Proyecto Fin de Carrera
Ingeniería de Tecnologías Industriales

Desarrollo de un afinador electrónico para instrumento musical basado en DSP TMS320LF28335

Autora: Irene Romero Romero
Tutor: Federico Barrero Garcia

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Carrera
Ingeniería de Tecnologías Industriales

Desarrollo de un afinador electrónico para instrumento musical basado en DSP TMS320LF28335

Autora:
Irene Romero Romero

Tutor:
Federico Barrero García
Catedrático de Universidad

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Proyecto Fin de Carrera: Desarrollo de un afinador electrónico para instrumento musical basado en DSP TMS320LF28335

Autora: Irene Romero

Tutor: Federico Barrero García

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Agradecer a mi familia, en especial a mis padres y mi hermana, que han sido los que han estado presentes durante estos cuatro años, apoyándome en los momentos más complicados de la carrera. El ánimo que me han aportado ha sido muy importante para seguir adelante con todas las dificultades.

Cabe destacar una mención a mis amigos. En primer lugar, a mis amigas de la infancia, por haber estado siempre conmigo y un apoyo con las que desahogarme. A Amalia, amiga y compañera de la carrera, quien ha sido un pilar fundamental durante estos años, con la que he pasado tardes infinitas estudiando, sufriendo, disfrutando y compartiendo experiencias juntas. También, agradecer a todos mis compañeros, que muchos se han convertido en amigos con los que hacían que el día a día en la ETSI fuera más llevadero.

Por último, agradecer a mi tutor del TFG y todos los profesores de la ETSI, por su esfuerzo, dedicación y todo el conocimiento que me han impartido para poder llegar a ser una buena ingeniera.

Irene Romero Romero

Sevilla, 2024

Resumen

En este documento se expone el desarrollo de un afinador de guitarra, el cual es capaz de detectar la frecuencia recogidas por el micrófono, y así poder afinar nuestro instrumento de forma correcta sin necesidad de tener conocimientos musicales. De esta manera, nos aseguramos de que realmente tenemos una buena afinación.

Este afinador contiene una serie de componentes que contribuyen a la obtención y procesamiento de datos, de entre los cuales está incluido un micrófono amplificador y el Microprocesador TMS320F28335. Además, incluye una pantalla LCD y un modulo bluetooth para interactuar con el dispositivo.

El sofware del afinador es el encargado de procesar la señal y hacer las operaciones necesarias para obtener los resultados que queremos. Por otro lado, establece las conexiones entre el microprocesador y el módulo bluetooth. Por último, también se incluye un programa para el control de la pantalla y mostrarnos la afinación del instrumento.

Abstract

This document explains the development of a guitar tuner, which is capable of detecting the frequency picked up by the microphone, and thus being able to tune our instrument correctly without having to have musical knowledge. Thanks to this, we make sure that we really have a good tuning.

This tuner contains a sort of components that help to data acquisition and processing, including a microphone amplifier and the TMS320F28335 Microprocessor. In addition, it includes an LCD screen and a Bluetooth module to interact with the device.

The tuner sofware is in charge of processing the signal and carring out the necesary operations to obtain the results we want. On the other hand, it establishes the connections between the microprocessor and the bluetooth module. Finally, a program is also included to control the screen and show us the tuning of the instrument.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

DSP	Digital Signal Processors
GPIO	General-Purpose Input Output
ADC	Convertidor analógico-digital
SCI	Serial Communication Interface
V	Voltios (unidad de diferencia de potencial eléctrico)
Hz	Hercios (unidad de frecuencia)
mm	milímetros
FFT	Transformada rápida de Fourier
DFT	Transformada de Fourier discreta
sin	Función seno
cos	Función coseno

1 INTRODUCCIÓN

Música y tecnología siempre han estado estrechamente ligadas. Concretamente, la tecnología ha estado al servicio de la música en todos los ámbitos que podamos imaginar. Desde la fabricación de instrumentos electrónicos, simuladores, editores de audio, reproductores de audio hasta afinadores. Todos estos avances en la música nos han permitido experimentar, crear nuevos sonidos y otras maneras de interactuar con ella.

1.1 Contexto y motivación

Durante este proyecto se pretende crear un afinador de guitarra que reúna las características necesarias para poner en práctica todos los conocimientos adquiridos durante toda la carrera, especialmente en electrónica. Además, se ha tratado de elegir un tema en el que involucre uno de los hobbies que más me emocionan: la música.

La electrónica ha conseguido acercarnos la música en todos los sentidos, como puede ser el tenerla al alcance de nuestra mano, pudiendo escuchar cualquier cosa con un simple clic, como también poder crear música con millones de programas que existen hoy en día en internet o incluso permitiéndonos aprender de manera sencilla a tocar un instrumento. Es este punto en el que se centra este documento. Desde mi experiencia personal, la música ha sido un factor importante en mi vida que siempre me ha acompañado en mi día a día. El hecho de querer aprender a tocar un instrumento hizo darme cuenta de lo importante que podía llegar a ser la tecnología. Antes de querer tocar cualquier instrumento tenemos que asegurarnos de que podemos sacarle el máximo rendimiento y que suene de manera correcta. En otro momento en el que la tecnología no estuviera tan avanzada, tendría que haber asistido a algún conservatorio o atender a clases particulares que me hubieran enseñado como se afina una guitarra. La manera clásica y la que siempre se ha llevado a cabo es la afinación a oído, y eso es algo que no podemos aprender al instante, ya que se necesita varios años de experiencia, para poder afinarla perfectamente. Es un paso muy simple, pero que puede impedirnos que queramos empezar a tocar un instrumento por nuestra cuenta. El afinador es un dispositivo que nos ahorra tiempo en aprender y dinero en que nos enseñen a cómo afinar un instrumento. El hecho de que la tecnología nos facilite algunos aspectos de nuestra vida es lo que ha hecho que me enamore de ella.

1.2 Exposición del problema

A lo largo de la historia, la manera de afinar la guitarra ha ido evolucionando y perfeccionándose. Los métodos clásicos que han existido desde siempre son los de afinación de oído o por armónicos. Básicamente son métodos los cuales simplemente mediante la escucha de las cuerdas, el guitarrista ya sabe si están afinadas o no, o si se prefiere, teniendo una nota de referencia. Normalmente, esta nota de referencia puede ser la de un diapasón. El diapasón es un objeto que vibra y que suena en la nota LA a 440 Hz, misma frecuencia a la que suena la quinta cuerda si pulsamos en el quinto traste. Por medio de comparación a oído, se tiene que ir ajustando el sonido, aflojando o apretando las cuerdas de la guitarra. A partir de esta cuerda, ya se podrá afinar el resto de las cuerdas por el mismo método.

Hoy en día, hay millones de aplicaciones móviles que trabajan como un afinador de guitarra. Simplemente, tocando una cuerda, nos va indicando en tiempo real si está baja o alta de afinación. El inconveniente de estas aplicaciones es que suelen estar llenas de anuncios, cosa que hace que el simple proceso de afinación sea muy tedioso ya que se tiene que estar constantemente esperando a que se pasen estos videos de publicidad. Además, muchas de ellas no llegan a funcionar como se espera, o bien porque no llegan a detectar el sonido y no indican

nada del estado de la afinación, o bien porque no afinan de manera correcta.

Si nos centramos en afinadores físicos, existen los afinadores de clip, de pedal, el cromático, de los cuales se hablarán más delante de forma más detallada. El afinador de pedal, por ejemplo, es un dispositivo bastante pesado. Esto puede no ser ideal para músicos que necesitan transportar su equipo con frecuencia o que prefieren un setup minimalista. El tamaño y peso adicionales pueden complicar el transporte y la configuración del equipo, especialmente en espacios reducidos o en situaciones donde la movilidad es crucial. Además, si se compara con otro tipo de afinadores, este suele ser más caro. Esto puede ser un factor limitante para músicos principiantes o aquellos con presupuestos ajustados, quienes podrían optar por alternativas más económicas, aunque menos precisas o convenientes.

Por otro lado, se tienen los afinadores de clip. Estos dispositivos son mucho más pequeños y baratos. Pese a sus ventajas, estos dispositivos tienen muchos más inconvenientes, como puede ser su durabilidad, sensibilidad a vibraciones externas o visibilidad de la pantalla. Debido a su diseño compacto y ligero, pueden ser menos duraderos que otros tipos de afinadores. Las partes móviles, como el clip en sí, pueden desgastarse o romperse con el tiempo, especialmente con un uso frecuente. Esto puede llevar a una vida útil más corta y la necesidad de reemplazos más frecuentes. Además, este tipo de afinadores funcionan detectando las vibraciones del instrumento, lo que los hace susceptibles a captar vibraciones de otras fuentes. Esto puede ser especialmente problemático en entornos ruidosos.

A pesar de la amplia variedad de opciones disponibles en el mercado para afinar la guitarra, cada una presenta desventajas significativas que pueden afectar la experiencia del usuario. Estas limitaciones demuestran que ninguno de los afinadores existentes logra satisfacer plenamente todas las necesidades de los guitarristas, por lo menos aquellos que son principiantes en el mundo de la música. Esta situación subraya la necesidad de un afinador que combine la precisión, la durabilidad, la facilidad de uso y la portabilidad sin comprometer ningún aspecto crítico.

Por ello, este Trabajo de Fin de Grado se dedica al desarrollo de un afinador de guitarra que busque cubrir todas estas necesidades, ofreciendo una solución innovadora y completa que supere las limitaciones de los afinadores actualmente disponibles. La meta es crear un dispositivo que proporcione una afinación precisa y confiable en cualquier entorno, sea accesible para músicos de todos los niveles y mejore la experiencia general de afinación de la guitarra.

1.3 Objetivo

El principal objetivo de este trabajo es la realización de un afinador que sea lo más intuitivo posible y que facilite la tarea de la afinación. Este dispositivo estará compuesto de una pantalla en el que se indique la frecuencia a la que está sonando las cuerdas de nuestra guitarra. Esto estará acompañado de unos seis leds, los cuales cada uno corresponden a una cuerda, y que se encenderán según la frecuencia que se detecte. Además, se podrá conectar con un dispositivo móvil para indicar concretamente que cuerda se quiera afinar.

En adición, en este trabajo se hará una comparación entre dos métodos para procesar la señal y así poder calcular la frecuencia. Finalmente, se verá cuál de ellos será el más efectivo y preciso.

1.4 Alcance

El alcance de este proyecto consiste en la integración de varios componentes conectados a través de los distintos periféricos de nuestro microprocesador, así como la programación del cálculo de la frecuencia. Se quiere estudiar la forma de onda que recoge nuestro micrófono y averiguar de qué manera se podrá procesar esta señal para obtener los resultados que se quieren. La finalidad es entender cómo funciona un microprocesador y ver las distintas funcionalidades que tiene.

1) Estudio de dos métodos por los que se calcule la frecuencia. A través de la experimentación y observación se podrán deducir cuales de estos métodos será el óptimo.
2) Integración de un micrófono amplificador, esencial para nuestro dispositivo. Es el encargado de recoger

la señal.

3) Integración de una pantalla en el que el usuario que use el afinador pueda ver a qué frecuencia está sonando cada una de las cuerdas.

4) Incorporación de un módulo Bluetooth para que se pueda conectar a un dispositivo móvil el afinador e indicar que cuerda de la guitarra se quiere afinar. Además, el afinador llevará incorporado 6 leds que indicarán la cuerda de la guitarra que se esté afinando.

1.5 Estructura del trabajo

Para abordar este trabajo, será esencial el orden en el que se vaya a abordar cada tarea. Es por eso que se han ordenado y descrito cada una de ellas:

- Búsqueda de información y referencias: para tener una guía sobre cómo abordar el problema planteado, se buscarán algunas referencias que pudieran facilitar el desarrollo del dispositivo.
- Selección de componentes: es un paso esencial, ya que dependiendo del componente que se escoja, el afinador tendrá unas características u otras. Además, para la validación de algunos componentes es imprescindible que se hagan pruebas para ver su comportamiento y, en consecuencia, actuar de la manera pertinente para su posterior implementación.
- Implementación de código: en este punto es cuando se comenzará a escribir el código del programa. A su misma vez, mientras que se añada un nuevo segmento de código este se irá probando para verificar que funciona correctamente.
- Simulaciones: una vez implementado el código completo, se harán test, donde se ponga a prueba como es de fiable y eficaz el dispositivo. Se hará un estudio cualitativo de cuáles han sido los resultados obtenidos.
- Análisis de resultados: finalmente, a partir del resultado final, se compararán los métodos abordados para obtener un mismo objetivo, y se decidirá cuál de ellos es el más favorable, encontrando un equilibrio entre fiabilidad, rapidez y exactitud.

2 REVISIÓN DE LA LITERATURA

La música y la ingeniería han estado relacionadas desde hace décadas. La ingeniería ha permitido el desarrollo de instrumentos musicales más sofisticados y de sistemas de amplificación y grabación de sonido que han revolucionado la forma en que escuchamos y producimos música.

Así, desde los instrumentos musicales más primitivos hasta el nacimiento del fonógrafo en la era industrial y del ordenador en la digital, la música ha experimentado diferentes cambios y modificaciones que han ido despertando en la sociedad nuevas modalidades de producción y recepción de las obras.

Hoy día, estos cambios se relacionan con la creciente digitalización de la música, aspecto más que evidente cuando observamos que toda la música que se produce actualmente se hace a través del ordenador e introduce elementos de informática musical.

2.1 Avances de la ingeniería en la música

Para remarcar la importancia de la ingeniería en la música, se va a hacer un repaso de los inventos más importantes a lo largo del último siglo. Son instrumentos que en hoy día se tienen muy normalizados, pero que en su momento significaron algo revolucionario para la historia.

2.1.1 El fonógrafo

Algo tan simple, y que se puede tener al alcance de las manos es el hecho de poder grabar y reproducir sonidos. Hace no tanto, poder escuchar música era un privilegio que solo tenían personas de la alta sociedad, ya que se necesitaba llamar a una orquesta para que fueran a tocar en directo.

En 1877, Thomas Edison patentó el fonógrafo, un dispositivo que efectivamente graba y reproduce sonidos. Consistía en un cilindro giratorio en el que se graban las ondas del sonido. A través de un diafragma y una aguja, se registran las vibraciones acústicas del sonido en el cilindro. Al reproducir el cilindro giratorio, el diafragma y la aguja generan vibraciones que se convierten en ondas sonoras amplificadas, creando así la reproducción del sonido grabado.

Ilustración 1 Fonógrafo

La evolución de este instrumento fue el gramófono, que lo único en lo que se diferenciaba era que, en lugar de grabar sobre un cilindro, lo hacía sobre un disco plano. Si pensamos en la importancia que ha tenido esto en posteriores inventos, esto es la base de como interactuamos con la música. El principio en el que se basa el micrófono es el mismo que el del fonógrafo, elemento esencial para el afinador de guitarra.

2.1.2 El sintetizador

La historia del sintetizador es fascinante y está estrechamente ligada al desarrollo de la música electrónica y la tecnología sonora. Este ha desempeñado un papel fundamental en la evolución de la música moderna y en la forma en que la sociedad percibe y experimenta el arte sonoro. Desde sus humildes comienzos en los laboratorios de investigación hasta su presencia omnipresente en la música popular y la cultura contemporánea, este instrumento ha sido un catalizador para la innovación creativa y la expresión musical.

Un sintetizador consta de generadores de sonido, moduladores, filtros y amplificadores, que trabajan en conjunto para crear y manipular señales eléctricas y generar una amplia gama de sonidos. Estos instrumentos permiten a los músicos controlar diversos parámetros, como la frecuencia, la amplitud y la forma de onda, ofreciendo una flexibilidad creativa sin precedentes.

Ilustración 2 Sintetizador

El sintetizador ha sido fundamental en la expansión de géneros musicales como la música electrónica, el synthpop y el ambient, proporcionando nuevos sonidos y texturas que han enriquecido el panorama musical. Su evolución continua sigue inspirando a músicos, artistas y oyentes en todo el mundo, demostrando su relevancia perdurable en un mundo cada vez más digitalizado y diverso.

2.1.3 Las mesas de mezclas

Un invento con el que quizás no toda la población esté tan familiarizada son las mesas de mezclas. Antes de la invención de las mesas de mezclas, la producción musical era un proceso considerablemente más rudimentario y limitado en comparación con las prácticas modernas. En las primeras etapas de la grabación musical, las actuaciones se registraban en vivo en una sola toma. Los músicos se reunían en un estudio de grabación y tocaban juntos, y la señal de audio se capturaba en un único canal o pista de grabación. Esta técnica requería una ejecución precisa y casi perfecta de principio a fin, ya que cualquier error o imperfección tenía que ser aceptado o corregido mediante la repetición de toda la toma.

En la década de 1950 comenzaron a surgir las primeras mesas de mezclas modernas, diseñadas para controlar y manipular múltiples canales de audio que permitieron a los ingenieros de sonido combinar múltiples fuentes de audio de manera precisa y controlada, creando una mezcla final que resalta las mejores cualidades de cada elemento. Desde entonces, las mesas han evolucionado desde simples consolas analógicas hasta complejos sistemas digitales con capacidades avanzadas de procesamiento de audio.

Ilustración 3 Mesa de mezcla

Las mesas de mezclas han sido y siguen siendo una parte integral de la producción musical moderna. Su evolución desde simples consolas analógicas hasta complejos sistemas digitales refleja la constante búsqueda de calidad y eficiencia en el mundo del sonido. Además, su importancia en la música contemporánea es innegable, ya que han permitido a músicos de todos los géneros y niveles de habilidad crear y compartir su arte con el mundo de manera más accesible que nunca.

2.1.4 El micrófono

Otro de los inventos con los que estamos muy familiarizados es el micrófono. Es un dispositivo esencial en el campo de la captura de audio. Su función principal es convertir las ondas sonoras en señales eléctricas, que luego pueden ser procesadas, almacenadas o transmitidas electrónicamente. Si pensamos en las aplicaciones de este instrumento, no solo está presente en la música, sino que también lo podemos ver la rama de las telecomunicaciones, como la televisión, la radio, la telefonía y muchos otros campos que nos podamos imaginar.

Ilustración 4 Micrófono

Con la aparición de la tecnología digital, este aparato ha desempeñado un papel aún más fundamental al permitir la digitalización del sonido. El micrófono desempeña un papel central en la digitalización del sonido al convertir las ondas sonoras en señales eléctricas que pueden ser muestreadas y convertidas en formato digital por dispositivos como convertidores analógico digital (ADC). Esta digitalización del sonido ha transformado la forma en que se graba, se edita y se reproduce el audio en la era moderna, proporcionando una mayor fidelidad y versatilidad en el proceso de producción musical y audiovisual.

2.2 Métodos de afinación usados por los guitarristas

Teniendo claro la importancia que ha tenido la ingeniería a lo largo de toda la historia, ahora nos centraremos en qué métodos han estado usando los guitarristas durante toda la historia desde que existe este instrumento.

2.2.1 Afinación mediante diapasón

El diapasón es una horquilla metálica, generalmente de acero, que al golpearlo o pellizcarlo en las dos ramas, comienza a vibrar en una frecuencia determinada que depende de su tamaño. Esta herramienta no emite sonido, por lo que dicha amplificación debe ser amplificada de alguna forma. En las guitarras clásicas o acústicas, se apoya sobre la propia caja de resonancia y esta amplifica la vibración.

Ilustración 5 Diapasón

Cuando se golpea el diapasón, emite un sonido puro de un LA, en concreto a 440 Hz. Si pulsamos la quinta cuerda en el quinto traste, este debe sonar a esta frecuencia, por lo tanto, mediante las clavijas tendremos que ajustar a oído la cuerda para que suene igual. Una vez que se ha afinado la cuerda de referencia, se utilizan intervalos musicales para afinar las cuerdas restantes en relación con esta cuerda. Por ejemplo, se puede usar la quinta cuerda como referencia para afinar la cuarta cuerda utilizando una quinta justa.

2.2.2 Afinador de clip

Un afinador de clip es un tipo específico de afinador electrónico que se sujeta directamente al instrumento musical utilizando un clip incorporado. Debido a su tamaño compacto, son fáciles de transportar y almacenar, lo que los hace ideales para músicos que están en movimiento.

Ilustración 6 Afinador de clip

El afinador de clip captura las vibraciones del instrumento a través del clip y utiliza estas vibraciones para detectar la frecuencia de la nota que se está reproduciendo. Luego, muestra la afinación en una pantalla LED o LCD, indicando si la nota está afinada correctamente, por encima o por debajo del tono deseado.

