

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:
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
Семинарист
Семинарист

Ахнашев А.В
Ахнашев А.В
Попович И.А

Новосибирск 2025 г.

СОДЕРЖАНИЕ

1	ЦЕЛЬ И ЗАДАЧИ	3
2	ЛЕКЦИЯ	4
3	ПРАКТИЧЕСКАЯ ЧАСТЬ	8
4	ВЫВОД	13

ЦЕЛЬ И ЗАДАЧИ

Цель:

Лучше освоить библиотеку SoapySDR, сформировать собственные семплы и попытаться отправить их, а потом принять и визуализировать полученный сигнал.

Задачи:

1. Лучше разобраться в библиотеке SoapySDR
2. Сформировать свои семплы
3. Принять семплы
4. Визуализировать сигнал

ЛЕКЦИЯ

Введение

На прошлом занятии мы работали с Adalm Pluto напрямую из C++, принимали семплы и записали их в файл. На этом занятии чуть более подробно углубимся в этот процесс и попробуем сформировать свои собственные семплы и отправим их с SDR.

Структура семплов в Pluto SDR

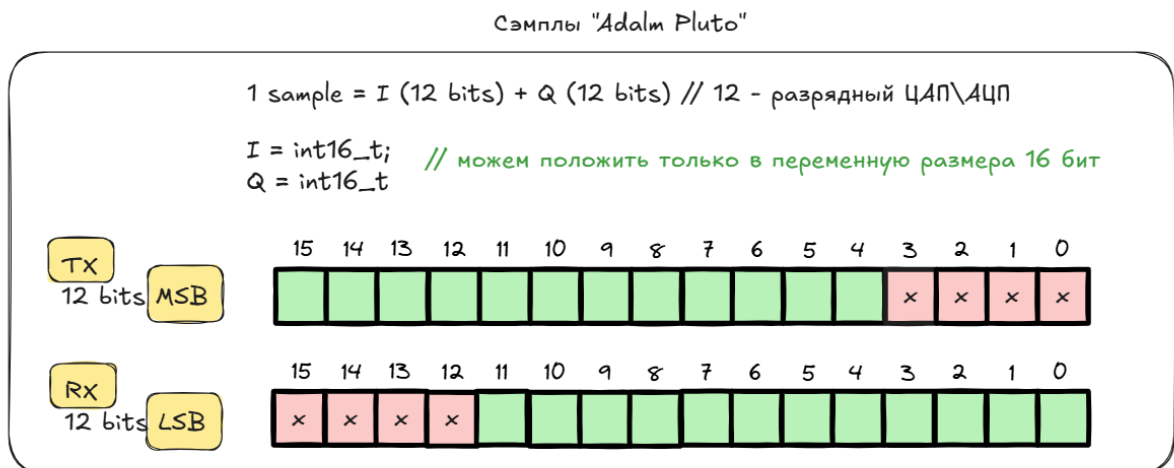


Рисунок 1 — Структура семплов

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 отрезка $[-2048; 2047]$. 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 или накладывать маски.

Стоит отметить, что Pluto SDR странно работает с TX семплами (которые хотим передавать) и интерпретирует как семплы последние 12 бит пере-

менной (если начинать отсчет от младшего бита), поэтому необходимо сдвигать значение I и Q на 4 бита влево («4») при работе с tx буффером, чтобы Pluto SDR корректно их интерпритировал.

