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

Разработка ПО для OFDM приемопередатчика на базе программно-конфигурируемого радио (SDR)

Примечание [КБ1]: Тему спросить у меня в тг, она написана в дневнике, у некоторых разная

Выполнил:
студент института информатики и вычислительной техники Иванов Иван Алексеевич
группа ИА-331

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

Проверил[1]
Руководитель практики от профильной организации _____/ Андреев А. В. /
(подпись) *(ФИО)*
«____» _____ 202__г.

Проверил:
Руководитель практики от СибГУТИ _____/ Брагин К. И. /
(подпись) *(ФИО)*

«____» _____ 202__г.

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

Новосибирск 2025

[1] В случае прохождения практики в профильной организации
[2] Заполняется во время промежуточной аттестации

План-график проведения
Производственной практики
вид практики
Иванов Иван Алексеевич
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 мобильных систем
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 29.01.2025 по 27.05.2025 (раз в неделю) и с 17.06.2025 по 13.07.2025 (ежедневно);

Содержание практики[3]:
Тема индивидуального задания практики Разработка ПО для OFDM приемопередатчика на базе программно-конфигурируемого радио (SDR)

Наименование видов деятельности	Дата (начало – окончание)
Прохождение инструктажей ОТ, ПБ, знакомство с структурой предприятия и его деятельностью	01.02
Временная и частотная область обработки, сигналы Формирование сигналов, визуализация в Python	08.02
Дискретизация сигналов. Спектр дискретных отсчетов сигналов. ДПФ Изучение основных параметров библиотеки PyAdi для Adalm Pluto SDR	15.02
Вычисление ДПФ, свойства ДПФ Изучение основных свойств ДПФ с помощью моделирования в Python/Spyder	22.02
Амплитудная модуляция Изучение свойств АМ-сигналов с помощью моделирования в Python/Spyder	29.02
Амплитудная модуляция. Передача\прием прямоугольного сигнала	07.03
Комплексное НЧ представление сигналов. Квадратурное представление сигналов. Цифровая квадратурная модуляция	14.03
Разработка кам модулятора, контроль на анализаторе спектра с разными параметрами сигнала. Разработка КАм демодулятора – символьная синхронизация, посимвольный прием, получение точек созвездия, визуализация. Изучение квадратурных искажений	21.03
КАМ-модулятор, демодулятор, синхронизация приемника и передатчика	28.03
Выполнение первой части отчета, подготовка и заполнение дневника	04.04
Квадратурная IQ модуляция. Общая схема формирования и приема сигналов с дискретной модуляцией. Дискретная АМ, формирование символов в формирующем фильтре, прием сигналов. коды Баркера	11.04
Дискретная АМ, формирование символов в формирующем фильтре, прием сигналов на согласованный фильтр. Формирование QPSK. Передача и прием QPSK-сигналов, синхронизация и фазовая коррекция	18.04

[3] В случае прохождения практики в профильной организации

Частотная синхронизация с фазовой автоподстройкой частоты (ФАПЧ)	25.04, 02.05
GNU Radio. Реализация FM-приемника, воспроизведение звука в real-time.	08.05, 16.05, 23.05
Работа по заданию над проектом. Стандарты для Python	17.06
Прием сигналов на согласованный фильтр, глазковая диаграмма, символьная синхронизация	18.06
Введение в OFDM, идея параллельной передачи. Понятие поднесущей. Частотный разнос между поднесущими	19.06
Синхронизация OFDM-сигналов во времени, циклический префикс	20.06
Баллистика. Эффект Доплера	21.06
Самостоятельная работа	22-23.06
Прием OFDM. Символьная синхронизация, частотная синхронизация, оценка канала и коррекция	24.06
Оценка канала по пилотным RS-сигналам при OFDM-приеме и коррекция принимаемого сигнала	24.06, 25.06
Измерения радиосигналов с помощью анализатора спектра R&S FSH4	26.06
Open source проекты для сетей мобильной связи. LTE srsRAN. настройка сети без радиомодуля	27.06
Open Source проект LTE srsRan. Подключение радиомодуля на базе SDR USRP N310. Настройка БС и UE srsRan	28.06
LTE. Архитектура построения сетей. Сетевые элементы, интерфейсы и их функции. Протоколы взаимодействия между сетевми элементами	28.06 – 30.07
LTE. Радиоподсистема. eNB. стек протоколов радиоинтерфейса. Логические, транспортные, физические каналы и сигналы. Структурная схема приемо-передатчика OFDM. MIMO	01.07 – 07.07
Open Source проект LTE srsRan. Анализ системных параметров из блоков SIB, конфигурация случайного доступа, настройки логгирования	08.07 – 11.07
Конкурс лучших SDR-проектов. Защита работ. Заполнение дневника и сдача отчетов.	12.07 – 13.07

В соответствии с рабочей программой практики

Руководитель практики от профильной организации* _____/ Андреев А. В. /

« 29 » января 2025 г. *(подпись)* *(ФИО)*

Руководитель практики от СибГУТИ _____/ Брагин К. И. /

« 29 » января 2025 г. *(подпись)* *(ФИО)*

Отзыв о работе студента

Примечание [КБ2]: Этот лист не трогать!

