Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería de Computadores

Pedal de efectos digital para guitarra basado en FPGA

Jose Ángel Gumiel Quintana

Septiembre 2017

informatika fakultatea
facultad de informática

Agradecimientos

Quiero aprovechar este apartado para mostrar mi gratitud a todos aquellos que me han apoyado en el camino: familia, amigos, profesores, compañeros de clase…

Gracias a los que habéis depositado confianza en mí y me habéis dado ánimos, a los que me habéis dado ideas o puntos de vista nuevos y diferentes, sin vosotros todo esto hubiera sido más complicado.

No puedo olvidarme de agradecer a mis padres, Ana y Víctor, y a mis tíos, por el primer ordenador que tuve a los seis años de edad, el cual fue el causante de mi interés por la informática y de que haya elegido esta carrera. Y gracias por haberme ayudado a comprar materiales y dejarme experimentar con la informática y la electrónica, pese a que no siempre todo saliera bien.

Por último, dar también las gracias a mi director de proyecto, Andoni, por haberme brindado la oportunidad de hacer este trabajo, de aprender cosas nuevas y por su ayuda.

Gracias a todos por vuestro apoyo incondicional.

Los proyectos que hemos hecho demuestran lo que sabemos,
los futuros proyectos decidirán lo que aprenderemos.
—Dr. Mohsin Tiwana

Resumen

En este proyecto se implementa un pedal de efectos para guitarra eléctrica sobre la placa de desarrollo Altera-DE1 SoC, que incluye una FPGA (dispositivo lógico programable). Para ello se recurre al procesado de señales digitales. El procedimiento es el siguiente:

Una fuente de sonido (en este caso una guitarra eléctrica) genera una corriente muy débil (entre 100mV y 1V) que es recogida a través de la entrada "Line-In" de la placa. La señal es muestreada y codificada en tiempo real. El cometido del pedal de efectos es procesar dicha señal y aplicarle cambios que consigan alterar el resultado final de la onda. Esto provoca la modificación del sonido recibido inicialmente. Por último, a través de la salida "Line-Out", se obtendrá el sonido modificado en formato analógico, que se podrá escuchar conectándolo a unos altavoces o auriculares.

Los pedales de efectos surgieron ante la necesidad de nuevos sonidos musicales. Llegó un momento en el que las posibilidades que ofertaba el instrumento y el amplificador se quedaron cortas, y surgieron circuitos analógicos que conseguían alterar las propiedades del sonido, ofreciendo así un mayor abanico de posibilidades para que los músicos pudieran encontrar el sonido que buscaban.

En la actualidad, aunque los amplificadores ofrezcan también algunos efectos adicionales, los músicos siguen usando pedales. Hoy en día conviven los pedales analógicos con otros digitales, y también hay algunos que son multi-efectos. Obtener un buen equipamiento sonoro no sale barato.

En este proyecto se ha construido un prototipo de unidad multi-efectos con capacidad de ser modificable y una buena calidad de sonido. Un pedal comercial no es modificable, en el mejor de los casos se puede ajustar para obtener un sonido más personal, pero no dejan de ser ligeros cambios. Al implementar el pedal de efectos en forma digital en una FPGA se consigue una unidad de efectos que está abierta a futuros cambios, y que puede albergar más de un efecto diferente. Esto significaría un ahorro de espacio y podría conllevar un ahorro económico.

Índice

Lista de Figuras y Tablas

FIGURAS

TABLAS

1

Introducción y definiciones

En este primer capítulo se hará una breve introducción para conocer qué es un pedal de efectos, en qué consiste, cuáles son sus aplicaciones… Por otro lado, también se comentan cuáles son las herramientas que se van a utilizar y bajo qué hardware se va a implementar.

1.1. ¿Qué es una unidad de efectos?

Es un dispositivo electrónico analógico o digital que modifica el sonido de un instrumento o de cualquier otra fuente de sonido. Algunos añaden un carácter sutil, aunque otros modifican la señal de una forma más notable.

1.1.1. ¿Dónde se pueden encontrar?