Структура буффера семплов в Pluto SDR

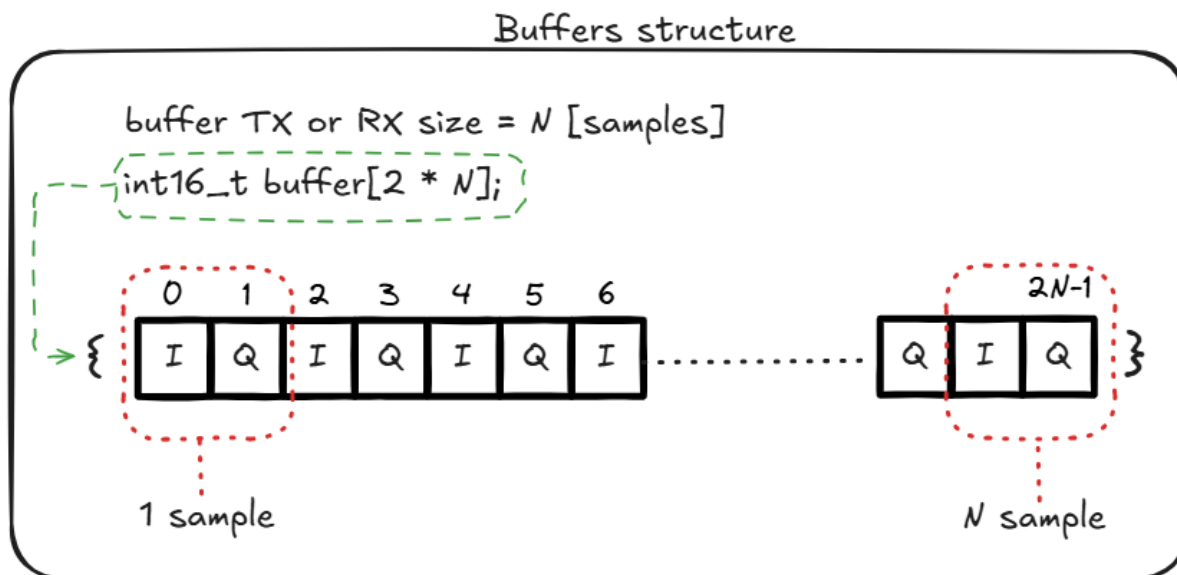


Рисунок 2 — Структура буффера

Семпл состоит из компонент I и Q, которые хранятся последовательно друг за другом, т.е в буфере 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.

Запись RX семплов в буффер в Pluto SDR

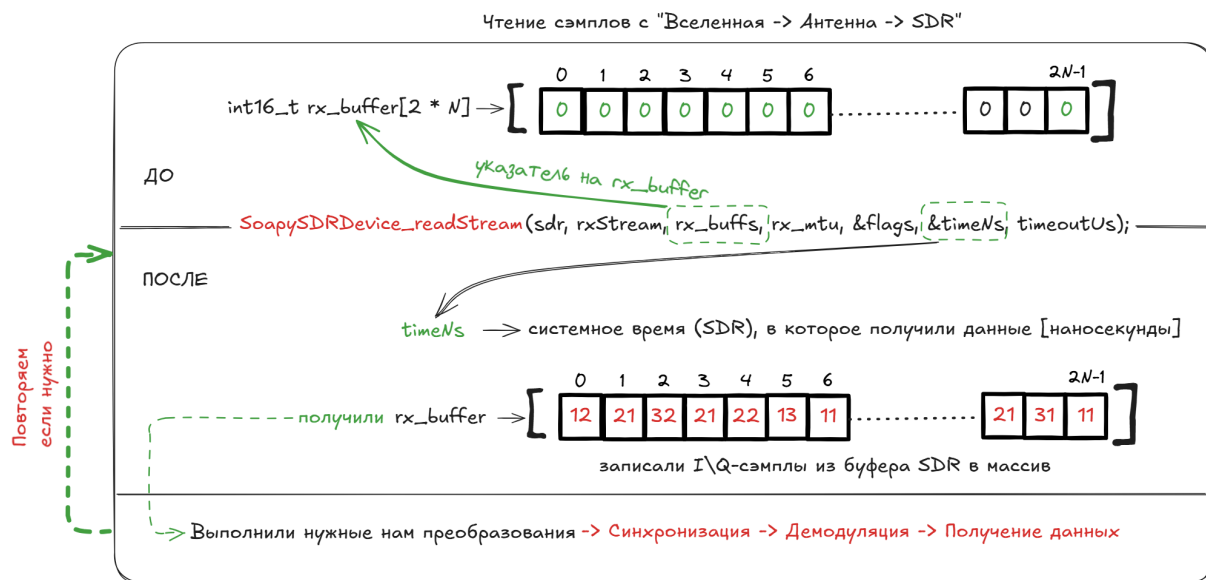


Рисунок 3 — Запись RX семплов

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.