_____

(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 _____ Брагин К.И.
должность руководителя практики *подпись* *ФИО руководителя практики*

« 13 » июля 2025 г.

Практика 1
Архитектура Adalm Pluto SDR.
GNU Radio. Построение радио-приёмника

Цель практики:

Знакомство с программой GNU Radio и построение радиоприемника с помощью блоков программы.

Краткие теоретические сведения

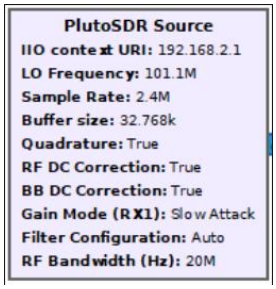
GNU Radio — это свободный набор инструментов для построения программно-определяемых радиосистем (**SDR**), который использует подход "потокового графа" для обработки сигналов. Этот подход позволяет пользователям, используя готовые блоки (например, фильтры, детекторы, преобразователи частоты), создавать радиосистемы для различных целей — как для работы с реальным оборудованием, так и для симуляции. Разработка может вестись в визуальной среде (**GNU Radio Companion**) или с помощью языков программирования, таких как **Python** и **C++.**

Выполнение

Для построения приемника мы используем следующие блоки:

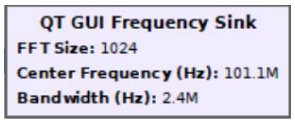
1. **PlutoSDR Source**

Этот блок необходимо привязать к реальному устройству с помощью переменной **IIO context URI**. Блок будет настроен не несущую частоту реального радио-передатчика. Зададим **Sample Rate** при помощи внешней переменной (Variable). Получим следующее:



2. **QT GUI Frequency Sink**

Этот блок необходим для отображения спектральной плотности мощности относительно несущей частоты и для более точного визуального поиска FM-частоты.



3. **Low Pass Filter**

Фильтр низких частот позволяет избавиться (подавить) от “лишнего” сигнала на частотах, отличных от среза искомой полосы FM-станции.

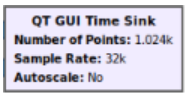
Важным параметром является **Cutoff Freq**, которая определяет половину ширины полосы искомого сигнала по обе стороны от несущей частоты. Известно, что FM-станции вещают в ширине полосы равной 200 [kHz].

Так как мы на выходе всей программы должны получить аудио-сигнал, который можно будет услышать, нам необходимо снизить частоту дискретизации примерно до 48 [kHz]. Для этого используется параметра **Decimation**. В фильтре низких частот можем избавиться от каждого пятого сэмпла

После **LPF** также необходимо визуально посмотреть на результат, добавляем **QT GUI Freq. Sink**.

4. **QT GUI Time Sink**

   Позволит отобразить сигнал во временной шкале.



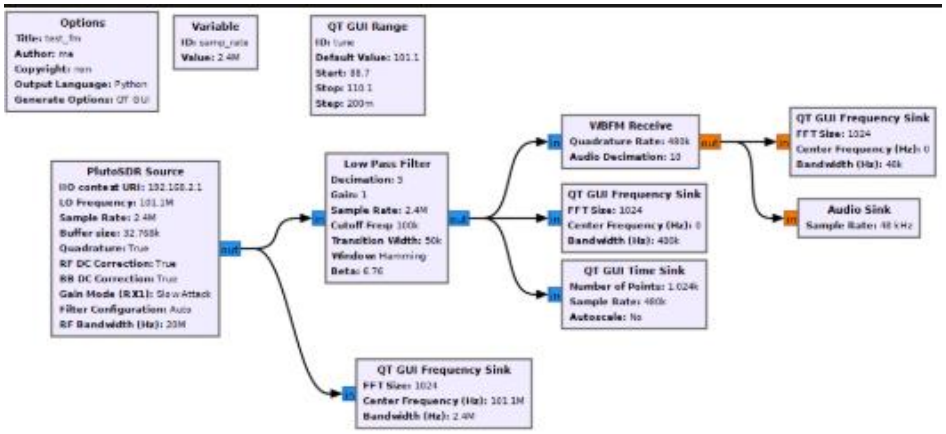
5. **WBFM Receive**

   Блок, позволяющий демодулировать широковещательный FM-сигнал. Важным параметром является **Decimation**, который необходимо настроить под sample rate аудио-потока для прослушивания.

