**АННОТАЦИЯ**

В данной работе создано устройство для настройки акустической гитары на базе микроконтроллера ESP32-S3. На вход системы подаётся сигнал от пьезоэлектрического датчика, закреплённого на деке гитары. Сигнал усиливается предусилителем и поступает на аналогово-цифровой преобразователь микроконтроллера, где преобразуется в цифровую форму и далее обрабатывается программно.  
 В цифровой части реализованы алгоритмы спектрального анализа, основанные на быстром преобразовании Фурье и методе Harmonic Product Spectrum для выделения основной частоты сигнала. Для повышения точности применяются оконные функции, параболическая интерполяция и медианная фильтрация.  
 Полученная частота сравнивается с эталонными значениями строя гитары, после чего результат выводится на индикацию. Разработанный тюнер обеспечивает высокую точность, устойчивость к шуму и пригоден для практического использования.

**SUMMARY**

In this work, a device for tuning an acoustic guitar based on the ESP32-S3 microcontroller has been created. The system receives a signal from a piezoelectric sensor mounted on the guitar's soundboard. The signal is amplified by a preamplifier and fed to the microcontroller's analog-to-digital converter, where it is converted into a digital form and further processed programmatically.

The digital part implements spectral analysis algorithms based on fast Fourier transform and Harmonic Product Spectrum method for extracting the fundamental frequency of the signal. To improve the accuracy, window functions, parabolic interpolation and median filtering.

The obtained frequency is compared with the reference values of the guitar tuning, after which the result is displayed on the display. The developed tuner provides high accuracy, noise resistance and is suitable for practical use.

Оглавление

[ВВЕДЕНИЕ 4](#_Toc28440)

[1.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc12095)

[1.1. Спектральные компоненты 5](#_Toc30550)

[1.1.1 Обертон 5](#_Toc285)

[1.1.2 Гармоника 6](#_Toc2701)

[1.1.3 Роль обертонов и гармоник 7](#_Toc21280)

[1.1.4 Анализ алгоритмов обработки 7](#_Toc32560)

[1.2 Таблица стандартных частот строя гитары. 8](#_Toc3159)

[1.3. Принципы цифровой обработки сигналов для анализа звука 9](#_Toc2258)

[1.3.1. Принцип аналогово-цифрового преобразования 9](#_Toc24510)

[1.3.2 Ключевые параметры АЦП 9](#_Toc3407)

[1.3.3. Влияние параметров АЦП на точность тюнера 11](#_Toc26452)

[1.4. Принципы работы выбранных алгоритмов анализа. 12](#_Toc1037)

[1.4.1 FAST FOURIER TRANSFORM 12](#_Toc29882)

[1.4.2 Оконные функции 14](#_Toc14490)

[1.3.3 HPS Harmonic Product Spectrum 15](#_Toc11250)

[1.3.5 Медианная фильтрация 18](#_Toc21237)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 20](#_Toc24825)

[1. Разработка аппаратной части 20](#_Toc23350)

[1.1 Моделирование схемы 20](#_Toc14160)

[1.2. Элементы сборки 23](#_Toc23422)

[1.3 Тестирование схемы 23](#_Toc15026)

[2. Разработка программного обеспечения 26](#_Toc6733)

[ПРОГРАММА ОБРАБОТКИ CSV ФАЙЛОВ С ПОМОЩЬЮ PYTHON 26](#_Toc8836)

[2. Анализ результатов тестирования 29](#_Toc30904)

[Заключение 53](#_Toc15813)

**ВВЕДЕНИЕ**

Настройка музыкального инструмента является важной задачей, напрямую влияющей на качество звучания и удобство исполнения. Традиционные механические камертоном или слуховые методы требуют опыта исполнителя и подвержены человеческому фактору. Современные электронные тюнеры позволяют автоматизировать процесс настройки, обеспечивая высокую точность и стабильность результатов.

В данной работе разработан электронный тюнер для акустической и классической гитары, сочетающий аналоговую и цифровую обработку сигнала. В качестве датчика используется пьезоэлемент, преобразующий колебания деки гитары в электрический сигнал. Далее сигнал усиливается предусилителем, после чего поступает на вход микроконтроллера ESP32-S3, выполняющего оцифровку и анализ.

В цифровой части реализованы методы обработки сигналов, обеспечивающие надёжное определение основной частоты даже при наличии шумов, негармонических составляющих и затухания струны. Использование БПФ, алгоритма HPS, параболической интерполяции и медианной фильтрации позволяет повысить точность определения частоты до десятых долей герца и обеспечить устойчивость показаний.

Целью проекта является разработка аппаратно-программной системы тюнера, включающей разработку схемы предусилителя, обоснование параметров АЦП, реализацию алгоритмов цифровой обработки сигналов и программной логики работы устройства.

**1.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1.1. Спектральные компоненты**

**Основной тон (основная частота)**

Когда музыкальный инструмент издаёт звук, возникает частота, которая воспринимается как высота звука. Эта частота называется основной частотой.

Основной тон задаёт базовую высоту звука, но никогда не звучит изолированно. Вместе с ним возникают дополнительные частоты, которые формируют полный спектр звучания. Эти частоты называются обертонами.

**1.1.1 Обертон**

Обертон представляет собой дополнительный тон, который звучит выше основного и связан с ним целочисленными кратными. У каждого звука есть свои обертоны, возникающие на строго определённых частотах.

Обертоны - это дополнительные частоты, которые звучат одновременно с основной частотой. Они появляются из-за сложных колебаний звукового источника, таких как струна или мембрана барабана.

Обертоны могут быть гармоническими (кратными основной частоте) или негармоническими (не кратными). Например, если основной тон звучит на частоте 100 Гц, то обертоны могут быть как 200 Гц, 300 Гц (гармонические), так и 275 Гц или 390 Гц (негармонические). Гармонические частоты образуют «упорядоченную» часть спектра, а негармонические - «хаотичную».

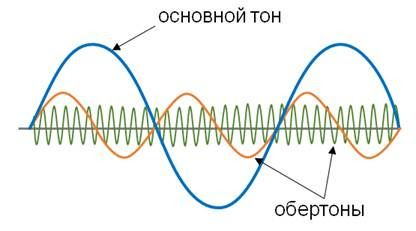


Рисунок 1 - Обертон.

**1.1.2 Гармоника**

Гармоника - это частота, выраженная целым числом герц, кратным основной частоте. Гармоники входят в группу обертонов, строго соответствующих кратным значениям.

Первая гармоника = основной тон (например, 100 Гц).

Вторая гармоника = 2×100 = 200 Гц.

Третья гармоника = 3×100 = 300 Гц и так далее.

Гармоники определяют музыкальность и стройность звука, особенно в музыкальных инструментах. Негармонические обертоны, могут создавать ощущение шума или нестабильности звука.

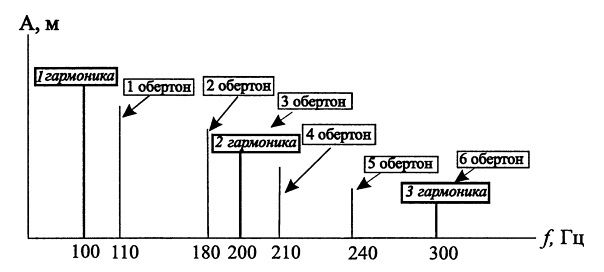


Рисунок 2 - Расположение обертонов.

**1.1.3 Роль обертонов и гармоник**

Частота колебаний струны зависит от множества факторов: её натяжения, материала и длины. Особое значение имеет длина струны, поскольку её изменение напрямую влияет на характер звучания, позволяя извлекать дополнительные звуки, известные как обертоны или гармоники.

Колебание целой струны даёт основной тон. Половинки струны создают первую гармонику. Третья часть - вторую гармонику и так далее.

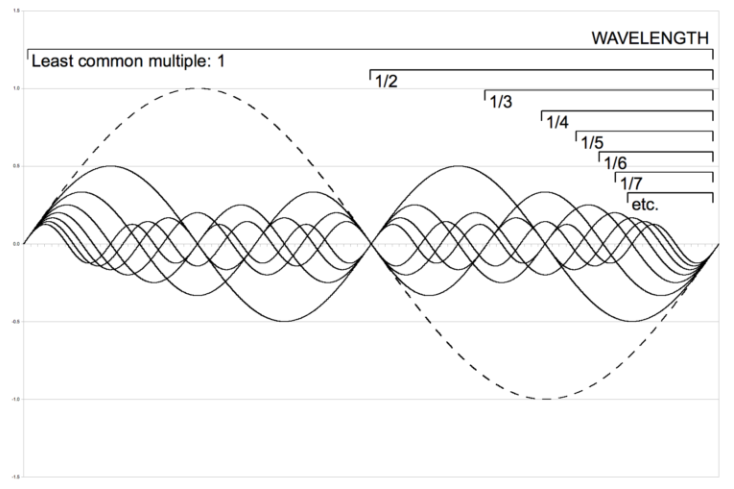


Рисунок 3 - Колебания. Появление обертонов.

**1.1.4 Анализ алгоритмов обработки**

Когда звучит гитарная струна:

Основной тон - самая низкая и самая громкая частота, которую мы воспринимаем как ноту.

Обертоны - более высокие частоты, которые звучат одновременно с основным тоном, но тише.

Отношение к основному тону вычисляется как n\*f0, где f0 это основной тон.

Основной тон - самый низкий по частоте из сильных пиков и зачастую самый громкий, но не всегда. Исходя из этого в программной части надо обрабатывать ситуации, в которых гармоники f2, f3… мощнее фундаментального пика f1, для обработки ситуаций, в которых алгоритм находит обертон и считает его за основную частоту.

**1.2 Таблица стандартных частот строя гитары.**

Настройка струны производится по контрольному звуку - звуку камертона, который настроен на ноту «ля» первой октавы. В данной работе используются эталонные значения, представленные в таблице 1.

Таблица 1 - Частоты открытых струн для классического стоя.

|  |  |  |
| --- | --- | --- |
| **Струна** | **Нота** | **Частота, Гц** |
| 1 | e¹ | 329,63 |
| 2 | b | 246,94 |
| 3 | g | 196,00 |
| 4 | d | 146,83 |
| 5 | A | 110,00 |
| 6 | E | 82,41 |

**1.3. Принципы цифровой обработки сигналов для анализа звука**

**1.3.1. Принцип аналогово-цифрового преобразования**

Аналогово-цифровое преобразование - процесс преобразования непрерывного аналогового сигнала в дискретный цифровой код, который может обрабатываться цифровыми системами.

Процесс включает три основных этапа:

1. Дискретизация (сэмплирование) - взятие отсчетов аналогового сигнала через равные промежутки времени
2. Квантование - представление каждого отсчета в виде конечного числа уровней
3. Кодирование - преобразование квантованных значений в цифровой код

**1.3.2 Ключевые параметры АЦП**

**Частота дискретизации:**

Частота дискретизации (f\_s) - количество отсчетов аналогового сигнала, взятых за одну секунду.

При f\_s = 10 кГц микроконтроллер измеряет мгновенное значение напряжения с пьезодатчика 10 000 раз в секунду

Интервал между отсчетами: Δt = 1/f\_s = 1/10000 = 0.1 мс

**Теорема Котельникова-Найквиста:**

Для точного восстановления аналогового сигнала из его дискретных отсчетов частота дискретизации должна быть как минимум в два раза выше максимальной частоты спектра сигнала: f\_s >= 2\*f\_max

где:

f\_s - частота дискретизации

f\_max - максимальная частота в спектре сигнала

Частота Найквиста: f\_N = f\_s / 2

Если в сигнале будут присутствовать частоты выше f\_N, они отразятся в рабочую полосу частот, что приведет к необратимым искажениям сигнала и некорректной работе тюнера.

**Разрядность:**

Разрядность - количество бит, используемых для представления одного отсчета. Определяет динамический диапазон и разрешение АЦП.

Для работы с ESP32-S3, максимальная разрядность: 12 бит (встроенный АЦП), количество уровней квантования: 2^12 = 4096

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Разрядность | 12 бит |
| Уровней квантования | 4096 |
| Напряжение опорное | 3.3 В |
| Шаг квантования | 3.3В / 4096 ≈ 0.8 мВ |

**Обоснование выбора параметров для ESP32-S3:**

Для гитарного тюнера:

Максимальная частота 1-й струны в классическом строе: 329.63 Гц

Выбор частоты дискретизации: 10 кГц

Количество значимых гармоник доходит 4-5 от основной частоты, то есть требуется 1.648 кГц. Тогда теоретически будет достаточно f\_s >= 3.2 кГц

Для обеспечения запаса и улучшения точности выбрано значение f\_s = 10 кГц

1. Теорема Найквиста: f\_s >= 2 × 1648 Гц = 3.3 кГц
2. БПФ:

БПФ дает N/2 частотных отсчетов

При f\_s = 10 кГц и N=2048: Δf = 10000/2048 = 4.9 Гц

Для точного определения частоты потребуется интерполяция.

3. Выбор разрядности: 12 бит

Таблица 2 - Сравнение вариантов разрядности.

|  |  |  |
| --- | --- | --- |
| **Разрядность** | **Уровней** | **Коментарий** |
| 12 бит | 4096 | Хорошее соотношение точность/скорость  Ограниченный динамический диапазон |
| 16 бит | 65536 | Высокая точность, большой динамический диапазон  Требуются дополнительные компоненты, сложность реализации возрастает |
| 10 бит | 1024 | Быстрее, меньше памяти  Низкая точность для слабых сигналов |
| 8 бит | 256 | Минимальные требования  Мало для обработки аудио |

1. Динамический диапазон сигнала:

Сигнал с пьезодатчика: 100-400 мВ до 1-2 В

Отношение сигнал/шум: 40-60 дБ

12 бит (66 дБ) покрывает этот диапазон

1. Требования к точности определения частоты:

Целевая точность: 0.5 Гц

**1.3.3. Влияние параметров АЦП на точность тюнера**

**Погрешность из-за дискретизации. Ошибка определения частоты:**

Δf\_дискретизация = f\_s / N

Δf = 10000 / 2048 ≈ 4.9 Гц

Для уменьшения ошибки используется:

1. Интерполяция - уменьшает ошибку до 0.1-0.5 Гц
2. Метод HPS - улучшает разрешение в частотной области.

**Погрешность из-за квантования**

**Теорема Крамера-Рао для оценки дисперсии частоты:**

f\_KR = (6\*fs²) / ((2π)² \* SNR \* N³)

Для параметров:

fs = 10000 SNR = 100 N = 2048

погрешность составляет примерно 0.05 Гц

Вывод: Параметры АЦП (10 кГц, 12 бит) обеспечивают достаточную точность для гитарного тюнера.

