UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo e Implementación de un Sintetizador Digital de Audio Empleando Metodologías Aditivas y Sustractivas

Trabajo de graduación presentado por Manuel Antonio Valenzuela Herrera para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2019

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo e Implementación de un Sintetizador Digital de Audio Empleando Metodologías Aditivas y Sustractivas

Trabajo de graduación presentado por Manuel Antonio Valenzuela Herrera para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

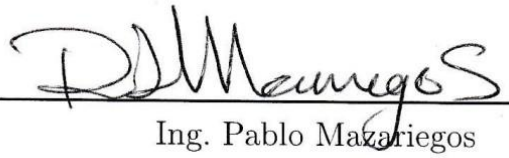
2019

Vo.Bo.:

(f) 

MSc. Miguel Zea

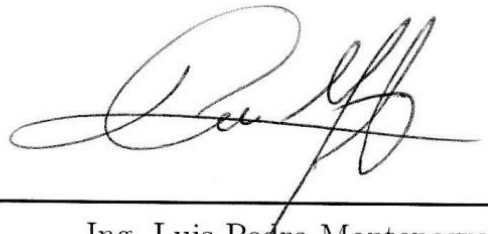
Tribunal Examinador:

(f) 

Ing. Pablo Mazariegos

(f) 

Ing. Kurt Kellner

(f) 

Ing. Luis Pedro Montenegro

Fecha de aprobación: Guatemala, 5 de diciembre de 2019.

Prefacio

En primer lugar quiero agradecer a Dios por darme la oportunidad de estudiar en esta universidad. Gracias a la Universidad del Valle de Guatemala por darme la oportunidad y las herramientas para dar un inicio a mi formación profesional en una rama que me apasiona.

Durante toda la carrera tuve a muchas personas que me ayudaron y apoyaron a poder hacer realidad este trabajo y mi sueño de ser ingeniero. Quiero agradecer a mis padres por todo su esfuerzo y sacrificio que hacen de este momento una realidad. A mis abuelos quienes siempre me apoyaron, en especial, a mi abuela Clara Luz Morales, quién deseaba incluso más que yo que finalizara mis estudios. Ojalá estuvieras aquí para ver en lo que se convirtió tu nieto. A mis tíos y hermanos por acompañarme durante toda esta experiencia. A Anahí Campos por su amor, comprensión y apoyo durante toda la carrera. Y así mismo quiero agradecer a mis maestros por compartir sus conocimientos, especialmente a Miguel Zea por asesorarme durante este proyecto.

Índice

Lista de figuras

Lista de cuadros

Resumen

El objetivo principal de este trabajo fue desarrollar un hardware que implemente algoritmos de síntesis aditiva y sustractiva y efectos de audio digitales que permitan modular los parámetros de diseño al gusto del usuario en tiempo real, lo cual crea una alternativa más fácil de transportar, instalar y utilizar en presentaciones en vivo a la configuración usual de utilizar un controlador *MIDI* en conjunto con una interfaz de audio y una computadora portátil. En el capítulo 7 se expone la metodología que se utilizó para seleccionar el hardware y software utilizado posteriormente para el desarrollo del sintetizador. En el capítulo 8 se muestra detalladamente el diseño del motor de síntesis y la interfaz de usuario. En el capítulo 9 se muestran los pasos detallados que se siguieron para desarrollar el sintetizador de audio en base al diseño planteado en el capítulo 8 y, por último, en el capítulo 10 se muestran los resultados obtenidos al realizar distintas pruebas al prototipo final que se obtuvo al seguir la metodología descrita en el capítulo 9. Según a la evidencia en los capítulos anteriormente mencionados se concluye que se logró desarrollar un hardware que implementa algoritmos de síntesis aditiva y sustractiva y efectos de audio digitales que permitan modular los parámetros de diseño al gusto del usuario en tiempo real.

Abstract

The main objective of this work was to develop a hardware that implements additive and subtractive synthesis algorithms and digital audio effects that allow modulating the design parameters to the user's taste in real time, which creates an easier alternative to transport, install and use in live presentations to the usual settings of using a *MIDI* controller in conjunction with an audio interface and a laptop. Chapter 7 describes the methodology that was used to select the hardware and software used later for the development of the synthesizer. Chapter 8 shows in detail the design of the synthesis engine and the user interface. Chapter 9 shows the detailed steps that were followed to develop the audio synthesizer based on the design outlined in chapter 8 and finally, in chapter 10 the results obtained by performing different tests on the final prototype that was obtained are shown by following the methodology described in chapter 9. Based on the evidence in the aforementioned chapters, it is concluded that it was possible to develop hardware that implements additive and subtractive synthesis algorithms and digital audio effects that allow modulating design parameters to taste of the user in real time.

CAPÍTULO 1

Introducción

En la industria musical se busca innovar y utilizar sonidos diferentes con los cuales se pueda experimentar y crear. Un sintetizador es un dispositivo que utiliza distintas técnicas para generar un sonido, este sonido pasa por varias etapas para ser acondicionado y procesado al gusto del usuario. En este trabajo se buscó desarrollar un sintetizador de audio explorando distintas metodologías que produzcan sonidos únicos, implementar una serie de efectos digitales que enriquezcan la musicalidad de este sonido y empaquetar esto en un diseño que sea versátil y portátil para que pueda ser utilizado en cualquier lugar.

El objetivo general del proyecto fue desarrollar un hardware que implemente algoritmos de síntesis aditiva y sustractiva y efectos de audio digitales que permitan modular los parámetros de diseño al gusto del usuario en tiempo real. Para lograr este objetivo general se tuvieron los siguientes objetivos específicos: Analizar el desempeño de algoritmos de síntesis aditiva y sustractiva implementadosen un procesador ARM Cortex-A53 de 64-bit; seleccionar y analizar una serie de efectos digitales que modifiquen el espectro enfrecuencia de la señal obtenida mediante los modelos de síntesis; y, por último, plantear un diseño físico portátil que pueda ser utilizado para presentaciones musicales en vivo y permita modificar los parámetros de síntesis y efectos digitales en tiempo real.

Este trabajo se divide en las siguientes partes:

- Descripción de una serie de pruebas preliminares las cuales guiaron a la metodología final.
- Descripción detallada del diseño un motor de síntesis implementable en *Hardware* de bajo poder computacional.
- Descripción detallada del desarrollo e implementación del motor de síntesis diseñado.
- Análisis de los resultados obtenidos de la implementación del motor de síntesis diseñado.

En el capítulo 7 se exponen las pruebas preliminares, en su mayoría fallidas, las cuales sirvieron para escoger el hardware y software que se estaría utilizando para realizar el diseño y la implementación. En el capítulo 8 se muestra detalladamente el diseño del motor de síntesis y la interfaz de usuario. En el capítulo 9 se muestran los pasos detallados que se siguieron para desarrollar el sintetizador de audio en base al diseño planteado en el capítulo 8 y por último, en el capítulo 10 se muestran los resultados obtenidos al realizar distintas pruebas al prototipo final que se obtuvo al seguir la metodología descrita en el capítulo 9.

CAPÍTULO 2

Antecedentes

Cuando la tecnología avanza esta va afectando y mejorando la forma de trabajar en cualquier ámbito. La industria musical no fue la exepción. Desde la década de los años 60 se fueron creando instrumentos que no generaban un sonido por medios de percusión, cuerdas o viento; en lugar de esto utilizaban algún sistema eléctrico u algoritmo para generar un oscilador, hacer vibrar una bocina y generar un sonido que no se encuentra en la naturaleza. Estos instrumentos son los sintetizadores de audio, los cuales han revolucionado el sonido y la forma de hacer música desde que fueron intruducidos. A lo largo del tiempo se han desarrollado distintas técnicas y enfoques para generar un sonido. Algunas para emular un sonido de la naturaleza, otras para crear uno completamente nuevo. Los primeros sintetizadores fueron analógicos, estos generan varios osciladores por medio de circuitos eléctricos. La desvantaja de estos es que son muy grandes debido a la cantidad de componentes eléctricos y electrónicos que necesitan, además de que la mayoría de estos se cablean por fuera para enrutar una señal. Luego, con los avances en la electrónica digital se desarrollaron sintetizadores con un enfoque más moderno, generar sonido por medio de software. Esto presenta una gran ventaja ante su contraparte analógica, ya que permite convertir casi cualquier dispositivo de computo en un sintetizador, siempre y cuando el poder computacional de este sea suficiente, además de reducir considerablemente el tamaño de los dispositivos de síntesis. En la actualidad en la mayoría de sonidos son generados o procesados por computadoras por medio de algún software.

Es importante tener presente que para construir un sintetizador de audio hay varios factores a considerar más allá del algorítmo o la metodología para crear sonido. Ya que este trabajo busca desarrollar un sintetizador puramente digital se consideraran solo los factores relevantes para este enfoque. Quizás el factor más importante al construir un sintetizador digital es la plataforma en donde van a correr los algorítmos de síntesis. Esta plataforma es un limitante a los algorítmos de síntesis ya que algunos son computacionalmente muy demandantes. Otro factor importante es la aplicación final del sintetizador, este va de la mano con la elección de la plataforma ya que los requerimientos de síntesis en tiempo real no son los mismos de la síntesis off line.

A continuación se muestran algunos antecedentes que tienen información relevante, útil y que sientan las bases de este trabajo, tanto en la parte de síntesis como la de implementación.

2.1. Sintetizador modular

En [1] se muestra un sintetizador modular en el cual se implementa en un diseño híbrido que incorpora tanto la implementación de hardware como la implementación de software. El usuario ensambla y conecta varios módulos de forma hexagonal para configurar una presentación visual de la generación de señales del sintetizador y las funciones de procesamiento de señales. El sintetizador replica visualmente el hardware incorporando controles de usuario en muchos de los módulos que están conectados a placas de circuito encerradas (integradas). Estas tarjetas de circuitos se comunican con una CPU del sistema (unidad de procesamiento de computadora) que opera el software softsynth residente dentro de la CPU para controlar la salida de audio. La configuración del softsynth está determinada por la disposición física de los módulos de hardware hexagonales que representan la funcionalidad de los módulos de software. Los módulos de hardware proporcionan elementos de la interfaz de usuario correspondientes a los parámetros de sus contrapartes de softsynth.

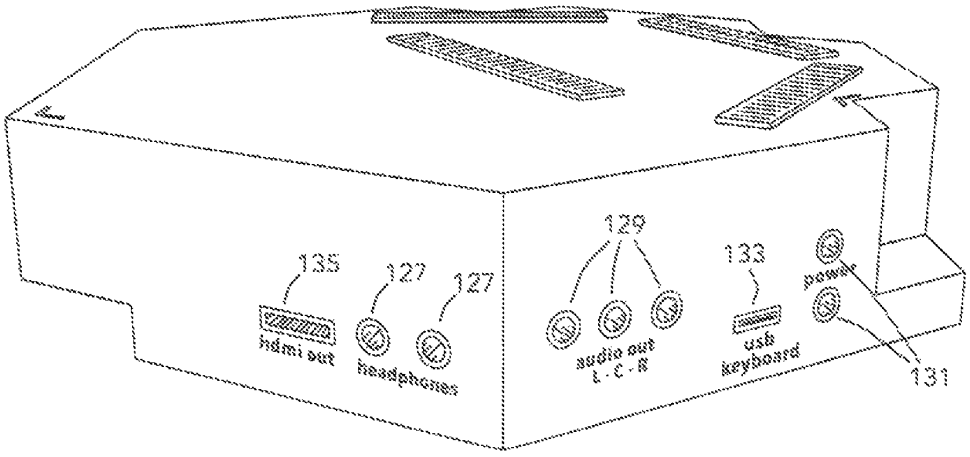


Figura 1: Diseño del sintetizador modular

2.2. Sintetizador de música digital FPGA

El proyecto que se muestra en [2] comprendió el desarrollo de un sintetizador de música digital capaz de realizar síntesis sustractivas con varias etapas de procesamiento de efectos de audio. El sistema se implementó en el Zedboard, una placa de desarrollo con un sistema en chip Xilinx Zynq (SoC), un dispositivo multifuncional que cuenta con un microprocesador ARM de doble núcleo y una lógica de matriz de puertas programable en campo (FPGA). El sintetizador completado fue capaz de producir una amplia variedad de tonos musicales, muchos de los cuales fueron completamente controlables por el usuario.

El control general del sistema fue proporcionado por el software integrado en tiempo real

que se ejecuta en el microprocesador ARM. Los módulos de generación de formas de onda y efectos de sintetizador se diseñaron en Verilog y se implementaron en lógica personalizada utilizando técnicas de procesamiento de señales digitales de punto fijo.

Figura 2: Implementación del sintetizador en un FPGA

2.3. PDA: Procesamiento de señales en tiempo real y generación de sonido en los dispositivos portátiles

En [3] se describe un puerto de Pure Data, esta vez no para un nuevo sistema operativo sino para un las computadoras PocketPC. El documento describe las capacidades de estos pequeños dispositivos de mano en la actualidad y luego describe diferentes aspectos del sistema de software derivado de Pure Data y llamado PDa. Es importante hacer énfasis en que este trabajo es del año 2003 por lo que la plataforma utilizada es obsoleta, sin embargo, muestra el desempeño de algorítmos de Síntesis FM en un dispositivo de procesamiento limitado.

Figura 3: Implementación de PDa en un PocketPC

2.4. Implementación de efectos de audio digitales mediante un enfoque de diseño de Hardware/Software

En [4] se menciona que los efectos de audio digital en tiempo real se realizan en software en casi todos los casos. Las plataformas de hardware utilizadas para este propósito abarcan desde procesadores multipropósito como la clase Intel Pentium sobre procesadores integrados (por ejemplo, la familia ARM) hasta DSP especializado. La próxima tecnología de sistemas completos en un solo chip programable contrasta con una solución centrada en el software, ya que combina software y hardware a través de alguna metodología de co-diseño y lo convierte en una alternativa prometedora para el futuro del audio en tiempo real.

En este trabajo se describe un ejemplo para un sistema de efectos digitales en tiempo real que fue desarrollado utilizando un método de co-diseño de hardware / software. Mientras que el procesamiento de audio digital en tiempo real se realiza en unidades de hardware dedicadas de baja latencia, el control y enrutamiento de las transmisiones de audio se realiza mediante un software que se ejecuta en un procesador de núcleo suave NIOS II de 32 bits. La implementación de las unidades de hardware se realiza mediante una metodología centrada en DSP para elevar el nivel de abstracción de las descripciones de VHDL sin dejar de utilizar las herramientas de síntesis de FPGA estándar de la plataforma. La implementación física del sistema completo utiliza una placa de creación rápida de prototipos adaptada para las comunicaciones y las aplicaciones de audio basadas en un FPGA Altera Cyclone II.

CAPÍTULO 3

Justificación

En los años 60 se empezó a experimentar un nuevo tipo de música con sonidos que no se encuentran en la naturaleza; algo nuevo para el ser humano. No puede negarse el impacto e influencia que tiene la música en la sociedad. A medida que la tecnología va avanzando, la manera en la que se hacían las cosas en todo ámbito van cambiando, y la música no es la excepción. Los sintetizadores digitales de audio permiten desarrollar sonidos mediante algoritmos que no están limitados a ninguna implementación física, dando rienda suelta a la imaginación y abriendo nuevas posibilidades de innovación en el ámbito musical.

Este trabajo pretende ser la primera iteración a un producto final que permita a cualquier usuario diseñar el sonido que tenga en su imaginación. Se busca implementar algorítmos de síntesis en hardware de bajo poder computacional y por ende, bajo costo, lo cual amplía el alcance en social de este producto. Además abrirá puertas para que se inicie en Guatemala el desarrollo de hardware de audio de bajo costo y alta calidad, industria que actualmente no existe en Guatemala.

Para que todo esto sea posible se necesita aplicar un conocimiento avanzado de procesamiento de señales, electrónica digital, programación y otras áreas de matemática, física e ingeniería. Se tomará como inspiración las ideas presentadas en [4] y [2], se continuará la línea de investigación presentada en [3] utilizando un hardware más moderno pero teniendo en mente llegar a un dispositivo que pueda ser utilizado para presentaciones en vivo.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar un hardware que implemente algoritmos de síntesis aditiva y sustractiva y efectos de audio digitales que permitan modular los parámetros de diseño al gusto del usuario en tiempo real.

4.2. Objetivos específicos

- Analizar el desempeño de algoritmos de síntesis aditiva y sustractiva implementados en un procesador ARM Cortex-A53 de 64-bit
- Seleccionar y analizar una serie de efectos digitales que modifiquen el espectro en frecuencia de la señal obtenida mediante los modelos de síntesis.
- Plantear un diseño físico portátil que pueda ser utilizado para presentaciones musicales en vivo y permita modificar los parámetros de síntesis y efectos digitales en tiempo real.

CAPÍTULO 5

Alcance

Dentro de este trabajo se busca sentar las bases para desarrollar el prototipo de un dispositivo de síntesis de audio y efectos musicales el cual permita la conexión con un controlador *MIDI* y pueda ser utilizado para dar presentaciones en vivo sin necesidad de una computadora. Así mismo, analizar desde una perspectiva científica que es lo que sucede en cada etapa del sintetizador.

El enfoque del trabajo es implementar distintos algoritmos de síntesis aditiva y sustractiva y así mismo efectos de audio digitales. Se busca medir el desempeño de los algoritmos utilizando el uso del *CPU* y la memoria *RAM* y según esto evaluar si es factible agregar las funcionalidades que requiere un producto final.

Este trabajo busca solamente ser la primera iteración del desarrollo de un sintetizador de audio utilizando hardware de bajo poder computacional. La latencia, calidad de sonido, automatización al arranque del sistema operativo y capacidad para guardar *presets* quedan fuera del alcance de este trabajo, así mismo es posible que durante el periodo de prueba aparezcan fallos que no fueron detectados durante la etapa de desarrollo, estos fallos sentarán las bases y líneamientos para continuar esta línea de desarrollo e investigación.

CAPÍTULO 6

Marco teórico

6.1. Sistemas operativos de tiempo real

La meta de un sistema operativo de tiempo real es crear un ambiente predecible en donde todas las tareas de tiempo real puedan medirse y tener un tiempo máximo de ejecución conocido.Como se menciona en [5] El propósito principal no es incrementar la velocidad del sistema o disminuir la latencia entre la acción y la respuesta, aunque estas si incrementan la calidad de un *RTO*. El propósito principal de un sistema de tiempo real es eliminar las "sorpresas" en lo referente a ejecución de tareas. Un error común es pensar que un *RTO* va a mejorar el rendimiento general del hardware en el que corre. Un *RTO* de calidad puede tener un rendimiento bueno, sin embargo, se sacrifica rendimiento por previsibilidad.

