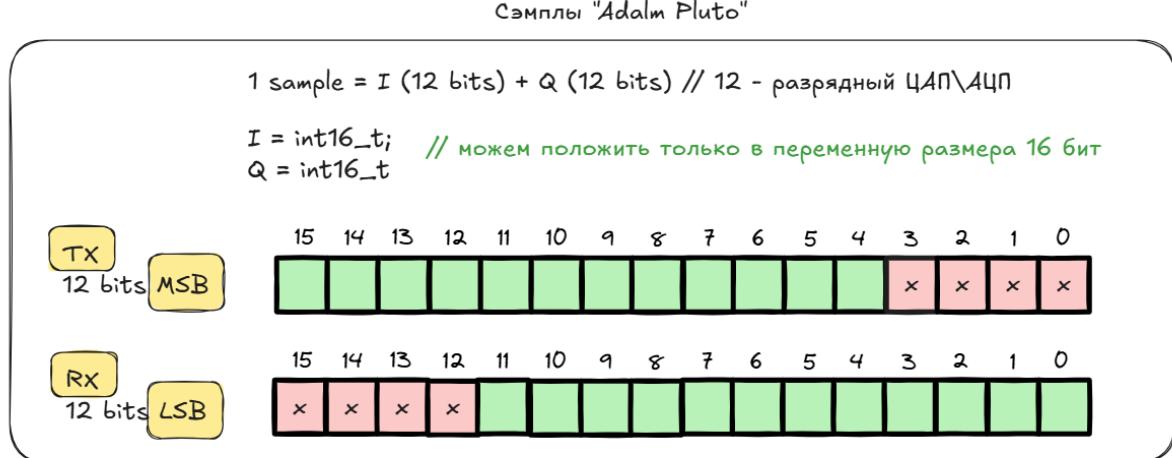


Рисунок 1 — Структура семплов

Pluto SDR имеет 12-ти битный АЦП, это значит, что для I и Q максимальное значение составляет  $2^{12} = 4096$ , но один бит займет знак, поэтому I и Q будут принимать значения из отреза [-2048;2047]. Для хранения I или Q в C++ используется тип данных `int16_t` (если брать меньше, то семпл не поместится). Таким образом один семпл занимает 4 байта памяти. Также есть вариант хранить семплы в одной переменной `int32_t`, но в таком случае для получения доступа к I или Q придется использовать битовые сдвиги и накладывать маски.

Стоит отметить, что Pluto SDR странно работает с TX семплами (которые хотим передавать) и интерпретировать как семплы только первые 12 бит переменной (big-indian), поэтому необходимо сдвигать значение I и Q на 4 бита влево («4») при работе с tx буффером, чтобы Pluto SDR корректно их интерпретировать.

## 2.2 Структура буфера семплов в Pluto SDR

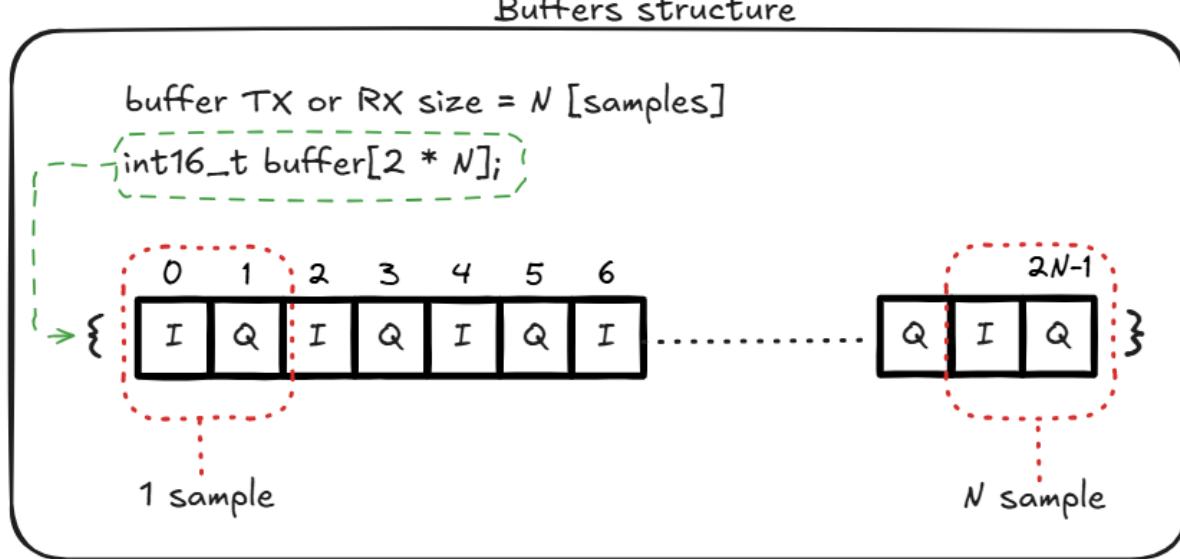


Рисунок 2 — Структура буфера

Семпл состоит из компонент  $I$  и  $Q$ , которые хранятся последовательно друг за другом, т.е в буффер будет иметь последовательность вида  $I_0, Q_0, I_1, Q_1, \dots, I_n, Q_n$ , поэтому если мы хотим принять/передать  $N$  семплов, то буффер должен быть размером  $2N$ , т.к семпл состоит из двух чисел.

## 2.3 Запись RX семплов в буффер с Pluto SDR

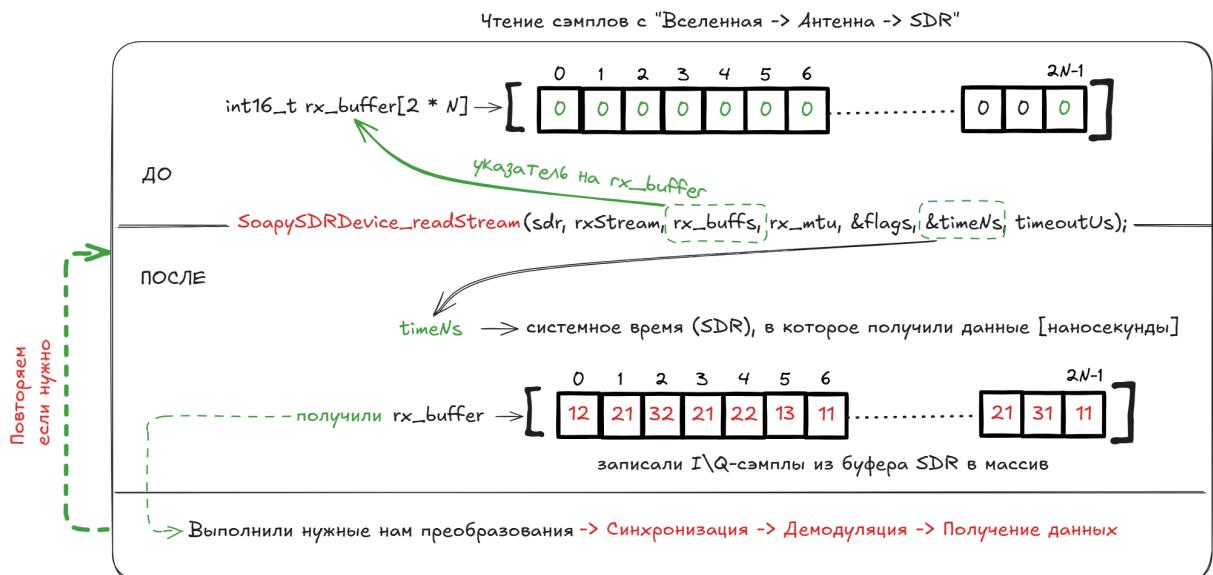


Рисунок 3 — Запись RX семплов

Для записи RX семплов в буффер используется функция SoapySDRDevice\_-readStream(), в которую необходимо передать указатель на буффер, в который хотим писать, и кол-во семплов, которые мы хотим записать, и еще некоторые дополнительные параметры. После выполнение функции в наш буффер будут записаны семплы, принятые SDR.

## 2.4 Создание своих семплов и их отправка в Pluto SDR

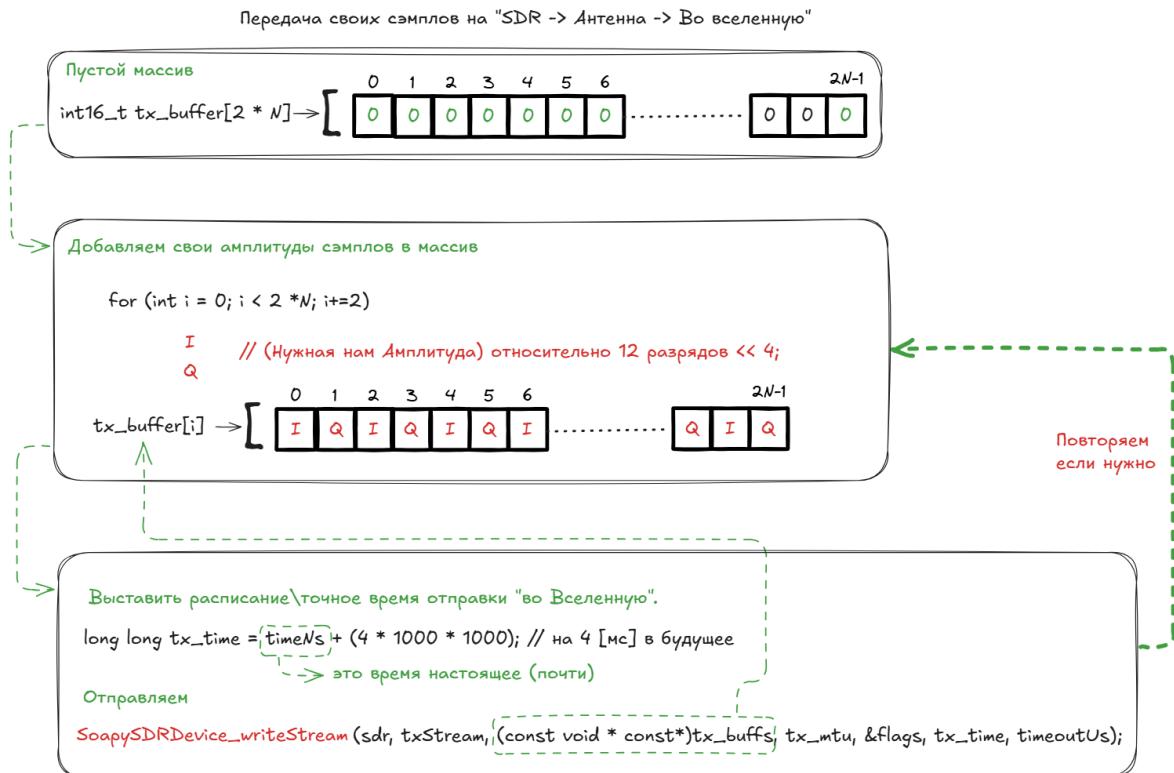


Рисунок 4 — Создание и отправка семплов

Для отправки семплов сначала необходимо заполнить tx буффер сами-ми семплами. Формировать сами I и Q можно разными способами, но самое главное разместить их в правильном порядке. Для самой отправки использу-ется функция SoapySDRDevice\_writeStream(), в которую необходимо пере-дать указатель на буффер с семплами и время, через которое семплы будут отправлены, а также дополнительные параметры. После вызова функции че-рез 4нс наши семплы отправятся с Pluto SDR.

## ПРАКТИКА

### 3.1 Формирование сэмплов

#### 3.1.1 Перевод строки в бинарный вид

Я буду формировать простой прямоугольный сигнал, который будет передавать небольшую строку. Чтобы передавать текст, нужно сначала перевести символы (char) в биты. Для этого я написал функцию:

```
int8_t* stob(char* str, int* out_bits_count) {
    // get string len
    int len = strlen(str);

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

    // array for bits
    int8_t* bits = (int8_t*)malloc(*out_bits_count *
        sizeof(int8_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;
```

}

Функция принимает саму строку и указатель на переменную, в которой вернет количество бит, чтобы знать размер битовой последовательности после завершения функции, и возвращает битовую последовательность.

## Пример работы функции:

Переведем в бинарный вид строку "Hello World":

```
int bits_count;
uint8 t* bits = stob("Hello World", &bits count);
```

На выходе получим такую последовательность:

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

Рисунок 5 — "Hello World" в бинарном виде

### **3.1.2 Способ кодирования**

Чтобы передать нули и единицы я буду использовать самый простой способ кодирования: 0 - ноль, 1 - максимальный I и минимальный Q.

### 3.1.3 Заполнение буфера семплами

Для заполнения tx буфера я написал отдельную функцию, которая принимает битовую последовательность (сообщение), длину битовой последовательности и размер буфера, а возвращает массив семплов.

```
int16_t* bits_to_samples(uint8_t* bits, int bits_count, int tx_mtu){

    // allocate memory
    int16_t* tx_buff = (int16_t*)malloc(sizeof(int16_t) * tx_mtu
        * 2);

    // iterate on bits
    for (int i = 0; i < bits_count; i += 1)
    {
        // fill tx buff with samples
    }
}
```

```

    for(int j = i*TAU_ON_BITS; j < i*TAU_ON_EL + 20 && j <
        tx_mtu*2; j+=2){
        if(bits[i]){
            tx_buff[j] = 2047 << 4; // I
            tx_buff[j+1] = -2047 << 4; // Q
        } else{
            tx_buff[j] = 0; //I
            tx_buff[j+1] = 0; //Q
        }
    }

    return tx_buff;
}

```

Самое важное здесь - учитывать продолжительность импульса, т.е то кол-во семплов, которое будет приходиться на один бит информации. Я выбрал 10 семплов на бит (TAU), а это значит, что на 1 бит информации будет приходиться 20 элементов массива (TAU\_ON\_EL). Чем больше длительность сигнала, тем выше шанс верно декодировать его на приемной стороне, но при этом падает скорость передачи данных. Работать заполнение будет так: берем первые 20 элементов массива и заполняем его идентичными значениями (в соответствии со состоянием бита), потом берем следующие 20 и т.д.

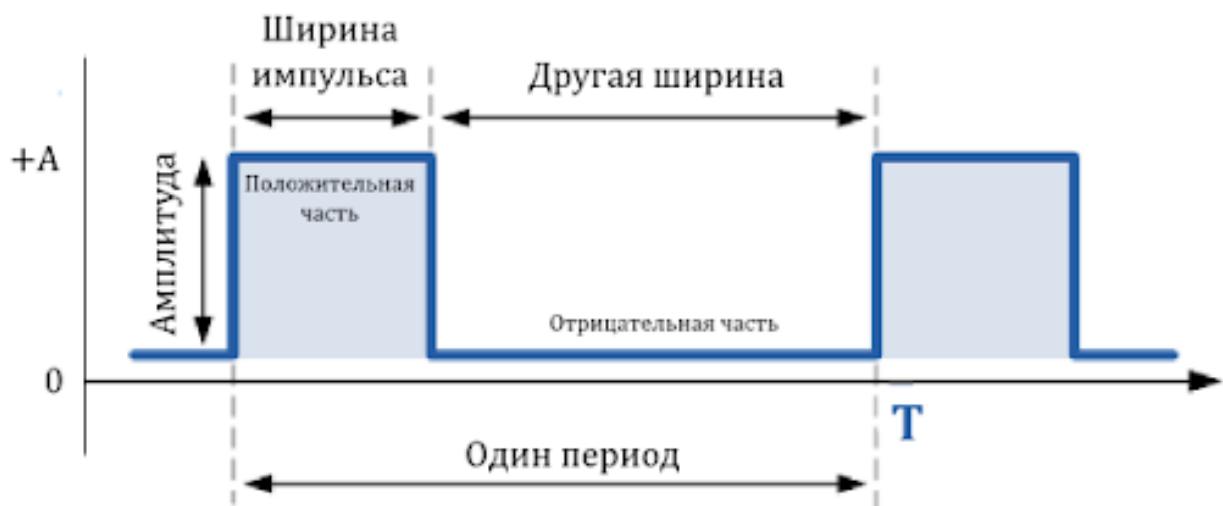


Рисунок 6 — Пример прямоугольного сигнала с обозначениями

## 3.2 Результат работы

### 3.2.1 Семплы до отправки

С помощью Python визуализируем семплы

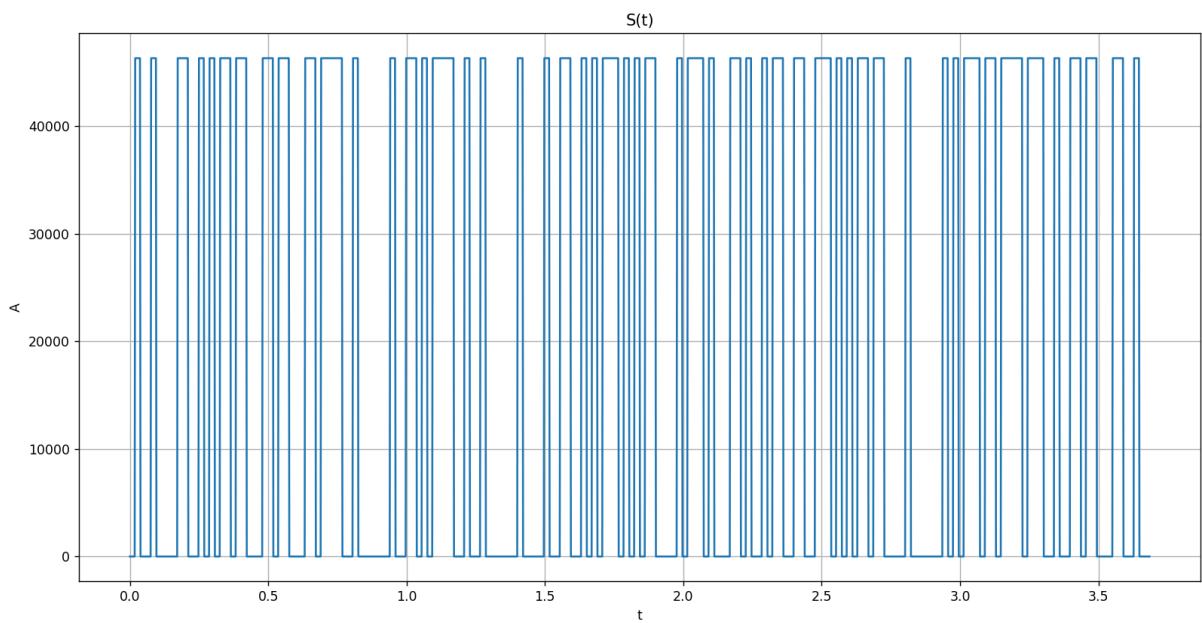


Рисунок 7 — Пример прямоугольного сигнала

Этот прямоугольный сигнал содержит наше сообщение.

### 3.2.2 Требование к длине сообщения

При работе с Pluto SDR рекомендуется использовать для отправки/приема 1920 семплов или число семплов кратное 1920. Если на 1 бит приходится 10 семплов, то сообщение должно быть длиной 192 бита, а т.к я пытаюсь передать символы размером 8 бит, то необходима строка длиной 24 символа или строка с длиной кратной 24.