Создание своих семплов и их отправка в Pluto SDR

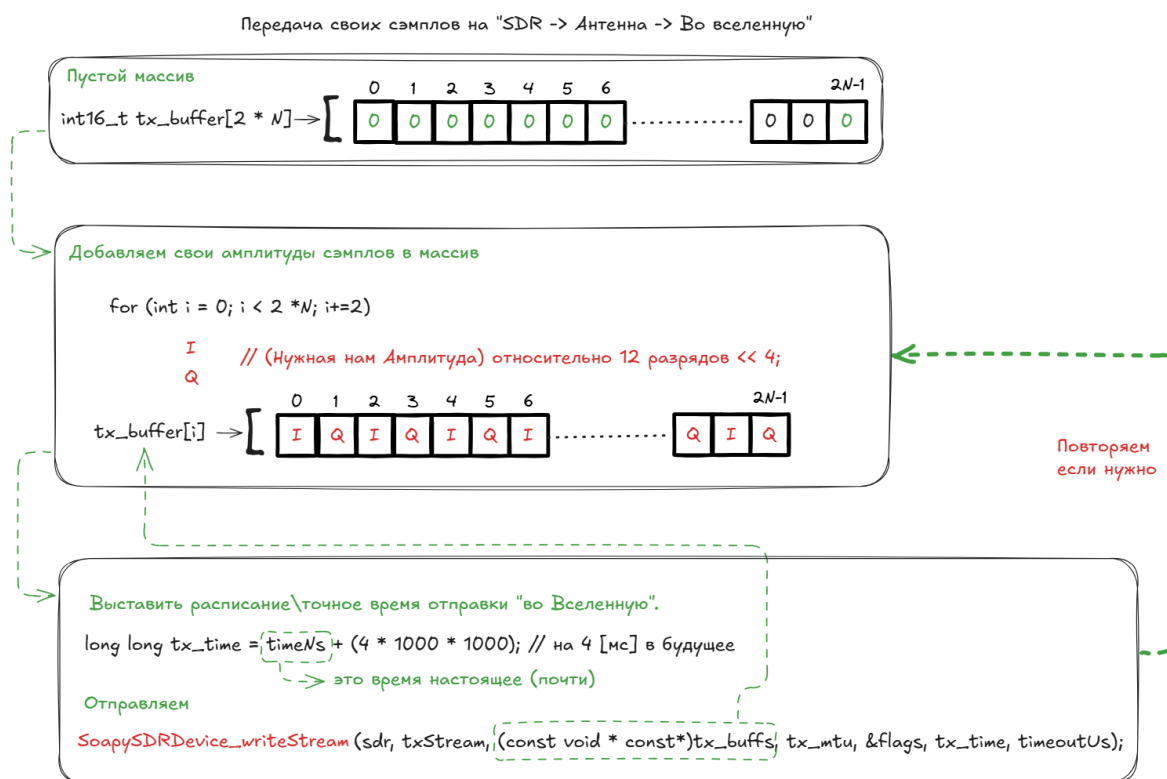


Рисунок 4 — Создание и отправка семплов

Для отправки семплов сначала необходимо заполнить tx буффер самими семплами. Формировать сами I и Q можно разными способами, но самое главное разместить их в правильном порядке. Для самой отправки используется функция `SoapySDRDevice_writeStream()`, в которую необходимо передать указатель на буффер с семплами и время, через которое семплы будут отправлены, а также дополнительные параметры. После вызова функции через 4нс наши семплы отправятся с Pluto SDR.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Введение

Необходимо заполнить tx буффер своими собственными семплами, отправить их, а потом принять и посмотреть, что примет Pluto SDR. То, как заполнять буффер семплами, решает студент.

Формирование семплов

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 специальную 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);  
  
    // massive 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 massive
```



```

        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 которую запишет число битов, чтобы знать размер массива 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:

```

01001000011001010110110001101100011011110010000001010111011011110111
00100110110001100100

```

Рисунок 5 — "Hello World" в бинарном виде

Способ кодирования

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

Заполнение буфера семплами

Для заполнения tx буфера я написал отдельную функцию, которая принимает битовую последовательность (сообщение), длину битовой последовательности и размер семпла, 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_BITS + 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_BITS). Чем выше длительность сигнала, тем выше шанс шанс верно декодировать его на принятой 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.