Para ilustrar este concepto se usará el ejemplo descrito en [5]. Un algorítmo hipotético puede completarse en un promedio de 250 micro-segundos en un sistema operativo que no es de tiempo real mientras que, en la misma máquina, un *RTO* tarda 300 microsegundos en completar el mismo algoritmo. La diferencia es que en un *RTO* se puede garantizar por adelantado el tiempo máximo de ejecución mientras que en un sistema no *RTO* esto no es posible.

6.1.1. Categorías de tiempo real

No existe una definición exacta de tiempo real, si bien nos da una idea de lo que se busca el término *tiempo real* es ambiguo. Cada aplicación tiene sus propios requerimientos de tiempo, por lo que no es fácil dar una definición general de tiempo real. Para este trabajo se utilizarán las siguientes definiciones propuestas en [6].

Fuerte - seguridad vital: Los requerimientos de *tiempo real* deben cumplirse el 100 % de las veces que el sistema ejecuta una tarea de tiempo real. Si no se cumple esto, alguien podría salir herido o morir y/o podría haber daño de equipo o propiedad.

Fuerte - 100 %: Los requerimientos de *tiempo real* deben cumplirse el 100 % de las veces que el sistema ejecuta una tarea de tiempo real. Si no se cumple esto, el resultado de la aplicación es totalmente inservible. Un ejemplo de esto es un proceso de control en donde los fallos de tiempo resultan en problemas de manufactura.

Fuerte - 95 %: Los requerimientos de *tiempo real* deben cumplirse por lo menos el 95 % de las veces que el sistema ejecuta una tarea de tiempo real. Un ejemplo de esto es el muestrear una señal de audio en donde fallos en el tiempo resultan en la distorsión de la señal y pérdida de información pero es aceptable el perder algunos datos, es decir, el producto final con la perdida de estos datos sigue siendo bueno.

Suave: Los requerimientos de *tiempo real* deberían de cumplirse la mayoría de las veces que el sistema ejecuta una tarea de tiempo real. Un ejemplo de esto es reproducir un video en la PC. El desempeño de tiempo real suave es subjetivo y queda a discreción del usuario.

6.1.2. Manejo de tiempo real con Linux

Linux es un sistema operativo de uso general de tipo *UNIX*, el *kernel* fue desarrollado en un principio por *Linus Torvalds*, diseñado para ser multiplataforma y multitarea. Sin embargo, el poder de Linux de correr en practicamente cualquier plataforma y ser de código abierto hizo que se empezara a intentar hacer modificaciones a este para poderlo utilizar en aplicaciones de tiempo real. Hay distintos enfoques o técnicas para implementar un *RTO* con Linux, los más comunes es utilizar un sistema de tareas con prioridades junto a un planificador (*scheduler*) [7] o un subsistema debajo del kernel de Linux dedicado a tareas de tiempo real como se explica en [6].

Para la realización de este trabajo se utilizó la distribución de linux *Patchbox OS* que es un sistema operativo que implementa las características de *Real Time Linux* pero diseñado para la *Raspberry Pi* y optimizado para aplicaciones de audio.

6.2. Desarrollo de algoritmos de audio en Supercollider

6.2.1. Generalidades de SuperCollider

Supercollider es una plataforma para síntesis de audio y composición algorítmica, usado por músicos, artistas e investigadores que trabajan con sonido. Actualmente está disponible para Windows, macOs y Linux.

SuperCollider se compone de tres componentes principales:

- **scsynth**, el cual es sun servidor de audio en tiempo real (*real-time audio server*), y conforma el núcleo de la platforma. Cuenta con más de 400 generadores de unidades (*'UGens'*) para análisis, síntesis y procesamiento. Su granularidad permite la combinación fluida de varias técnicas de audio conocidas y desconocidas, desde síntesis aditiva, sustractiva, FM, granular, FFT y hasta modelado físico. Inclusive es posible escribir nuevos *UGens* en *C++*.

- **sclang**, es un lenguaje de programación interpretado. Se centra en el sonido, pero no se limita a ningún dominio específico. Sclang controla scsynth (el servidor de audio) mediante el protocolo *Open Sound Control*, el cual es un protocolo para la comunicación entre computadoras, sintetizadores de sonido y otros dispositivos multimedia que está optimizado para la tecnología de red moderna. Puede usarse para la composición y secuenciación algorítmica, encontrar nuevos métodos de síntesis de sonido, conectar una aplicación a un hardware externo, incluidos los controladores MIDI, música de red, GUI de escritura y pantallas visuales. También cuanta con un stock de extensiones contribuidas por usuarios llamadas Quarks.

- **scide** es un editor para sclang con un sistema de ayuda integrado.

SuperCollider fue desarrollado por James McCartney y lanzado originalmente en 1996. En 2002, lo lanzó generosamente como software libre bajo la Licencia Pública General de GNU. Ahora es mantenido y desarrollado por una comunidad activa y entusiasta. Si se desean más detalles acerca del funcionamiento de SuperCollider se recompienta referirse a [8]

6.3. Desarrollo de algoritmos de audio en Pure Data

Pure Data (o simplemente Pd) es un lenguaje de programación visual de código abierto para multimedia. Su distribución principal (también conocida como Pd Vanilla) es desarrollada por Miller Puckette. Pd-L2ork / Purr-Data es una distribución alternativa (originalmente basada en el proyecto Pd-Extended ahora sin mantenimiento, muerto y en desuso), con una GUI renovada y muchas bibliotecas externas incluidas.

Pd permite a músicos, artistas visuales, artistas, investigadores y desarrolladores crear software gráficamente sin escribir líneas de código. Pd se puede utilizar para procesar y generar sonido, video, gráficos 2D / 3D y sensores de interfaz, dispositivos de entrada y MIDI. Pd puede trabajar fácilmente en redes locales y remotas para integrar tecnología portátil, sistemas de motores, equipos de iluminación y otros equipos. Es adecuado para aprender métodos básicos de procesamiento multimedia y programación visual, así como para realizar sistemas complejos para proyectos a gran escala.

Las funciones algorítmicas se representan en Pd mediante cuadros visuales llamados objetos colocados dentro de una ventana de parche llamada lienzo. El flujo de datos entre objetos se logra a través de conexiones visuales llamadas cables de conexión. Cada objeto realiza una tarea específica, que puede variar en complejidad desde operaciones matemáticas de muy bajo nivel hasta funciones de audio o video complicadas, tales como reverberación, transformaciones FFT o decodificación de video. Los objetos incluyen objetos básicos de Pd vanilla, objetos externos o externos (objetos Pd compilados de C o C ++) y abstracciones (parches Pd cargados como objetos).

La información anterior fue obtenida del sitio oficial de *Pure Data* [9].

6.4. Estándar MIDI

Por sus siglas en inglés *MIDI* significa *Musical Instrument Digital Interface*. Es el estándar más utilizado en la industria musical por lo que aveces es confuso lo que *MIDI* es y no es. Se empezará por aclarar lo que *MIDI* es y lo que no es.

- *MIDI* es una manera de interconectar instrumentos musicales, computadoras y otros dispositivos, una manera de reproducir sonidos sintetizados en una tarjeta de sonido, un protocolo de comunicación estandarizado y una manera de controlar dispositivos no musicales, como la iluminación del teatro, grabadoras, etc. MIDI tiene que ver con el desempeño pero no con el audio como tal. Es una representación compacta de una presentación musical. Es modificable, una interpretación en secuencia puede editarse fácilmente para cambiar las notas, sincronización, e incluso los sonidos hechos por un instrumento.
- *MIDI* no es audio. Solo un instrumento *MIDI* receptor puede hacer sonido. No es demasiado lento para una buena temporización: los eventos *MIDI* se envían muy rápidamente, lo suficientemente rápido para la mayoría de las secuencias para jugar con la sensación del oido humano. Sin embargo, tampoco es infinitamente rápido demasiadas notas pueden obstruir los trabajos y causar retrasos

6.4.1. Canales MIDI

Hay 16 canales MIDI, numerados del 1 al 16, donde los valores de 0 a F en hexadecimal corresponden a los canales MIDI 1 a 16, respectivamente. Todos los mensajes, como *note-on* y *note-off*, son específicos de un canal MIDI. Esto nos da 16 canales de control independientes para controlar dispositivos MIDI.

Un solo dispositivo puede ser capaz de enviar o recibir mensajes en más de un canal MIDI. Por ejemplo:

- Un teclado normalmente envía mensajes en un solo canal a la vez.
- Un secuenciador puede enviar mensajes en cualquiera o en todos los canales
- Un módulo de sonido suele ser capaz de recibir mensajes en varios canales diferentes a la vez.

6.4.2. Tipos de mensajes MIDI

Los mensajes MIDI que son específicos de un canal MIDI se conocen como mensajes de canal. Los mensajes MIDI que afectan a todo el sistema MIDI (o al menos a un dispositivo MIDI completo) se conocen como mensajes del sistema. Los mensajes del sistema no están asociados con un canal MIDI.

Los mensajes del canal y del sistema se subdividen en las siguientes clases:

- **Mensajes de voz del canal:** Mensajes que inician, alteran o detienen un sonido o sonidos que se reproducen.
- **Mensajes de modo de canal:** Mensajes de control que afectan a todo el canal, como cambiar del modo polifónico al modo monofónico y desactivar todas las notas
- **Mensajes en tiempo real del sistema:** Estos son utilizados por los secuenciadores para regular y sincronizar la temporización. Consisten en un byte de estado solamente; No se utilizan bytes de datos.
- **Mensajes comunes del sistema:** Incluye mensajes como 'puntero de posición de canción', 'selección de canción', marco de cuarto de código de tiempo MIDI, que son utilizados por los secuenciadores.
- **Mensajes exclusivos del sistema** Generalmente se utiliza para extensiones específicas del dispositivo al protocolo MIDI. Cada fabricante es libre de definir sus propios mensajes exclusivos del sistema. Los mensajes de muestra de volcado estándar, mensaje completo de control de tiempo MIDI y otras extensiones también se implementan como mensajes exclusivos del sistema

En general, los mensajes exclusivos del sistema son los únicos mensajes MIDI que tienen más de unos pocos bytes. En este trabajo se utilizarán con regularidad 2 tipos de mensajes: Mensajes de voz y de modo del canal.

6.4.3. Mensajes de voz del canal

Los mensajes de voz portan información referente a la interpretación musical.

Note On: Este mensaje se envía cuando una tecla es presionada en el controlador MIDI. El mensaje incluye cual nota fue presionada, información codificada en un rango de enteros entre 0 y 127. Se incluye el canal en el cual se envió la nota (1-16) y la velocidad con la que se presionó la tecla, valor que varía de 0 a 127. Este ultimo dato provee información de la intensidad con la que se debe reproducir la nota correspondiente a la tecla presionada.

Note Off: Este mensaje se envía en el momento en el que se libera una tecla previamente presionada. Este se envía con el propósito de finalizar la reproducción de la nota activada con el mensaje Note On. El mensaje también puede incluir información de la velocidad con la que se liberó la tecla, aunque la mayoría de instrumentos MIDI no utilizan este dato.

Aftertouch: Este es un mensaje que se envía cuando el usuario ejerce mayor presión en una tecla que mantiene presionada. Usualmente se asigna al control de efecto de vibrato del sonido, sin embargo, también se utiliza en otros parámetros como el volumen. Este mensaje puede ser monofónico o polifónico. En el primer caso basta con que una tecla envíe el mensaje para afectar todas las teclas presionadas en el momento. En el caso polifónico cada tecla es afectada individualmente por el mensaje.

Pitch Bend: Este mensaje es controlado por la rueda o palanca de portamento en el controlador MIDI. Este se utiliza para subir o bajar el tono generado por una nota mientras esta se reproduce. Con la finalidad de brindar un control más exacto, puede adquirir un valor de 0 a 16,383, en comparación con el rango usual de 8 bits.

6.4.4. Mensajes de modo del canal

Mensaje de cambio del programa: Este mensaje es utilizado para cambiar el programa del canal. El programa está asociado al instrumento digital que cuenta con un banco de voces. Este mensaje tiene un rango de 0 a 127. En la actualidad un controlador MIDI puede tener mucho más que 128 opciones de instrumentos y sonidos a sintetizar, por lo que los programas se organizan en bancos. Debido a ello es necesario incluir un mensaje de cambio de banco junto con el cambio de programa.

Mensaje de cambio de control: Este mensaje permite controlar varios parámetros del canal MIDI. Existen 128 distintos controles a modular. Algunos controles están estandarizados para todos los controladores MIDI.

6.5. Síntesis de audio digital

6.5.1. Representación digital del sonido

Como se describe en [10] El sonido resulta de la perturbación mecánica de algún objeto en un medio físico como el aire. Estas perturbaciones mecánicas generan vibraciones que pueden representarse como señales eléctricas por medio de dispositivos (por ejemplo, un micrófono), que convierten estas vibraciones en voltaje variable en el tiempo. El resultado de la conversión se llama señal analógica. Las señales analógicas son continuas en el sentido de que consisten en un valores continuos en el tiempo, en lugar de valores en tiepo escalonado, es decir, en ciertos instantes de tiempo.

Una señal analógica se puede grabar en una cinta magnética utilizando tecnología electromagnética. Para reproducir sonidos grabados en cinta magnética, la señal se escanea y se envía a un altavoz que reproduce las vibraciones de sonido en el aire. Los sintetizadores analógicos funcionan básicamente mediante la creación de sonidos desde cero, utilizando dispositivos electrónicos capaces de producir señales adecuadas para hacer vibrar a los altavoces.

Para procesar los sonidos en la computadora, la señal de sonido analógica debe convertirse a un formato digital; es decir, el sonido debe representarse utilizando números binarios. A la inversa, la señal digital debe convertirse en voltaje analógico para reproducir un sonido desde la computadora. Por lo tanto, la computadora musical debe contar con dos tipos de convertidores de datos: analógico a digital (ADC) y digital a analógico (DAC).

La conversión se basa en el concepto de muestreo. El proceso de muestreo funciona midiendo la tensión (es decir, la amplitud) de la señal continua a intervalos de igual duración. Cada valor de medición se denomina muestra y se registra en formato binario. El resultado de todo el proceso de muestreo es una secuencia de números binarios correspondientes a los voltajes en lapsos de tiempo sucesivos.

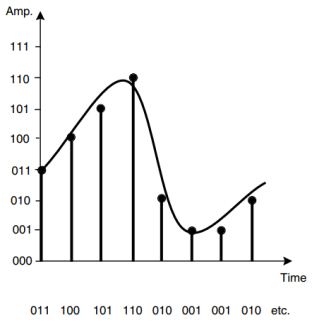


Figura 4: Proceso de muestreo de una señal analógica [10]

6.5.2. Sintetizadores de audio

Los sintetizadores de audio se diseñan principalmente con tres tipos de componentes:

- **Fuentes:** Objetos que renderizan una señal de audio. Estos pueden ser osciladores o generadores de ruido.
- **Modificadores:** Objetos que alteran la señal de audio que genera la fuente. Estos pueden ser filtros o efectos de audio.
- **Controladores:** Objetos que controlan parámetros tanto de las Fuentes como de los Modificadores. Estos pueden ser perillas en un controlador MIDI o señales que varian en el tiempo como envolventes.

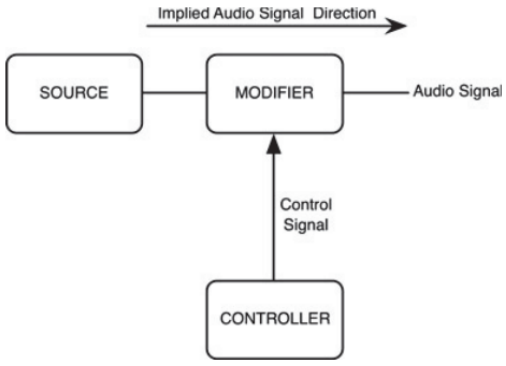


Figura 5: Estructura de un sintetizador básico de audio [11]

Osciladores

Un oscilador es un objeto qu renderiza audio sin la necesidad de una entrada. Un sintetizador puede construirse de diferentes maneras:

- Matemáticamente
- Utilizando *wavetables*

- Extrayendo fragmenteos de un archivo de audio
- Modulando frecuencia (FM) o Fase (PM)

CAPÍTULO 7

Pruebas preliminares

En este capitulo se muestran y describen las pruebas y procedimientos que se realizaron en primera instancia y que convergieron en los capítulos posteriores. Es importante hacer hincapie en que los procedimientos presentados a continuación no forman parte del diseño final del sintetizador de audio debido a que no se obtuvieron los resultados esperados o se presentaron demasiadas dificultades en su implementación, sin embargo son relevantes ya que sin estas pruebas fallidas no se hubiera llegado al resultado final.

7.1. Selección de hardware

Para llegar al diseño actual del sintetizador de audio se realizaron pruebas con distintos equipos con la idea de probar si el hardware se adaptaba a las necesidades del proyecto. Al inicio se probó utilizar un procesador digital de señales *ds-pic* con el cual se intentó implementar un oscilador sencillo modulable. Luego de algunas pruebas se descartó este microcontrolador debido a que la documentación de este era muy pobre. La seguna prueba que se hizo fue con una tarjeta de desarrollo *Cypress* con la cual se logró implementar una *wavetable* sencilla. Se pudo formar un acorde y algunas notas modulables en el código fuente. Sin embargo, el uso de periféricos no era el ideal en esta tarjeta ya que el api desarrollada por *Cypress* no estaba optimizada para el modelo de tarjeta que se tenía lo que implicaba tener que realizar todo esto en el ensamblador del microcontrolador lo que complicaba el desarrollo de este proyecto si que esta complicación estuviera justificada por una mejora significativa en el desempeño al correr los algoritmos. Dadas estas complicaciones se decidió utilizar una *raspberry pi* con una tarjeta de sonido externa. La utilización de este hardware tiene la ventaja (ante los anteriormente mencionados) de que tiene muy buena documentación y una comunidad de desarrollo grande lo cual hace más fácil encontrar una solución cuando se tiene un problema de código o implementación.

7.2. Selección de software

Las primeras pruebas se realizaron con *MATLAB*. Se utilizó este software para observar el comportamiento de distintos osciladores al variar los parámetros de diseño del sonido. Esto se utilizó para familiarizarse con el proceso de diseño de sonido, el cual se implementó posteriormente en hardware. Al tener claro el algoritmo que se buscaba implementar en hardware se iniciaron las pruebas en hardware. Como el primer hardware con el que se probó fue una tarjeta Cypress y posteriormente un Ds-pic, se implementó un algorítmo de síntesis utilizando una *wavetable* hecha en *C++*. Debido a las limitaciones de hardware mencionadas anteriormente no se continuó con el desarrollo de este algoritmo.

