

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

09.03.01 Информатика и вычислительная техника
(направление подготовки/специальность)
Программное обеспечение систем мобильной связи
(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 Шкляев Денис Викторович
группа ИА-331

_____ / Шкляев Д. В. /
«____» ____ 202__г. *(подпись)* *(ФИО)*

Проверил¹
Руководитель практики от профильной
организации _____ / Андреев А. В. /
«____» ____ 202__г. *(подпись)* *(ФИО)*

Проверил:
Руководитель практики от СибГУТИ _____ / Брагин К. И. /
«____» ____ 202__г. *(подпись)* *(ФИО)*

«____» ____ 202__г.
отметка ² _____ «____» ____ 202__г.

Новосибирск 2024

¹ В случае прохождения практики в профильной организации

² Заполняется во время промежуточной аттестации

**План-график проведения
Производственной практики**
вид практики
Шкляев Денис Викторович
Фамилия Имя Отчество студента

института ИВТ, курса 3, гр. ИА-331

Направление: 09.03.01 Информатика и вычислительная техника

Код – Наименование направления (специальности)

Направленность (профиль)/ специализация: Программное обеспечение систем мобильной связи

Место прохождения практики: г. Новосибирск, ул. Бориса Богаткова, д. 51, ауд. 469

Объем практики: 360/10 часов/ЗЕ

Тип практики: Технологическая (проектно-технологическая) практика

Срок практики: с 16.09.2025 по 26.05.2026 (раз в неделю);

Содержание практики³:

Тема индивидуального задания практики TODO

Наименование видов деятельности	Дата (начало – окончание)
Архитектура Adalm Pluto SDR. GNU Radio. Построение радио-приёмника	16.09
Введение в архитектуру SDR-устройств Знакомство с библиотеками Soapy SDR, Libiio для работы с Adalm Pluto SDR. Инициализация SDR-устройства. Работа с буфером: получение цифровых IQ-отсчетов.	23.09
Принципы работы библиотеки Soapy SDR и работы с Adalm Pluto Работа с библиотеками Soapy SDR, Libiio . Формирование и передача с SDR сигналов произвольной формы	30.09, 07.10
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 (семплов)	14.10, 21.10
Моделирование формирования и приема QPSK-сигналов Реализация приема и передачи BPSK-сигналов Алгоритм дискретной свертки	28.10, 11.11
Прием BPSK/QPSK, прием на согласованный фильтр, глазковая диаграмма, поиск оптимального отсчетного значения и необходимость символьной синхронизации Прием и фильтрация сигнала. Прямоугольный и приподнятый косинус	18.11
Символьная синхронизация, детектор временной ошибки, схема Гарднера Программная реализация детектора временной ошибки (синхронизация приемника и передатчика) на SDR Написание функций петли (контура) синхронизации	25.11, 02.12

³ В случае прохождения практики в профильной организации

В соответствии с рабочей программой практики
Руководитель практики от профильной
организации*
«___» ____ 202__ г.

_____ / Андреев А. В. /
(подпись) (ФИО)

Руководитель практики от СибГУТИ
«___» ____ 202__ г.

_____ / Брагин К. И. /
(подпись) (ФИО)

Отзыв о работе студента

(ФИО студента)

Уровень освоения компетенций

Компетенции	Уровень сформированности компетенций
ПК-1 Способен разрабатывать требования и проектировать программное обеспечение	

ПК-3 Способен осуществлять эксплуатацию и развитие транспортных сетей и сетей передачи данных, включая спутниковые системы

Уровень компетенций: высокий, средний, низкий, не сформирована

Руководитель практики от СибГУТИ:

Старший преподаватель

Кафедры ТС и ВС

должность руководителя практики подпись

Брагин К.И.

ФИО руководителя практики

«____» ____ 202__ г.

СОДЕРЖАНИЕ

АРХИТЕКТУРА ADALM PLUTO SDR. GNU RADIO. ПОСТРОЕНИЕ РАДИО-ПРИЁМНИКА	3
ВВЕДЕНИЕ В АРХИТЕКТУРУ SDR-УСТРОЙСТВ. ЗНАКОМСТВО С БИБЛИОТЕКАМИ SOapy SDR, LIBPPO. ИНИЦИАЛИЗАЦИЯ SDR-УСТРОЙСТВА.	
РАБОТА С БУФЕРОМ: ПОЛУЧЕНИЕ I/Q-ОТСЧЁТОВ	10
ПРИНЦИПЫ РАБОТЫ SOapy SDR И РАБОТА С ADALM PLUTO. ФОРМИРОВАНИЕ И ПЕРЕДАЧА СИГНАЛОВ ПРОИЗВОЛЬНОЙ ФОРМЫ	18
ИМИТАЦИЯ АНАЛОГОВОЙ ПЕРЕДАЧИ ЗВУКА И ЕГО ПРИЁМ НА SDR. ВЛИЯНИЕ ЧУВСТВИТЕЛЬНОСТИ И УСИЛЕНИЯ	22
МОДЕЛИРОВАНИЕ ФОРМИРОВАНИЯ И ПРИЁМА QPSK. РЕАЛИЗАЦИЯ ПРИЁМА И ПЕРЕДАЧИ BPSK. АЛГОРИТМ ДИСКРЕТНОЙ СВЁРТКИ	26
ПРИЁМ QPSK/BPSK. СОГЛАСОВАННЫЙ ФИЛЬТР, ГЛАЗКОВАЯ ДИАГРАММА, ПОИСК ОПТИМАЛЬНОГО ОТСЧЁТА, НЕОБХОДИМОСТЬ СИМВОЛЬНОЙ СИНХРОНИЗАЦИИ	32
СИМВОЛЬНАЯ СИНХРОНИЗАЦИЯ. ДЕТЕКТОР ВРЕМЕННОЙ ОШИБКИ. СХЕМА ГАРДНЕРА. РЕАЛИЗАЦИЯ ДЕТЕКТОРА НА SDR. НАПИСАНИЕ ФУНКЦИЙ ПЕТЛИ (КОНТУРА) СИНХРОНИЗАЦИИ	36

АРХИТЕКТУРА ADALM PLUTO SDR. GNU RADIO. ПОСТРОЕНИЕ РАДИО-ПРИЁМНИКА

Цель работы: Ознакомиться с архитектурой SDR-устройства ADALM Pluto. Сформировать радиоприемник для приёма и воспроизведения радиосигналов в реальном времени при помощи фреймворка GNU Radio и Adalm Pluto SDR.

Краткие теоретические сведения

GNURadio - это инструмент который позволяет при помощи “строительных блоков” создать конфигурацию радиоустройства, не написав ни одной строчки кода, и запустить программу непосредственно с использованием модуля **SDR** (программно-определенное радио), например Adalm-Pluto, LimeSDR, и др.

Что такое SDR?

SDR (Software Defined Radio) — это программируемое радио, в котором большинство функций традиционного радиоприёмника и радиопередатчика реализуются программно, а не аппаратно. В классических радиосистемах такие операции, как фильтрация, модуляция, демодуляция, обработка спектра и синхронизация выполняются с помощью аналоговых электронных блоков: фильтров, смесителей, детекторов, генераторов и т. д.

В SDR эти операции переносятся в цифровую область и обрабатываются программно, на компьютере или встроенном процессоре. Аппаратная часть SDR сведена к минимуму и включает только аналоговый фронтенд, АЦП и ЦАП, необходимые для преобразования сигнала между аналоговой и цифровой формами. Благодаря этому SDR позволяет быстро изменять параметры работы радиосистемы — частоту, полосу, тип модуляции и другие настройки — без переделки оборудования, лишь за счёт изменения программной конфигурации.

Adalm Pluto имеет два основных компонента — аналоговый радиочастотный приемопередатчик **AD9363** и система на кристалле (SoC) Xilinx Zynq 7000 Series.

AD9363 содержит необходимые усилители, фильтры, а также цифро-аналоговые и аналого-цифровые преобразователи (**12-bit ADC** и **DAC**). Позволяет принимать и предоставлять IQ-сэмплы, формируемые или принимаемые системой на кристалле **Zynq SoC**. Пользователь может настроить несущую частоту, частоту дискретизации и т.д.

Система на кристалле оснащена процессором **ARM Cortex A9**, работающим на частоте **667 МГц** в сочетании с программируемой пользователем вентильной матрицей (**FPGA**). Пользовательские аппаратные модули реализованы в **FPGA**, обеспечивая связующий слой между прикладными процессорами (**ARM Cortex**) и радиочастотным приемопередатчиком (**AD9363**). Позволяет передавать и получать IQ-сэмплы пользовательскими приложениями, работающими на прикладных процессорах, через драйверы, размещенные в ядре **Linux**.

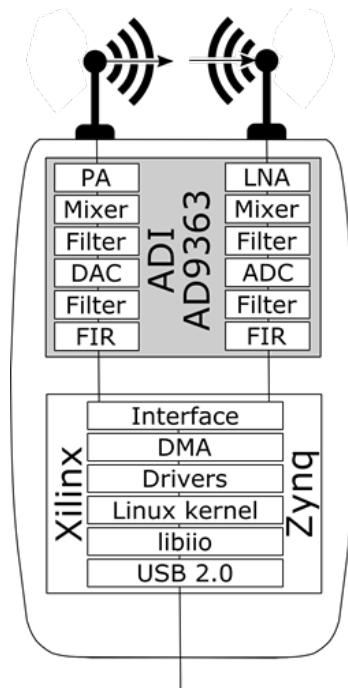


Рисунок 1 — Структура SDR

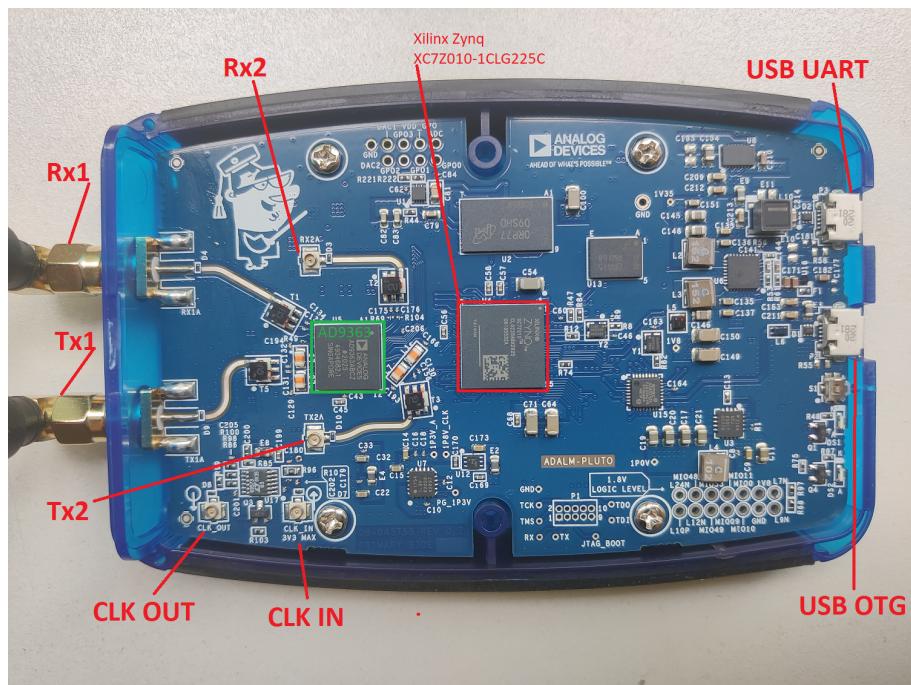


Рисунок 2 — Устройство SDR Adalm Pluto

Ход работы

1. Установка GNURadio

В большинстве версий Ubuntu в менеджере приложений (Ubuntu Software) присутствует пакет GNU Radio Companion.

Я установил GNU Radio через менеджер приложений - Ubuntu Software Center.

2. Сборка FM-радио приёмника

Необходимые блоки:

- **PlutoSDR Source** - источник сигнала с устройства ADALM Pluto.
 - IIO context URI - IP адрес для подключения к устройству.
 - LO Frequency - несущая частота FM-станции.
 - Sample Rate - частота дискретизации.
 - Buffer Size - размер буфера.
 - RF Bandwidth - ширина полосы пропускания.

