МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Кафедра телекоммуникационных систем и вычислительных средств
(ТС и ВС)

Отчет по производственной практике
по дисциплине
SDR

по теме:

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

ЦЕЛЬ И ЗАДАЧИ

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

Задачи:

1. Прослушать и законспектировать лекцию, познакомиться с упрощенной архитектурой приемника.
2. Настроить работу ПК с SDR.
3. Поработать с SDR через C++.

ЛЕКЦИЯ

На прошлом занятии мы рассматривали архитектуру простого передатчика и то, как в нем формируется и отправляется сигнал.

На этом занятии разберем архитектуру простого приемника и то, как он принимает сигнал и преобразует его обратно в данные.

Архитектура простого приемника

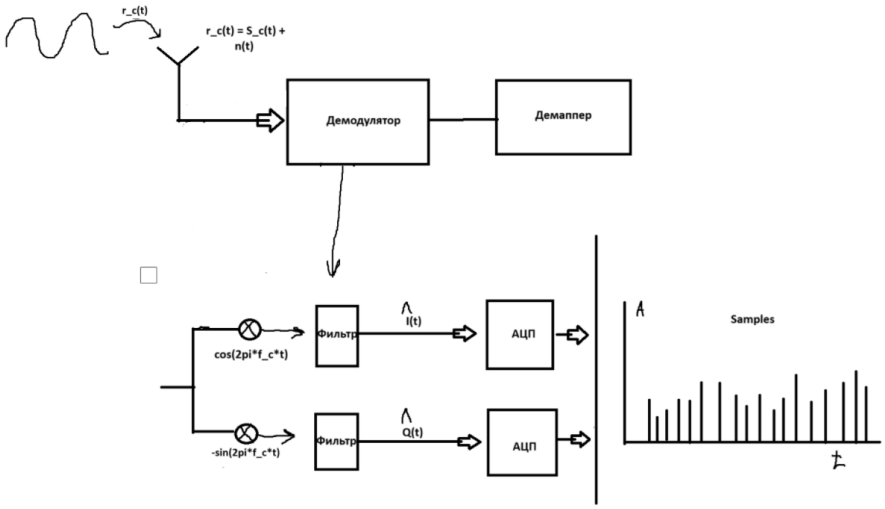

Рисунок 1 — Архитектура простого приемника

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 информаци 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)).

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

Рисунок 3 — Тригонометрические формулы

Теперь произведем перемножения входного сигнала на несущие:

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

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

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

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

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 семплы. В этих семплах нужно произвести символьную синхронизацию, чтобы правильно выделить переданную информацию, но это тема следующих занятий.

ПРАКТИЧЕСКАЯ ЧАСТЬ

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.

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 дескритизации, 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.

Установка необходимых библиотек и зависимостей

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
```

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 цифро-аналоговыми 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
```

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
```

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
```

Основне моменты работы с Adalm Pluto напрямую из C++

Подключение библиотек

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

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);
```

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];
```

Получение I/Q семплов

```
// 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 = SoapySDRDevice_readStream(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]);
    }
}
```

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);
```

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

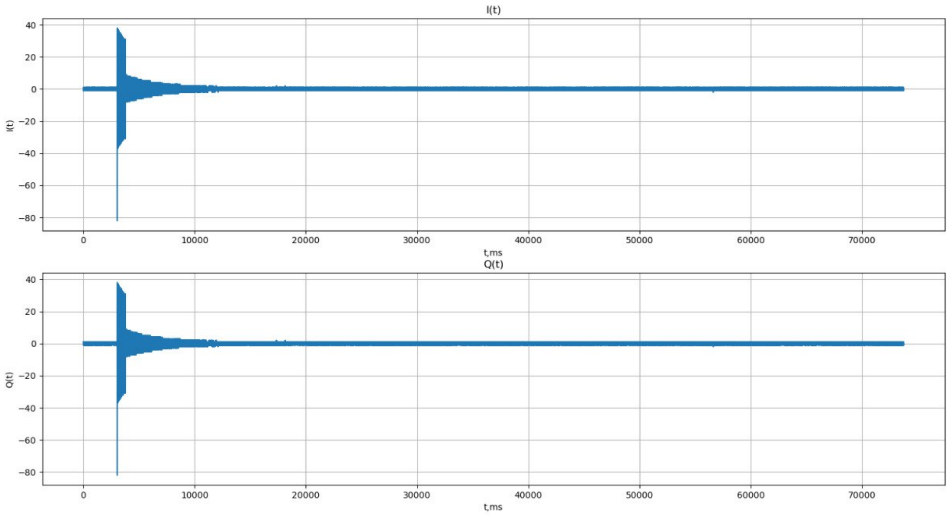

Рисунок 4 — Графики I(t) и Q(t)

Можем наблюдать, что графики напонимают прямоугольный сигнал.

ВЫВОД

В ходе проделанной работы я познакомился с архитектурой простого приемника. Познакомился с работой с Adalm Pluto SDR напрямую из C++. Отправил, а потом принял семплы и сделал их анализ.