Al migrar la implementación a la *Raspberry Pi* se abrieron las puertas a entornos para el desarrollo del software para el motor de síntesis. En principio se utilizó *Supercollider*, la ventaja de este entorno de desarrollo en conjunto con la *Raspberry Pi* es la opción de multiplataforma, lo cual permite probar los algoritmos en cualquier PC antes de hacer la implementación en la *Raspberry Pi*. Se logró implementar algorítmos de síntesis aditiva y sustractiva, efectos de *Delay* y *Reverb* y una comunicación serial sencilla para controlar los parámetros de síntesis mediante un microcontrolador. Sin embargo, no se siguió utilizando software desarrollado en este entorno ya que se presentaron dificultades en hacer una interfaz para un control más completo, externo a *Supercollider*, en la *Raspberry Pi*. Debido a estas dificultades se migró finalmente a *Pure Data* en donde se implementaron todos los algorítmos se síntesis y efectos digitales.

CAPÍTULO 8

Diseño del sintetizador

8.1. Diseño general

Luego de las pruebas preliminares mencionadas en el capítulo 7 se llegó a un diseño final, el cual sería implementado. Los detalles se explican en capítulos posteriores. El sintetizador está compuesto por cinco etapas conectadas en serie:

1. Osciladores: Se tienen tres señales como osciladores. Una señal sinusoidal, una señal cuadrada y una señal diente de sierra.

2. ADSR: Son las señales envolventes de los osciladores. En esta implementacion se usaron funciones lineales para todos los componentes del *ADSR*.

3. Filtros: Se incluyeron tres filtros: Un filtro pasa bajas con frecuencia de corte modulable, un filtro pasa altas con frecuencia de corte modulable y un *LFO* con frecuencia de oscilación, frecuencia central y resonancia modulables.

4. Delay: Se implementó un efecto de *Delay* digital con *Tempo*, *Feedback* y *Mix* modulables.

5. Reverb: Se implementó un *Reverb* digital basado en *All pass Filters* conectados en serie.

Estas etapas toman como entrada las señales enviadas por el controlador *MIDI* y tienen como salida un *DAC*. En la Figura 6 se muestra el diagrama de flujo del algorítmo que se sigue. Cada etapa tiene varios parámetros que modifican el sonido que produce la señal de salida. Cada uno de estos parámetros puede modularse por medio de la interfaz de usuario.

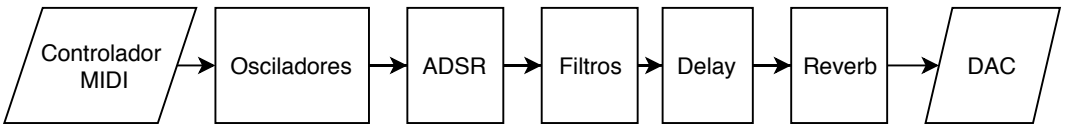


Figura 6: Algoritmo que implementa el sintetizador

8.2. Diseño del motor de síntesis

8.2.1. Osciladores

Como se mencionó anteriormente, los osciladores son el corazón del sintetizador. Este módulo es el encargado de generar la señal de audio. Cada oscilador recibe como entrada una señal *MIDI* la cual se decodifica para obtener la frecuencia de la nota y tiene como salida la señal respectiva a la frecuencia indicada. Por cada oscilador tienen dos osciladores desfasados para generar un efecto de *detune*. Estos osciladores reciben la misma entrada *MIDI* que el oscilador principal, pero adicional a esto reciben un valor de desfase en frecuencia. Estos dos osciladores se conectan en paralelo por medio de un sumador. En la Figura 7 se muestra la el diagrama de bloques que se sigue para la síntesis de cada nota enviada vía *MIDI*.

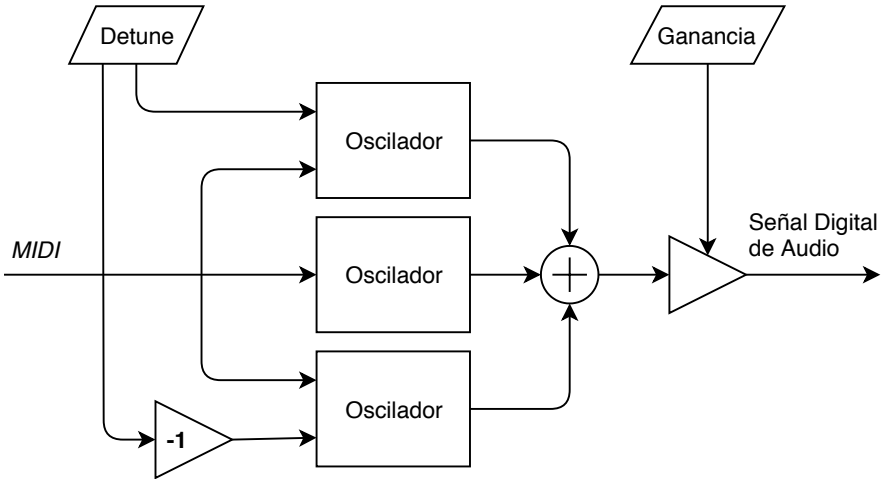


Figura 7: Diagrama de bloques de un oscilador con *detune*

Es importante mencionar y recalcar que lo mostrado en la Figura 7 se realiza tres veces por cada nota recibida para generar las señales sinusoidal, diente de sierra y cuadrada. En la Figura 8 se muestra el proceso completo.

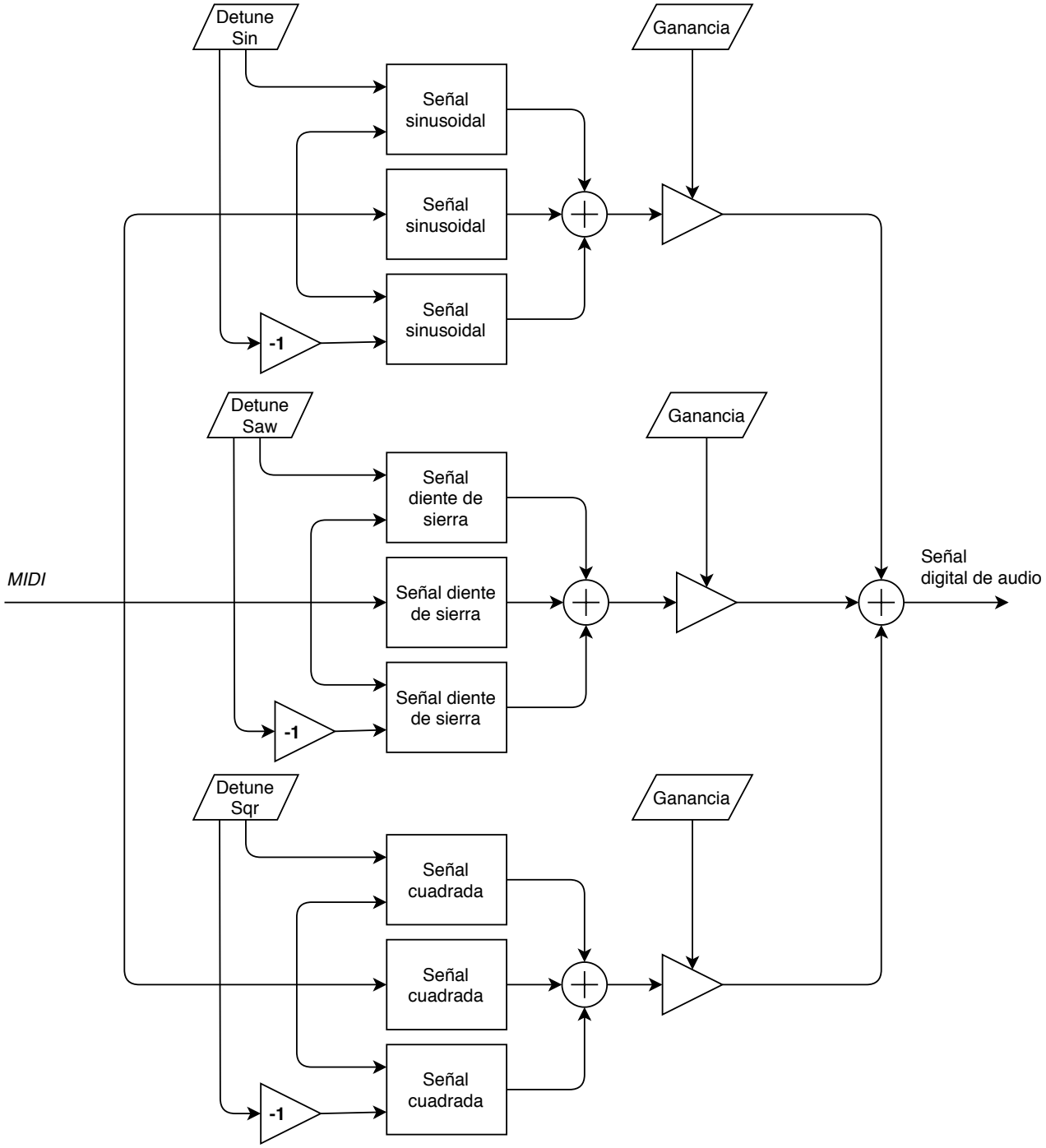


Figura 8: Diagrama de bloques del oscilador procesado por cada nota

Los parámetros de *detune* y ganancia se modulan por medio de la interfaz de usuario. Estos parámetros se detallan en el Cuadro 1.

8.2.2. ADSR

La sección ADSR, la cual representa *Attack*, *Decay*, *Sustain* y *Release*, se diseñó para que cada parte fuera lineal. Aunque lo ideal es tener envolventes exponenciales (ya que el oido humano percibe el sonido de forma logarítmica) debido a que hace que el sonido se escuche más natural, en este primer prototipo se implementaron transientes lineales ya que estos son menos caros computacionalmente y la implementación final se estaría haciendo en hardware de bajo poder computacional. En la Figura 9 se muestra la señal *ADSR* implementada y en la Figura 6 se muestra su entrada y salida. De esta señal se pueden modular los tiempos de *Attack*, *Decay*, *Sustain* y *Release* mas no los valores finales.

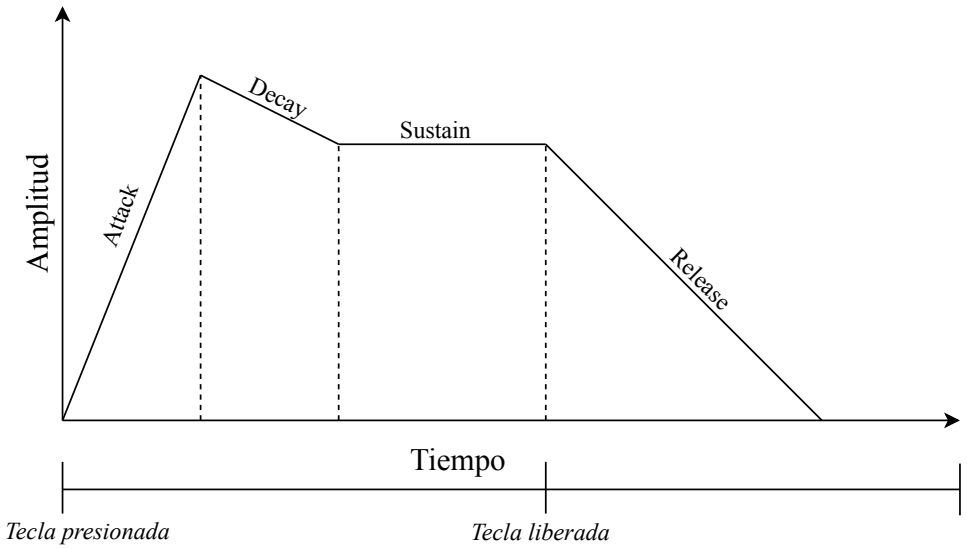


Figura 9: Señal *ADSR* envolvente

8.2.3. Filtros

Se incluyeron tres filtros. El primero es un filtro pasa-bajas con una frecuencia de corte *cutoff* modulable. El segundo es un filtro pasa-altas con una frecuencia de corte también modulable y el tercero es un *LFO*, el cual es un filtro pasa-banda con una frecuencia central ω_c y un factor de calidad Q. En este filtro *LFO* puede modularse la frecuencia con la que oscila la frecuencia central ω_c, la resonancia y la cantidad de modulación. En la Figura 10 se muestra el diagrama de bloques de la implementación de dichos filtros. Como se observa, toda señal de audio pasa a través del filtro pasa-bajas. Esto es debido a que este filtro es el más usado en la industria del audio ya que "suaviza"los sonidos. Esa "suavización.[es] particularmente útil en los cambios de intensidad en la música ya que se tiene una frecuencia de corte más baja cuando la intensidad es baja y una álta cuando la intensidad de la música aumenta.

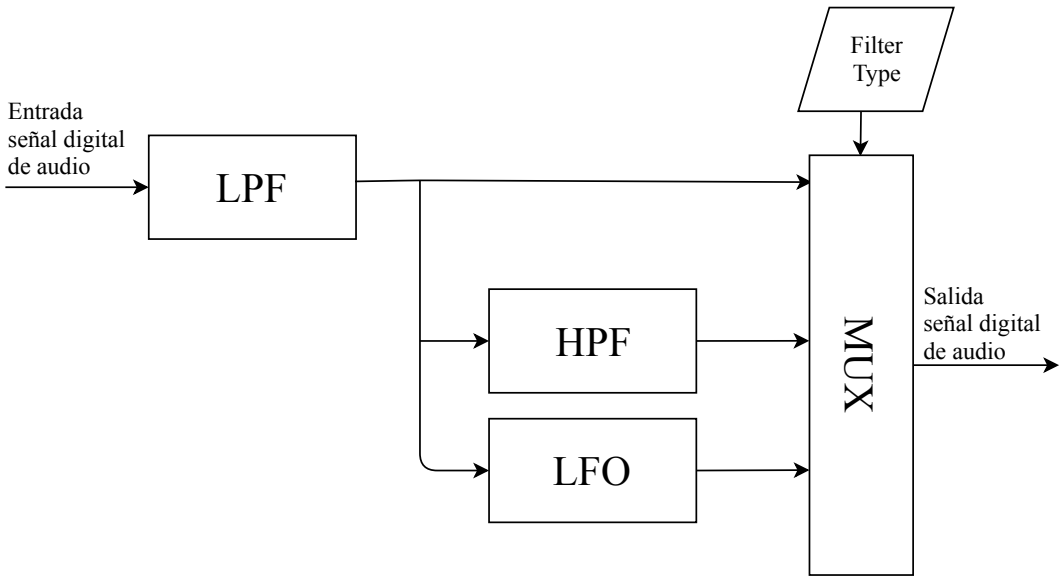


Figura 10: Diagrama de bloques de filtros implementados

8.2.4. Delay

Para el delay se utilizó el algorítmo mostrado en la Figura 11. Con este esquema se puede modular el *Tempo*, el *Feedback* y el *Mix*.

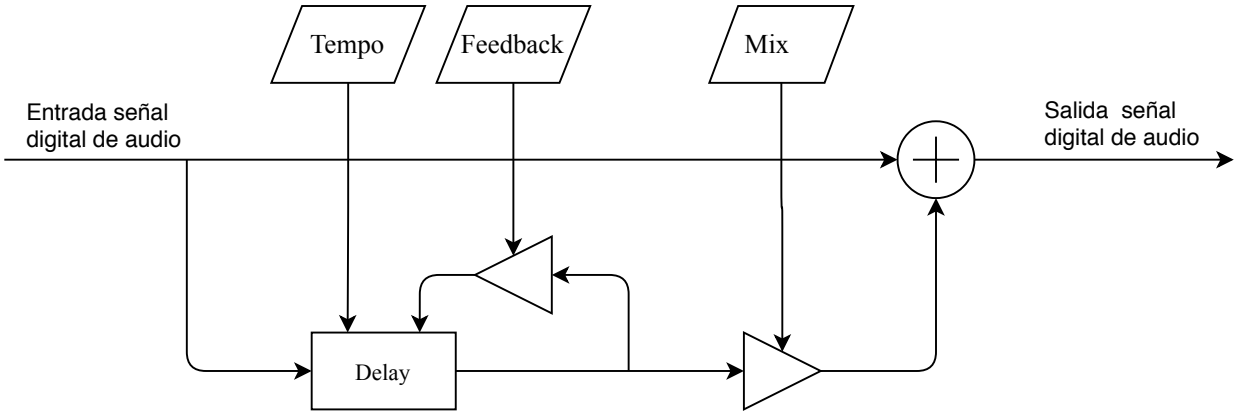


Figura 11: Diagrama del efecto de delay implementado

8.2.5. Reverb

El *Reverb* es uno de los efectos más utilizados en la música. Esto se debe a que impregna en la señal de audio la respuesta impulsional de un *ambiente* o *lugar*. Este efecto logra que sonidos simples (con un espectro en frecuencia simple) resuenen. Hay varios algoritmos para implementar un *Reverb*, algorítmos de convolución, basados en *delay networks*, y *comb filters* en paralelo seguidos por una serie de *all pass filter*; sin embargo, estos algoritmos son considerablemente complejos por lo que la implementación de este efecto estará determinada

por las herramientas disponibles en el software utilizado para la implementación. En el capítulo 9 se detalla esta implementación.

8.3. Diseño de iterfaz de usuario

La interfaz de usuario se diseño para que la modulación de todos los parámetros de diseño fuera fácil e intuitiva. La interfaz está dividida en seis grupos, control de volumen y ganancia, sección *ADSR*, sección de filtros, sección de osciladores, sección de *Delay* y sección de *Reverb*.