Pedales: Son unidades pequeñas que se actúan con el pie, presionando un interruptor, son muy habituales, tanto para profesionales como para principiantes. Hay muchos tipos de pedales diferentes, se catalogarán más adelante.

Racks: estos suelen ser más complejos y tener diferentes tipos de efectos. Normalmente son más usados en los estudios de grabación y por músicos profesionales. También pueden actuarse con el pie mediante un controlador.

Built-in: Los amplificadores también suelen tener efectos. Los más comunes son los efectos de reverberación y distorsión/saturación. Actualmente hay amplificadores que llevan efectos digitales, lo que permite disponer de varios efectos en una sola unidad.

Multiefectos: Es un único pedal que permite tener un conjunto de efectos predefinidos por el usuario y la combinación de varios de ellos, posibilitando acceder de forma rápida y sencilla, con una sola pisada.

1.1.2. Tipos de efectos:

Hay diferentes tipos de efectos. Existen diferentes formas de catalogarlos, pero se podrían enumerar en 6 tipos:

Tono (timbre): Altera el sonido de la guitarra. Los efectos incluidos que modifican esta propiedad del sonido son las distorsiones y saturaciones (“overdrive”). Hay más efectos que juegan con el tono, como por ejemplo el “fuzz”, que transforma la señal de entrada en una onda casi cuadrada, o los de alta ganancia, que acentúan los agudos mediante la amplificación de estos.

Filtros: Modifican el contenido de la señal de audio que los atraviesa, modificando su espectro (composición frecuencial). Dentro de esta conjunto se encuentran efectos como el ecualizador, el “wah-wah”, “auto-wah” o “envelope filter”, “talk box”…

Tiempo: Los efectos basados en el tiempo añaden retardos a la señal o añaden ecos. Algunos ejemplos de unidades pertenecientes a este grupo son los efectos de reverberación, que tratan de simular el eco natural que se produce en un lugar determinado. Generalmente replican el sonido del eco dentro de una habitación, aunque algunos tienen varios presets. También estaría el efecto de retardo o “delay”, que añade una señal eléctrica duplicada a la señal original con un tiempo de diferencia. Si el intervalo es muy corto, sería un efecto de eco. Otro ejemplo de pedal que se basa en el tiempo es el “loop”, es muy útil cuando no se

dispone de otro músico. Este efecto permite grabar y reproducir un bucle un fragmento de sonido, ofreciendo al músico la posibilidad de tocar diferentes melodías sobre la base que acaba de grabar.

Volumen: Modifican la amplitud de la señal. Estos tipos de efectos son de los primeros que se introdujeron para los guitarristas. Dentro de esta agrupación hay diversos ejemplos, pero cabe mencionar tres de ellos. El más característico es el "boost", o lo que viene a ser una etapa de preamplificación. Este pedal lo que hace es incrementar la amplitud de la onda que sale de la guitarra, es muy útil en momentos en los que el músico desea que su instrumento suene más alto, por ejemplo, durante un solo instrumental. Los hay limpios, pero lo general en un "boost" es que al llevarlo al máximo haya una leve saturación de la señal. Otro efecto de este conjunto es el compresor. En guitarra, la amplitud de la onda de salida también depende de la fuerza con la que el músico rasguee las cuerdas. Este pedal actúa de forma dinámica, ajustando el volumen para que quede igualado. El último ejemplo es el trémolo, que varía la amplitud haciéndola oscilar en el tiempo.

Modulación: El término "modulación" significa que el efecto está basado en el cambio de algún parámetro en el tiempo. El cambio suele estar regulado por un oscilador de baja frecuencia que afecta a la señal. Los pedales de este tipo son un poco especiales, ya que efectúan una combinación de cambios en la ganancia, en la frecuencia y en el tiempo. En esta categoría se incluyen efectos como el "chorus", que produce la sensación de varios instrumentos sonando en conjunto sobre de la señal original, el "phaser", que dobla la señal aplicándole un retraso y sumándosela a la señal original, o el "trémolo", que varía la amplitud de la señal en función del tiempo de forma periódica.