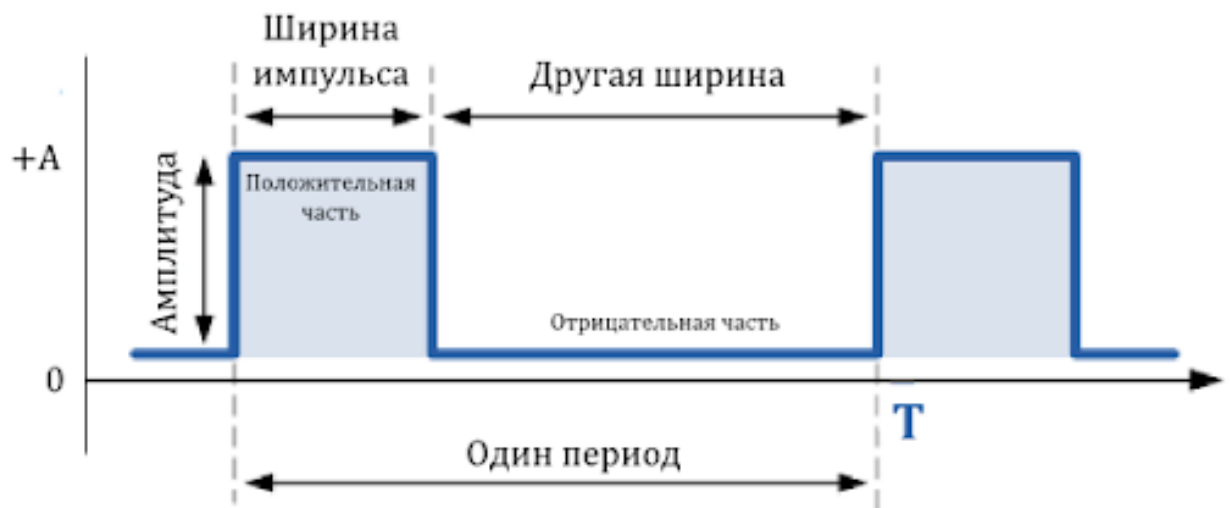


Рисунок 6 — Пример прямоугольного сигнала с обозначениями

Визуализация полученного сигнала с помощью скрипта на Python

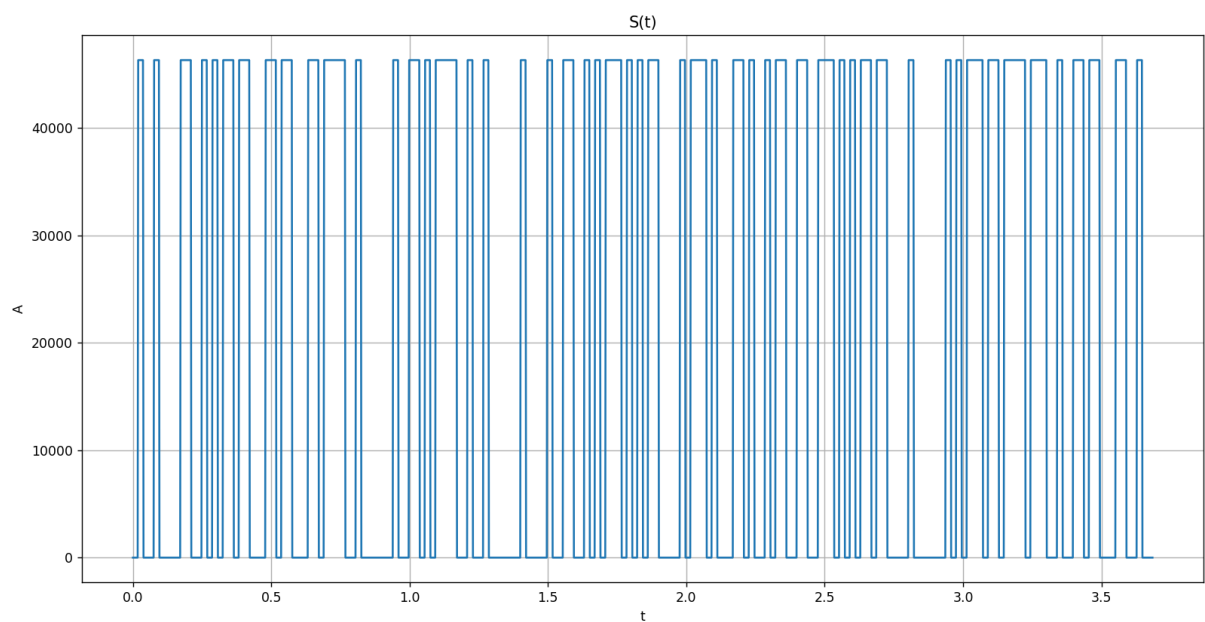


Рисунок 7 — Пример полученного прямоугольного сигнала

Этот прямоугольный сигнал содержит наше сообщение.

Требование к длине сообщения

Для 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.

Прием семплов

На приеме получили следующий сигнал:

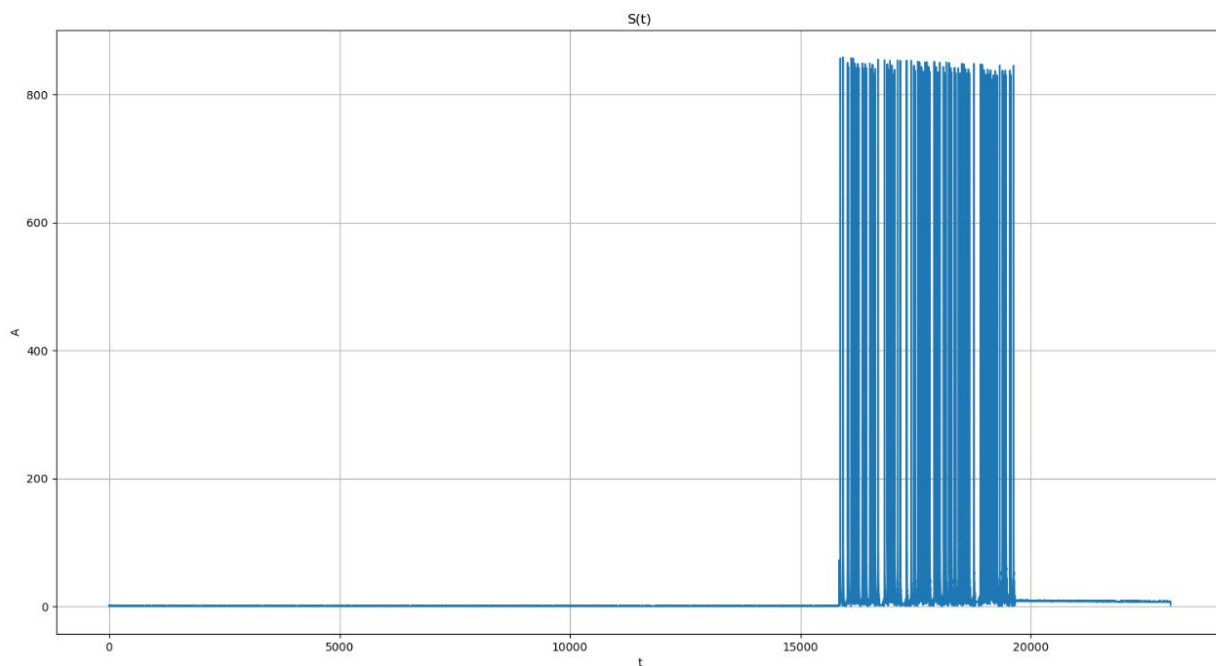


Рисунок 8 — Сигнал на приеме

Где то среди этого сигнала наше сообщение, но чтобы правильно его декодировать необходимо синхронизировать устройства с помощью специальных последовательностей. Эта тема будет рассматриваться в дальнейшем.

ВЫВОД

В ходе проделанной работы я лучше разобрался в библиотеке SoapySDR, сформировал собственные семплы, отправил их в радиоканал, а потом принял сигнал и визуализировал его.