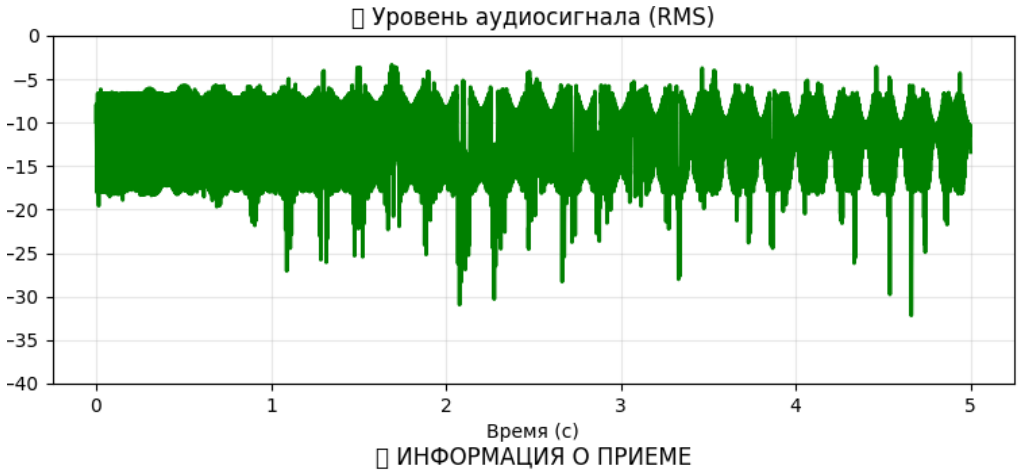
6. **Audio Sink**

   Воспроизведение аудио-потока в динамиках\наушниках компьютера.

Итоговая схема выглядит след. образом:



Результат работы:



Практика 2

Знакомство с библиотеками Soapy SDR,
Libio для работы с Adalm Pluto SDR.
Инициализация SDR-устройства.
Работа с буфером: получение цифровых IQ-отсчетов.

Цель практики:

Изучить основы работы с библиотеками SoapySDR и Libiio для взаимодействия с Adalm Pluto SDR. Освоить инициализацию устройства, настройку параметров и получение цифровых IQ-отсчетов.

Основные теоретические сведения:

Библиотека SoapySDR предоставляет кроссплатформенный API для работы с SDR-оборудованием, абстрагируя особенности конкретных устройств. Libiio (Industrial I/O) является низкоуровневым фреймворком ядра Linux для управления аналогово-цифровыми преобразователями и периферийными устройствами.

ADALM Pluto SDR использует чип AD9363 с 12-битными ЦАП/АЦП, поддерживающий работу в диапазоне частот 325 МГц - 3.8 ГГц. Для представления сигналов используется IQ-формат, где I (in-phase) - синфазная компонента, Q (quadrature) - квадратурная компонента, вместе образующие комплексный сигнал.

Выполнение

Инициализация устройства и базовой конфигурации

```
SoapySDRKwargs args = {};
SoapySDRKwargs_set(&args, "driver", "plutosdr");
SoapySDRKwargs_set(&args, "uri", "ip:192.168.2.1");
SoapySDRDevice *sdr = SoapySDRDevice_make(&args);

// Базовая конфигурация приемника
SoapySDRDevice_setSampleRate(sdr, SOAPY_SDR_RX, 0, 1e6);
SoapySDRDevice_setFrequency(sdr, SOAPY_SDR_RX, 0, 800e6, NULL);
SoapySDRDevice_setGain(sdr, SOAPY_SDR_RX, 0, 10.0);
```

Код инициализирует PlutoSDR через сетевой интерфейс с частотой дискретизации 1 МГц, несущей частотой 800 МГц и усилением 10 дБ.

Создание и активация потока приема

```
size_t channel = 0;
SoapySDRStream *rxStream;
SoapySDRDevice_setupStream(sdr, &rxStream, SOAPY_SDR_RX, SOAPY_SDR_CS16,
&channel, 1, NULL);
SoapySDRDevice_activateStream(sdr, rxStream, 0, 0, 0);
```

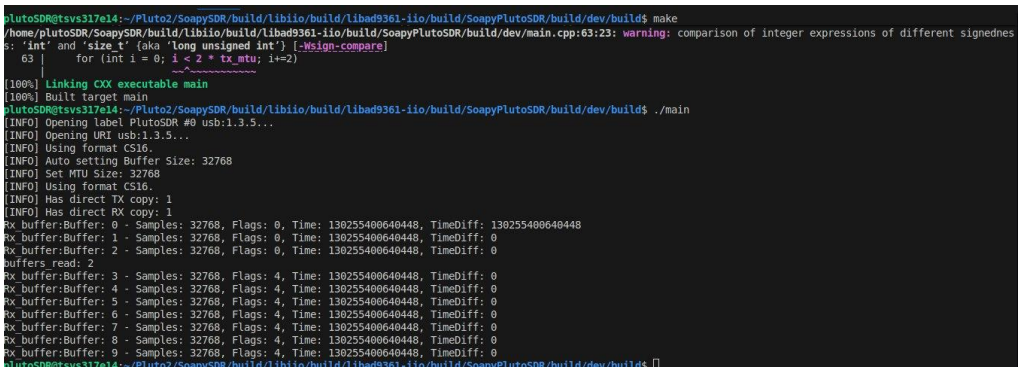
Создается RX-поток с форматом CS16 (16-битные целочисленные I/Q samples) и активируется для работы.