## 3.3 Семплы на приеме

На приеме получили следующий сигнал:

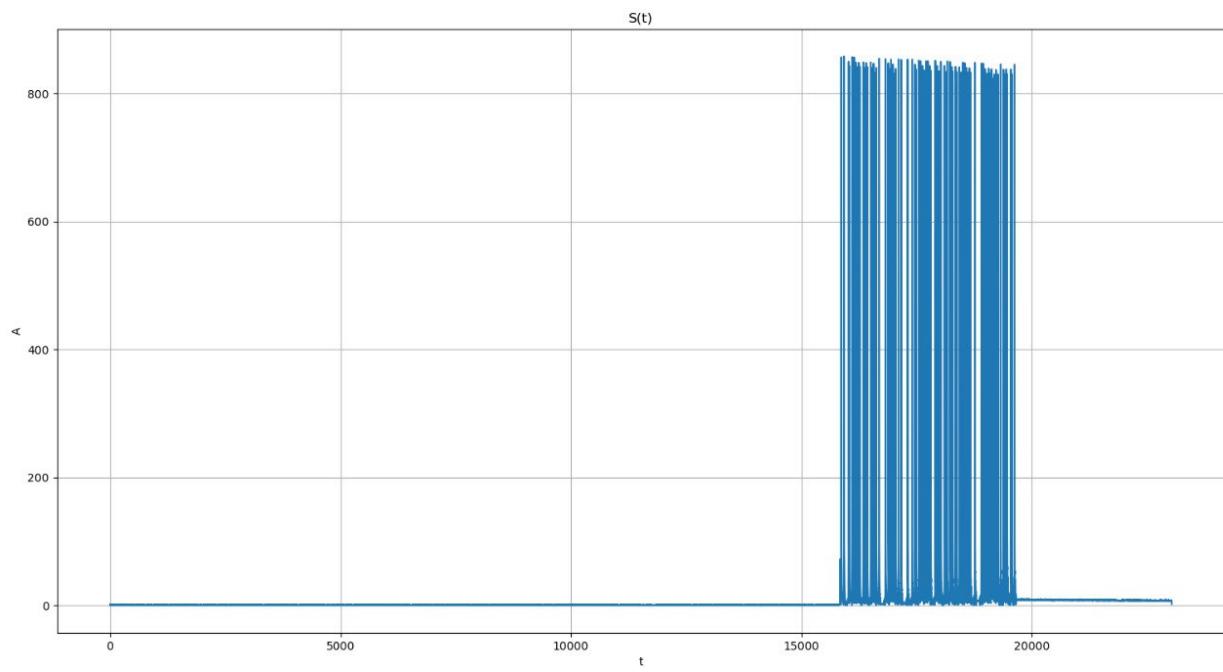


Рисунок 8 — Сигнал на приеме

Где то среди этого сигнала наше сообщение, но чтобы правильно его декодировать, необходимо реализовать логику приемной стороны и выполнить символьную синхронизацию. Эти темы будет рассматриваться в дальнейшем.

## **ВЫВОД**

В ходе проделанной работы я улучшил свои навыки работы с SDR с помощью C/C++ и библиотеки SoapySDR. Научился формировать семплы и отправлять/принимать их с SDR.

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  
Занятие №4

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

1	ВВЕДЕНИЕ .....	3
	1.1 Введение .....	3
	1.2 Цель: .....	3
2	ЛЕКЦИЯ .....	4
	2.1 Задачи цифровой системы связи .....	4
	2.2 Архитектура цифровой системы связи .....	4
	2.2.1 Архитектура передатчика .....	4
	2.3 Схемы модуляции .....	7
	2.3.1 BPSK .....	7
	2.3.2 QPSK .....	9
3	ПРАКТИКА .....	11
	3.1 Треугольный сигнал .....	11
	3.1.1 Программная реализация .....	11
	3.2 Параболический сигнал .....	12
	3.2.1 Программная реализация .....	12
	3.3 Реализация BPSK модуляции .....	13
	3.3.1 BPSK модулятор .....	13
	3.3.2 Формирующий фильтр .....	14
	3.3.3 Перемножение с несущим колебанием .....	14
	3.4 Реализация QPSK модуляции .....	15
	3.4.1 QPSK модулятор .....	15
	3.4.2 Формирующий фильтр .....	16
	3.4.3 Перемножение с несущим колебанием .....	17
	3.5 Результат работы .....	18
	3.5.1 Визуализация параболического сигнала до отправки ..	18
	3.5.2 Визуализация параболического сигнала на приеме .....	18
	3.5.3 Визуализация треугольного сигнала на приеме .....	19
4	ВЫВОД .....	20

## ВВЕДЕНИЕ

### 1.1 Введение

На прошлом занятии мы формировали семплы произвольной формы с целью улучшения навыков работы с SDR с помощью C/C++ и SoapySDR. На этом занятии продолжим формировать произвольные сигналы и перейдем к работе с BPSK/QPSK модуляцией.

### 1.2 Цель:

Повторить архитектуру систем цифровой связи. Узнать, что такое модуляция и познакомиться с BPSK и QPSK модуляциями. Построить на языке Python модель, которая будет поэтапно визуализировать процесс модуляции.

# ЛЕКЦИЯ

## 2.1 Задачи цифровой системы связи

Какие задачи у системы связи? Задача системы связи заключается в надежном передать поток бит на заданной скорости по каналу связи. Для передачи по каналу связи мы используем электромагнитные колебания -  $\sin$  и  $\cos$  какой-то частоты.

## 2.2 Архитектура цифровой системы связи

### 2.2.1 Архитектура передатчика

Базовая архитектура цифровой системы связи в простейшем случае состоит из приемника, передатчика и радиоканала. Рассмотрим упрощенную архитектуру передатчика:

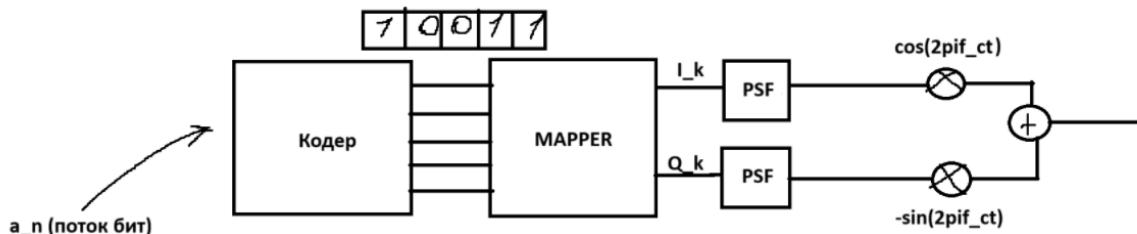


Рисунок 1 — Архитектура передатчика

Формируется поток бит, который поступает на кодер. В нашем случае кодер просто делит поток бит на блоки определенной длины. У кодера один вход, по которому последовательно поступают биты, а на выходе N-битная шина, которая уже параллельно передает биты на мапер. Кодер можно представить в виде схемы следующим образом:

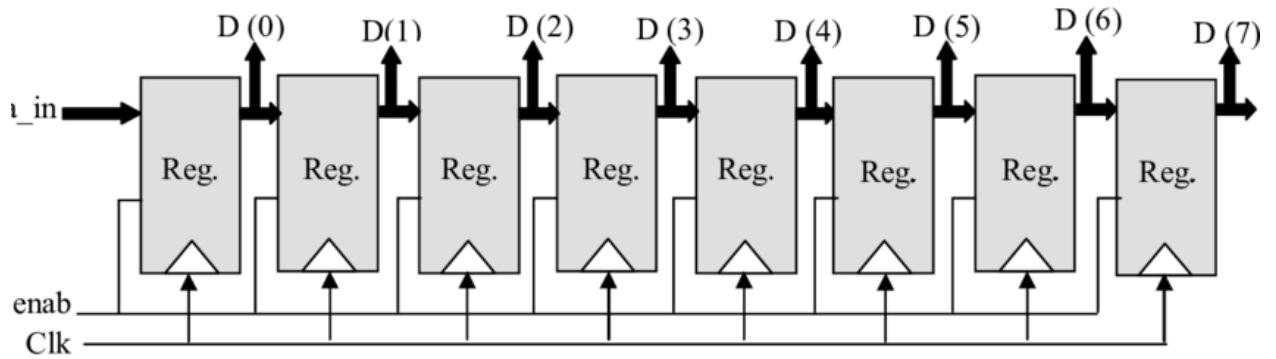


Рисунок 2 — Схема последовательно-параллельного преобразователя

Имеется  $N$  регистров с общим тактовым сигналом, которые соединены последовательно. По каждому тактовому сигналу поток бит будет "продвигаться" по триггерам и попадать на  $N$ -битную шину, т.е параллельно выводиться из устройства.

Далее блоки битов попадают на маппер, который ставит в соответствие каждому блоку битов числа  $I$  и  $Q$ . Если блок бит имеет размерность  $N$ , то в маппере будет заложено  $2^N$  комбинаций. На выходе маппера параллельно получим числа  $I$  и  $Q$ .

Сигнал, который мы хотим сформировать имеет вид  $S_k(t) = I_k \cos(2\pi f_c t) - Q_k \sin(2\pi f_c t)$ , где  $k$  - номер текущего символа,  $I_k$ ,  $Q_k$  - координаты символа в сигнальной диаграмме. Эти координаты мы получаем на выходе маппера. Эти числа в будущем станут параметрами колебания.

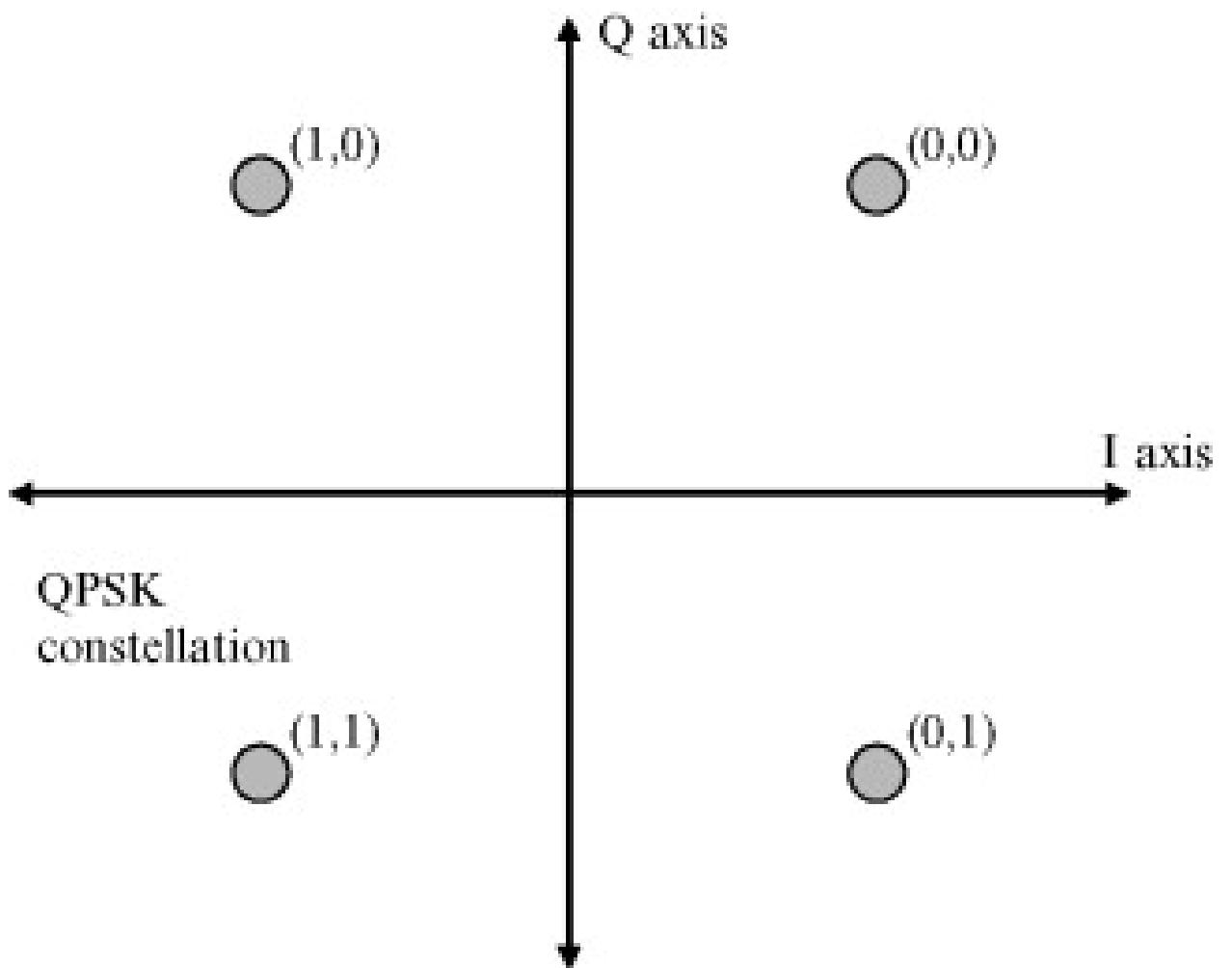


Рисунок 3 — Пример сигнальной диаграммы для QPSK модуляции

Далее  $I$  и  $Q$  попадают на Pulse Shaping Filter (PSF). Фильтр выполняет 2 задачи: определяет ширину спектра радиосигнала и его форму, а также применяется на приемной стороне для символьной синхронизации. Фильтр характеризуется важным параметром во временной области - импульсной характеристикиой  $g(t)$ . Импульсная характеристика может быть различной, но для простоты восприятия будем рассматривать только прямоугольную характеристику. На выходе фильтра получим прямоугольные отрезки сигналов.

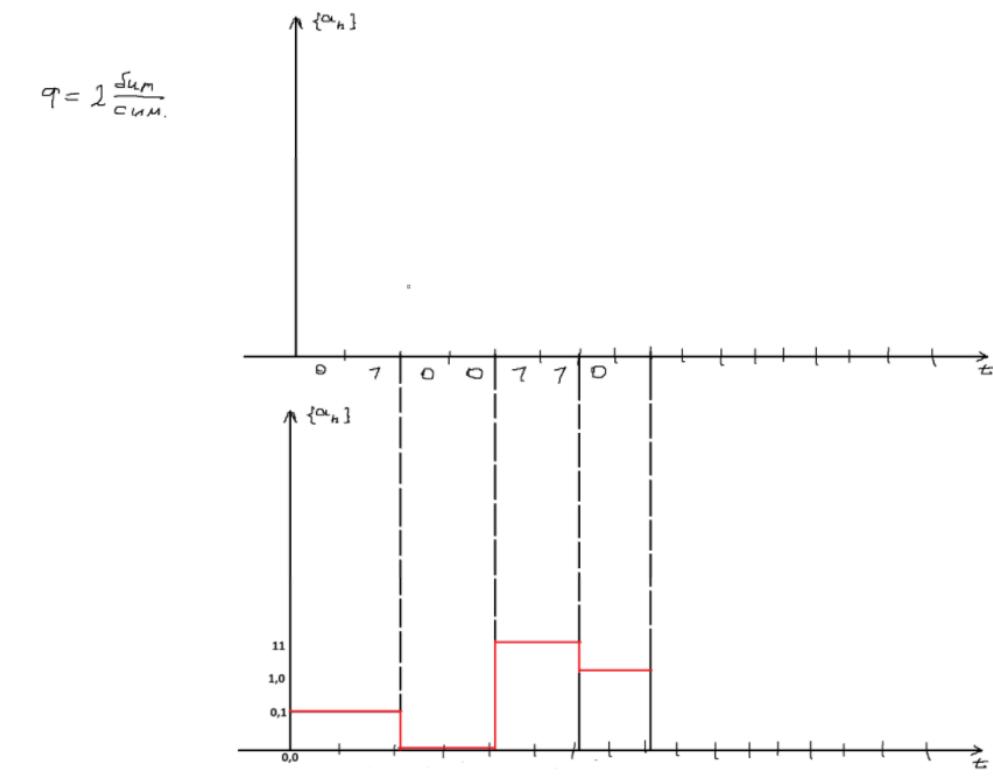


Рисунок 4 — Пример формирования символов

Длительность символов вычисляется как  $\frac{1}{R_s}$ , где  $R_s$  - символьная скорость. В свою очередь символьная скорость  $R_s$  вычисляется как  $\frac{R_b}{\log_2(M)}$ , где  $R_b$  - битовая скорость (та скорость, с которой биты поступают на маппер),  $M$  - кол-во точек созвездия. После маппера битовая скорость переходит в символьную, которая определяет длительность прямоугольного импульса.

## 2.3 Схемы модуляции

### 2.3.1 BPSK

BPSK (Binary Phase Shift Keying) - двухпозиционная фазовая манипуляция. В такой схеме модуляции на 1 бит приходится 1 символ.

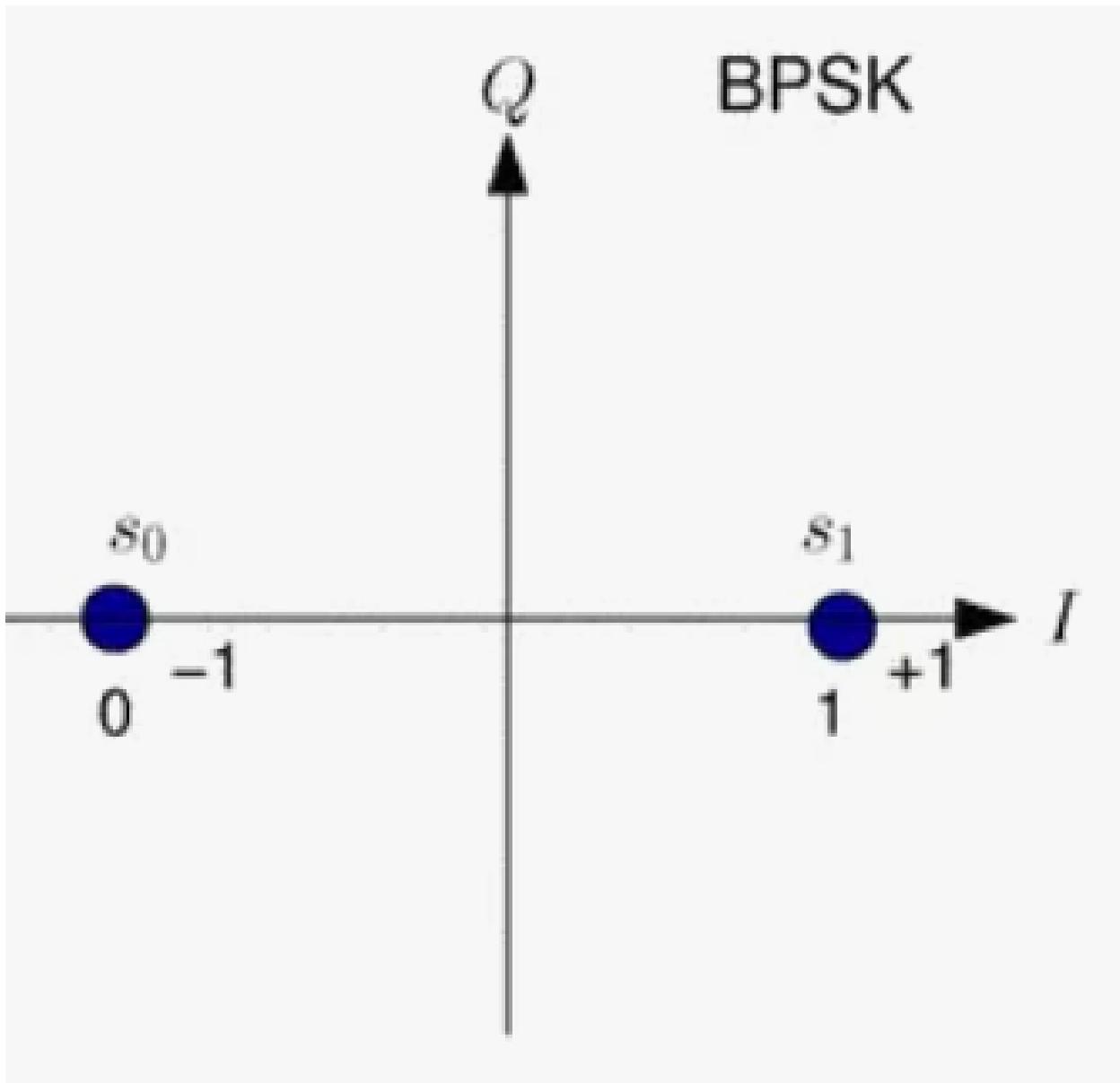


Рисунок 5 — Созвездие BPSK модуляции

Логика BPSK следующая: если бит имеет состояние 0, то он будет соответствовать сигналу с начальной фазой  $\phi$  равной 0, т.е точке с координатой  $(1, 0)$ , если бит находится в состоянии 1, то биту будет соответствовать сигнал с начальной фазой  $\phi$  равной  $\pi$ , т.е точке с координатой  $(-1, 0)$ . Можем заметить, что квадратурная составляющая всегда равна 0, это значит, что в сигнале будет только  $\cos$  составляющая, т.к  $\sin$  составляющая занулится (видно из уравнения сигнала  $S_k(t) = I_k \cos(2\pi f_c t) - Q_k \sin(2\pi f_c t)$ ).