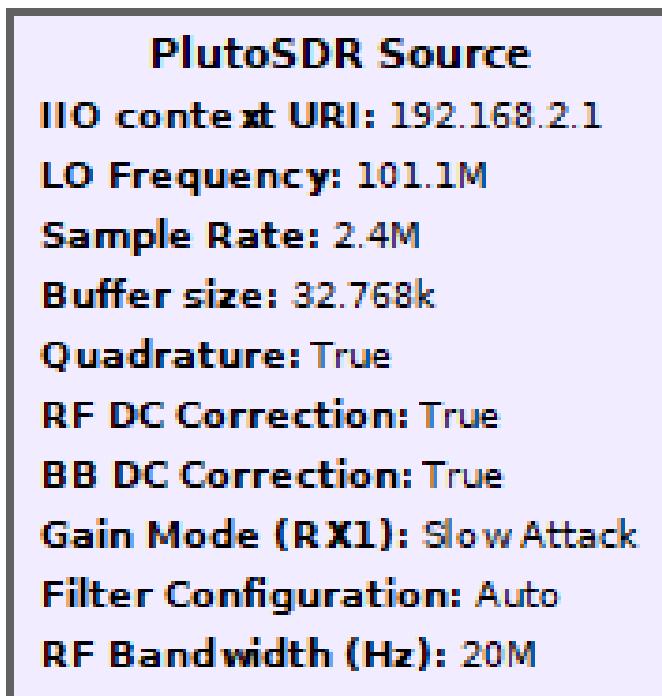


Рисунок 3 — PlutoSDR Source

- **QT GUI Frequency Sink** - блок для визуализации спектра сигнала в реальном времени.

Поможет нам визуально для более точного поиска FM-частоты.

- FFT Size - размер БПФ.
- Center Frequency - центральная частота.
- Bandwidth - полоса пропускания.

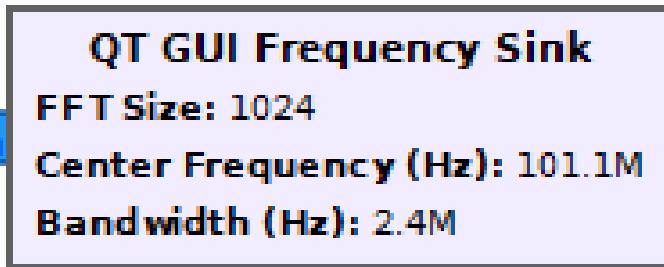


Рисунок 4 — QT GUI Frequency Sink

- **Low Pass Filter** - блок для фильтрации шумов.

Позволяет избавиться (подавить) от “лишнего” сигнала на частотах, отличных от среза исключаемой полосы FM-станции.

- Decimation - параметр, который необходимо настроить под sample rate аудио-потока для прослушивания.
- Gain - коэффициент усиления.
- Sample Rate - частота дискретизации.
- Cutoff Frequency - частота среза фильтра.
- Transition Width - ширина переходной полосы.
- Window - тип окна.

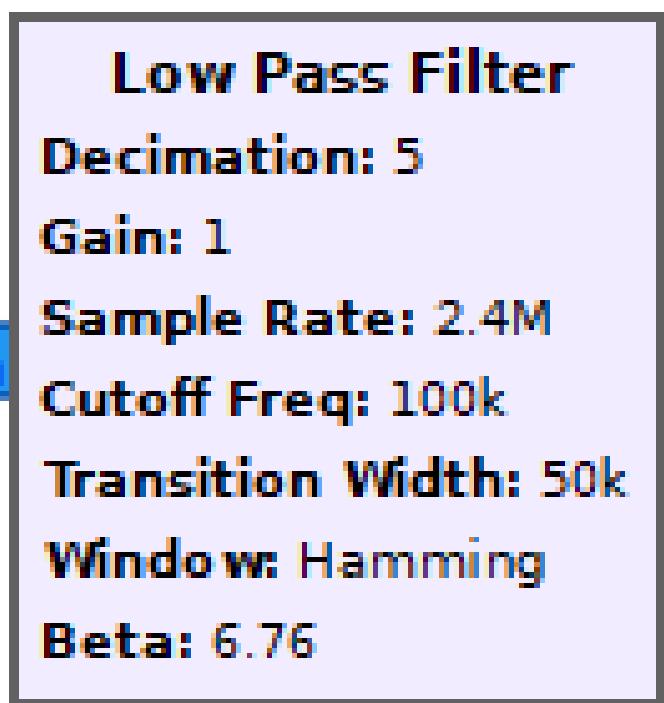


Рисунок 5 — Low Pass Filter

- **WBFM Receive** - блок позволяющий демодулировать широковещательный FM-сигнал.
 - Quadrature Rate - частота дискретизации.
 - Audio Decimation - параметр, который необходимо настроить под sample rate аудио-потока для прослушивания.

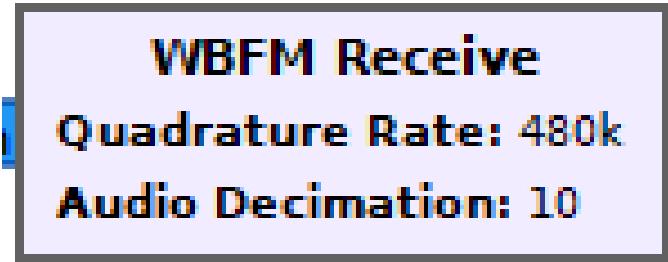


Рисунок 6 — WBFM Receive

- **Audio Sink** - блок для вывода звука на аудиоустройство
 - Sample Rate - частота дискретизации.



Рисунок 7 — Audio Sink

- **QT GUI Time Sink** - блок для визуализации сигнала во временной области.
 - Number of Points - количество точек на экране.
 - Sample Rate - частота дискретизации.
 - Autoscale - авто масштабирование.

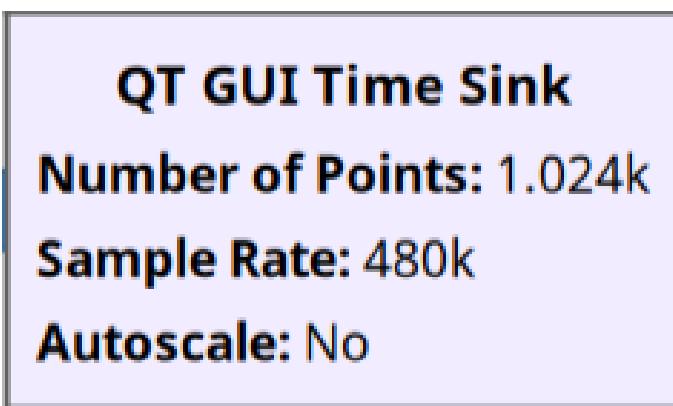


Рисунок 8 — QT GUI Time Sink

- **Variables** - блок переменных.
 - samp_rate: 2.4M - частота дискретизации.
 - bw: 480k - ширина полосы пропускания.
- **QT GUI Range** - GUI ползунок для изменения каких либо параметров в реальном времени.
 - ID: tune - идентификатор переменной.

- Default Value: 106.2M - значение по умолчанию.
- Start: 88.7M - минимальное значение.
- Stop: 110.1M - максимальное значение.
- Step: 12.5k - шаг изменения значения.

Итоговая схема FM-радио приёмника:

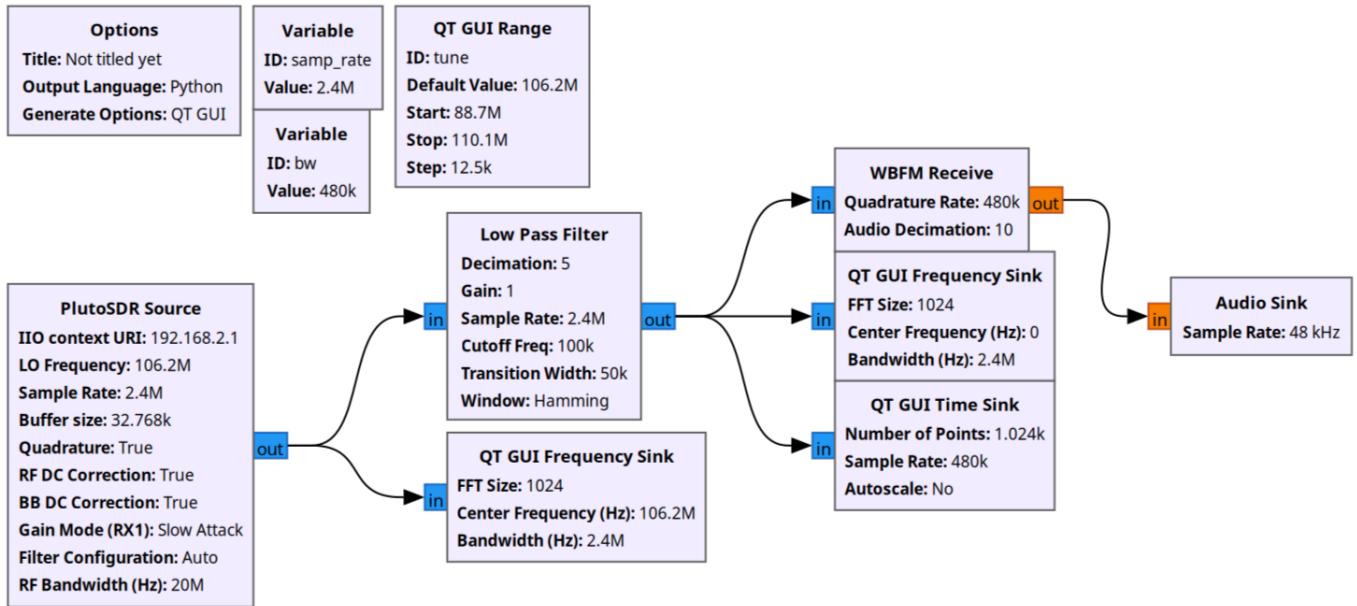


Рисунок 9 — FM Radio Receiver

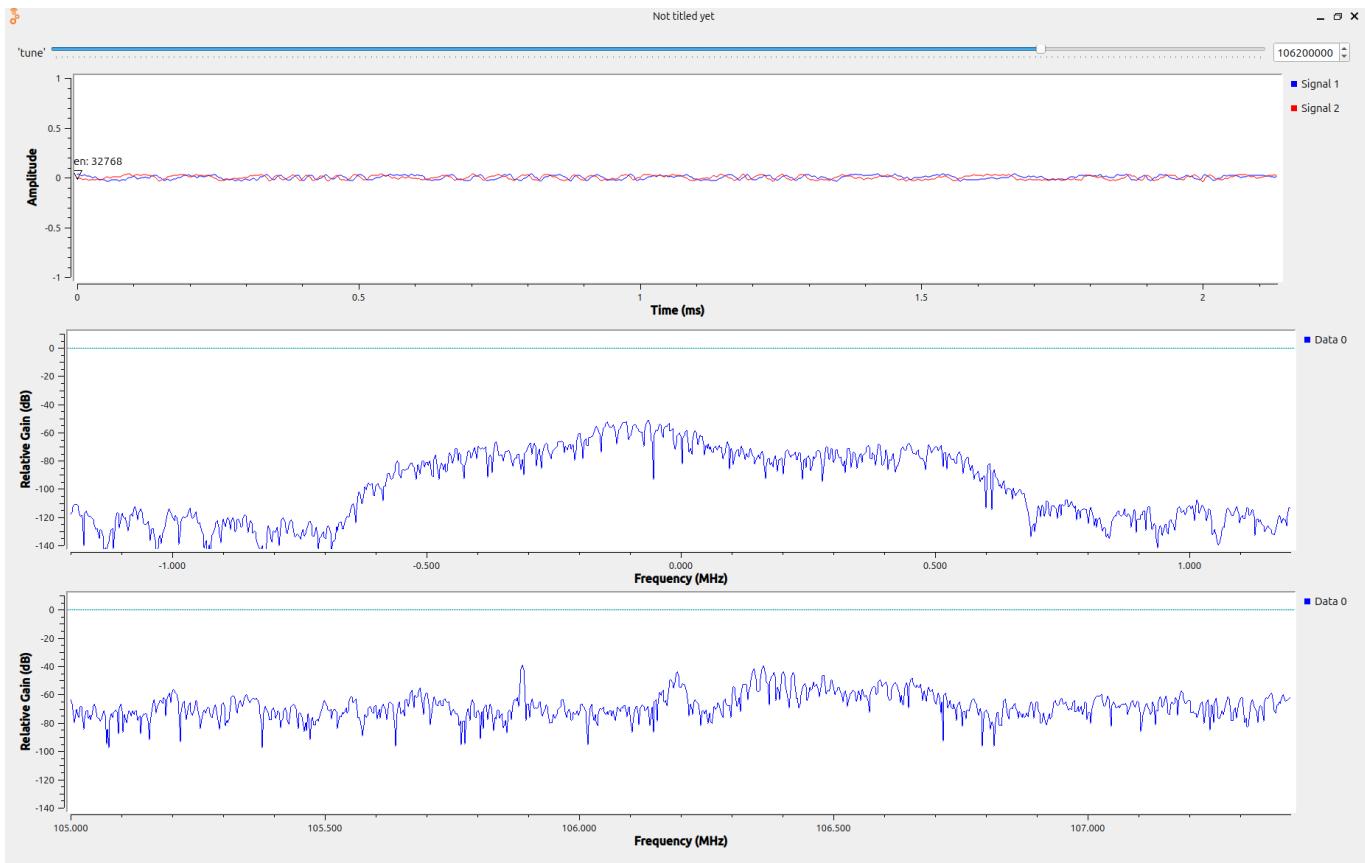


Рисунок 10 — Пример GUI FM-радио приёмника

Вывод

В ходе выполнения лабораторной работы была изучена архитектура программно-определенного радио ADALM-Pluto, а также особенности его взаимодействия с фреймворком GNU Radio. На практике был собран и протестирован FM-радиоприемник, работающий в реальном времени. В процессе работы были освоены базовые блоки GNU Radio, настроены параметры источника сигнала, фильтрации, демодуляции и аудиовывода. Построенная схема позволила успешно принимать, демодулировать и воспроизводить FM-радиосигналы, а также визуализировать спектр и временную область сигнала. Работа показала удобство и гибкость SDR-подхода, позволяющего изменять функциональность радиосистемы за счет программной конфигурации без необходимости модификации аппаратной части.



