# Клиентская часть проекта

Необходимо организовать потоковую передачу звука, снятого с микрофона на сервер. Задача состоит из трех частей:

1. Снятие данных с АЦП;
2. Обработка данных (опционально), и отправка их на сервер;
3. Организация одновременной работы оцифровывания данных и отправки их на сервер.
4. Способы ускорения

## Устройство микрофона

Перед тем, как описать процесс записи, разберёмся с тем, как работает микрофон.

В упрощённом виде, микрофон состоит из следующих частей: мембраны, прикреплённой к ней катушке индуктивности и магнита. Из курса физики, вы можете помнить, что при движении проводника в магнитном поле, в нём возникает индуцированный ток.

Звуком вы можете назвать колебания воздуха. Колебания воздуха представляют собой распространение в пространстве областей с высоким и низким давлением:



Когда эти области доходят до мембраны, они заставляют её колебаться (области низкого давления воздуха тянут мембрану в одну сторону, области высокого давления — давят в другую). Движение мембраны вызывает движение прикреплённой к ней катушки индуктивности. Движение катушки индуктивности в магнитном поле вызывает в ней индуцированный ток, представляющий собой электрический сигнал, форма которого повторяет форму звукового сигнала. Этот электрический сигнал и есть результат работы микрофона.

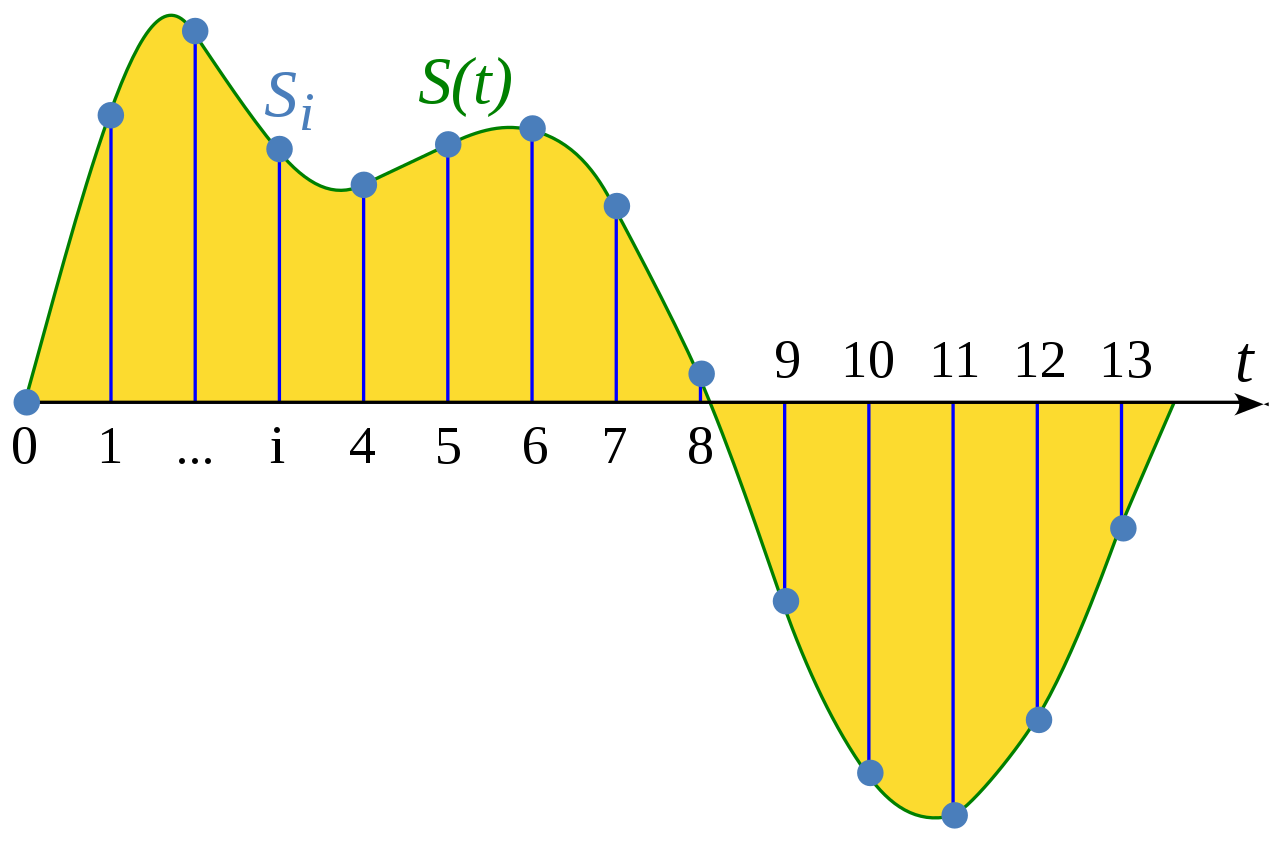
## Снятие данных с АЦП

В esp32 есть два многоканальных 12-битных АЦП. Вам достаточно выбрать один канал и настроить на него АЦП. Чтение значения на входе может быть выполнено функцией [adc1\_get\_raw()](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html#_CPPv412adc1_get_raw14adc1_channel_t). Перед вызовом этой функции, необходимо сконфигурировать АЦП. Помните, что АЦП 12-битный, поэтому желательно обнулить старшие биты результата вызова функции (там может содержаться мусор). Для этого необходимо наложить битовую маску:

res &= 0x0FFF;

Пример настройки и использования одиночного чтения вы можете увидеть [здесь](https://github.com/espressif/esp-idf/tree/master/examples/peripherals/adc/single_read/single_read).