2.2.3 Afinador de pedal

Un afinador de pedal es un tipo de afinador electrónico que se utiliza comúnmente con guitarras eléctricas, bajos u otros instrumentos que se conectan a un amplificador a través de una pedalera de efectos.

Ilustración 7 Afinador de pedal

El músico conecta su instrumento al afinador de pedal a través de un cable de instrumento estándar. Cuando se activa el pedal, el músico puede tocar una cuerda y el afinador mostrará la nota detectada y su afinación en una pantalla LED o LCD. Los afinadores de pedal son ideales para actuaciones en vivo, ya que permiten al músico afinar su instrumento sin tener que depender de dispositivos externos como afinadores de clip o aplicaciones móviles. Sin embargo, para una guitarra que no tenga una salida de señal del sonido, no nos vale para nada este afinador.

2.2.4 Afinador cromático

Los afinadores cromáticos son dispositivos electrónicos diseñados para detectar y mostrar cualquier nota musical, sin limitarse a notas específicas de un instrumento en particular.

Ilustración 8 Afinador cromático

Estos afinadores muestran la nota detectada por el dispositivo y si está afinada correctamente, por encima o por debajo del tono deseado. Utilizan una pantalla digital que muestra la nota detectada y una aguja o indicador luminoso que indica si la nota está afinada. Algunos afinadores cromáticos ofrecen diferentes modos de afinación, como afinación estándar, afinación por temperamento igual, afinación por intervalos justos, etc. Esto permite adaptar el afinador a las necesidades específicas del músico.

3 FUNDAMENTOS TEÓRICOS

Para la realización de este trabajo, es importante destacar los conocimientos adquiridos en las asignaturas de informática, electrónica industrial y sistemas electrónicos digitales, en la que se mostró los conocimientos básicos de la programación, y en concreto de la programación de un microprocesador y sus distintos periféricos. Gracias a ellos se podrá integrar la programación en nuestro proyecto con gran fluidez. Además de ello, se debe conocer el instrumento que se va a afinar con nuestro dispositivo.

3.1 La guitarra

Antes de introducirnos en conceptos más técnicos, es necesario que se conozca el instrumento que se va a tratar de afinar: la guitarra. La guitarra tiene una gran variedad de tamaños, tipos y formas, pero algunos aspectos son comunes en todas ella. Una guitarra comprende de tres partes básicas: un cuerpo, un mástil, un clavijero y las cuerdas. Generalmente este instrumento usa seis cuerdas de diferente grosor para generar el sonido.

Ilustración 9 Partes de una guitarra

3.1.1 El cuerpo

Es la parte más grande y voluminosa de la guitarra. Por lo general, está hecho de madera, como el cedro, el abeto o el palosanto, entre otros tipos de madera. El cuerpo está dividido en dos partes principales: la tapa armónica y la caja de resonancia.

La tapa armónica es la parte superior del cuerpo y es donde se encuentra el agujero de resonancia, que permite que el sonido se amplifique. La caja de resonancia es la parte inferior del cuerpo y amplifica el sonido producido por las cuerdas. En el interior de la caja de resonancia se encuentra el refuerzo, una estructura de madera que ayuda a mantener la integridad estructural del cuerpo y a mejorar la calidad del sonido.

3.1.2 El mástil

Es la parte larga y delgada de la guitarra que se extiende desde el cuerpo hasta el clavijero. En el mástil se encuentra el diapasón, una superficie plana y lisa sobre la que se colocan las cuerdas y se presionan para producir diferentes notas. A lo largo de todo el mástil, se encuentran los trastes, pequeñas divisiones metálicas incrustadas en el diapasón que ayudan a dividir el mástil en diferentes notas musicales cuando se presionan las cuerdas.

3.1.3 El clavijero

Es la parte superior del mástil, donde se ubican las clavijas de afinación. Las clavijas son pequeñas piezas metálicas que se utilizan para ajustar la tensión de las cuerdas y afinar el instrumento. Cada clavija corresponde a una cuerda específica, y al girarlas en sentido horario o antihorario, se aumenta o disminuye la tensión de la cuerda, lo que modifica su afinación.

3.1.4 Las cuerdas

Es la parte de la guitarra que produce el sonido, y que según cómo estén ajustadas sonarán de una manera u otra. Cada cuerda de la guitarra debe sonar a una frecuencia distinta, y tienen los siguientes valores:

1) Primera cuerda: 82 Hz. Es la que tiene el sonido más grave.
2) Segunda cuerda: 110 Hz.
3) Tercera cuerda: 146 Hz.
4) Cuarta cuerda: 196 Hz.
5) Quinta cuerda: 246 Hz.
6) Sexta cuerda: 329 Hz. Es la que tiene el sonido más agudo.

Además, es importante resaltar que las cuerdas de una guitarra clásica están hechas de diferentes materiales y tienen distintos grosores, factor que afectará al sonido y duración del sonido que emite cada cuerda.

- Cuerdas Graves (bajas): Generalmente son más gruesas y están hechas de un núcleo de nylon envuelto con alambre metálico. Estas cuerdas tienden a vibrar más tiempo debido a su mayor masa y a la mayor cantidad de energía que pueden almacenar.
- Cuerdas Agudas (altas): Son más delgadas y están hechas solo de nylon. Tienen menos masa y, por lo tanto, menos inercia, lo que resulta en una menor duración de la vibración.

3.2 Formas de onda

Al puntear una cuerda ésta empieza a vibrar. Esta vibración se transmite a través del puente al cuerpo del instrumento, provocándola resonancia en su interior, esa vibración recorre la tapa armónica y la tapa trasera, se crean las ondas sonoras en el interior del cuerpo que se expandirán a través de la boca.

Las formas de onda que producen las cuerdas de la guitarra son ondas senoidales. Las ondas que emiten las cuerdas no son ideales, por lo que, si estas se estudian a través de un osciloscopio, se verá que no se tiene una senoide perfecta. Esto se produce por la presencia de armónicos presentes cuando se toca una cuerda de la guitarra.

Antes de empezar a plantear el método por el cual se va a abordar este proyecto, es muy importante observar el comportamiento de una onda real, con sus armónicos correspondientes. Para ello, se realizaron una serie de mediciones en el osciloscopio para ver la onda que recogía el micrófono. Los resultados obtenidos fueron los siguientes:

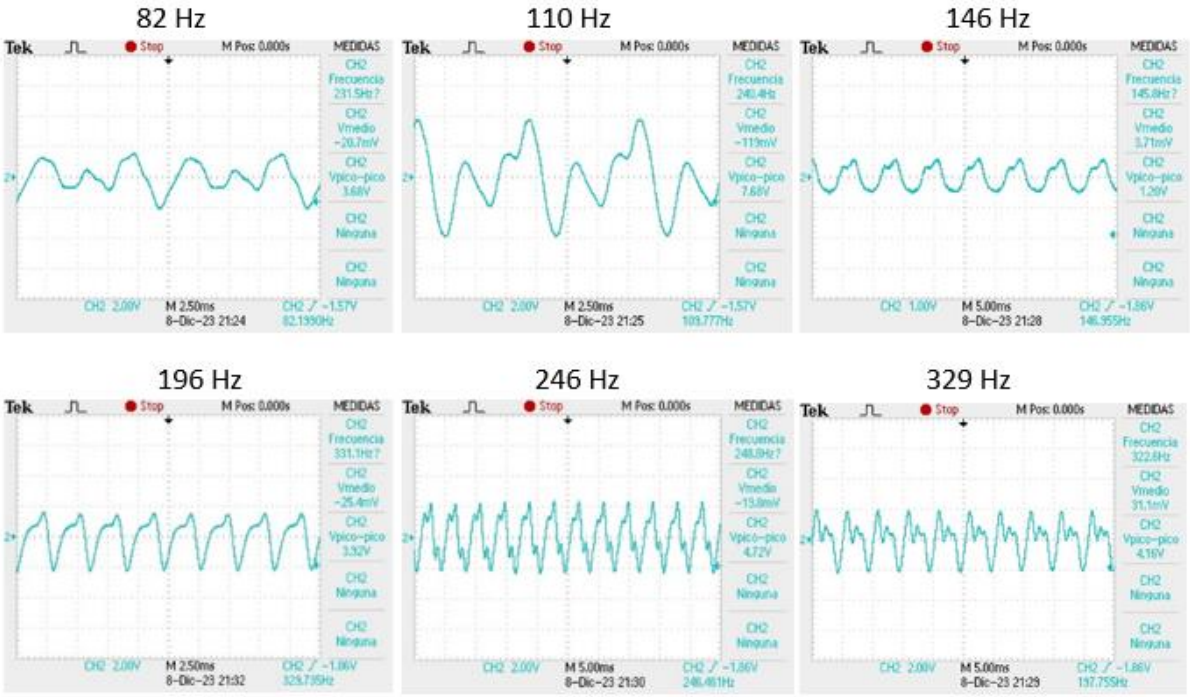

Ilustración 10 Formas de onda de las cuerdas

Como se puede observar, las ondas no son senoidales, pero sí periódicas. Esto ocurre, como se ha comentado antes, por la presencia de armónicos en su sonido. Teniendo esta información, podremos tratar de averiguar qué método encaja más con lo que se quiere obtener.

3.3 Métodos de procesamiento de señal

Este trabajo se centra en la detección de la frecuencia de un sonido. Para poder llevar a cabo el procesamiento de esta señal, se van a explicar cada uno de los métodos que se probarán y los principios matemáticos en los que se basan cada uno de ellos.

3.3.1 Método del cruce por cero

El método del cruce por cero implica la identificación de los puntos en los que una señal periódica cruza el eje horizontal (eje de las abscisas) durante un período de tiempo específico. Estos puntos de cruce representan cambios de polaridad en la señal, lo que indica un ciclo completo de la onda.

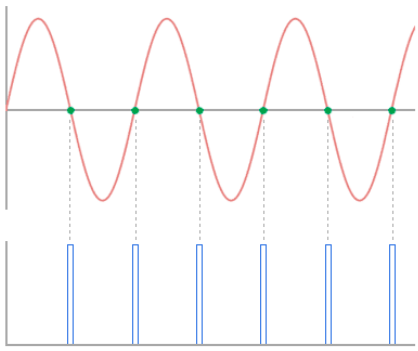

Ilustración 11 Método del cruce por cero

Aplicando este método a nuestro afinador, se sabe que cuando el programa detecte que la señal ha cruzado por el eje x, significa que ya tenemos media onda, es decir, ha pasado medio periodo. Esto quiere decir, que, si se quiere obtener directamente el periodo de la onda completa, se tendrá que ver que la señal cruce por cero 3 veces. Finalmente, el periodo está relacionado con la frecuencia mediante la siguiente ecuación:

$$F = \frac{1}{T}$$

siendo F la frecuencia que se quiere obtener y T el periodo de la onda.

3.3.2 Método de la transformada rápida de Fourier

La Transformada Rápida de Fourier (FFT) es un algoritmo eficiente para calcular la Transformada Discreta de Fourier (DFT). La DFT es una técnica matemática que convierte una señal de su dominio del tiempo (o del espacio) a su dominio de frecuencias.

La DFT de una señal discreta x(n) de longitud N se define como:

$$X(k) = \sum_{n=0}^{N-1} x(n) * e^{-i*2\pi*k*n/N}$$

Donde X(k) es el valor transformado en la frecuencia y k el índice de la frecuencia que varía de 0 a N-1.

La FFT es un algoritmo que reduce significativamente la complejidad computacional de la DFT, lo cual es mucho más manejable para grandes cantidades de datos. La FFT aprovecha la simetría y periodicidad de los factores exponenciales en la fórmula de la DFT para dividir el problema en subproblemas más pequeños.

Para ilustrar la FFT, se supone una señal de audio representada por una serie de muestras discretas. Aplicando la FFT a estas muestras, se puede obtener un espectro de frecuencias indica qué frecuencias están presentes en la señal y con qué amplitudes.

3.3.3 Algoritmo de Goertzel

Para detectar la frecuencia de una señal se puede abordar de varias maneras. El método primero que se ha visto, a partir de la medición de tiempo que pasa entre un cruce y otro se podía averiguar el valor numérico de la frecuencia. Este nuevo algoritmo se basa en otra filosofía y, en este contexto también puede encajar para el cálculo de frecuencia. Más concretamente, este algoritmo es un método eficiente para detectar una o varias frecuencias específicas en una señal. En una guitarra se saben las frecuencias a las que deben sonar las cuerdas. Gracias a esto, el algoritmo se enfoca en estas frecuencias, lo que lo hace más eficiente el procesamiento de la señal. El objetivo principal del algoritmo de Goertzel es identificar la presencia y la amplitud de frecuencias específicas dentro de una señal de entrada. Este algoritmo se fundamenta en una relación de recurrencia que permite transformar la señal de entrada, destacando las frecuencias de interés. A diferencia de la Transformada Rápida de Fourier (FFT), que calcula todas las frecuencias en un rango determinado, el algoritmo de Goertzel se enfoca en calcular solo una o un conjunto pequeño de frecuencias. Para abordar este método se debe aplicar el siguiente algoritmo:

1. Cálculo del índice de frecuencia k: se calcula el índice de la frecuencia k usando la siguiente fórmula.

$$k = \frac{N * f_k}{f_s}$$

   Siendo N el número de muestras, f_s la tasa de muestreo y f_k es la frecuencia específica que se trata de encontrar.

2. Cálculo de w_k: se calcula el valor angular w_k para el índice k.

$$w_k = \frac{2\pi k}{N}$$

3. Cálculo de los coeficientes del algoritmo: mediante w_k, se calculan el seno y el coseno.

$$seno = \sin(w_k)$$

$$coseno = cos(w_k)$$

El coeficiente utilizado en la fórmula recursiva es:

$$coeff = 2 * \cos(w_k)$$

4. Proceso recursivo: se usa una ecuación recursiva del algoritmo de Goertzel para acumular la energía en la frecuencia objetivo.

$$s(n) = x(n) + a_k * s(n-1) - s(n-2)$$

Donde x(n) es la n-ésima muestra de la señal y s(n) el estado interno de la ecuación recursiva.

5. Magnitud de la frecuencia: una vez procesadas todas las muestras, los valores de s(n) se utilizan para calcular los componentes real e imaginario del resultado:

$$real = s(n-1) - s(n-2)\cos(w_k)$$

$$imag = s(n-2)\sin(w_k)$$

La magnitud de la frecuencia detectada se calcula como:

$$magnitud = \sqrt{real^2 - imag^2}$$

Esta magnitud indica la fuerza o presencia de la frecuencia f_k en la señal analizada.

4 HARDWARE

En este capítulo se va a describir las partes de las que consta el afinador. Se hará una breve descripción de los componentes y por qué se han elegido estos. Por otro lado, se verá cómo se han hecho las conexiones entre ellos y su respectivo montaje.

4.1 Implementación del prototipo

Para visualizar el conjunto del proyecto, se ha realizado un esquema de conexionado. Así, se podrá ver los componentes usados y a los periféricos que van conectados de una forma clara y concisa:

Ilustración 12 Esquema de conexionado de los componentes

4.2 Elección de componentes

4.2.1 Microprocesador TMS320F28335

El microprocesador que se va a emplear es el "DSP TMS320F28335" de Texas Instruments. La elección de este microprocesador radica en su alta frecuencia de reloj y su capacidad de procesamiento, pues puede manejar tareas complejas en tiempo real, lo que es crucial para este proyecto. Además, integra una serie de periféricos que permite diseñar sistemas complejos sin necesidad de añadir componentes externos, reduciendo así el costo y la complejidad del diseño. Entre ellos, para esto proyecto hemos usado los siguientes periféricos:

- GPIO (General-Purpose Input Output): canales digitales de entrada/salida.
- ADC (Analog to Digital Converter): conversión de señales analógicas de entrada.
- SCI (Serial Communication Interface): Puerto serie de entrada/salida para comunicación asíncrona.
- Interrupt System: eventos de interrupción software o hardware al programa principal.

Ilustración 13 Microprocesador TMS320F28335

4.2.2 Micrófono amplificador

El micrófono que se va a utilizar en el "Analog sound sensor v2.2". Este sensor de sonido se utiliza normalmente para detectar el volumen en el ambiente. En este caso, va a ser el encargado de recoger el sonido de las cuerdas de la guitarra. Sus características son las siguientes:

- Voltaje de alimentación: 3,3 V a 5 V.
- Interfaz: Analógica.
- Tamaño: 22x32 mm (0,87 x 1,26 pulgadas).

La elección de este componente es crucial para un correcto funcionamiento del dispositivo. En un principio, se trató de emplear otro micrófono el cual no llego a ser utilizado, ya que al medir la señal de salida en un osciloscopio se vio que era muy poco sensible al sonido de la guitarra.

Ilustración 14 Analog sound sensor v2.2

4.2.3 Módulo Bluetooth

El módulo Bluetooth usado es el "ZS-040". A través de este dispositivo, se puede conectar el dispositivo móvil al afinador para recibir y transmitir información e indicar que cuerda se va a afinar en el momento. Finalmente, se recibirá la frecuencia que se está tocando. Este módulo tiene seis pines, de los cuales solo utilizaremos cuatro de ellos:

- RX: leemos la información que nos llega del microprocesador.
- TX: transmitimos la información hacia el microprocesador.
- GND: tierra del módulo.
- VCC: alimentación a 5V.

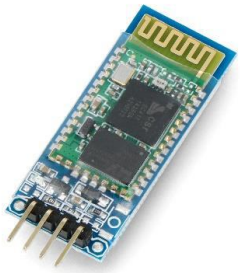

Ilustración 15 Modulo Bluetooth ZS-040

4.2.4 Pantalla LCD

También se implementará una pantalla, que será de gran ayuda durante la afinación ya que estará indicando continuamente la frecuencia en la que suenan las cuerdas de la guitarra. El modelo que se utilizará es una pantalla LCD que es la "HD44780". Cada uno de sus pines tiene la siguiente función:

- VSS: GND
- VDD: alimentación a 5V.
- VO: debe ir conectado a un potenciómetro que nos permite ajustar el brillo de la pantalla.
- RS: pin de control para indicar si estamos enviando un comando o una cadena de caracteres.
- RW: GND.
- E: enable.
- D1, D2, …D8: es un bus de 8 datos.
- A: alimentación a 5V.
- K: GND.

La razón por la que se quiso implementar una pantalla fue para facilitar el proceso de afinación en tiempo real. Este componente estará actualizando constantemente la afinación procesada. De esta manera, se sabrá de qué manera se tendrá que proceder para obtener una correcta afinación.

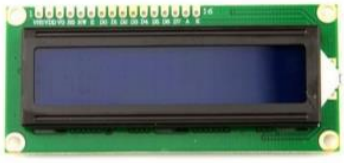

Ilustración 16 Pantalla LCD

4.3 Montaje del prototipo

Una vez seleccionados todos los componentes del afinador, se deben realizar las conexiones entre ellas, eligiendo los pines correspondientes.

4.3.1 Micrófono amplificador

Al obtener una señal analógica de este dispositivo, se necesita convertirla a una señal digital para que pueda ser procesada por el programa. Para ello, el microprocesador tiene 16 canales ADC con resolución de 12 bits, del cual solo se hará uso de un canal, ya que el micrófono solo tiene una salida. El canal que se va a usar será el pin A0.

4.3.2 Luces LEDS

Para la configuración de los leds se han elegido 6 pines pertenecientes al GPIO:

Tabla 1 Asignación de los pines con los leds

LED	PIN DSP
Cuerda 1	GPIO 17
Cuerda 2	GPIO 19
Cuerda 3	GPIO 21
Cuerda 4	GPIO 23
Cuerda 5	GPIO 27
Cuerda 6	GPIO 31

4.3.3 Módulo Bluetooth

El ZS-040 tiene los pines RX y TX que se encargan de la transmisión y lectura de los datos intercambiados con el microprocesador. Para este proyecto se van a usar los pines 28 y 29 del GPIO. El pin 28 es el de lectura (RX) del micro por lo que tendrá que ir conectado al de transmisión del módulo. Por otro lado, el pin 29 es el de transmisión (TX), que se conectará al de lectura del módulo.