Tono/Frecuencia: Modifican el tono alterando la frecuencia fundamental de una onda acústica o añadiendo nuevos armónicos. Un ejemplo son los "octavadores". Lo que hacen estos circuitos es añadir a la señal original otra réplica exacta pero una o dos octavas por encima o por debajo.

1.1.3. ¿Cómo se utilizan?

Los pedales de efectos se sitúan entre el instrumento y el amplificador. Cuando se usan pedales individuales van conectados entre sí en cadena. Cada dispositivo tiene como mínimo dos posiciones, una de ellas deja pasar la señal sin modificarla (bypass) y la otra hace que la señal pase por el circuito activo, aplicando el efecto y generando una salida distinta a la original.

Al poner las unidades de efectos en cadena se consigue que puedan aplicarse varios cambios en la señal y generar nuevos sonidos. Sin embargo, el orden de colocación puede afectar de diferente forma a la señal, y la combinación de dos pedales en distinto orden puede dar como resultado sonidos distintos.

En la siguiente imagen se ve el esquema de conexionado de los pedales de efectos. En este caso se muestra también un orden a la hora de colocar los efectos. Por ejemplo, suele ser recomendable poner los efectos de amplificación o de distorsión al principio de la

cadena, y los de modulación al final. Sin embargo, no hay ningún orden definitivo, es tarea del músico probar diferentes combinaciones y escoger la que más se adapte a sus gustos o necesidades.

Figura 1.1 Esquema del conexionado en cadena de los pedales de efectos (Pedalera).

1.2. Hardware

Se va a utilizar una FPGA (Field Programmable Gate Array). Se trata de un dispositivo programable que contiene bloques de lógica que pueden ser programados mediante un lenguaje de descripción de Hardware, se usará VHDL.

La placa elegida ha sido la DE1-SoC de Terasic, que incluye una FPGA de la familia Cyclone V de Altera. Los factores determinantes han sido que cuenta con un códec de audio integrado, por lo que puede efectuar el muestreo del sonido de forma nativa, y además, tiene un conversor analógico digital, lo que facilita el uso de potenciómetros para interactuar con la placa. Las resistencias variables serán utilizadas para cambiar algunos parámetros y modificar así los efectos.