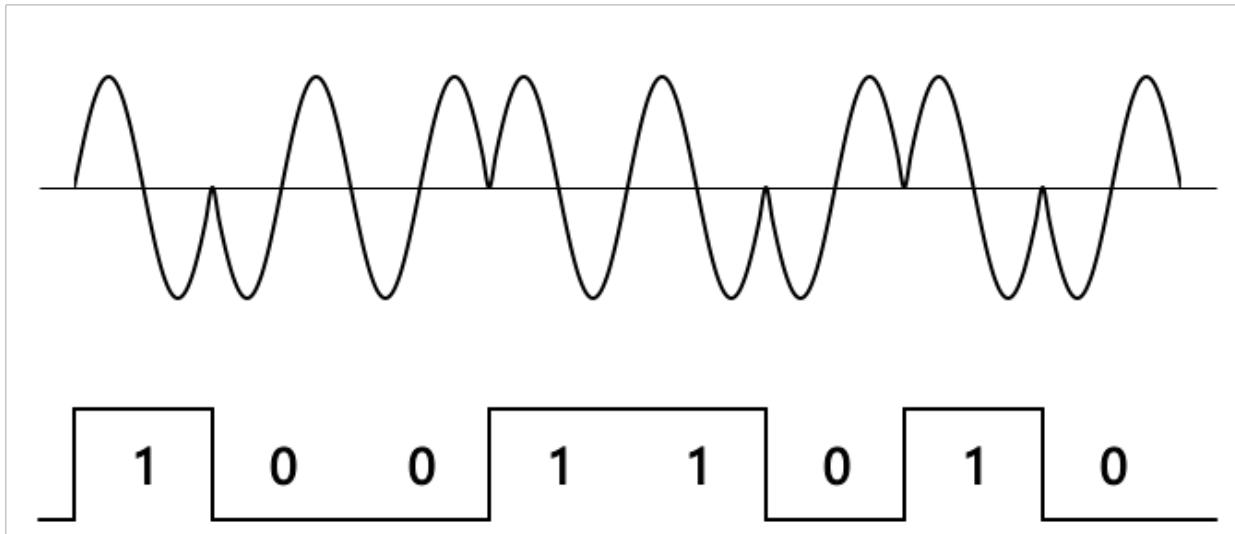


Рисунок 6 — Пример работы BPSK

### 2.3.2 QPSK

QPSK (Quadrature Phase Shift Keying) - квадратурная фазовая манипуляция. 2 бита кодируются в одном символе. Используется 4 различных фазы для представления пар битов. Логика этой схемы модуляции такая же, как у BPSK, но уже добавляется квадратурные составляющие, т.е в сигнале будут как  $\cos$  составляющая, так и  $\sin$  составляющая. Точке  $(0, 0)$  будет соответствовать фаза  $\frac{\pi}{4}$ ,  $(1, 0)$  будет соответствовать фаза  $\frac{3\pi}{4}$ ,  $(1, 1)$  будет соответствовать фаза  $\frac{5\pi}{4}$ ,  $(0, 1)$  будет соответствовать фаза  $\frac{7\pi}{4}$ ,

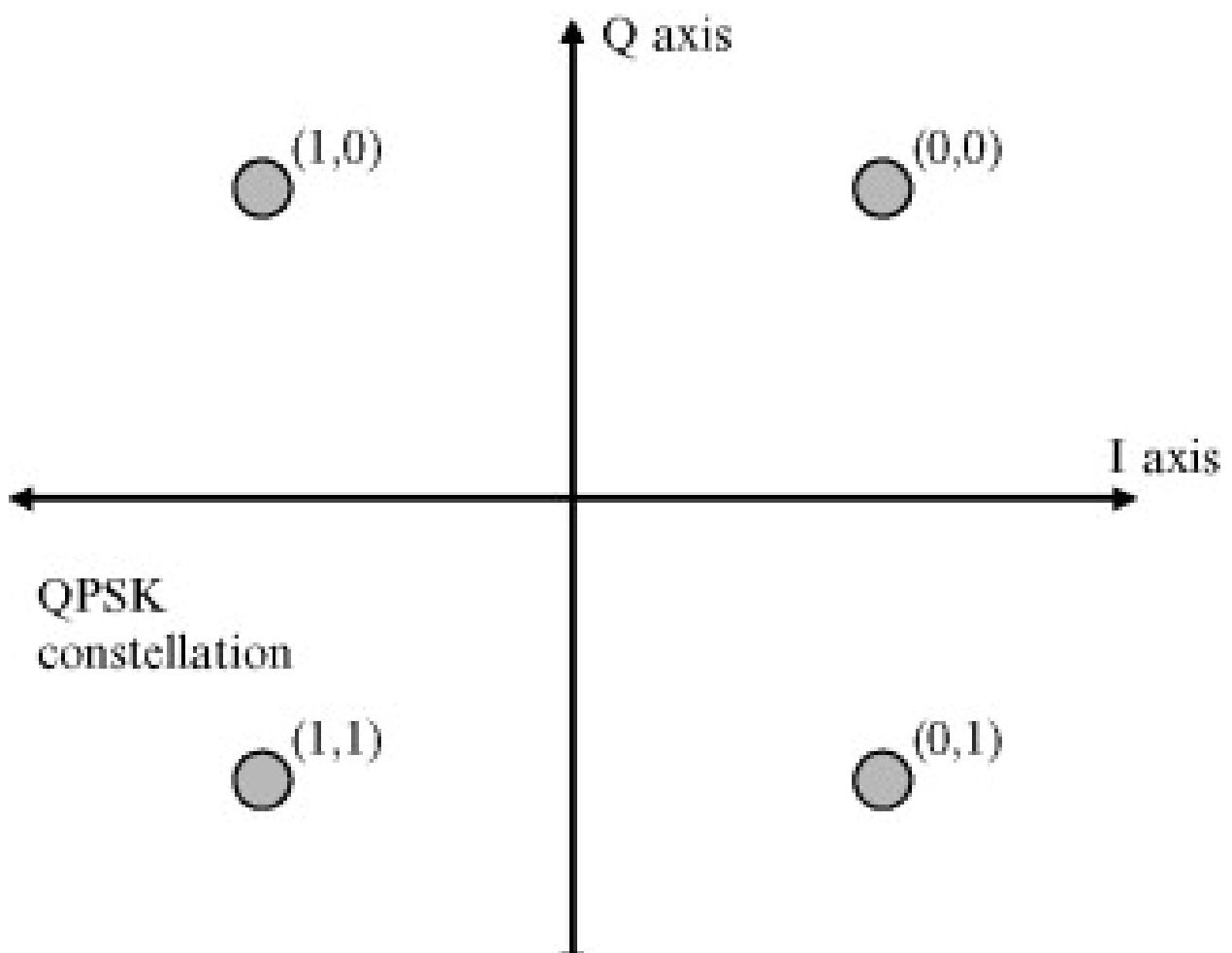


Рисунок 7 — QPSK созвездие

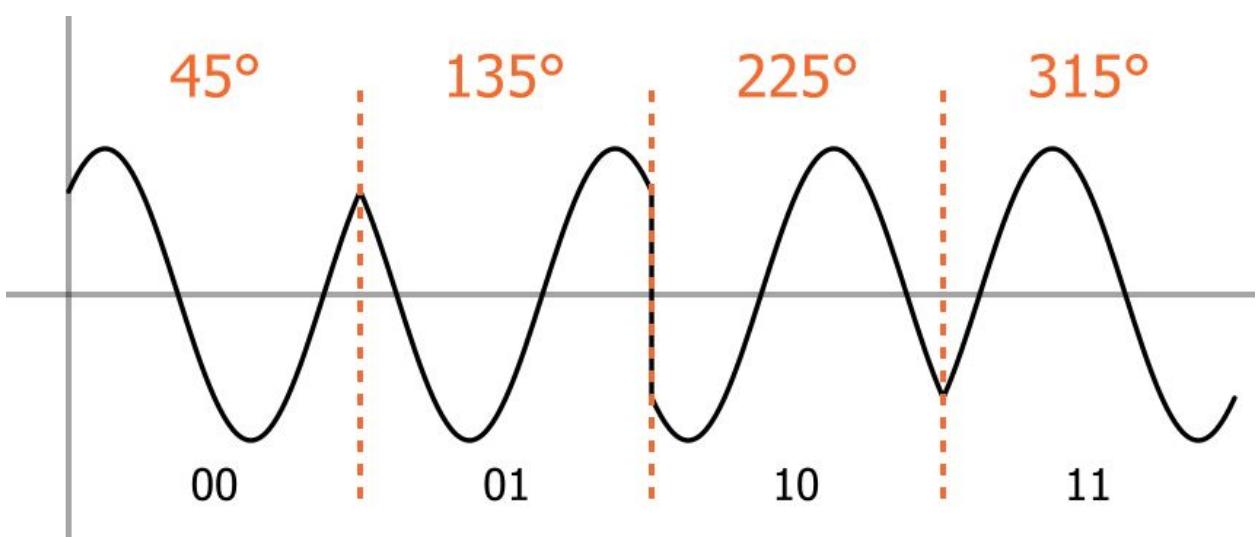


Рисунок 8 — Пример работы QPSK

## ПРАКТИКА

### 3.1 Треугольный сигнал

Я буду формировать такой сигнал по следующему принципу:

1. Если бит равен единице, то в момент длительности сигнала  $\frac{\tau}{2}$  буду задавать максимальное значение, а в остальных случаях буду задавать нули. Таким образом в одной точке будет образовываться пик, который и будет придавать сигналу форму треугольника.
2. Если бит равен 0, то на всей длительности  $\tau$  сигнал будет принимать значение 0.

#### 3.1.1 Программная реализация

```
#define TAU 10
#define TAU_ON_ELEMENT 20

int16_t* bits_to_triangle_signal(uint8_t* bits, int
bits_count, int tx_mtu){

    // allocate memory
    int16_t* tx_buff = (int16_t*)malloc(sizeof(int16_t) * tx_mtu
* 2);

    // iterate on bits
    for (int i = 0; i < bits_count; ++i)
    {
        // fill tx_buff with samples
        for(int j = i*TAU_ON_ELEMENT; j < i*TAU_ON_ELEMENT + 20
        && j < tx_mtu*2; j+=2){
            if(bits[i] && j % 10 == 5){
                tx_buff[j] = 2047 << 4;      // I
                tx_buff[j+1] = -2047 << 4; // Q
            } else{
                tx_buff[j] = 0;           // I
                tx_buff[j+1] = 0;         // Q
            }
        }
    }
}
```

```

        }
    }

    return tx_buff;
}

```

Код полностью идентичен коду, который использовался для создания прямоугольного сигнала. Единственное отличие заключается в условии  $j \% 10 == 5$ , которое позволяет выбирать середины символов.

## 3.2 Параболический сигнал

Сформируем параболу, перебирая точки от  $-\frac{tx\_mtu}{10}$  до  $\frac{tx\_mtu}{10}$ , где  $tx\_mtu$  - кол-во семплов, передаваемых за раз. Деление на 10 используется для масштабирования, чтобы не возводить большие числа в квадрат.

### 3.2.1 Программная реализация

```

int16_t* parabola_signal(int tx_mtu){

    // allocate memory
    int16_t* tx_buff = (int16_t*)malloc(sizeof(int16_t) * tx_mtu
        * 2);

    float coef = -tx_mtu / 10;

    for (int i = 0; i < 2 * tx_mtu; i+=2)
    {
        tx_buff[i] = int16_t(coef * coef);    // I
        tx_buff[i+1] = int16_t(coef * coef); // Q

        coef += 0.1;
    }

    return tx_buff;
}

```

### 3.3 Реализация BPSK модуляции

Шаги по реализации:

1. Написать BPSK модуляцию (логика маппера)
2. На основе I и Q, полученных на прошлом шаге, сгенерировать прямоугольные импульсы (логика формирующего фильтра)
3. Перемножить прямоугольные импульсы на несущее колебание
4. Проанализировать полученные результаты и сделать вывод

#### 3.3.1 BPSK модулятор

```
double* BPSK_modulation(int* bits, int bits_count){
    //allocate memory
    double* IQ_samples = (double*)malloc(sizeof(double) *
        bits_count * 2);
    //iterate on bits
    for(int i,j = 0; i < bits_count * 2; i+=2){
        if(bits[j]){
            IQ_samples[i] = 1;          // I
            IQ_samples[i + 1] = 0;      // Q
        }else{
            IQ_samples[i] = -1;        // I
            IQ_samples[i + 1] = 0;     // Q
        }
        ++j;
    }

    return IQ_samples;
}
```

Сначала выделяем память под IQ семплы. Т.к на каждый бит приходит-ся 1 семпл, а на 1 семпл - 2 числа (I и Q), то размер массива с семплами должен быть вдвое больше кол-ва бит. Далее перебираем биты и в зависимости от его значения формируем семпл. В результирующем массиве I и Q идут в строгой последовательности  $I_0, Q_0, I_1, Q_1, \dots, I_N, Q_N$ . Массив в дальнейшем запишется в файл.

### 3.3.2 Формирующий фильтр

На языке Python парсим файл с семплами из прошлого шага. Далее формируем прямоугольные импульсы.

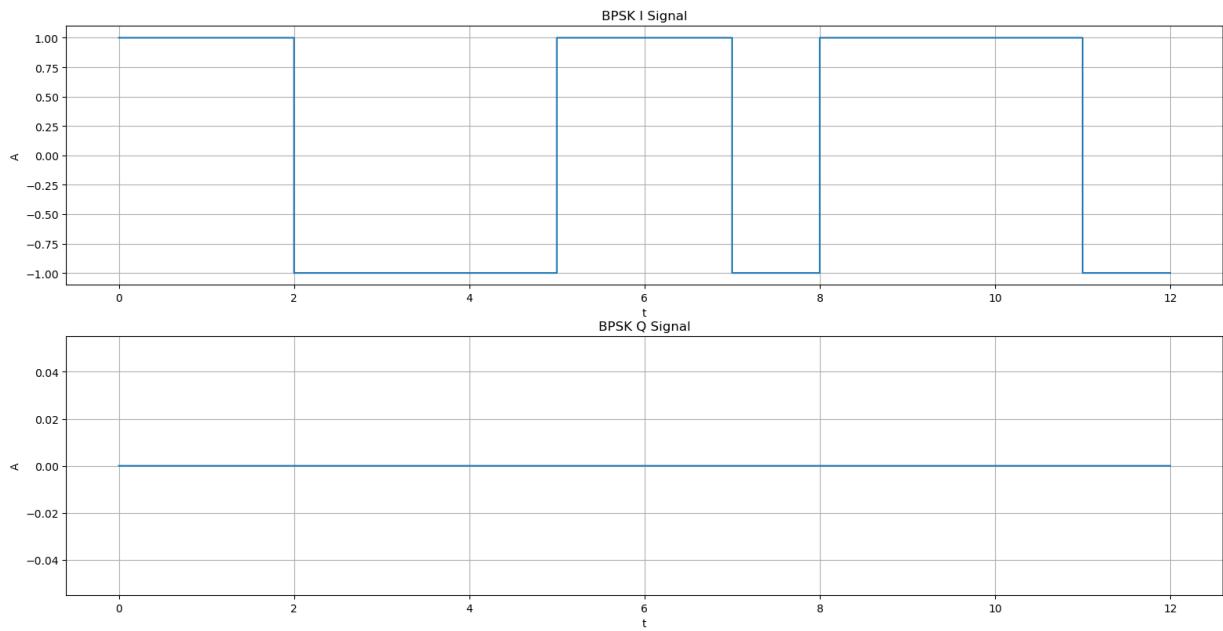


Рисунок 9 — BPSK-символы

Здесь явно видно, что в BPSK модуляции квадратурная составляющая всегда равна 0. Я передавал последовательность длиной 12 бит, и это заняло 12 единиц времени, т.е на каждый бит приходился 1 символ, который длился 1 единицу времени.

### 3.3.3 Перемножение с несущим колебанием

Теперь перемножим символы с несущим колебанием. В роли несущего колебания я использовал  $\cos(2\pi ft)$  и  $-\sin(2\pi ft)$  с частотой  $f = 1$ . В реальности частоты несущих колебаний в разы больше, но для наглядности я взял маленькое значение. Этот процесс можно сравнить с наложением маски, где маской является прямоугольный сигнал. Если посмотреть на схему цифровой системы связи, которая была дана в разделе с теорией, то этот процесс происходит в цепи, которая следует после формирующего фильтра.

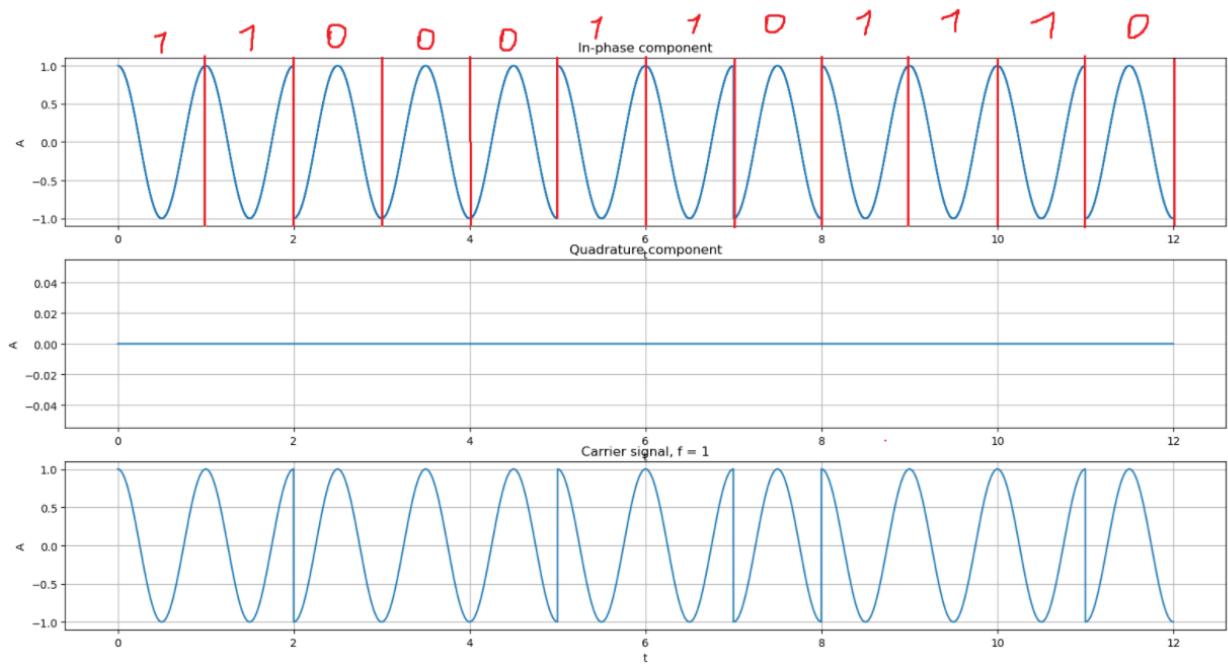


Рисунок 10 — Перемножение BPSK символов с несущим сигналом

Можем видеть, что квадратурная компонента по прежнему равна 0, т.е в сигнале присутствует только  $\cos$ . В синфазной компоненте можем видеть, как меняется фаза колебания, которая отражает значение бита. Если фаза равна 0, то передается 1, если фаза равна  $\pi$ , то передается 0. В местах, где меняется фаза, происходит смена значения бита. Таким образом можно восстановить передаваемую битовую последовательность. Результирующий сигнал будет иметь вид синфазной компоненты, т.к квадратурная всегда равна 0.