**1.4. Принципы работы выбранных алгоритмов анализа.**

**1.4.1 FAST FOURIER TRANSFORM**

Быстрое преобразование Фурье - [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) ускоренного вычисления [дискретного преобразования Фурье](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A4%D1%83%D1%80%D1%8C%D0%B5). Под быстрым преобразованием Фурье понимается один из алгоритмов, называемый алгоритмом прореживания по частоте - времени, имеющий [сложность](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) O(Nlog(N)).

Пошаговый результат работы X = np.fft.fft(x):

Допустим, x\_clean - это массив из N вещественных чисел.

Шаг 1: Входные данные (временная область)

x\_clean = [x[0], x[1], x[2], ..., x[N-1]]

Сигнал во временной области (например, напряжение, записанное в моменты времени t0, t1, ...).

Шаг 2: Выполнение БПФ

Алгоритм (Cooley-Tukey) эффективно вычисляет формулу ДПФ:

X[k] = Σ (x[n] \* exp(-2πj \* k \* n / N)) для n = 0...N-1

Где:

j - мнимая единица.

exp(-2πj \* k \* n / N) - это комплексная экспонента, представляющая гармонику с частотой k.

Шаг 3: Выходные данные (частотная область)

На выходе получается массив X той же длины N, но состоящий из комплексных чисел.

X = [X[0], X[1], X[2], ..., X[N-1]]

X[0] - это постоянная составляющая (DC offset). Среднее значение сигнала. Амплитуда "нулевой частоты".

X[1] до X[N//2 - 1] - положительные частоты.

X[1] соответствует частоте 1 \* Fs / N (самая низкая нетривиальная частота).

X[2] соответствует частоте 2 \* Fs / N…

Fs - частота дискретизации.

X[N//2] - это частота Найквиста (Fs / 2). Максимальная частота, которую можно восстановить по данному сигналу.

X[N//2 + 1] до X[N-1] - отрицательные частоты (для вещественного входного сигнала они являются комплексно-сопряженными к положительным и не несут новой информации).

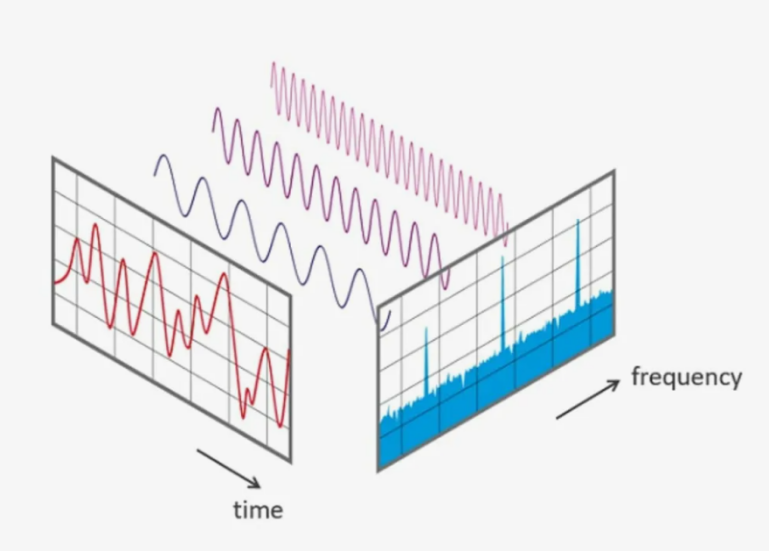


Рисунок 4 - Быстрое преобразование Фурье.

**1.4.2 Оконные функции**

Для аналоговых сигналов его спектр - дельта-функция на частоте сигнала. На практике спектр реального ограниченного по времени гармонического сигнала эквивалентен функции sin(x)/х=sinc(x) а ширина главного лепестка зависит от длительности интервала анализа сигнала Т. Ограничение по времени - умножение сигнала на прямоугольную огибающую.

В задачах цифровой обработки сигналов созданы окна различной формы, которые при наложении на сигнал во временной области, позволяют качественно улучшить его спектральные характеристики.

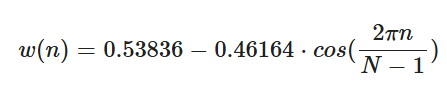
Правило: чем сильнее подавление боковых лепестков спектра, тем шире главный лепесток спектра, и наоборот.

Так как анализ сигнала выполняется на конечном временном интервале фиксированной длины, реальный сигнал «обрезается» прямоугольным окном. Это приводит к спектральной утечке — энергии одной частоты распределяется по соседним частотным бинам БПФ, образуя боковые лепестки.  
 В результате пики гармоник «размываются», появляются ложные спектральные максимумы, ухудшается точность определения основной частоты, особенно при наложении гармоник.

Чтобы снизить эту проблему, сигнал умножается на оконную функцию.

Выбранное окно в проекте: Окно Хэмминга.

Характеристики:



Максимальный уровень боковых лепестков: -42 дБ.

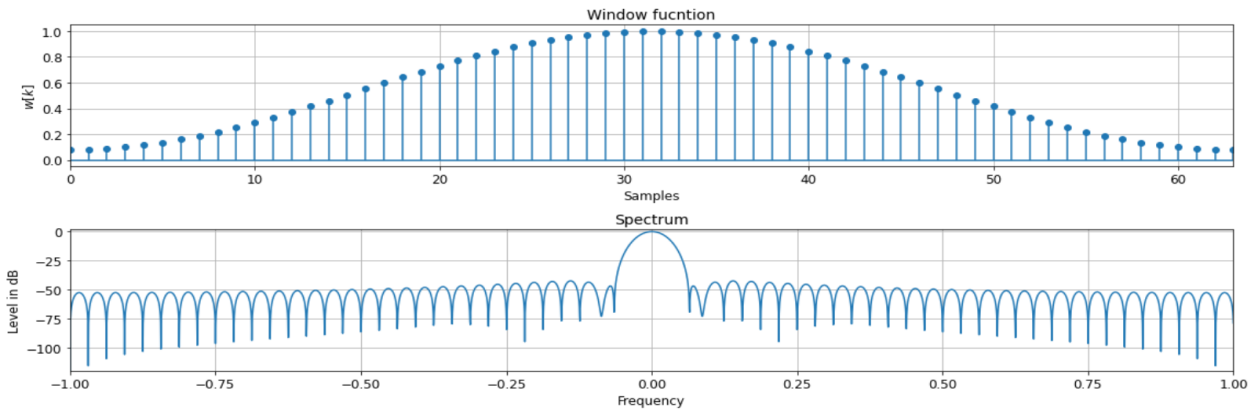


Рисунок 5 - Окно Хэмминга.

Окно Хэмминга выбрано как оптимальный баланс между частотным разрешением и подавлением спектральной утечки.  
 Его применение позволяет снизить влияние боковых лепестков примерно до −42 дБ, уменьшить паразитные пики, повысить точность и устойчивость определения основной частоты, улучшить работу метода HPS без значительного увеличения вычислительных затрат.

**1.3.3 HPS Harmonic Product Spectrum**

Если входной сигнал представляет собой музыкальную ноту, то его спектр должен состоять из ряда пиков, соответствующих основной частоте с гармоническими составляющими, кратными основной частоте. Следовательно, если спектр сжимается несколько раз (выполняется понижающая дискретизация) и сравнивается с исходным спектром, то самые сильные гармонические пики выстраиваются в ряд. Первый пик в исходном спектре совпадает со вторым пиком в спектре, сжатом в два раза, который, в свою очередь, совпадает с третьим пиком в спектре, сжатом в три раза. Следовательно, при перемножении различных спектров в результате образуется четкий пик на основной частоте.

**Описание метода:**

Сначала входной сигнал делится на сегменты, применяя окно Ханнинга, где размер окна и шаг задаются в качестве входных параметров. Для каждого окна используется быстрое преобразование Фурье. После преобразования входных данных в частотную область применяется метод гармонического спектра к каждому окну.

Метод HPS состоит из двух этапов: понижающей дискретизации и умножения. Для понижающей дискретизации спектр дважды сжимается в каждом окне с помощью повторной дискретизации: в первый раз исходный спектр сжимается в два раза, а во второй - в три раза. После этого перемножается три спектра и находится частота, соответствующая пику (максимальному значению). Эта частота представляет собой основную частоту конкретного окна.

**Ограничения метода HPS:**

К преимуществам этого метода можно отнести его вычислительную экономичность, достаточную устойчивость к аддитивному и мультипликативному шуму, а также возможность настройки под различные входные данные. Например, можно изменить количество используемых сжатых спектров и заменить спектральное умножение на спектральное сложение. Однако при оценке низких частот точность снижается. Это связано с особенностями спектрального представления и процедурой сжатия спектра, из-за которых уменьшается частотное разрешение и ухудшается выделение фундаментальной частоты при низких значениях F0.

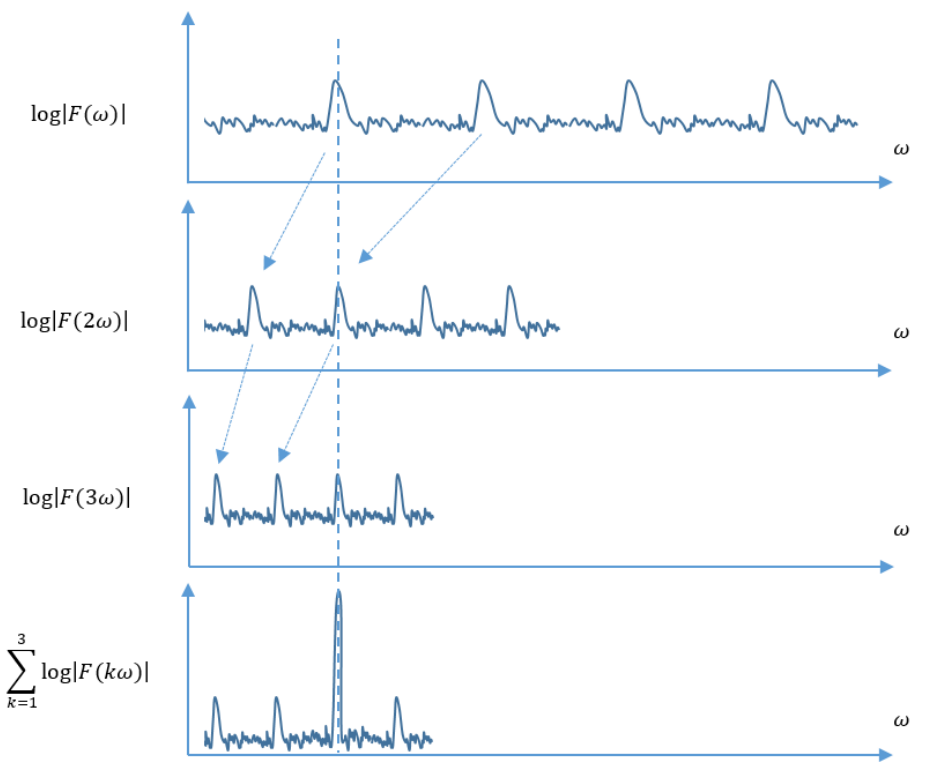
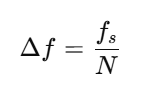
****

Рисунок 6 - Визуализация принципа работы HPS.

Недостатком метода HPS является то, что его разрешающая способность зависит от длины БПФ, используемого для вычисления спектра. Если используется короткое и быстрое БПФ, то количество дискретных частот, которые можно учитывать, будет ограничено. Чтобы повысить разрешающую способность на выходе, нужно использовать более длинное БПФ, что требует больше вычислений.

**1.3.4 Параболическая интерполяция**

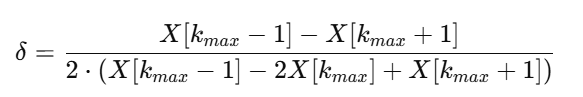
При использовании быстрого преобразования Фурье определение частоты обычно выполняется по положению максимального пика спектра. Однако частота реального сигнала редко совпадает точно с центром дискретного частотного отсчета. В результате максимум спектра распределяется между несколькими соседними отсчетами, что приводит к частотной погрешности порядка шага частотной развертки:



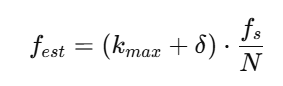
где  
 fs – частота дискретизации,  
 N – длина окна БПФ.

Чтобы повысить точность, используется параболическая интерполяция максимума спектра. В её основе лежит предположение, что вблизи истинного максимума огибающая спектральной линии может быть приближена параболой. Если X[k] – амплитуды спектральных отсчётов, а максимум БПФ найден в отсчете kmax​, то для уточнения положения пика используется информация о соседних отсчетах X[kmax−1], X[kmax] и X[kmax+1].

Смещение максимума относительно центрального бина вычисляется по формуле:



Тогда уточнённая частота определяется как:



Таким образом, даже при дискретной частотной сетке удаётся получить лучшее разрешение. На практике это позволяет уменьшить ошибку измерения частоты с нескольких герц до десятых долей герца, что важно для гитарного тюнера, где требуется высокая точность определения высоты ноты.

**1.3.5 Медианная фильтрация**

Вычисленная частота со временем может незначительно «прыгать». Это вызвано шумами пьезодатчика, нестабильностью амплитуды сигнала, неидеальной дискретизацией, процессами затухания струны, негармоническими составляющими.

Если такие колебания отображать напрямую, показания тюнера становятся нестабильными. Для устранения этого эффекта используется медианная фильтрация последовательности измеренных частот во времени.

Принцип работы медианного фильтра:

1. Формируется скользящее окно из нескольких последних измерений, 7 значений частоты.
2. После накопления окна значения копируются во временный массив и сортируются простым квадратичным алгоритмом (по принципу сортировки выбором). Благодаря сортировке элементы располагаются по возрастанию, после чего медианное значение выбирается как центральный элемент массива.
3. В качестве результата выбирается медиана — среднее по порядку значение.

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

1. **Разработка аппаратной части**

Структурная схема устройства:

Тюнер: пьезоэлектрическая мембрана -> предусилитель -> ESP32-S3 -> OLED-дисплей.

**1.1 Моделирование схемы**

Первым этапом было построение схемы предусилителя в среде Multisim (рис. 8).