Tabla 2 Asignación de pines al módulo bluetooth

Módulo BTH	PIN DSP
RX	GPIO 29
TX	GPIO 28

4.3.4 Pantalla LCD

El control de la pantalla LCD HD44780 se realiza a través del GPIO. Para ello, se debe de hacer la configuración necesaria de los pines para que se transmita la información desde el microprocesador y viceversa de forma correcta. Como antes se ha visto, esta pantalla utiliza un bus de 8 datos, pero para este proyecto basta con uno de 4. Por lo tanto, la configuración final de los pines quedaría de la siguiente manera:

Tabla 3 Asignación de pines para la pantalla

PANTALLA LCD	PIN DSP
D4	GPIO 7
D5	GPIO 9
D6	GPIO 11
D7	GPIO 49
E	GPIO 61
RS	GPIO 63

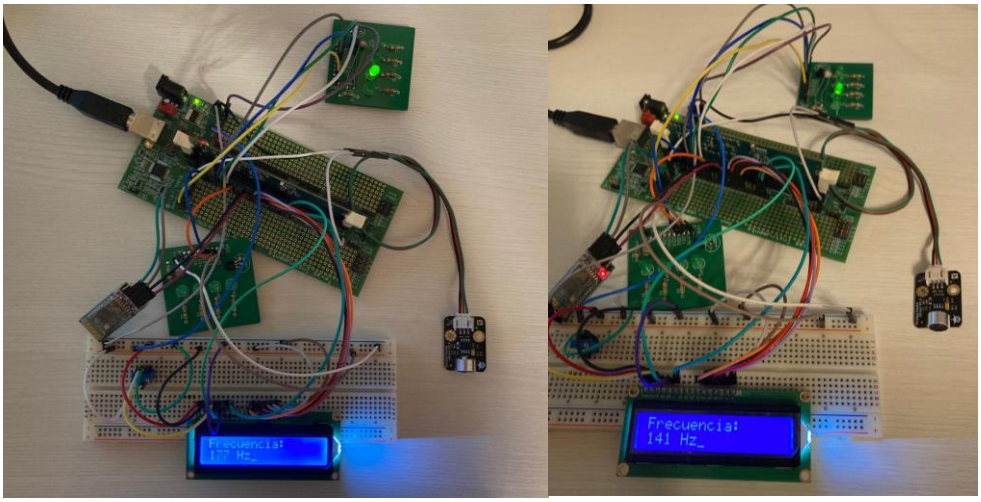

Ilustración 17 Montaje del afinador

5 SOFTWARE

Este capítulo va a dedicarse para mostrar el interior de nuestro afinador, cómo funciona realmente y cómo se procesa la señal. Además, se va a tratar de explicar el código del programa, de forma que se pueda entender de qué manera está estructurado y montado el programa.

5.1 Descripción de la aplicación

Aunque gracias a este dispositivo el proceso de afinación de nuestro instrumento se haga más intuitivo, se ha visto necesario hacer un breve resumen de cómo se tendría que usar el afinador para sintonizarlo de la manera correcta y obtener los resultados correctos. Para ello, se han implementado dos modos de funcionamiento que se explicarán en los próximos apartados.

Ilustración 18 Mensaje de inicio mostrado en la pantalla

En el momento en el que se inicia el programa, a través de la comunicación por bluetooth, se tendrá la oportunidad de elegir el método que se quiere usar para afinar la guitarra. El programa enviará un mensaje describiendo que carácter se tiene que enviar para seleccionar el método deseado. Si introduce, afinará con el método del cruce por cero. Si por el contrario introduce un 2, afinará con el algoritmo de Goertzel.

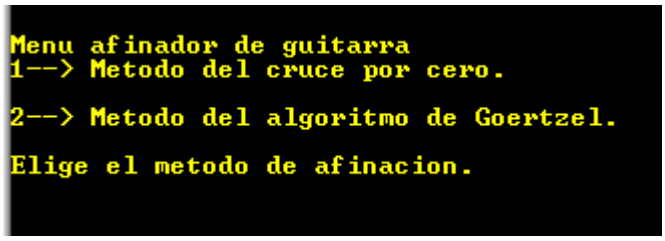

Ilustración 19 Menú del afinador transmitido por periférico SCI

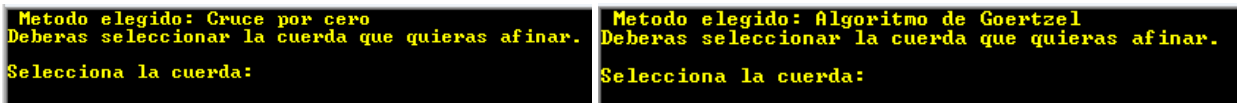

Ilustración 20 Interfaz con usuario al seleccionar un método

Si más adelante se quiere cambiar el método de procesamiento de señal basta con enviar un número que no se encuentre en el intervalo de 0 a 6, incluidos estos dos números.

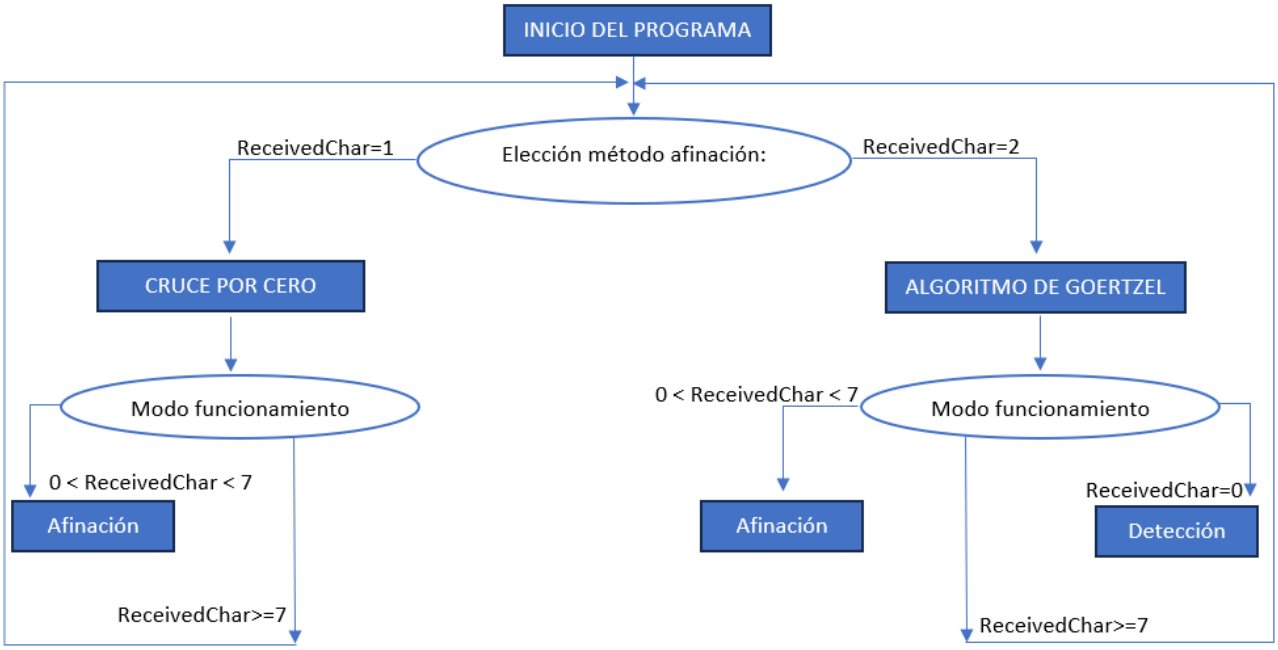

Ilustración 21 Esquema de funcionamiento del programa

5.1.1 Modos de funcionamiento

Una vez elegido el método por el que se quiera afinar, el programa preguntará que cuerda se quiere afinar. Dependiendo del valor que se introduzca para la afinación de la guitarra, el modo de funcionamiento será distinto. Si, por un lado, se introduce que la cuerda que se quiere afinar es la número cero, quiere decir que el programa entrará en el modo detección. Si, por otro lado, se introduce que la cuerda que se quiere afinar es cualquiera desde la 1 hasta la 6, el programa entrara en el modo de afinación. Si, por el contrario, se introduce otro valor distinto a los mencionados anteriormente, el programa dejará de procesar la señal de entrada con el método (cruce por cero o Goertzel) que se indicó al principio, pudiendo cambiarlo a otro.

5.1.1.1 Afinación

Si se indica que se quiere afinar cualquiera de las cuerdas, introduciendo un valor entre 1 y 6, el programa entrará en el modo de afinación. Este modo indicará por la pantalla la frecuencia detectada y la acción a realizar sobre la cuerda para afinarla que será aflojar, tensar o afinado.

Si por ejemplo se quiere afinar la primera cuerda, la más grave de todas, se insertaría en el dispositivo móvil que está conectado al afinador el número uno. Luego, se mostrará un mensaje que indica que esta cuerda debe sonar a una frecuencia de 82 Hz, y la frecuencia a la que está sonando en ese momento.

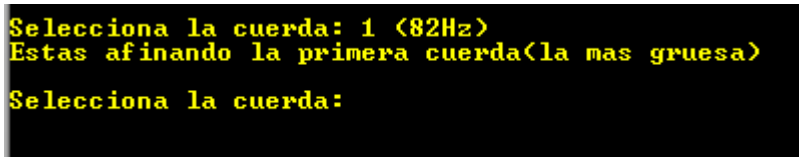

Ilustración 22 Interfaz con usuario al seleccionar una cuerda

Imagine que la pantalla esté mostrando una frecuencia de 79 Hz. En este caso, se debe de tensar la cuerda, es decir, hay que girar la clavija hacia la izquierda. Hasta que no se vea que la frecuencia mostrada en la pantalla sea de 82 Hz, se debe seguir girando la clavija.

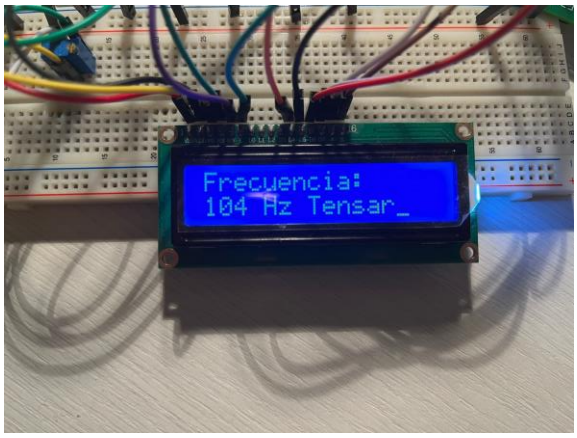

Ilustración 23 Mensajes mostrados en pantalla con el modo de afinación

En el caso contrario, en el que la pantalla estuviera mostrando una frecuencia de 85 Hz, habría que hacer lo contrario. En este caso, se debe de aflojar la cuerda, es decir, hay que girar la clavija hacia la derecha. Hasta que no se vea que la frecuencia mostrada en la pantalla sea de 82 Hz, se debe seguir girando la clavija.

Finalmente, si se repite este mismo proceso con el resto de las cuerdas, se conseguiría la afinación completa del instrumento. Es un proceso, el cual si nunca se has realizado antes quizás se tarde un poco más, pero una vez se hayas hecho más de tres veces, se convierte en algo totalmente rutinario y sencillo.

5.1.1.2 Detección

Si se indica que se quiere afinar la cuerda 0, el programa entrará en el modo de detección. Este modo de funcionamiento solo está disponible si se ha elegido el algoritmo de Goertzel como método de afinación. La funcionalidad de este modo sirve para detectar la cuerda que se está tocando. La pantalla indicará la cuerda tocada, y la frecuencia a la que está sonando. Una de las razones por las que se ha implementado este método es para obtener una afinación más rápida de la guitarra. Al no tener que introducir qué cuerda se quiere afinar, el proceso es mucho más sencillo y práctico.

Ilustración 24 Mensajes mostrados en pantalla con el modo de detección

En ocasiones, este modo también puede ser bastante útil para casos en los que las cuerdas de las guitarras más desafinadas de lo normal. Esto puede ocurrir cuando la guitarra lleva mucho tiempo sin usarse o afinarse. Un

ejemplo puede ser cuando se toca la segunda cuerda de la guitarra, y esta se ha aflojado tanto que suena a la frecuencia de la primera cuerda. Al indicarse que se ha detectado la primera cuerda, en lugar de la segunda, el usuario tiene una visión más clara de que el número de vueltas que va a girar la clavija será mayor al caso en el que se detectase de forma correcta la cuerda tocada.

5.2 Procesamiento de señal

5.2.1 Implementación del método del cruce por cero

Anteriormente se explicó el concepto del cruce por cero, y en qué consistía. En este apartado, lo que se estudiará será la implementación de este método, cómo se ha conseguido pasar de la teoría a la realidad. Se observará que, en este código, la esencia de lo que se explicó está reflejada, aunque con algunos ajustes que tuvieron que realizarse debido a que es imposible trabajar en un mundo ideal.

Por recapitular un poco, este método consiste en detectar el momento en el que la señal tome un valor nulo. Justo en ese momento se empezará a medir y esperar a detectar otro cruce por cero. Como la onda es senoidal la segunda vez que se detecte un cruce por cero, se sabrá que ha pasado medio periodo de la onda. Como el convertidor no puede dar valores negativos y casi nunca va a llegar a mostrar valores nulos por ruido ambiental existente, se establecerá otro nivel de cruce por 0. En este caso, tras realizar varias pruebas, se estableció que 150 era el valor más adecuado. Sabiendo todo esto, cada vez que se detecte que el valor binario pasa de ser mayor que 150 a menor que este nivel establecido, se comienza a contar el tiempo que transcurre. Cuando se vuelva a detectar el mismo evento, se calculará la frecuencia y se pondrán a cero las variables utilizadas para medir el tiempo.

CÓDIGO

```
if(Vbin <= 150 && Vbin_ant > 150){
    contador++;
    start = 1;
}


Vbin_ant = Vbin;

if(start==1){
    if(contador == 2){ //cuando haya contado dos máximos, es decir, un periodo, calcula frecuencia
        frecuencia=pow(t,-1);
        contador=0;
        start = 0;
        t=0;
        tint=0;
    }
}
```

Para comprender mejor su funcionamiento, se ha realizado un diagrama donde se puede ver más claro:

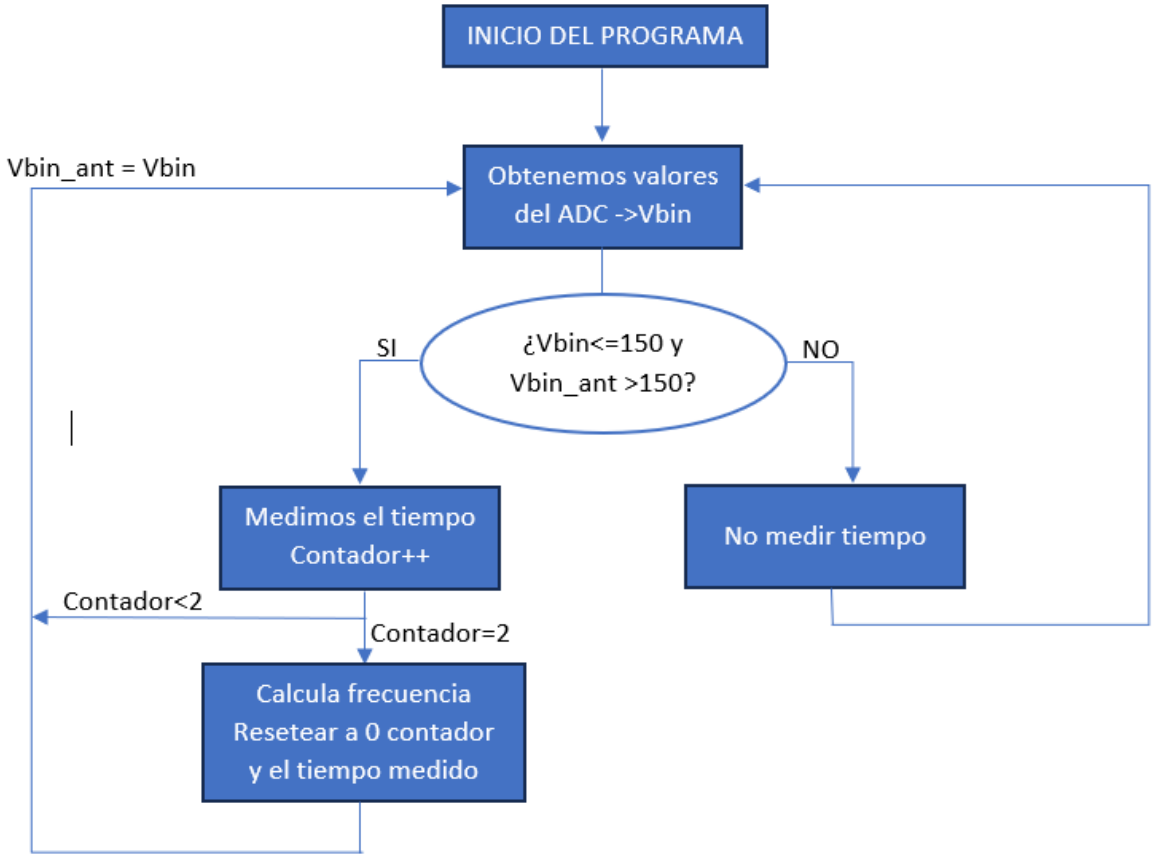

Ilustración 25 Esquema del funcionamiento del cruce por cero

Como con este método se obtienen muchas interferencias por culpa de los armónicos que suenan cuando se toca una cuerda, se ha implementado una serie de filtros, ya que se sabe que las frecuencias a las que sonarán las cuerdas de la guitarra se encontrarán en el intervalo de 50 y 350 Hz:

CÓDIGO

```
if(cuerda==0){
    if(frecuencia>500 || frecuencia<50){
        frecuencia=frecuencia_ant;
    }
}

else if(cuerda==1){
    if(frecuencia>110 || frecuencia<50){
        frecuencia=frecuencia_ant;
    }
}

else if(cuerda == 2 || cuerda == 3){
    if(frecuencia>180 || frecuencia<98){
        frecuencia=frecuencia_ant;
    }
}
else if(cuerda==4 || cuerda == 5 || cuerda == 6){
    if(frecuencia>350 || frecuencia<150){
        frecuencia=frecuencia_ant;
    }
}
```

Finalmente, para conseguir un valor más fiable se ha implementado un código que realice una media aritmética

de las frecuencias que se han ido calculando en un breve espacio de tiempo. Cuando esta media llegue a un número de muestras determinado, se pondrá a cero y volverá a hacer otra media.

CÓDIGO

```
if(frecuencia!=0){
    frec++;
    sumfrec=frecuencia+sumfrec;
    frecmed=sumfrec/frec;
}
if(frec>=3000){
    frec=0;
    sumfrec=0;
}
```

5.2.2 Implementación del algoritmo de Goertzel

Para empezar, se van a almacenar las muestras obtenidas del micrófono en un vector. Hasta que este vector no esté lleno por completo, no se empezará a procesar la señal mediante el algoritmo.

CÓDIGO

```
if (sample_index < N) {
    Vbin = AdcMirror.ADCRESULT0;       // Almacenaje de resultados medidos
    samples[sample_index++] = (double)(Vbin - ADC_MID_VALUE); // Center the value around 0

    if (sample_index >= N) {
        buffer_full = 1;
        sample_index = 0;
    }
}
```

Cuando se activa la señal del vector completo, se llama a la función llamada analyze_frecuencies, en el que se le pasa las frecuencias objetivo que se están buscando en la señal.

CÓDIGO

```
void analyze_frequencies(double *target)
{
    double cosine, sine, coeff;
    int j;
    for (j = 0; j < NUM_FREQS; j++) {
        int k = (int)(0.5 + ((N * target[j]) / SAMPLE_RATE));  // Target frequency index
        goertzel_init(&cosine, &sine, &coeff, k);
        magnitudes[j] = goertzel_filter(samples, coeff, cosine, sine);
    }
}
```

Esta función calcula el índice de frecuencia k para cada frecuencia objetivo. A continuación, se ejecutan dos funciones más para llevar a cabo el cálculo. Estas dos funciones reflejan las ecuaciones que fueron explicadas

en el apartado 3 de fundamentos teóricos.

CÓDIGO

```
void goertzel_init(double *cosine, double *sine, double *coeff, int k)
{
    double omega = (2.0 * M_PI * k) / N;
    *cosine = cos(omega);
    *sine = sin(omega);
    *coeff = 2.0 * (*cosine);
}

double goertzel_filter(double *samples, double coeff, double cosine, double sine)
{
    double s_prev = 0.0;
    double s_prev2 = 0.0;
    double s;

    for (i = 0; i < N; i++) {
        s = samples[i] + coeff * s_prev - s_prev2;
        s_prev2 = s_prev;
        s_prev = s;
    }

    double real_part = s_prev - cosine * s_prev2;
    double imag_part = sine * s_prev2;
    return sqrt(real_part * real_part + imag_part * imag_part);
}
```

Al final del proceso, se obtiene un vector con las magnitudes de cada frecuencia objetivo. Este valor indica la fuerza o presencia de cada frecuencia. Por lo tanto, el valor mayor de este vector indicará la frecuencia que está sonando.