Важно понимать, электрические колебания, которые вы оцифровываете — непрерывные. Одним из свойств непрерывных функций является то, что между каждыми двумя отсчётами этой функции есть ещё бесконечно много других отсчётов. Разумеется, вы не можете сохранить все отсчёты в памяти устройства, ведь их бесконечно. Но и сохранять их беспорядочно вы тоже не можете. Необходимо выполнить [дискретизацию](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F):



Непрерывной функции S(t) сопоставили дискретную функцию S(ti). То же необходимо сделать и вам. Шаг между двумя аргументами дискретной функции называется периодом T, а обратная ему величина 1/T называется частотой дискретизации и обозначается как f.

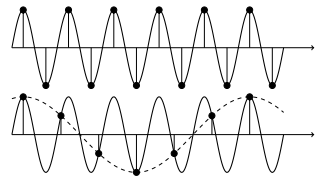
Чтобы понять, какой шаг дискретизации вам нужно выбрать, необходимо воспользоваться теоремой Котельникова.

## Теорема Котельникова

Представьте себе часы с циферблатом. Минутная стрелка часов движется в 12 раз быстрее часовой (за 60 минут минутная стрелка совершает полный оборот, а часовая только доходит до отметки в 1 час (или пять минут)).

Предположим, что начиная с времени 00:00, мы начинаем фотографировать циферблат. Каждую новую фотографию мы делаем примерно через 65.5 минут (В эти моменты минутная стрелка совершает полный оборот и догоняет часовую). На фотографиях будет видно лишь одну стрелку (разумеется, все зависит от исполнения часов, но предположим, что две стрелки внешним видом сливаются в одну). Если показать эти фотографии человеку, не знакомому с устройством часов, и сказать, что фотографии были сделаны с периодом в 65.5 минут, а затем попросить воспроизвести зафиксированный фотографиями процесс, то этот человек наверняка воспроизведёт лишь ход часовой стрелки, медленно перемещая её подобно тому, как это происходило на фотографиях. Информация о перемещениях минутной стрелки будет безвозвратно потеряна. Делая фотографии чаще, мы бы могли увидеть на них ещё одну стрелку и характер её движения.

Рассмотрим ещё один пример:



Имеется один и тот же непрерывный сигнал, который оцифровали двумя способами. В первом случае, сигнал оцифровывали с частотой, вдвое превышающей частоту сигнала (на один период синусоиды пришлось два дискретных отсчёта). Во втором случае, сигнал оцифровывали с частотой, меньшей чем частота сигнала. Пунктирная линия показывает, какой сигнал, мы можем получить, если попытаемся восстановить его по имеющимся отсчётам (можем спутать высокочастотный сигнал с низкочастотным).