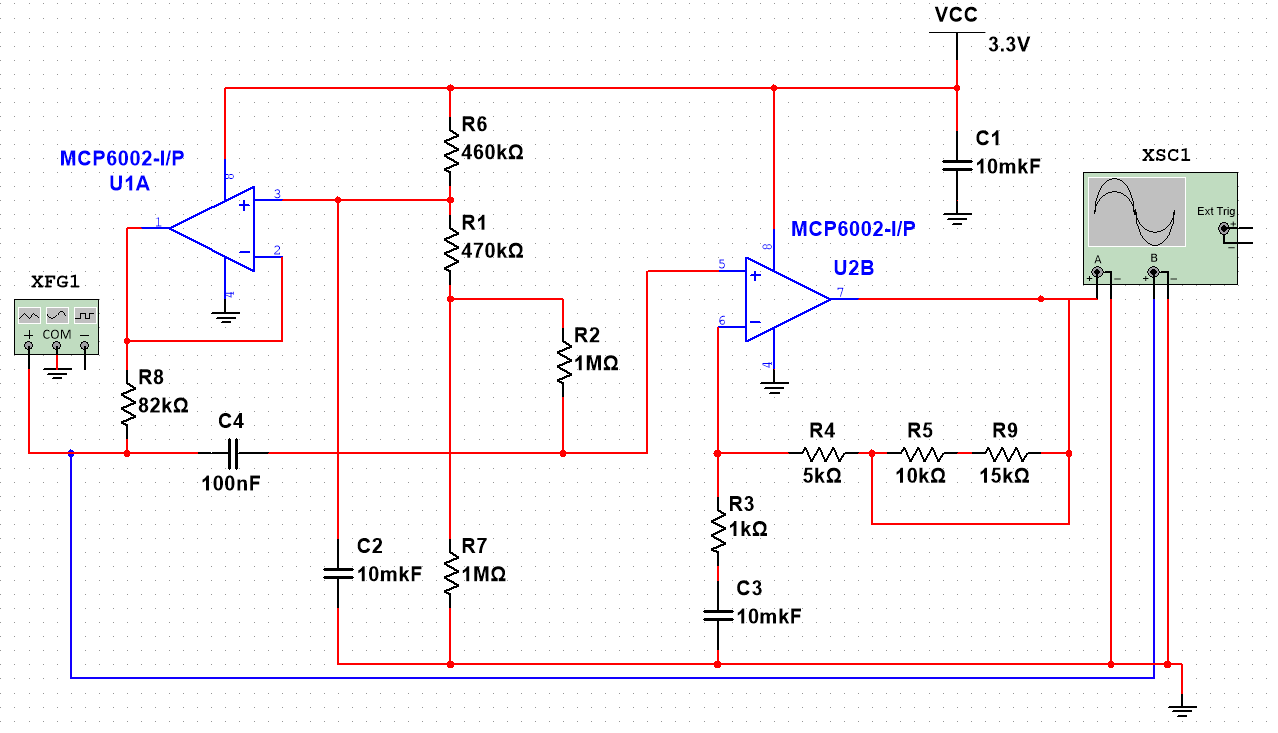


Рисунок 7 - Схема предусилителя

**Описание схемы предусилителя**

Сигнал, поступающий на вход, проходит через С4 на неинвертирующий вход операционного усилителя.

Постоянное напряжение на этом выходе определяется [делителем](https://www.joyta.ru/7328-delitel-napryazheniya-na-rezistorax-raschet-onlajn/) R6, R1, R7 и составляет чуть больше половины напряжения питания Vref ≈ 1,65 В. Входное сопротивление самого усилителя для переменных низкочастотных сигналов равно сумме сопротивлений R6 и [параллельного соединения резисторов](https://www.joyta.ru/7362-parallelnoe-soedinenie-rezistorov/) R1, R7 и составляет примерно 1,3 МОм.

Фильтрация питания снижает шум и подверженность самовозбуждению. За счет применения в делителе резисторов с большим сопротивлением и конденсатора С2 емкостью 10 мкФ эффективно фильтруется напряжение.

Первый каскад ОУ (U1A) выполнен как повторитель напряжения с коэффициентом усиления Ku1 = 1. Входная цепь включает разделительный конденсатор C4 и резистор R8, формирующие фильтр высоких частот для блокировки постоянной составляющей сигнала.

Второй каскад ОУ (U2B) реализован как неинвертирующий усилитель с коэффициентом усиления, изменяющимся в зависимости от сопротивления. Коэффициент усиления определяется отношением сопротивлений R4, R5, R9 к R3. Для возможности регулировки усиления в схеме использованы три резистора (R7-R9), которые в физической схеме будут заменены на переменный резистор.

Для снятия сигнала в Multisim смоделирована схема с ОУ MCP6002, который будет использоваться в конечной версии схемы. В этом случае выходной сигнал должен колебаться в диапазоне от 0 до 3.3 вольт, что соответствует разрешенному напряжению для esp32s3.

Частота входной синусоиды в тестовых запусках Multisim: 166 Гц. Амплитуда: 410 мВ.

**Графики сигналов с разным усилением**

Рассмотрим осциллограмму случая, когда коэффициент усиления в теории равен 5/1+1 = 6. На рисунке 9 видно, что средняя точка синусоиды на выходе (красная линия) поднимается до 1.65, сигнал колеблется в полосе от 0 до 3.3 вольт. Рассчитанный коэффициент усиления равен: 2375/449+1 = 6.3.

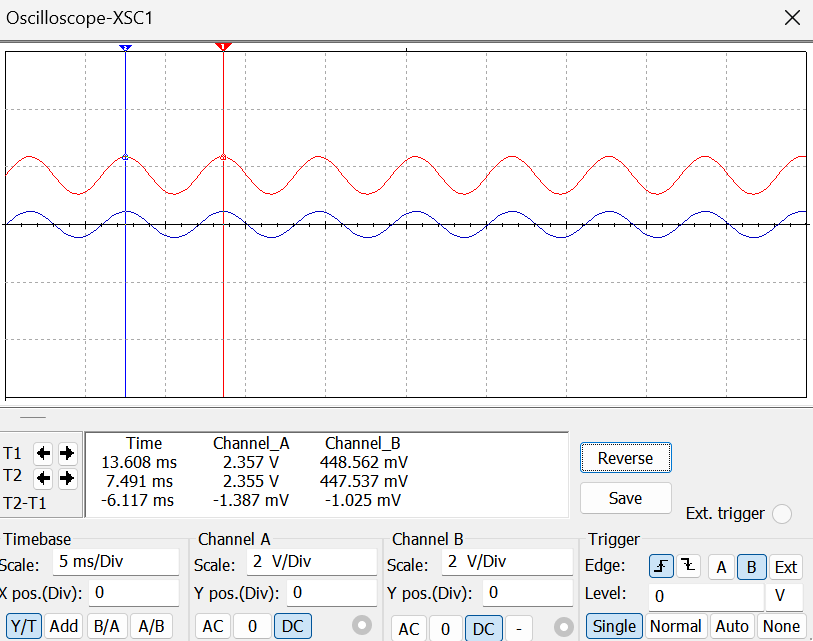


Рисунок 8 - Осциллограмма сигнала при усилении в 6 раз.

Во втором случае теоретический коэффициент усиления составляет 30/1+1=31. При таком усилении динамические свойства сигнала ухудшились, что сделано намеренно - это позволяет предотвратить подачу на микроконтроллер напряжений, превышающих допустимые уровни.

Нижнее ограничение также является корректным, поскольку микроконтроллер не работает с отрицательными уровнями сигнала. В результате сигнал «обрезается» при достижении значений 0 В и 3.3 В (рис. 9). Такое поведение является необходимым и обеспечивает безопасный диапазон входного сигнала для микроконтроллера.



Рисунок 9 - Осциллограмма сигнала при усилении в 31 раз.

**1.2. Элементы сборки**

Компоненты для сборки предусилителя:

1. CBC2065BAL, d=20мм, 6.5кГц, L=90мм, Пьезоэлектрическая диафрагма (1)

2. MCP6002-I/P, Операционный усилитель 1МГц, 2-канала, Rail-to-Rail [DIP-8.] (1)

3. CF-100 (С1-4) 1Вт, 470 кОм, Резистор углеродистый (2)

4. ECAP (К50-35 мини), 10мкФ, Конденсатор электролитический алюминиевый (2)

5. CF-100 (С1-4) 1Вт, 82 кОм, Резистор углеродистый (1)

6. RCER71E104K0A2H03B, Конденсатор: керамический; 100нФ (1)

7. CF-100 (С1-4) 1Вт, 1 МОм, Резистор углеродистый (2)

8. СП3-4АМ, 0.25Вт, 100 Ом, Резистор переменный (1)

9. Беспаечная макетная плата 750 точек 98мм 84 мм (1)

10. OLED дисплей 0.96 128x64, I2C IIC(1)

11. Плата макетная двусторонняя 70 х 90 мм (Шаг:2.54) (1)

12. Модуль плата ESP32-S3-DevKitC-1 (1)

**1.3 Тестирование схемы**

При снятии сигнала на макетной плате NI ELVIS в схеме использовался операционный усилитель TL072 и подавались 5 вольт. В качестве тестовой ситуации вместо сигнала, поступающего с пьезоэлемента, используются генераторы сигналов с нужной амплитудой и частотой.

Рассмотрим осциллограммы. При коэффициенте усиления 0 наблюдаем синусоиду, которая просто сдвинута к средней точке и повторяет по форме входной сигнал (рис. 10).

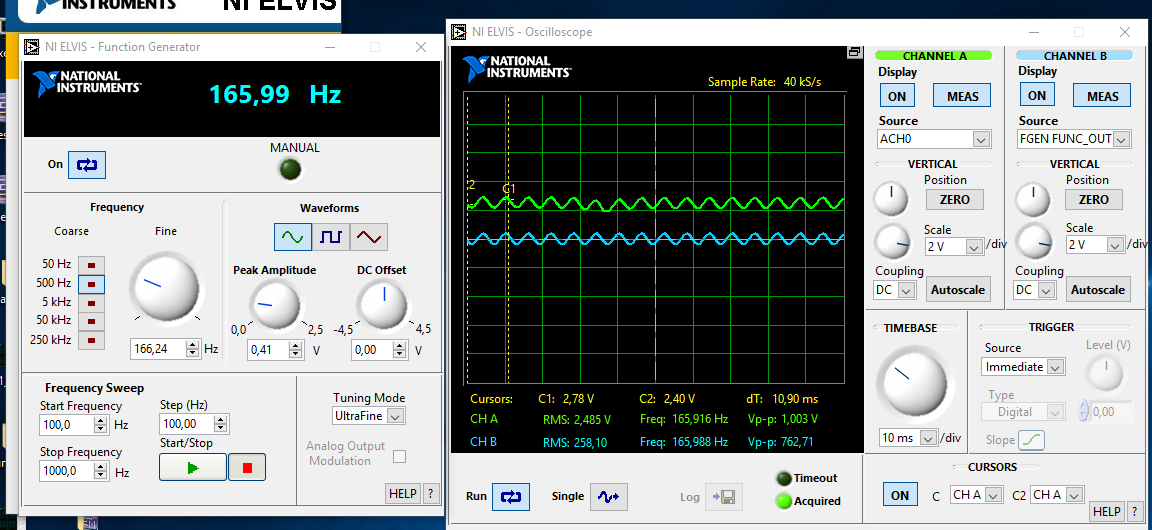


Рисунок 10 - Осциллограмма сигнала с усилением в 0 раз.

Изменяем сопротивление с помощью потенциометра и видим, что синусоида на выходе растягивается и размещается в диапазоне от 0 до 5 вольт (рис. 11).

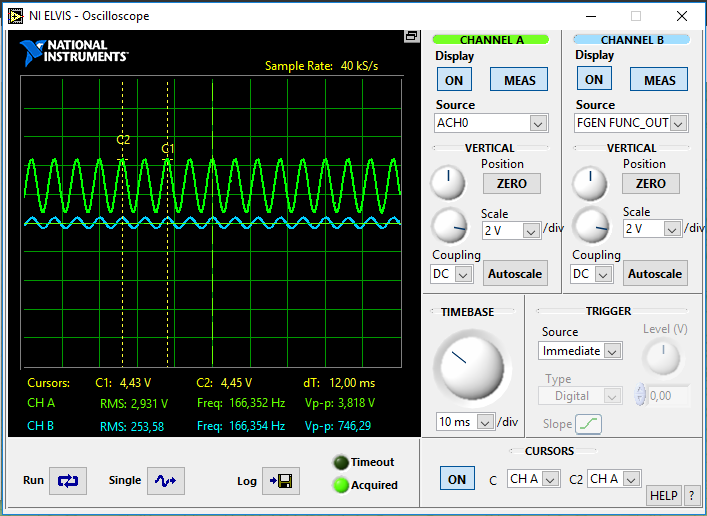


Рисунок 11 - Осциллограмма сигнала со средним коэфф-ом усиления

При большом значении сопротивления на переменном резисторе наблюдаем ту же ситуацию, что и при моделировании в Multisim. Сигнал “обрубается” в нуле и пяти вольтах, что является корректным показателем и сигнализирует о том, что установку можно соединять с микроконтроллером после подбора верного операционного усилителя.

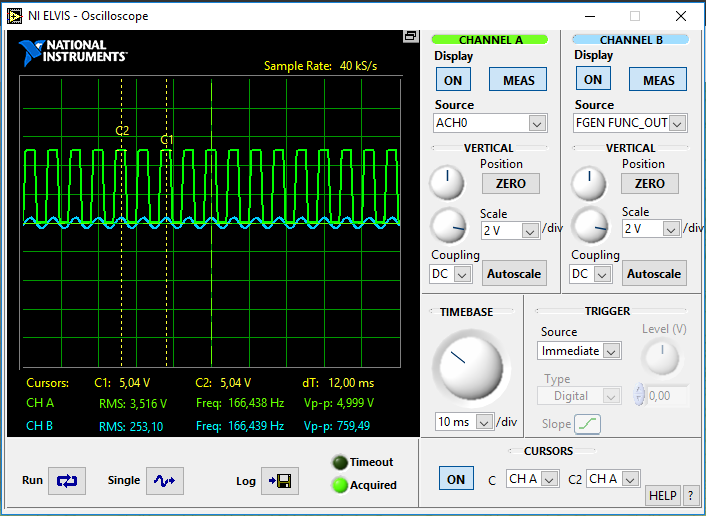


Рисунок 12 - Осциллограмма сигнала с максимальным коэфф-ом усиления.

1. **Разработка программного обеспечения**

**ПРОГРАММА ОБРАБОТКИ CSV ФАЙЛОВ С ПОМОЩЬЮ PYTHON**

Данная программа выполняет анализ аналого-цифрового преобразователя с помощью обработки сигнала.