Организация буфера приема и чтение данных

```
size_t mtu = SoapySDRDevice_getStreamMTU(sdr, rxStream);
int16_t *rx_buffer = malloc(2 * mtu * sizeof(int16_t));

// Цикл приема данных
void *buffs[] = {rx_buffer};
int samples_read = SoapySDRDevice_readStream(sdr, rxStream, buffs, mtu, &flags,
&timeNs, timeoutUs);
```

=

Получим следующий вывод отсчетов



Для визуализации отсчетов используем простую программу на Python

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

# Параметры сигнала
A = 2.0  # Амплитуда
f0 = 5.0  # Частота (Гц)
phi = -0.3  # Начальная фаза (рад)
T = 1 / f0  # Период
Ts = 0.001  # Шаг дискретизации

# Вектор времени (несколько периодов)
t = np.arange(-2 * T, 2 * T, Ts)

# Гармоническое колебание
x = A * np.cos(2 * np.pi * f0 * t + phi)

# Вычисление коэффициентов Фурье для n = 0,1,2,3,4
n_max = 4
a_coeffs = np.zeros(n_max + 1)
b_coeffs = np.zeros(n_max + 1)
A_coeffs = np.zeros(n_max + 1)
phi_coeffs = np.zeros(n_max + 1)

for n in range(n_max + 1):
    # Опорные колебания
    cos_n = np.cos(2 * np.pi * n * f0 * t)
    sin_n = np.sin(2 * np.pi * n * f0 * t)

    # Интегрирование произведений
    a_n = (2 / T) * np.sum(x * cos_n) * Ts
    b_n = (2 / T) * np.sum(x * sin_n) * Ts

    a_coeffs[n] = a_n
    b_coeffs[n] = b_n

    # Амплитуда и фаза гармоники
    A_coeffs[n] = np.sqrt(a_n + b_n)
    phi_coeffs[n] =  - np.arctan2(b_n, a_n)
```

```
print("Коэффициенты для гармонического колебания (φ = 0):")
print("n\t a_n\t\t b_n\t\t A_n\t\t φ_n")
for n in range(n_max + 1):
    print(f"{n}\t {a_coeffs[n]:.4f}\t {b_coeffs[n]:.4f}\t {A_coeffs[n]:.4f}\t
{phi_coeffs[n]:.4f}")

# Графики для гармонического колебания
plt.figure(figsize=(12, 8))

# Сигнал и его спектр
plt.subplot(2, 2, 1)
plt.plot(t, x)
plt.title('Гармоническое колебание')
plt.xlabel('Время (с)')
plt.ylabel('Амплитуда')
plt.grid(True)

# Амплитудный спектр
plt.subplot(2, 2, 2)
n_values = np.arange(0, n_max + 1)
plt.stem(n_values, A_coeffs)
plt.title('Амплитудный спектр (An)')
plt.xlabel('Номер гармоники (n)')
plt.ylabel('Амплитуда')
plt.grid(True)

# Фазовый спектр
plt.subplot(2, 2, 3)
plt.stem(n_values, phi_coeffs)
plt.title('Фазовый спектр (φn)')
plt.xlabel('Номер гармоники (n)')
plt.ylabel('Фаза (рад)')
plt.grid(True)

# Коэффициенты a_n и b_n
plt.subplot(2, 2, 4)
plt.stem(n_values, a_coeffs, 'b', markerfmt='bo', label='a_n')
plt.stem(n_values, b_coeffs, 'r', markerfmt='ro', label='b_n')
plt.title('Коэффициенты a_n и b_n')
plt.xlabel('Номер гармоники (n)')
plt.ylabel('Значение')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

Вывод такой:

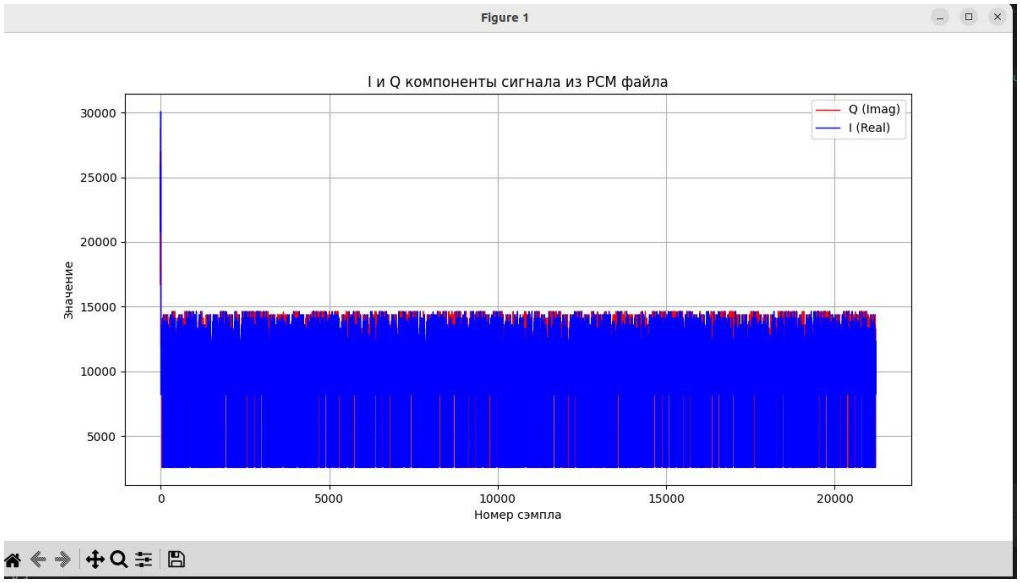


Figure 1
I и Q компоненты сигнала из PCM файла
Q (Imag)
I (Real)
Значение
Номер сэмпла
30000
25000
20000
15000
10000
5000
0
5000
10000
15000
20000

Практика 3

Работа с библиотеками Soapy SDR, Libio .