Описанный в двух примерах выше эффект называется эффектом наложения частот, или алиасингом.

Теорема Котельникова описывает способ оцифровки сигнала, позволяющий избежать этого эффекта:

*любую функцию F(t), спектр которой ограничен 0 и f1, можно непрерывно передавать с любой точностью при помощи чисел, следующих друг за другом через 1/(2f1) секунд.*

Иными словами, для того, чтобы точно восстановить непрерывный сигнал, достаточно оцифровать его с частотой, вдвое превышающей наибольшую частоту в спектре сигнала.

В человеческой речи используются частоты от 300 до 3400 Гц. Если округлить верхнюю границу до 4000 Гц, чтобы иметь запас, то из теоремы Котельникова получим, что частота дискретизации должна быть 8000 Гц. Именно эту частоту дискретизации использовали все голосовые аудиокодеки.

**Таким образом, 8000 отсчётов в секунду — минимальная частота сэмплирования, к которой вы должны стремиться, снимая данные с АЦП.**

Ниже приведены две записи, одна с подходящей частотой дискретизации, другая — с меньшей чем необходимо:

* [запись с подходящей частотой;](https://nag.ru/upload/article-files/20190208-0001Vinni_orig.mp3)
* [запись с частотой, меньшей чем необходимо](https://nag.ru/upload/article-files/20190208-0001Vinni_Alias.wav).

## Отправка их на сервер

Вы уже создавали tcp-клиент, и отправляли данные на сервер. Эта часть ничем не отличается, от того, что вы уже делали. Важно помнить следующее:

При снятии отсчётов с АЦП, их необходимо располагать в памяти один за другим, без каких-либо разрывов — так вы сможете отправлять данные одним вызовом send. Этого можно достичь, используя массивы.

Кроме того, поскольку АЦП 12-битный, каждый отсчёт будет занимать два байта памяти, а значит вам необходимо позаботиться о порядке байт.

Может показаться, что поскольку передача будет потоковая, вам больше подойдёт UDP, однако лучше воздержаться от использования этого протокола, поскольку вам придётся позаботиться о порядке пришедших пакетов и [их размере](https://ru.wikipedia.org/wiki/UDP#%D0%94%D0%BB%D0%B8%D0%BD%D0%B0_%D0%B4%D0%B0%D1%82%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B).

## Организация одновременной работы оцифровывания данных и отправки их на сервер

Для оцифровки данных с заранее заданной частотой, удобно будет воспользоваться [высокоточным таймером](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/esp_timer.html). Примеры его использования вы можете найти [здесь](https://github.com/espressif/esp-idf/tree/95b3dad771/examples/system/esp_timer). Важно понимать, что в функции-обработчике, срабатывающей по каждому тику таймера, вы должны совершать как можно меньше операций, поскольку должны завершить её выполнение до следующего тика. Это значит, что отправку оцифрованных данных лучше совершать где-то в другом месте.

Лучшим способом, будет организовать два буфера. Сначала оцифрованные данные складываются в первый буфер, затем, когда он будет заполнен, данные начнут складываться во второй буфер, а первый в этот момент начнёт отправляться на сервер. Затем, когда второй буфер будет заполнен, данные начнут складываться в первый, а содержимое второго будет отправлено на сервер и т.д.

Поскольку esp32 использует RTOS в качестве операционной системы, вы можете использовать многозадачность (снимать данные по таймеру, одновременно с отправкой). Кроме того, в процессоре esp32 используется два ядра, что даёт ещё больше возможностей для многозадачности. Работа программы в esp32 основана на так называемых “тасках” (задачах). У каждой задачи есть свой приоритет и способ вызова. Какие-то задачи работают постоянно, какие-то запускаются по событиям. Важно понимать, что даже если задача работает постоянно, при правильной организации её выполнение может быть временно приостановлено для выполнения более приоритетной задачи.