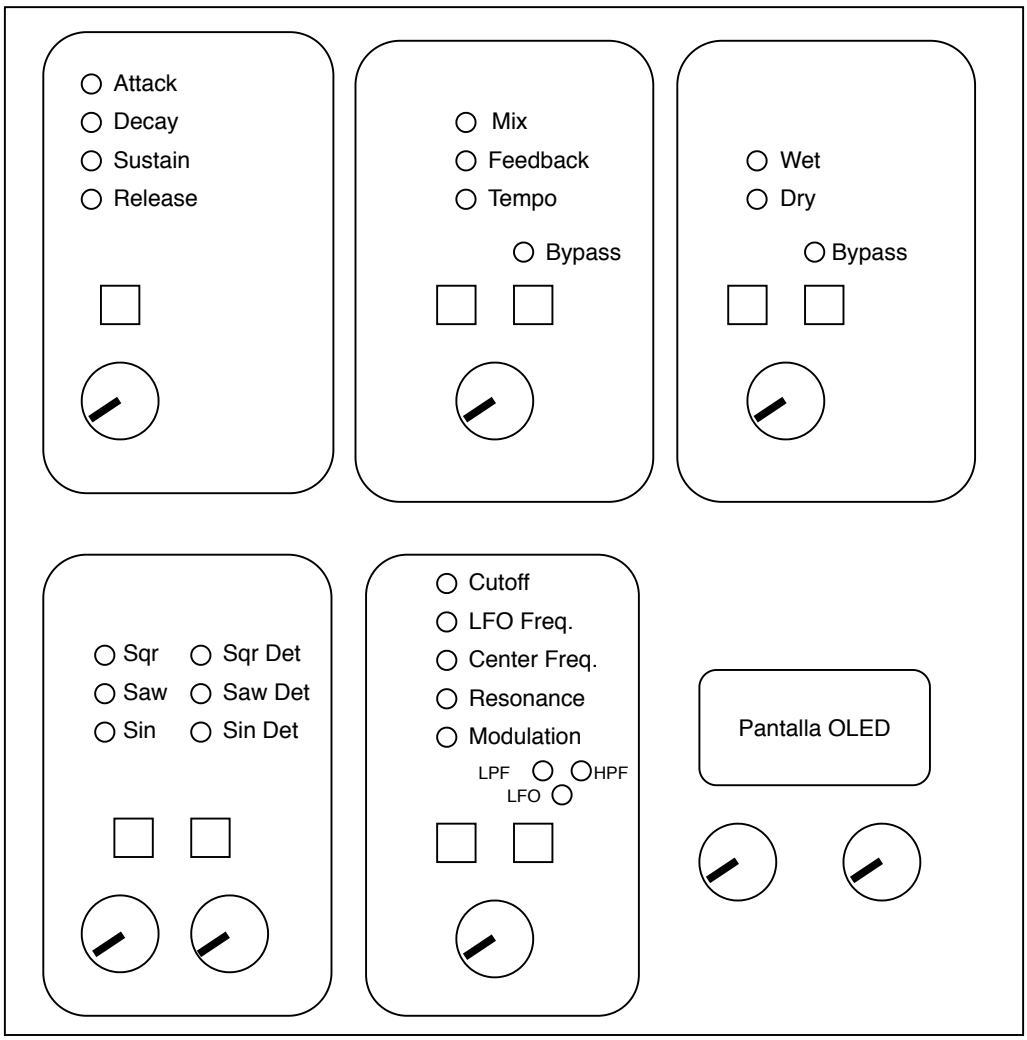


Figura 12: Diseño de interfaz de usuario

A continuación se muestran los parámetros que pueden modularse por medio de la interfaz de usuario.

Master	ADSR	Filtros		
		LPF	HPF	LFO
Volumen	Attack	Cutoff	Cutoff	LFO Freq.
Ganancia	Decay			Center Freq.
	Sustain			Resonance
	Release			Modulation

Cuadro 1: Parámetros del sintetizador

Osciladores		Delay	Reverb
Señal	Detune		
Sin Gain	Sin Detune Gain	Mix	Wet
Saw Gain	Saw Detune Gain	Feedback	Dry
Sqr Gain	Sqr Detune Gain	Tempo	Bypass
		Bypass	

Cuadro 2: Continuación parámetros del sintetizador

8.4. Planteamiento de diseño

En la Figura 13 se muestra el bosquejo del diseño planteado. Con este diseño se cumple el requerimiento de hacer portátil el motor de síntesis.

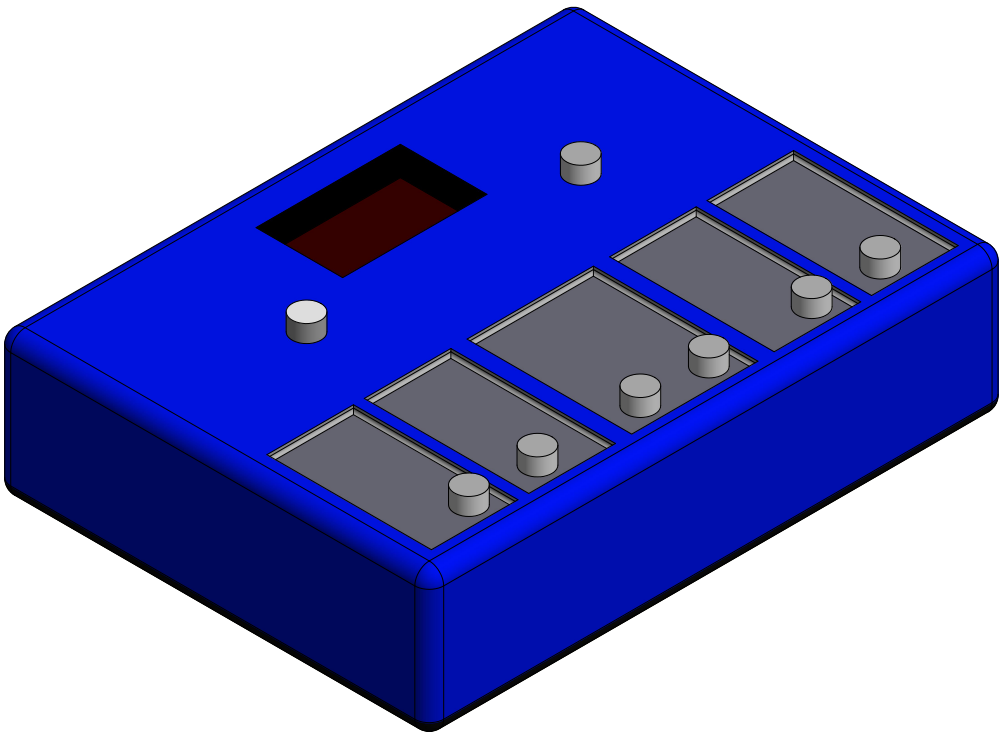


Figura 13: Diseño de interfaz de usuario

CAPÍTULO 9

Desarrollo del sintetizador

En el diseño final del sintetizador digital de audio se utilizó una *Raspberry Pi* en conjunto con una tarjeta de audio *HifiBerry DAC+* para la implementación del motor de Síntesis y un *Arduino MEGA* para el controlador físico. Se decidió utilizar este hardware ya que la *Raspberry Pi* provee una gama más amplia de opciones en el desarrollo de software y la opción de utlizar un *RTO*. Adicional a esto, permitía el uso de una tarjeta de audio con la cual se tiene un DAC con una resolución de 24 bits y una frecuencia de muestreo de 192kHz. Por otro lado se decidó utilizar el *Arduino MEGA* por la facildad de programación e implementación para la etapa de prototipado. En este capítulo se muestra a detalle como se implementó el sintetizador en todas sus fases. A continuación se muestra una lista detallada de los componentes utilizados para el desarrollo del sintetizador:

- *Raspberry Pi* modelo 3 B+.
- Tarjeta de audio *Hifiberry* DAC+ RTC
- *Arduino Mega* 2560 REV3.
- Una pantalla *OLED* SSD1306.
- 8 Potenciómetros e 10K Ohm.
- 10 Botones.
- 28 LEDs.

9.1. Configuraciones en Linux

9.1.1. Configuración ASLA para audio y MIDI

ALSA que por sus siglas en inglés significa *Advanced Linux Sound Architecture* utiliza el *kernel* para implementar implementar funciones de audio y MIDI de bajo nivel. ALSA proporciona varias aplicaciones útiles, es la capa que admite tarjetas de sonido, que es el término general de Linux para interfaces de audio de hardware, interfaces MIDI y más. La instalación de este *software* es de suma importancia ya que si no se hace correctamente no se podrá utilizar una tarjeta de audio externa ni un controlador *MIDI*. De primero se debe instalar *ALSA utils* esto puede hacerse con el siguiente comando:

```
sudo apt-get install alsa-utils
```

Al completar la instalación se tendrán las siguientes funcionalidades:

```
    alsactl    Change and save settings for an audio device
    amixer     Adjust volume and sound controls (ncurses version)
    alsamixer  Adjust volume and sound controls (ncurses version)
    aconnect   Make MIDI connections
    aseqview   Display ALSA sequencer events (e.g., note ON, note OFF)
    aplay      Play back an audio file from the command line
    arecord    Record an audio file from the command line
```

Para tener un control más facil de las configuraciones *MIDI* puede instalarse la interfaz gráfica *aconnectgui* la cual se muestra en la Figura 14.

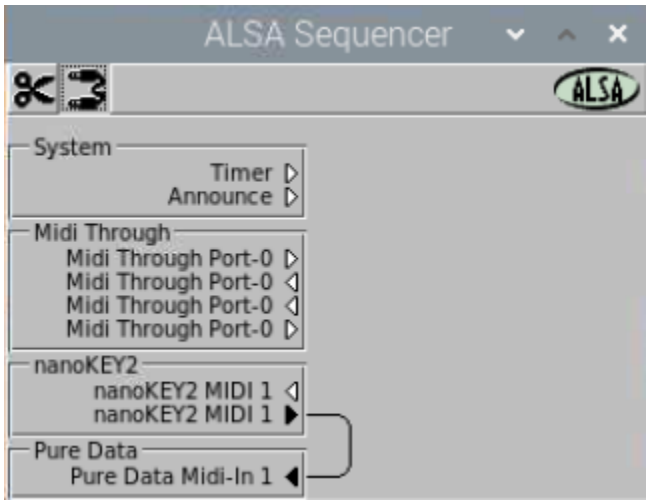


Figura 14: Interfaz gráfica para control de dispositivos *MIDI* en *Linux*

9.1.2. Configuración de tarjeta de audio Hifi Berry

La tarjeta de audio utilizada fue una *HIFIBERRY DAC+ RTC*. Esta es una tarjeta de audio que tiene un *DAC* de alta calidad con una resolución de 24 bits y una frecuencia de

muestreo de 192 KHz. Para configurar esta tarjeta se deben ejecutar los siguientes comandos:

```
1    echo "dtoverlay=i2c-rtc,ds1307" >> /boot/config.txt
2    echo "dtoverlay=hifiberry-dac" >> /boot/config.txt
3    echo "dtparam=i2c_arm=on" >> /boot/config.txt
4
5    echo "i2c-bcm2835" >> /etc/modules
6    echo "i2c-dev" >> /etc/modules
7    echo "rtc-ds1307" >> /etc/modules
8
9    echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
10   hwclock -r
11
12   echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
13   hwclock -w
14
15   echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
16   hwclock -s
17
18   ACTION=="add", SUBSYSTEM=="rtc", KERNEL=="rtc0", ATTR{hctosys}=="0", RUN
    +="/sbin/hwclock --rtc=%N --hctosys --utc"
```

9.2. Motor de síntesis

En este capítulo se expone el desarrollo del diseño detallado en el capítulo 8. El desarrollo del motor de síntesis se realizó en *Pure Data*. Se decidió utilizar este software luego de varias pruebas, las cuales se detallan en el capítulo 7. Las funcionalidades desarrolladas en *Pure Data* son:

- Interpretación de señales *MIDI*.
- Interpretación de señales *OSC* para la interfaz de usuario.
- Síntesis y efectos de audio.

En la Figura 15 se muestra la secuencia de objetos utilizados para la implementación total del motor de síntesis.

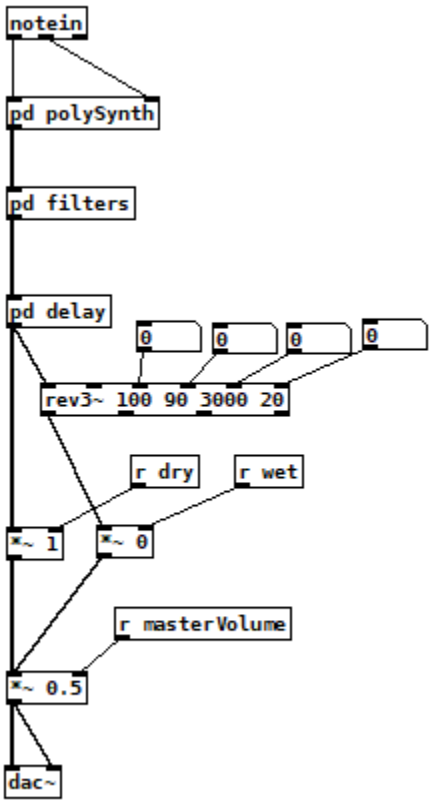


Figura 15: Implementación final del motor de síntesis en *Pure Data*

A continuación se explica cómo se implementaron los osciladores detallados en la Figura 8. Todo lo relacionado con los osciladores se encuentra dentro del objeto `polySynth` mostrado en la Figura 15.

9.2.1. Implementación de osciladores

Para empezar con el desarrollo del motor de síntesis se implementó lo mostrado en la Figura 8, este es un oscilador mono fónico ya que solamente recibe una entrada *MIDI*, la cual es codificada para interpretarse como una frecuencia. En la Figura 16 se muestra esta implementación en *Pure data*.

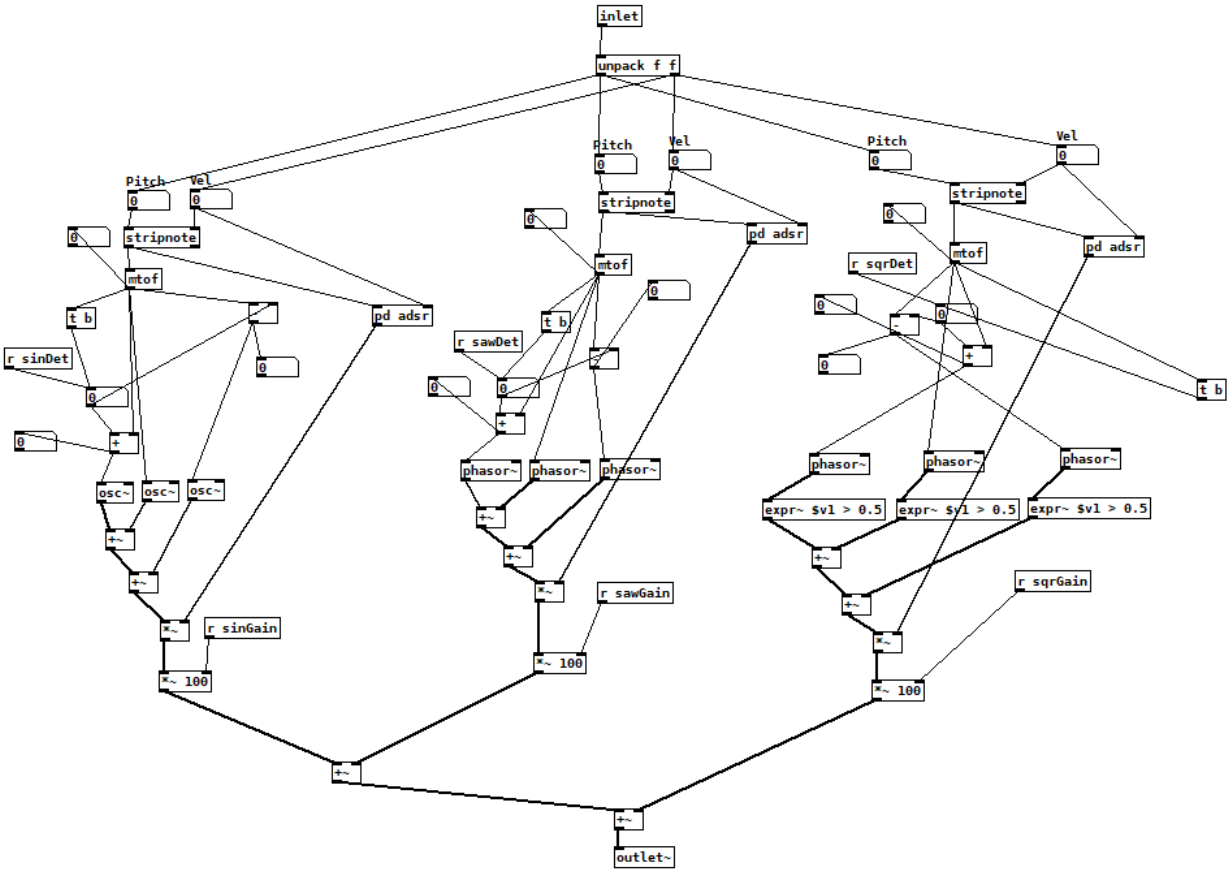


Figura 16: Implementación de oscilador mono fónico en *Pure Data*

Una de las consideraciones importantes del desarrollo de este motor de síntesis la capacidad de ser polifónico, es decir, realizar lo planteado en la Figura 6 en paralelo por cada nota presionada en el controlador *MIDI*. Para esto se empaquetó lo mostrado en la Figura 16 en un objeto que se definió con el nombre de `synth` y se utilizaron los objetos `poly`, `pack` y `route` que incluye *Pure Data* en su distrubución *Vanilla*. En la Figura 17 se detalla este procedimiento. De esta manera se logra tener un sintetizador capaz de procesar 8 notas en paralelo. Es importante notar que si se desea procesar más notas en paralelo se puede simplemente hacer más grande el objeto de `route` y agregar otro objeto `synth` sin embargo se decidió tener un límite de 8 notas ya que en las pruebas realizadas con la *Raspberry Pi* al agregar más se empezaban a tener problemas de saturación del procesador, además que, en general, 8 notas simultáneas es suficiente para las aplicaciones musicales.

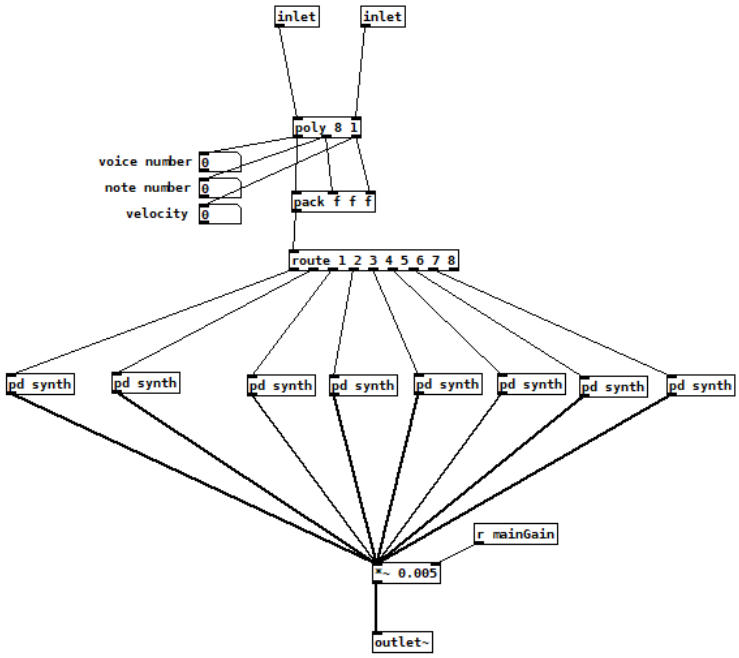


Figura 17: Implementación de oscilador polifónico en *Pure Data*

9.2.2. Implementación de ADSR

En la Figura 16 puede observarse un objeto definido como `adsr`. Dentro de este objeto se encuentra la implementación de la señal envolvente *ADSR* detallada en la Figura 9. Para esto se utilizó el objeto de *Pure Data* `vhline~`, en la Figura 18 se muestra la implementación.