CÓDIGO

```
max_magnitude = 0.0;

int i;
for (i = 0; i < NUM_FREQS; i++) {
    if (magnitudes[i] > max_magnitude) {
        max_magnitude = magnitudes[i];
        max_index = i;
    }
}
}
```

Para entender mejor este proceso, se ha realizado un esquema en el que se enseña los pasos que da el programa para la detección de la frecuencia de la señal.

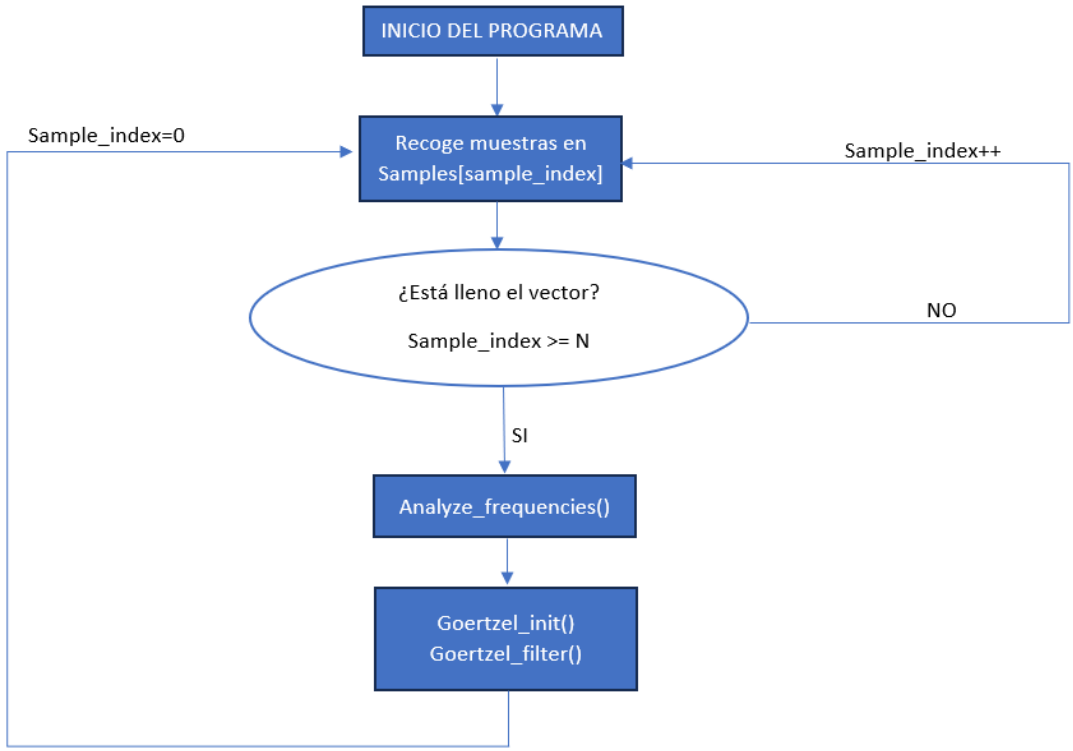

Ilustración 26 Esquema del funcionamiento del algoritmo de Goertzel

5.3 Implementaciones adicionales

5.3.1 ADC

Para la programación del micrófono, es necesario hacer uso de un convertidor analógico digital para poder convertir la información de una señal analógica a una digital y así poder procesar la información. A continuación, se muestra la inicialización del periférico:

CODIGO

```
void InitAdcRegs(void)
{
    AdcRegs.ADCTRL1.all = 0;
    AdcRegs.ADCTRL1.bit.ACQ_PS = 1;          // Sample window = 2*(1/ADCCLK)
    AdcRegs.ADCTRL1.bit.SEQ_CASC = 0;        // Modo dual
    AdcRegs.ADCTRL1.bit.CPS = 1;             // CPS = 2 --> ADCCLK = FCLK/(CPS+1)
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0;        // Modo start-stop

    AdcRegs.ADCTRL2.all = 0;
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;    // Se habilita interrupcion SEQ1INT
    AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;    // Modo de interrupcion cada EOS

    AdcRegs.ADCTRL3.bit.SMODE_SEL = 0;       // Modo muestreo secuencial
    AdcRegs.ADCTRL3.bit.ADCCLKPS = 1;        // ADCCLKPS = 3 --> FCLK = HSPCLK / 2 * ADCCLKPS
                                             // HSPCLK = 75 MHz
                                             // FCLK = 37.5 MHz
                                             // ADCCLK = 12.5 MHz

    // Una sola medida
    AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 0;    // 1 conversion por secuencia

    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;     // CONVV00 = ACDINA0.

}
```

Este ADC tiene una resolución de 12 bits por lo que se van a recibir valores de 0 a 4095. En este caso, se han visto a través de mediciones con el osciloscopio, que el micrófono recoge valores negativos además de positivos. Esto se reflejará en el ADC de manera que los valores negativos serán valores nulos y los positivos se convertirán en valores en el rango de 0 a 4095.

5.3.1.1 Configuración ADC con el método del cruce por cero

En el caso del método del cruce por cero, la frecuencia de muestreo utilizada será distinta al otro método seguido. Para obtener los valores convertidos se debe de muestrear a una frecuencia mayor que las frecuencias que tienen las cuerdas de la guitarra, para así obtener varias muestras en un solo periodo. Si, por ejemplo, la cuerda que tiene mayor frecuencia es la de 329Hz y se quiere obtener 100 muestras por periodo, se debe de muestrear cada 30 microsegundos. Para hallar esto se ha aplicado la siguiente fórmula:

$$Periodo_{cuerda} = nºmuestras * Periodo_{muestreo}$$

De esta manera, si se muestrea a la misma velocidad con el resto de las cuerdas, se obtendrán más muestras por periodo, llegando a tener un número máximo de muestras con la cuerda de 82 Hz (406 muestras). Para configurar que el ADC haga el muestreo a la frecuencia requerida, se hará uso del periférico del timer. Cada vez que se active el timer, este despertará al convertidor y hará una medida, que se guardará en una variable flotante, tal y como se ve a continuación:

CÓDIGO

```
interrupt void  adc_SEQ1_isr(void)
{
    Vbin = AdcMirror.ADCRESULT0;       // Almacenaje de resultados medidos

    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;       // Reset SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;     // Clear INT_SEQ1 para habilitar la siguiente interrupcion
    PieCtrlRegs.PIEACK.bit.ACK1 = 1;        // Clear PEACK  habilitar siguiente interrupcion
}
```

5.3.1.2 Configuración ADC con el algoritmo de Goertzel

En el caso del algoritmo de Goertzel, la frecuencia de muestreo que se usa es distinta debido a la manera en la que calcula la frecuencia. Cuando se ha explicado el funcionamiento de este algoritmo varios apartados atrás, se explicaba que antes empezar a procesar los datos, estos se van almacenando en un vector hasta que se llena. Una vez que este vector esté completo entonces es cuando se empieza a procesar la señal. Si se muestreara a la frecuencia que se usó para el otro método, este vector se llenaría mucho más rápido, y por lo tanto se procesaría una porción muy pequeña de la señal. Es por esto por lo que se usa una frecuencia más lenta, para que en un mismo vector se almacenen una serie de puntos con los que el algoritmo sea capaz de identificar a que frecuencia corresponde.

Por lo tanto, a través de experimentación y tratando de llegar a un equilibrio entre precisión y fiabilidad se ha escogido hacer un muestreo a una frecuencia de 2050 Hz. La función del ADC implementado quedaría de la siguiente manera:

CÓDIGO

```
__attribute__((ramfunc))
interrupt void  adc_SEQ1_isr(void)
{

    if (sample_index < N) {
        Vbin = AdcMirror.ADCRESULT0;       // Almacenaje de resultados medidos
        samples[sample_index++] = (double)(Vbin - ADC_MID_VALUE); // Center the value around 0

        if (sample_index >= N) {
            buffer_full = 1;
            sample_index = 0;
        }
    }

    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;       // Reset SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;     // Clear INT_SEQ1 para habilitar la siguiente interrupcion
    PieCtrlRegs.PIEACK.bit.ACK1 = 1;        // Clear PEACK  habilitar siguiente interrupcion
}
```

5.3.2 Timer

La implementación de este periférico es una de los más importantes para el funcionamiento del dispositivo. La función de este periférico consiste en la activación periódica de una función (normalmente conocido como interrupciones), la cual realizará las funciones que se le indiquen. Para configurar este periférico se le indicará el periodo con el que se quiera que actúen estas interrupciones. Dependiendo del método que se esté implementando, se usará una frecuencia u otra de muestreo. Para saber cómo se han elegido estas frecuencias, ya fueron explicadas en el apartado de la implementación del ADC.

En concreto, la función que ha empeñado el periférico del timer en este trabajo ha sido para que active el convertidor y recoja muestras. Este código ha sido común para ambos métodos. Solo para el caso del cruce por cero se implementó una pequeña función para medir el tiempo transcurrido cada vez que se detectaba un “cruce por cero”.

CÓDIGO

```
interrupt void cpu_timer0_isr(void)
{

    // cuando se activa el timer, empieza a convertir el CAD
    AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;       // Activacion de conversion ADC por software

    PieCtrlRegs.PIEACK.bit.ACK1 = 1;        // Clear PIEACK para habilitar siguiente interrupcion
}
```

5.3.3 GPIO

5.3.3.1 Leds

Se harán uso de los GPIO para encender o apagar los leds según la cuerda que se esté tocando. Para ello, primero se tiene que configurar los pines e indicarse que se van a usar como GPIO y que van a ser de salida:

CÓDIGO

```
void Gpio_select(void)
{
    EALLOW;
    //CUERDA 1
    GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0;    // GPIO17 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO17 = 1;     // GPIO17 como salida: LED
    //CUERDA 2
    GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 0;    // GPIO19 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO19 = 1;     // GPIO19 como salida: LED
    //CUERDA 3
    GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0;    // GPIO21 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO21 = 1;     // GPIO21 como salida: LED
    //CUERDA 4
    GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;    // GPIO23 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO23 = 1;     // GPIO23 como salida: LED
    //CUERDA 5
    GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 0;    // GPIO29 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO27 = 1;     // GPIO29 como salida: LED
    //CUERDA 6
    GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;    // GPIO31 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;     // GPIO31 como salida: LED

    EDIS;

}
```

Luego, en el main se tendrá que escribir la siguiente estructura para encender o apagar los leds según la cuerda elegida. Por ejemplo, este código indica que se está afinando la cuerda 1.

CÓDIGO

```
    GpioDataRegs.GPASET.bit.GPIO17 = 1;    //enciende led conectado al pin 17
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;  //apaga led conectado al pin 19
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;  //apaga led conectado al pin 21
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;  //apaga led conectado al pin 23
    GpioDataRegs.GPACLEAR.bit.GPIO29 = 1;  //apaga led conectado al pin 29
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;  //apaga led conectado al pin 31
```

5.3.3.2 Pantalla LCD

Para la implementación de la pantalla también se hace a través del GPIO. Para hacer el proceso mucho más fácil, se han hecho las inicializaciones de los pines y las funciones que controlan el LCD en otros archivos distintos, que luego los incluimos como includes en la cabecera de nuestro programa principal.

La función de configuración de los pines se llama lcd_ioinit(), y la función que indica si estamos enviando un comando o una cadena de caracteres se llama lcd_ioset y son los siguientes:

CODIGO

```
uint16_t lcd_ioinit(void)
{
    uint16_t i = 0;

    EALLOW;
    // Configure control lines as outputs
    GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO61 = 1;

    GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO63 = 1;

    // Configure bus data lines as outputs
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO7 = 1;

    GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;

    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;

    GpioCtrlRegs.GPBMUX2.bit.GPIO49 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;

    EDIS;

    // Set all the pins to "low" state
    for (i = 0; i < 11; i++)
        lcd_ioset(i, FALSE);

    // Return bus length
    return 4;

}
```

```
void lcd_ioset(enum enLCDControlPins pin, uint16_t out)
{
    switch (pin) {
    case E_D4_PIN:
        // si out es distinto de False, se pone el pin
        if(out!=0)
            GpioDataRegs.GPASET.bit.GPIO7=1;
        else
            GpioDataRegs.GPACLEAR.bit.GPIO7=1;
        break;
    case E_D5_PIN:
        if(out!=0)
            GpioDataRegs.GPASET.bit.GPIO9=1;
        else
            GpioDataRegs.GPACLEAR.bit.GPIO9=1;
        break;
    case E_D6_PIN:
        if(out!=0)
            GpioDataRegs.GPASET.bit.GPIO11=1;
        else
            GpioDataRegs.GPACLEAR.bit.GPIO11=1;
        break;
    case E_D7_PIN:
        if(out!=0)
            GpioDataRegs.GPBSET.bit.GPIO49=1;
        else
            GpioDataRegs.GPBCLEAR.bit.GPIO49=1;
        break;
    case E_RS_PIN:
        if(out!=0)
            GpioDataRegs.GPBSET.bit.GPIO63=1;
        else
            GpioDataRegs.GPBCLEAR.bit.GPIO63=1;
        break;
    case E_EN_PIN:
        if(out!=0)
            GpioDataRegs.GPBSET.bit.GPIO61=1;
        else
            GpioDataRegs.GPBCLEAR.bit.GPIO61=1;
        break;
    }
}
```

Una vez hecho esto, lo que hay que saber para controlar la pantalla son varias funciones básicas. Primero se tiene lcd_init que es donde se inicializa la pantalla y se indica el número de filas y columnas que se quieren usar.

Antes de empezar a escribir en la pantalla, se debe de activar el cursor con la función lcd_cursor. Con ella, se puede encender, apagar o hacer que parpadee el cursor.

Otra función muy útil es lcd_home, que moverá el cursor al principio de la pantalla sin afectar al contenido que haya escrito en ella.

Una función imprescindible es lcd_on que encenderá la pantalla y mostrará lo que se ha escrito en ella.

Para escribir en la pantalla una cadena de caracteres se usará la función lcd_puts. Además, si se quiere cambiar de línea donde se está escribiendo, se utilizará la función lcd_goto que mueve el cursor a la posición de la pantalla que se le indique.

Por último, otra función muy importante que también se va a emplear es lcd_clear, que como su nombre indica, borrará todo lo que esté escrito en la pantalla.

Finalmente, en el código principal se tendrá lo siguiente:

CÓDIGO

```
//PANTALLA
// Initialize lcd driver
lcd_init(0, 16, 2);
lcd_cursor(E_LCD_CURSOR_ON);
lcd_home();
lcd_on();
// Display message on the first row
lcd_puts("AFINADOR DE");
lcd_goto(0,1);
lcd_puts("  GUITARRA");
```

Esta parte del código solo se ejecutará una vez, y sirve para inicializar la pantalla. Cuando se inicie el programa, en la pantalla aparecerá durante un tiempo el mensaje de "AFINADOR DE GUITARRA".

Más adelante, en el bucle while se ha implementado el código que va mostrando la frecuencia cada segundo.

CÓDIGO

```
//PANTALLA LCD
lcd_clear();
lcd_puts("Frecuencia:");
lcd_goto(0,1);
sprintf(pantalla,"%3d", (int)frecmed);
lcd_puts(pantalla);
lcd_puts(" Hz");
```

5.3.4 SCI

A través del SCI se conseguirá la transmisión de los valores por puerto serie a otro dispositivo e indicar la cuerda que se esté tocando. Primeramente, se inicializará la SCI y se configurará la manera en la que se transmiten los datos. Esto se implementará de la siguiente manera:

CÓDIGO

```
// Inicializacion de la FIFO del modulo SCI
void scia_fifo_init()
{
    SciaRegs.SCIFFTX.all=0xE040;
    SciaRegs.SCIFFRX.all=0x204f;
    SciaRegs.SCIFFCT.all=0x0;
}
```

```
void scia_echoback_init()
{
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function

    SciaRegs.SCICCR.all =0x0007;   // 1 stop bit,  No loopback
                                   // No parity, 8 char bits,
                                   // async mode, idle-line protocol
    SciaRegs.SCICTL1.all =0x0003;  // enable TX, RX, internal SCICLK,
                                   // Disable RX ERR, SLEEP, TXWAKE
    SciaRegs.SCICTL2.all =0x0003;
    SciaRegs.SCICTL2.bit.TXINTENA =1;
    SciaRegs.SCICTL2.bit.RXBKINTENA =1;

    SciaRegs.SCIHBAUD    =0x0001;  // 9600 baud @LSPCLK = 37.5MHz.
    SciaRegs.SCILBAUD    =0x00E7;


    SciaRegs.SCICTL1.all =0x0023;  // Relinquish SCI from Reset
}
```

Estas funciones, y otras más están definidas aparte del programa de forma que para que puedan implementarse en el programa, se han incluido como si fueran otros includes. Esto se ha hecho para que en el programa principal solo esté lo esencial para el funcionamiento del afinador y no fuera tan denso. Todo este material está incluido en los anexos.

Finalmente, en el main, se han reflejado el nombre de todas estas funciones, de manera que se tendrá un menú del afinador de guitarra:

CÓDIGO

```
//MENU DEL BLUETHOOTH

LoopCount = 0; // Para contar cuantos caracteres hemos enviado

scia_fifo_init();      // Inicializamos la FIFO del SCI
scia_echoback_init();  // Inicializamos los parametros del SCI

scia_msg("\e[1;1H\e[2J"); //limpiamos pantalla

msg = "\r\n\n\nMenu afinador de guitarra\0"; // Cadena de bienvenida
// Enviamos toda la cadena //
scia_msg(msg);

msg = "\r\nDeberas seleccionar la cuerda que quieras afinar.\n\0";
scia_msg(msg);

msg = "\r\nSelecciona la cuerda: \0";
scia_msg(msg);
```

Como se puede observar en esta parte del código, se inicializa la pantalla como se ha comentado antes, y se envían los primeros mensajes. Esta parte del código no está metido en un bucle while, por lo que solo se ejecutará una sola vez.

Más adelante, dentro de un bucle while, tenemos la otra parte del SCI implementado. Se ha situado dentro de un while porque el programa tiene que estar constantemente comprobando si se enviado algo por el puerto serie. El mecanismo para que este periférico funcione es que siempre esté buscando que el buffer de lectura esté lleno. Si efectivamente está lleno, entrará en la condición y dependiendo del dato leído se entrará en un caso u otro.

Finalmente se envían los mensajes correspondientes por puerto serie y volvemos a buscar que nuestro buffer esté lleno.