ВВЕДЕНИЕ В АРХИТЕКТУРУ SDR-УСТРОЙСТВ. ЗНАКОМСТВО С БИБЛИОТЕКАМИ SOAPY SDR, LIBIIO. ИНИЦИАЛИЗАЦИЯ SDR-УСТРОЙСТВА. РАБОТА С БУФЕРОМ: ПОЛУЧЕНИЕ I/Q-ОТСЧЁТОВ

Цель работы: Ознакомиться с архитектурой SDR-устройства. Изучить библиотеки Soapy SDR, Libiio. Научиться инициализировать SDR-устройство и работать с буфером для получения I/Q-отсчётов.

Краткие теоретические сведения

SoapySDR - это универсальная библиотека с открытым исходным кодом, предназначенная для взаимодействия с различными программно-определяемыми радиоустройствами (SDR). Она предоставляет единый интерфейс для работы с множеством SDR-устройств от разных производителей, упрощая процесс разработки приложений, использующих SDR.

Передача сэмплов

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) значительно снижается. Целевое значение (из опыта) частоты дискретизации желательно задавать в пределах 6 Msps.

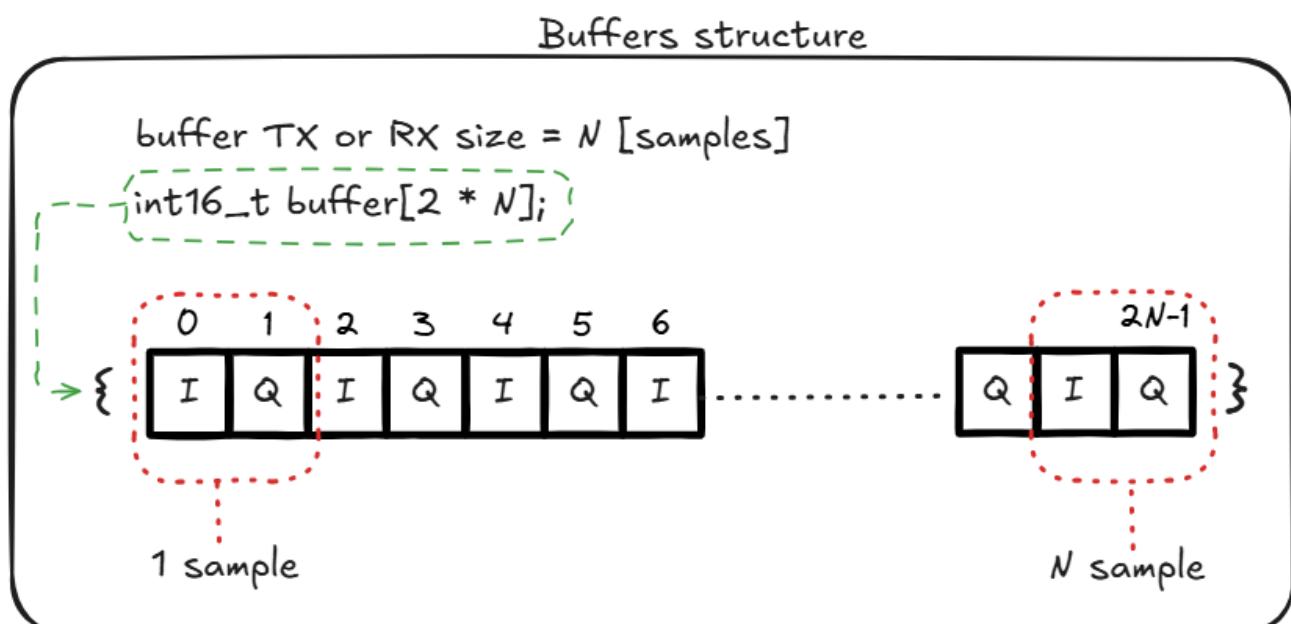


Рисунок 11 — Структура буфера IQ-сэмплов

Передача своих сэмплов на "SDR -> Антenna -> Во вселенную"

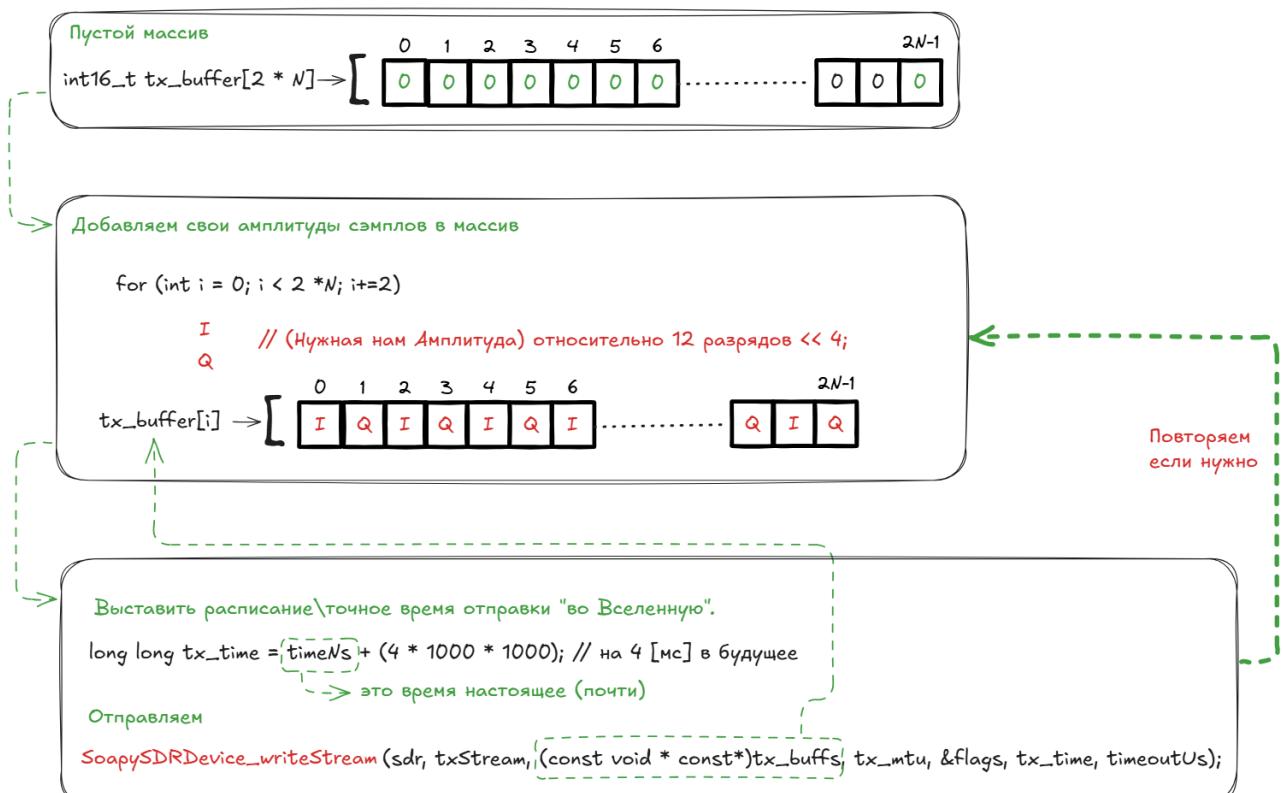


Рисунок 12 — Генерация и запись IQ-сэмплов

Чтение сэмплов с "Вселенная -> Антenna -> SDR"

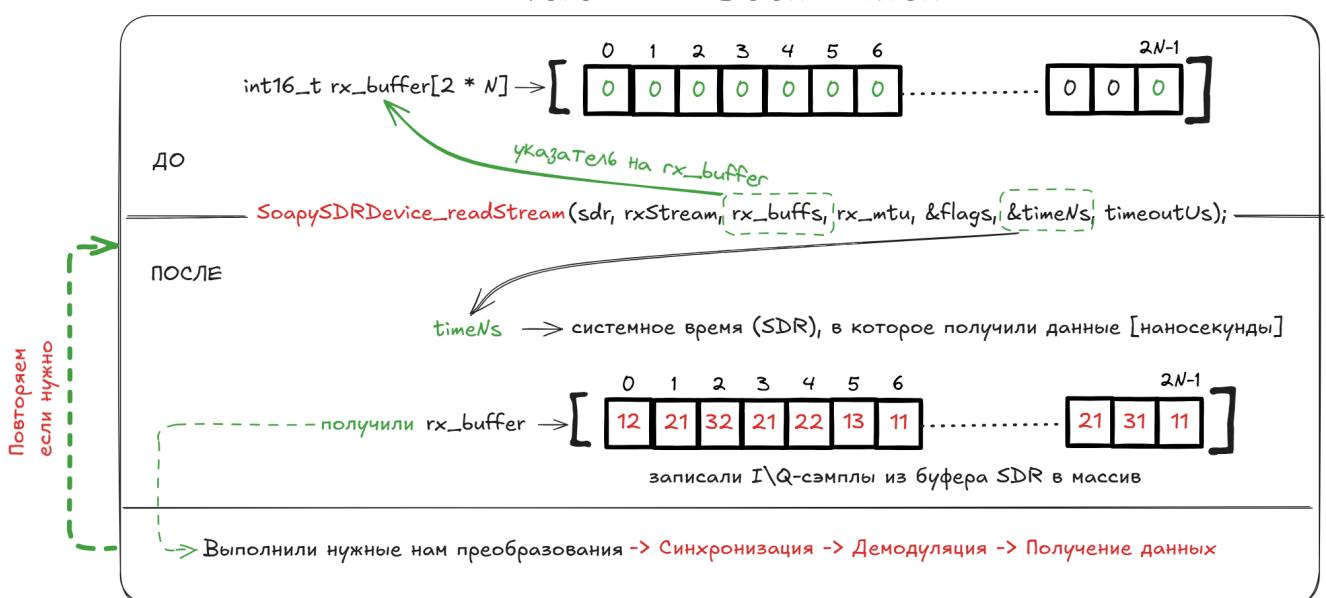


Рисунок 13 — Чтение IQ-сэмплов

Timestamp

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 дескрайтизации,

при больших значениях которой, USB 2.0 не может обеспечить полноценную передачу и прием (одновременных) сэмплов из Adalm Pluto на хост-компьютер.