Формирование и передача с SDR сигналов произвольной формы

Цель работы:

Освоить формирование комплексных сигналов произвольной формы и их передачу через PlutoSDR.

Краткие теоретические сведения

Комплексный сигнал представляется как I + jQ. Для PlutoSDR требуется сдвиг на 4 бита влево при формировании samples.

Выполнение

Генерация тонального сигнала

```
void generate_complex_tone(int16_t *buffer, size_t num_samples,
                           double amplitude, double frequency, double sample_rate)
{
    for (size_t i = 0; i < num_samples; i++) {
        double t = (double)i / sample_rate;
        double i_val = amplitude * cos(2 * M_PI * frequency * t);
        double q_val = amplitude * sin(2 * M_PI * frequency * t);

        buffer[2*i] = (int16_t)i_val << 4;
        buffer[2*i + 1] = (int16_t)q_val << 4;
    }
}
```

Функция генерирует комплексный тональный сигнал с заданной амплитудой и частотой, применяя сдвиг на 4 бита.

Настройка передатчика и передача

```
SoapySDRDevice_setSampleRate(sdr, SOAPY_SDR_TX, 0, 1e6);
SoapySDRDevice_setFrequency(sdr, SOAPY_SDR_TX, 0, 800e6, NULL);
SoapySDRDevice_setGain(sdr, SOAPY_SDR_TX, 0, -50.0);
SoapySDRStream *txStream;
SoapySDRDevice_setupStream(sdr, &txStream, SOAPY_SDR_TX, SOAPY_SDR_CS16,
&channel, 1, NULL);
SoapySDRDevice_activateStream(sdr, txStream, 0, 0, 0);

// Включение циклического режима
SoapySDRDevice_setTxCyclic(sdr, true);
void *tx_buffs[] = {tx_buffer};
int tx_result = SoapySDRDevice_writeStream(sdr, txStream, tx_buffs, mtu, &flags,
timeNs, 1000000);
```

Настраивается передатчик с минимальным усилением для безопасности, создается TX-поток и запускается циклическая передача.

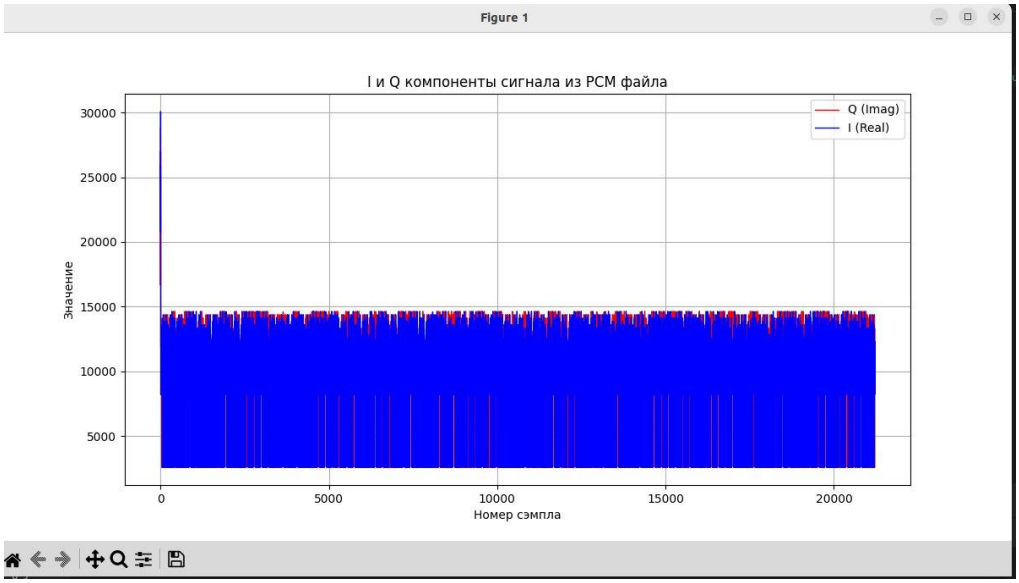
Для визуализации отсчетов используем простую программу на Python

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

# Параметры сигнала
A = 2.0  # Амплитуда
f0 = 5.0  # Частота (Гц)
phi = -0.3  # Начальная фаза (рад)
T = 1 / f0  # Период
Ts = 0.001  # Шаг дискретизации