CÓDIGO

```
// Transmisión por puerto serie
        if(SciaRegs.SCIFFRX.bit.RXFFST == 1) {

            scia_msg("\e[1;1H\e[2J");
            msg = "\r\nSelecciona la cuerda: \0";
            scia_msg(msg);

            ReceivedChar = SciaRegs.SCIRXBUF.all;
            scia_xmit(ReceivedChar);

        if(ReceivedChar == '1'){
            scia_msg(" (82Hz)");
            msg = "\r\nEstas afinando la primera cuerda(la mas gruesa)\n\0";
            cuerda=1;

            GpioDataRegs.GPASET.bit.GPIO17 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;

            if(frecmed>=82-tolerancia && frecmed<=82+tolerancia){
                msg2="\rBuena afinacion\n\r\0";
            }
            else if(frecmed>82+tolerancia){
                msg2= "\rAflojar cuerda\n\r\0";
            }
            else if(frecmed<82-tolerancia){
                msg2= "\rTensar cuerda\n\r\0";
            }


        }else if(ReceivedChar == '2'){
            scia_msg(" (110Hz)");
            msg = "\r\nEstas afinando la segunda cuerda \n\0";
            cuerda=2;

            GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
            GpioDataRegs.GPASET.bit.GPIO19 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;

            if(frecmed>=110-tolerancia && frecmed<=110+tolerancia){
                msg2="\rBuena afinacion\n\0";
            }
            else if(frecmed>110+tolerancia){
                msg2= "\rAflojar cuerda\n\0";
            }
            else if(frecmed<110-tolerancia){
                msg2= "\rTensar cuerda\n\0";
            }
```

```
}else if(ReceivedChar == '3'){
    scia_msg(" (146Hz)");
    msg = "\r\nEstas afinando la tercera cuerda \n\0";
    cuerda=3;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPASET.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;

    if(frecmed>=146-tolerancia && frecmed<=146+tolerancia){
        msg2="\rBuena afinacion\n\0";
    }
    else if(frecmed>146+tolerancia){
        msg2= "\rAflojar cuerda\n\0";
    }
    else if(frecmed<146-tolerancia){
        msg2= "\rTensar cuerda\n\0";
    }

}else if(ReceivedChar == '4'){
    scia_msg(" (196Hz)");
    msg = "\r\nEstas afinando la cuarta cuerda \n\0";
    cuerda=4;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPASET.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;

    if(frecmed>=196-tolerancia && frecmed<=196+tolerancia){
        msg2="\rBuena afinacion\n\0";
    }
    else if(frecmed>196+tolerancia){
        msg2= "\rAflojar cuerda\n\0";
    }
    else if(frecmed<196-tolerancia){
        msg2= "\rTensar cuerda\n\0";
    }

}else if(ReceivedChar == '5'){
    scia_msg(" (246Hz)");
    msg = "\r\nEstas afinando la quinta cuerda \n\0";
    cuerda=5;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPASET.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;

    if(frecmed>=246-tolerancia && frecmed<=246+tolerancia){
        msg2="\rBuena afinacion\n\0";
    }
    else if(frecmed>246+tolerancia){
        msg2= "\rAflojar cuerda\n\0";
    }
    else if(frecmed<246-tolerancia){
        msg2= "\rTensar cuerda\n\0";
    }
```

```
}else if(ReceivedChar == '6'){
    scia_msg(" (329Hz)");
    msg = "\r\nEstas afinando la sexta cuerda(la mas delgada) \n\0";
    cuerda=6;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPASET.bit.GPIO31 = 1;


    if(frecmed>=329-tolerancia && frecmed<=329+tolerancia){
        msg2="\rBuena afinacion\n\0";
    }
    else if(frecmed>329+tolerancia){
        msg2= "\rAflojar cuerda\n\0";
    }
    else if(frecmed<329-tolerancia){
        msg2= "\rTensar cuerda\n\0";
    }

}else{
    msg = "\r\nHe receibido un comando desconocido \n\0";
    cuerda=0;
    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
}

scia_msg(msg);
scia_msg(msg2);
transmite_frecuencia(frecmed);

msg = "\r\nSelecciona la cuerda: \0";
scia_msg(msg);

LoopCount++;
}
```

En resumen, para entender mejor el funcionamiento de este periférico, se ha realizado un diagrama de bloques que explica el proceso realizado por el SCI:

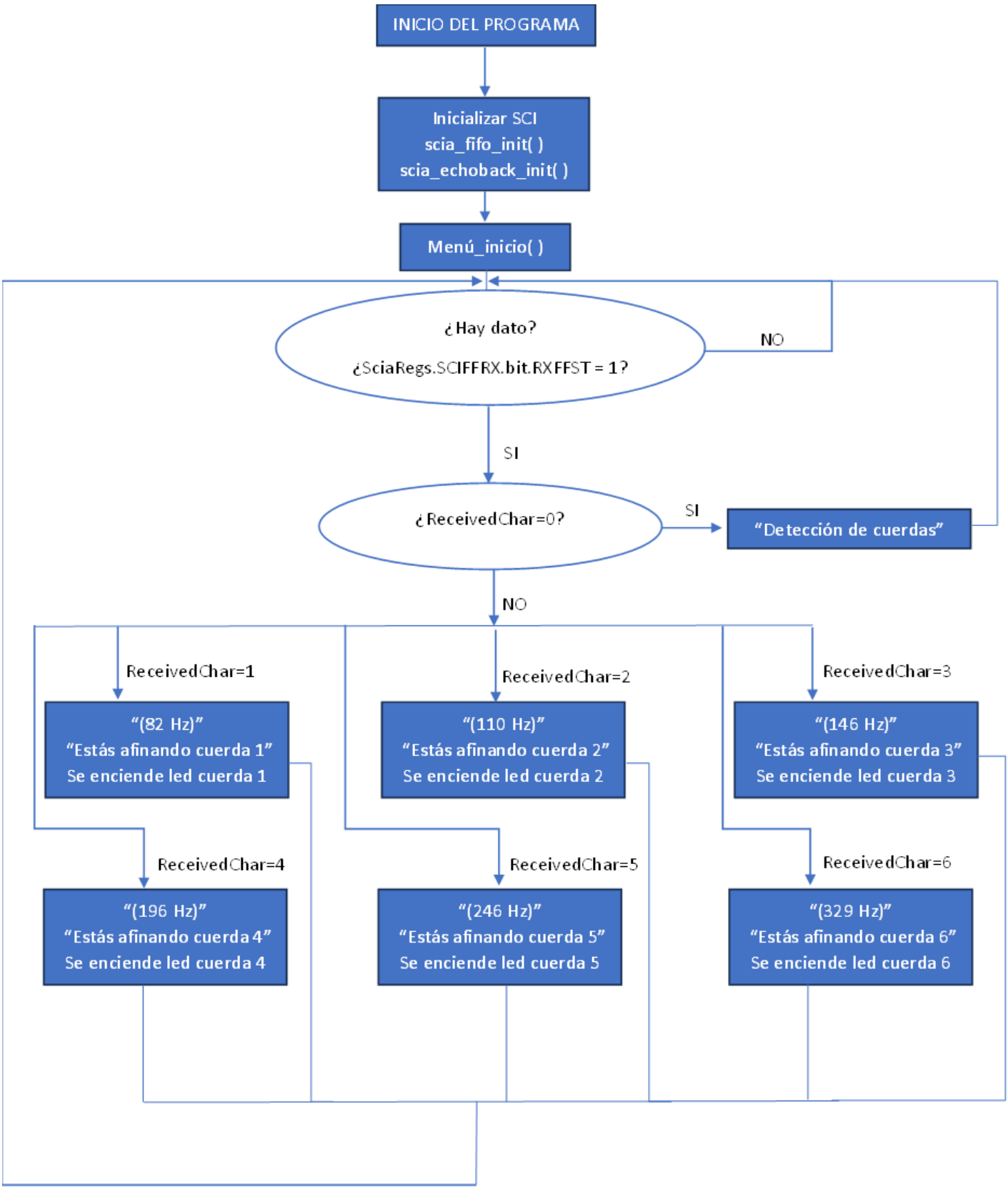

Ilustración 27 Esquema del funcionamiento del periférico SCI

6 RESULTADOS Y EVALUACIÓN

Este capítulo va a dedicarse a la evaluación y comparación de los métodos usados para llegar al objetivo de este proyecto. En el capítulo tres se vieron tres métodos que se podían abordar para el cálculo de la frecuencia. De entre estas tres opciones, finalmente se llevaron a cabo dos de ellos: el método del cruce por cero, y el algoritmo de Goertzel.

La elección del cruce por cero se realizó debido a su una forma sencilla y eficiente de detectar cambios y características importantes en una señal temporal. Al tratar de implementar y probar este algoritmo se han visto una serie de ventajas:

1. Simplicidad: El algoritmo para detectar cruces por cero es simple y fácil de implementar.
2. Baja complejidad computacional: Requiere pocas operaciones, por lo que es muy rápido.
3. Detección de frecuencia: Puede ser útil para una estimación rápida y aproximada de la frecuencia fundamental de una señal.

Sin embargo, también se han encontrado una serie de desventajas:

1. Precisión limitada: No proporciona información precisa sobre el contenido espectral de la señal.
2. Sensibilidad al ruido: El método puede ser muy sensible al ruido, lo que puede llevar a falsos cruces por cero. Para mitigar este inconveniente, como se sabía el rango de frecuencias que se querían detectar, se implementó en la programación un filtro.

Por otro lado, entre la FFT y el algoritmo de Goertzel se optó por esta última opción. Su elección final resultó por una serie de ventajas que presentaba:

1. Eficiencia en frecuencia específica: Es más eficiente que la FFT cuando se necesita calcular la DFT en una o pocas frecuencias específicas.
2. Aplicaciones en tiempo real: Puede ser más adecuado para aplicaciones en tiempo real donde se necesita monitorear frecuencias específicas continuamente.
3. Menor uso de memoria: Requiere menos memoria que la FFT, ya que sólo necesita mantener un pequeño estado de la señal.

Debido al contexto en el que se ha aplicado este método, se ha podido llevar a cabo. Si no se supieran las frecuencias que se están buscando, no podría haber sido posible elegir este camino, ya que no proporciona una visión completa del espectro de frecuencias de la señal. Por tanto, tendría que haberse implementado la FFT.

Todo ello es importante que se tenga en cuenta, para poder entender los resultados que se van a obtener. Para comprobar el funcionamiento, se han sometido estos dos métodos a dos escenarios distintos para la detección de las frecuencias.

6.1 Obtención de resultados

Antes de realizar un análisis sobre los resultados obtenidos, se hará una breve descripción de los distintos escenarios donde se experimentó con el afinador. Para ambos casos se utilizó una guitarra afinada. Sabiendo esto los contextos son los siguientes:

1) Escenario sin ruido: para este caso se llevó a cabo la detección de frecuencias en una habitación cerrada y sin ningún tipo de ruido. Como se puede deducir, es el caso que más se aproxima a uno ideal.
2) Escenario con ruido: en este caso se quiso introducir ruido en la detección de frecuencias. Para ello, se puso música con el volumen máximo cerca del micrófono. Por lo tanto, en contraposición con el otro

escenario, este es el que más se acerca al peor caso posible.

Una vez que se sabe todo esto, se plantea la hipótesis de que los mejores resultados que se podrían obtener sería que el caso más desfavorable se acercara lo máximo posible al caso ideal, independientemente de si la guitarra estuviera afinada o no. Es esencial remarcar que en ambos escenarios no se cambió la afinación de la guitarra. En caso contrario, estos resultados no serían válidos ya que no podríamos aplicar la hipótesis descrita.

Para la obtención de los resultados se han obtenido una serie de gráficas donde se representan las diferentes muestras obtenidas a lo largo del tiempo, y las frecuencias obtenidas por la pantalla del dispositivo. Para cada método se han elaborado 6 gráficas, una por cada cuerda de la guitarra.

6.1.1 Resultados con el método del cruce por cero

En este apartado, se mostrarán las 6 gráficas que reflejan los resultados obtenidos por medio del método del cruce por cero.

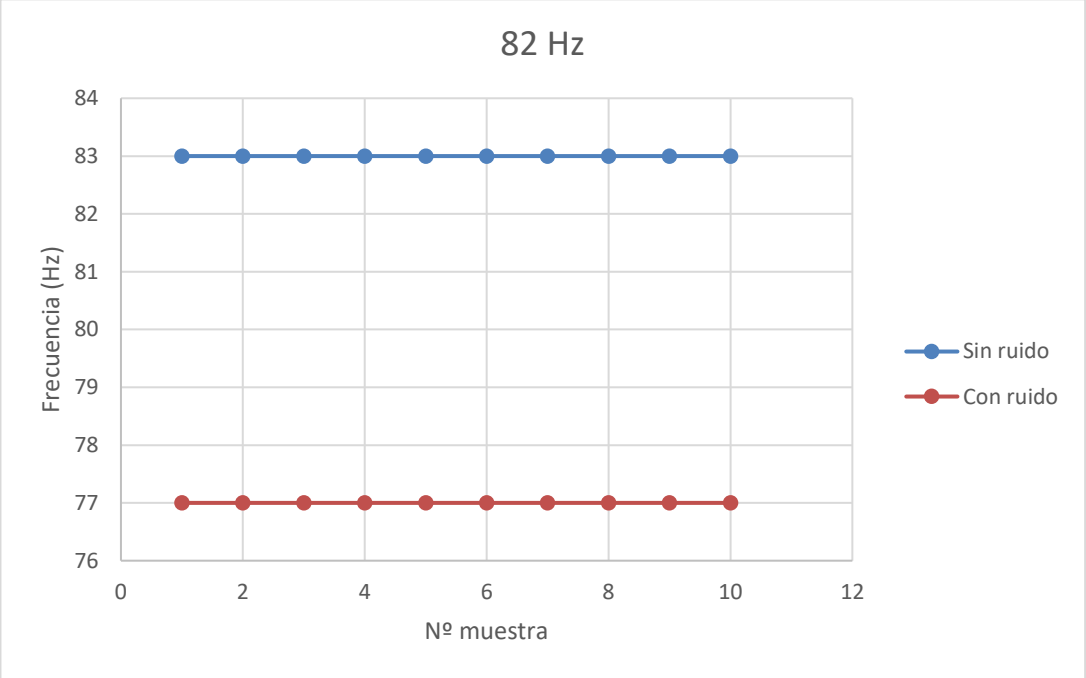

Ilustración 28 Detección de la primera cuerda (cruce por cero)

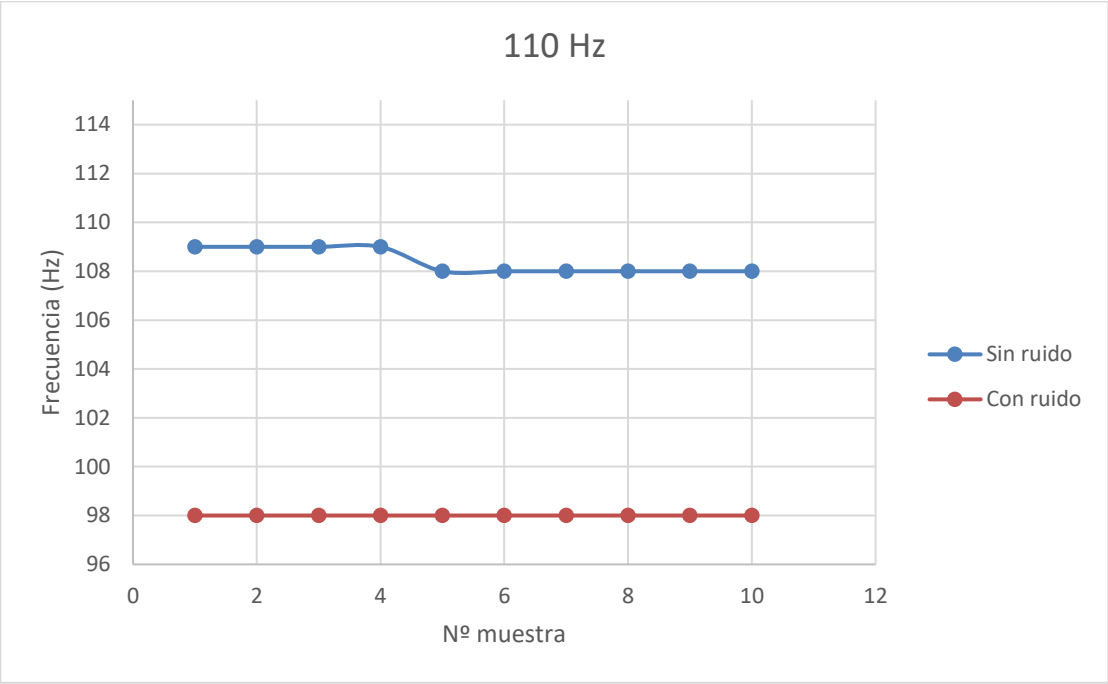

Ilustración 29 Detección de la segunda cuerda (cruce por cero)

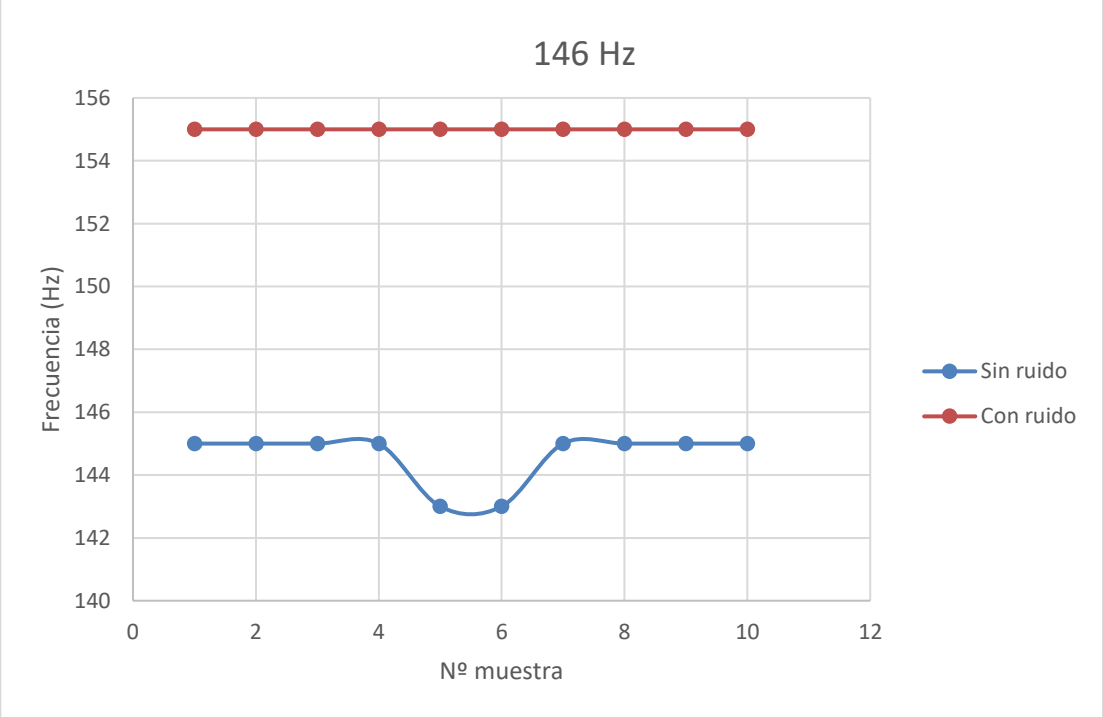

Ilustración 30 Detección de la tercera cuerda (cruce por cero)

Si se observa la detección de las tres primeras cuerdas, se puede ver que, en cuanto al caso ideal, se detectan muy bien las frecuencias y es prácticamente estable a lo largo del tiempo. Por otro lado, cuando se tiene ruido en la detección de la frecuencia, el afinador da un resultado erróneo, aunque da siempre la misma frecuencia, es decir, no oscilan los resultados.