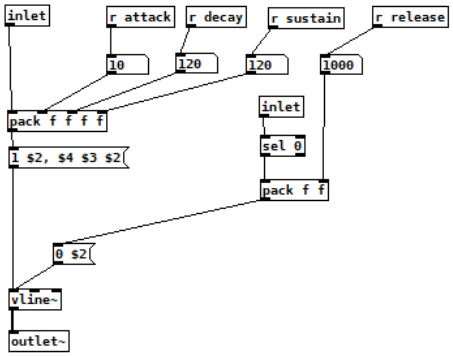


Figura 18: Implementación de la señal envolvente *ADSR* en *Pure Data*

9.2.3. Implementación de filtros

En la Figura 16 puede observarse un objeto definido como `filters`. Dentro de este objeto se encuentra la implementación en *Pure Data* de los filtros mostrados anteriormente en la Figura 10. En la Figura 19 se muestran los bloques de esta implementación en *Pure Data*.

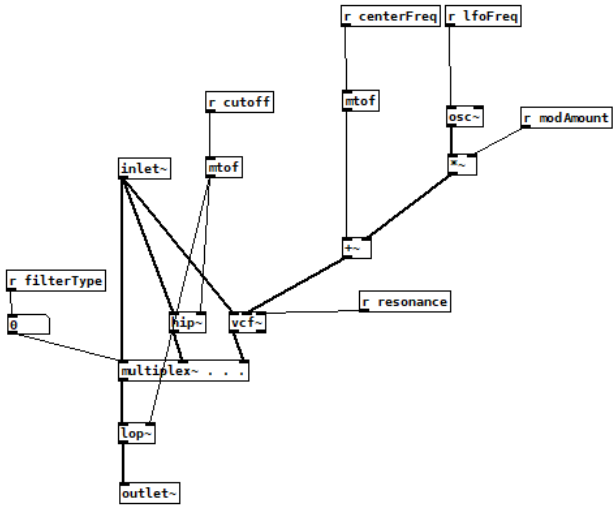


Figura 19: Implementación de filtros en *Pure Data*

9.2.4. Implementación de Delay

En la Figura 16 puede observarse un objeto definido como `delay`. Dentro de este objeto se encuentra la implementación de este efecto digital. En la Figura 20 se muestran los bloques de la implementación en *Pure Data*.

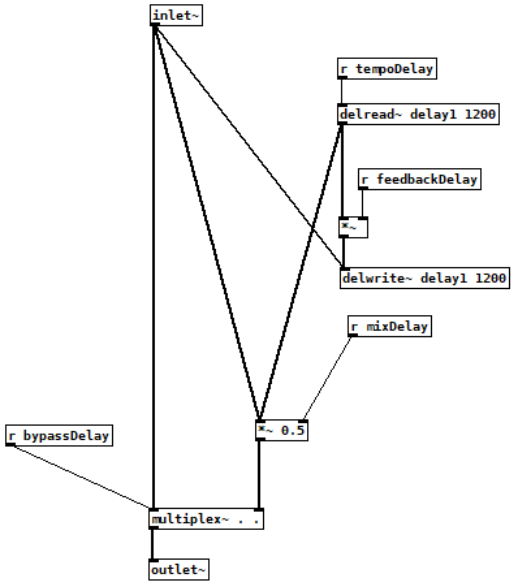


Figura 20: Implementación del efecto digital *delay* en *Pure Data*

9.2.5. Implementación de Reverb

Por último, se implementó el efecto de *Reverb*. En la Figura 16 puede observarse un objeto llamado `rev3~`. Este es un objeto que viene incluido en *Pure Data* en su versión

Vanilla. En la Figura 21 se muestran los bloques internos de este objeto.

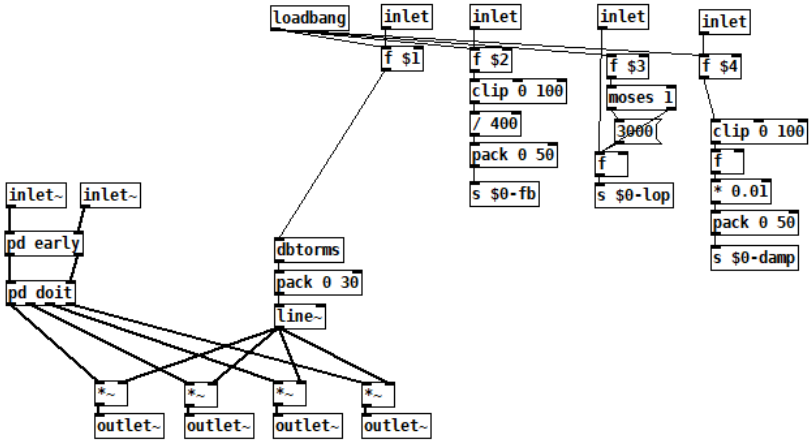


Figura 21: Implementación del efecto digital *Reverb* en *Pure Data*

9.3. Interfaz de usuario

Para la implementación de la interfaz de usuario se utilizaron potenciómetros para modificar el valor de los parámetros, botones para hacer la selección, una pantalla y LEDs para indicar lo que se está modificando. La interfaz de usuario cuenta con 18 entradas, 8 analógicas y 10 digitales, y 29 salidas incluyendo la pantalla. En la Figura 22 se muestra el circuito físico que implementa la interfaz de usuario.

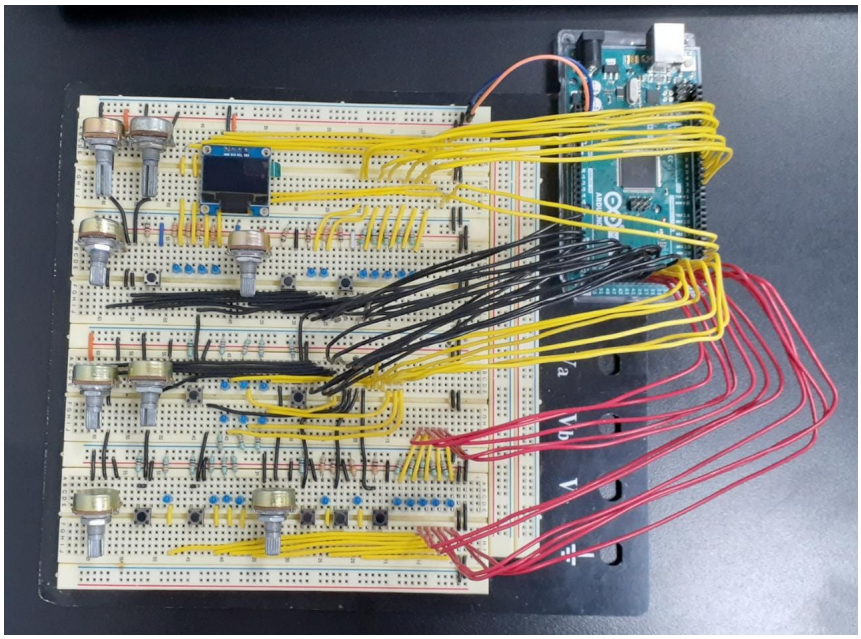


Figura 22: Implementación física de la interfaz de usuario

El propósito de los potenciómetros es modular el valor de los parámetros del motor de síntesis implementado en la *Raspberry Pi*. El funcionamiento deseado es que se pueda seleccionar un parámetro, capturar el valor del potenciómetro y enviarlo solamente cuando el valor esté cambiando, es decir, no se desea leer ni enviar varias veces el mismo valor. Esto presentó un reto debido a que el uso de potenciómetros tiene un problema inherente a las señales analógicas, el ruido. Para resolver este problema y poder leer y enviar el valor capturado del parámetro fue necesario implementar dos filtros digitales como se muestra en la Figura 23.

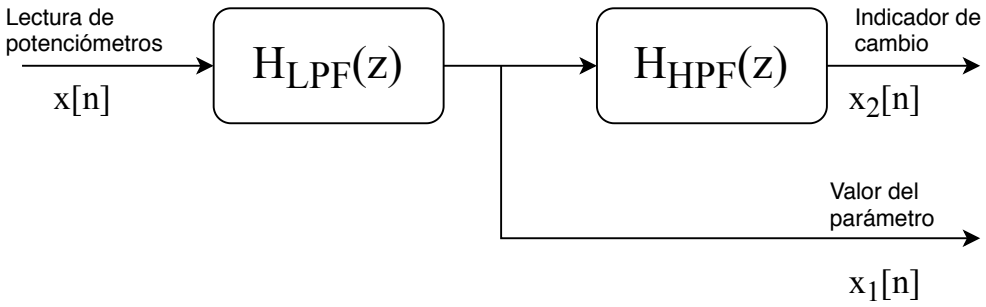


Figura 23: Filtrado de potenciómetros

El primero es un filtro pasa bajas de primer orden, con una frecuencia de corte en 0.5 Hz, este filtro se utilizó para limpiar el ruido inherente a componentes físicos que venía impregnado en la señal del potenciómetro así como para forzar a que la modificación de los parámetros fuera siempre suave. El segundo es un filtro pasa altas de primer orden con una frecuencia de corte en 100 Hz. Esta frecuencia de corte se determinó experimentalmente hasta obtener el resultado deseado. El propósito de este filtro es determinar cuando la señal obtenida está cambiando. Esto se logra ya que un filtro pasa altas remueve el componente DC de la señal obtenida, lo cual hace que cuando la señal está estática

$$|x_2[n]| \approx 0 \tag{1}$$

y cuando la señal cambia

$$x_2[n]| \gg 0 \tag{2}$$

de esta manera se puede determinar cuando los parámatros están variando, capturar y mostrar su valor en pantalla y mandarlos al motor de síntesis.

9.4. Comunicación entre la interfaz de usuario y el motor de síntesis

Como se mencionó anteriormente, para este proyecto se utilizó una *Raspberry Pi* para implementar el motor de síntesis y un *Arduino Mega* como controlador de la interfaz de

usuario. Para poder modular la interfaz de los parámetros de cada módulo en *Pure Data* se tuvo que enviar paquetes de datos desde el *Arduino* a *Pure Data*. Esto se logró haciendo la conexión que se muestra en la Figura 24.

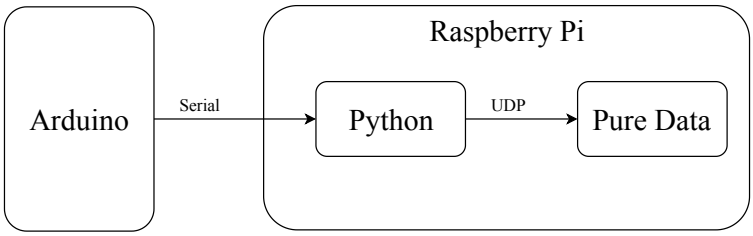


Figura 24: Conexión entre *Arduino* y *Pure Data*

La conexión entre *Python* y *Pure Data* se hizo mediante un protocolo *UDP*. Se elijió este protocolo sobre *TCP* ya que este funciona más rápido. Desde el lado de *Pure Data* esta conexión se manejó con un objeto llamado `netreceive` y un objeto `route`. En la Figura 25 se muestra la implementación del lado de *Pure Data*.

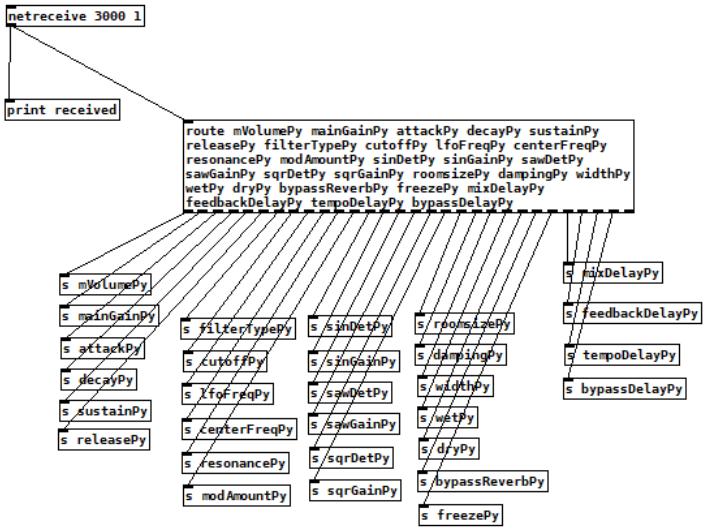


Figura 25: Implementación de la conexión UDP en *Pure Data*

El script hecho en *Python* maneja tanto la comunicación serial con *Arduino* como la comunicación *UDP* con *Pure Data*. En este caso, como *Python* corre sobre *Linux* se aprovechó el comando `echo` de `bash` y las funcionalidades que ofrece *Pure Data* desde una consola de *Linux* lo cual permitió hacer el *script* más corto. El código se muestra a continuación:

```
import serial
import io
import re
import os

ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)  # open serial port
sio = io.TextIOWrapper(io.BufferedRWPair(ser, ser))
```

```
8
9  print(ser.name) # check which port was really used
10
11 while True:
12     message = sio.readline()      # write a string
13     os.system("echo '" + message + "' | pdsend 3000 localhost udp");
14
15 ser.close() # close port
```

Listing 9.1: Comunicación entre Arduino y Pure Data

Y por último, desde *Arduino* se empaquetó la información de la siguiente manera:

```
1 Parametro_1 valor, Parametro_2 valor, Parametro_N valor
```

CAPÍTULO 10

Resultados finales

En este capítulo se exponen los resultados obtenidos a partir del prototipo final. Uno de los objetivos del presente proyecto era seleccionar y analizar efectos digitales que modifican el espectro de frecuencias de la señal de audio obtenida. En este capítulo se muestran las pruebas que se realizaron para cumplir este objetivo.

En el capítulo 8 se detallaron los parámetros modulables de cada sección del sintetizador. Cada uno de estos parámetros modifica directa o indirectamente la señal de audio de salida. Para analizar y estudiar el efecto que tiene cada sección sobre la señal de salida se realizaron las siguientes mediciones:

- Medición de la señal de salida variando parámetros de la sección de osciladores.
- Medición de la señal de salida con una configuración de filtros fija a una misma entrada de la sección de osciladores.
- Medición de la señal de salida con una configuración de *Delay* fija a una misma entrada de la sección de osciladores.
- Medición de la señal de salida con una configuración de *Reverb* fija a una misma entrada de la sección de osciladores.

Cada señal de salida fue medida en tiempo y en frecuencia (utilizando la opción de *FFT* en el osciloscopio utilizado). Es importante tener la medición en los dos dominios, tiempo y frecuencia, para poder entender mejor lo que sucede con la señal.

Si bien el sintetizador implementado es polifónico, se utilizó solamente una nota para todas las pruebas. Esto debido a que la combinación de varias notas saturan las gráficas obtenidas por el osciloscopio y no permiten visualizar con claridad lo que sucede, además, el

agregar varias notas solamente suma el mismo espectro en frecuencia pero con una frecuencia dominante diferente por cada una. Se eligió la nota de Do central (261.63 Hz) ya que las gráficas obtenidas con el osciloscopio fueron de mejor calidad a otras pruebas como un LA a 432 Hz por mencionar alguno. Sin embargo, si se realizan las mismas pruebas con cualquier otra nota que no sea un Do central, las gráficas deberían tener la misma forma, variando solamente su frecuencia dominante.

10.1. Osciladores

En esta sección se exponen los resultados obtenidos en el módulo de osciladores del sintetizador. Este módulo cuenta con tres osciladores: Una señal sinusoidal, una señal cuadrada y una señal diente de sierra. Cada señal cuenta con un *detune*. En este sintetizador, el *detune* suma dos osciladores extra de la misma forma pero desfasados en frecuencia hacia delante y hacia atrás. Para analizar este módulo del sintetizador se tienen tres configuraciones diferentes:

- Osciladores simples
- Osciladores combinados
- Osciladores con *Detune*

Para hacer el análisis es importante mencionar que no se tomaron todas las combinaciones de parámetros posibles ya que eran demasiados, en cambio, se eligieron variaciones de parámetros que expongan su efecto al cambiar en la señal de salida. Así mismo es importante mencionar que todo el análisis se hizo teniendo como entrada una señal *MIDI* de un Do central (261.62 Hz).

10.1.1. Osciladores simples

En esta prueba se analizaron por separado las tres señales implementadas en el sintetizador. Se obtuvieron las gráficas en tiempo y frecuencia. Esta sección sirve para poder observar las señales limpias, es decir, sin ningún efecto, y así tener una referencia o punto de partida para comparar los resultados obtenidos posteriormente al aplicar los distintos efectos implementados.