### 3.4 Реализация QPSK модуляции

Шаги по реализации будут в точности такие же, как у BPSK модуляции.

#### 3.4.1 QPSK модулятор

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

```

    IQ_samples[i] = -1;           //I
    if(bits[i + 1]){
        IQ_samples[i + 1] = -1;   //Q
    }else{
        IQ_samples[i + 1] = 1;   // Q
    }
}else{
    IQ_samples[i] = 1;           //I
    if(bits[i + 1]){
        IQ_samples[i + 1] = 1;   //Q
    }else{
        IQ_samples[i + 1] = -1;  //Q
    }
}

return IQ_samples;
}

```

Заметим, что в случае QPSK модуляции массив под семплы вдвое меньше, т.к в QPSK модуляции на 1 семпл приходится 2 бита.

### 3.4.2 Формирующий фильтр

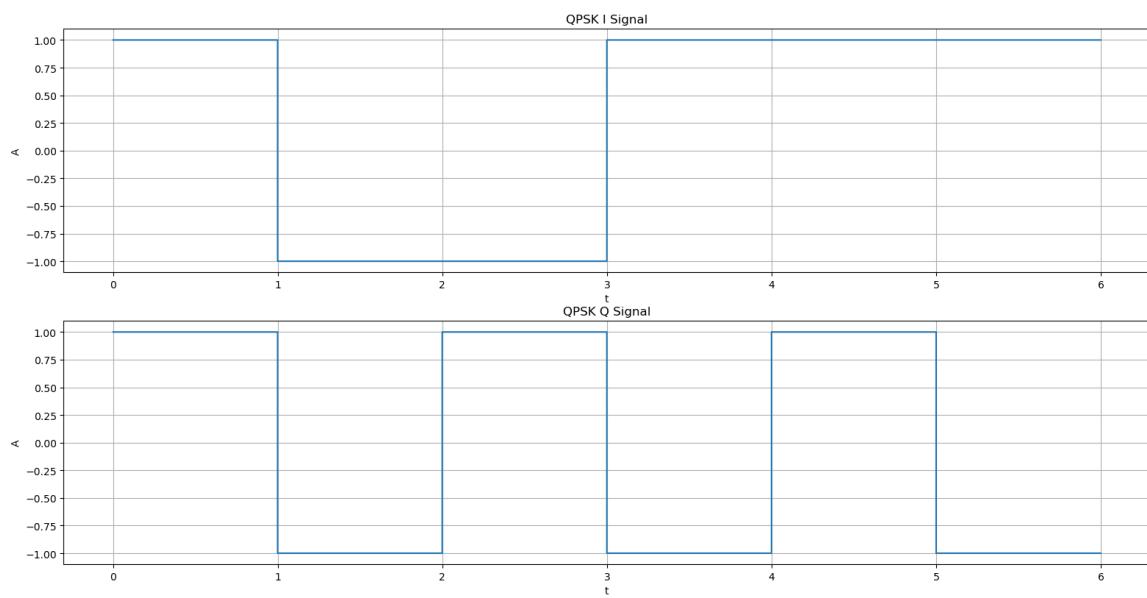


Рисунок 11 — QPSK символы

Последовательность из 12 бит передается уже за 6 единиц времени. Это связано с тем, что в QPSK на семпл приходится 2 бита. Можно сделать вывод о том, что чем больше точек в сигнальном созвездии, тем выше скорость передачи данных. 1 символ длится уже не 1 единицу времени, а 0.5 единиц времени (на графике чуть неверный масштаб)

### 3.4.3 Перемножение с несущим колебанием

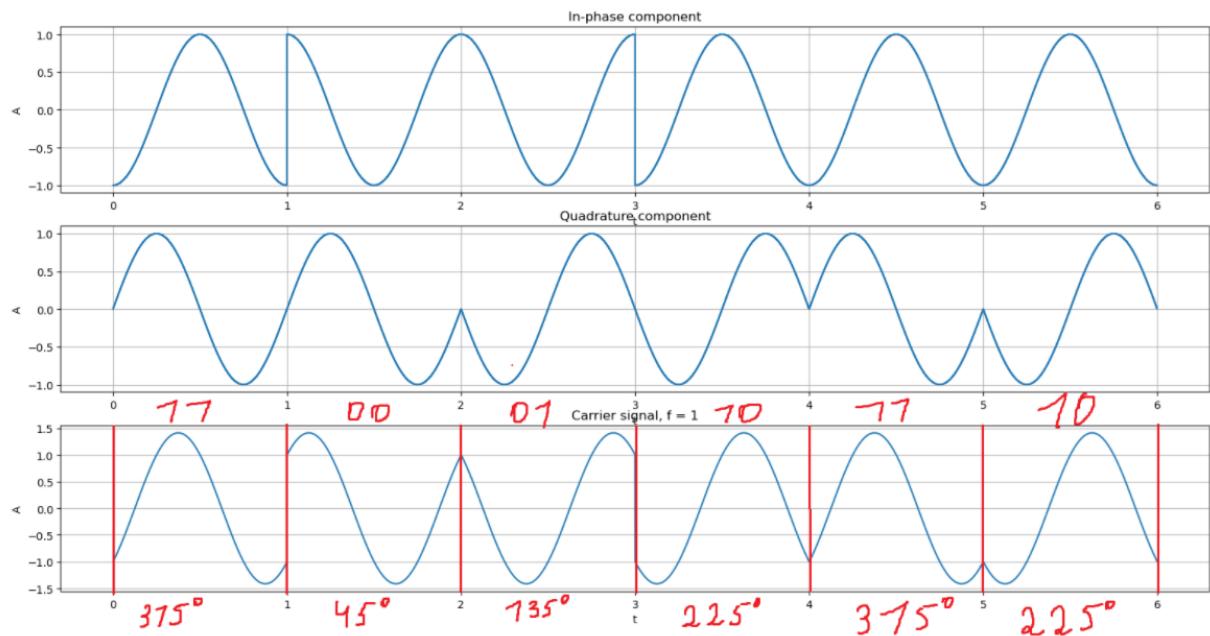


Рисунок 12 — Перемножение QPSK символов с несущим сигналом

Можем видеть, что квадратурная компонента теперь не равна 0, т.е в сигнале присутствует  $\cos$  и  $\sin$ . Еще можно заметить, что максимальная амплитуда в результирующем сигнале уже не равна 1, а равна  $\sqrt{I^2 + Q^2} = \sqrt{1^2 + 1^2} = \sqrt{2}$ . Результирующий сигнал будет иметь вид  $I\cos(2\pi f_c t) - Q\sin(2\pi f_c t)$ .

## 3.5 Результат работы

### 3.5.1 Визуализация параболического сигнала до отправки

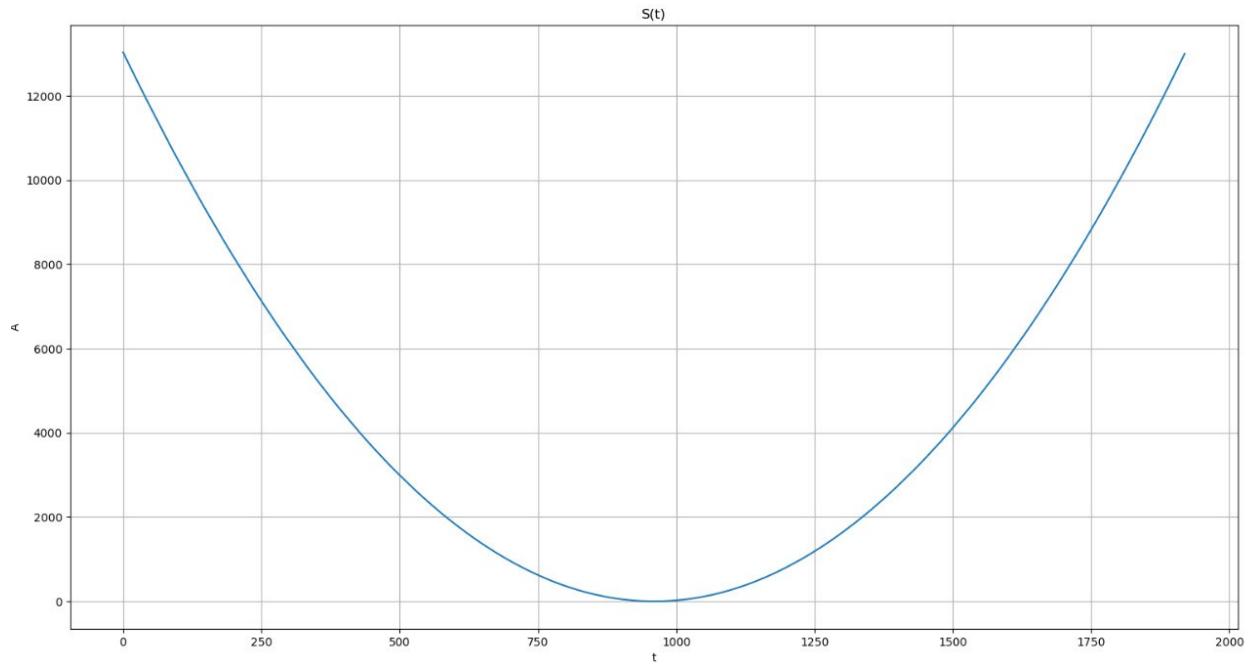


Рисунок 13 — Параболический сигнал до отправки

### 3.5.2 Визуализация параболического сигнала на приеме

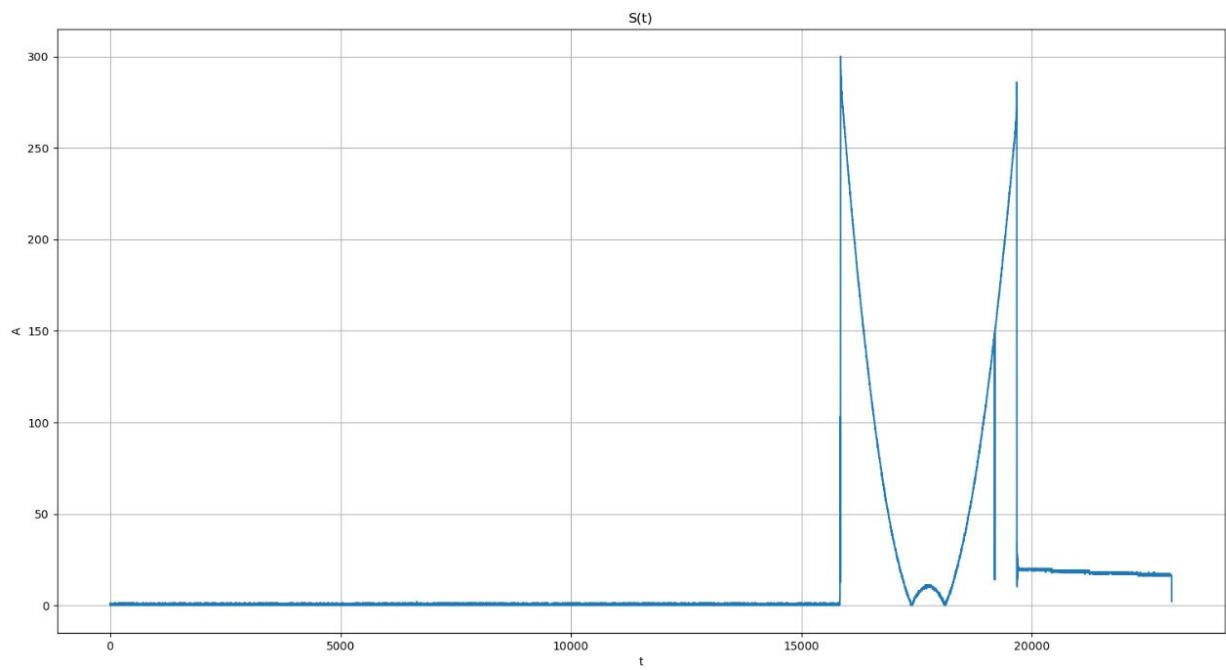


Рисунок 14 — Параболический сигнал на приеме

Можем наблюдать параболу в нашем сигнале, по центру парабола отображена вверх, потому что я строил график модуля сигнала.

### 3.5.3 Визуализация треугольного сигнала на приеме

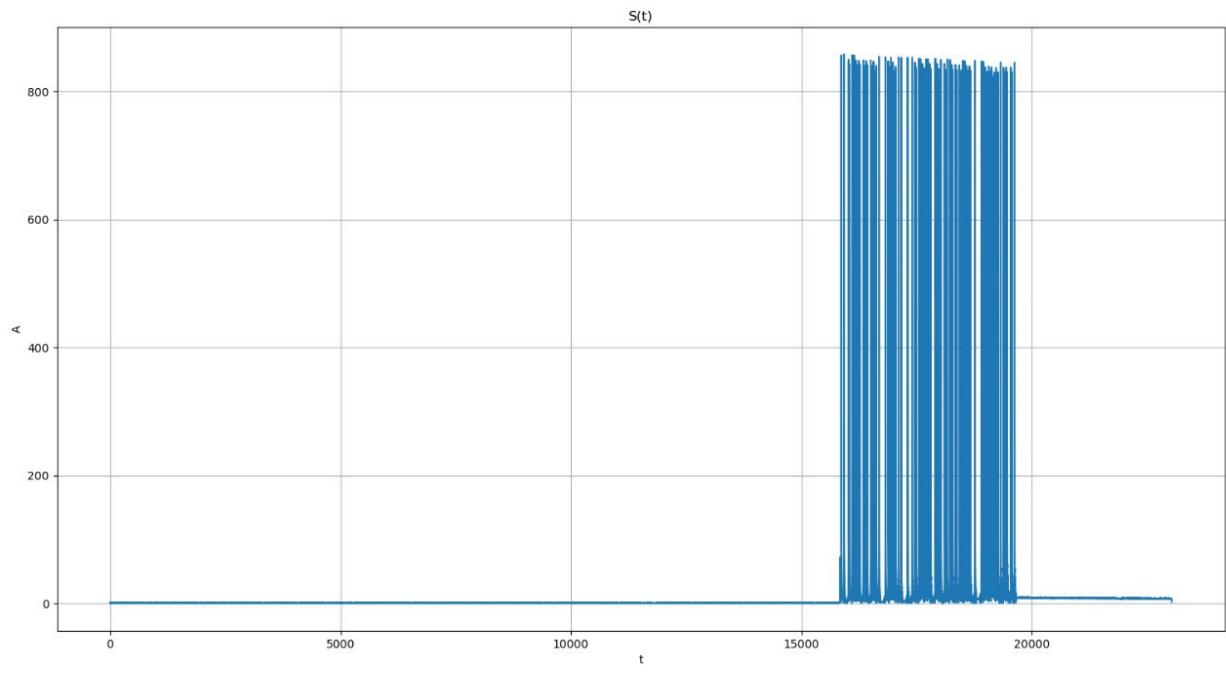


Рисунок 15 — Треугольный сигнал на приеме

## ВЫВОД

В ходе проделанной работы я реализовал сигналы разной формы: параболической и треугольной, и визуализировал их. Также изучил базовые способы модуляции: BPSK и QPSK. На языке Python построил модель, которая подробно визуализирует процесс модуляции и перемножения с несущим колебанием.

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  
Занятие №5

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

1 ВВЕДЕНИЕ .....	3
1.1 Введение .....	3
1.2 Цель .....	3
2 ЛЕКЦИЯ .....	4
2.1 Почему форма символов так важна? .....	4
2.2 Upsampling .....	4
2.3 Техническая реализация Upsampling .....	5
2.4 Формирующий фильтр .....	6
3 ПРАКТИКА .....	9
3.1 Реализация логики формирующего фильтра .....	9
4 ВЫВОД .....	13

## ВВЕДЕНИЕ

### 1.1 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: повторял значений I и Q L раз. За счёт этого получал  $I(t)$  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.

### 1.2 Цель

Более детально рассмотреть принцип работы формирующего фильтра. Познакомиться с понятием свертки. Программно реализовать алгоритм свертки между входящим сигналом и импульсной характеристикой формирующего фильтра.

# ЛЕКЦИЯ

## 2.1 Почему форма символов так важна?

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

## 2.2 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  $\frac{\text{sample}}{\text{symbol}}$  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 отвечает параметр  $f_{symb}$  (символьная скорость), который показывает скорость, с которой символы поступают из маппера. Соответственно 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  $T_s = \frac{1}{f_s}$ .