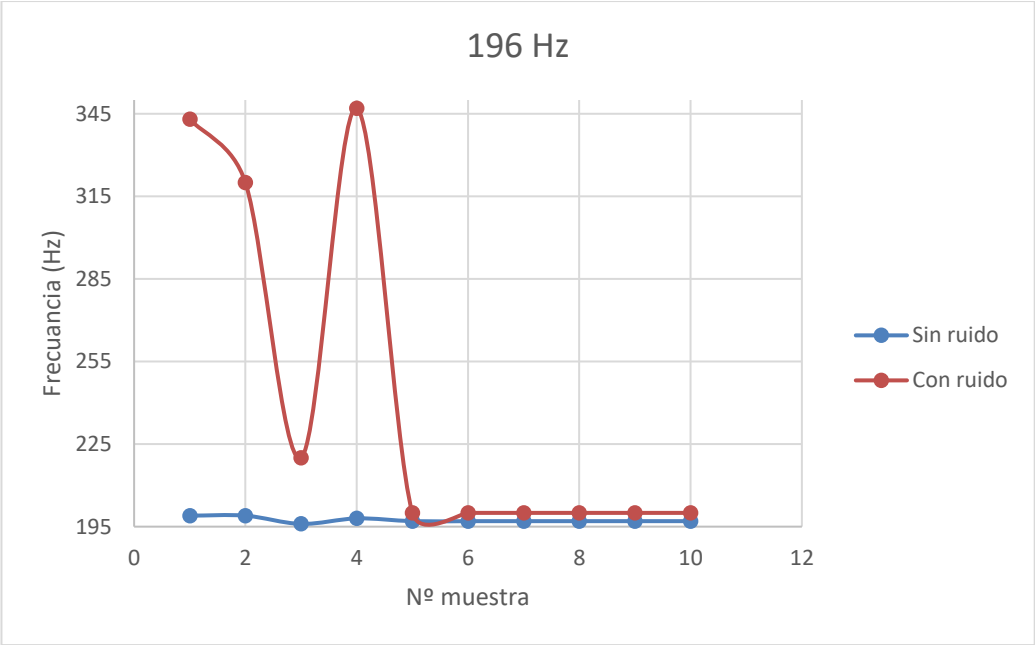

Ilustración 31 Detección de la cuarta cuerda (cruce por cero)

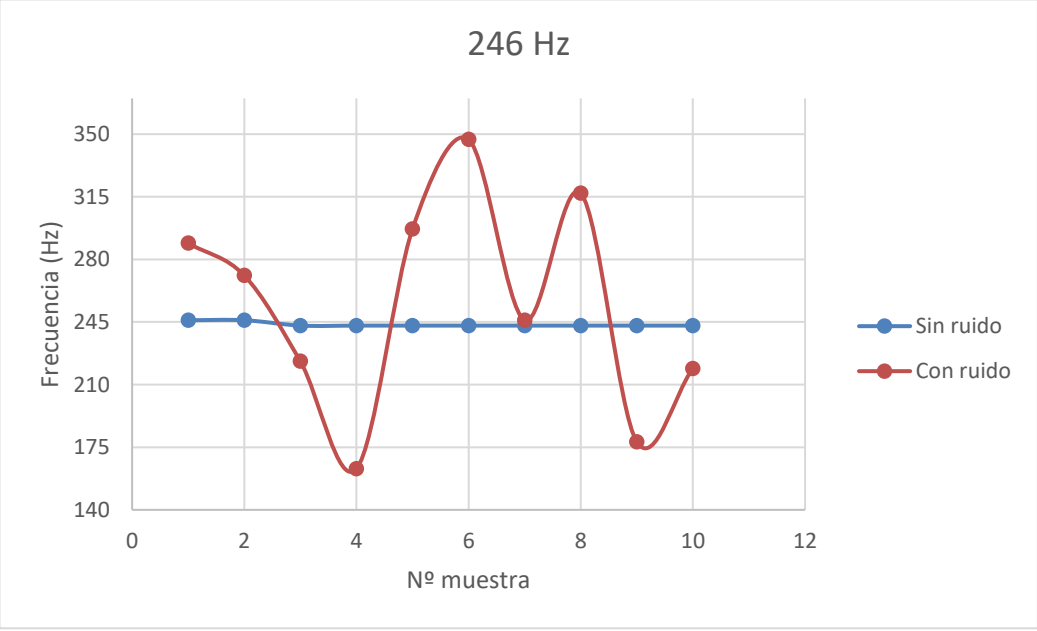

Ilustración 32 Detección de la quinta cuerda (cruce por cero)

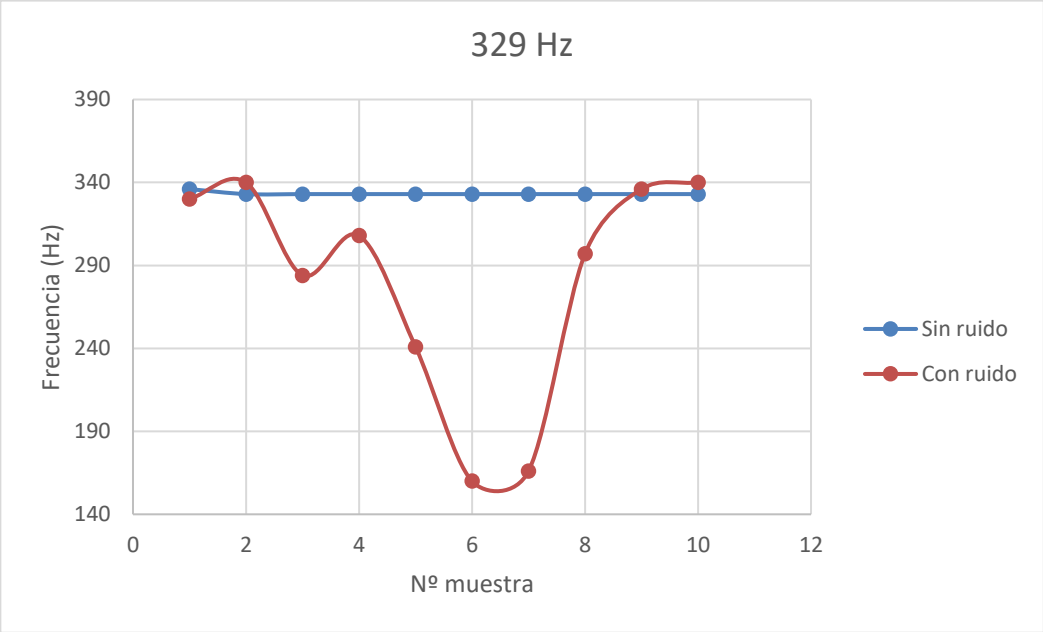

Ilustración 33 Detección de la sexta cuerda (cruce por cero)

Al observar la detección de las tres últimas cuerdas, se obtiene un resultado correcto y estable a lo largo del tiempo. En cambio, cuando se tiene ruido en la detección de la frecuencia, los resultados que muestra el afinador son erróneos y son poco claros ya que siempre oscilan y no se estabilizan en ningún momento.

Para poder observar los resultados obtenidos de una manera más global, se ha realizado una tabla en el que se reflejan los errores cuadráticos medios de las medidas con ruido y sin ruido.

Tabla 4 Errores cuadráticos medios con el método del cruce por cero

	ERROR CUADRÁTICO MEDIO	
CUERDA	SIN RUIDO	CON RUIDO
1 (82 Hz)	1	25
2 (110 Hz)	2.8	244
3 (146 Hz)	2.6	81
4 (196 Hz)	2.8	6045.8
5 (246 Hz)	7.2	3308.8
6 (329 Hz)	19.3	6665.6

6.1.2 Resultados con el algoritmo de Goertzel

En este apartado, se mostrarán las 6 graficas que reflejan los resultados obtenidos por medio del algoritmo de Goertzel.

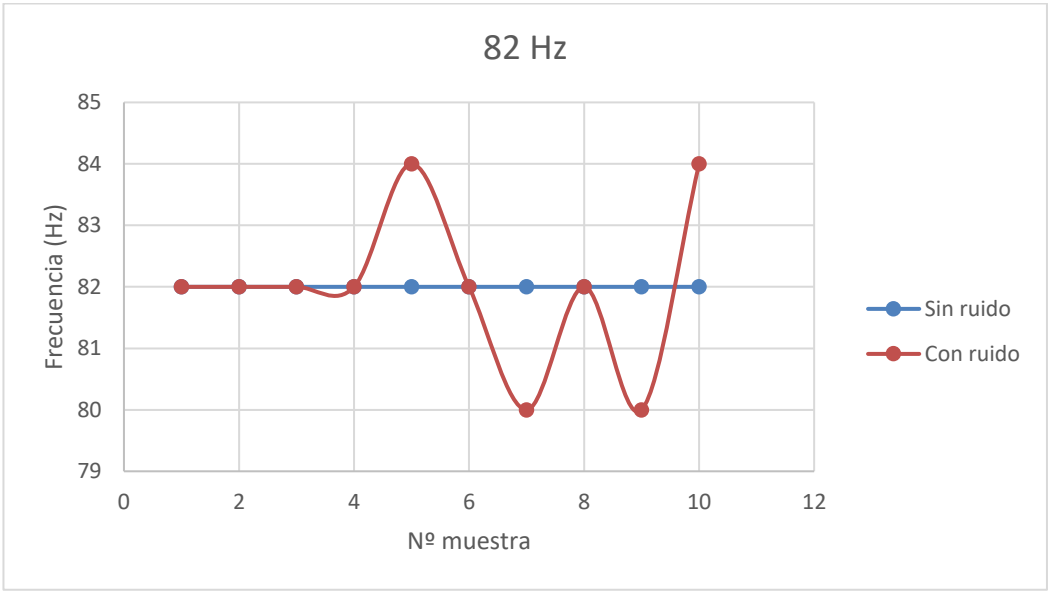

Ilustración 34 Detección de la primera cuerda (algoritmo de Goertzel)

En la detección de esta primera frecuencia, obtenemos bastantes oscilaciones cuando hay ruido en la detección de frecuencias. Sin embargo, en los primeros instantes sí que es capaz de dar un resultado correcto. Esto ocurre porque al trascurrir el tiempo, el sonido de la guitarra va disminuyendo y por tanto tienen más protagonismo otras frecuencias.

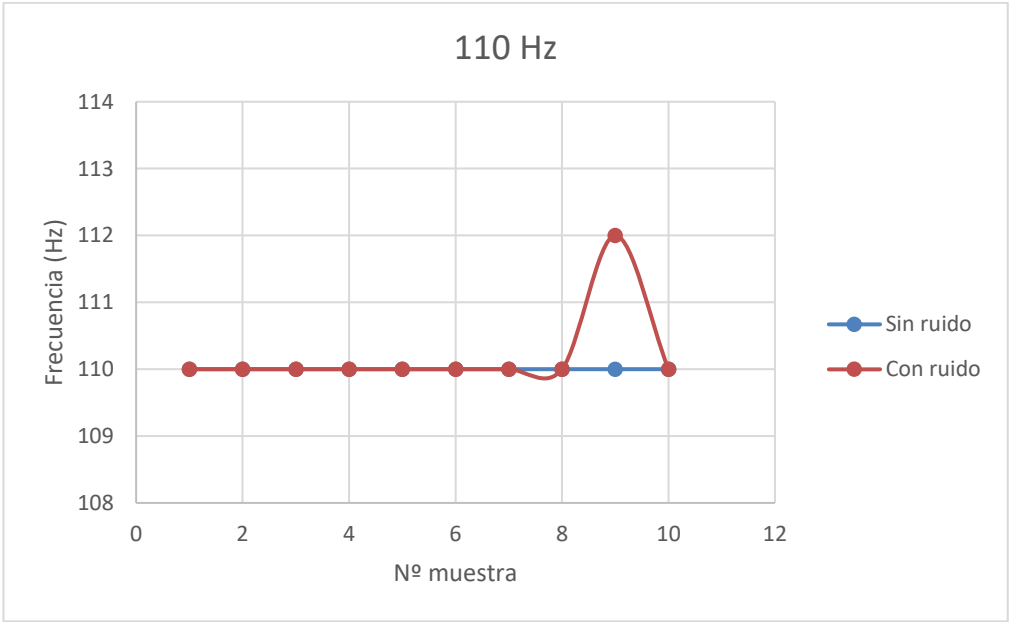

Ilustración 35 Detección de la segunda cuerda (algoritmo de Goertzel)

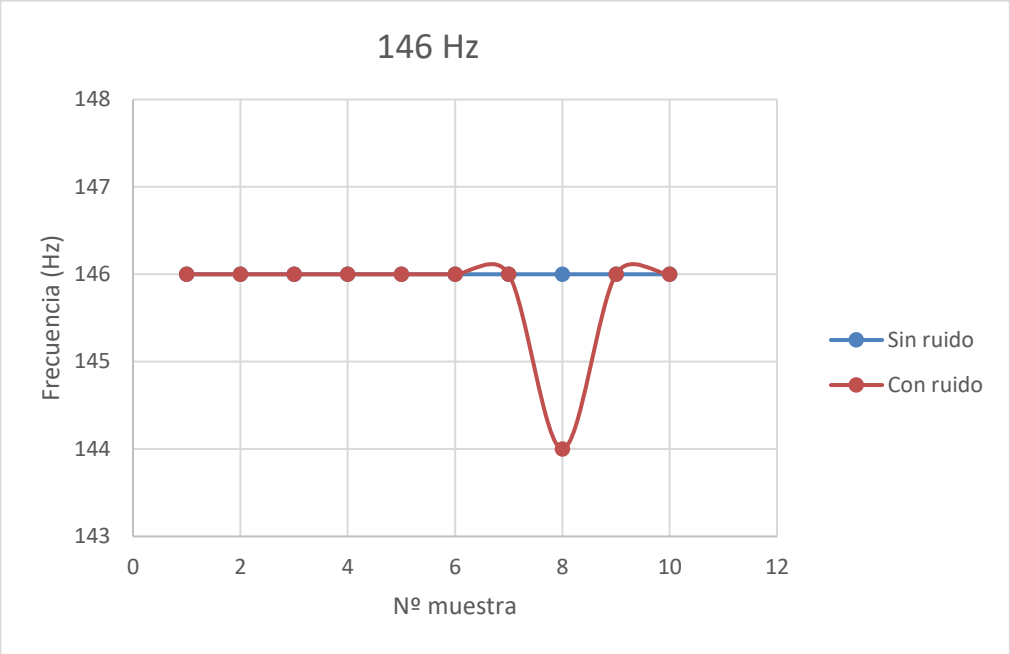

Ilustración 36 Detección de la tercera cuerda (algoritmo de Goertzel)

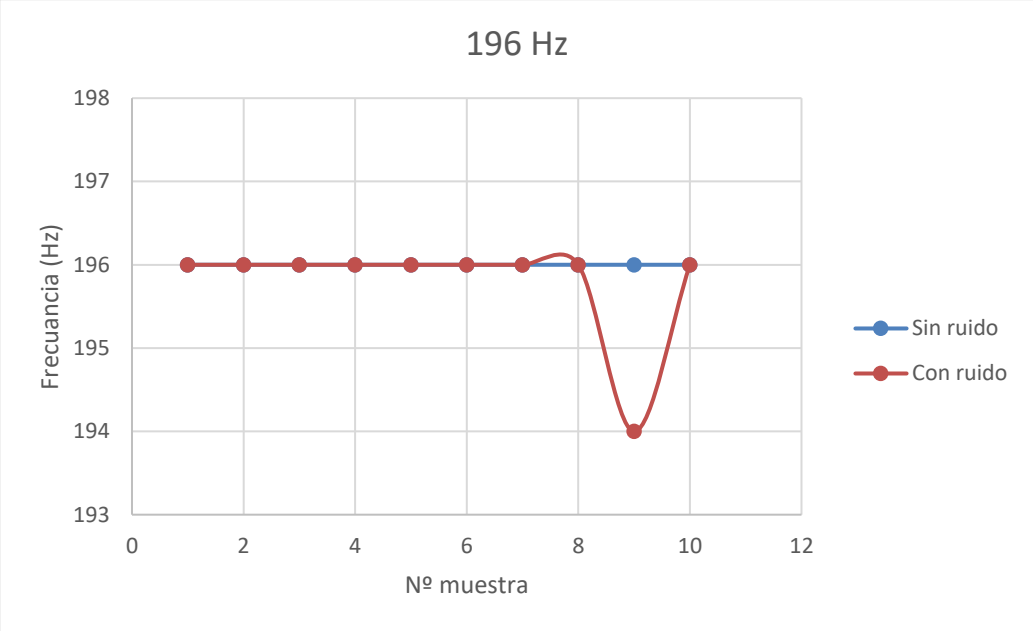

Ilustración 37 Detección de la cuarta cuerda (algoritmo de Goertzel)

En la afinación de la segunda, tercera y cuarta cuerda se obtienen resultados óptimos. Por un lado, en el caso ideal, la detección de la frecuencia es inmediata y a lo largo del tiempo sigue detectando la frecuencia. Por otro lado, en el escenario donde se ha introducido ruido a la medida, se tiene un comportamiento bastante parecido para estas tres cuerdas. Se es capaz de detectar la frecuencia correcta a lo largo de la duración del sonido, excepto los últimos instantes, que al igual que antes, al haber una disminución de la magnitud de la cuerda tocada, se detectan otras frecuencias.

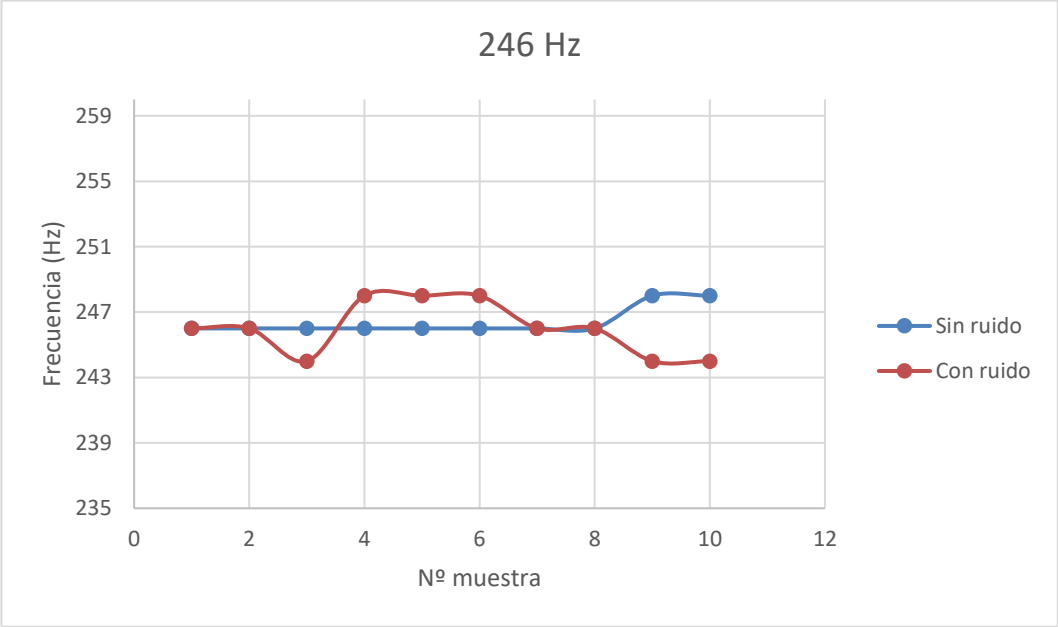

Ilustración 38 Detección de la quinta cuerda (algoritmo de Goertzel)

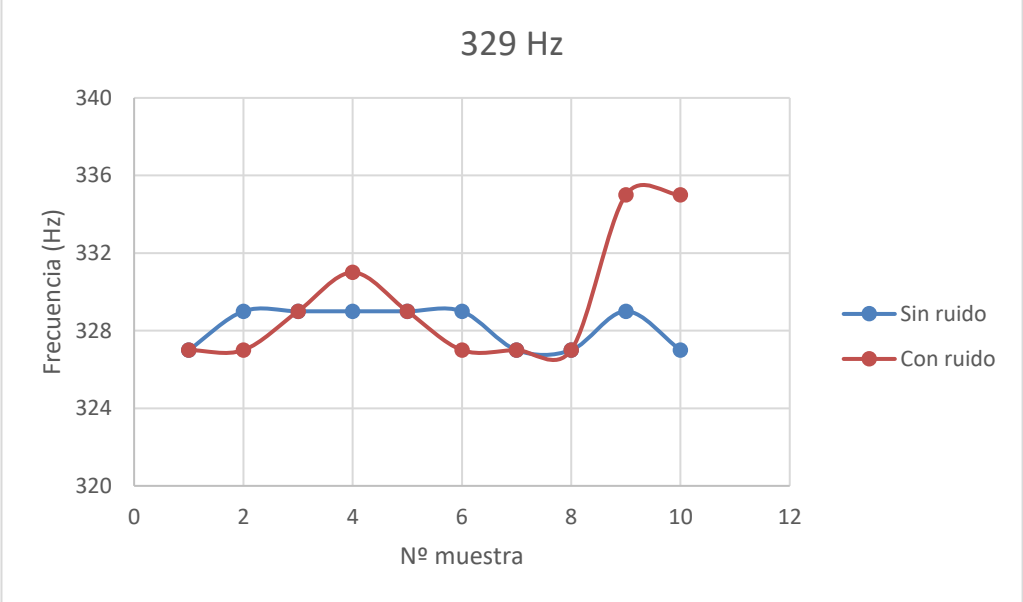

Ilustración 39 Detección de la sexta cuerda (algoritmo de Goertzel)

Por último, para la detección de las últimas dos cuerdas se puede ver un comportamiento algo distinto al resto de las cuerdas. Para entender este comportamiento, se debe saber que cuando se toca cualquiera de estas dos cuerdas la duración del sonido emitido es mucho menor. Esto está influido por varios factores, entre ellos, por el material y el grosor de estas, concepto que ya se explicó en el capítulo de fundamentos teóricos. Por otro lado, también se puede ver afectado por la tensión de las cuerdas. Las cuerdas con mayor tensión pueden tener una duración de sonido más corta porque la energía de la vibración se disipa más rápidamente. Por estas razones, la detección de la frecuencia dura mucho menos, aunque este problema se puede solucionar tocando repetidamente las veces que sean necesarias las cuerdas. En consecuencia, cuando se introduce ruido en la medida, la precisión de la detección de la frecuencia se ve afectada, aunque se aproxima bastante al resultado ideal.

Al igual que se hizo en el apartado anterior, se ha realizado una tabla en la que se resumen los resultados reflejados en las gráficas. Para ello, se ha calculado el error cuadrático medio con ruido y sin ruido.

Tabla 5 Errores cuadráticos medios con el algoritmo de Goertzel

	ERROR CUADRÁTICO MEDIO	
CUERDA	SIN RUIDO	CON RUIDO
1 (82 Hz)	0	1.6
2 (110 Hz)	0	0.4
3 (146 Hz)	0	0.4
4 (196 Hz)	0	0.4
5 (246 Hz)	0.8	2.4
6 (329 Hz)	1.6	9.6

6.2 Conclusiones

Para concluir todo este proceso de evaluación del funcionamiento del afinador se concluye que, en este contexto, el método de Goertzel es bastante más efectivo y fiable que el cruce por cero. Ambas metodologías tienen sus desventajas, y el hecho de que elijamos un método u otro hace que estemos perdiendo las ventajas del otro.