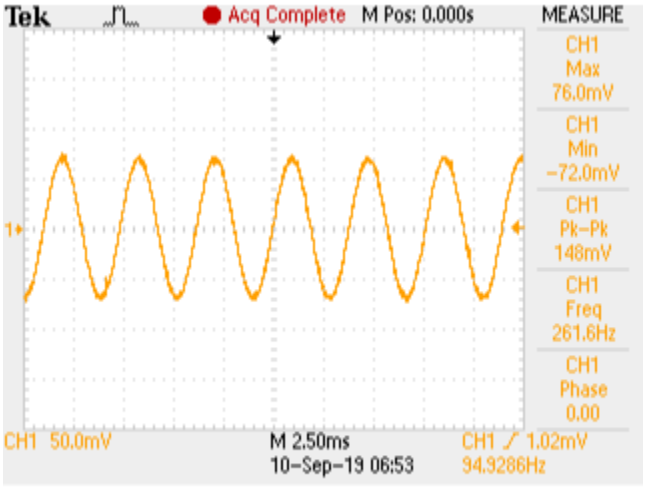


Figura 26: Señal sinusoidal

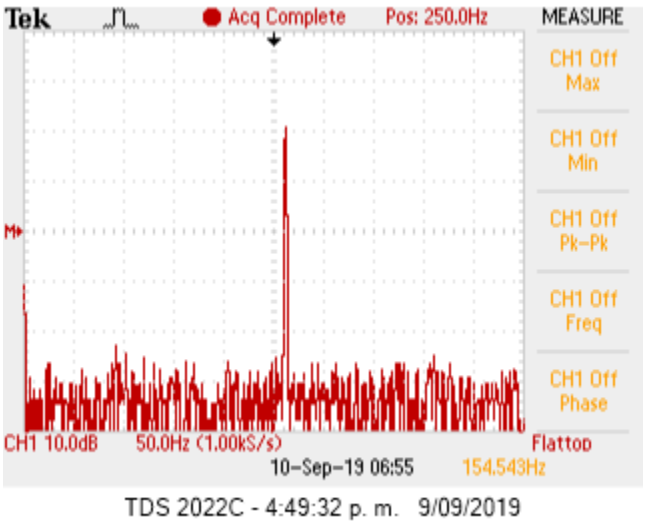


Figura 27: Espectro en frecuencia señal sinusoidal

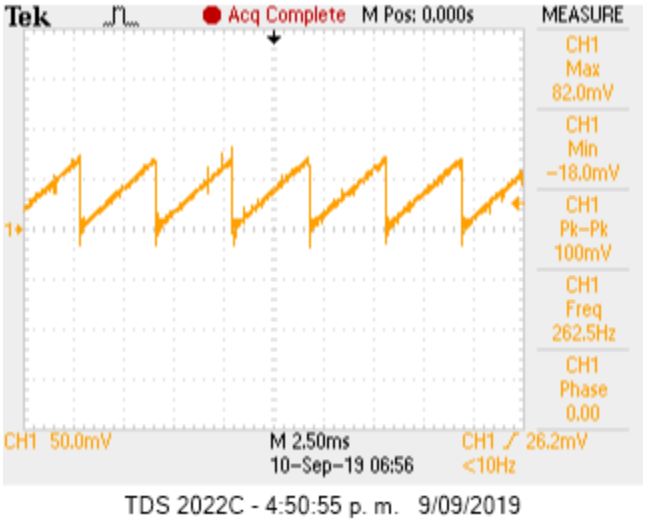


Figura 28: Señal diente de sierra

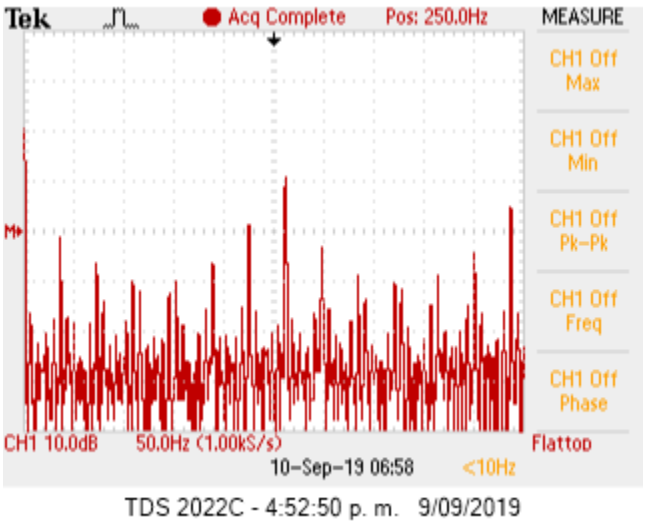


Figura 29: Espectro en frecuencia señal diente de sierra

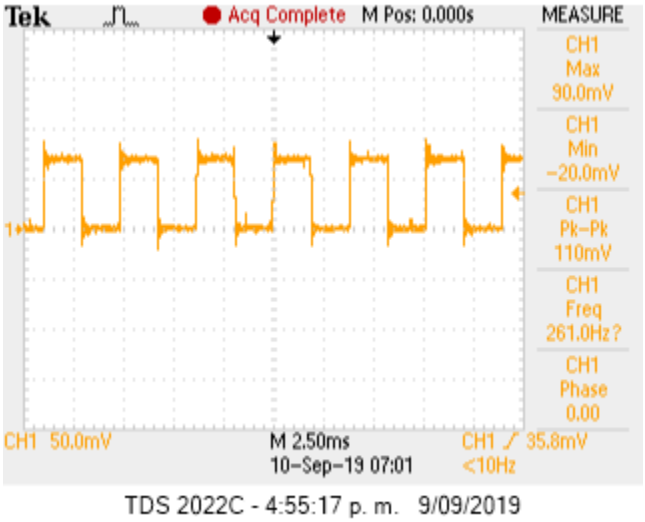


Figura 30: Señal cuadrada

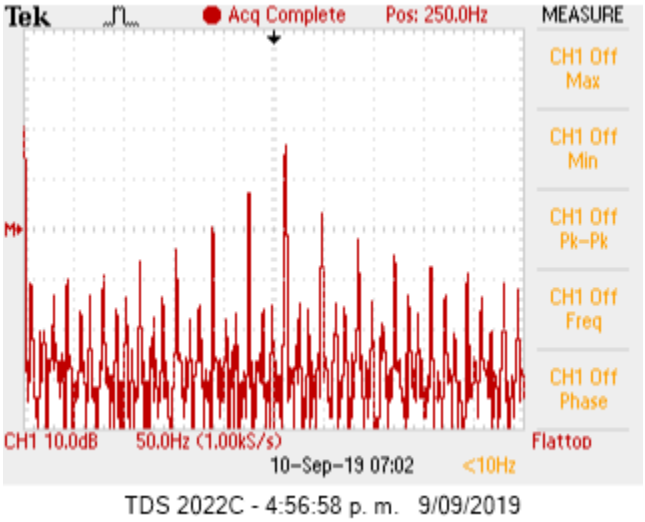


Figura 31: Espectro en frecuencia señal cuadrada

10.1.2. Osciladores combinados

En esta prueba se muestran las distintas combinaciones de osciladores. Estas configuraciones son importantes ya que en la mayoría de casos se utilizan varios osciladores combinados y no solamente uno. En esta prueba en particular, cuando se menciona que un oscilador está activado se refiere a que tiene 50 % de ganancia, por ejemplo, en la Figura 38 se muestra una señal resultante de combinar un diente de sierra y una señal cuadrada. Esto quiere

decir que tanto la señal de diente de sierra como la señal cuadrada tienen 50 % de ganancia mientras que la señal sinusoidal tiene 0 %.

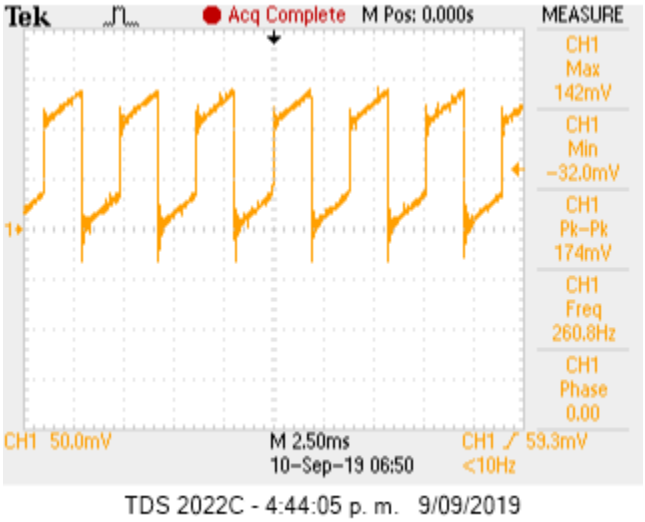


Figura 32: Señal diente de sierra y cuadrada combinadas

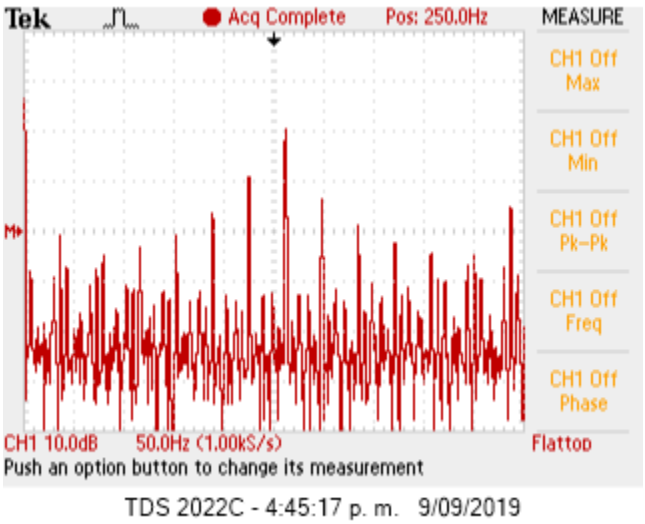


Figura 33: Espectro en frecuencia señal diente de sierra y cuadrada combinadas

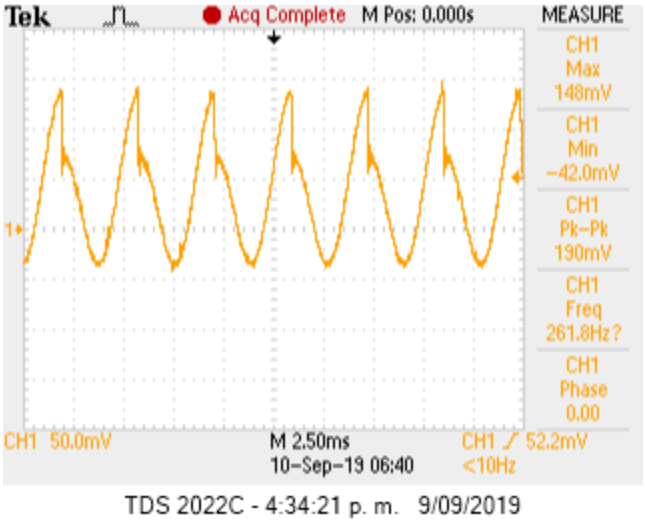


Figura 34: Señal sinusoidal y diente de sierra combinadas

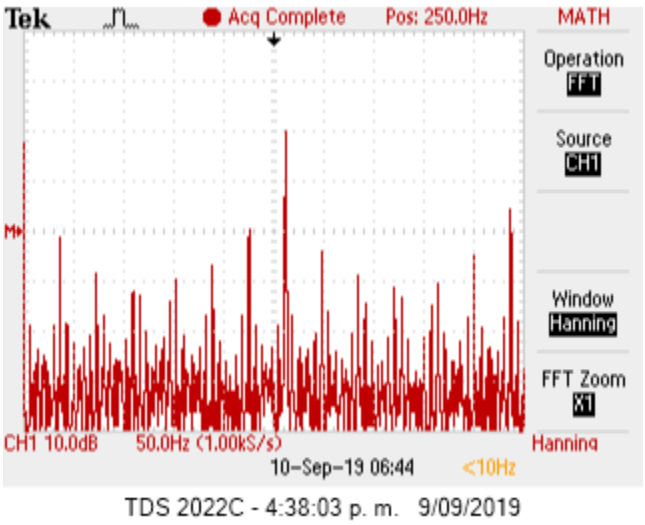


Figura 35: Espectro en frecuencia señal sinusoidal y diente de sierra combinadas

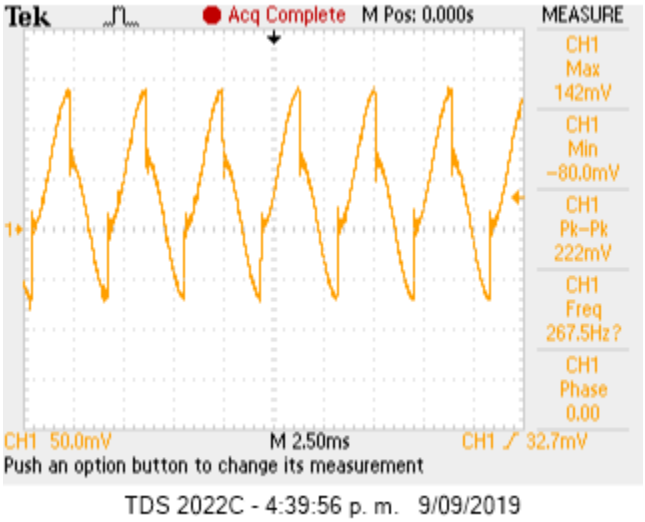


Figura 36: Señal sinusoidal y cuadrada combinadas

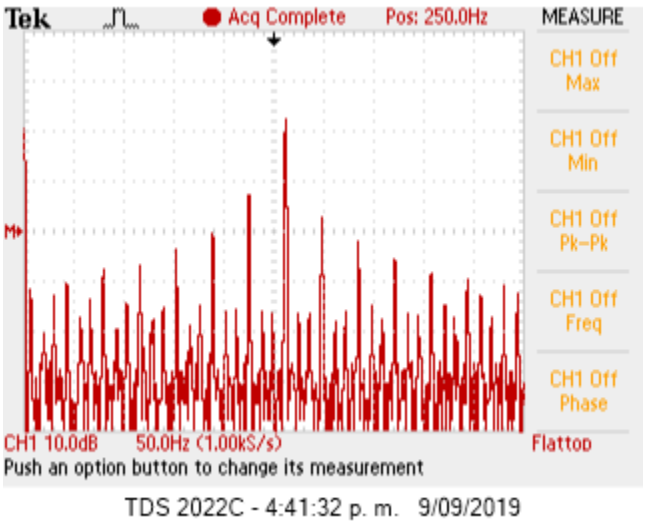


Figura 37: Espectro en frecuencia señal sinusoidal y cuadrada combinadas

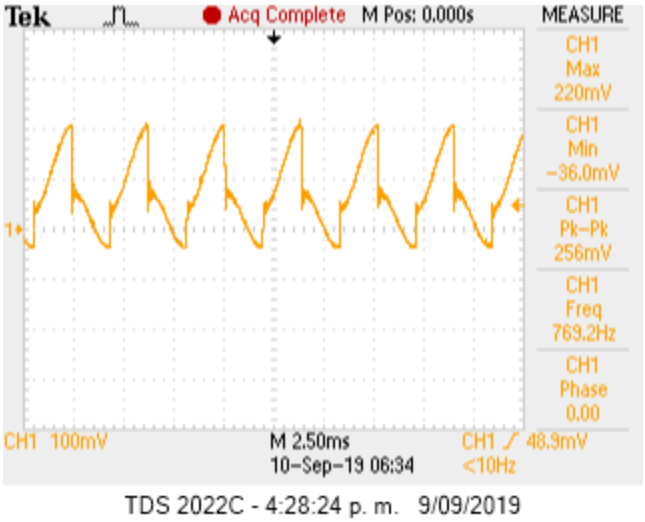


Figura 38: Señal sinusoidal, diente de sierra y cuadrada combinadas

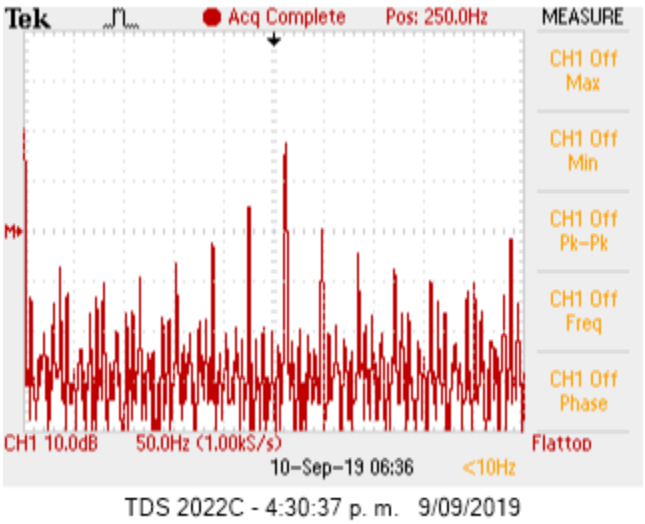


Figura 39: Espectro en frecuencia señal sinusoidal, diente de sierra y cuadrada combinadas

10.1.3. Osciladores con Detune

En esta parte se muestran los resultados del efecto de *Detune*. Se realizaron varias pruebas con distintas combinaciones de osciladores sin embargo se pudo observar que las señales obtenidas eran bastante complejas en el domino del tiempo por lo que se decidió, que para apreciar de una mejor manera solamente el efecto que produce el *Detune* era mejor observar cada señal por separado, es decir, la señal sinusoidal con 50 % de ganancia con el *Detune* al

50 % y lo mismo para las señales diente de sierra y cuadrada.