Вычитание среднего значения из всех отсчетов сигнала.

def fourier\_transform(x, fs, N, use\_hps):

x\_clean = x - np.mean(x)

Выполнение БПФ.

X = np.fft.fft(x\_clean)

Создается массив частот для спектра. 1/fs - период дискретизации. в результате массив частот от -fs/2 до fs/2.

freq = np.fft.fftfreq(N, 1 / fs)

Вычисляем амплитудный спектр. Убираем фазу с помощью abs и нормируем на количество отсчетов N.

amplitude\_spectrum = abs(X) / N

Оставляем положительную половину частот.

half = N // 2

freq\_half = freq[:half]

ampl\_half = amplitude\_spectrum[:half]

Игнорируем нулевую частоту (постоянную составляющую) и компенсируем сдвиг индекса с помощью +1. Берем индекс максимальной гармоники с помощью argmax.

  main\_idx = np.argmax(ampl\_half[1:]) + 1

    main\_freq = freq\_half[main\_idx]

    main\_ampl = ampl\_half[main\_idx]

Нахождение основной гармоники с использованием окна.

Создаем окно Ханнинга длиной N.

window = np.hamming(N)

x\_windowed = x\_clean \* window

БПФ оконного сигнала, подготовка амплитудного спектра

X\_window = np.fft.fft(x\_windowed)

amplitude\_spectrum\_window = abs(X\_window) / N

ampl\_half\_window = amplitude\_spectrum\_window[:half]

Аналогичный поиск индекса основной гармоники

main\_idx\_win = np.argmax(ampl\_half\_window[1:]) + 1

main\_freq\_win = freq\_half[main\_idx\_win]

main\_ampl\_win = ampl\_half\_window[main\_idx\_win]

Переименуем главную гармонику из базового анализа. Максимальный номер гармоники = 3.

hps\_freq = main\_freq

if use\_hps:

max\_harmonic = 3

HPS = ampl\_half.copy()

Прореживаем спектр: берем каждый второй, потом каждый третий элемент. Делаем поэлементное умножение.

for k in range(2, max\_harmonic + 1):

downsampled = ampl\_half[::k]

HPS[:len(downsampled)] \*= downsampled

Ищем индекс основной частоты.

fund\_idx = np.argmax(HPS)

hps\_freq = freq\_half[fund\_idx]

Расчет теоретической дисперсии оценки частоты. SNR - отношение сигнал/шум. Используется формула Крамера-Рао для оценки частоты. Вычисляем числитель и знаменатель. В результате получаем дисперсию оценки частоты (Гц^2).

*# f\_KR = (6\*fs^2) / ((2\*pi)^2 \* SNR \* N^3)*

num = 6 \* fs \*\* 2

denom = (2 \* np.pi) \*\* 2 \* SNR \* N \*\* 3

Расчет SNR. Определяем длину сигнала, берем количество отсчетов до момента разделения. Получаем индекс массива.

N = len(x)

noise\_idx = int(noise\_end\_time \* fs)

Выделяем шумовую часть сигнала и сигнальную часть сигнала.

noise\_section = x[:noise\_idx]

signal\_section = x[noise\_idx:]

Считаем мат. ожидание и СКО шума.

noise\_mean = np.mean(noise\_section)

noise\_std = np.std(noise\_section)

Нормировка сигнала, без постоянной составляющей.

signal\_norm = signal\_section - noise\_mean

СКО нормированного сигнала

signal\_std = np.std(signal\_norm)

Мощность шума и мощность сигнала. СКО в квадрате.

noise\_power = noise\_std \*\* 2

signal\_power = signal\_std \*\* 2

Линейная шкала и шкала в децибелах.

snr\_linear = signal\_power / noise\_power

snr\_db = 10 \* np.log10(snr\_linear)

**2. Анализ результатов тестирования**

**РЕЗУЛЬТАТЫ ОБРАБОТКИ ЗАПИСАННЫХ CSV ФАЙЛОВ В PYTHON**

В ходе разработки алгоритма программа тестировалась на записанных файлах с помощью собранной установки предусилителя. Данные с АЦП записаны в csv файл и проанализированы по разработанному алгоритму. В ходе записи каждая струна была записана отдельно по одному удару.

**6 струна**

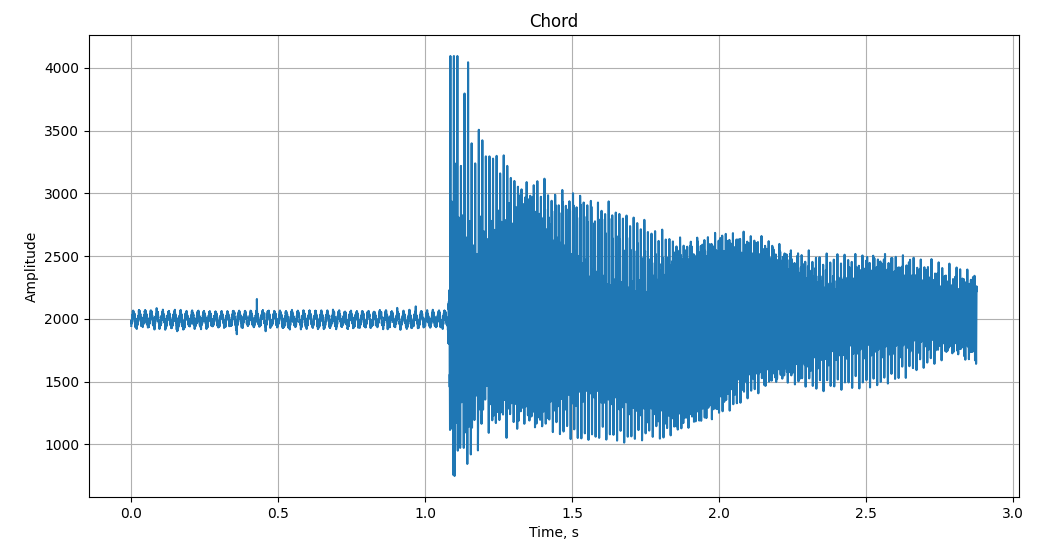


Рисунок 13 - Сигнал во временной области, 6 струна.

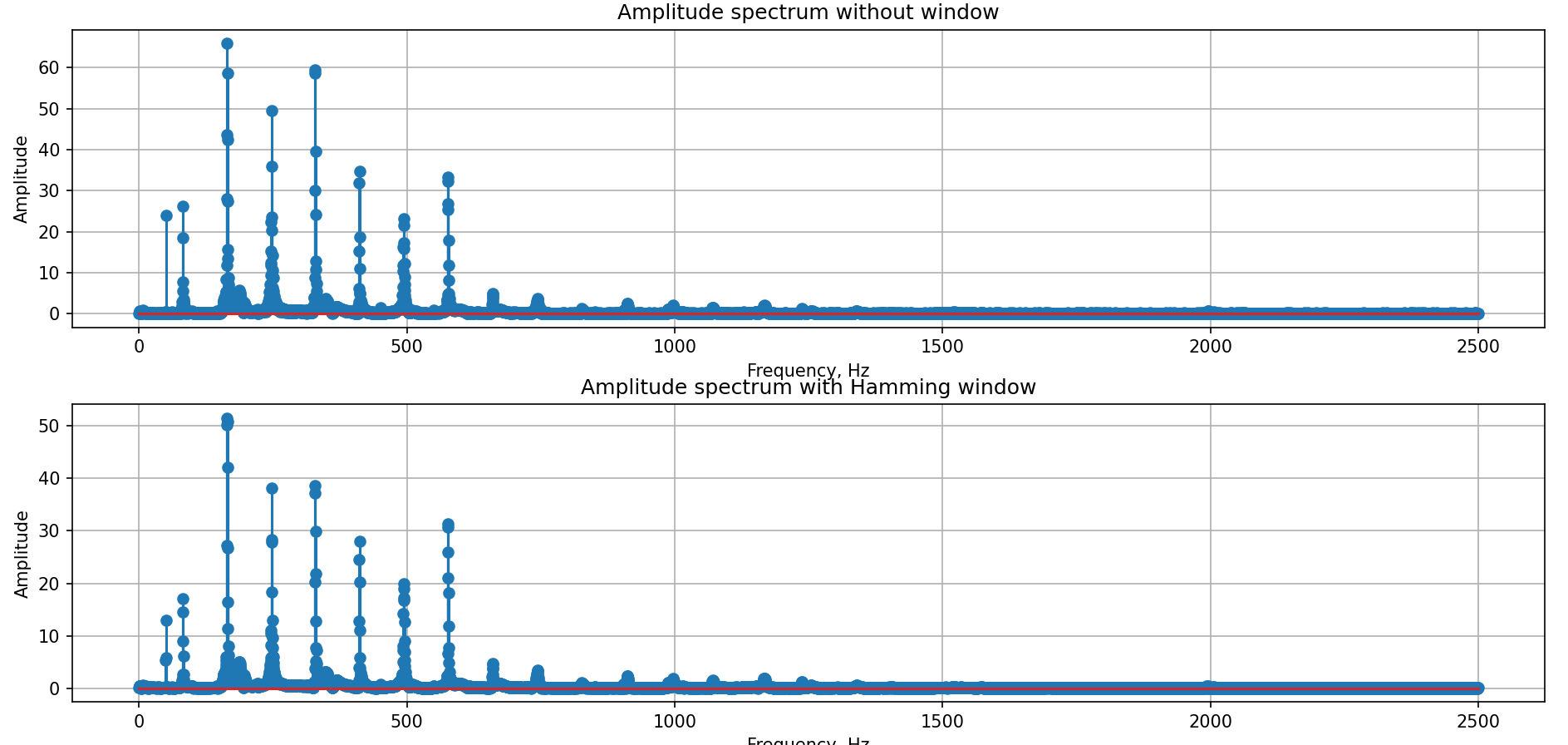


Рисунок 14 - Сигнал в частотной области без применения и с применением окна, 6 струна.

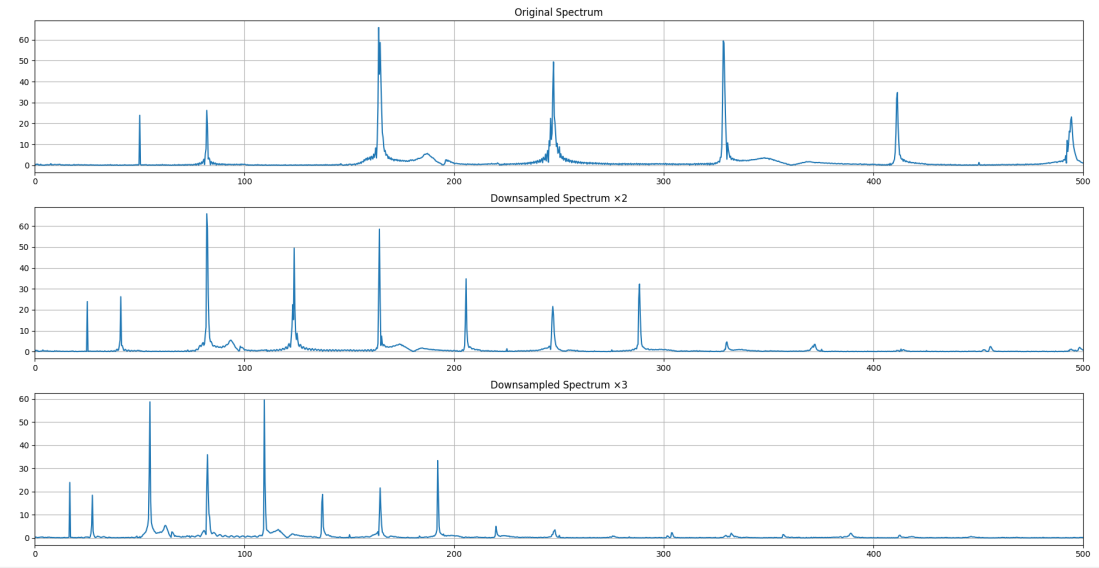


Рисунок 15 - Применение HPS, 6 струна.

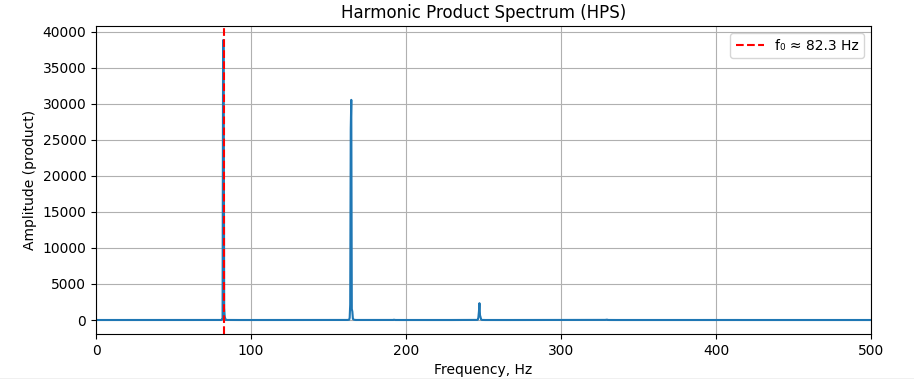


Рисунок 16 - Полученный результат после применения HPS, 6 струна.

Основная гармоника: частота = 164.00 Гц, амплитуда = 65.88

Основная гармоника with Hanning: частота = 164.35 Гц, амплитуда = 52.26

Основная гармоника (HPS): частота = 82.35 Гц

**5 струна**

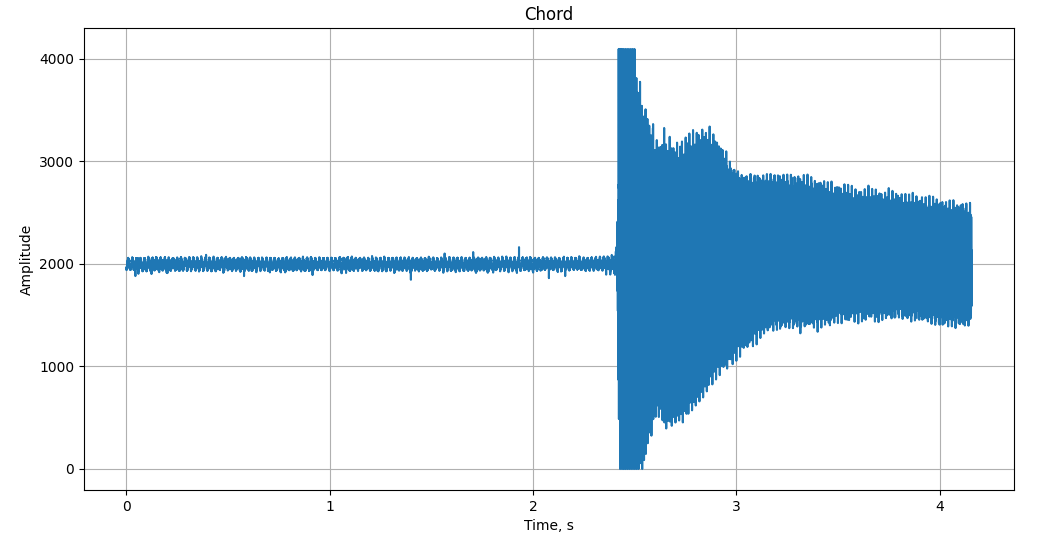


Рисунок 17 - Сигнал во временной области, 5 струна.