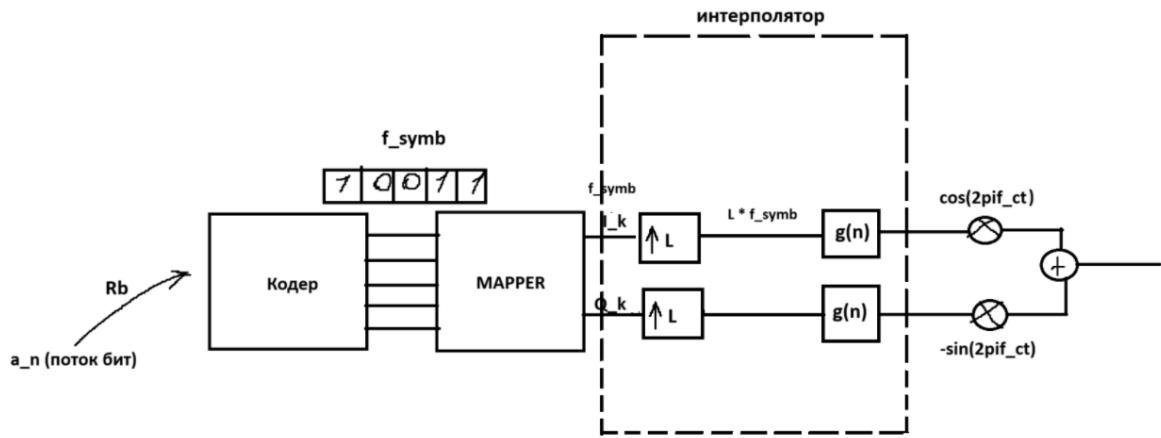


Рисунок 1 — Визуализация архитектуры передатчика

### 2.3 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.

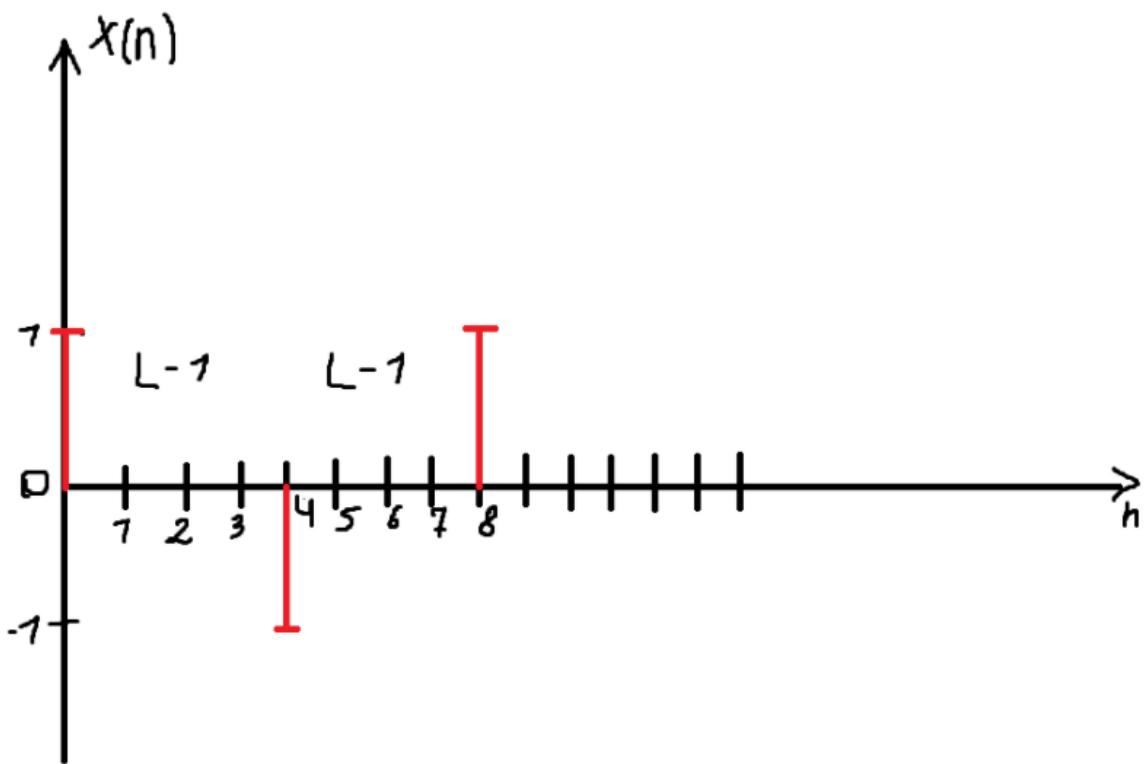


Рисунок 2 — Визуализация работы Upsampling

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

## 2.4 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.

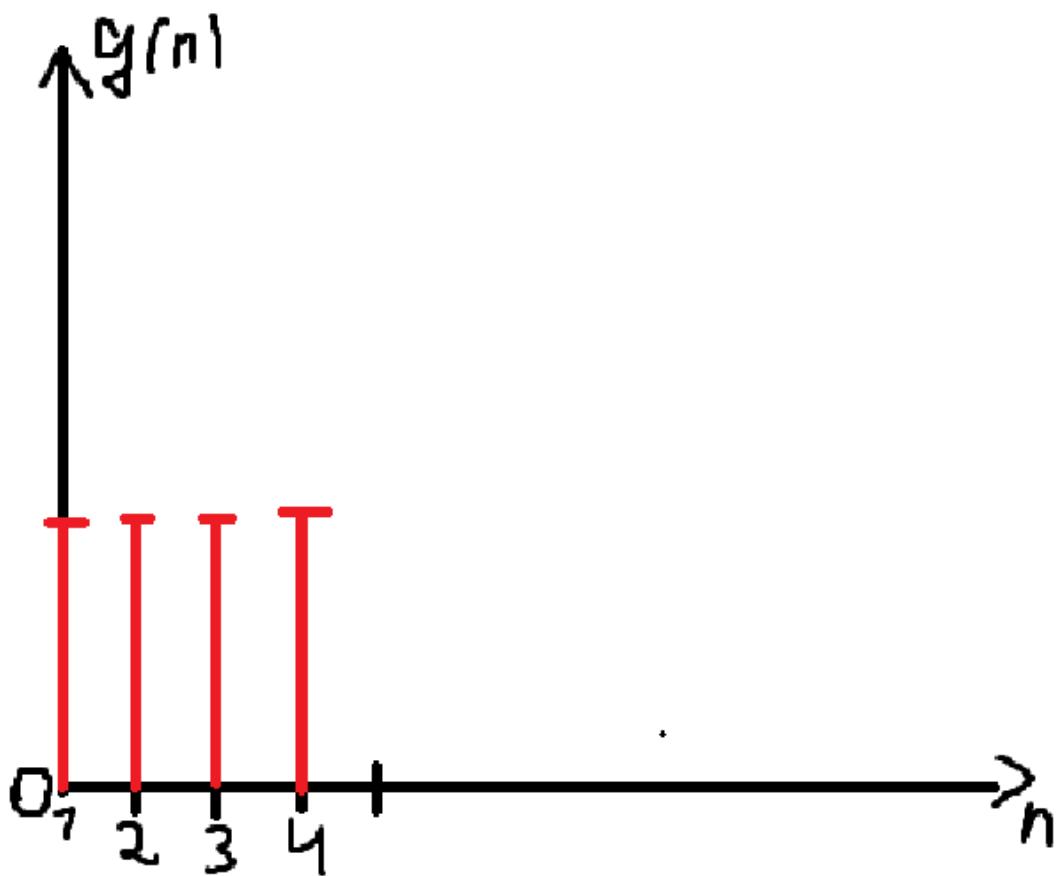


Рисунок 3 — Пример импульсной характеристики

Для иллюстрации работы фильтра произведем вычисления по формуле выше:

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

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

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

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

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

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

Визуализируем символы после выхода из фильтра:

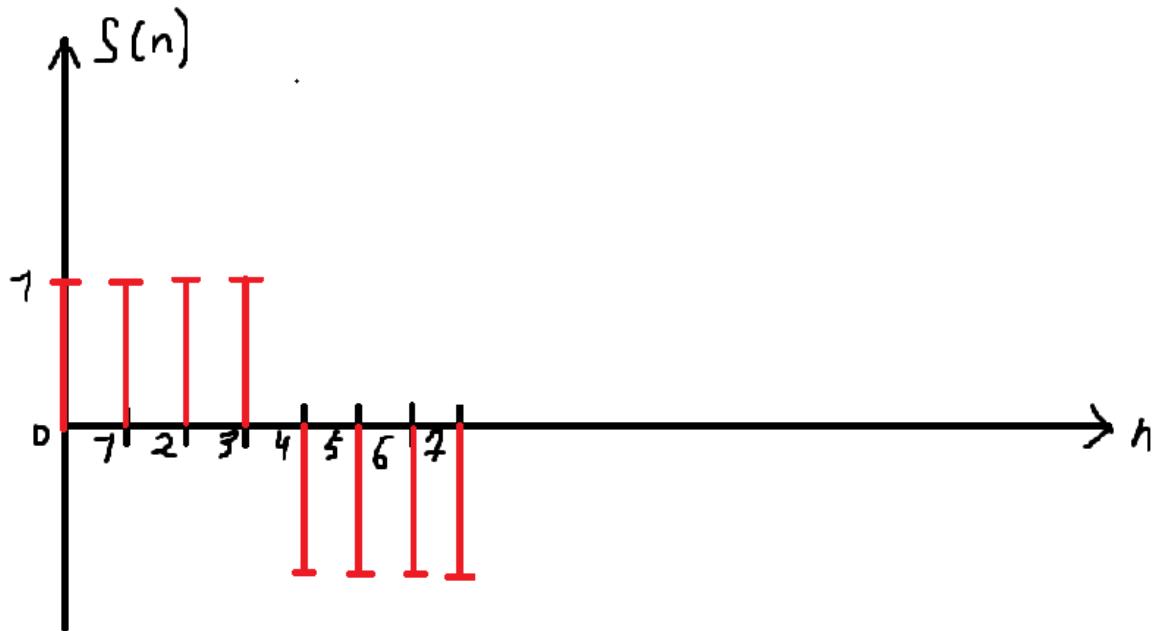


Рисунок 4 — Пример символов после выхода из фильтра

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

## ПРАКТИКА

### 3.1 Реализация логики формирующего фильтра

В теории был указан корректный метод для формирования дляящихся 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)
```

Считываем и парсим из файла символы I и Q. После этого повышаем их частоту 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.

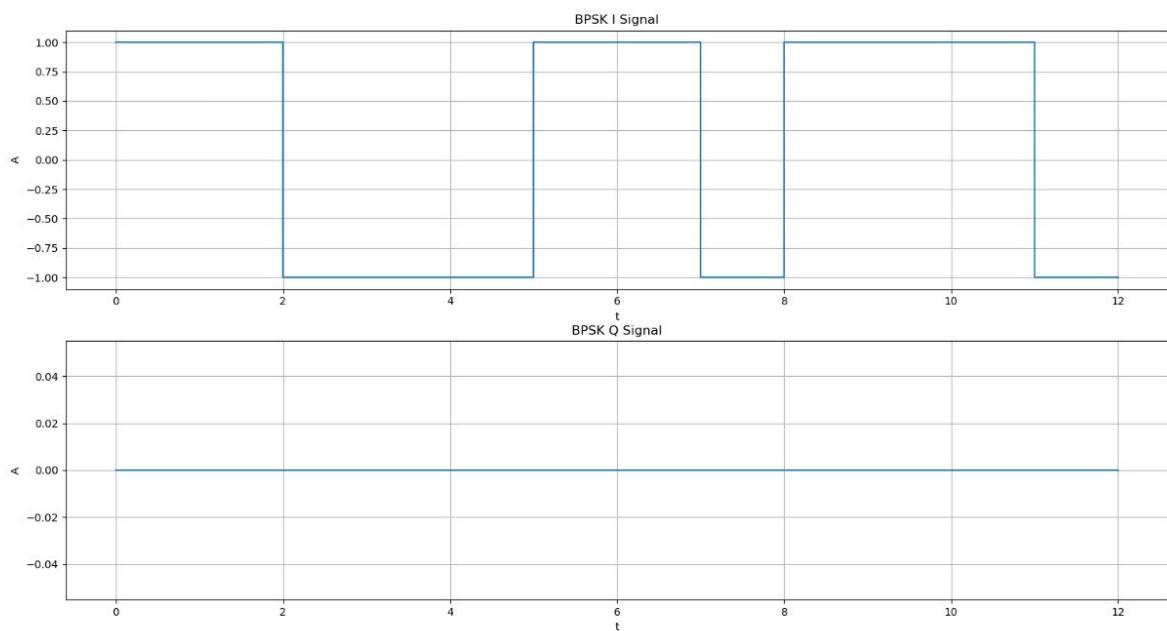


Рисунок 5 — Символы во времени (BPSK)

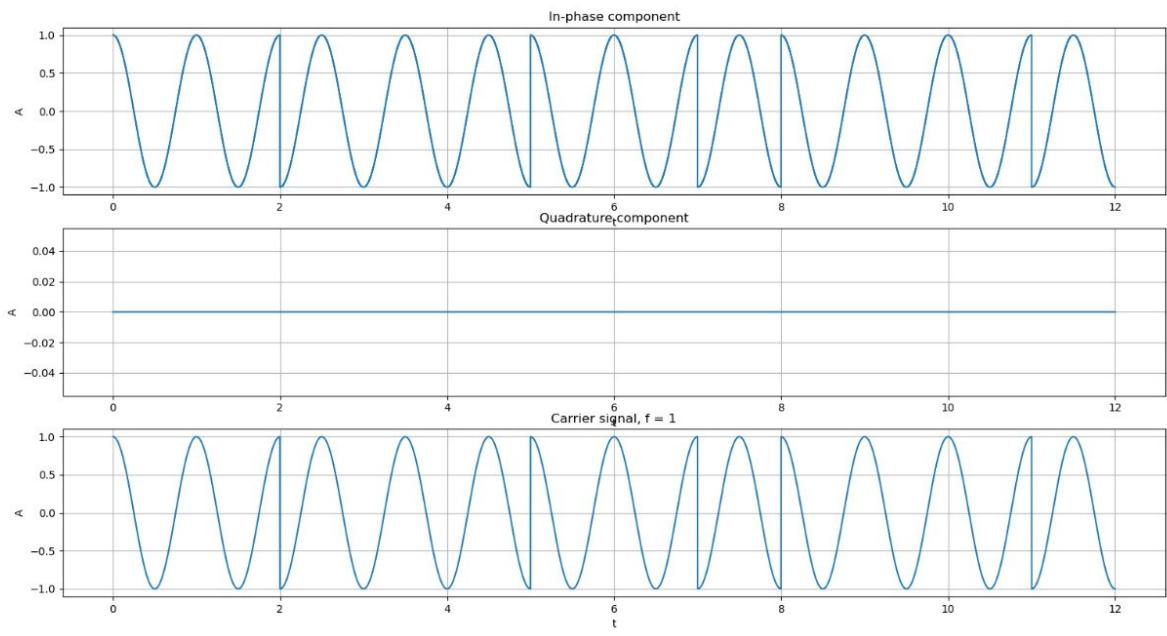


Рисунок 6 — Перемножение несущей на символы (BPSK)

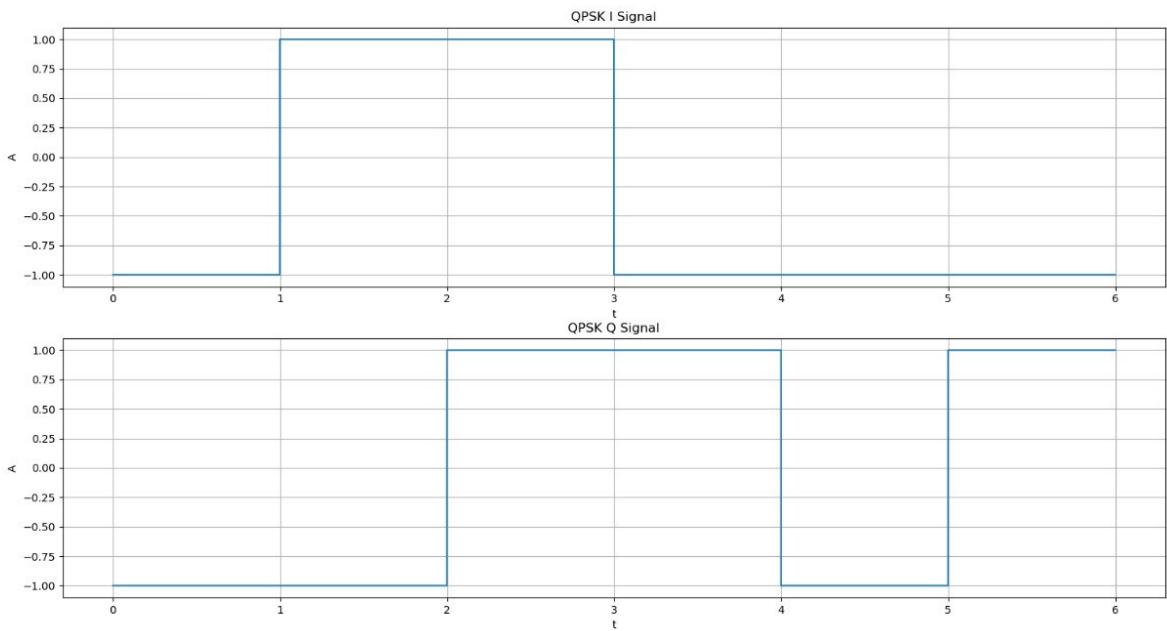


Рисунок 7 — Символы во времени (QPSK)

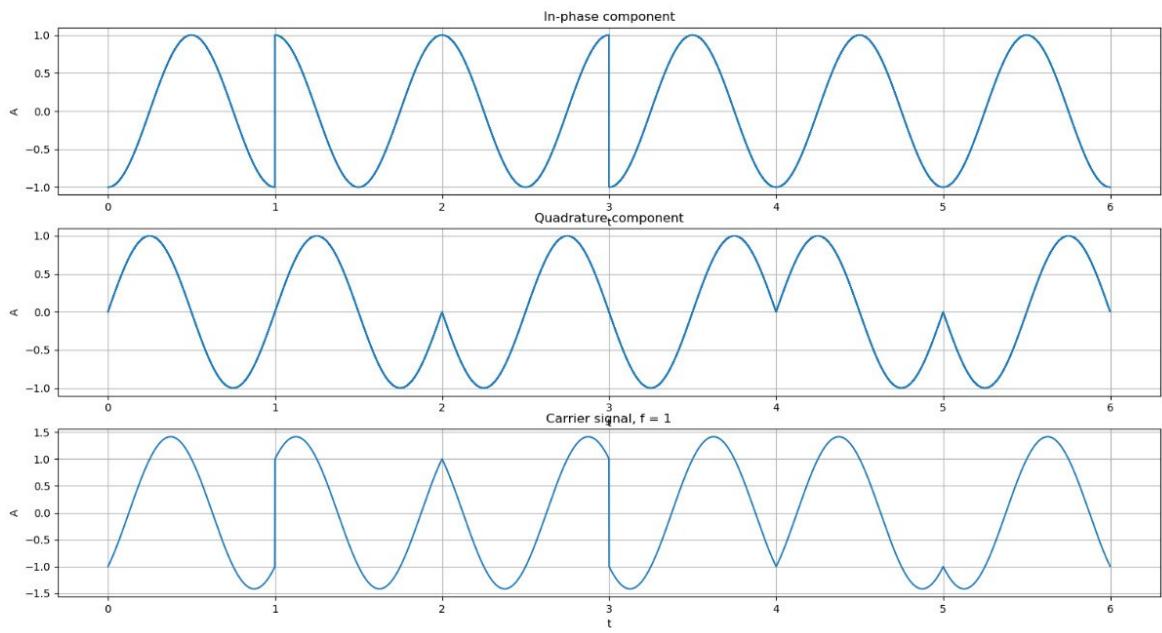


Рисунок 8 — Перемножение несущей на символы (QPSK)

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

## **ВЫВОД**

В ходе проделанной работы я более детально рассмотрел работу формирующего фильтра, познакомился с операцией свертки

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  
Занятие №6

по теме:

ИМИТАЦИЯ АНАЛОГОВОЙ ПЕРЕДАЧИ ЗВУКА И ЕГО ПРИЕМ С  
ИСПОЛЬЗОВАНИЕМ SDR. АНАЛИЗ ВЛИЯНИЯ ЧУВСТВИТЕЛЬНОСТИ  
ПРИЕМНИКА И УСИЛЕНИЯ ПЕРЕДАТЧИКА НА КАЧЕСТВО  
ПРИНЯТЫХ ОТСЧЕТОВ СИГНАЛА (СЕМПЛОВ)

Студент:

*Группа ИА-331*

*Я.А Гмыря*

Предподаватели:

*Лектор*

*Калачиков А.А*

*Семинарист*

*Ахпашев А.В*

*Семинарист*

*Попович И.А*

Новосибирск 2025 г.

## СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ .....	3
1.1 Введение .....	3
1.2 Цель .....	3
2 ПРАКТИКА .....	4
2.1 Конверторы .....	4
2.1.1 Из .mp3 в .pcm .....	4
2.1.2 Из .pcm в .mp3 .....	4
2.2 Отправка и прием мелодии .....	5
2.2.1 Чтение .pcm файла в C++ .....	5
2.3 Качество мелодии .....	6
2.3.1 Отправка и прием .....	6
2.4 Эксперимент .....	7
3 ВЫВОД .....	9

## **ВВЕДЕНИЕ**

### **1.1 Введение**

На этом занятии реализуем аналоговую передачу/прием мелодии.

### **1.2 Цель**

Реализовать конверторы аудио в рсм и рсм в аудио. Реализовать передачу/прием мелодии. Проанализировать разницу между исходной мелодией и принятой. Проанализировать влияние помех и усиления на качество передаваемой информации.

## ПРАКТИКА

### 2.1 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.

#### 2.1.1 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, умножив их на 32767.

#### 2.1.2 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.

## 2.2 Отправка и прием мелодии

### 2.2.1 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.

## 2.3 Качество мелодии

Принятая мелодия слегка ”хрипит но в целом звучит отчетливо. Если мелодия слишком тихая, то можно решить это путем изменения значений усиления на приеме/передаче