# Вектор времени (несколько периодов)
t = np.arange(-2 * T, 2 * T, Ts)

# Гармоническое колебание
x = A * np.cos(2 * np.pi * f0 * t + phi)

# Вычисление коэффициентов Фурье для n = 0,1,2,3,4
```

```
n_max = 4
a_coeffs = np.zeros(n_max + 1)
b_coeffs = np.zeros(n_max + 1)
A_coeffs = np.zeros(n_max + 1)
phi_coeffs = np.zeros(n_max + 1)

for n in range(n_max + 1):
    # Опорные колебания
    cos_n = np.cos(2 * np.pi * n * f0 * t)
    sin_n = np.sin(2 * np.pi * n * f0 * t)

    # Интегрирование произведений
    a_n = (2 / T) * np.sum(x * cos_n) * Ts
    b_n = (2 / T) * np.sum(x * sin_n) * Ts

    a_coeffs[n] = a_n
    b_coeffs[n] = b_n

    # Амплитуда и фаза гармоники
    A_coeffs[n] = np.sqrt(a_n + b_n)
    phi_coeffs[n] =  - np.arctan2(b_n, a_n)

print("Коэффициенты для гармонического колебания (φ = 0):")
print("n\t a_n\t\t b_n\t\t A_n\t\t φ_n")
for n in range(n_max + 1):
    print(f"{n}\t {a_coeffs[n]:.4f}\t {b_coeffs[n]:.4f}\t {A_coeffs[n]:.4f}\t
{phi_coeffs[n]:.4f}")

# Графики для гармонического колебания
plt.figure(figsize=(12, 8))

# Сигнал и его спектр
plt.subplot(2, 2, 1)
plt.plot(t, x)
plt.title('Гармоническое колебание')
plt.xlabel('Время (с)')
plt.ylabel('Амплитуда')
plt.grid(True)

# Амплитудный спектр
plt.subplot(2, 2, 2)
n_values = np.arange(0, n_max + 1)
plt.stem(n_values, A_coeffs)
plt.title('Амплитудный спектр (An)')
plt.xlabel('Номер гармоники (n)')
plt.ylabel('Амплитуда')
plt.grid(True)

# Фазовый спектр
plt.subplot(2, 2, 3)
plt.stem(n_values, phi_coeffs)
plt.title('Фазовый спектр (φn)')
plt.xlabel('Номер гармоники (n)')
plt.ylabel('Фаза (рад)')
plt.grid(True)

# Коэффициенты a_n и b_n
plt.subplot(2, 2, 4)
plt.stem(n_values, a_coeffs, 'b', markerfmt='bo', label='a_n')
plt.stem(n_values, b_coeffs, 'r', markerfmt='ro', label='b_n')
plt.title('Коэффициенты a_n и b_n')
plt.xlabel('Номер гармоники (n)')
plt.ylabel('Значение')
plt.legend()
```

```
plt.grid(True)

plt.tight_layout()
plt.show()
```

Вывод такой:



Практика 4

Примеры формирования I/Q-сэмплов произвольной формы.

Работа с буфером приема SDR

Цель работы

Освоить технику формирования сложных сигналов и организацию одновременного приема/передачи.

Краткие теоретические сведения

Full-duplex режим позволяет одновременно передавать и принимать данные. MTU определяет оптимальный размер блоков данных.

Выполнение

Генерация многотонального сигнала

```
void generate_multitone_signal(int16_t *buffer, size_t num_samples, double
sample_rate) {
    double frequencies[] = {1000.0, 5000.0, 15000.0};
    double amplitudes[] = {800.0, 1200.0, 600.0};

    for (size_t i = 0; i < num_samples; i++) {
        double t = (double)i / sample_rate;
        double i_val = 0.0, q_val = 0.0;

        for (int j = 0; j < 3; j++) {
            i_val += amplitudes[j] * cos(2 * M_PI * frequencies[j] * t);
            q_val += amplitudes[j] * sin(2 * M_PI * frequencies[j] * t);
        }

        buffer[2*i] = (int16_t)i_val << 4;
        buffer[2*i + 1] = (int16_t)q_val << 4;
    }
}
```

Функция создает сигнал из суммы трех тонов с разными частотами и амплитудами.

Организация полного дуплекса

```
// Создание обоих потоков
SoapySDRStream *rxStream, *txStream;
SoapySDRDevice_setupStream(sdr, &rxStream, SOAPY_SDR_RX, SOAPY_SDR_CS16,
&channel, 1, NULL);
SoapySDRDevice_setupStream(sdr, &txStream, SOAPY_SDR_TX, SOAPY_SDR_CS16,
&channel, 1, NULL);

// Активация потоков
SoapySDRDevice_activateStream(sdr, rxStream, 0, 0, 0);
SoapySDRDevice_activateStream(sdr, txStream, 0, 0, 0);