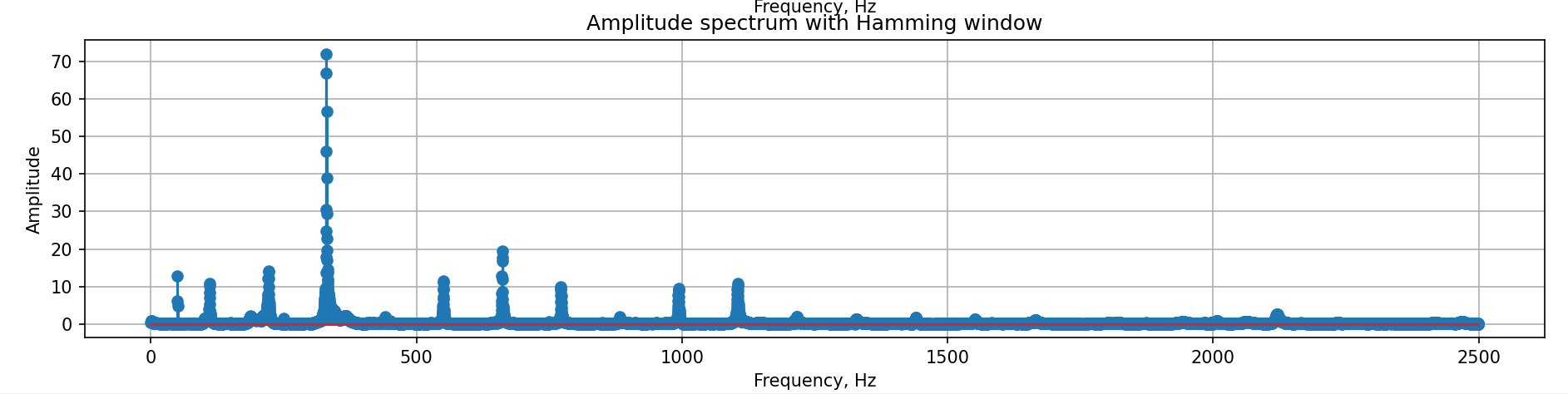


Рисунок 18 - Сигнал в частотной области с применением окна, 5 струна.

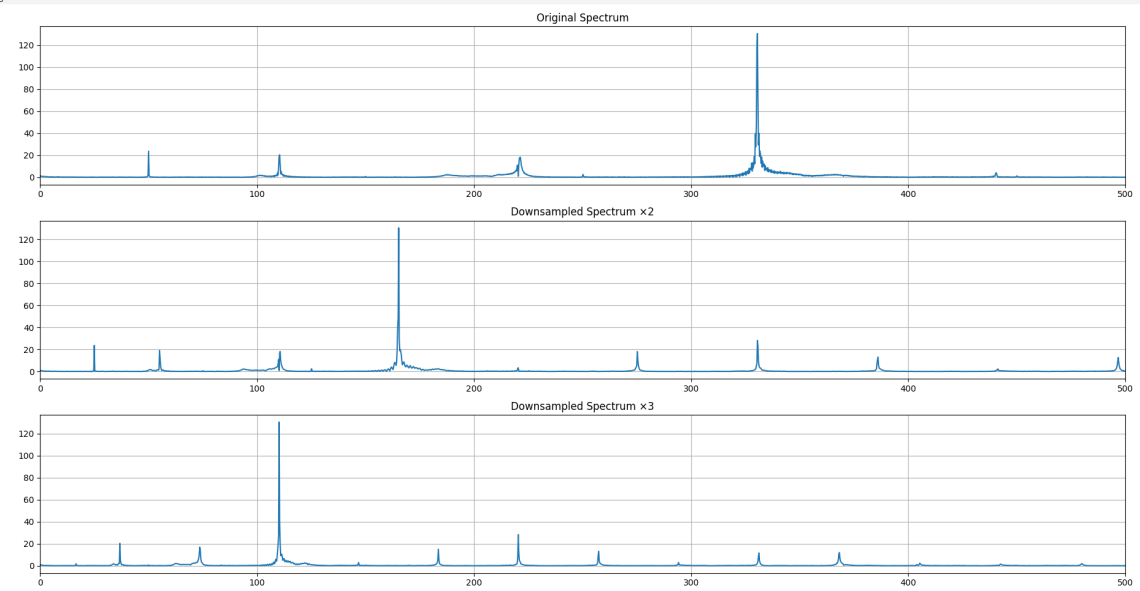


Рисунок 19 - Применение HPS, 5 струна.

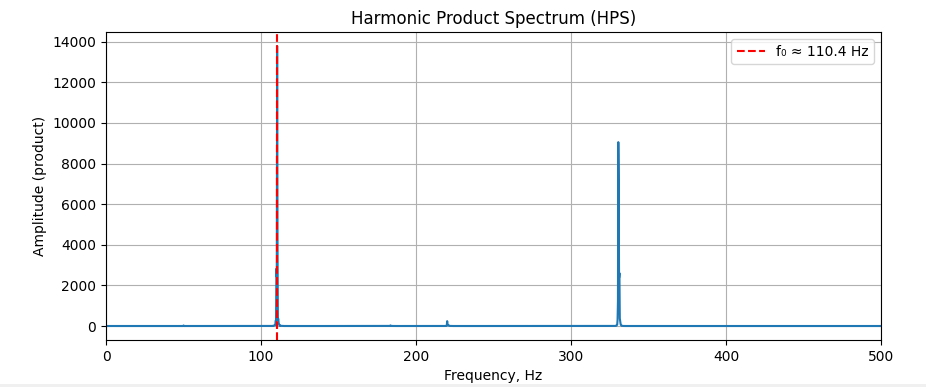


Рисунок 20 - Полученный результат после применения HPS, 5 струна.

Основная гармоника: частота = 330.53 Гц, амплитуда = 130.57

Основная гармоника with Hanning: частота = 330.53 Гц, амплитуда = 67.11

Основная гармоника (HPS): частота = 110.42 Гц

**4 струна**

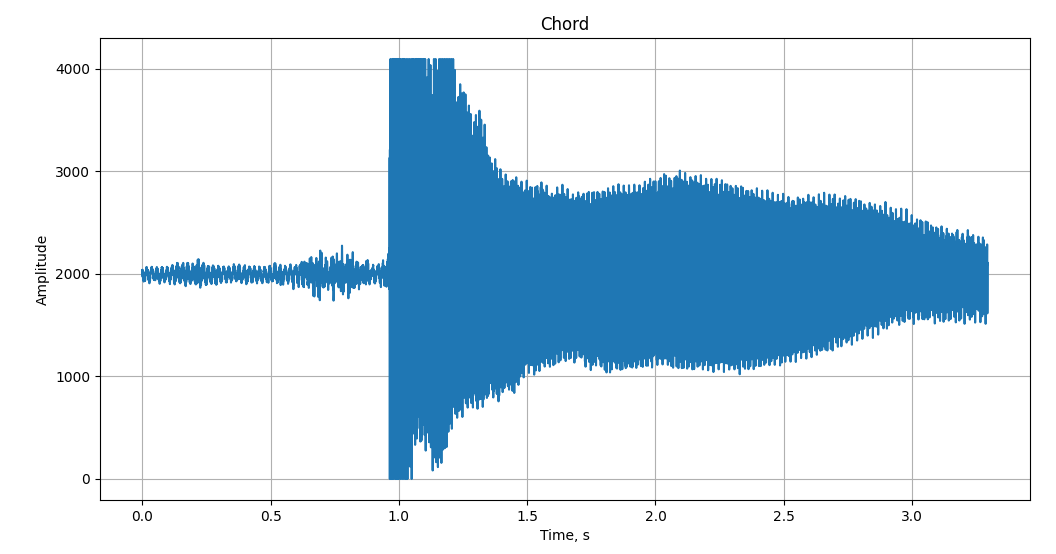


Рисунок 21 - Сигнал во временной области, 4 струна.

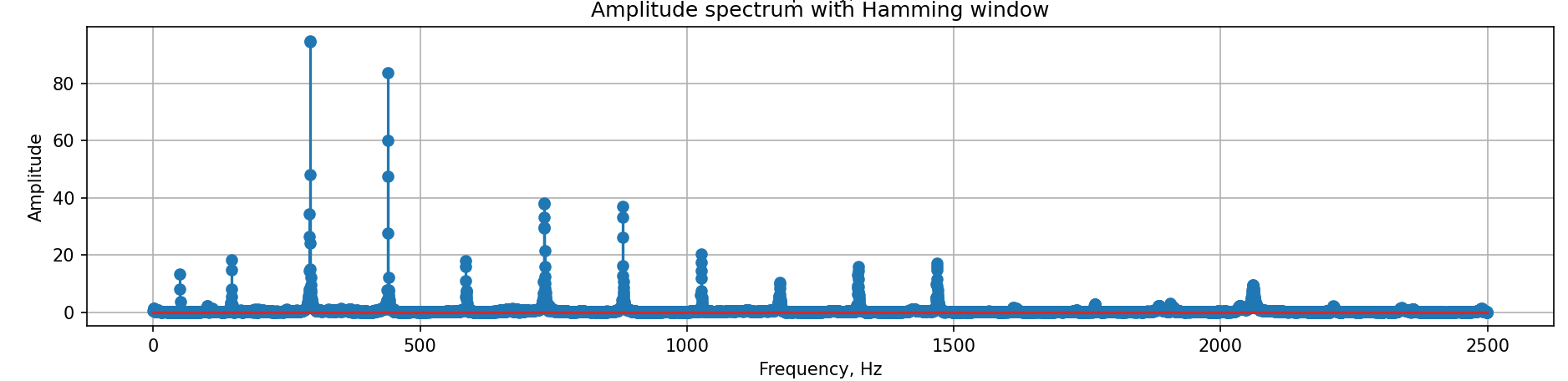


Рисунок 22 - Сигнал в частотной области с применением окна, 4 струна.

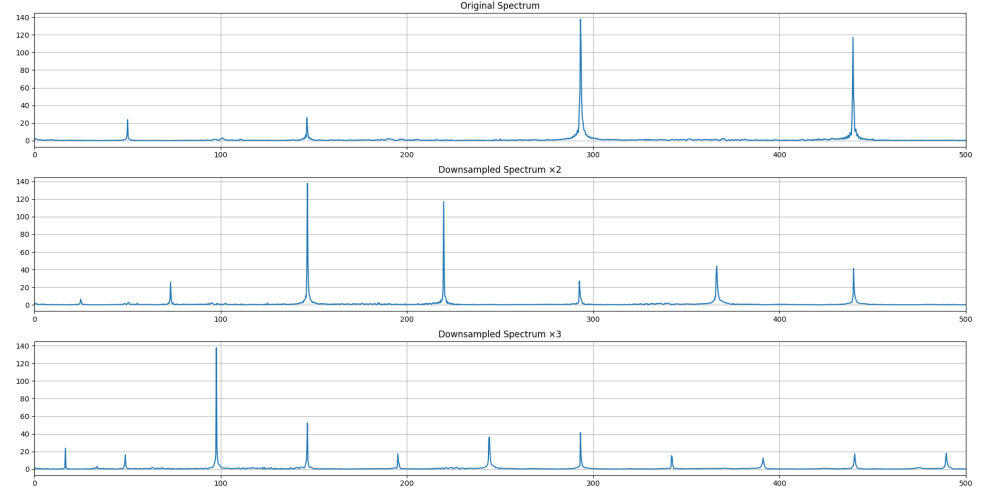


Рисунок 23 - Применение HPS, 4 струна.

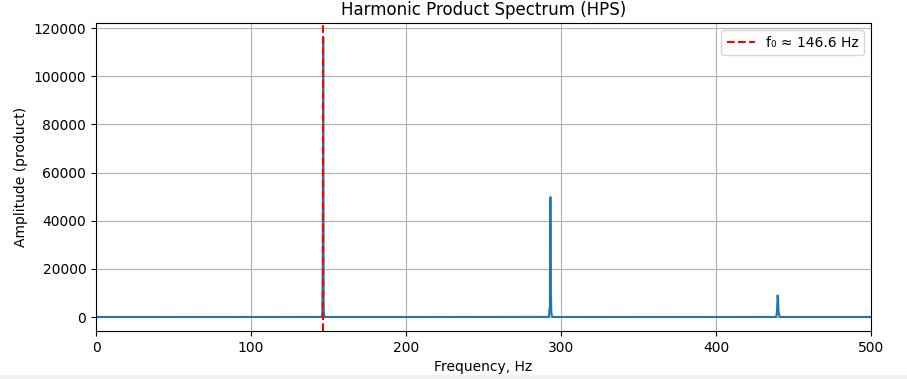


Рисунок 24 - Полученный результат после применения HPS, 4 струна.