```

SoapySDRDevice_setGain(sdr, SOAPY_SDR_RX, channels[0],
    10.0); // RX
SoapySDRDevice_setGain(sdr, SOAPY_SDR_TX, channels[0],
    -90.0); // TX

```

Изменять усиление нужно аккуратно, потому что качество мелодии может стать еще хуже, т.к усиление также вносит искажения.

### 2.3.1 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.

## 2.4 Эксперимент

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

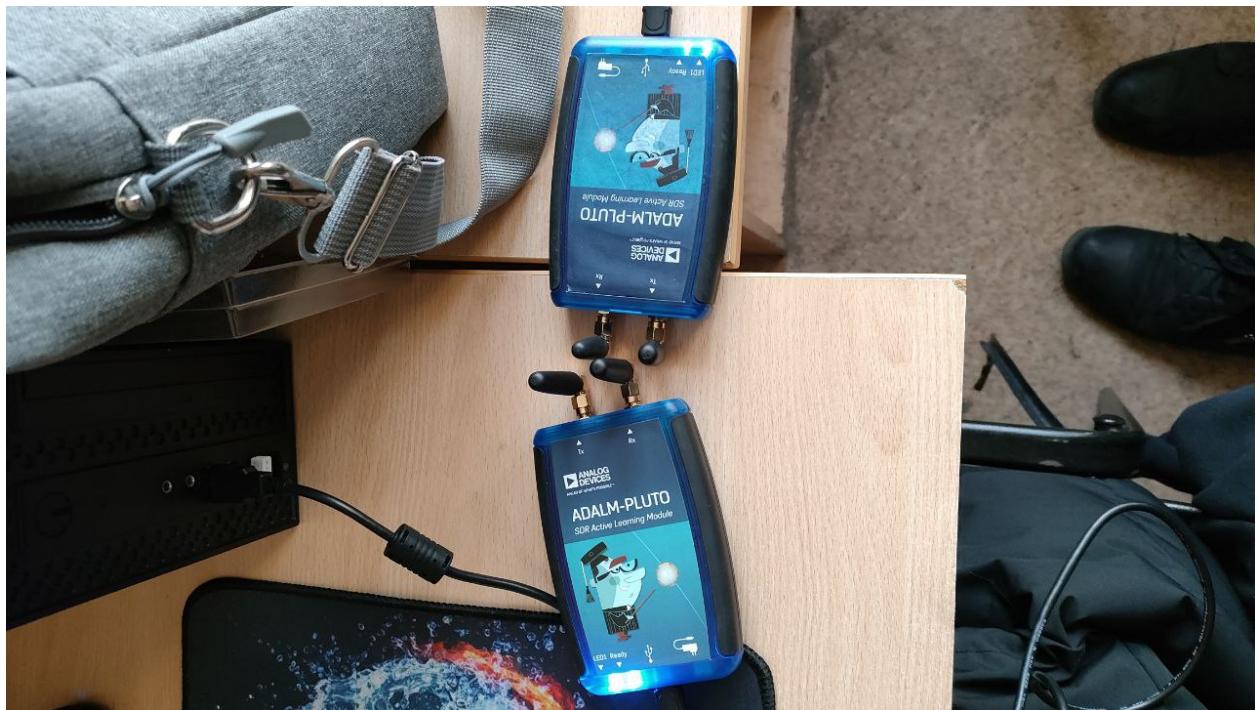


Рисунок 1 — Эксперимент

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

## **ВЫВОД**

В ходе проделанной работы я реализовал конверторы аудио в рсм и рсм в аудио. Реализовал передачу/прием мелодии. Проанализировал разницу между исходной мелодией и принятой. На практике убедился в главном недостатке аналоговых систем радиосвязи - интерференции от других устройств.

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  
Занятие №7

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

1 ВВЕДЕНИЕ .....	3
1.1 Введение .....	3
1.2 Цель .....	3
2 ПРАКТИКА .....	4
2.1 Общая структура модели .....	4
2.2 Перевод строки в биты .....	4
2.3 Модуляция .....	5
2.4 upsampling .....	6
2.5 Формирующий фильтр .....	8
2.6 Масштабирование значения семплов под SDR .....	9
2.7 Формирование семплов .....	9
2.8 Результат работы .....	10
2.8.1 BPSK модуляция .....	11
2.8.2 QPSK модуляция .....	12
3 ВЫВОД .....	13

## ВВЕДЕНИЕ

### 1.1 Введение

На одном из прошлых занятий мы реализовали программную модель на языке Python, которая формировала сигнал по битовой последовательности и подробно визуализировала весь процесс. Мы убедились в правильности построенной модели и теперь построим эту модель на языке C/C++, после чего отправим семплы в радиоканал, примим их и построим сигнальное созвездие.

### 1.2 Цель

Опираясь на программную модель формирования семплов с применением BPSK/QPSK модуляции, построенную ранее на языке Python, реализовать подобную модель на языке C/C++ с целью дальнешей передачи по радиоканалу.

## ПРАКТИКА

### 2.1 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 в соответствии с формой импульсной характеристики формирующего фильтра (на самом деле задается форма сигнала), делается это с помощью дискретной свертки входного сигнала и импульсной характеристики фильтра. На выходе фильтра получим готовые семплы, которые после масштабирования можно отправлять.

### 2.2 Перевод строки в биты

Для преобразования строки в бинарный вид на одном из прошлых занятий я написал функцию:

```

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;
}

```

## 2.3 Модуляция

```

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;
}

```

## 2.4 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 сэмплов, приходящихся на 1 символ. Таким образом мы делаем символы I,Q дляящихся 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  $I_0, Q_0, I_1, Q_1, \dots, I_n, Q_n$ , 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.

## 2.5 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 передается массив samples - семплы после блока upsampling 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.

## 2.6 Масштабирование значения семплов под SDR

На данный момент наши семплы - набор чисел из множества -1, 0, 1. Pluto SDR имеет 12-ти битный АЦП/ЦАП, т.е максимальное значение отправляемых/принимаемых семплов находится в диапазоне [-2048; 2047] (один бит уходит под знак). Помимо этого Pluto SDR странно интерпретирует семплы в переменной int16\_t - она считает за семплы только 12 первых бит (big indian) переменных. Поэтому необходимо делать битовый сдвиг на 4 знака влево (<<4). Таким образом, чтобы отмасштабировать семплы под Pluto SDR необходимо каждый семпл умножить на  $(2047 \ll 4)$ .

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;
}
```

## 2.7 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.

## 2.8 Результат работы

Считаем файл, полученный на прошлом шаге, отправим семплы, примим их и проанализируем. Принятые семплы запишем в .pcm файл и с помощью программы на Python визуализируем сигнал.

### 2.8.1 BPSK модуляция

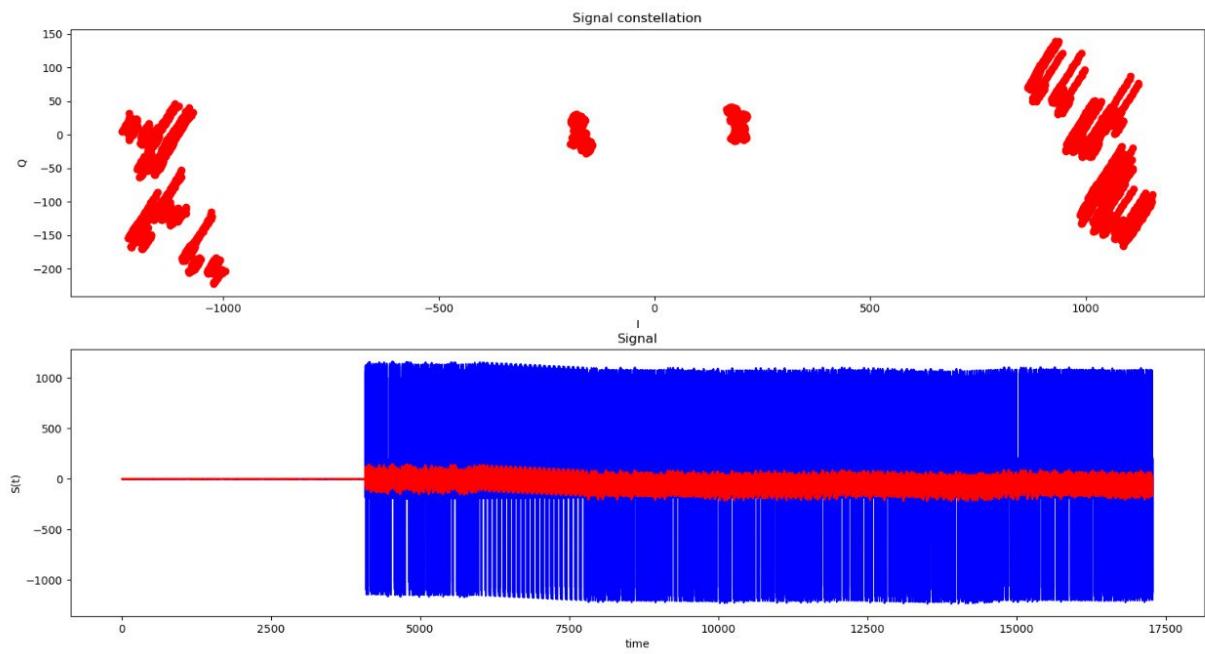


Рисунок 2 — Принятый сигнал и его сигнальное созвездие (BPSK)

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 ненулевым. 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 и отсутствия символной синхронизации.

## 2.8.2 QPSK модуляция

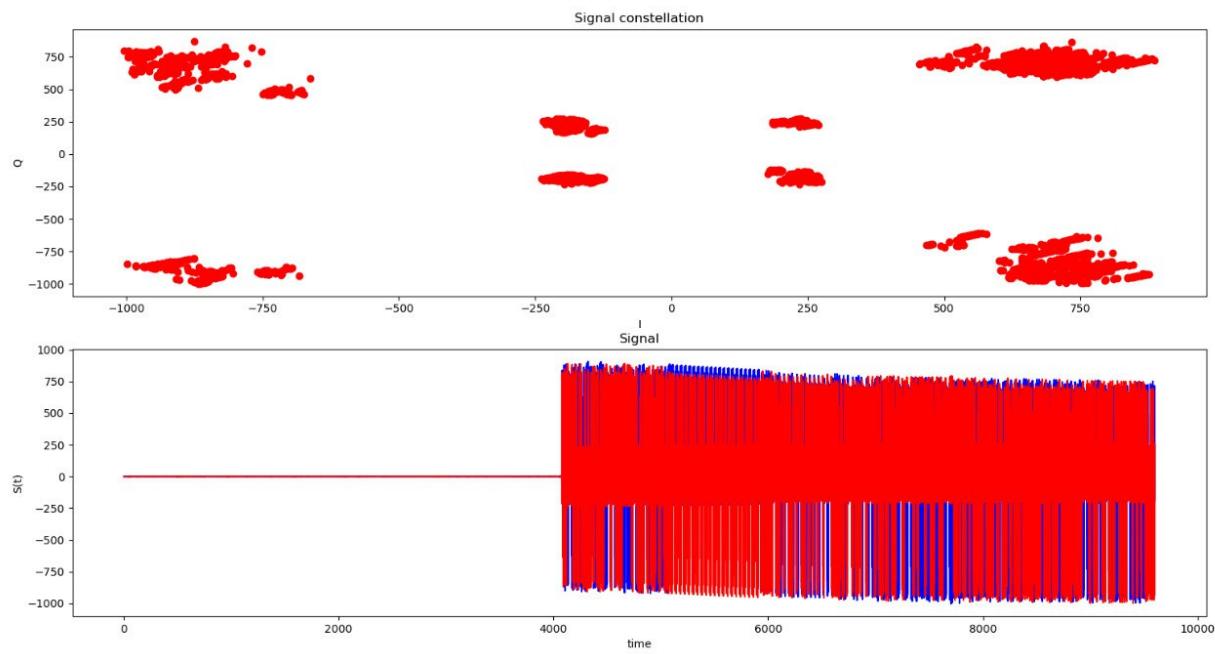


Рисунок 3 — Принятый сигнал и его сигнальное созвездие (QPSK)

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 по центру, что очень похоже на сигнальную диаграмму QPSK модуляции, но очень искаженную.

## **ВЫВОД**

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

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  
Занятие №8

по теме:  
**ДИСКРЕТНАЯ СВЕРТКА. РЕАЛИЗАЦИЯ ПРИЕМА И ПЕРЕДАЧИ  
BPSK-СИМВОЛОВ**

Студент:  
*Группа ИА-331*

*Я.А Гмыря*

Предподаватели:  
*Лектор*  
*Семинарист*  
*Семинарист*

*Калачиков А.А*  
*Ахпашев А.В*  
*Попович И.А*

Новосибирск 2025 г.

## СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ .....	3
1.1 Введение .....	3
1.2 Цель .....	3
2 ПРАКТИКА .....	4
2.1 Приподнятый косинус .....	4
2.2 Создание импульсной характеристики .....	5
2.3 Результат работы .....	6
2.3.1 Визуализация сигнала до отправки .....	6
2.3.2 Визуализация сигнала на приеме .....	7
3 ВЫВОД .....	8

## ВВЕДЕНИЕ

### 1.1 Введение

На прошлом занятии мы формировали семплы, используя BPSK/QPSK модуляцию. Мы использовали формирующий фильтр, имеющий импульсную характеристику прямоугольной формы. Прямоугольная импульсная характеристика не используется в реальном оборудовании, потому что прямоугольный сигнал будет занимать слишком много спектра. На этом занятии зададим для формирующего фильтра импульсную характеристику с формой приподнятого косинуса. Такая форма может использоваться в реальном оборудовании.

### 1.2 Цель

Осуществить формирование и передачу QPSK символов, используя формирующий фильтр, импульсная характеристика которого имеет форму приподнятого косинуса.

## ПРАКТИКА

### 2.1 Приподнятый косинус

Формула приподнятого косинуса имеет вид:

$$\frac{\cos\left(\frac{\pi\alpha t}{T}\right)}{1 - \left(\frac{2\alpha t}{T}\right)^2}$$

Здесь  $t$  - номер отсчета,  $T$  - период,  $\alpha \in [0; 1]$  - некоторый параметр.

$\alpha$  определяет полосу пропускания, занимаемую импульсом и скорость, с которой затухают хвосты импульса. При  $\alpha = 0$  самая узкая полоса пропускания и самая низкая полоса затухания. Для  $\alpha = 1$  все наоборот. Чем меньше  $\alpha$  тем сильнее уменьшается межсимвольная интерференция, но возрастает чувствительность к джиттеру. Для беспроводной передачи данных берут  $0.2 \leq \alpha \leq 0.3$ .

При помощи скрипта на Python визуализируем приподнятый косинус и выполним операцию свертки над приподнятым косинусом и прямоугольным сигналом.

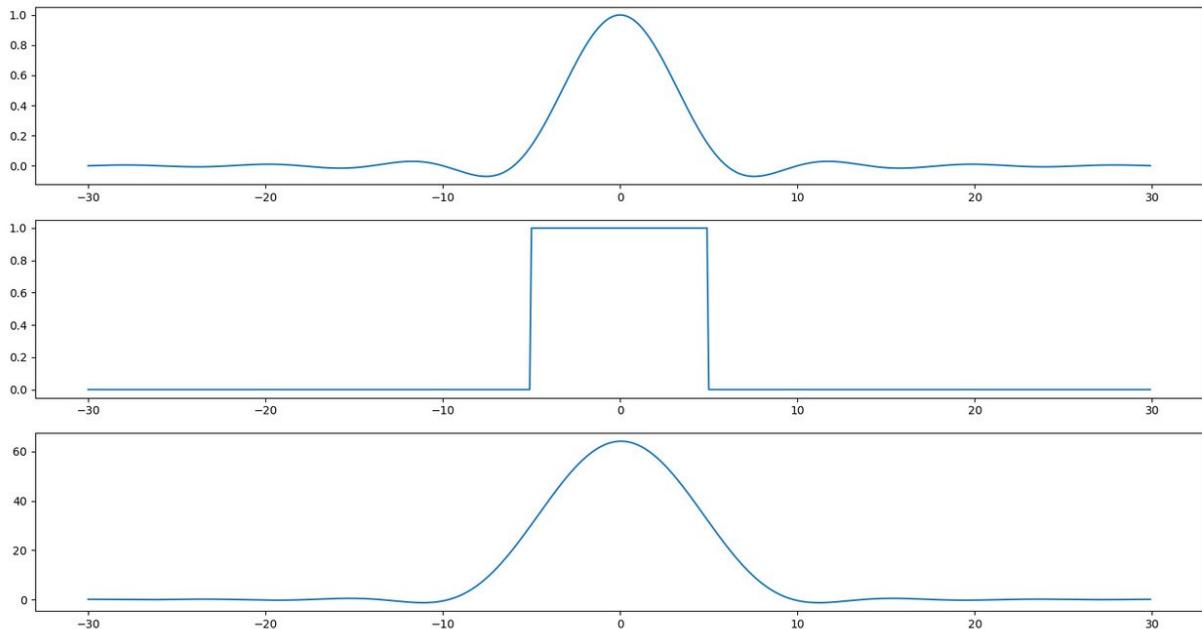


Рисунок 1 — Приподнятый косинус

Видим, что после операции свертки получили новый сигнал, который имеет свойства обоих: он имеет форму приподнятого косинуса, но в то же время растянут, как прямоугольный.

## 2.2 Создание импульсной характеристики

Будем использовать приподнятый косинус с параметри  $\alpha = 0.25$ ,  $tin[-5; 5]$  (т.к мы используем 10 семплов на символ),  $T = 0.25$ .

```
#define T 0.25
#define ALPHA 0.25

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

    //impulse response
    double g[SAMPLES_ON_BIT];

    for(int i = 0; i < SAMPLES_ON_BIT; ++i){
        int t = i - SAMPLES_ON_BIT/2;
        g[i] = cos((M_PI * ALPHA * t) / T) / (1 -
            pow((2*ALPHA*t/T), 2));
    }

    //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 = QPSK_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,
qpsk_samples);

fclose(qpsk_samples);
return 0;
}

```

В цикле заполняем импульсную характеристику значениями, посчитанными по формуле.  $t$  выбираем таким, чтобы график был симметричен относительно оси Y.