El concepto teórico utilizado en cada método es completamente distinto. El cruce por cero mide el tiempo que tarda en cruzar un determinado valor, y por medio de la medición del tiempo se obtiene la frecuencia. Este método tiene sus ventajas ya se calcula la frecuencia directamente sobre la onda que recoge el micrófono. En un escenario ideal es un método que funciona muy bien porque no existen interferencias, pero en la vida real es muy difícil que alguien se encuentre en una situación así. Cuando se introducen interferencias las ondas de cada sonido se suman, y la señal resultante es imposible que tenga la frecuencia de la cuerda que se ha tocado.

En cambio, el método del algoritmo de Goertzel mide la magnitud de las posibles frecuencias que podrían estar sonando. Aquella frecuencia que tenga una mayor magnitud será la frecuencia que está sonando. De esta manera, aunque la afinación se realice en un ambiente donde haya interferencias, será muy difícil que influyan en los resultados ya que se están buscando frecuencias muy concretas. El inconveniente que se puede encontrar en este método es que se pierde precisión en la medida, aunque eso no es tan importante ya que el oído humano es incapaz de diferenciar dos frecuencias con una diferencia entre ellas de 2 hercios. Por otro lado, este método no calcula la frecuencia sobre la onda que recoge el micrófono, y, por lo tanto, no nos dice realmente la frecuencia a la que suenan las cuerdas de la guitarra. Solo mide la magnitud de varias frecuencias que se han indicado en nuestro programa a las que podría estar sonando la guitarra, y se concluye que aquella que tenga más peso, es la que está sonando. En esta aplicación en concreto, este inconveniente no importa demasiado ya que lo que realmente se quiere saber es si la guitarra está afinada o no y si hay que aflojar o tensar las cuerdas.

Como conclusión, el método final por el que se ha optado para el diseño de este afinador ha sido el método de Goertzel. Por medio de la experimentación, y una revisión y comparación de los resultados, se ha visto que el método óptimo. Es un algoritmo por el cual la afinación es muy rápida, ya que la detección de las frecuencias es inmediata. Además, se ha visto que la presencia de interferencias en la medida influye apenas en el resultado. Para el diseño de este tipo siempre se debe encontrar un equilibrio entre la precisión y la fiabilidad de los resultados. Dependiendo del dispositivo que se quiera construir y la importancia de estos dos factores en la

aplicación, elección del método será distinto. En este caso, la fiabilidad es un factor mucho más esencial a la hora de la afinación de la guitarra, ya que la obtención de frecuencias erróneas no permitiría obtener el resultado por el que está diseñado este dispositivo. Por estas razones es por las que finalmente se eligió este método.

7 CONCLUSIONES

Los objetivos de este proyecto han sido superados por completo, a pesar de las dificultades y obstáculos que se han presentado durante el proceso. Ha sido un reto desafiante, en el que se han podido poner en práctica todos los conocimientos que se han adquirido a lo largo de estos años en la carrera.

Aunque ha sido un proyecto que ha culminado en un éxito rotundo, durante el camino se encontraron algunas dificultades y limitaciones. Entre ellas, cabe destacar la elección del micrófono. En un principio se compró otro micrófono, pero cuando se realizaron mediciones a través del osciloscopio, se vio que, debido a la sensibilidad, no era capaz de detectar el sonido de la guitarra. Otra dificultad remarcable fue la implementación de la pantalla LCD. Para su programación se tuvo que buscar librerías que manejaban la pantalla. Normalmente, las librerías que hay en internet no están adaptadas para todos los DSP existentes, por tanto, primero se tuvo que entender cómo funcionaba la pantalla internamente, y luego, realizar la adaptación del código.

Una vez finalizado este trabajo, se puede concluir que casi todos los objetivos que se plantearon en un principio han sido alcanzados. Sin embargo, como con cualquier proyecto, algunas propuestas no se llevaron a cabo debido a la falta de tiempo. Una de las cosas que fueron planteadas fue la realización de un modelo 3D, que actuara como una carcasa para la cubrir todos los componentes del afinador, incluyendo una batería independiente para que nuestro dispositivo fuese portable.

7.1 Líneas futuras de trabajo

Una posible ampliación bastante sencilla y viable sería la realización de un afinador para otros instrumentos de cuerdas, que tengan un proceso de afinación parecido al de la guitarra, como un ukelele, un bajo o un violín entre otros. Ajustando los parámetros correspondientes programados en el DSP, se podría conseguir esta idea en cuestión de tan solo horas. Por otro lado, si nos centramos en el tratamiento de la señal que se ha aplicado a este trabajo, este concepto podría extenderse a otros campos fuera de la música, como puede ser en la medicina, aplicándose a los electrocardiogramas de manera que ayude a analizar las señales. También se podría aplicar a otros ámbitos como en las telecomunicaciones o en sistemas de control y automatización industrial.

REFERENCIAS

Todas las referencias y fuentes citadas siguiendo las normas APA y usando el software propuesto por la universidad de Sevilla Mendeley Reference Manager

[1] Chassaing, R. (2005). *Digital signal processing and applications with the C6713 and C6416 DSK*. Wiley-Interscience.

[2] Texas Instruments. (2009). *TMS320x2833x, TMS320x2823x technical reference manual* (Rev. C). Texas Instruments.

[3] DFRobot. (n.d.). *Analog sound sensor SKU: DFR0034*. DFRobot Wiki. https://wiki.dfrobot.com/Analog_Sound_Sensor_SKU__DFR0034

[4] Talos Electronics. (n.d.). *Display LCD 16x2 HD44780*. Talos Electronics. https://www.taloselectronics.com/blogs/tutoriales/display-lcd-16-2-hd44780

[5] AllDataSheet. (n.d.). *ZS-040 datasheet*. AllDataSheet. https://pdf1.alldatasheet.com/datasheet-pdf/view/1253161/ETC1/ZS-040.html

[6] Goertzel algorithm. (2024, July 4). En *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Goertzel_algorithm

ANEXO A: CÓDIGO DEL PROYECTO

```
//###########################################################################
// AFINADOR DE GUITARRA
//###########################################################################
#include "DSP2833x_Device.h"
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include "sci.h"
#include "LCD.h"

//define
#define SAMPLE_RATE 2050 // sample_rate = 1/timer
#define N 1024
#define ADC_MAX_VALUE 4095   // Maximum value for a 12-bit ADC
#define ADC_MID_VALUE (ADC_MAX_VALUE / 2)
#define NUM_FREQS 6          // Number of frequencies to check


// Protoripado de funciones externas
extern void InitAdc(void);
extern void InitSysCtrl(void);
extern void InitFlash(void);
extern void InitSciaGpio(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitCpuTimers(void);
extern void ConfigCpuTimer(struct CPUTIMER_VARS *, float, float);


// Prototype de funciones internas
void goertzel_init(double *cosine, double *sine, double *coeff, int k);
double goertzel_filter(double *samples, double coeff, double cosine, double sine);
void analyze_frequencies(double *target);
void afina_cuerda(double *target);

void Gpio_select(void);
void InitAdcRegs(void);
interrupt void cpu_timer0_isr(void);
interrupt void adc_SEQ1_isr(void);


// Global counts
Uint16 LoopCount;
Uint16 ErrorCount;
// Variables globales
float Vbin=0;
double samples[N];
volatile int sample_index = 0;
volatile int buffer_full = 0;
int i;
int timer=488;
// Frequencies of guitar strings: E2, A2, D3, G3, B3, E4
double target_freqs[NUM_FREQS] = {82, 110.00, 146.83, 196.00, 246.94, 329.63};
double cuerda1[NUM_FREQS] = {76, 78, 80, 82, 84, 86};
double cuerda2[NUM_FREQS] = {104, 108, 110, 112, 116, 120};
```

```
double cuerda3[NUM_FREQS] = {140, 144, 146.83, 148, 154, 160};
double cuerda4[NUM_FREQS] = {186, 194, 196, 198, 206, 214};
double cuerda5[NUM_FREQS] = {236, 244, 246.94, 248, 254, 262};
double cuerda6[NUM_FREQS] = {319, 327, 329.63, 331, 335, 339};
double magnitudes[NUM_FREQS];
int max_index = 0;
double max_magnitude = 0.0;
float frecuencia=0;
int cuerda=0;
int metodo=0;
float Vbin_ant=0;
float frecuencia_ant=0;
float t=0;
int start=0;
int contador=0;
int tint=0;
float frecmed=0;
int frec=0;
float sumfrec=0;
int inc_lcd=0;
int t_lcd=0;

#ifdef __TI_COMPILER_VERSION__
    #if __TI_COMPILER_VERSION__ >= 15009000
        #pragma CODE_SECTION(adc_SEQ1_isr, ".TI.ramfunc");
        #pragma CODE_SECTION(cpu_timer0_isr, ".TI.ramfunc");
    #else
        #pragma CODE_SECTION(adc_SEQ1_isr, "ramfuncs");
        #pragma CODE_SECTION(cpu_timer0_isr, "ramfuncs");
    #endif
#endif

extern Uint16 RamfuncsLoadStart;
extern Uint16 RamfuncsLoadEnd;
extern Uint16 RamfuncsRunStart;
extern Uint16 RamfuncsLoadSize;

void update_timer(int timer_value) {
    ConfigCpuTimer(&CpuTimer0, 150, timer_value);
    CpuTimer0Regs.TCR.bit.TSS = 0;  // Iniciar el timer
}
//###########################################################################
//                       main code
//###########################################################################
void main(void)
{

    Uint16 ReceivedChar;
    char *msg;
    char *msg2;
    char *msg3;
    char *detect;
    char *afina;
    char pantalla[16]={'\0'};

    InitSysCtrl();                          // Inicializacion del reloj y del
watchdog

                                            // SYSCLKOUT = 150MHz
                                            // HSPCLK = 75MHz
    Gpio_select();
```

```
    InitSciaGpio();

    DINT;                                   // Deshabilitacion de todas las
interrupciones

    InitCpuTimers();                        // Inicializacion de los Timers

    ConfigCpuTimer(&CpuTimer0,150,timer);   // Inicializacion Timer0 en microsegundos
(ej. 30us)

    InitPieCtrl();                          // Inicializacion de la PIE
    IER = 0x0000;
    IFR = 0x0000;



    InitPieVectTable();                     // Inicializacion de la tabla PIE

    // Modificacion tabla PIE para las interrupciones del Timer0 y del ADC
    EALLOW;
    PieVectTable.TINT0 = &cpu_timer0_isr;
    PieVectTable.SEQ1INT = &adc_SEQ1_isr;
    EDIS;

    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;      // Habilitacion de interrupcion por lonea
INT1.7 (Timer0)
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;      // Habilitacion de interrupcion por lonea
INT1.6 (ADC, SEQ1)

    IER = 0x0001;                           // Habilitacion de interrupcion por lonea
INT1

    EINT;                                   // Habilitacion de todas las
interrupciones

    memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (Uint32)&RamfuncsLoadSize);
    InitFlash();

    InitAdc();                              // Inicializacion basica del ADC
                                            // Habilita reloj del ADC
                                            // power-up del ADC
                                            // Hace la calibracion

    InitAdcRegs();                          // Inicializacion resto de registros del
ADC

    CpuTimer0Regs.TCR.bit.TSS = 0;          // Inicio de la cuenta del timer0

    //MENU DEL BLUETHOOTH

    LoopCount = 0; // Para contar cuantos caracteres hemos enviado

    scia_fifo_init();       // Inicializamos la FIFO del SCI
    scia_echoback_init();  // Inicializamos los parametros del SCI

    menu_inicio();

    //PANTALLA
    // Initialize lcd driver
```

```
    lcd_init(0, 16, 2);
    lcd_cursor(E_LCD_CURSOR_ON);
    lcd_home();
    lcd_on();
    lcd_puts("AFINADOR DE");
    lcd_goto(0,1);
    lcd_puts("  GUITARRA");

    while(1)
    {

        if(SciaRegs.SCIFFRX.bit.RXFFST == 1) {
            cuerda=0;
            scia_msg("\e[1;1H\e[2J");
            ReceivedChar = SciaRegs.SCIRXBUF.all;
            if (ReceivedChar == '1'){
                scia_msg(" Metodo elegido: Cruce por cero");
                timer=30;
                metodo=1;
                update_timer(timer);
            }else if (ReceivedChar == '2'){
                scia_msg(" Metodo elegido: Algoritmo de Goertzel");
                timer=488;
                metodo=2;
                update_timer(timer);
            }else{
                metodo=0;
            }
            scia_msg("\r\nDeberas seleccionar la cuerda que quieras afinar.\n\0");

            scia_msg("\r\nSelecciona la cuerda: \0");
        }
        while(metodo==1){ //cruce por cero

            if(Vbin <= 150 && Vbin_ant > 150){
                contador++;
                start = 1;
            }


            Vbin_ant = Vbin;


            if(start==1){
                if(contador == 2){ //cuando haya contado dos máximos, es decir, un
periodo, calcula frecuencia
                    frecuencia=pow(t,-1);
                    contador=0;
                    start = 0;
                    t=0;
                    tint=0;
                }
            }

            if(cuerda==0){
                if(frecuencia>500 || frecuencia<50){
                    frecuencia=frecuencia_ant;
                }
            }
```

```
            else if(cuerda==1){
                if(frecuencia>110 || frecuencia<50){
                    frecuencia=frecuencia_ant;
                }
            }

            else if(cuerda == 2 || cuerda == 3){
                if(frecuencia>180 || frecuencia<98){
                    frecuencia=frecuencia_ant;
                }
            }
            else if(cuerda==4 || cuerda == 5 || cuerda == 6){
                if(frecuencia>350 || frecuencia<150){
                    frecuencia=frecuencia_ant;
                }
            }

            if(frecuencia!=0){
                frec++;
                sumfrec=frecuencia+sumfrec;
                frecmed=sumfrec/frec;
            }
            if(frec>=3000){
                frec=0;
                sumfrec=0;
            }

            frecuencia_ant=frecuencia;


            // Transmisión por puerto serie
            if(SciaRegs.SCIFFRX.bit.RXFFST == 1) {


                scia_msg("\e[1;1H\e[2J");
                msg = "\r\nSelecciona la cuerda: \0";
                scia_msg(msg);

                ReceivedChar = SciaRegs.SCIRXBUF.all;
                scia_xmit(ReceivedChar);

            if(ReceivedChar == '1'){
                scia_msg(" (82Hz)");
                msg = "\r\nEstas afinando la primera cuerda(la mas gruesa)\n\0";
                cuerda=1;

                GpioDataRegs.GPASET.bit.GPIO17 = 1;    //enciende led conectado al
pin 17
                GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;  //apaga led conectado al pin
19
                GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;  //apaga led conectado al pin
21
                GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;  //apaga led conectado al pin
23
                GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;  //apaga led conectado al pin
27
                GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;  //apaga led conectado al pin
31
```

```
}else if(ReceivedChar == '2'){
    scia_msg(" (110Hz)");
    msg = "\r\nEstas afinando la segunda cuerda \n\0";
    cuerda=2;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPASET.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;


}else if(ReceivedChar == '3'){
    scia_msg(" (146Hz)");
    msg = "\r\nEstas afinando la tercera cuerda \n\0";
    cuerda=3;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPASET.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;



}else if(ReceivedChar == '4'){
    scia_msg(" (196Hz)");
    msg = "\r\nEstas afinando la cuarta cuerda \n\0";
    cuerda=4;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPASET.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;


}else if(ReceivedChar == '5'){
    scia_msg(" (246Hz)");
    msg = "\r\nEstas afinando la quinta cuerda \n\0";
    cuerda=5;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPASET.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;


}else if(ReceivedChar == '6'){
    scia_msg(" (329Hz)");
    msg = "\r\nEstas afinando la sexta cuerda(la mas delgada) \n\0";
    cuerda=6;

    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
```

```
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPASET.bit.GPIO31 = 1;



}else if (ReceivedChar == '0'){
    msg = "\r\nDeteccion frecuencia \n\0";

    cuerda=0;
    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
}else{
    metodo=0;
    cuerda=-1;
    menu_inicio();
}

if(metodo!=0){
scia_msg(msg);

msg = "\r\nSelecciona la cuerda: \0";
scia_msg(msg);
}

LoopCount++;
}

if (cuerda==1){
    if((int)frecmed==82){
        msg2=" Afinado";
    }
    else if((int)frecmed>82){
        msg2= " Afloja";
    }
    else if((int)frecmed<82){
        msg2= " Tensar";
    }
}
else if (cuerda==2){
    if((int)frecmed==110){
        msg2=" Afinado";
    }
    else if((int)frecmed>110){
        msg2= " Afloja";
    }
    else if((int)frecmed<110){
        msg2= " Tensar";
    }
}
else if (cuerda==3){
    if((int)frecmed==146){
        msg2=" Afinado";
```

```
        }
        else if((int)frecmed>146){
            msg2= " Afloja";
        }
        else if((int)frecmed<146){
            msg2= " Tensar";
        }
    }
    else if (cuerda == 4){
        if((int)frecmed==196){
            msg2=" Afinado";
        }
        else if((int)frecmed>196){
            msg2= " Afloja";
        }
        else if((int)frecmed<196){
            msg2= " Tensar";
        }

    }
    else if (cuerda==5){
        if((int)frecmed==246){
            msg2=" Afinado";
        }
        else if((int)frecmed>246){
            msg2= " Afloja";
        }
        else if((int)frecmed<246){
            msg2= " Tensar";
        }

    }
    else if (cuerda==6){
        if((int)frecmed==329){
            msg2=" Afinado";
        }
        else if((int)frecmed>329){
            msg2= " Afloja";
        }
        else if((int)frecmed<329){
            msg2= " Tensar";
        }
    }

    if(t_lcd>=7){
        //PANTALLA LCD
        if(cuerda>=1 && cuerda<=6){
            lcd_clear();
            lcd_puts("Frecuencia:");
            lcd_goto(0,1);
            sprintf(pantalla,"%3d", (int)frecmed);
            lcd_puts(pantalla);
            lcd_puts(" Hz");
            lcd_puts(msg2);
        }
        else if(cuerda==0){
            lcd_clear();
            lcd_puts("Frecuencia:");
            lcd_goto(0,1);
            sprintf(pantalla,"%3d", (int)frecmed);
```

```
                    lcd_puts(pantalla);
                    lcd_puts(" Hz");
                }

                t_lcd=0;
            }
            if (cuerda == -1){
                lcd_clear();
                lcd_puts("AFINADOR DE ");
                lcd_goto(0,1);
                lcd_puts("GUITARRA");
            }
        }
        while(metodo==2){ //goertzel
        if (buffer_full) {
            buffer_full = 0;
            if(cuerda==0){
            analyze_frequencies(target_freqs);
            // Find the frequency with the maximum magnitude
            max_magnitude = 0.0;

            int i;
            for (i = 0; i < NUM_FREQS; i++) {
                if (magnitudes[i] > max_magnitude) {
                    max_magnitude = magnitudes[i];
                    max_index = i;
                }
            }
            }
            if(magnitudes[max_index]>2000 || cuerda!=0){
                   if(cuerda==1 || max_index==0){
                       afina_cuerda(cuerda1);
                       detect="Primera cuerda";
                       if (frecuencia>82){
                           afina=" Afloja";
                       }else if (frecuencia<82){
                           afina=" Tensar";
                       }else if (frecuencia==82){
                           afina=" Afinado";
                       }
                       GpioDataRegs.GPASET.bit.GPIO17 = 1;    //enciende led
conectado al pin 17
                       GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;  //apaga led conectado
al pin 19
                       GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;  //apaga led conectado
al pin 21
                       GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;  //apaga led conectado
al pin 23
                       GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;  //apaga led conectado
al pin 27
                       GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;  //apaga led conectado
al pin 31
                   }
                   else if(cuerda==2 || max_index==1){
                       afina_cuerda(cuerda2);
                       detect="Segunda cuerda";
                       if (frecuencia>110){
                           afina=" Afloja";
                       }else if (frecuencia<110){
                           afina=" Tensar";
```

```
        }else if (frecuencia==110){
            afina=" Afinado";
        }
        GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
        GpioDataRegs.GPASET.bit.GPIO19 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;

    }
    else if(cuerda==3 || max_index==2){
        afina_cuerda(cuerda3);
        detect="Tercera cuerda";
        if (frecuencia>146.83){
            afina=" Afloja";
        }else if (frecuencia<146.83){
            afina=" Tensar";
        }else if (frecuencia==146.83){
            afina=" Afinado";
        }
        GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
        GpioDataRegs.GPASET.bit.GPIO21 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
    }
    else if(cuerda==4 || max_index==3){
        afina_cuerda(cuerda4);
        detect="Cuarta cuerda";
        if (frecuencia>196){
            afina=" Afloja";
        }else if (frecuencia<196){
            afina=" Tensar";
        }else if (frecuencia==196){
            afina=" Afinado";
        }
        GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
        GpioDataRegs.GPASET.bit.GPIO23 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
    }
    else if(cuerda==5 || max_index==4){
         if(magnitudes[max_index]>6000 || cuerda!=0){
        afina_cuerda(cuerda5);

        detect="Quinta cuerda";
        if (frecuencia>246.94){
            afina=" Afloja";
        }else if (frecuencia<246.94){
            afina=" Tensar";
        }else if (frecuencia==246.94){
            afina=" Afinado";
        }
        GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
```

```
                    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
                    GpioDataRegs.GPASET.bit.GPIO27 = 1;
                    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
                    }
                }
                else if(cuerda==6 || max_index==5){
                    if(magnitudes[max_index]>8000 || cuerda!=0){
                    afina_cuerda(cuerda6);
                    detect="Sexta cuerda";
                    if (frecuencia>329.63){
                        afina=" Afloja";
                    }else if (frecuencia<329.63){
                        afina=" Tensar";
                    }else if (frecuencia==329.63){
                        afina=" Afinado";
                    }

                    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
                    GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
                    GpioDataRegs.GPACLEAR.bit.GPIO21 = 1;
                    GpioDataRegs.GPACLEAR.bit.GPIO23 = 1;
                    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
                    GpioDataRegs.GPASET.bit.GPIO31 = 1;
                    }
                }
            }

             // Transmisión por puerto serie
                    if(SciaRegs.SCIFFRX.bit.RXFFST == 1) {

                        scia_msg("\e[1;1H\e[2J");
                        msg = "\r\nSelecciona la cuerda: \0";
                        scia_msg(msg);

                        ReceivedChar = SciaRegs.SCIRXBUF.all;
                        scia_xmit(ReceivedChar);

                    if(ReceivedChar == '1'){
                        scia_msg(" (82Hz)");
                        msg = "\r\nEstas afinando la primera cuerda(la mas
gruesa)\n\0";
                        cuerda=1;
                        max_index=0;

                    }else if(ReceivedChar == '2'){
                        scia_msg(" (110Hz)");
                        msg = "\r\nEstas afinando la segunda cuerda \n\0";
                        cuerda=2;
                        max_index=1;

                    }else if(ReceivedChar == '3'){
                        scia_msg(" (146Hz)");
                        msg = "\r\nEstas afinando la tercera cuerda \n\0";
                        cuerda=3;
                        max_index=2;

                    }else if(ReceivedChar == '4'){
                        scia_msg(" (196Hz)");
                        msg = "\r\nEstas afinando la cuarta cuerda \n\0";
                        cuerda=4;
```

```
                    max_index=3;

                }else if(ReceivedChar == '5'){
                    scia_msg(" (246Hz)");
                    msg = "\r\nEstas afinando la quinta cuerda \n\0";
                    cuerda=5;
                    max_index=4;

                }else if(ReceivedChar == '6'){
                    scia_msg(" (329Hz)");
                    msg = "\r\nEstas afinando la sexta cuerda(la mas delgada)
\n\0";
                    cuerda=6;
                    max_index=5;

                }else if(ReceivedChar == '0'){
                    msg = "\r\nAfinacion manual \n\0";
                    msg3 = "\r\n\n\0";
                    cuerda=0;
                    max_index=0;

                }else{
                    metodo=0;
                    cuerda=-1;
                    max_index=0;
                    menu_inicio();
                }

                if(metodo!=0){
                scia_msg(msg);
                scia_msg(msg3);

                msg = "\r\nSelecciona la cuerda: \0";
                scia_msg(msg);
                }

                LoopCount++;
                }

                if (cuerda==0){
                    lcd_clear();
                    lcd_puts(detect);
                    lcd_goto(0,1);
                    sprintf(pantalla,"%3d", (int)frecuencia);
                    lcd_puts(pantalla);
                    lcd_puts(" Hz");
                }

                if(cuerda!=0){
                    //PANTALLA LCD
                    lcd_clear();
                    lcd_puts("Frecuencia:");
                    lcd_goto(0,1);
                    sprintf(pantalla,"%3d", (int)frecuencia);
                    lcd_puts(pantalla);
                    lcd_puts(" Hz");
                    lcd_puts(afina);
                }
                if (cuerda==-1){
                    lcd_clear();
```

```
                        lcd_puts("AFINADOR DE ");
                        lcd_goto(0,1);
                        lcd_puts("GUITARRA");
                    }
        }


        }

    } //cierro while
} //cierro main

void Gpio_select(void)
{
    EALLOW;
    //CUERDA 1
    GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0;    // GPIO17 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO17 = 1;     // GPIO17 como salida: LED
    //CUERDA 2
    GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 0;    // GPIO19 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO19 = 1;     // GPIO19 como salida: LED
    //CUERDA 3
    GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0;    // GPIO21 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO21 = 1;     // GPIO21 como salida: LED
    //CUERDA 4
    GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;    // GPIO23 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO23 = 1;     // GPIO23 como salida: LED
    //CUERDA 5
    GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 0;    // GPIO29 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO27 = 1;     // GPIO29 como salida: LED
    //CUERDA 6
    GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;    // GPIO31 = General Purpose I/O
    GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;     // GPIO31 como salida: LED

    EDIS;

}

void InitAdcRegs(void)
{
    AdcRegs.ADCTRL1.all = 0;
    AdcRegs.ADCTRL1.bit.ACQ_PS = 1;         // Sample window = 2*(1/ADCCLK)
    AdcRegs.ADCTRL1.bit.SEQ_CASC = 0;       // Modo dual
    AdcRegs.ADCTRL1.bit.CPS = 1;            // CPS = 2 --> ADCCLK = FCLK/(CPS+1)
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0;       // Modo start-stop

    AdcRegs.ADCTRL2.all = 0;
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;   // Se habilita interrupcion SEQ1INT
    AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;   // Modo de interrupcion cada EOS

    AdcRegs.ADCTRL3.bit.SMODE_SEL = 0;      // Modo muestreo secuencial
    AdcRegs.ADCTRL3.bit.ADCCLKPS = 1;       // ADCCLKPS = 3 --> FCLK = HSPCLK / 2 *
ADCCLKPS
                                            // HSPCLK = 75 MHz
                                            // FCLK = 37.5 MHz
                                            // ADCCLK = 12.5 MHz

    // Una sola medida
    AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 0;   // 1 conversion por secuencia
```

```
    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;     // CONVV00 = ACDINA0.

}

__attribute__((ramfunc))
interrupt void cpu_timer0_isr(void)
{

    if(metodo==1){
    if(start==1){
           tint++;
           t = tint * timer*pow(10,-6);
       }

       inc_lcd++;
           if(inc_lcd>=5000){
               t_lcd++;
               inc_lcd=0;
           }
}

    // cuando se activa el timer, empieza a convertir el CAD
    AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;        // Activacion de conversion ADC por
software

    PieCtrlRegs.PIEACK.bit.ACK1 = 1;         // Clear PIEACK para habilitar siguiente
interrupcion
}


__attribute__((ramfunc))
interrupt void  adc_SEQ1_isr(void)
{
    if(metodo==1){
        Vbin = AdcMirror.ADCRESULT0;
    }

    if(metodo==2){
        if (sample_index < N) {
            Vbin = AdcMirror.ADCRESULT0;       // Almacenaje de resultados medidos
            samples[sample_index++] = (double)(Vbin - ADC_MID_VALUE); // Center the
value around 0

            if (sample_index >= N) {
                buffer_full = 1;
                sample_index = 0;
            }
        }
    }

    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;        // Reset SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;      // Clear INT_SEQ1 para habilitar la
siguiente interrupcion
    PieCtrlRegs.PIEACK.bit.ACK1 = 1;         // Clear PEACK  habilitar siguiente
interrupcion
}

void goertzel_init(double *cosine, double *sine, double *coeff, int k)
{
    double omega = (2.0 * M_PI * k) / N;
```

```
    *cosine = cos(omega);
    *sine = sin(omega);
    *coeff = 2.0 * (*cosine);
}

double goertzel_filter(double *samples, double coeff, double cosine, double sine)
{
    double s_prev = 0.0;
    double s_prev2 = 0.0;
    double s;

    for (i = 0; i < N; i++) {
        s = samples[i] + coeff * s_prev - s_prev2;
        s_prev2 = s_prev;
        s_prev = s;
    }

    double real_part = s_prev - cosine * s_prev2;
    double imag_part = sine * s_prev2;
    return sqrt(real_part * real_part + imag_part * imag_part);
}

void analyze_frequencies(double *target)
{
    double cosine, sine, coeff;
    int j;
    for (j = 0; j < NUM_FREQS; j++) {
        int k = (int)(0.5 + ((N * target[j]) / SAMPLE_RATE));  // Target frequency
index
        goertzel_init(&cosine, &sine, &coeff, k);
        magnitudes[j] = goertzel_filter(samples, coeff, cosine, sine);
    }
}

void afina_cuerda(double *target)
{
    max_magnitude = 0.0;
    analyze_frequencies(target);
    int j;
    for (j = 0; j < NUM_FREQS; j++) {
        if (magnitudes[j] > max_magnitude) {
            max_magnitude = magnitudes[j];
            frecuencia=target[j];
        }
    }
}

//===========================================================================
// End of SourceCode.
//===========================================================================
```