Основная гармоника: частота = 293.15 Гц, амплитуда = 137.83

Основная гармоника with Hanning: частота = 293.46 Гц, амплитуда = 93.95

Основная гармоника (HPS): частота = 146.58 Гц

**3 струна**

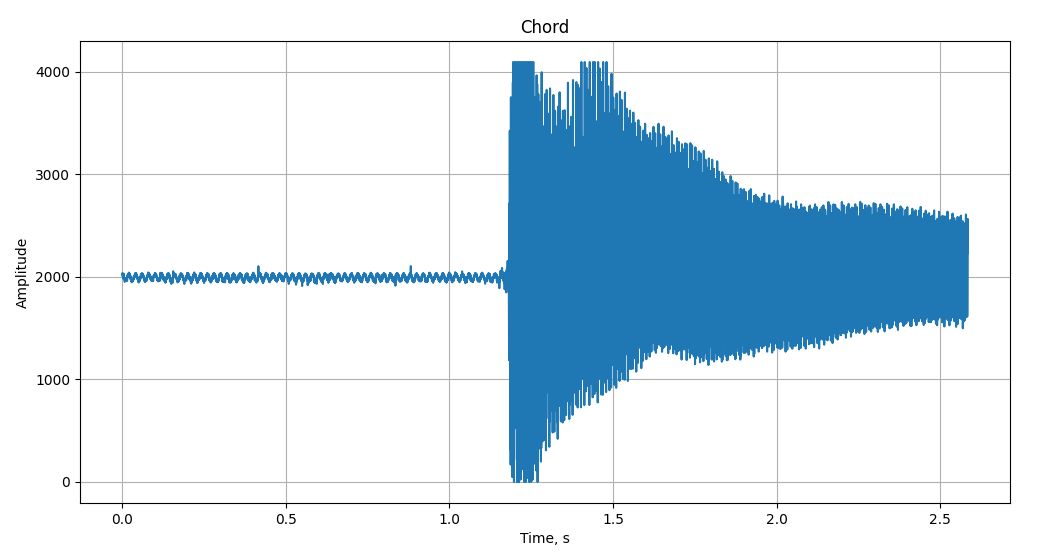


Рисунок 25 - Сигнал во временной области, 3 струна.

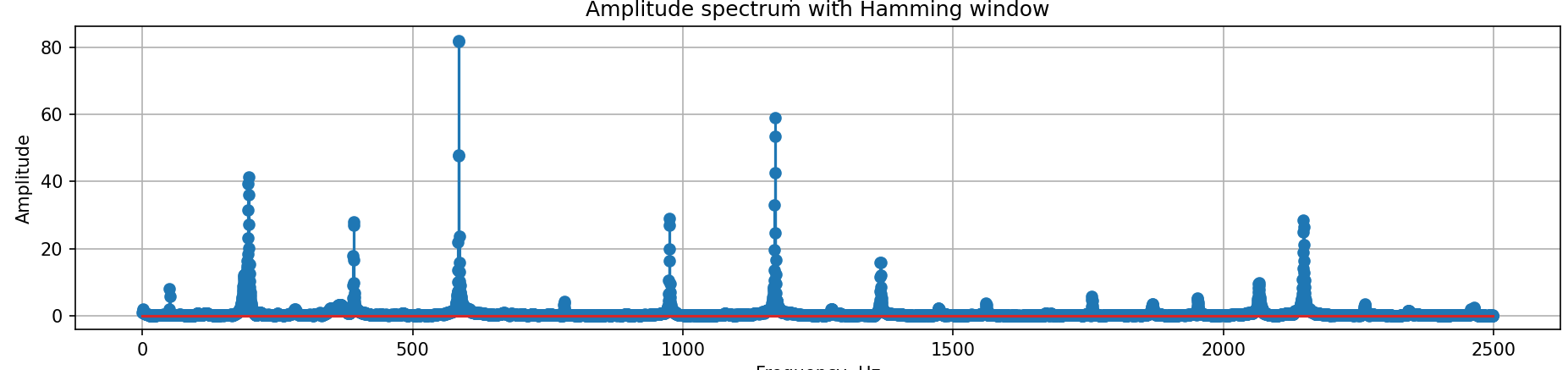


Рисунок 26 - Сигнал в частотной области с применением окна, 3 струна.

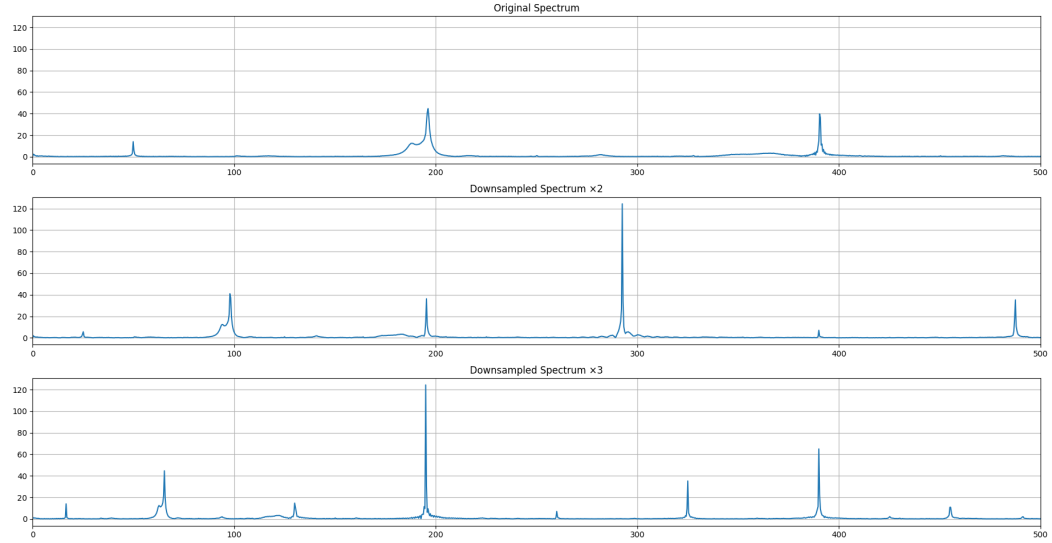


Рисунок 27 - Применение HPS, 3 струна.

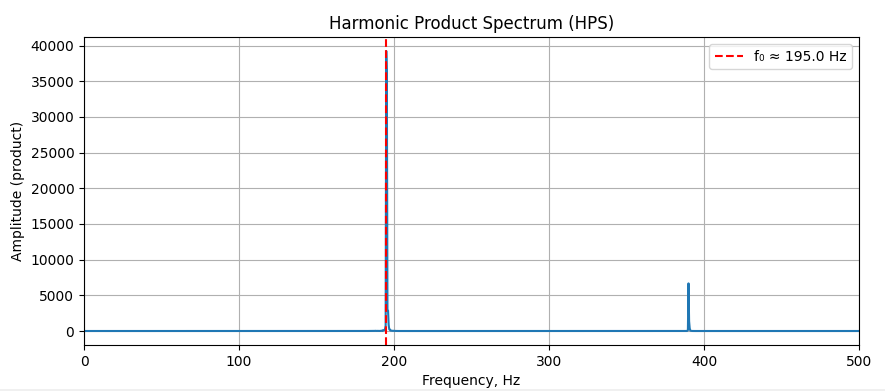


Рисунок 28 - Полученный результат после применения HPS, 3 струна.

Основная гармоника: частота = 585.00 Гц, амплитуда = 124.32

Основная гармоника with Hanning: частота = 585.00 Гц, амплитуда = 78.79

Основная гармоника (HPS): частота = 195.00 Гц

**2 струна**

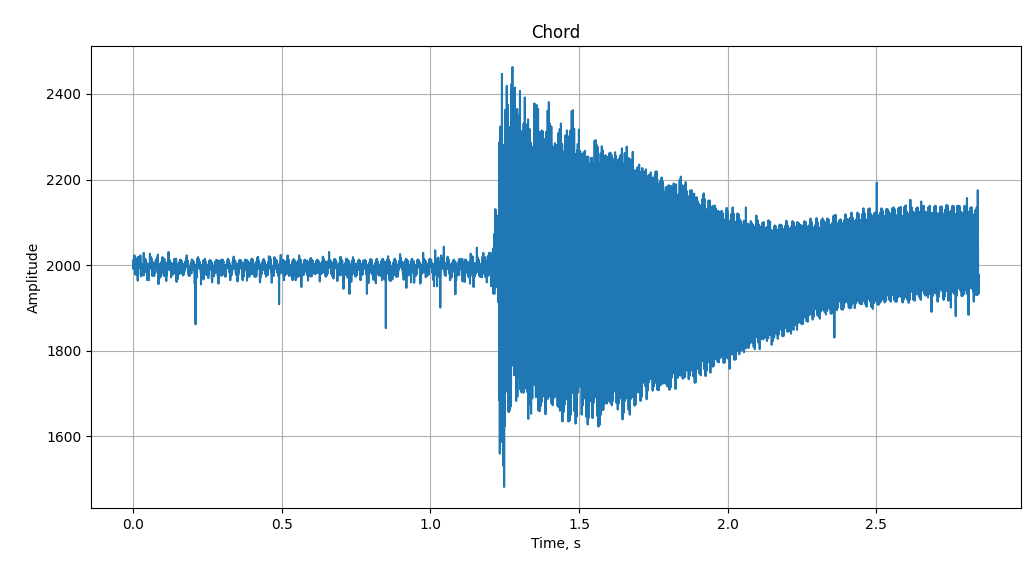


Рисунок 29 - Сигнал во временной области, 2 струна.

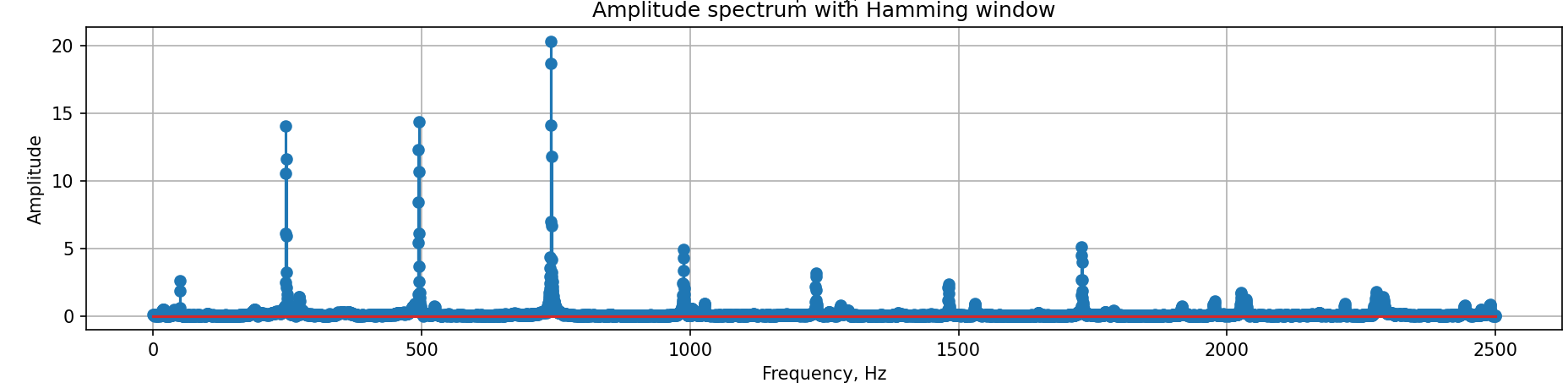


Рисунок 30 - Сигнал в частотной области с применением окна, 2 струна.

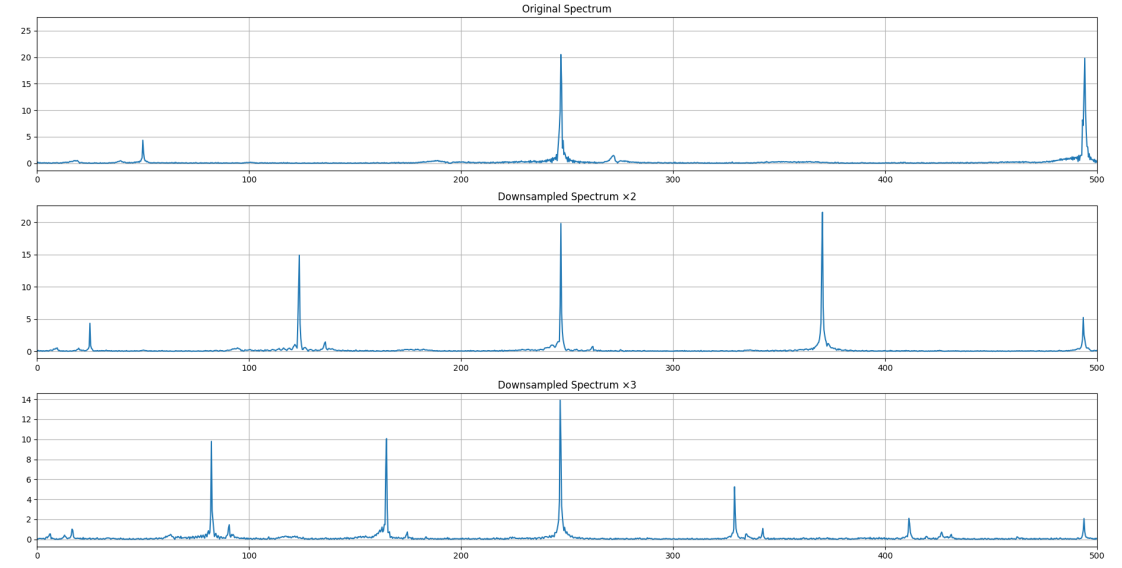


Рисунок 31 - Применение HPS, 2 струна.

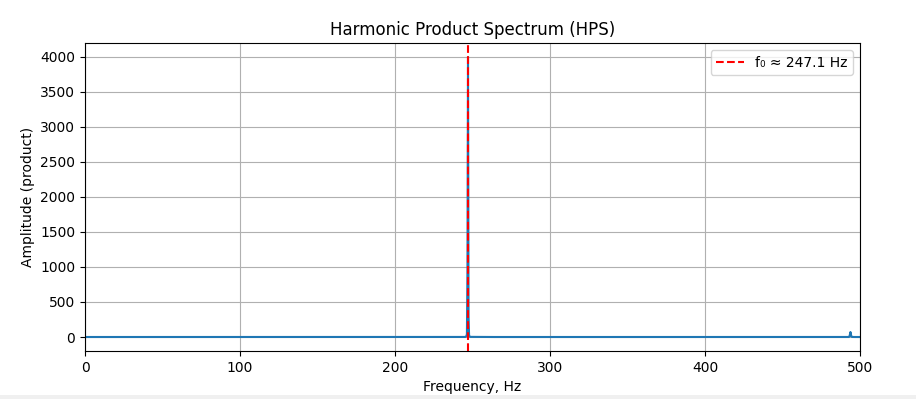


Рисунок 32 - Полученный результат после применения HPS, 2 струна.