## 2.3 Результат работы

### 2.3.1 Визуализация сигнала до отправки

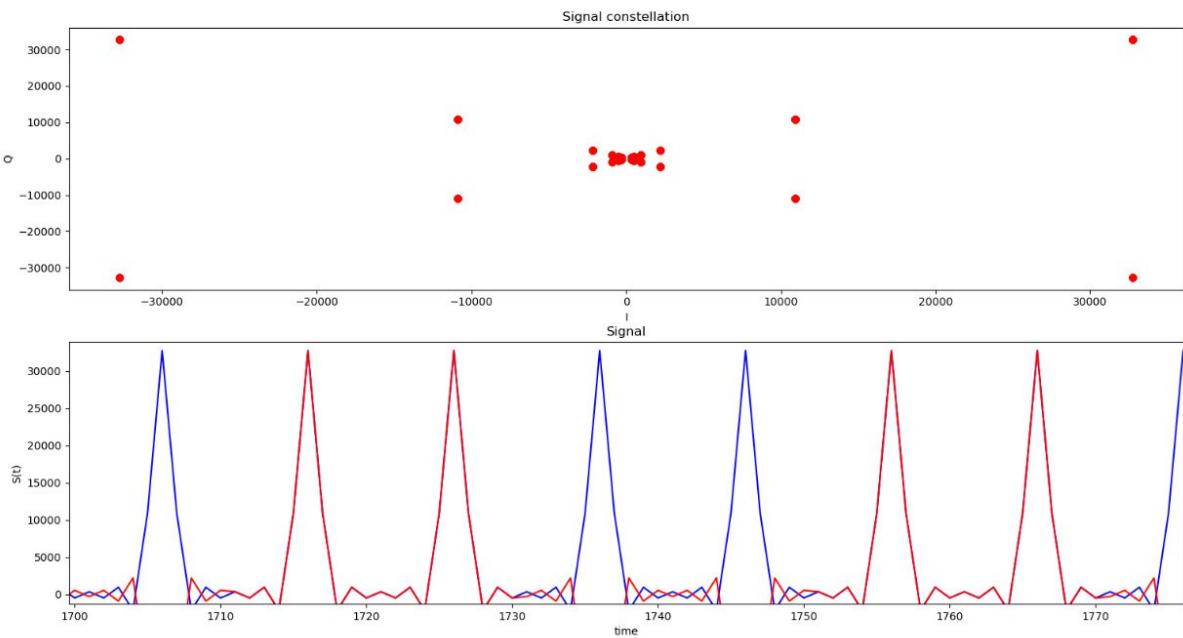


Рисунок 2 — Сигнал на передаче

Можем заметить, что сигнал больше не имеет прямоугольную форму, а имеет форму приподнятого косинуса, но из-за малого количества отсчетов для импульсной характеристики сигнал выглядит ломанным. Также можно

заметить, что изменилось сигнальное созвездие, потому что теперь сигнал принимает не только значения  $[-1, 0, 1]$ , но все равно сигнальное созвездие схоже с QPSK созвездием.

### 2.3.2 Визуализация сигнала на приеме

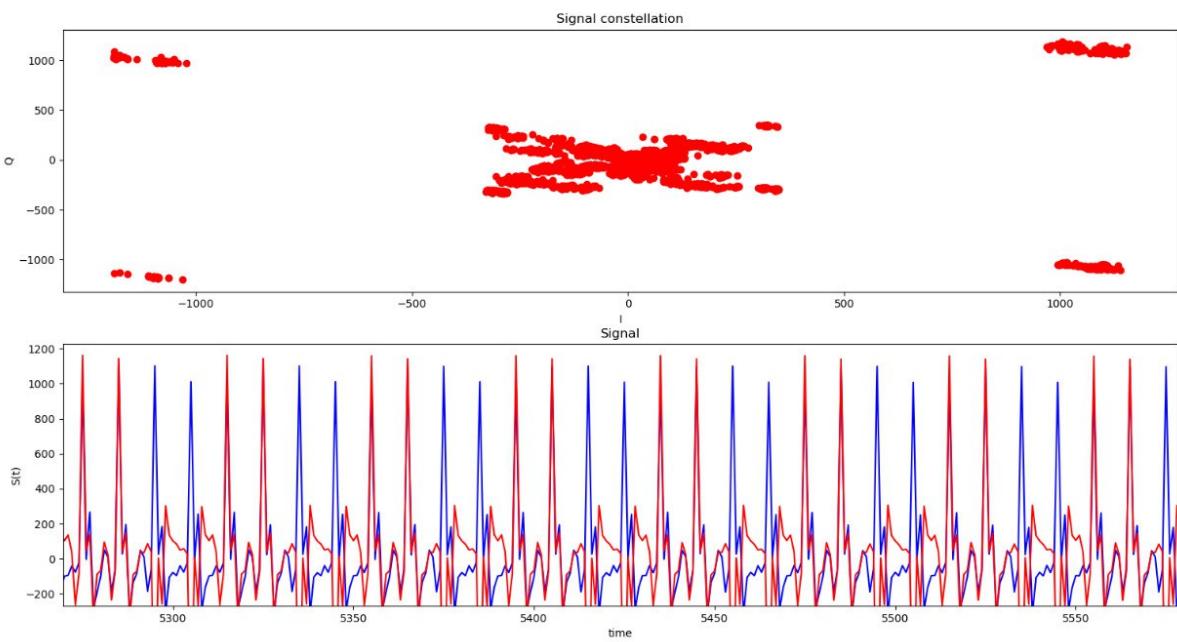


Рисунок 3 — Сигнал на приеме

Сигнал сильно искажился, но он все равно похож на приподнятый косинус. Сигнальное созвездие тоже искажилось, но остается похожим на сигнальное созвездие QPSK.

## **ВЫВОД**

В ходе работы я осуществил формирование и передачу QPSK символов, используя формирующий фильтр, импульсная характеристика которого имеет форму приподнятого косинуса.

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  
Занятие №9

по теме:

ПРИЕМ QPSK BPSK, ПРИЕМ НА СОГЛАСОВАННЫЙ ФИЛЬТР,  
ГЛАЗКОВАЯ ДИАГРАММА, ПОИСК ОПТИМАЛЬНОГО ОТСЧЕТНОГО  
ЗНАЧЕНИЯ И НЕОБХОДИМОСТЬ СИМВОЛЬНОЙ СИНХРОНИЗАЦИИ.  
ПРИЕМ И ФИЛЬТРАЦИЯ СИГНАЛА. ПРЯМОУГОЛЬНЫЙ И  
ПРИПОДНЯТЫЙ КОСИНУС

Студент:

*Группа ИА-331*

*Я.А Гмыря*

Предподаватели:

*Лектор*

*Калачиков А.А*

*Семинарист*

*Ахпашев А.В*

*Семинарист*

*Попович И.А*

Новосибирск 2025 г.

## СОДЕРЖАНИЕ

1	ВВЕДЕНИЕ .....	3
1.1	Введение .....	3
1.2	Цель .....	3
2	ЛЕКЦИЯ .....	4
2.1	Архитектура приемника .....	4
2.1.1	Радиоканал и SDR .....	4
2.1.2	Matched filter .....	4
2.1.3	Downsampling .....	5
2.1.4	Quantization .....	6
2.1.5	Demapper .....	7
2.1.6	Decoder .....	7
3	ПРАКТИКА .....	8
3.1	Match filter .....	8
3.2	Downsampling .....	8
3.3	downscale .....	9
3.4	Quantization .....	10
3.5	Demapper .....	11
3.6	Влияние сдвига на созвездие .....	13
3.7	Тест .....	13
3.8	Результат работы .....	15
4	ВЫВОД .....	16

## ВВЕДЕНИЕ

### 1.1 Введение

До этого занятия мы рассматривали логику работы передатчика. Мы узнали, как передатчик формирует сигнал, как осуществить отправку данных с SDR с помощью C/C++ и библиотеки SoapySDR. С этого занятия будет рассматриваться приемник. В ходе этих занятий мы реализуем логику работы приемной стороны, символьную синхронизацию, чтобы извлечь из принятых семплов передаваемые биты. Конкретно на этом занятии реализуем логику по обработке принимаемого сигнала без синхронизации.

### 1.2 Цель

Реализовать логику работы применой стороны на языке C/C++. Выяснить, как выбор семпла в качестве значения символа влияет на созвездие. Убедиться в необходимости символьной синхронизации.

# ЛЕКЦИЯ

## 2.1 Архитектура приемника

Рассмотрим архитектуру простого приемника

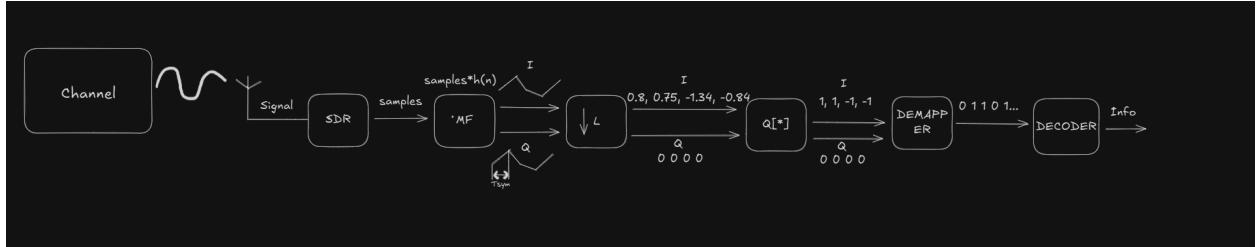


Рисунок 1 — Архитектура простого приемника

### 2.1.1 Радиоканал и SDR

Из радиоканала сигнал попадает на антенну, где преобразуется в электрический сигнал и попадает в SDR, которая дискретизирует сигнал, производит какие-то дополнительные манипуляции и на выходе выдает отсчеты сигнала.

### 2.1.2 Matched filter

Далее семплы попадают на согласованный фильтр (matched filter), который позволяет увеличить SNR (Signal Noise Ration), т.е выделить полезный сигнал из общего сигнала с шумом. Импульсная характеристика фильтра должна согласовываться с импульсной характеристикой фильтра в передатчике. Допустим, передатчик имеет импульсную характеристику  $g(n)$ , тогда на приемной стороне ИХ фильтра будет следующей:

$$h(n) = K g * (L - n)$$

ИХ согласованного фильтра на приемнике - ИХ, которая является комплексно-сопряженной (зеркальной) с ИХ передатчика и сдвинутой на  $L$  отсчетов, где  $L$  - кол-во семплов на символ.  $K$  - коэффициент.

Мы используем прямоугольную ИХ на передатчике, поэтому на приеме эта ИХ никак не изменится.

На выходе фильтра получаем пилюобразный график:

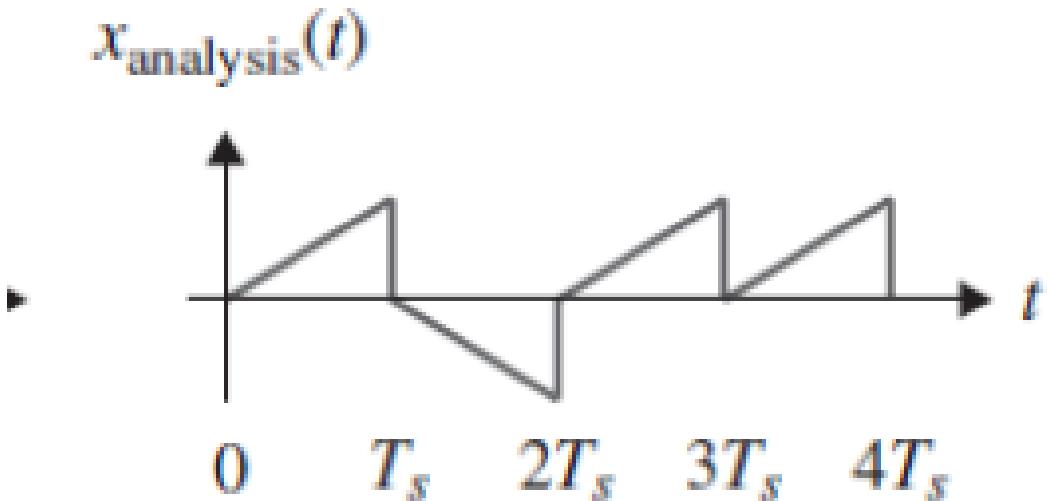


Рисунок 2 — Выход matched filter

### 2.1.3 Downsampling

На стороне передатчика мы делали операцию upsampling, которая повышала частоту дискретизации. Это позволяло нам сделать так, чтобы на 1 символ приходилось 10 семплов. На стороне приема делается обратная операция - downsampling, т.е нам нужно из каждого 10 семплов выбрать один такой, который наиболее точно описывает переданный символ. Но какой семпл нужно выбирать? После прохождения через фильтр каждый 10-ый семпл будет иметь наибольший SNR, именно его и нужно выбирать. Но с какого семпла нам нужно начинать выполнять эту операцию? Ведь мы можем "промахнуться" и начать выбирать не те семплы. На этот вопрос отвечает символьная синхронизация, но это тема последующих занятий. На этом занятии будет начинать downsampling с первого семпла.

## 2.1.4 Quantization

Проходя через радиоканал, сигнал искажается и принятые символы будут искаженными. Если на передаче отправлялись символы {1,0,1,0,-1,0}, то на приеме можем получить {0.74,-0.32,1.24,0.2,-1.35,0.01}. Четко определить, какой последовательности битов соответствуют символы невозможно, поэтому перед тем, как начать этап преобразования символов в биты, нужно понять, к какой из возможных точек созвездия эти точки находятся ближе всего, и как-бы округлить координаты точек так, чтобы они точно соответствовали координатам истинных точек созвездия. Чтобы оценивать, к какой истинной точке на созвездии ближе всего находится точка с искаженными координатами, я буду вычислять расстояние между искаженной точкой и всеми истинными точками.

Пример вычисления расстояния между точками сигнального созвездия BPSK:

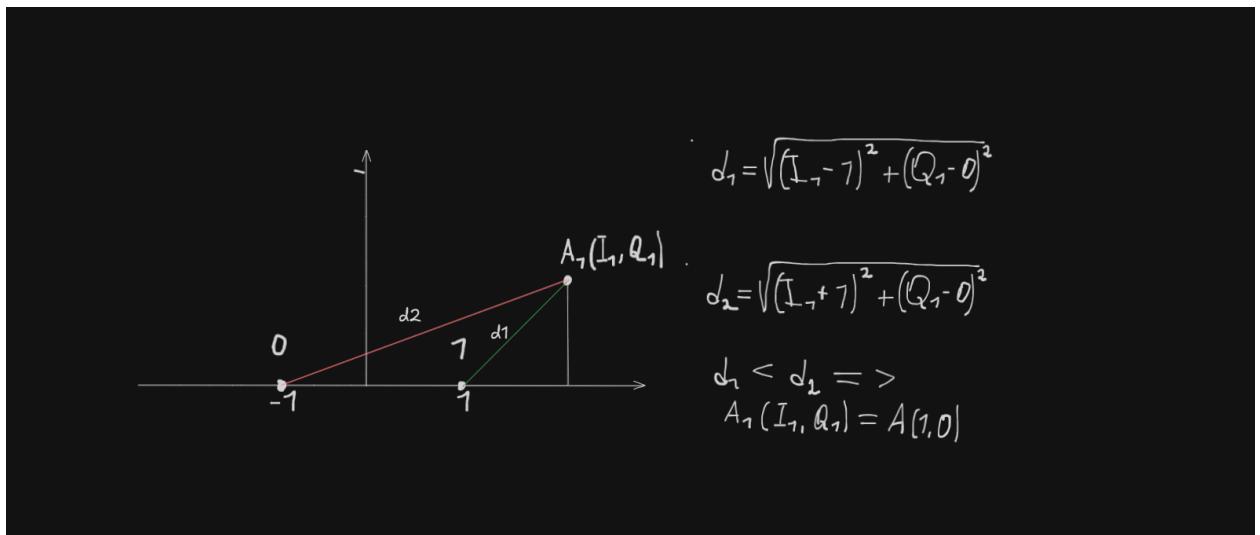


Рисунок 3 — Расстояние между точками созвездия

В созвездии BPSK имеются 2 точки с координатами  $(-1,0)$  и  $(1,0)$ , точка А имеет искаженные координаты. По теореме Пифагора можно найти расстояния между точками. Точка А ближе находится к точке  $(1,0)$ , поэтому логично предположить, что передавалась единица, а не ноль.

### 2.1.5 Demapper

На передаче мы подавали биты на специальное устройство под названием Mapper, которое ставило в соответствие последовательности бит символов. Demapper делает обратную задачу - символам сопоставляет биты.

### 2.1.6 Decoder

На стороне передатчика передается текст, поэтому на стороне приема нужен блок, который преобразовывал бы биты в текст. Этой задачей будет заниматься блок Decoder.

## ПРАКТИКА

### 3.1 Match filter

```

int16_t* match_filter(int16_t* samples, int samples_count, int
L, double* h, int* size){
    *size = samples_count;
    int16_t* new_samples = (int16_t*)malloc(samples_count *
sizeof(int16_t));

    //convolve
    for(int n = 0; n < samples_count; ++n){
        int16_t tmp = 0;
        for(int m = 0; m < L; ++m){
            if (n - m >= 0){
                tmp += samples[n-m]*h[m];
            }
        }
        new_samples[n] = tmp;
    }

    return new_samples;
}

```

Функция принимает массив семплов, размер этого массива, число L, которое показывает, сколько семплов приходится на бит, ИХ h, переменную size, в которой вернется размер массива на выходе. Функция выполняет дискретную свертку входного сигнала с ИХ.

### 3.2 Downsampling

```

int16_t* down_sampling(int16_t* samples, int samples_count, int
L, int start){
    int symbols_size = samples_count / L - start + 1;

    int16_t* symbols = (int16_t*)malloc(symbols_size *
sizeof(int16_t));

```

```

    for(int i = start; i < symbols_size; ++i){
        symbols[i] = samples[i*L];
    }

    return symbols;
}

```

Функция принимает массив исходных семплов, размер массива, число L, которое указывает на то, с каким периодом мы будем делать downsampling (сохраняем только каждое L-ое значение). Параметр start - индекс, с которого функция начнет брать каждый L-ый элемент. В функции итерируемся по новому массиву и заполняем его каждым L-ым значением.

### 3.3 downscale

```

double max(int16_t* arr, int arr_size){
    double max = __DBL_MAX__;
    for(int i = 0; i < arr_size; ++i){
        if(arr[i] > max){
            max = arr[i];
        }
    }

    return max;
}

double* downscale(int16_t* samples, int samples_count){
    double* down_scale_samples = (double*)malloc(samples_count *
        sizeof(double));

    double max_value = max(samples, samples_count);

    for(int i = 0; i < samples_count; i+=2){
        down_scale_samples[i] = samples[i] / max_value;
    }

    return down_scale_samples;
}

```

Для того, чтобы дальше работать с символами, нужно нормализовать их, т.е привести к значениям, которые будут близки к значениям из отрезка [-1;1]. В качестве нормирующего коэффициента будет использоваться символ с наибольшим значением. Функция max используется для поиска символа с максимальным значением. В функции downscale итерируемся по исходному массиву символов и делим каждый на максимальное значение среди символов.