Ход работы

Установка необходимых библиотек

SoapySDR:

```
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`nproc` # nproc - количество потоков, например make -j16
sudo make install
sudo ldconfig
```

LibIIO:

```
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`nproc`
sudo make install
```

LibAD9361:

```

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

mkdir build && cd build

cmake ../

make -j `nproc`
sudo make install
sudo ldconfig

```

SoapyPlutoSDR:

```

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

mkdir build && cd build

cmake ../

make -j `nproc`
sudo make install
sudo ldconfig

```

Настройка параметров SDR

Используемые библиотеки:

```

#include <SoapySDR/Device.h>    // Инициализация устройства
#include <SoapySDR/Formats.h>   // Типы данных, используемых для записи
→ СЭМПЛОВ
#include <cstdio>
#include <cstdlib>
#include <cstdint>
#include <complex>

```

Инициализация устройства:

```

SoapySDRKwargs args = {};
SoapySDRKwargs_set(&args, "driver", "plutosdr");           // Говорим какой
→ Тип устройства
if (1) {

```

```

    SoapySDRKwargs_set(&args, "uri", "usb:");
                                // Способ обмена
    → сэмплами (USB)
} else {
    SoapySDRKwargs_set(&args, "uri", "ip:192.168.2.1"); // Или по IP-адресу
}
SoapySDRKwargs_set(&args, "direct", "1");           //
SoapySDRKwargs_set(&args, "timestamp_every", "1920"); // Размер буфера +
→ временные метки
SoapySDRKwargs_set(&args, "loopback", "0");        // Используем
→ антенны или нет
SoapySDRDevice *sdr = SoapySDRDevice_make(&args);     // Инициализация
SoapySDRKwargs_clear(&args);

```

Настройка параметров устройств TX/RX:

```

int sample_rate = 1e6;
int carrier_freq = 800e6
// Параметры RX части
SoapySDRDevice_setSampleRate(sdr, SOAPY_SDR_RX, 0, sample_rate);
SoapySDRDevice_setFrequency(sdr, SOAPY_SDR_RX, 0, carrier_freq, NULL);

// Параметры TX части
SoapySDRDevice_setSampleRate(sdr, SOAPY_SDR_TX, 0, sample_rate);
SoapySDRDevice_setFrequency(sdr, SOAPY_SDR_TX, 0, carrier_freq, NULL);

// Инициализация количества каналов RX/TX (в AdalmPluto он один, нулевой)
size_t channels[] = {0}
// Настройки усилителей на RX/TX
SoapySDRDevice_setGain(sdr, SOAPY_SDR_RX, channels, 10.0); //
→ Чувствительность приемника
SoapySDRDevice_setGain(sdr, SOAPY_SDR_TX, channels, -90.0); // Усиление
→ передатчика

```

Инициализация потоков (stream) для передачи и приема сэмплов:

```

size_t channel_count = sizeof(channels) / sizeof(channels[0]);
// Формирование потоков для передачи и приема сэмплов
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);

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

```

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

Подготовка массива для передачи (TX buffer):

```
//заполнение tx_buff значениями сэмплов первые 16 бит - I, вторые 16 бит -
→ Q.
for (int i = 2; i < 2 * tx_mtu; i+=2)
{
    // ЗДЕСЬ БУДУТ ВАШИ СЭМПЛЫ
    tx_buff[i] = 1500 << 4;    // I
    tx_buff[i+1] = 1500 << 4; // Q
}

for(size_t i = 0; i < 2; i++)
{
    tx_buff[0 + i] = 0xffff;
    // 8 x timestamp words
    tx_buff[10 + i] = 0xffff;
}

last_time = timeNs;

// Переменная для времени отправки сэмплов относительно текущего приема
long long tx_time = timeNs + (4 * 1000 * 1000); // на 4 [мс] в будущее

// Добавляем время, когда нужно передать блок tx_buff, через tx_time
→ -наносекунд
for(size_t i = 0; i < 8; i++)
{
    uint8_t tx_time_byte = (tx_time >> (i * 8)) & 0xff;
    tx_buff[2 + i] = tx_time_byte << 4;
}

// Здесь отправляем наш tx_buff массив
void *tx_buffs[] = {tx_buff};
if( (buffers_read == 2) ){
    printf("buffers_read: %d\n", buffers_read);
    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: %i\n", st);
    }
}
```

```
    }  
}
```

Полученные отчеты из функции `SoapySDRDevice_readStream` записываются в файл `symbols.pcm` в бинарном формате записи

Отчетычитываются из файла с помощью Python (matplotlib и numpy):

```
import matplotlib.pyplot as plt  
import numpy as np  
  
rx = np.fromfile("f"/home/plutoSDR/sdr/pluto/dev/rx.pcm", dtype=np.int16)  
  
samples = []  
  
for x in range(0, len(rx), 2):  
    samples.append(rx[x] + 1j * rx[x+1])  
  
ampl = np.abs(samples)  
phase = np.angle(samples)  
time = np.arange(len(samples))  
  
# plot  
plt.subplot(3,1,1)  
plt.legend  
plt.plot(time, ampl)  
plt.title("Amplitude")  
plt.grid(True)  
  
plt.subplot(3,1,2)  
plt.legend  
plt.plot(time, phase)  
plt.title("Phase")  
plt.grid(True)  
  
plt.subplot(3,1,3)  
plt.legend  
plt.plot(time, rx[0::2])  
plt.plot(time, rx[1::2])  
plt.title("I - blue, Q - orange")  
plt.grid(True)
```

```
plt.show()
```

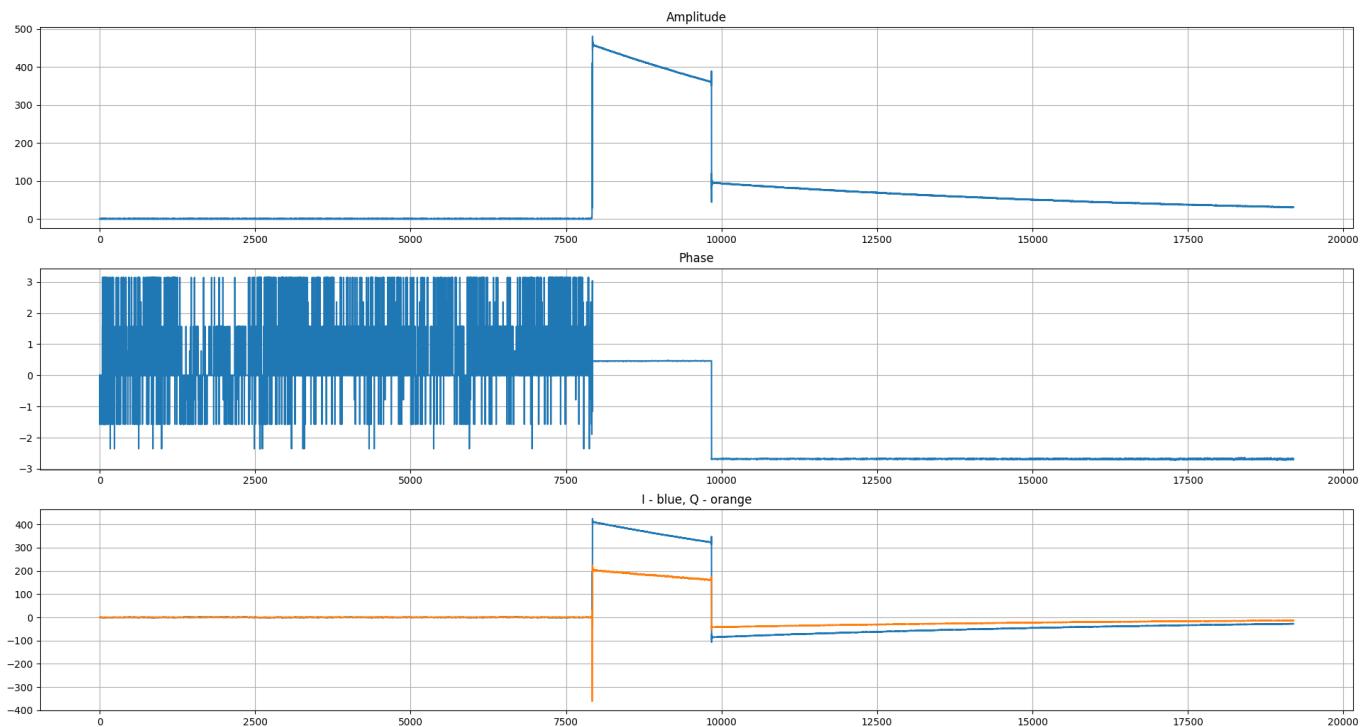


Рисунок 14 — Графики принятых I/Q-сэмплов

На графике можно наблюдать прямоугольный сигнал

Вывод

В данной работе я ознакомился с архитектурой SDR-устройств на примере Adalm Pluto. Изучил библиотеки SoapySDR и Libiio, научился инициализировать SDR-устройство и работать с буфером для получения I/Q-отсчётов. В процессе выполнения работы я установил необходимые библиотеки, настроил параметры устройства, инициализировал потоки для передачи и приёма сэмплов, а также реализовал передачу и приём IQ-сэмплов. В результате работы были получены графики принятых I/Q-сэмплов, что подтвердило успешность выполненных действий.



ПРИНЦИПЫ РАБОТЫ SOAPY SDR И РАБОТА С ADALM PLUTO. ФОРМИРОВАНИЕ И ПЕРЕДАЧА СИГНАЛОВ ПРОИЗВОЛЬНОЙ ФОРМЫ

Цель работы: Получить практическое понимание работы SoapySDR и интерфейсов управления Adalm Pluto, а также освоить формирование и передачу произвольных I/Q-сигналов.

Краткие теоретические сведения

ADALM-PLUTO использует 12-битный ЦАП/АЦП, поэтому каждый из компонентов I и Q может принимать значения от -2^{11} до $2^{11} - 1$, то есть от -2048 до 2047. Для их хранения в C++ удобнее использовать тип `int16_t`, так как меньшие типы не вмещают весь диапазон. В этом случае один комплексный сэмпл (I+Q) занимает 4 байта.

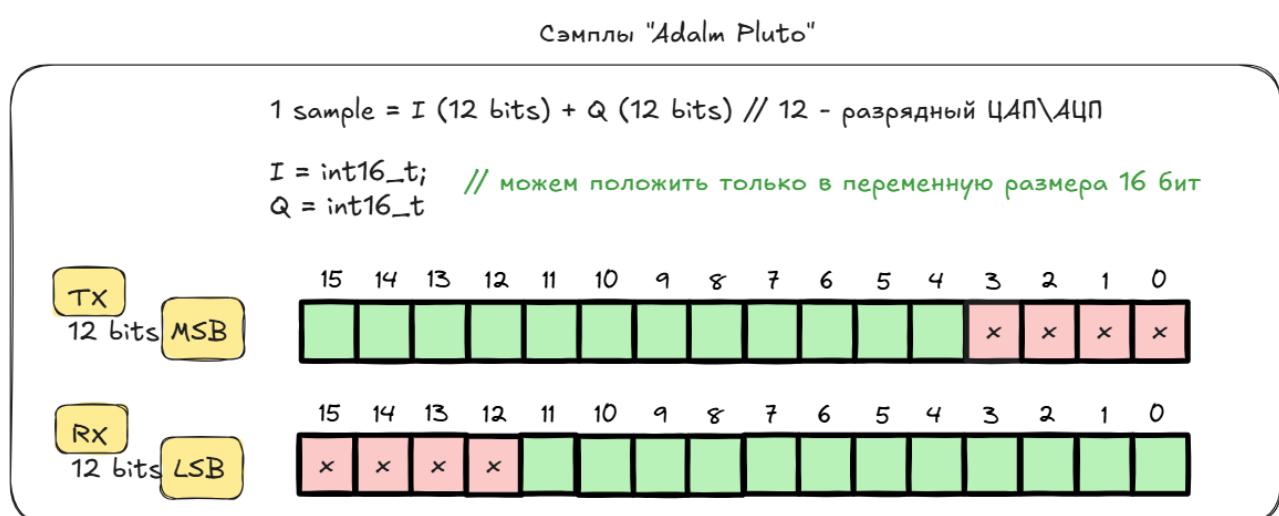


Рисунок 15 — Структура сэмпла

Ход работы

Для формирования сигнала произвольной формы необходимо правильно заполнять буфер сэмплов.

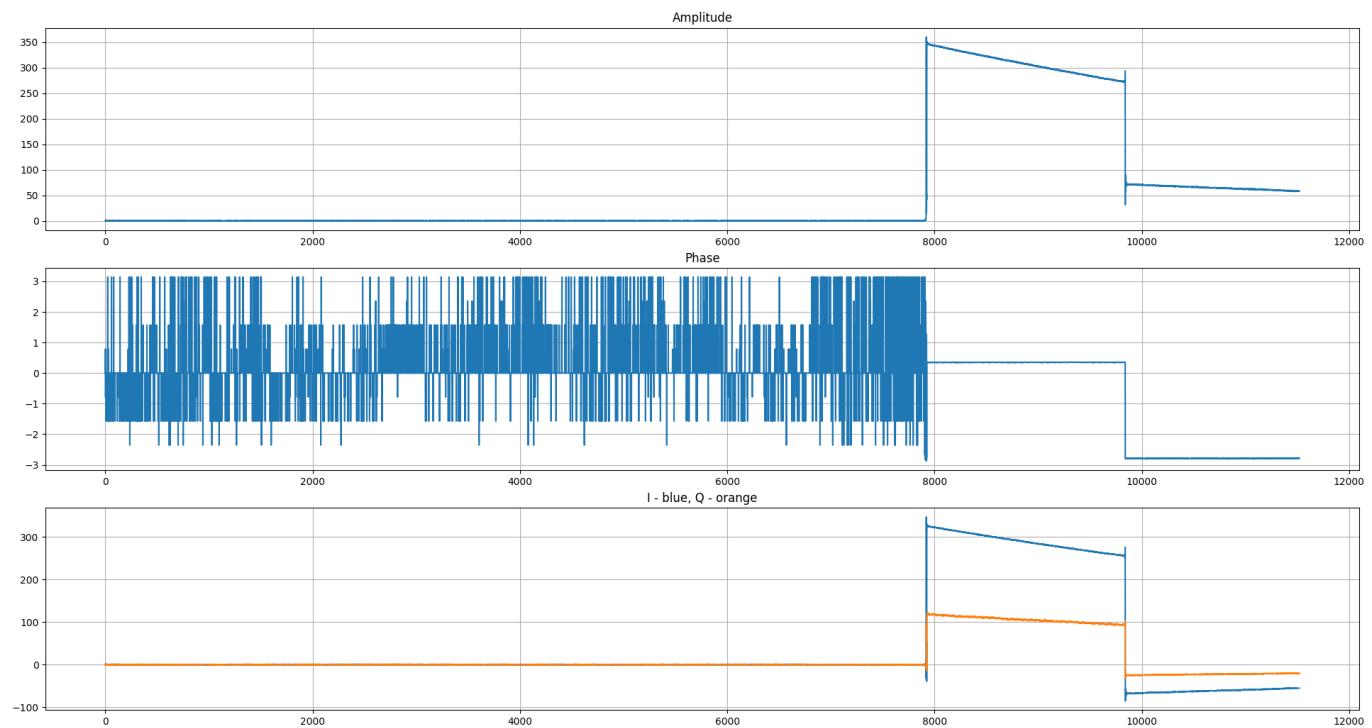


Рисунок 16 — Пример сигнала из буфера заполненного $1500 \ll 4$

Я попробовал следующие варианты:

- Заполнение буфера значениями $1500 \ll 4$ и отправка 3 раза

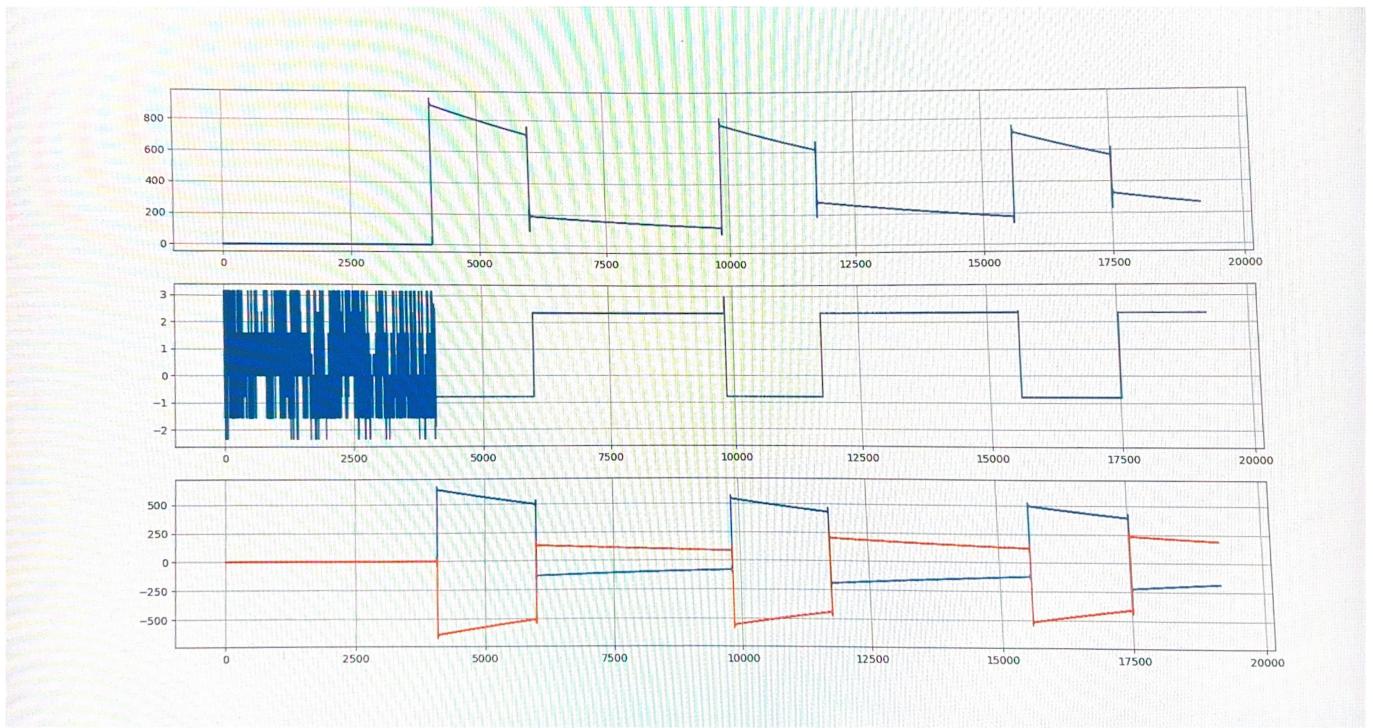


Рисунок 17 — Пример сигнала из буфера заполненного $1500 \ll 4$

- Заполнение буфера значением $(2047 - i) \ll 4$ для I и Q и отправка 50 раз

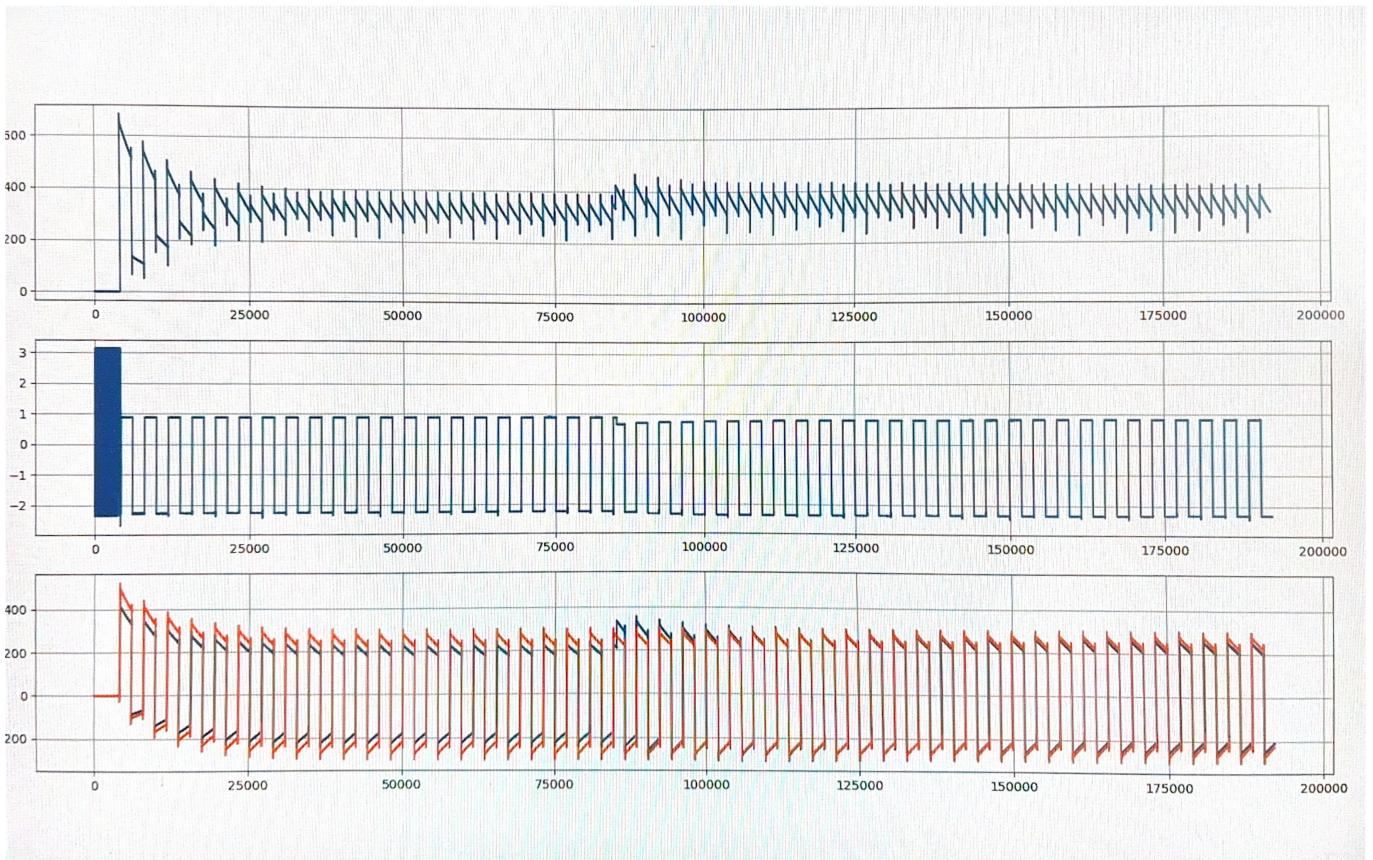


Рисунок 18 — Пример сигнала в форме пицы

Вывод

Работа дала практическое понимание того, как через SoapySDR и интерфейсы управления Pluto можно напрямую формировать и отправлять произвольные I/Q-сигналы. Появилось ясное представление о цепочке: генерация выборок → настройка устройства → передача. А значит, стало понятно, что качество и корректность радиосигнала полностью зависят от точности собственной цифровой обработки



ИМИТАЦИЯ АНАЛОГОВОЙ ПЕРЕДАЧИ ЗВУКА И ЕГО ПРИЁМ НА SDR. ВЛИЯНИЕ ЧУВСТВИТЕЛЬНОСТИ И УСИЛЕНИЯ

Цель работы: Исследование влияния шума, чувствительности и усиления на качество сигнала. Создание алгоритмов конвертации audio→pcm→samples→pcm→audio

Краткие теоретические сведения

Чувствительность приёмника определяет минимальный уровень сигнала, который можно различить на фоне собственного шума тракта. Усиление RX/TX влияет на амплитуду принимаемых и передаваемых выборок: чрезмерное усиление приводит к насыщению и искажению, недостаточное — к снижению отношения сигнал/шум.

Усиление — это коэффициент усиления, который определяет, насколько сильно цифровые I/Q-выборки будут увеличены перед подачей на ЦАП/АЦП. Малое значение даёт слишком слабый радиосигнал, большое — вызывает перегрузку тракта, искажения и спектральные выбросы. Правильный выбор усиления — это подбор уровня, при котором передатчик работает в линейном режиме и форма сигнала соответствует исходным выборкам.

Шум в SDR — это смесь теплового шума, квантования АЦП, помех тракта и внешних источников. Вся обработка в итоге сводится к контролю уровней усиления, динамического диапазона и сохранению линейности канала. Корректное кодирование PCM и обратное преобразование позволяют восстановить исходный аудиосигнал, если цепочка audio→PCM→samples→PCM→audio не вносит систематических искажений.

Ход работы

Для начала необходимо подготовить аудиофайл в формате PCM. Исходный файл в формате MP3 был конвертирован в PCM с помощью следующего скрипта на Python:

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

#-----MP3==>PCM-----
y, sr = librosa.load("../1.mp3", sr=44100, mono=False)
if y.ndim == 2:
    y = y.T.reshape(-1)
pcm_data = (y * 32767).astype(np.int16)
```

```
pcm_data.tofile("../1.pcm")
```

Далее была использована следующая функция для чтения PCM в C++:

```
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);

    *sample_count = file_size / sizeof(int16_t);
    int16_t *samples = (int16_t *)malloc(file_size);
    size_t sf = fread(samples, sizeof(int16_t), *sample_count, file);

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

    fclose(file);

    return samples;
}
```

После чтения PCM-файла, буферы заполняются отчетами и передаются на передачу и приём с помощью следующего кода:

```
size_t sample_count = 0;
int16_t *samples = read_pcm(filename, &sample_count);
if (!samples) return 1;
int buffs_size = tx_mtu;
int buffs_count = (sample_count/buffs_size);
int remainder = sample_count - buffs_count * buffs_size;
int full_size = buffs_count + (int)(bool)remainder;
printf("Количество сэмплов: %ld\nКоличество буферов: %d == %d по %d +
→ %d\n", sample_count, full_size, buffs_count, buffs_size,
→ remainder);

// Количество итерация чтения из буфера
size_t iteration_count = 10;
long long last_time = 0;
```

```

FILE* file = fopen("../2.pcm", "wb");
int16_t *rx_buffer = (int16_t *)malloc((rx_mtu * 2 *
→ sizeof(int16_t)));
void *rx_buffs[] = {rx_buffer};
int flags;           // flags set by receive operation
long long timeNs; //timestamp for receive buffer
long timeoutUs = 400000;

flags = SOAPY_SDR_HAS_TIME;
for (size_t b = 0; b < full_size; b++)
{
    size_t current_size = (b == full_size - 1 && remainder > 0) ?
→ remainder : buffs_size;
const void *one_buff = samples + b * buffs_size * 2;

int sr = SoapySDRDeviceReadStream(sdr, rxStream, rx_buffs,
→ rx_mtu, &flags, &timeNs, timeoutUs);

long long tx_time = timeNs + (4 * 1000 * 1000); // на 4 [мс] в
→ будущее

int st = SoapySDRDeviceWriteStream(sdr, txStream, &one_buff,
→ tx_mtu, &flags, tx_time, timeoutUs);

if (st < 0)
printf("TX Failed on buffer %zu: %i\n", b, st);
fwrite(rx_buffer, 2 * rx_mtu * sizeof(int16_t), 1, file);
last_time = tx_time;
printf("Buffer: %lu - Samples: %i, Flags: %i, Time: %lli,
→ TimeDiff: %lli\n", b, sr, flags, timeNs, (timeNs - last_time)
→ * (last_time > 0));
}

```

После выполнения передачи и приёма, полученный PCM-файл был конвертирован обратно в MP3 с помощью следующего скрипта на Python:

```

-----PCM==>MP3-----
pcm_data = np.fromfile("../2.pcm", dtype=np.int16)
mono_stereo = 2
audio = AudioSegment(data=pcm_data.tobytes(), sample_width=2,
→ frame_rate=44100, channels=mono_stereo)

```

```
audio.export("../2.mp3", format="mp3", bitrate="159k")
```

В при передачи сигнала в "аналоговом" виде любой шум влиял на значения и итоговый звук изменился, результат сложения двух сигналов - [2.mp3](#) - первые 3 секунды - исходный сигнал (хорошо различимый), далее - сигнал с шумом.

Вывод

В ходе работы была реализована последовательная схема преобразования аудиоданных: mp3 → PCM → набор отсчётов → PCM → аудио. Практическая реализация конвертаций позволила подтвердить корректность обработки сигнала в каждом из этапов и обеспечить восстановление исходного звучания после передачи данных в виде необработанных PCM-отсчётов. Эксперимент показал, что качество восстановленного сигнала существенно зависит от уровня шума, параметров чувствительности и усиления при передаче через тракт.



МОДЕЛИРОВАНИЕ ФОРМИРОВАНИЯ И ПРИЁМА QPSK. РЕАЛИЗАЦИЯ ПРИЁМА И ПЕРЕДАЧИ BPSK. АЛГОРИТМ ДИСКРЕТНОЙ СВЁРТКИ.

Цель работы: Реализовать формирование и передачу/прием QPSK/BPSK-сигнала. Разработать алгоритм дискретной свёртки для фильтра.

Краткие теоретические сведения

В **QPSK** (Quadrature Phase Shift Keying) информация кодируется изменением фазы несущего сигнала в четырёх возможных состояниях, что позволяет передавать два бита на символ. В **BPSK** (Binary Phase Shift Keying) используется два состояния фазы, что позволяет передавать один бит на символ.

Mapper - блок, преобразующий входную битовую последовательность в последовательность комплексных символьных значений (модуляция). Для QPSK используется по два бита на символ, для BPSK - один бит.

Upsampler - (нулевой вставщик) увеличивает частоту дискретизации последовательности символов, вставляя между соседними символами $L - 1$ нулевых отсчётов, где L - фактор апсемплинга. Upsampler подготавливает дискретную последовательность для последующей фильтрации (pulse shaping) и цифрового преобразования на несущую. Фактически он задаёт число отсчётов на символ.

Pulse shaping filter - (формирующий фильтр) ограничивает спектр сигнала, уменьшает межсимвольную интерференцию (ISI) и задаёт желаемую временную форму символа. Как правило, корневой фильтр Найквиста (например, root-raised cosine) - выполняет свёртку передискретизированной последовательности символов с импульсной характеристикой, удовлетворяющей критерию отсутствия межсимвольной интерференции в момент выборки.

Constellation diagram - (сигнальное созвездие) представление всевозможных значений комплексной амплитуды манипулированных радиосигналов на комплексной плоскости.

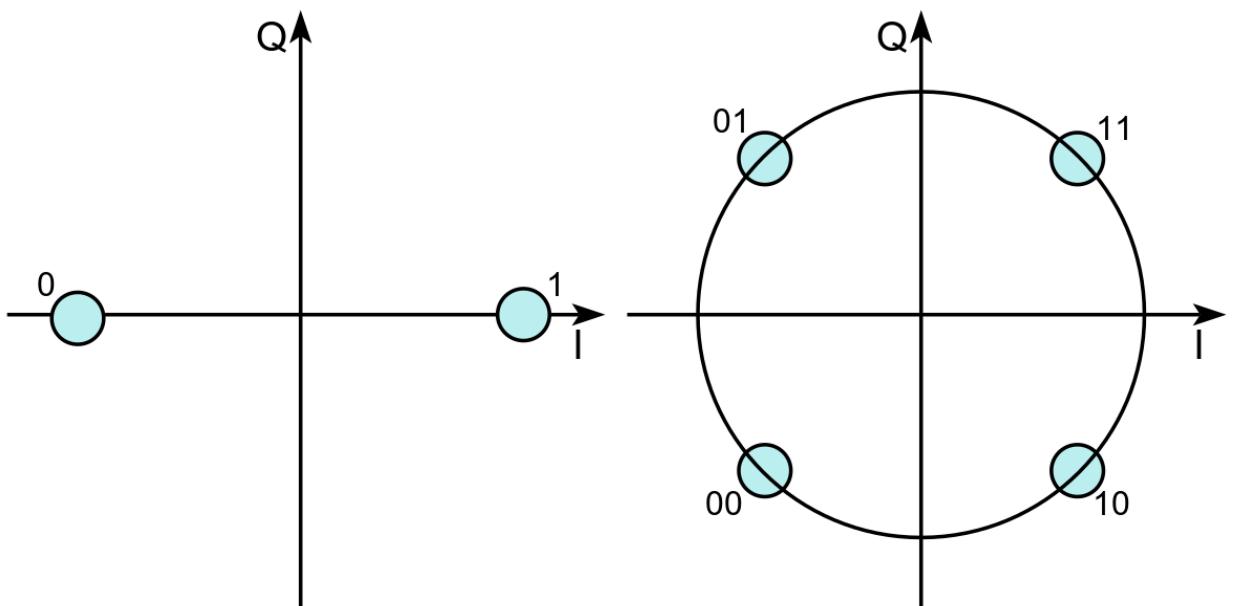


Рисунок 19 — Сигнальное созвездие BPSK и QPSK

Ход работы

В данной работе была реализована передача только QPSK сигнала

Для начала необходимо реализовать маппер.

Mapper QPSK:

```
void mapper_q(const vector<int> &bits, vector<cp> &symbols)
{
    /*
    Map input bits to QPSK symbols and store them in 'symbols'.
    'bits' is the input vector of bits (0s and 1s).
    'symbols' is the output vector of complex symbols.
    00 -> +1 + 1j
    01 -> +1 - 1j
    10 -> -1 + 1j
    11 -> -1 - 1j
    */
    for (size_t i = 0; i < symbols.size(); ++i)
        symbols[i] = cp(bits[2 * i] * -2.0 + 1.0, bits[2 * i + 1] * -2.0
                        + 1.0);
}
```

Mapper BPSK:

```

void mapper_b(const vector<int> &bits, vector<cp> &symbols)
{
    /*
    Map input bits to BPSK symbols and store them in 'symbols'.
    'bits' is the input vector of bits (0s and 1s).
    'symbols' is the output vector of complex symbols.
    0 -> +1 + 0j
    1 -> -1 - 0j
    */

    for (size_t i = 0; i < symbols.size(); ++i)
        symbols[i] = cp(bits[i] * -2.0 + 1.0, 0.0);
}

```

После реализации маппера был реализован апсемплер:

```

void upsample(const vector<cp> &symbols, vector<cp> &upsampled, int up =
→ 10)
{
    /*
    Upsample the input symbols with zeros by a factor of 'up' and store the
    → result in 'upsampled'.
    'symbols' is the input vector of complex symbols.
    'upsampled' is the output vector of complex samples after upsampling.
    'up' is the upsampling factor (default is 10).
    */

    if (upsampled.size() < symbols.size() * up)
    {
        printf("Ошибка: недостаточный размер вектора для апсемплинга!\n");
        return;
    }
    fill(upsampled.begin(), upsampled.end(), cp(0, 0));

    for (size_t i = 0; i < symbols.size(); ++i)
    {
        upsampled[i * up] = symbols[i];
    }
}

```

Функция формирующего фильтра (дискретная свёртка):

```

void filter_i(const vector<cp> &a, const vector<double> &b, vector<int>
→ &y)
{
    /*
    Convolve input signal 'a' with filter coefficients 'b' and store the
    → result in 'y'.
    'a' is a vector of complex samples.
    'b' is a vector of filter coefficients (real numbers), constant 1 in our
    → case.
    'y' is the output vector of integers (filtered signal).
    */
    const int nb = b.size();
    const int na = a.size();

    y.assign(na, 0);

    for (int n = 0; n < na; ++n)
    {
        int acc = 0;
        for (int m = 0; m < nb; ++m)
        {
            if (n - m >= 0)
                acc += a[n - m].real() * b[m];
        }
        y[n] = acc;
    }
}

```

Функция формирующего фильтра (дискретная свёртка) - мнимая часть:

```

void filter_q(const vector<cp> &a, const vector<double> &b, vector<int>
→ &y)
{
    /*
    Convolve input signal 'a' with filter coefficients 'b' and store the
    → result in 'y'.
    'a' is a vector of complex samples.
    'b' is a vector of filter coefficients (real numbers), constant 1 in our
    → case.
    'y' is the output vector of integers (filtered signal).
    */
    const int nb = b.size();

```

```

const int na = a.size();

y.assign(na, 0);