// Цикл полного дуплекса
void *tx_buffs[] = {tx_buffer};
void *rx_buffs[] = {rx_buffer};

int tx_result = SoapySDRDevice_writeStream(sdr, txStream, tx_buffs, mtu, &flags,
timeNs, 100000);
int rx_result = SoapySDRDevice_readStream(sdr, rxStream, rx_buffs, mtu, &flags,
&timeNs, 100000);
```

Создаются и активируются одновременно RX и TX потоки, организуется цикл одновременной передачи и приема.

Вывод будет следующим

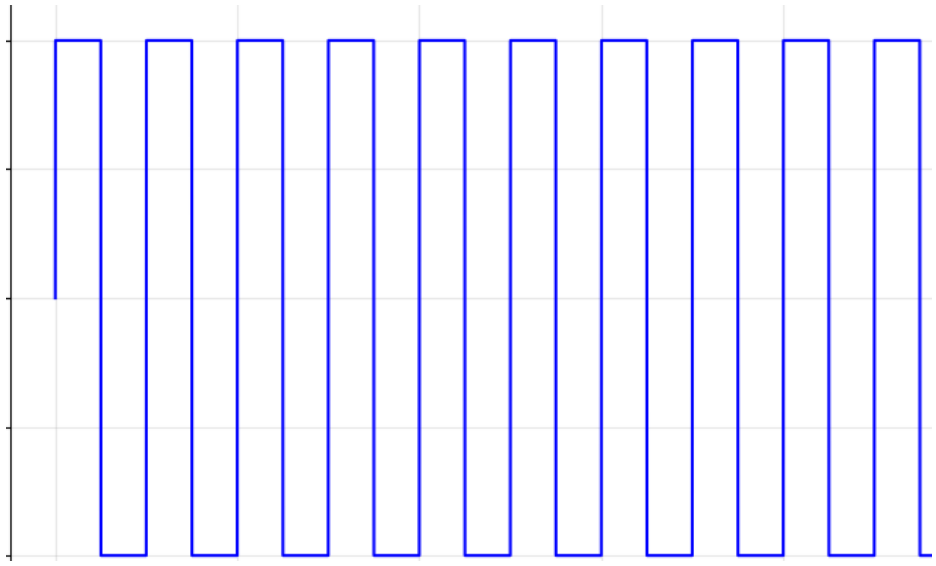
```
plutoSDR@tsvs317e14:~/SoapySDR/build/libiio/build/libad9361-iio/build/SoapyPlutoSDR/build/dev/build$ ./main
  Записано в файл: 1920 samples (7680 байт)
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
Buffer: 10 - Samples: 1920
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
Buffer: 20 - Samples: 1920
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
Buffer: 30 - Samples: 1920
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
Buffer: 40 - Samples: 1920
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
[WARNING] Backwards timestamp step!
Данные записаны в файл txdata.pcm
Программа завершена. Размер файла: 384000 байт
```

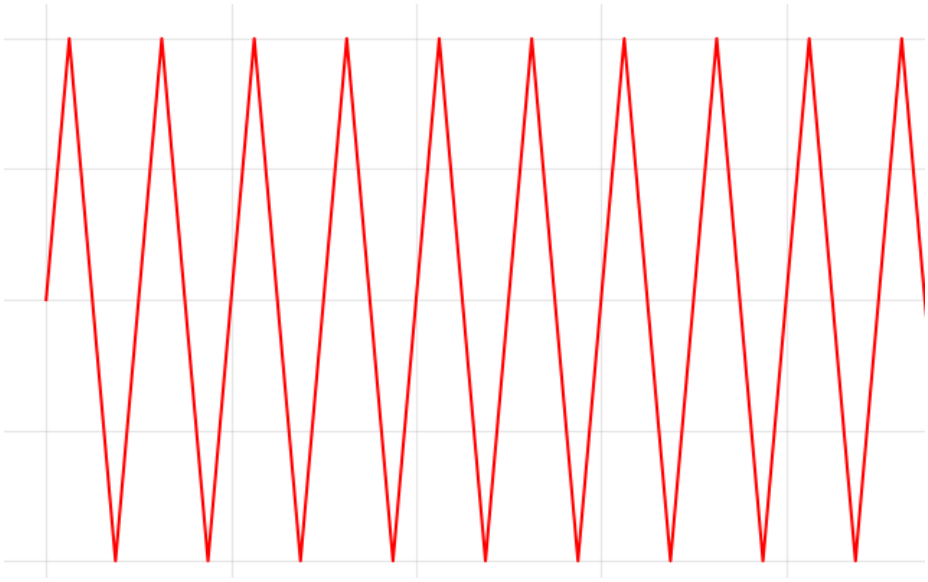
Генерация треугольных и квадратных сигналов

```c
#include <math.h>
#include <stdint.h>

// Генерация прямоугольного сигнала
void generate_square_wave(int16_t *buffer, size_t num_samples,
                          double amplitude, double frequency, double sample_rate)
{
    size_t samples_per_period = sample_rate / frequency;

    for (size_t i = 0; i < num_samples; i++) {
        int phase = (i / samples_per_period) % 2;
        int16_t value = (phase == 0) ? amplitude : -amplitude;

        buffer[2*i] = value << 4;       // I компонента
        buffer[2*i + 1] = 0 << 4;       // Q компонента = 0
    }
}
```