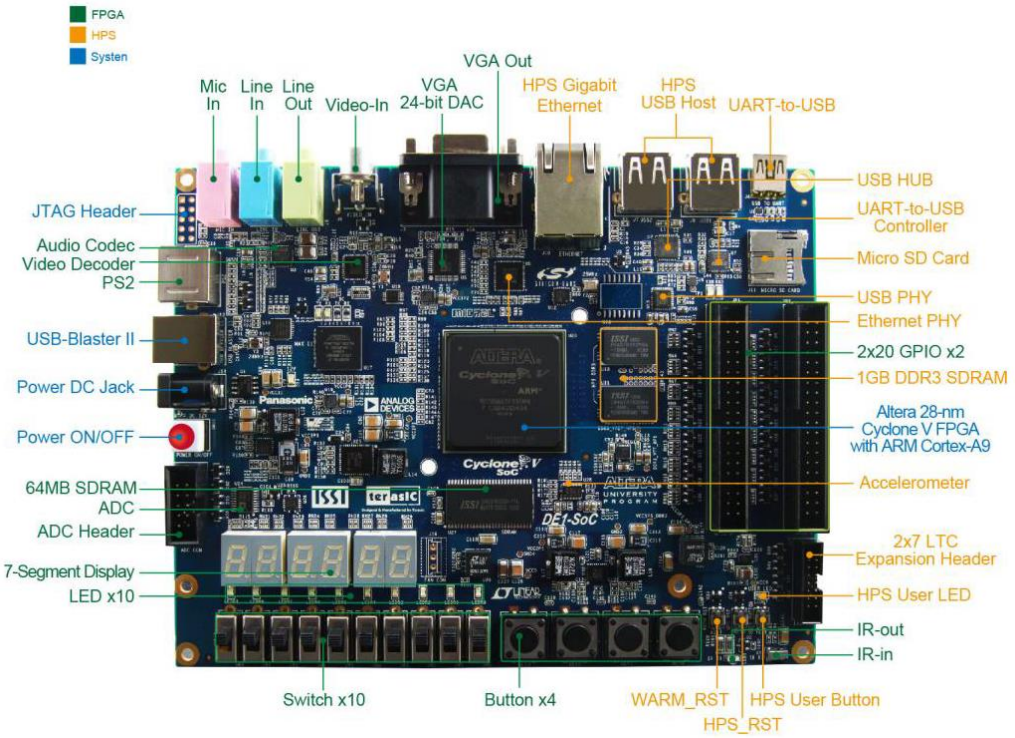


Figura 1.2 Esquema del hardware de Terasic DE1-SoC.

En la imagen superior se observa la placa sobre la que se implementará el proyecto. Es un pequeño esquema que muestra sus componentes principales y su ubicación.

Como se puede observar, esta placa cuenta con dos entradas de audio (una de ellas para micrófono) y una salida de sonido. Para muestrear este tipo de señales tiene un códec que soporta audio en una representación de 24-bits. Para este proyecto se tomarán muestras de 16-bits a una velocidad de muestreo de 48kHz. Teniendo en cuenta que el espectro audible del oído humano está situado aproximadamente entre los 20Hz y los 20kHz, y bajo el amparo del teorema de Nyquist, se puede decir que la tasa de muestreo es adecuada. Se está utilizando calidad CD.

La placa también permite otro tipo de entrada de señales analógicas y trae un conversor analógico-digital (CAD). Esté será usado para añadir otros elementos físicos que el usuario pueda manejar, como por ejemplo potenciómetros o interruptores.

También cuenta con interruptores que se podrán utilizar para interactuar con el hardware y poder hacer diferentes selecciones. Por ejemplo, los interruptores podrán servir para activar o desactivar efectos, y en caso que un efecto tenga más de un modo, poder seleccionar aquel que se desee. Los músicos también suelen tener un pedal denominado “kill switch”, sirve para interrumpir la señal de la guitarra y que no continúe a lo largo de la cadena hasta el amplificador. Tiene más usos, pero la aplicación fundamental es para hacer un cambio de instrumento sin introducir ruido al desconectar y conectar la guitarra. También se puede añadir esta funcionalidad al prototipo.

1.3. Software

Para el desarrollo de este proyecto se está utilizando el siguiente software:

- ModelSim SE-64 10.5
- Quartus Prime 15.1 Standard Edition
- Altera Monitor Program 15.1
- National Instruments MultiSim 13.0
- Mindjet MindManager 17

Los tres primeros programas se utilizan con la FPGA.

ModelSim es un entorno de simulación. Permite comprobar la sintaxis del código escrito en HDL (Hardware Description Language) y hacer pruebas antes de compilar el programa para grabarlo en la FPGA.

Quartus es una herramienta para el análisis y síntesis de diseños en HDL. Además, permite obtener los diagramas RTL del hardware diseñado. Este software se utiliza también para compilar el código y grabarlo en la FPGA. En la versión utilizada hay soporte multiprocesador, por lo que se usan todos los núcleos de la CPU del sistema, acelerando así el tiempo de compilación.

Altera Monitor Program se utiliza para compilar, ensamblar, descargar y depurar programas para el Nios II, un procesador embebido de 32 bits diseñado específicamente para la familia de FPGAs de Altera.

Adicionalmente, se ha usado el programa MultiSim para diseñar circuitos analógicos y hacer pruebas de su funcionamiento antes de construirlo en una placa de puntos.

También se ha usado el software MindManager para el apartado de planificación, ya que esta herramienta permite hacer diagramas, y se ha utilizado para crear el EDT y el diagrama de Gantt.