Основная гармоника: частота = 740.60 Гц, амплитуда = 26.21

Основная гармоника with Hanning: частота = 740.60 Гц, амплитуда = 19.82

Основная гармоника (HPS): частота = 247.10 Гц

**1 струна**

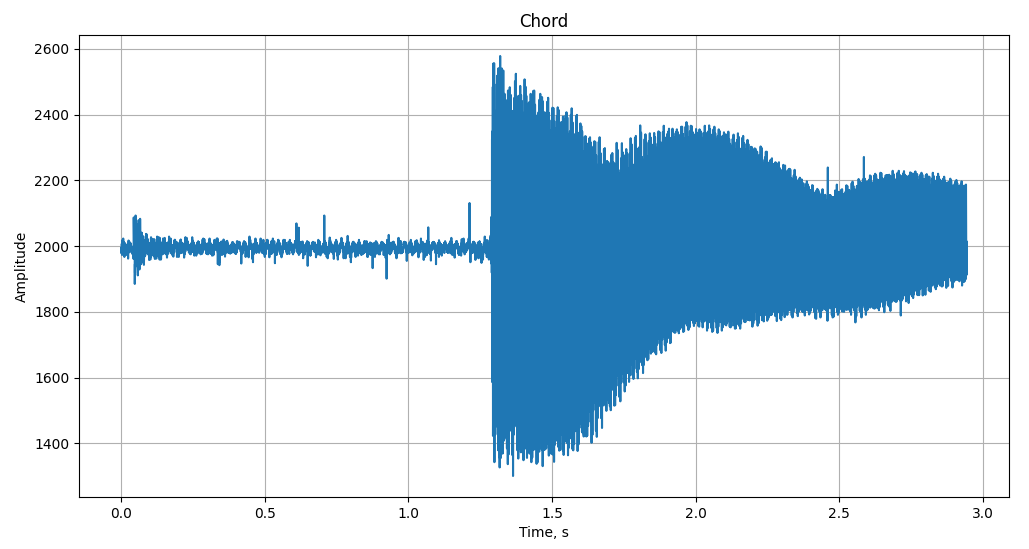


Рисунок 33 - Сигнал во временной области, 1 струна.

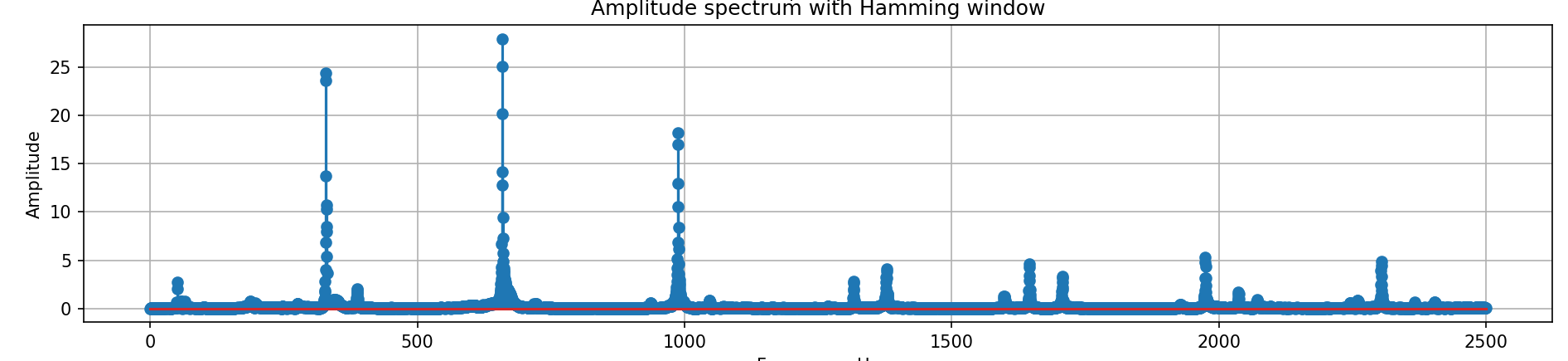


Рисунок 34 - Сигнал в частотной области с применением окна, 1 струна.

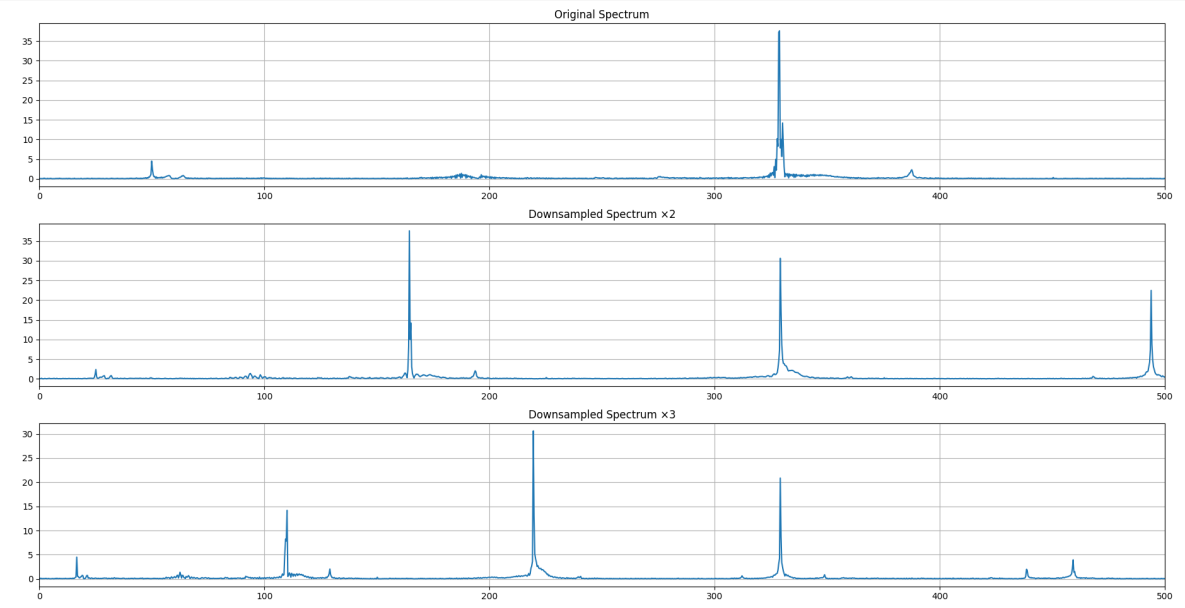


Рисунок 35 - Применение HPS, 1 струна.

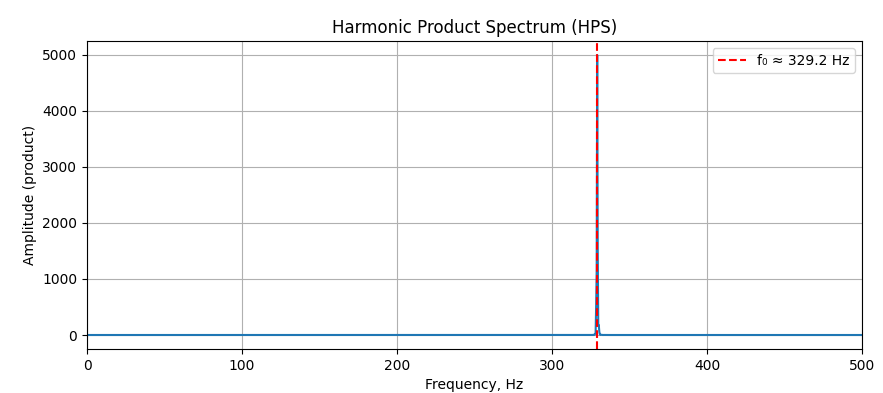


Рисунок 36 - Полученный результат после применения HPS, 1 струна.

Основная гармоника: частота = 328.83 Гц, амплитуда = 37.59

Основная гармоника with Hanning: частота = c, амплитуда = 27.27

Основная гармоника (HPS): частота = 329.17 Гц

**Сводная таблица результатов тестовых запусков алгоритма**

Таблица 3 - Результаты анализа csv файлов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Струна** | **Эталонная частота** | **Частота, найденная без применения окна** | **Частота, найденная с применением окна** | **Частота, найденная после применения HPS** | **Ошибка** |
| **6** | 82,41 | 164.00 | 164.35 | 82.35 | 0,14 |
| **5** | 110,00 | 330.53 | 330.53 | 110.42 | 0,42 |
| **4** | 146,83 | 293.15 | 293.46 | 146.58 | 0,25 |
| **3** | 196,00 | 585.00 Гц | 585.00 | 195.00 | 1,00 |
| **2** | 246,94 | 740.60 | 740.60 | 247.10 | 0,16 |
| **1** | 329,63 | 328.83 | 658.67 | 329.17 | 0,54 |

Примечание: ошибка может быть связана не только с работой алгоритма, но и со степенью настройки струны в момент записи и анализа.

**Таблица определения ошибки по методу Крамера-Рао**

Неравенство Крамера–Рао задаёт теоретический нижний предел дисперсии любой несмещенной оценки параметра:

Таблица 4 - Результаты анализа ошибки методом Крамера-Рао

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Струна** | **SNR (лин)** | **SNR (дБ)** | **f\_KR** | **sqrt(f\_KR) (Гц)** |
| 6 | 109.73 | 20.40 | 1.16e-08 | 0.000108 |
| 5 | 224.09 | 23.50 | 1.89e-09 | 0.000043 |
| 4 | 84.85 | 19.29 | 1.00e-08 | 0.000100 |
| 3 | 480.73 | 26.82 | 3.66e-09 | 0.000061 |
| 2 | 19.61 | 12.93 | 6.73e-08 | 0.000259 |
| 1 | 175.12 | 22.43 | 6.80e-09 | 0.000082 |

**Анализ результатов**

**1. Эффективность алгоритма HPS**

Метод гармонического произведения спектров (HPS) успешно решает проблему ошибочного определения гармоник за основную частоту.

Для струн 2, 3,4, 5 и 6 простой поиск пика спектра определял 2-ю или 3-ю гармонику как основную:

Струна 6: 164.00 Гц (2×82.41) - HPS 82.35 Гц

Струна 5: 330.53 Гц (3×110.18) - HPS 110.42 Гц

Струна 4: 293.15 Гц (2×146.58) - HPS 146.58 Гц

Струна 3: 585.00 Гц (3×195.00) - HPS 195.00 Гц

Исключение: Для 1-й струны алгоритм без окна показал верный результат (328.83 Гц), что объясняется преобладанием основной гармоники в спектре этой струны.

**2. Влияние оконной функции**

Окно Хэмминга:

Уменьшает амплитуду спектральных компонентов (65.88 - 52.26 для 6-й струны). Не решает проблему выбора гармоник, только улучшает форму спектра.

Критический случай: Для 1-й струны применение окна привело к ошибочному определению 2-й гармоники (658.67 Гц вместо 329.63 Гц), что демонстрирует необходимость комбинации окна с HPS.

**3. Точность алгоритма**

Средняя точность: ±0.46 Гц (максимальная ошибка 1.00 Гц для 3-й струны)

1. Наилучшая точность: струны 2 и 6 (0.16 Гц и 0.14 Гц соответственно)
2. Наихудшая точность: струна 3 (1.00 Гц)

Возможные причины неидеальная настройка струны во время записи, особенности спектрального состава (преобладание высших гармоник).

**4. Анализ потенциальной точности (Крамер-Рао)**

Шум не является ограничивающим фактором для точности определения частоты.

Теоретический предел точности √f\_KR: 0.00006-0.00026 Гц

Фактическая точность: 0.1 - 1.00 Гц.

Высокие значения SNR (12.93-26.82 дБ) обеспечивают идеальные условия для алгоритмов оценки частоты.

Ограничения точности связаны с алгоритмическими факторами:

Конечное разрешение БПФ (Δf = f\_s/N = 2.44 Гц при N=2048), точность интерполяции, конечная длина анализируемого сегмента.

**5. Влияние параметров системы**

Частота дискретизации f\_s = 5000 Гц:

Частота теорема Найквиста: f\_max = 2500 Гц

Покрывает весь диапазон гитарных струн (82-330 Гц) с гармониками

Длительность сигнала и размер БПФ:

N = 2048 точек при f\_s = 5000 Гц - Δf =2.44 Гц

Основной источник погрешности при прямом поиске пика

HPS и интерполяция уменьшают эту погрешность

**Выводы об эффективности алгоритма**

Алгоритм HPS эффективно устраняет ошибки определения гармоник за основную частоту  
 Достигнута точность ±0.46 Гц в среднем  
 Алгоритм устойчив к вариациям спектрального состава разных струн

Рейтинг методов (по убыванию эффективности):

1. HPS + окно Ханна - 100% успешных определений основной частоты
2. Простой поиск пика с окном - (2 из 6 струн)
3. Простой поиск пика без окна - (1 из 6 струн)

Для гитарного тюнера алгоритм обеспечивает достоверную настройку всех 6 струн, возможность реализации в реальном времени на микроконтроллере (ESP32-S3)

Вывод: Алгоритм готов к практическому применению в гитарном тюнере на базе ESP32-S3 и представляет пример эффективного использования методов цифровой обработки сигналов для решения прикладных музыкальных задач.

**3. ПРОГРАММА ДЛЯ МИКРОКОНТРОЛЛЕРА (ARDUINO ДЛЯ РАБОТЫ С ESP32S3)**

Программа гитарного тюнера на Arduino (ESP32) с OLED-дисплеем, использующая методы цифровой обработки сигналов.

Необходимые библиотеки:

#include <math.h>

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#include <algorithm>

math.h - математические функции (логарифмы, модуль и т.д.).

Wire.h - для I2C связи с OLED-дисплеем.

Adafruit\_GFX/SSD1306 - библиотеки для работы с дисплеем.

algorithm - для функций типа min(), max().

const int ADC\_GPIO = 5;

const int SDA\_PIN = 17;

const int SCL\_PIN = 18;

ADC\_GPIO - пин для подключения микрофона/звукового датчика.

SDA\_PIN/SCL\_PIN - пины для данных и тактового сигнала I2C.

const uint32\_t SAMPLE\_RATE = 10000;

Частота дискретизации (10 кГц)

const uint16\_t FFT\_SIZE = 2048;

Размер преобразования Фурье

const uint8\_t MAX\_HARMONIC = 3;

Макс. гармоника для HPS

const uint16\_t RMS\_BLOCK = 512;

Кол-во отсчётов для вычисления громкости

const double RMS\_DB\_THRESHOLD = -25.0;

Порог громкости для анализа

const int CALIBRATE\_SAMPLES = 512;

Кол-во отсчётов для калибровки

SAMPLE\_RATE 10 кГц позволяет анализировать частоты до 5 кГц, что покрывает весь диапазон гитары.

FFT\_SIZE 2048 - компромисс между точностью по частоте и скоростью расчёта.

MAX\_HARMONIC - ключевой параметр для метода HPS.