```c
// Генерация треугольного сигнала
void generate_triangle_wave(int16_t *buffer, size_t num_samples,
                            double amplitude, double frequency, double
sample_rate) {
    size_t samples_per_period = sample_rate / frequency;
    double step = (2.0 * amplitude) / (samples_per_period / 2);

    for (size_t i = 0; i < num_samples; i++) {
        size_t position_in_period = i % samples_per_period;
        int16_t value;

        if (position_in_period < samples_per_period / 2) {
            // Восходящий фронт
            value = -amplitude + (step * position_in_period);
        } else {
            // Нисходящий фронт
            size_t descending_pos = position_in_period - samples_per_period / 2;
            value = amplitude - (step * descending_pos);
        }

        buffer[2*i] = value << 4;       // I компонента
        buffer[2*i + 1] = 0 << 4;       // Q компонента = 0
    }
}
```

Графики выведем с помощью той же программы, что и для вывода обычных сэмплов





Практика 5
Имитация аналоговой передачи звука и его прием с использованием SDR. Анализ влияния чувствительности приемника и усиления передатчика на качество принятых отсчетов сигнала (семплов)

Цель работы

Реализовать имитацию аналоговой передачи аудиосигнала через SDR и исследовать влияние параметров усиления.

Основные теоретические сведения

Имитация аналоговой передачи звука с использованием SDR включает в себя модулирование аналогового звука, его передачу в радиоэфир и последующий приём с помощью SDR-приёмника. Чувствительность приёмника и усиление передатчика критически важны: низкая чувствительность или слишком низкое усиление могут привести к появлению помех и потере полезного сигнала, а излишне высокое усиление может вызвать перегрузку приёмника и искажение сигнала.

PCM (Pulse Code Modulation) - импульсно-кодовая модуляция, raw данные без сжатия
MP3 - формат сжатия аудио с потерями по стандарту MPEG-1/2 Layer 3
I/Q компоненты - синфазная и квадратурная составляющие комплексного сигнала
Частота дискретизации - определяет полосу частот сигнала (1 МГц для SDR)

Выполнение

Чтение PCM файла и разделение на I/Q компоненты

```
def read_pcm_file(filename):
"""Чтение PCM файла и разделение на I/Q компоненты"""
data = []
imag = []
real = []
count = []
counter = 0

print(f"Чтение PCM файла: {filename}")

with open(filename, "rb") as f:
index = 0
while (byte := f.read(2)):
if index % 2 == 0:
# I компонента (Real)
real_val = int.from_bytes(byte, byteorder='little', signed=True)
real.append(real_val)
counter += 1
count.append(counter)
else:
# Q компонента (Imaginary)
imag_val = int.from_bytes(byte, byteorder='little', signed=True)
imag.append(imag_val)
index += 1

print(f"Прочитано {len(real)} I/Q пар")
return np.array(real), np.array(imag), np.array(count)
```

Функция открывает бинарный PCM файл и читает данные по 2 байта за раз

```
index % 2 == 0 - четные байты относятся к I-компоненте (синфазной)

index % 2 == 1 - нечетные байты относятся к Q-компоненте (квадратурной)

int.from_bytes(byte, byteorder='little', signed=True) - преобразует 2 байта в 16-
битное целое число со знаком в little-endian формате
```

count - массив номеров samples для построения графиков

Возвращает три массива: real (I), imag (Q) и count (номера samples)

Анализ характеристик сигнала

```
def analyze_signal_characteristics(real, imag):
    """Анализ характеристик сигнала"""
    # Создаем комплексный сигнал
    complex_signal = real + 1j * imag

    print("\n=== АНАЛИЗ ХАРАКТЕРИСТИК СИГНАЛА ===")
    print(f"Общее количество samples: {len(complex_signal)}")
    print(f"Длительность (при 1 МГц): {len(complex_signal)/1e6:.3f} секунд")

    # Статистика
    print(f"\nСТАТИСТИКА I КОМПОНЕНТЫ:")
    print(f"  Среднее: {np.mean(real):.2f}")
    print(f"  СКО: {np.std(real):.2f}")
    print(f"  Максимум: {np.max(real)}")
    print(f"  Минимум: {np.min(real)}")

    # Мощность сигнала
    power = np.mean(np.abs(complex_signal)**2)
```

```
    print(f"\nМощность сигнала: {power:.2f}")

    return complex_signal
```

complex_signal = real + 1j * imag - создает комплексный сигнал из I и Q компонент

np.mean(real) - вычисляет среднее значение I-компоненты (показывает наличие DC смещения)

np.std(real) - стандартное отклонение (показывает разброс значений)

np.abs(complex_signal) - вычисляет амплитуду комплексного сигнала

Мощность сигнала - средний квадрат амплитуды, важный параметр для оценки уровня сигнала
Преобразование PCM в MP3

```
def pcm_to_mp3_conversion(real, imag, output_filename, original_sample_rate=1e6,
audio_sample_rate=48000):
    """Преобразование PCM в MP3 с ресемплингом"""

    # Используем только I компоненту для аудио
    audio_samples = real.astype(np.float32)

    # Нормализация к диапазону [-1, 1]
    max_val = np.max(np.abs(audio_samples))
    if max_val > 0:
        audio_samples = audio_samples / max_val
```

PCM данные имеют диапазон ±32767 (16-бит)
Для аудио обработки нужно нормализовать к ±1.0
audio_samples / max_val - масштабирует значения к диапазону [-1, 1]