for (int n = 0; n < na; ++n)
{
    int acc = 0;
    for (int m = 0; m < nb; ++m)
    {
        if (n - m >= 0)
            acc += a[n - m].imag() * b[m];
    }
    y[n] = acc;
}
}

```

Полученный сигнал:

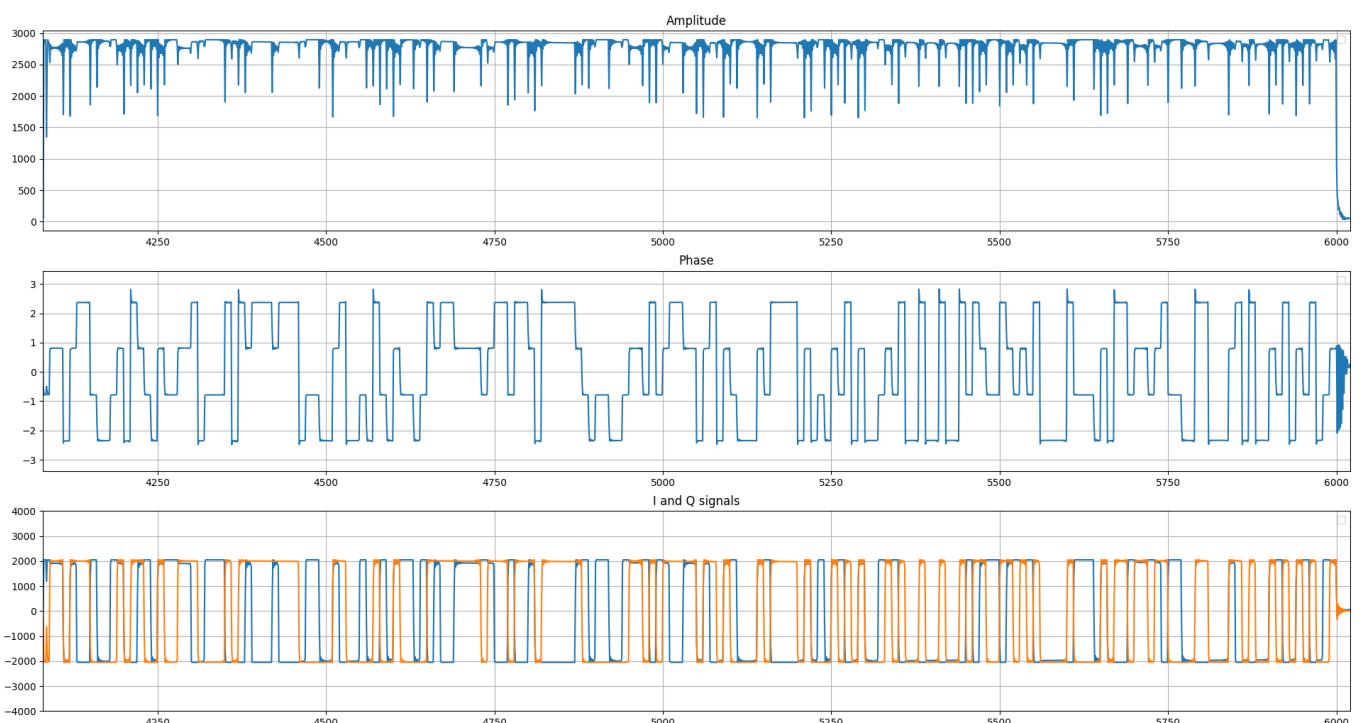


Рисунок 20 — Полученный QPSK сигнал

На графике так же можно заметить что сигналы I и Q лежат в пределах [-2048; 2047] что соответствует 12-битному АЦП.

Вывод

В ходе работы были реализованы ключевые элементы цифрового тракта QPSK/BPSK: маппинг битовых последовательностей в комплексные символы, апсемплирование и дискретная свёртка с формирующим фильтром. Реализация показала, что даже минимальный набор алгоритмов обеспечивает корректное формирование временной структуры сигнала и отображение ожидаемого созвездия. Апсемплирование и фильтрация задали требуемую форму импульса и определили реальную динамику сигнала на уровне отсчётов, что подтверждается попадающими в 12-битный диапазон значениями I/Q.

ПРИЁМ QPSK/BPSK. СОГЛАСОВАННЫЙ ФИЛЬТР, ГЛАЗКОВАЯ ДИАГРАММА, ПОИСК ОПТИМАЛЬНОГО ОТСЧЁТА, НЕОБХОДИМОСТЬ СИМВОЛЬНОЙ СИНХРОНИЗАЦИИ

Цель работы: Реализовать согласованный фильтр

Краткие теоретические сведения

Matched Filter - фильтр смыслом которого является увеличение отношения сигнала/шум (SNR). Этот эффект достигается методом свертки входного сигнала с импульсной характеристикой формирующего фильтра передатчика, обращенной во времени. Это позволяет максимизировать выходной сигнал в момент времени принятия решения, что улучшает вероятность правильного принятия символа.

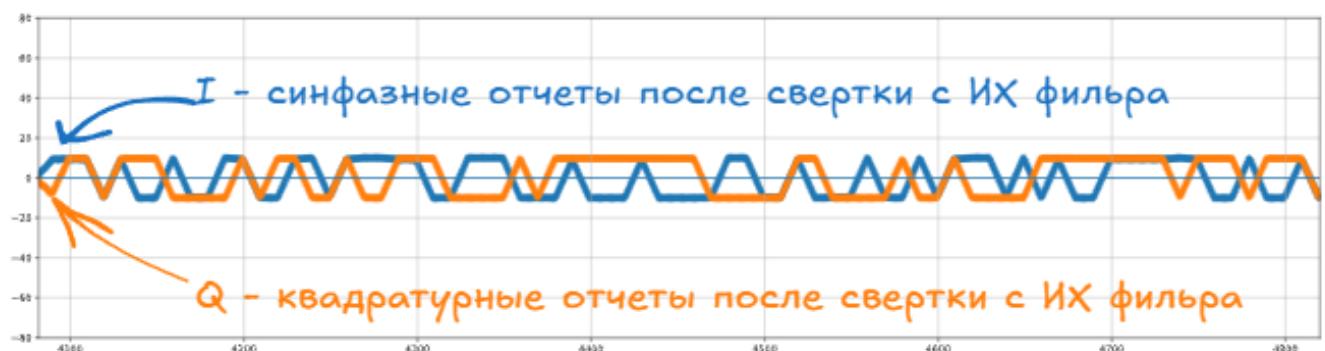


Рисунок 21 — Выход с Matched Filter

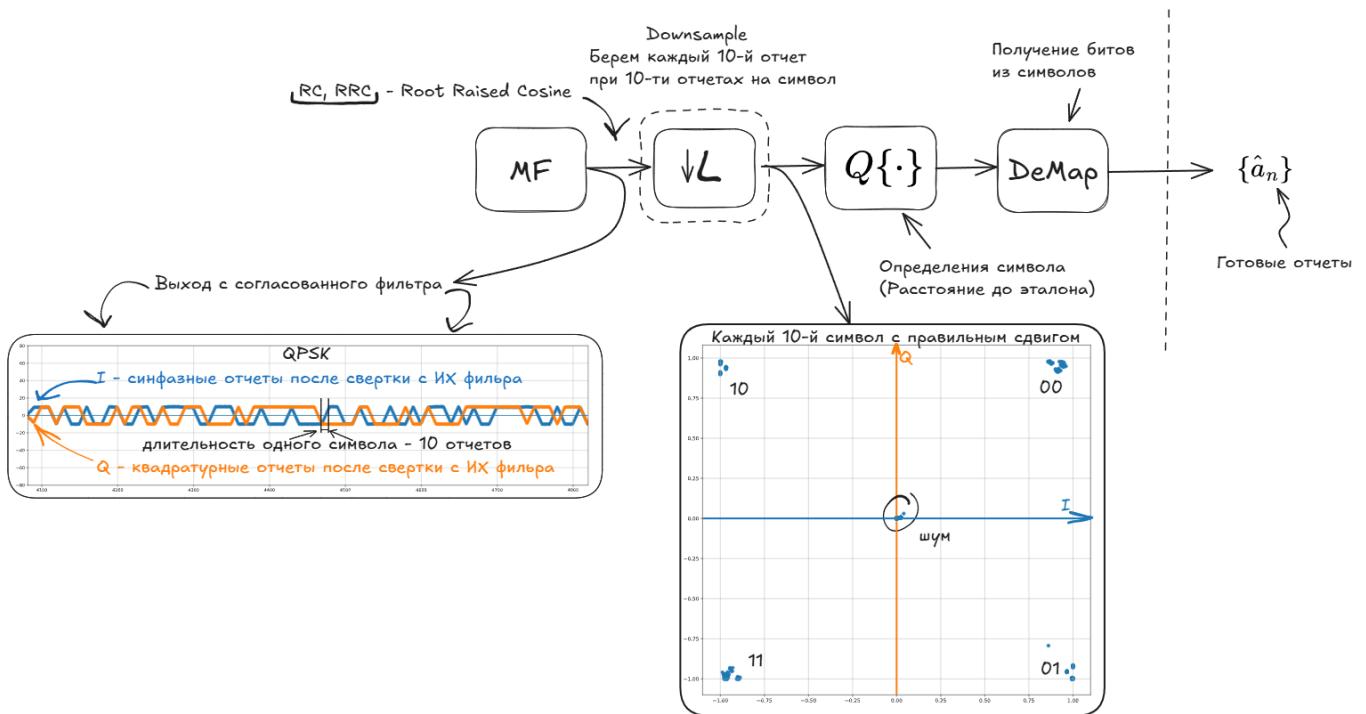


Рисунок 22 — Упрощенная архитектура приемника (демодулятора)

Глазковая диаграмма (Eye Diagram) — это графическое отображение цифрового сигнала, построенное путём наложения нескольких последовательных периодов символа на один интервал времени символа T_s . Она предназначена для анализа качества передачи и выявления интерсимвольной интерференции (ISI), шумов и амплитудных искажений.

Ход работы

Реализация MF на Python:

```

rx = np.fromfile("home/plutoSDR/sdr/pluto/dev/rx1.pcm", dtype=np.int16)
samples_rx = []

for x in range(0, len(rx), 2):
    samples_rx.append((rx[x]+ 1j * rx[x+1])/np.max(rx))

a = np.ones(10)
Is = np.real(samples_rx); Qs = np.imag(samples_rx)
pila = np.convolve(Is, a) + 1j*np.convolve(Qs, a)