Для данной задачи предлагается использовать один таск и одну функцию обратного вызова ([колбэк](https://ru.wikipedia.org/wiki/Callback_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))). Таск будет ожидать наполнения буфера и отправлять данные на сервер, а колбэк будет срабатывать по каждому срабатыванию таймера, снимать данные с АЦП и класть их в буфер. Чтобы колбэк мог сообщить таску о том, что буфер наполнен, можно использовать [очередь](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html#queue-api) (для этого используется специальный тип QueueHandle\_t. В колбэке таймера будет вызываться функция xQueueSend, а в таске — xQueueReceive.

## Способы ускорения

Главной проблемой подобной реализации является то, что одиночное считывание данных с АЦП является очень долгой операцией, которая должна успеть завершиться до очередного срабатывания таймера.

Есть два способа быстрого считывания большого количества данных с АЦП: использование контроллера прямого доступа к памяти (DMA) и использование аудио контроллера i2s.

### DMA

Контроллер прямого доступа к памяти позволяет периферийному устройству (АЦП) записывать данные в память напрямую, минуя процессор. В этом случае, процессору достаточно настроить контроллер DMA (какое количество отсчётов, с какой частотой и куда их необходимо записать), после чего контроллер DMA возьмёт на себя всю работу по получению данных с АЦП, а процессор сможет использовать свои вычислительные возможности для прочих целей (например, отправки данных по TCP).

Пример использования получения данных с АЦП с помощью DMA вы можете найти [здесь](https://github.com/espressif/esp-idf/tree/master/examples/peripherals/adc/dma_read).

### I2S

[I2S (Inter-IC Sound)](https://ru.wikipedia.org/wiki/I%C2%B2S) — это последовательный синхронный протокол обмена, который обычно используется для передачи аудиоданных между двумя цифровыми устройствами. ESP32 содержит два i2s-контроллера, каждый из которых может работать как передатчик (выводить данные на динамик), так и приёмник (получать данные с микрофона). В esp-idf есть api как для работы с i2s-аудиоустройствами, так и АЦП/ЦАП напрямую. Это значит, что данные с микрофона можно получать через api i2s. Кроме того, i2s-контроллер в процессе работы может использовать dma, что так же позволит снять все вычислительные затраты с процессора, оставив на том только необходимость в настройке контроллера, и вызова функций чтения.

Пример чтения данных с аналогового микрофона контроллером i2s вы можете найти [здесь](https://github.com/espressif/esp-idf/tree/master/examples/peripherals/i2s/i2s_adc_dac).

## Рекомендации

Рекомендуется сперва попробовать выполнить задачу на одиночных чтениях с АЦП, а уже после достижения минимально рабочего результата пробовать улучшить его путём использования dma/i2s.

За основу проекта лучше взять пример tcp-клиента. В этом случае, у вас уже будут реализована функция подключения к wifi (example\_connect). Настройка данных wifi-сети выполняется через менюконфиг (вызывается командой idf.py menuconfig). В подменю Example Connection Configuration. Кроме того, в подменю Example Configuration можно указать ip-адрес и порт сервера (но можно это сделать и из файла с исходниками, в то время как настройки wifi в исходниках будет указать сложнее).

# Серверная часть проекта

Данные, которые будет передавать клиент — это двухбайтные отсчёты, снятые с АЦП. Иными словами, это амплитуда колебаний воздуха. Чем выше значение — тем выше была амплитуда колебаний, следовательно — тем выше был звук. Высота звука зависит от частоты колебаний этих амплитуд.

Серверу необходимо принять соединение клиента, и начать считывать отправляемые данные (позаботившись о порядке следования байт). Эти данные можно записывать напрямую в wav-файл (вы уже работали с wav-файлами на предыдущих этапах олимпиады), без выполнения какого-либо сжатия или кодирования. Единственное, что необходимо сделать перед записью – это настроить данный файл на частоту, с которой звук микрофона был оцифрован, и указать разрядность данных (2 байта). Однако, следует обратить внимание на то, что поскольку АЦП является двухбитным, а в wav-файле будет указано, что данные в нем шириной в два байта (шестнадцатибитные), воспроизводимый звук будет в 16 раза тише максимальной громкости. Это можно исправить путём простого умножения всех значений на 4-8.