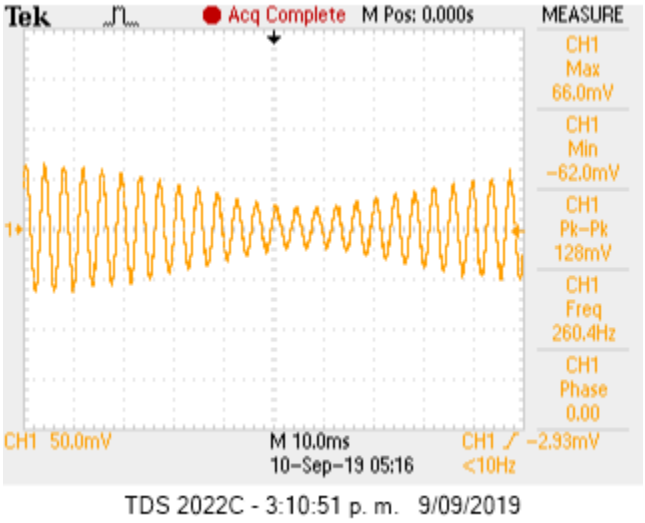


Figura 40: Señal sinusoidal con detune

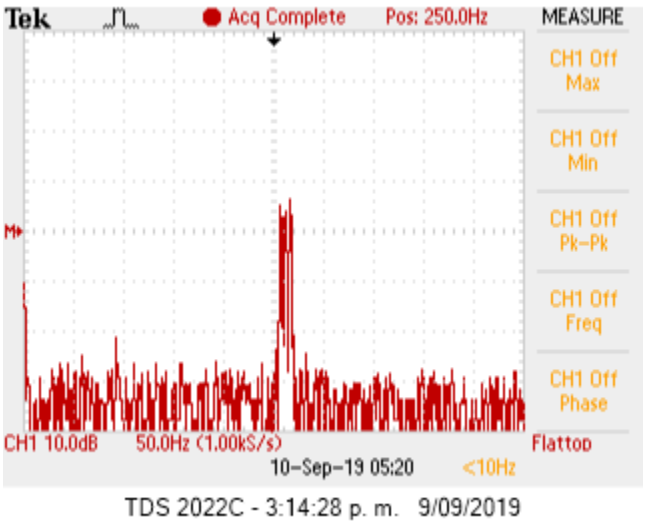


Figura 41: Espectro en frecuencia señal sinusoidal con detune

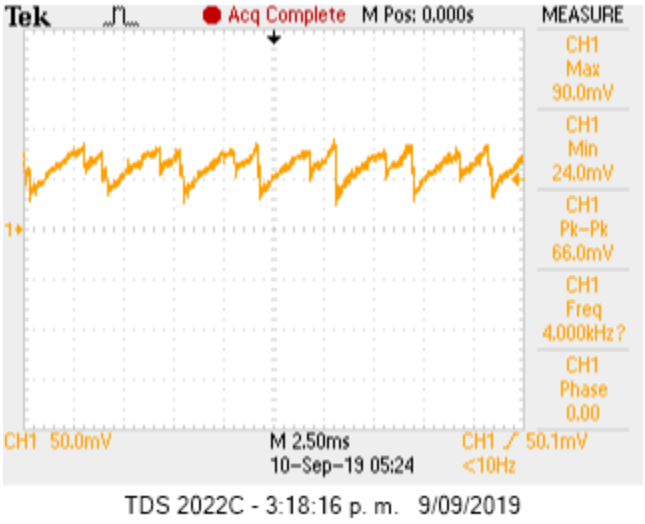


Figura 42: Señal diente de sierra con detune

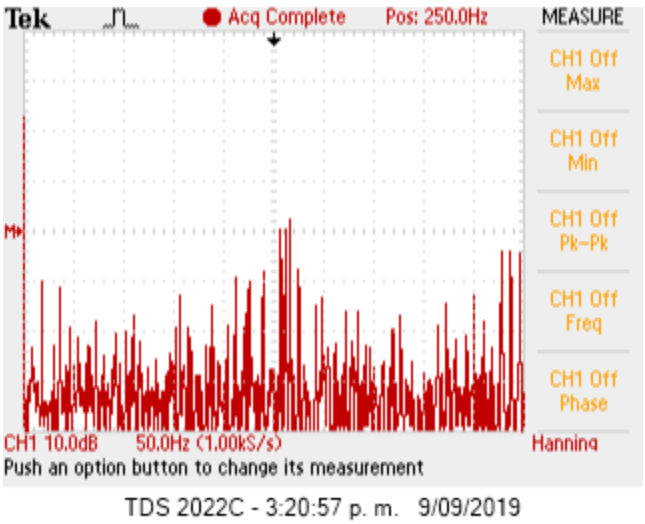


Figura 43: Espectro en frecuencia señal diente de sierra con detune

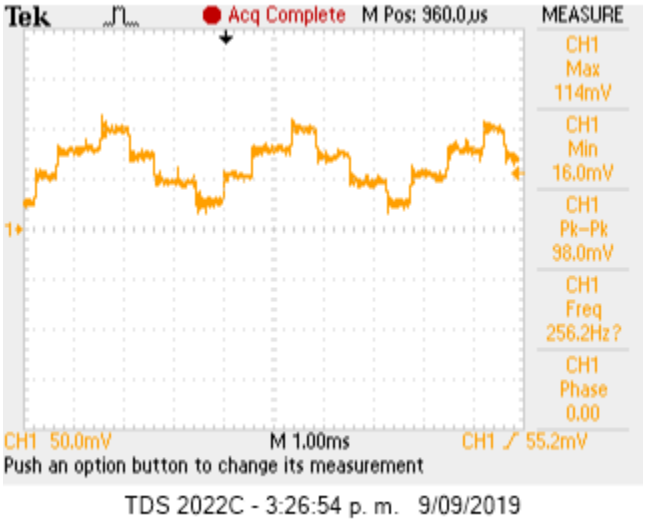


Figura 44: Señal cuadrada con detune

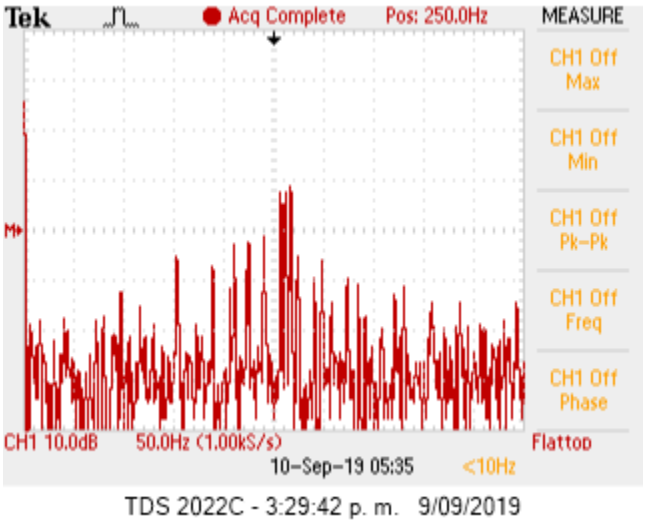


Figura 45: Espectro en frecuencia señal cuadrada con detune

10.2. Filtros

En esta parte se muestran los resultados obtenidos con las pruebas realizadas en el modulo de filtros. Como se detalla en el capítulo 9 estos filtros tienen una frecuencia de modulable. En este caso, como se deseaba conocer el solamente el efecto que tenían sobre la señal de entrada se probó con configuraciones estáticas, las cuales se detallan en cada

Figura. Para todas las configuraciones se utilizó la señal mostrada en la Figura 38 como entrada.

Señal de entrada: Señal Sinusoidal, Diente de Sierra y Cuadrada Combinadas.

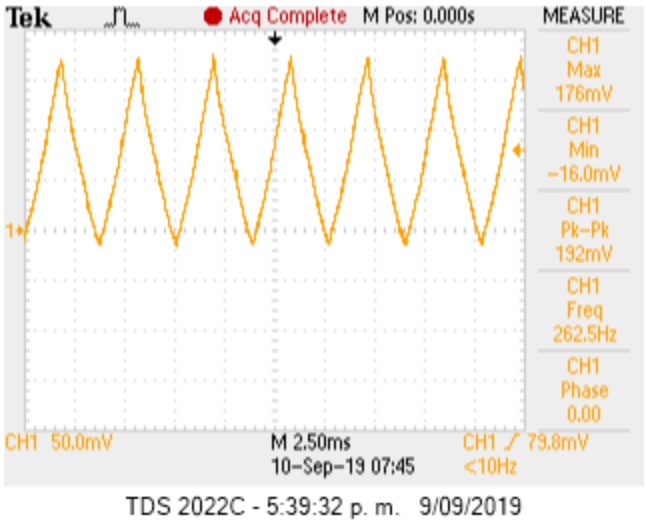


Figura 46: Señal de salida de filtro LPF con frecuencia de corte en 440 Hz en el dominio del tiempo

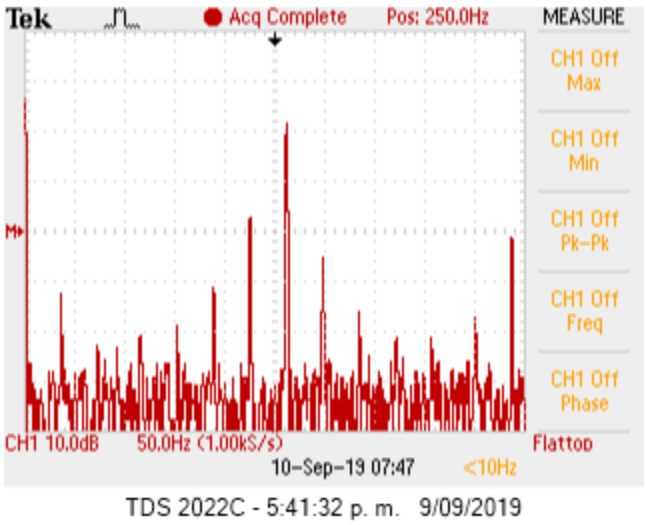


Figura 47: Señal de salida de filtro LPF con frecuencia de corte en 440 Hz en el dominio de frecuencia

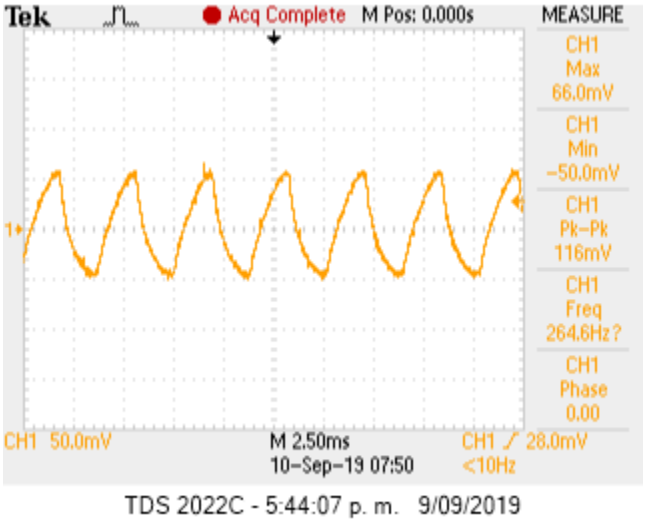


Figura 48: Señal de salida de filtro HPF con frecuencia de corte en 440 Hz en el dominio del tiempo

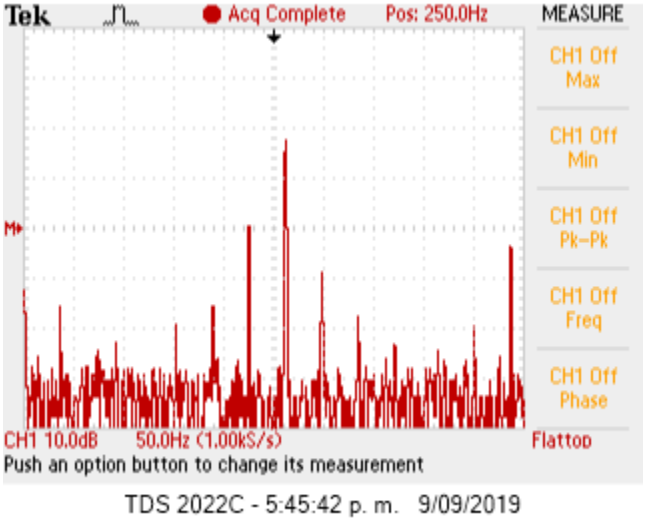


Figura 49: Señal de salida de filtro HPF con frecuencia de corte en 440 Hz en el dominio de frecuencia

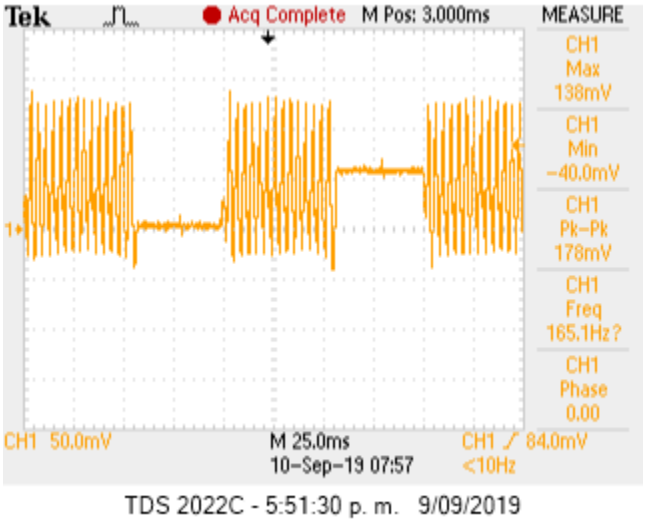


Figura 50: Señal de salida de filtro LFO oscilando a 31Hz con una frecuencia central de 440Hz en el dominio del tiempo

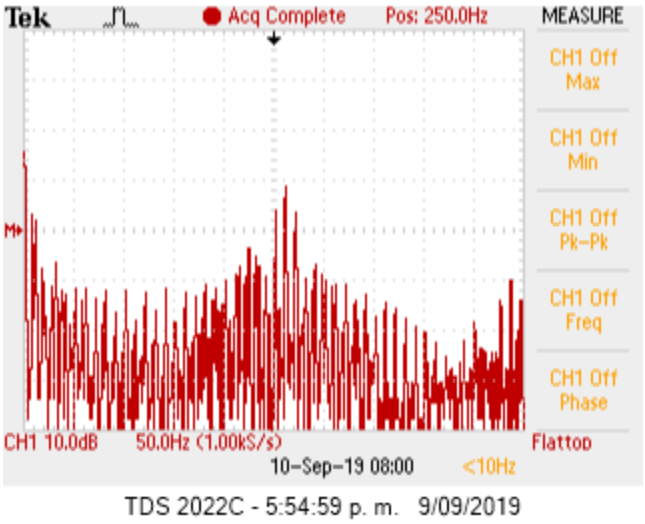


Figura 51: Señal de salida de filtro LFO oscilando a 31Hz con una frecuencia central de 440Hz en el dominio de frecuencia

10.3. Efectos de audio

10.3.1. Reverb

En esta parte se muestran los resultados obtenidos con el efecto de *Reverb*. se utilizó la señal mostrada en la Figura 38 como entrada.

Señal de entrada: señal sinusoidal, diente de sierra y cuadrada combinadas.

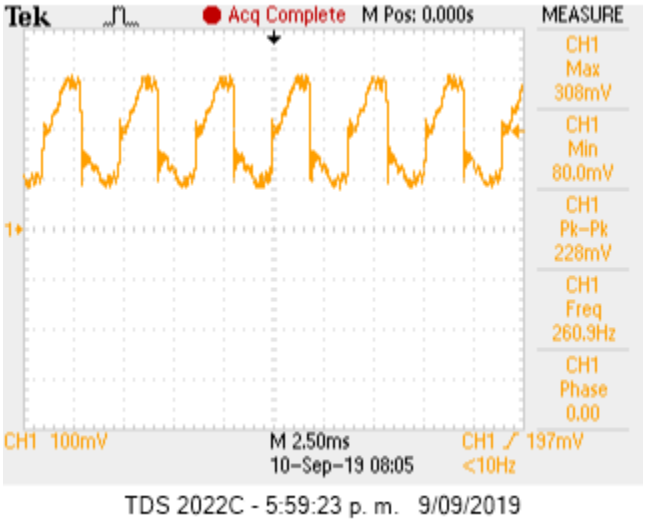


Figura 52: Señal de salida Reverb en el dominio del tiempo

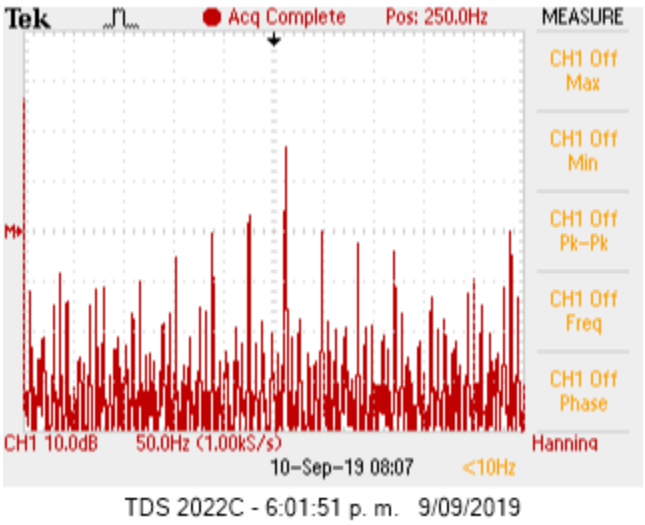


Figura 53: Señal de salida Reverb en el dominio de la frecuencia

10.3.2. Delay

En esta sección se muestran los resultados obtenidos con el efecto de *Delay*. Se utilizó la señal mostrada en la Figura 38 como entrada. En este caso se obtuvo solamente la señal en el dominio del tiempo ya que este efecto no modifica el espectro en frecuencia.

Señal de entrada: señal sinusoidal, diente de sierra y cuadrada combinadas.

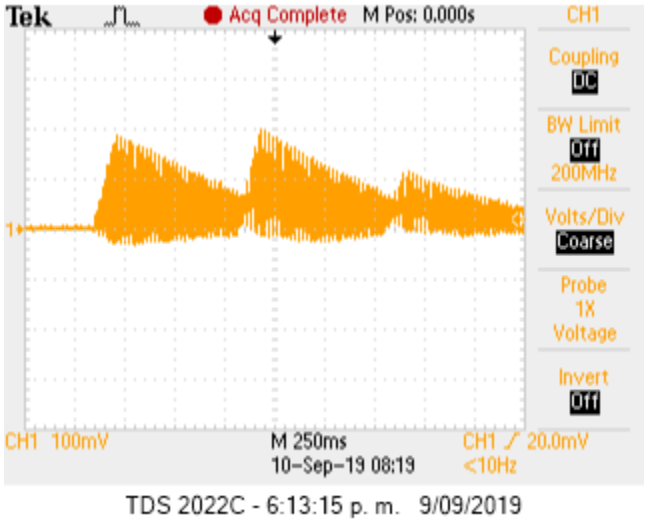


Figura 54: Señal de salida Delay en el dominio del tiempo

10.4. Análisis de desempeño de hardware

En esta sección se exponen los resultados obtenidos al monitorear el rendimiento de la *Raspberry Pi* mientras se procesaban los distintos algorítmos de síntesis. Las pruebas que se hicieron son las siuientes:

- Monitoreo del desempeño al procesar una señal sinusoidal, diente de sierra y cuadrada combinadas, como la que se muestra en la Figura 38.
- Monitoreo del desempeño al procesar una señal sinusoidal, diente de sierra y cuadrada combinadas cada una con su *detune* respectivo con un 50 % de ganancia.
- Monitoreo del desempeño al procesar una señal sinusoidal, diente de sierra y cuadrada combinadas cada una con su *detune* respectivo con un 50 % de ganancia y un filtro *LFO* activado.
- Monitoreo del desempeño al procesar una señal sinusoidal, diente de sierra y cuadrada combinadas cada una con su *detune* respectivo con un 50 % de ganancia, un filtro *LFO*, *delay* y *reverb* activados.

Las pruebas mencionadas anteriormente se escogieron ya que se buscaban configuraciones que presentaran una mayor demanda computacional.