signal = []
for x in range(9, len(pila), 10): # Оптимальный сдвиг найден
    → экспериментально
    signal.append(pila[x])

```

Полученный сигнал после MF:

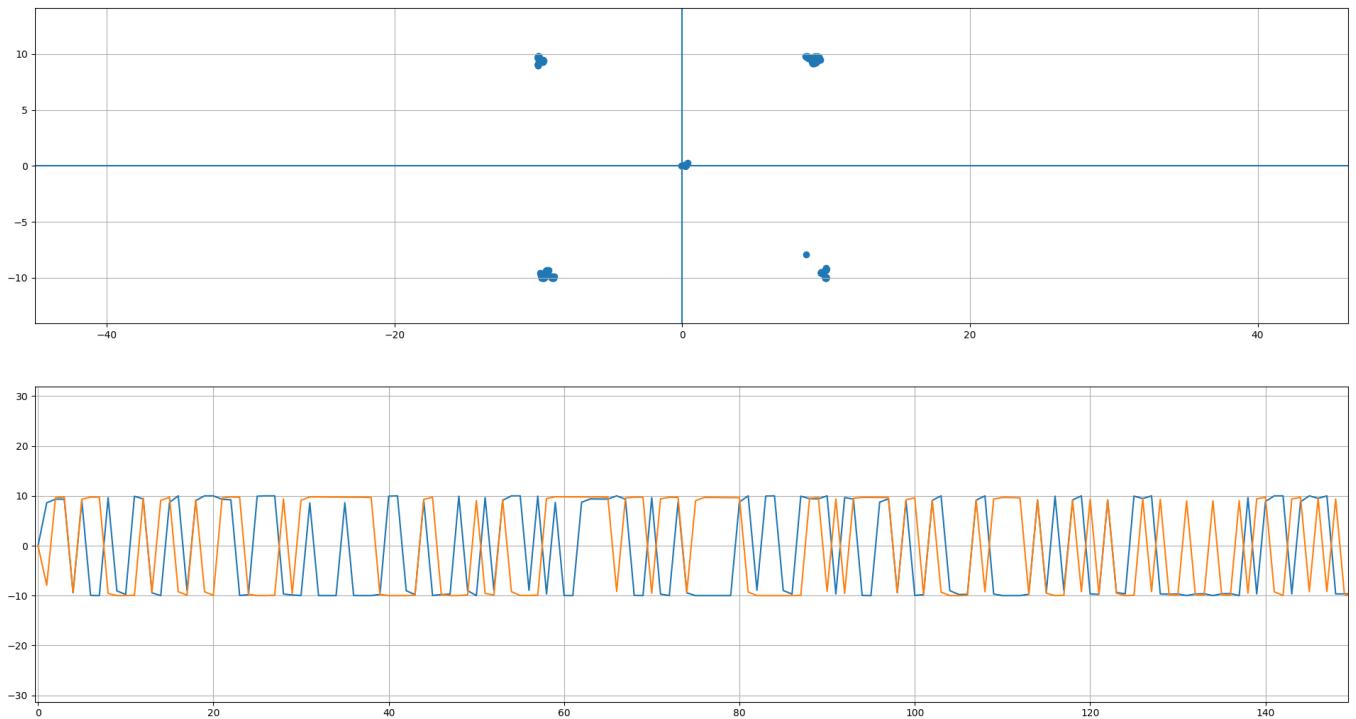


Рисунок 23 — График сигнала после MF, сигнальное созвездие

Сдвиг для оптимального отсчёта был найден экспериментально, равен 9. Для нахождения оптимального отсчёта в реальном времени как раз и необходимо использовать автоматическую символьную синхронизацию.

Так же была построена глазковая диаграмма путем взятия отрезков сигнала длиной в T_s то есть 10 отсчётов (длина символа) с шагом в 10 отсчётов и наложением их друг на друга:

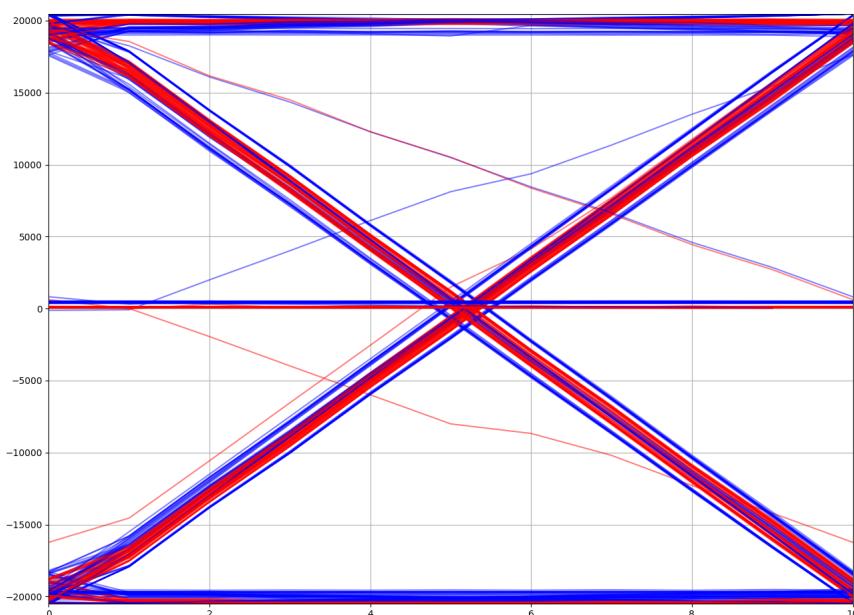


Рисунок 24 — Глазковая диаграмма переходов символа между состояниями. (Синий - I, Красный - Q)

Вывод

Была реализована и протестирована на практике работа согласованного фильтра для приёма QPSK/BPSK сигналов. Проведён анализ полученного сигнала с помощью глазковой диаграммы, что позволило визуально оценить качество передачи и выявить возможные искажения. Работа показала важность использования согласованных фильтров в цифровой связи для улучшения качества приёма сигналов. Так же стало понятно, что для корректного приёма необходимо реализовывать автоматическую символьную синхронизацию для поиска оптимального отсчёта в реальном времени.

СИМВОЛЬНАЯ СИНХРОНИЗАЦИЯ. ДЕТЕКТОР ВРЕМЕННОЙ ОШИБКИ. СХЕМА ГАРДНЕРА. РЕАЛИЗАЦИЯ ДЕТЕКТОРА НА SDR. НАПИСАНИЕ ФУНКЦИЙ ПЕТЛИ (КОНТУРА) СИНХРОНИЗАЦИИ

Цель работы: Реализовать схему Гарднера, контур синхронизации

Краткие теоретические сведения

TED - (Timing Error Detection) функциональный узел, оценивающий отклонение момента выборки от оптимального положения внутри символного интервала. TED принимает сигнал после matched filter и формирует скалярную ошибку $e[n]$, которая затем используется петлёй синхронизации (loop filter) для корректировки фазового накопителя или интерполятора.

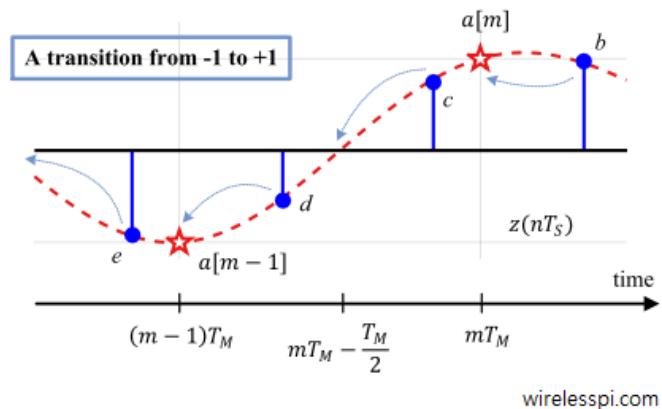


Рисунок 25 — Визуализация ошибки пика символа из за дискретизации

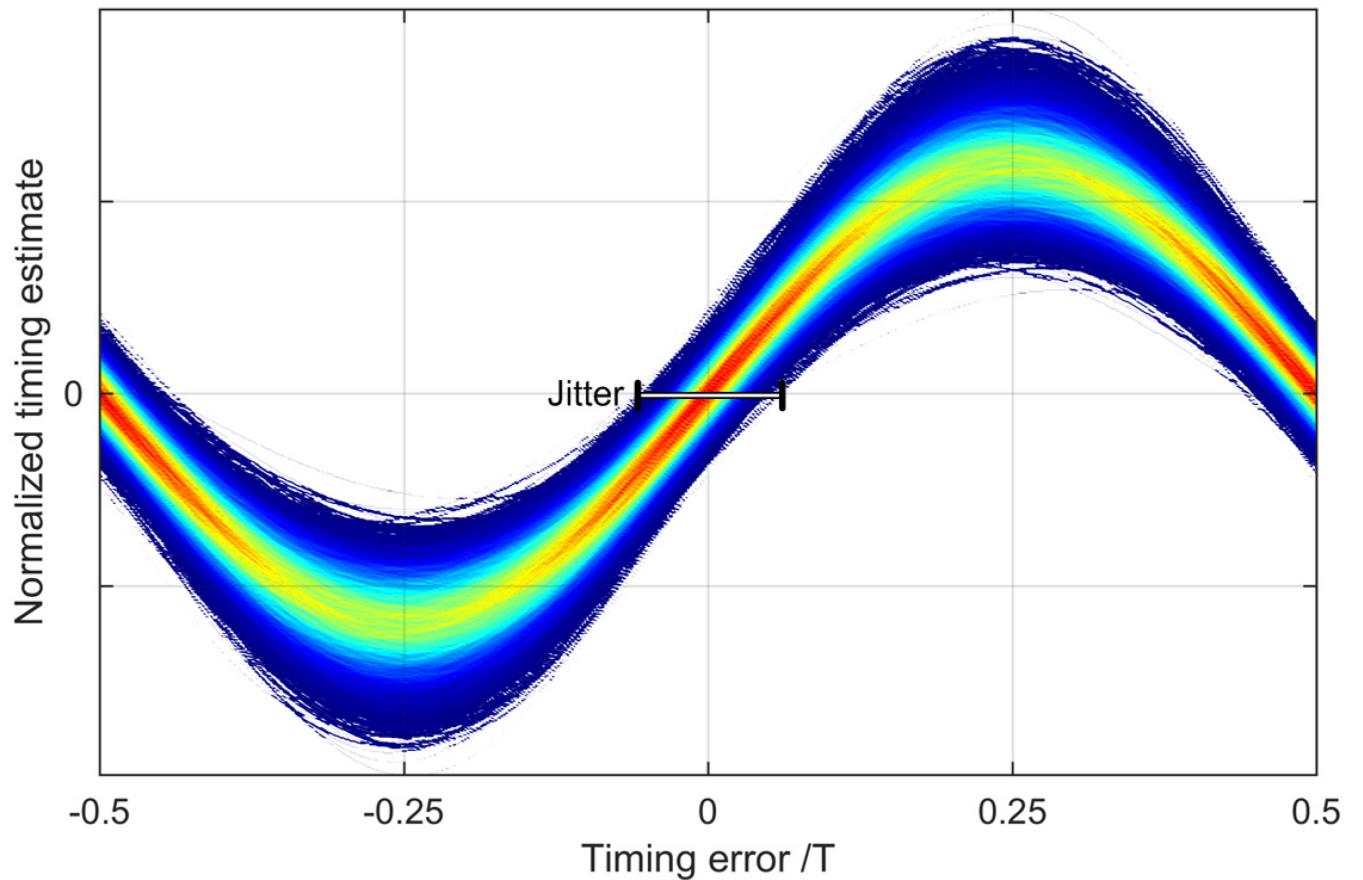


Рисунок 26 — График характеристики TED

Zero-Crossing - простейший детектор, построенный на предположении, что фильтрованный манипулированный сигнал симметричен относительно нуля, а символные выборки должны находиться вблизи амплитудных экстремумов. Граница между символами часто проходит в точке изменения знака производной или самого сигнала. Если выборка приходит ”не вовремя“ то соседние отсчёты будут иметь характерную смену знака, что позволяет оценить временную ошибку.

$$e[n] = x[n - \frac{1}{2}](x[n] - x[n - 1])$$

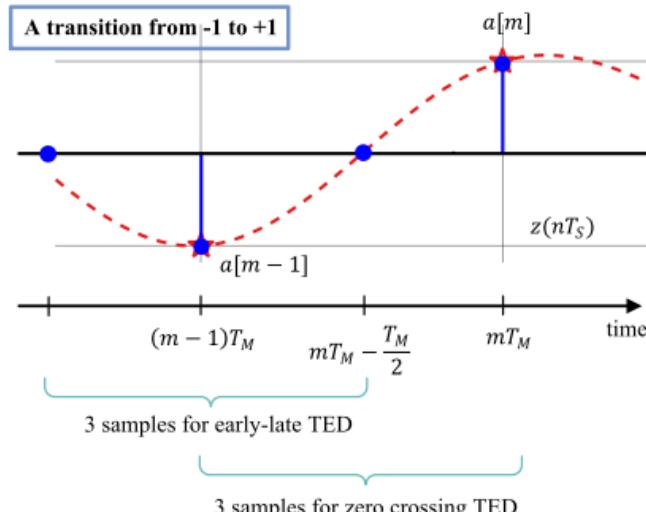


Рисунок 27 — Визуализация работы Zero-Crossing

Схема Гарднера - наиболее распространённый TED в QPSK/ $\pi/4$ -QPSK трактах. Он не требует знания фазы несущей (некогерентный), устойчив к шумам и не использует дифференциальные операции, что снижает чувствительность к ISI. Он основан на предположении, что оптимальная точка выборки должна находиться в центре символьного интервала, а выборка в середине промежутка между ними должна быть равна среднему значению соседних символов.

Loop Filter - (фильтр контура синхронизации) элемент петли восстановления времени, который формирует динамику реакции системы на ошибку. Определяет, насколько петля будет агрессивно корректировать фазу выборки, вычисляет сдвиг.

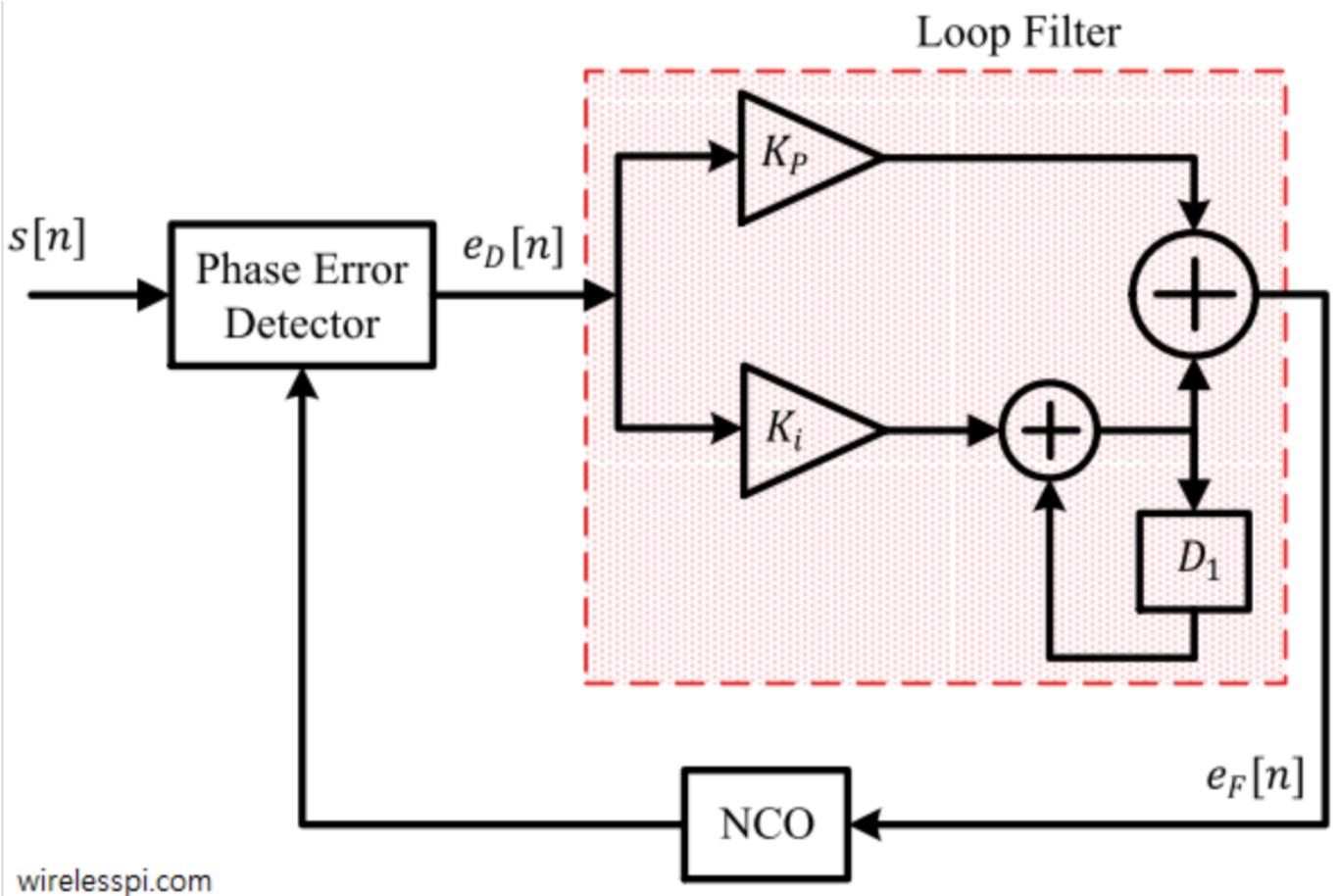


Рисунок 28 — Общая схема с контуром синхронизации

Для алгоритма Loop Filter используются коэффициенты:

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

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

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

- N_{sps} - количество отчетов на символ - τ .
- ζ - коэффициент демпфирования контура синхронизации.
- $B_n T_s$ - нормированная полоса пропускания контура B_n на период символа - T_s .
- K_p - коэффициент пропорциональной составляющей фильтра

Вывод

В ходе работы была изучена схема символьной синхронизации на основе детектора временной ошибки Гарднера. Проведённый анализ показал, что Gardner TED эффективно оценивает сдвиг выборки относительно идеальной точки символа, формируя скалярную ошибку, которая может быть напрямую использована в контуре синхронизации.