### 3.4 Quantization

```

int16_t* BPSK_quantizater(double* symbols, int symbols_size){
    int16_t* rounded_symbols = (int16_t*)malloc(symbols_size *
        sizeof(int16_t));

    int point0[] = {-1, 0};
    int point1[] = {1, 0};

    double distance0 = 0, distance1 = 0;

    double dx = 0, dy = 0;

    for(int i = 0; i < symbols_size; i+=2){
        dx = point0[0]-symbols[i];
        dy = point0[1]-symbols[i+1];

        distance0 = sqrt(dx*dx + dy*dy);

        dx = point1[0]-symbols[i];
        dy = point1[1]-symbols[i+1];

        distance1 = sqrt(dx*dx + dy*dy);

        if(distance0 > distance1){
            rounded_symbols[i] = 1;
            rounded_symbols[i+1] = 0;
        }else{
            rounded_symbols[i] = -1;
            rounded_symbols[i+1] = 0;
        }
    }
}

```

```

    return rounded_symbols;
}

```

Функция принимает массив искаженных символов и размер массива. В функции в виде массивов размером в 2 элемента задаются истинные точки, которые есть в BPSK созвездии. Далее в цикле перебираем символы (точки на созвездии) и вычисляем расстояние между искаженной точкой и заданными истинными точками. В результирующий массив запишем координаты истинной точки, с которой искаженная точка имеет наименьшее расстояние.

### 3.5 Demapper

```

int8_t* BPSK_demodulator(int16_t* symbols, int symbols_size){
    int bits_size = symbols_size/2;

    int8_t* bits = (int8_t*)malloc(bits_size * sizeof(int8_t));

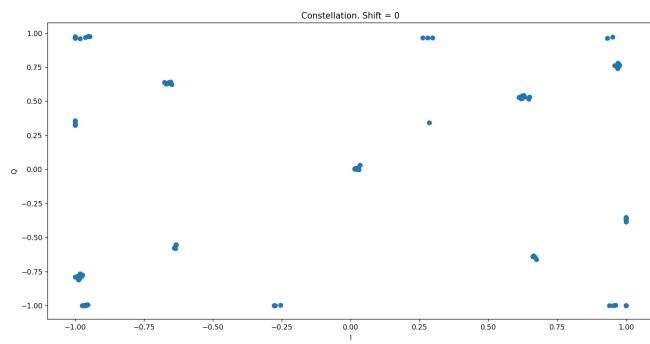
    int j = 0;

    for(int i = 0; i < symbols_size; i+=2){
        if(symbols[i] == 1){
            bits[j++] = 1;
        } else{
            bits[j++] = 0;
        }
    }

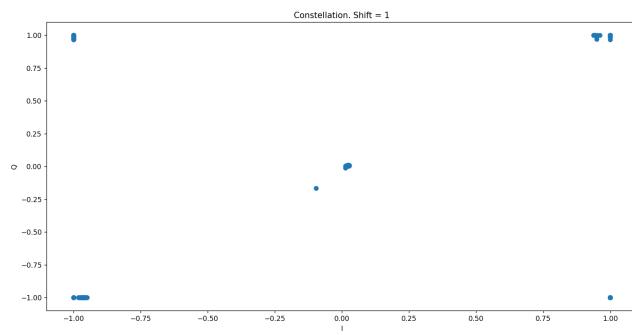
    return bits;
}

```

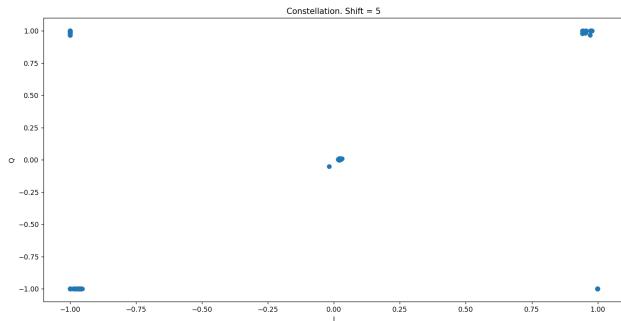
Функция принимает массив округленных символов и размер массива. Функция итерируется по массиву и с помощью условий выбирает, какой бит сопоставить символу.



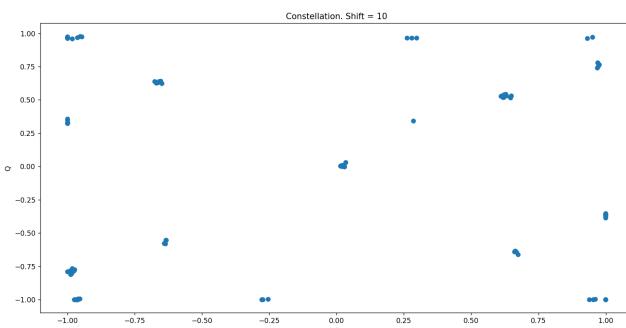
(a) Shift = 0



(б) Shift = 1



(в) Shift = 5



(г) Shift = 10

Рисунок 4 — Влияние сдвига семплов на созвездие

### 3.6 Влияние сдвига на созвездие

Можем заметить, что при нулевом сдвиге созвездие "распалось" т.е мы начали брать отсчеты неправильно. При сдвиге равном 1 созвездие "собралось" во что-то похожее на QPSK созвездие. При сдвиге равном 5 созвездие стало еще более похожим на QPSK созвездие. На сдвиге равном 10 созвездие снова "распалось". Этот процесс будет повторяться периодически на каждом символе. Символьная синхронизация позволяет автоматизировать этот процесс и подобрать оптимальный момент для начала взятия отсчетов.

### 3.7 Тест

Для проверки корректности работы приемника сгенерируем семплы на стороне передатчика и запишем в файл, а на приемной стороне считаем файл и попытаемся превратить семплы в информацию, которую отправлял передатчик. Попытаемся передать символ "Н"

```
int main(){

    size_t samples_count = 0;

    int16_t* samples = read_pcm("bpsk_samples.pcm",
        &samples_count);

    if(samples == nullptr){
        printf("File not found!\n");
        exit(1);
    }

    printf("samples:\n");

    for(int i = 0; i < samples_count; ++i){
        printf("%d ", samples[i]);
    }

    printf("\n");

    int h[] = {1,1,1,1,1,1,1,1,1,1};

    int conv_size = 0;
```

```

int* conv = match_filter(samples, samples_count, 10, h,
&conv_size);

printf("After convolve:\n");

for(int i = 0; i < conv_size; ++i){
    printf("%d ", conv[i]);
}

printf("\n");

int* ds_samples = down_sampling(conv, samples_count, 10, 0);

printf("Samples after down sampling:\n");

for(int i = 0; i < conv_size/10; ++i){
    printf("%d ", ds_samples[i]);
}

printf("\n");

double* down_scale_samples = down_scaler(ds_samples,
samples_count/10);

printf("Samples after downscaling:\n");

for(int i = 0; i < conv_size/10; ++i){
    printf("%f ", down_scale_samples[i]);
}

printf("\n");

int16_t* q_samples = BPSK_quantizater(down_scale_samples,
samples_count/10);

printf("rounded symbols:\n");

for(int i = 0; i < samples_count/10; ++i){
    printf("%d ", q_samples[i]);
}

```

```
printf("\n");

int8_t* bits1 = BPSK_demodulator(q_samples,
    samples_count/10);

printf("bits:\n");

for(int i = 0; i < samples_count/20; ++i){
    printf("%d ", bits1[i]);
}

printf("\n");

int str_len = 0;
char* str = btos(bits1, samples_count/20, &str_len);

printf("string: %s\n", str);

return 0;
}
```

### 3.8 Результат работы

Рисунок 5 — Процесс обработки принятого сигнала

Приемнику удалось верно преобразовать семплы в текст, значит, удалось верно реализовать логику работы приемника.

## **ВЫВОД**

В ходе проделанной работы я реализовал логику работы применой стороны на языке C/C++. Выяснил, как выбор семпла в качестве значения символа влияет на созвездие. Убедился в необходимости символьной синхронизации.

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  
Занятие №10

по теме:

СИМВОЛЬНАЯ СИНХРОНИЗАЦИЯ, ДЕТЕКТОР ВРЕМЕННОЙ  
ОШИБКИ, СХЕМА ГАРДНЕРА. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ  
ДЕТЕКТОРА ВРЕМЕННОЙ ОШИБКИ (СИНХРОНИЗАЦИЯ  
ПРИЕМНИКА И ПЕРЕДАТЧИКА) НА 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

1	ВВЕДЕНИЕ .....	3
1.1	Введение .....	3
1.2	Цель .....	3
2	ЛЕКЦИЯ .....	4
2.1	Символьная синхронизация .....	4
2.2	Архитектура приемника .....	4
2.3	Принцип работы символьной синхронизации .....	4
2.4	Timing correction .....	4
2.5	TED .....	5
2.6	Loop filter .....	6
3	ВЫВОД .....	9

## **ВВЕДЕНИЕ**

### **1.1 Введение**

На прошлых занятиях мы реализовали логику работы передатчика и частично логику работы приемника. В нашей системе на стороне приемника не хватает только символьной синхронизации. На этом и последующих занятиях будем заниматься реализацией символьной синхронизации, которая позволит нам автоматически выбирать оптимальный момент взятия отсчетов.

### **1.2 Цель**

Узнать, что такое символьная синхронизация, понять ее назначение. Изучить принцип работы схемы Гарднера - способа реализации символьной синхронизации.

# ЛЕКЦИЯ

## 2.1 Символьная синхронизация

Символьная синхронизация - процесс поиска (определения) оптимального времени взятия отсчетов принятого сигнала. Она определяет задержку (сдвиг отсчетов сигнала) перед процедурой downsampling, которая позволяет "попасть" в оптимальные моменты взятия отсчетов.

## 2.2 Архитектура приемника

Доработаем схему приемника, которую мы сделали на прошлом занятии, добавив блоки, выполняющие символьную синхронизацию.

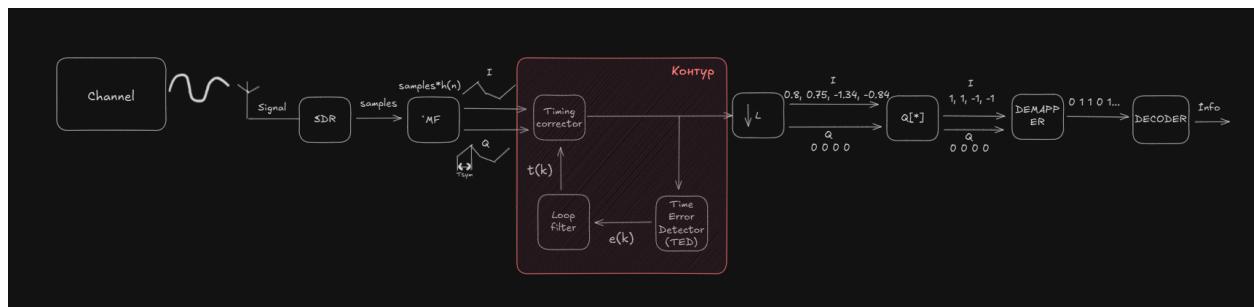


Рисунок 1 — Архитектура приемника

## 2.3 Принцип работы символьной синхронизации

Между согласованным фильтром и процедурой downsampling добавили контур, в котором будет выполняться символьная синхронизация.

## 2.4 Timing correction

Семплы, выходящие из согласованного фильтра поступают на блок Timing corrector, где скорректируются таким образом, чтобы правильно начать выбирать символы.

## 2.5 TED

Принцип работы символьной синхронизации основан на формировании сигнала ошибки  $e(k)$  при принятых отсчетам и коррекции временной задержки.

Задача заключается в том, чтобы найти моменты времени, когда производная от сигналов равна 0 (моменты смены знака). Будем использовать метод Гарднера. Он основан на разнице между отсчетами с интервалом в 1 символ. Рассчитать  $e(k)$  по схеме Гарднера можно следующим образом:

$$e(k) = (s[k] - s[k - 2]) * s[k - 1]$$

Здесь  $s[k]$  - определенный отсчет сигнала.

Визуализация метода:

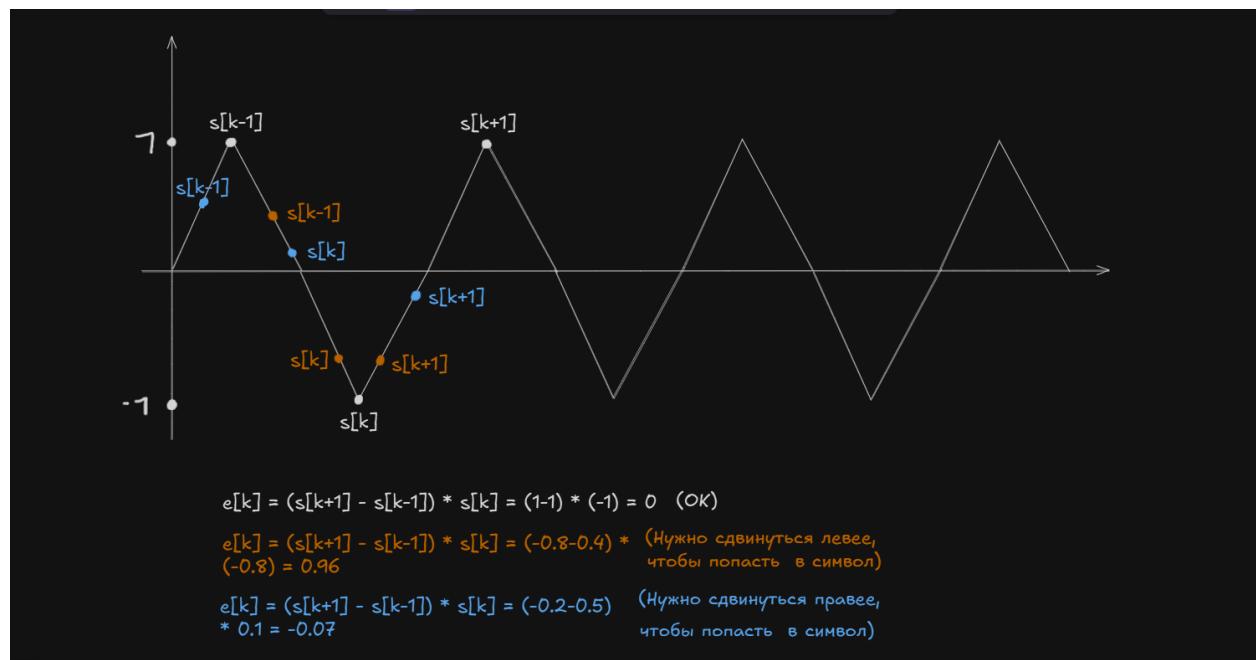


Рисунок 2 — Расчет TED

Если визуализировать коэффициенты TED, то должна получиться кривая, схожая с  $\sin$  или  $\cos$

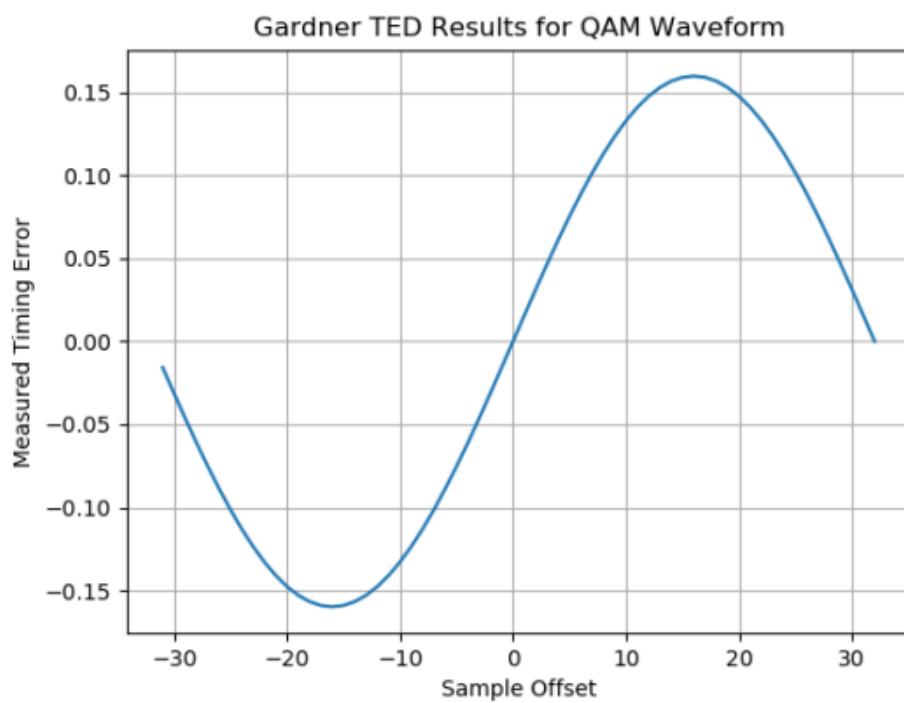


Рисунок 3 — График TED

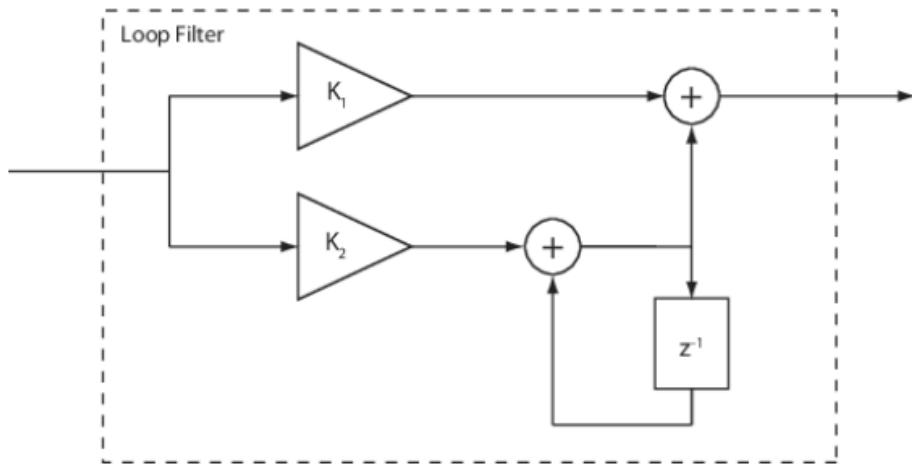
## 2.6 Loop filter

Этот блок по определенному алгоритму усредняет (сглаживает) TED и вычисляет сдвиг.

Схема работы Loop filter

### 3.2 Фильтр контура

Фильтр выполнен в виде пропорционально-интегрирующего звена по схеме



Параметры фильтра

Рисунок 4 — Схема Loop filter

Формулы для вычисления K1, K2

$$K_1 = \frac{-4\zeta\theta}{(1 + 2\zeta\theta + \theta^2)K_p}$$

and

$$K_2 = \frac{-4\theta^2}{(1 + 2\zeta\theta + \theta^2)K_p}.$$

The interim term,  $\theta$ , is given by

$$\theta = \frac{\frac{B_n T_s}{N_{sps}}}{\zeta + \frac{1}{4\zeta}},$$

where:

- $N_{sps}$  is the number of samples per symbol.
- $\zeta$  is the damping factor.
- $B_n T_s$  is the loop bandwidth ( $B_n$ ) normalized to the symbol rate ( $T_s$ ).
- $K_p$  is the detector gain.

Рисунок 5 — Формулы

В вычислениях будем использовать следующие константы (получены экспериментальным путем)

$B_n T_s = 0.01$
$N_{sps} = 10$
$K_p = 0.002$
$\zeta = \sqrt{2} / 2$

## **ВЫВОД**

В ходе работы я узнал, что такое символьная синхронизация, понял ее назначение. Изучил принцип работы схемы Гарднера.