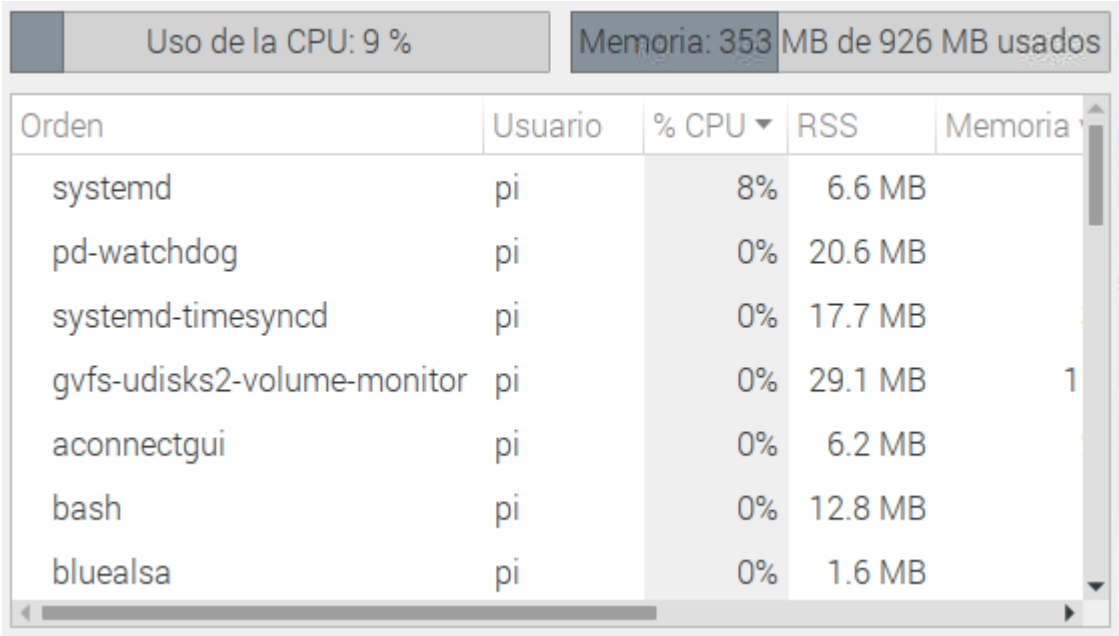


Figura 55: Desempeño de la *Raspberry Pi* en el procesamiento de una señal sinusoidal, diente de sierra y cuadrada combinadas

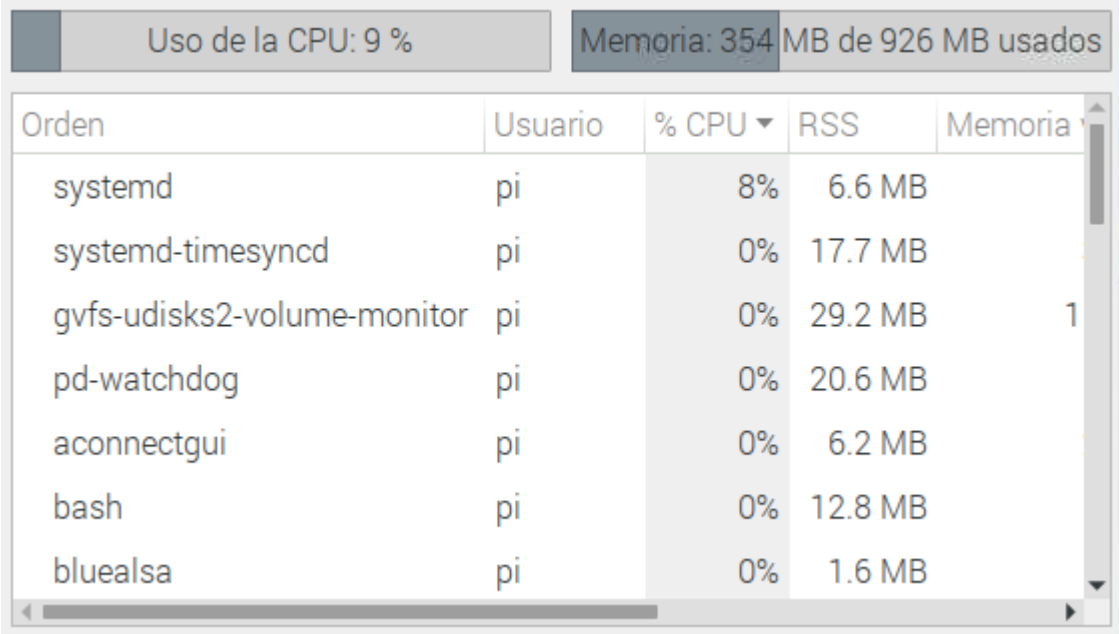


Figura 56: Desempeño de la *Raspberry Pi* en el procesamiento de una señal sinusoidal, diente de sierra y cuadrada combinadas con *detune*

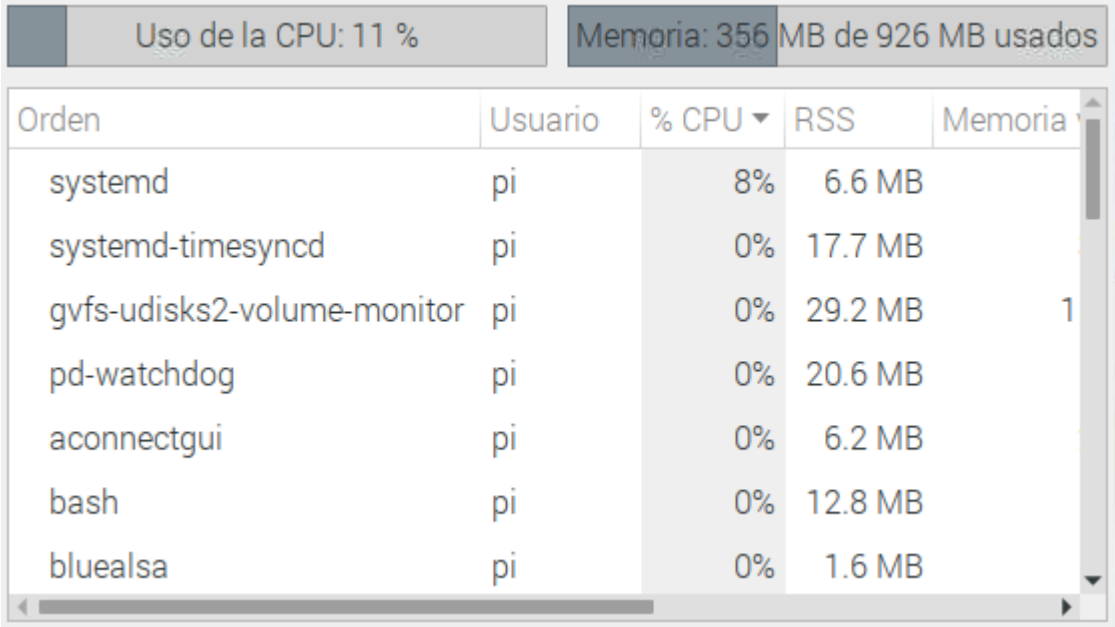


Figura 57: Desempeño de la *Raspberry Pi* en el procesamiento de una señal sinusoidal, diente de sierra y cuadrada combinadas con *detune* y un filtro LFO

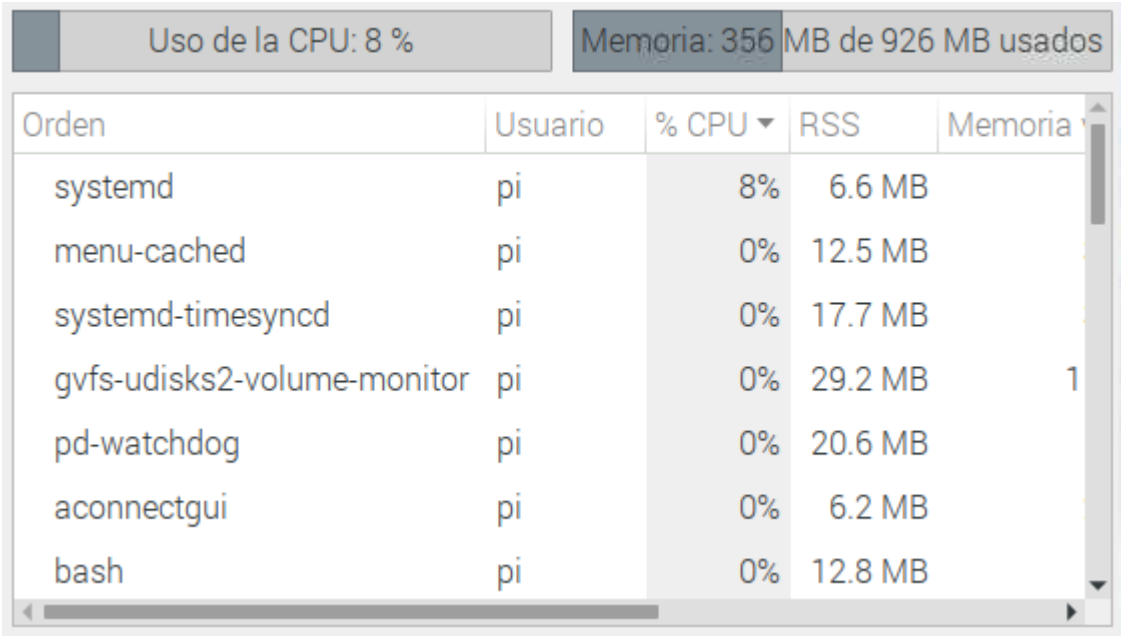


Figura 58: Desempeño de la *Raspberry Pi* en el procesamiento de una señal sinusoidal, diente de sierra y cuadrada combinadas con *detune*, un filtro LFO, *delay* y *reverb*

10.5. Discusión de resultados finales

El objetivo general de este trabajo fue desarrollar un hardware que implemente algoritmos de síntesis aditiva y sustractiva y efectos de audio digitales que permitan modular los parámetros de diseño al gusto del usuario en tiempo real. Para cumplir este objetivo se seleccionaron una serie de efectos digitales que modifiquen hasta cierto punto el espectro de frecuencia de las señal obtenida mediante los modelos de síntesis, se analizó el desempeño de los algoritmos implementados en un procesador *ARM Cortex-A53* de 64 bit y se planteó un diseño físico para una interfaz de usuario.

Para empezar se tomarán las tres señales principales, sinusoidal, diente de sierra y cuadrada mostradas en las Figuras 26, 28 y 30 respectivamente. Como puede observarse en las Figuras 27, 29 y 31 la señal sinusoidal tiene solamente una frecuencia, mientras que las señales diente de sierra y cuadrada tienen un pico en la misma frecuencia, 261.63 Hz, la cual es la frecuencia de la nota de prueba, un Do central, pero tienen varios armónicos en patrones diferentes, estos armónicos corresponden a las frecuencias de los sinusoides de su serie de Fourier respectiva. Basado en este espectro de frecuencias para cada una de las señales, en este diseño particular, cada una de esas señales tiene una función. La señal sinusoidal tiene la función de formar la base a la nota que desea sintetizarse, por otro lado las señales diente de sierra y cuadrada tienen la función de agregar armónicos, los cuales tienen el efecto de cambiar el sonido. Luego en las Figuras 40, 42 y 44 se muestran las señales sinusoidal, diente de sierra y cuadrada cada una con un *detune* al 50 % y en las Figuras 41, 43 y 45 los espectros de frecuencia correspondientes. Como puede observarse estos espectros de frecuencias son parecidos en forma a los de las señales sin *detune* sin embargo, en las señales con detune cada armónico tiene dos armónicos adyacentes, uno a la izquierda y otro a la derecha, practicamente con la misma magnitud. Esta suma de armónicos desfasados proporciona un sonido que en la industria musical se describe con el término *más profundo*. Adicional a esto, si se observan las señales con *detune* en el domino del tiempo puede notarse que esta suma de armónicos también resulta en una modulación de amplitud, esto resulta particularmente visible en la Figura 40, esto se debe a que el desfase de frecuencia entre la señal principal y las dos señales de detune es *pequeña*. En las Figuras 32, 34, 36 y 38 se muestran las distintas combinaciones de las tres señales principales y en las Figuras 33, 35, 37 y 39 los espectros de frecuencia correspondientes. Todo lo mencionado anteriormente forma la parte de síntesis aditiva implementada, la posibilidad de combinar estas tres señales y sus señales de *detune* respectivo, lo que permite *jugar* con los armónicos y la frecuencia en la que se hace énfasis.

En la siguiente parte se muestra la etapa de filtros. Se implementaron tres filtros, un filtro pasa bajas, un pasa altas y un filtro pasa-banda modulado por un *LFO*. En la Figura 46 se muestra la señal de la Figura 38 luego de pasar por el filtro pasa bajas implementado con una configuración de frecuencia de corte en 440 Hz. Como puede observarse en la Figura 47 los armónicos del lado derecho fueron atenuados. Este filtro se diseñó para que siempre estuviera activo, esto es porque el efecto que tiene este filtro resulta particularmente útil en música ya que puede utilizarse un mismo oscilador para intensidades altas, un espectro de frecuencia completo, y para intensidades bajas, solo la parte baja del espectro de frecuencias. También resulta interesante observar la señal en el dominio del tiempo, ya que la señal de entrada fue una combinación de un sinusoide, una señal diente de sierra y una señal cuadrada y la señal de salida fue una señal triangular. En la Figura 48 se muestra un filtro pasa-altas y en la Figura 49 se muestra su espectro de frecuencias respectivo. Como este filtro se encuentra

en serie con el filtro pasa bajas, para tratar de capturar solamente el efecto del filtro pasa altas se configuró el filtro pasa bajas con una frecuencia de corte de 2000 Hz, la cual es la máxima. Como puede observarse en la Figura 48 el componente DC de la señal de salida fue removido, así mismo el lado izquierdo del espectro de frecuencias mostrado en la Figura 49 se atenuó por lo que se concluye que el filtro funciona como se esperaba. Este filtro no es tan utilizado como el pasa bajas, sin embargo puede ser útil para la creación de sonidos específicos por lo cual se implementó. Por último se muestra en la Figura 50 la misma señal de entrada que en los últimos dos filtros pero, con un filtro pasa banda modulado por un *LFO* y en la Figura 51 se muestra el espectro de frecuencias con una persistencia de un segundo. Resulta interesante ver que la variación de frecuencias no solamente influye en la densidad del espectro de frecuencias pero también resulta en una modulación de la amplitud de la señal de salida, este último filtro puede tomarse como un sistema variante en el tiempo.

En la siguiente parte se muestran los resultados de los efectos de audio elegidos, el *delay* y el *reverb*. Se eligieron estos efectos debido a que son de los más utilizados en la industria musical. En la Figura 52 puede observarse la señal de salida luego del efecto tomando como entrada la misma señal utilizada anteriormente, y en la Figura 53 se muestra su espectro de frecuencia respectivo. Para poder analizar y comprender este efecto debe compararse las Figuras anteriormente mencionadas con las Figuras 38 y 39. Se puede observar que la señal de entrada y la señal de salida conservan la misma forma hasta cierto punto, pero la señal de salida tiene una especie de *ruido* impregnado. Al ver y comparar el espectro de frecuencias de estas dos señales puede observarse que el efecto de *reverb* agrega armónicos con baja amplitud a lo largo de todo el espectro frecuencial y esto enriquece la *musicalidad* del sonido obtenido. Por otro lado, en la Figura 54 puede observarse la señal obtenida luego de aplicar el efecto de *delay*. En este caso se muestra solamente la señal en el dominio del tiempo ya que este efecto solamente repite la señal de entrada con una señal envolvente decreciente y no se altera el espectro de frecuencias. Este efecto resulta particularmente útil cuando se necesita *llenar* los espacios vacíos (en sonido).

Por último se encuentran los resultados al monitorear el desempeño del procesador de la *Raspberry Pi* utilizada. Para hacer el monitoreo se utilizaró la herramienta de monitoreo integrada en la distribución de *Linux* utilizada. En las Figuras 55, 56, 57 y 58 se muestra el uso del *CPU*, la memoria y las tareas procesadas al momento para distintas configuraciones. Trataron de elegirse configuraciones que precentaran una demanda computacional alta. Como se observa, el uso del *CPU* y la memoria, aunque fluctuaron durante las distintas configuraciones se mantuvieron en un rango del 8 % al 11 % y de 353 MB a 356 MB respectivamente. Esto era lo que se esperaba ya que se configuró la *Raspberry Pi* para que todos los procesos relacionados con audio se procesaran en la tarjeta de audio externa. Con esto se confirma que la configuración funciona correctamente y se abre la posibilidad de agregar más funcionalidades al sintetizador de audio utilizando el procesador de la *Raspberry Pi*.

CAPÍTULO 11

Conclusiones

- Según la evidencia mostrada en la Figura 15 la cual muestra el software que implementa el motor de síntesis, en la Figura 22 la cual muestra el circuito desarrollado para la interfaz de usuario y en los resultados mostrados en las figuras 26 a la 54 se concluye que se logró desarrollar un hardware que implemente algoritmos de síntesis aditiva y sustractiva y efectos de audio digitales que permitan modular los parámetros de diseño al gusto del usuario en tiempo real
- Según los resultados mostrados en las figuras 55 a la 58 las cuales muestran el uso de la *CPU* y la *RAM* bajo distintas configuraciones se concluye que se logró un buen rendimiento al procesar los algoritmos de síntesis, dejando espacio de procesamiento para agregar funcionalidades extras en una iteración posterior.
- De acuerdo a lo mostrado en el capítulo 8 en la Figura 20, en el capítulo 9 en las figuras 20 y 21 y en el capítulo 10 en las figuras 26 a la 54 en donde se muestra los resultados de los efectos digitales bajo diferentes configuraciones se concluye que se seleccionaron e implementaron una serie de efectos digitales que modifican el espectro de frecuencia de las señales obtenidas mediante los modelos de síntesis.
- Según la evidencia presentada en la última sección del capítulo 8 en la Figura 13 se determina que se planteó un diseño físico portátil que pueda ser utilizado para presentaciones musicales en vivo.

CAPÍTULO 12

Recomendaciones

- Se recomienda para futuras iteraciones implementar una interfaz de usuario utilizando una pantalla touch ya que en los resultados obtenidos se demostró que utilizando la tarjeta de audio externa el procesador tiene la capacidad de realizar otros procesos.
- Se recomienda agregar una señal triangular a la sección de osciladores, experimentar con efectos de *chorus*, *flanger*, *phaser*, *tremolo*, *ring modulation* y explorar la posibilidad de agregar un *looper* multi-canal para hacer un sintetizador más completo.
- Se recomienda investigar los procesos de manufactura óptimos para construir el diseño planteado al final del capítulo 8.

CAPÍTULO 13

Bibliografía

[1] M. Friesen, *Modular music synthesizer*, US Patent 9,530,395, dic. de 2016.

[2] E. G. Briggs y S. L. Veilleux, "FPGA Digital Music Synthesizer", 2015.

[3] G. Geiger, "PDa: Real Time Signal Processing and Sound Generation on Handheld Devices.", en *ICMC*, 2003.

[4] M. Pfaff, D. Malzner, J. Seifert, J. Traxler, H. Weber y G. Wiendl, "Implementing digital audio effects using a hardware/software co-design approach", en *10th International Conference on Digital Audio Effects*, 2007, págs. 1-8.

[5] S. Rostedt y D. V. Hart, "Internals of the RT Patch", en *Proceedings of the Linux symposium*, vol. 2, 2007, págs. 161-172.

[6] J. H. Brown y B. Martin, "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications", en *proc. of the 12th Real-Time Linux Workshop (RTLWS'12)*, 2010, págs. 1-17.

[7] *Frequently Asked Questions*, sep. de 2012. dirección: `https://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions#How_does_the_CONFIG_PREEMPT_RT_patch_work.3F`.

[8] S. Wilson, D. Cottle y N. Collins, *The SuperCollider Book*. The MIT Press, 2011.

[9] *Pd Community Site*. dirección: `https://puredata.info/`.

[10] E. Miranda, *Computer sound design: synthesis techniques and programming*. Routledge, 2012.

[11] W. Pirkle, *Designing Software Synthesizer Plug-Ins in C+*. Taylor & Francis, 2015.

CAPÍTULO 14

Anexos

14.1. Código fuente

A continuación se proporciona el siguiente enlace al repositorio en donde se encuentra todo el código fuente tanto del motor de síntesis como de la interfaz de usuario:

- https://gitlab.com/manu_valenz/midi-portable-synth.git