ANEXO B: LIBRERÍAS ADICIONALES

7.2 Librería para el periférico SCI

Código del SCI:

```
#include "sci.h"

void menu_inicio()
{

    scia_msg("\e[1;1H\e[2J"); //limpiamos pantalla

    // Enviamos toda la cadena //
    scia_msg("\r\n\n\nMenu afinador de guitarra\0"); // Cadena de bienvenida


    scia_msg("\r\n1--> Metodo del cruce por cero.\n\0");
    scia_msg("\r\n2--> Metodo del algoritmo de Goertzel.\n\0");

    scia_msg("\r\nElige el metodo de afinacion.\n\0");

}

void scia_echoback_init()
{
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function

    SciaRegs.SCICCR.all =0x0007;   // 1 stop bit,  No loopback
                                   // No parity, 8 char bits,
                                   // async mode, idle-line protocol
    SciaRegs.SCICTL1.all =0x0003;  // enable TX, RX, internal SCICLK,
                                   // Disable RX ERR, SLEEP, TXWAKE
    SciaRegs.SCICTL2.all =0x0003;
    SciaRegs.SCICTL2.bit.TXINTENA =1;
    SciaRegs.SCICTL2.bit.RXBKINTENA =1;

    SciaRegs.SCIHBAUD    =0x0001;  // 9600 baud @LSPCLK = 37.5MHz.
    SciaRegs.SCILBAUD    =0x00E7;


    SciaRegs.SCICTL1.all =0x0023;  // Relinquish SCI from Reset
}

// Funcion para mandar por la UART un caracter.
void scia_xmit(int a)
{
    // Nos esperamos hasta que la FIFO de salida se haya procesado entera
    while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
    // Cuando la fifo de salida se haya trasmitido, escribimos un caracter en el
buffer de trasmision.
    SciaRegs.SCITXBUF=a;
}
```

```
// Rutina para transmitir caracter por caracter una cadena de caracteres
void scia_msg(char * msg)
{
    int i;
    i = 0;
    while(msg[i] != '\0')
    {
        scia_xmit(msg[i]);
        i++;
    }
}

// Inicializacion de la FIFO del modulo SCI
void scia_fifo_init()
{
    SciaRegs.SCIFFTX.all=0xE040;
    SciaRegs.SCIFFRX.all=0x204f;
    SciaRegs.SCIFFCT.all=0x0;
}

//función para transmitir la variable de la frecuencia calculada en el programa
void transmite_frecuencia(float frecmed){
unsigned int digito=0;
char *dato=0;
int i=0;

for(i=0;i<3;i++){
    dato[i]=0+0x30;
}


if(frecmed>100){
    digito=(unsigned int)(frecmed/100);
    frecmed = frecmed - digito*100;
    dato[0]=digito+0x30;
}
if(frecmed>10){
    digito=(unsigned int)(frecmed/10);
    frecmed = frecmed - digito*10;
    dato[1]=digito+0x30;
}
if(frecmed>0){
    digito=(unsigned int)(frecmed/1);
    dato[2]=digito+0x30;
}
scia_msg("\rFrecuencia: \0");
for(i=0;i<3;i++){
    scia_xmit(dato[i]);
}
scia_msg("Hz\n\0");
}
```

7.3 Librería para la pantalla LCD

```
#include "LCD.h"

// Local variables
const uint16_t rowaddr[4] = {0x00, 0x40, 0x14, 0x54};
uint16_t lcdrows = 2;
uint16_t lcdcolumns = 16;
// Local copy of the Display on off control register
uint16_t dispctrl = 0x00;
uint16_t iomode = 0;

uint16_t lcd_init(void * iodata, uint16_t cols, uint16_t rows)
{
    // Initialize IO pins
    iomode = lcd_ioinit();
    lcdrows = rows;
    lcdcolumns = cols;

    // Initial delay after power-up
    delay_us(100000);

    //  Check if LCD interface is 8 or 4 bit mode
    if (iomode == 4) {
        // Begin LCD controller Initialization (HD44780 page 45-46)
        // Put a loop here to send these three?
        lcd_iowrite4(0x03);
        delay_us(100000);
        lcd_iowrite4(0x03);
        delay_us(120); //120
        lcd_iowrite4(0x03);
        delay_us(120);
        lcd_iowrite4(0x02);
        delay_us(120);
        lcd_command(E_FUNCTION_SET | BIT_DL_DATALENGTH_4 | BIT_N_DISP_LINES_2 |
BIT_F_FONT_5_10);
        delay_us(50);
    } else if (iomode == 8) {
        // Begin LCD controller Initialization (HD44780 page 45-46)
        lcd_command(E_FUNCTION_SET);
        delay_us(5000);
        lcd_command(E_FUNCTION_SET);
        delay_us(120);
        lcd_command(E_FUNCTION_SET);
        delay_us(120);
        lcd_command(E_FUNCTION_SET | BIT_DL_DATALENGTH_8 | BIT_N_DISP_LINES_2 |
BIT_F_FONT_5_10);
        delay_us(50);
    }
    // Configure display after power up
    lcd_command(E_DISPLAY_ON_OFF_CTRL | BIT_D_DISPLAY_OFF);
    delay_us(50);
    lcd_command(E_CLEAR_DISPLAY);
    delay_us(2*micro);
```

```
    lcd_command(E_ENTRY_MODE_SET | BIT_S_AUTOSCROLL_OFF | BIT_ID_INCREMENT_CURSOR);
    delay_us(50);

    return TRUE;
}

void delay_us(long end)
{
    long i;
    for (i = 0; i < end; i++)
    {
        asm ("NOP:");
    }
}


void lcd_clear()
{
    lcd_command(E_CLEAR_DISPLAY);
    delay_us(25000);
}


void lcd_home()
{
    lcd_command(E_RETURN_HOME);
    delay_us(2000);
}

void lcd_on()
{
    dispctrl |= BIT_D_DISPLAY_ON;
    lcd_command(E_DISPLAY_ON_OFF_CTRL | dispctrl);
    delay_us(50);
}

void lcd_off()
{
    dispctrl &= ~BIT_D_DISPLAY_ON;
    lcd_command(E_DISPLAY_ON_OFF_CTRL | dispctrl);
    delay_us(50);
}

void lcd_cursor(enum enLCDCursorModes mode)
{
    dispctrl &= 0xFC;
    dispctrl |= mode;
    lcd_command(E_DISPLAY_ON_OFF_CTRL | dispctrl);
    delay_us(50);
}

void lcd_cursor_left()
{
    lcd_command(E_CURSOR_DISPLAY_SHIFT | BIT_SC_SHIFT_CURSOR | BIT_RL_SHIFT_LEFT);
    delay_us(50);
}

void lcd_cursor_right()
{
    lcd_command(E_CURSOR_DISPLAY_SHIFT | BIT_SC_SHIFT_CURSOR | BIT_RL_SHIFT_RIGHT);
```

```
    delay_us(50);
}

void lcd_scroll_left()
{
    lcd_command(E_CURSOR_DISPLAY_SHIFT | BIT_SC_SHIFT_DISPLAY | BIT_RL_SHIFT_LEFT);
    delay_us(50);
}

void lcd_scroll_right()
{
    lcd_command(E_CURSOR_DISPLAY_SHIFT | BIT_SC_SHIFT_DISPLAY | BIT_RL_SHIFT_RIGHT);
    delay_us(50);
}

void lcd_autoscroll_on()
{
    lcd_command(E_ENTRY_MODE_SET | BIT_S_AUTOSCROLL_ON | BIT_ID_INCREMENT_CURSOR);
    delay_us(50);
}

void lcd_autoscroll_off()
{
    lcd_command(E_ENTRY_MODE_SET | BIT_S_AUTOSCROLL_OFF | BIT_ID_INCREMENT_CURSOR);
    delay_us(50);
}

void lcd_goto(uint16_t col, uint16_t row)
{
    // Apply limits for Rows and Columns
    if (row >= lcdrows)
        row = lcdrows - 1;
    if (col >= lcdcolumns)
        col = lcdcolumns - 1;

    lcd_command(E_SET_DDRAM_ADDR | (col + rowaddr[ row ]));
}

void lcd_send(uint16_t data, uint16_t rs)
{
    // Write logic one to send characters or logic 0 to send a command
    if (rs)
        lcd_iohigh(E_RS_PIN);
    else
        lcd_iolow(E_RS_PIN);
    // Clear the RW pin if used
    lcd_iolow(E_RW_PIN);

    if (iomode == 4) {
        lcd_iowrite4(data >> 4);
        lcd_iowrite4(data);
    }

    //delay_us(8000);
}

void lcd_puts(const char * string)
{
    while (*string != '\0')
        lcd_write(*string++);
```

```
}



void lcd_create_char(uint16_t charnum, const uint16_t * chardata)
{
    uint16_t i;
    charnum &= 0x07;
    lcd_command(E_SET_CGRAM_ADDR | (charnum << 3));
    for (i = 0; i < 8; i++)
        lcd_write(chardata[i]);
}

uint16_t lcd_ioinit(void)
{
    uint16_t i = 0;

    EALLOW;
    // Configure control lines as outputs
    GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO61 = 1;

    GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO63 = 1;

    // Configure bus data lines as outputs
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO7 = 1;

    GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;

    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;

    GpioCtrlRegs.GPBMUX2.bit.GPIO49 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;

    EDIS;

    // Set all the pins to "low" state
    for (i = 0; i < 11; i++)
        lcd_ioset(i, FALSE);

    // Return bus length
    return 4;

}

void lcd_ioset(enum enLCDControlPins pin, uint16_t out)
{
    switch (pin) {
    case E_D4_PIN:
        // si out es distinto de False, se pone el pin a 1; en caso contrario se pone
a 0.
        if(out!=0)
            GpioDataRegs.GPASET.bit.GPIO7=1;
        else
            GpioDataRegs.GPACLEAR.bit.GPIO7=1;
```

```
            break;
        case E_D5_PIN:
            if(out!=0)
                GpioDataRegs.GPASET.bit.GPIO9=1;
            else
                GpioDataRegs.GPACLEAR.bit.GPIO9=1;
            break;
        case E_D6_PIN:
            if(out!=0)
                GpioDataRegs.GPASET.bit.GPIO11=1;
            else
                GpioDataRegs.GPACLEAR.bit.GPIO11=1;
            break;
        case E_D7_PIN:
            if(out!=0)
                GpioDataRegs.GPBSET.bit.GPIO49=1;
            else
                GpioDataRegs.GPBCLEAR.bit.GPIO49=1;
            break;
        case E_RS_PIN:
            if(out!=0)
                GpioDataRegs.GPBSET.bit.GPIO63=1;
            else
                GpioDataRegs.GPBCLEAR.bit.GPIO63=1;
            break;
        case E_EN_PIN:
            if(out!=0)
                GpioDataRegs.GPBSET.bit.GPIO61=1;
            else
                GpioDataRegs.GPBCLEAR.bit.GPIO61=1;
            break;
        }
}

void lcd_iowrite4(uint16_t data)
{
    uint16_t i;

    for (i = 4; i < 8; i++)
        lcd_ioset(i,( (data & (1 << i - 4))!=0) ? TRUE : FALSE);

    lcd_iohigh(E_EN_PIN);
    delay_us(1500);
    lcd_iolow(E_EN_PIN);
}
```