Настройки дисплея:

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, -1);

Глобальные переменные для обработки сигнала:

ArduinoFFT<double>\* FFT = nullptr;

Указатель на объект FFT.

double \*vReal = nullptr;

Массив для действительной части данных

double \*vImag = nullptr;

Массив для мнимой части данных

double \*spec = nullptr;

Массив для спектра (амплитуд)

double \*hps = nullptr;

Массив для результата HPS

double adcCenter = 2048.0;

Среднее значение АЦП ("ноль" сигнала)

Память под массивы выделяется динамически в setup().

Фильтр медианный:

const uint8\_t MEDIAN\_SIZE = 7;

double freqHistory[MEDIAN\_SIZE] = {0};

uint8\_t histIndex = 0;

Сглаживает измеренные частоты, отбрасывая случайные выбросы.

2. Модель данных для струн

struct GuitarString {

const char\* name;

double freq; Эталонная частота (Гц)

double lowBound;

double highBound;

};

Таблица эталонных частот стандартного строя гитары (E4, B3, G3, D3, A2, E2).

const GuitarString CLASSIC\_TUNING[] = {

{"1 (E)", 329.63, 320.0, 340.0},

{"2 (B)", 246.94, 240.0, 254.0},

{"3 (G)", 196.00, 190.0, 202.0},

{"4 (D)", 146.83, 142.0, 152.0},

{"5 (A)", 110.00, 106.0, 114.0},

{"6 (E)", 82.41, 80.0, 85.0}

};

Блок 1: Выделение динамической памяти и настройка нуля

vReal и vImag - массивы для действительной и мнимой частей сигнала (по 2048 элементов)

spec и hps - массивы для спектра и результата HPS (по 1024 элемента)

vReal = (double\*)malloc(sizeof(double) \* FFT\_SIZE);

vImag = (double\*)malloc(sizeof(double) \* FFT\_SIZE);

spec = (double\*)malloc(sizeof(double) \* (FFT\_SIZE/2));

hps = (double\*)malloc(sizeof(double) \* (FFT\_SIZE/2));

analogReadResolution(12);

analogSetAttenuation(ADC\_11db);

double sum = 0;

uint32\_t calStartTime = micros();

Калибровка нулевого уровня

Считывается 512 отсчетов с АЦП при отсутствии сигнала

Вычисляется среднее значение

Результат сохраняется в переменной adcCenter

for (int i = 0; i < CALIBRATE\_SAMPLES; i++) {

sum += analogRead(ADC\_GPIO);

while (micros() - calStartTime < i \* (1000000.0 / SAMPLE\_RATE)) {}

}

adcCenter = sum / CALIBRATE\_SAMPLES;

Создание объекта FFT

FFT = new ArduinoFFT<double>(vReal, vImag, FFT\_SIZE, SAMPLE\_RATE);

}

Блок 2: Определение наличия сигнала

void loop() {

double sum = 0;

uint32\_t rmsStartTime = micros();

Расчет уровня сигнала (RMS)

Считывается 512 отсчетов

Для каждого отсчета вычитается калибровочное значение

Вычисляется сумма квадратов отклонений

Извлекается квадратный корень для получения RMS

for (uint16\_t i = 0; i < RMS\_BLOCK; i++) {

double s = analogRead(ADC\_GPIO) - adcCenter;

sum += s \* s;

while (micros() - rmsStartTime < i \* (1000000.0 / SAMPLE\_RATE)) {}

}

Преобразование в dBFS и проверка порога

double dbfs = 20.0 \* log10((rms + 1e-12) / adcCenter);

if (dbfs < RMS\_DB\_THRESHOLD) { // -25 дБ

display.println("No signal");

return; // Выход из loop() если сигнал слабый

}

RMS преобразуется в децибелы относительно полной шкалы (dBFS)

Сравнивается с порогом -25 дБ

Если сигнал слишком слабый - отображается "No signal" и цикл прерывается

double rms = sqrt(sum / RMS\_BLOCK);

double dbfs = to\_dbfs(rms, adcCenter);

uint32\_t sampleStartTime = micros();

БЛОК 3: Захват и предобработка сигнала

for (uint16\_t i = 0; i < FFT\_SIZE; i++) {

vReal[i] = analogRead(ADC\_GPIO) - adcCenter;

vImag[i] = 0.0;

while (micros() - sampleStartTime < i \* (1000000.0 / SAMPLE\_RATE)) {}

}

Считывается 2048 отсчетов (FFT\_SIZE) с частотой 10 кГц

Каждый отсчет корректируется вычитанием adcCenter

Мнимая часть инициализируется нулями

Расчет фактической частоты дискретизации

Измеряется реальное время захвата 2048 отсчетов и вычисляется фактическая частота дискретизации (для мониторинга).

uint32\_t actualDuration = micros() - sampleStartTime;

float actualFs = (FFT\_SIZE \* 1000000.0) / actualDuration;

БЛОК 4: Спектральный анализ сигнала

4.1. Применение оконной функции и БПФ

FFT->windowing(FFTWindow::Hamming, FFTDirection::Forward);

FFT->compute(FFTDirection::Forward);

FFT->complexToMagnitude();

vReal[0] = 0.0;

Окно Хэмминга: Умножение сигнала на оконную функцию для уменьшения эффекта "растекания спектра"

БПФ: Быстрое преобразование Фурье (2048 точек)

complexToMagnitude: Вычисление модуля комплексного спектра

uint32\_t half = FFT\_SIZE / 2;

double deltaF = (double)SAMPLE\_RATE / FFT\_SIZE;

4.2. Нормализация спектра

Находится максимальное значение спектра

Все значения спектра делятся на максимум (нормализация к диапазону [0, 1])

Добавляется малое значение 1e-12 для избежания проблем с log(0)

double maxSpec = 1e-12;

for (uint32\_t i = 0; i < half; i++) {

spec[i] = vReal[i] + 1e-12;

if (spec[i] > maxSpec) maxSpec = spec[i];

}

for (uint32\_t i = 0; i < half; i++) {

spec[i] /= maxSpec;

}

БЛОК 5: Алгоритм HPS (Harmonic Product Spectrum)

5.1. Инициализация HPS логарифмами спектра

Создается начальный массив HPS как логарифм спектра. Логарифмирование используется вместо умножения для:

Уменьшения вычислительной сложности (сложение вместо умножения)

Улучшения численной устойчивости

for (uint32\_t i = 0; i < half; i++) {

hps[i] = log(spec[i] + 1e-12);

}

Понижающая дискретизация и сложение.

Для k = 2: берется каждый 2-й элемент спектра (децимация в 2 раза)

Для k = 3: берется каждый 3-й элемент (децимация в 3 раза)

Логарифмы прореженных спектров складываются с исходным.

for (uint8\_t k = 2; k <= MAX\_HARMONIC; k++) {

for (uint32\_t i = 0; i < half / k; i++) {

hps[i] += log(spec[i \* k] + 1e-12);

}

}

БЛОК 6: Определение основной частоты

Поиск пика в ограниченном диапазоне

Определяется диапазон поиска (70-400 Гц) - охватывает все гитарные струны.

Ищется индекс максимального значения в массиве HPS.

Находится частота: rawFreq = maxHpsIdx × deltaF.

deltaF = SAMPLE\_RATE / FFT\_SIZE = 10000 / 2048 = 4.88 Гц (разрешение по частоте).

uint32\_t minIdx = max(1, (int)round(70.0 / deltaF));

uint32\_t maxIdx = min((int)half - 1, (int)round(400.0 / deltaF));

double maxHps = -1e300;

int maxHpsIdx = minIdx;

for (uint32\_t i = minIdx; i <= maxIdx; i++) {

if (hps[i] > maxHps) {

maxHps = hps[i];

maxHpsIdx = i;

}

}

double rawFreq = maxHpsIdx \* deltaF;

6.2. Параболическая интерполяция

1. Берутся три точки вокруг пика: hps[k-1], hps[k], hps[k+1]

Строится парабола через эти точки

Находится точное положение вершины параболы (дробный индекс)

Уточненная частота: fineFreq = (k + delta) × deltaF

double fineFreq = parabolic\_interpolation(hps, maxHpsIdx, half) \* deltaF;

double parabolic\_interpolation(const double \*mag, int k, int Nhalf) {

if (k <= 0 || k >= Nhalf - 1) return (double)k;

double alpha = mag[k - 1];

double beta = mag[k];

double gamma = mag[k + 1];

double denom = alpha - 2.0 \* beta + gamma;

if (fabs(denom) < 1e-12) return (double)k;

return k + 0.5 \* (alpha - gamma) / denom;

}

double to\_dbfs(double rms, double fullscale) {

return 20.0 \* log10((rms + 1e-12) / fullscale);

}

6.3. Медианная фильтрация

Создается массив из 7 последних измерений

При каждом новом измерении массив сортируется

Возвращается медианное значение (середина отсортированного массива). Это делается для подавления случайных выбросов, сглаживания результатов, обнаружения смены струны (резкое изменение >20 Гц).

double stableFreq = medianFilter(fineFreq);

int detectedString = detectString(stableFreq);

String tuningAdvice = "";

double freqDiff = 0;

if (detectedString >= 0) {

freqDiff = stableFreq - CLASSIC\_TUNING[detectedString].freq;

tuningAdvice = getTuningAdvice(stableFreq, CLASSIC\_TUNING[detectedString].freq);

}

double medianFilter(double newFreq) {

static int initialized = 0;

static double lastValidFreq = 0;

// скользящее среднее

if (initialized < MEDIAN\_SIZE) {

freqHistory[initialized] = newFreq;

initialized++;

// Среднее арифметическое для начала записи

double sum = 0;

for(int i = 0; i < initialized; i++) {

sum += freqHistory[i];

}

lastValidFreq = sum / initialized;

if (initialized == MEDIAN\_SIZE) {

Serial.print("Filter initialized. Initial median: ");

Serial.println(lastValidFreq, 1);

}

return lastValidFreq;

}

// Проверка на смену струны

if (lastValidFreq > 0 && fabs(newFreq - lastValidFreq) > 20.0) {

for(int i = 0; i < MEDIAN\_SIZE; i++) {

freqHistory[i] = newFreq;

}

histIndex = 0;

double medianAfterReset = newFreq; // сначала все значения будут одинаковые

lastValidFreq = medianAfterReset;

return medianAfterReset;

}

freqHistory[histIndex] = newFreq;

histIndex = (histIndex + 1) % MEDIAN\_SIZE;

double temp[MEDIAN\_SIZE];

memcpy(temp, freqHistory, sizeof(temp));

for (int i = 0; i < MEDIAN\_SIZE - 1; i++) {

for (int j = i + 1; j < MEDIAN\_SIZE; j++) {

if (temp[i] > temp[j]) {

double swap = temp[i];

temp[i] = temp[j];

temp[j] = swap;

}

}

}

double currentMedian = temp[MEDIAN\_SIZE / 2];

lastValidFreq = currentMedian;

return currentMedian;

}

БЛОК 7: Идентификация струны и рекомендации

7.1. Определение номера струны

Проверяется, попадает ли частота в диапазон одной из струн

Если нет - ищется струна с минимальной разницей частот

if (detectedString >= 0) {

String stringNum = String(CLASSIC\_TUNING[detectedString].name);

int spacePos = stringNum.indexOf(' ');

if (spacePos > 0) {

stringNum = stringNum.substring(0, spacePos);

}

}

}

int detectString(double frequency) {

for (int i = 0; i < STRING\_COUNT; i++) {

if (frequency >= CLASSIC\_TUNING[i].lowBound &&

frequency <= CLASSIC\_TUNING[i].highBound) {

return i;

}

}

int closestString = 0;

double minDiff = 1000.0;

for (int i = 0; i < STRING\_COUNT; i++) {

double diff = fabs(frequency - CLASSIC\_TUNING[i].freq);

if (diff < minDiff) {

minDiff = diff;

closestString = i;

}

}

return closestString;

}

7.2. Генерация рекомендаций по настройке

На основе разницы между измеренной и эталонной частотой генерируется текстовое указание.

String getTuningAdvice(double currentFreq, double targetFreq) {

double diff = currentFreq - targetFreq;

double absDiff = fabs(diff);

Функция cleanup(): Управление памятью и корректное завершение работы.

delete FFT вызывает деструктор класса ArduinoFFT<double>, который освобождает внутренние ресурсы.

Присвоение nullptr делает указатель "пустым".

free(указатель) - освобождение блока динамической памяти.

void cleanup() {

if (FFT != nullptr) {

delete FFT;

FFT = nullptr;

}

if (vReal != nullptr) free(vReal);

if (vImag != nullptr) free(vImag);

if (spec != nullptr) free(spec);

if (hps != nullptr) free(hps);

}

**Заключение**

В ходе выполнения проекта была проведена проверка работоспособности разработанного тюнера для акустической/классической гитары, включая оценку корректности определения частоты, устойчивости показаний во времени и сравнение результатов с эталонными значениями.

Проведённые испытания показали, что устройство корректно определяет основную частоту звучания всех шести струн гитары в рабочем диапазоне частот. Определение частоты происходит стабильно как при устойчивом сигнале, так и при постепенном затухании колебаний струны. По сравнению с эталонным электронным тюнером погрешность измерений не превышает приблизительно 0,5 Гц, что соответствует требованиям к точности настройки музыкального инструмента и обеспечивает комфортную работу пользователя.

Благодаря использованию алгоритма Harmonic Product Spectrum и параболической интерполяции удалось повысить точность определения частоты по спектру и снизить влияние дискретизации по частоте в БПФ. Дополнительно применён медианный фильтр, который обеспечивает сглаживание случайных выбросов и стабилизацию показаний во времени. В результате показания частоты не «прыгают», изменения отображаются плавно и адекватно реальному процессу настройки струны.

Задержка реакции системы является минимальной и не ощущается пользователем при работе. Обновление результатов происходит достаточно быстро, что позволяет выполнять настройку инструмента в реальном времени.

Таким образом, разработанный тюнер является работоспособным и пригодным для практического применения. Реализованные алгоритмы цифровой обработки сигнала подтвердили свою эффективность, обеспечив высокую точность и устойчивость определения частоты. Поставленные в проекте задачи выполнены, цели разработки достигнуты.

**Список используемых источников**

<https://github.com/hukenovs/dsp-theory/tree/master>

<https://www.jobilize.com/course/section/harmonic-product-spectrum-by-openstax>

<https://github.com/rtek1000/YD-ESP32-23>

**Ссылка на гитхаб:**

<https://github.com/Olezzhha/Piezo_tuner>