1.4. Justificación del proyecto

Siempre he tenido un interés especial por el sonido y cualquier cosa que esté relacionada con este. Hace siete años empecé con un circuito analógico de amplificación. El tener una guitarra y descubrir los diferentes elementos que usan los músicos para cambiar su sonido me llevó a investigar, y acabé construyendo pedales de efectos. Esto me ha servido para estudiar y aprender el funcionamiento de algunos circuitos analógicos. Transversalmente me ha dado otras aptitudes, como el aprender una disciplina de trabajo, manejar distintas herramientas y también me ha enseñado a pensar y plantearme problemas y soluciones.

Dado que estudio Ingeniería Informática, y estoy en la especialidad de Ingeniería de Computadores, pensé que este proyecto sería una buena forma de aunar dos de mis

pasiones. Además, me serviría para aprender nuevas lecciones y poner en práctica otras que he aprendido durante mi formación en la carrera.

La idea fue propuesta al profesor Andoni Arruti, quien me dio su visto bueno y me planteó utilizar una FPGA como plataforma de desarrollo.

1.5. Problema inicial

Hoy en día se pueden encontrar en el mercado una gran variedad de efectos y en diferentes rangos de precio. Se pueden conseguir tanto efectos independientes, que podrán formar una cadena conectando varios pedales en serie, como estaciones multi-efectos. Sin embargo, la variedad para un novato puede hacer complicada la tarea de escoger su equipo.

Los músicos tienden a buscar un sonido con el que sentirse a gusto, que incluso los pueda caracterizar. Además, dependiendo del estilo musical que toquen, escogerán diferentes configuraciones. Esto también afecta a los pedales.

El diseñar un pedal digital sobre una FPGA permite lo siguiente:

- Tener un único dispositivo que permita generar múltiples efectos, ahorrando espacio.
- Disponer de un pedal completamente reconfigurable,
- Poder añadir nuevos efectos o modificar los ya existentes, de forma que se adapten al músico.

1.6. Objetivos del proyecto

El objetivo principal es crear un sistema de sonido que actúe como procesador de efectos de guitarra. Para ello se utilizarán técnicas de procesado de sonido digital. El desarrollo del proyecto también tiene otros objetivos transversales que serán fundamentales para llegar al objetivo final. Destacan los siguientes:

- Ganar conocimiento adicional sobre el tratamiento de señales y su digitalización.
- Aplicar la experiencia con circuitos analógicos al entorno digital.
- Profundizar en el aprendizaje del diseño de Hardware.
- Adquirir más conocimientos para describir Hardware en VHDL.
- Estudiar los comportamientos de las señales bajo un circuito analógico para poder replicarlas de forma digital, manteniendo la máxima fidelidad a su homóloga.

2

Planificación

Apartado dedicado a la gestión del tiempo invertido en completar el proyecto. Incluye cómo se han dividido las tareas a realizar y su planificación, así como información sobre la metodología de trabajo utilizada.

2.1. Introducción

El trabajo se ha realizado a lo largo del presente curso. Fue iniciado en Noviembre de 2016, y se ha ido avanzando en función del tiempo disponible, dependiendo de la carga lectiva de las diferentes semanas. Se puede decir que se ha trabajado sin prisa, pero sin pausa.

2.2. Alcance

Se pretende cumplir totalmente con las tareas representadas en el EDT. Al finalizar el proyecto se espera que, dada una señal de entrada, ésta pueda ser modificada mediante la aplicación de los efectos de amplificación, distorsión, overdrive y trémolo.

Adicionalmente, el proyecto queda abierto a cualquier otro tipo de mejoras, ya sea a través de la implementación de nuevos efectos o de otras herramientas que faciliten el uso del producto final, como podrían ser un display, que simplifique su uso y aporte información, u otros medios de actuación que no sean manuales, switches que se puedan actuar con los pies.

2.3. Estructura de Descomposición del Trabajo (EDT)

El EDT es una herramienta que consiste en la descomposición jerárquica del trabajo a realizar. El propósito de éste es organizar y definir el alcance del proyecto. La estructura utilizada para el desarrollo de este proyecto es la siguiente:

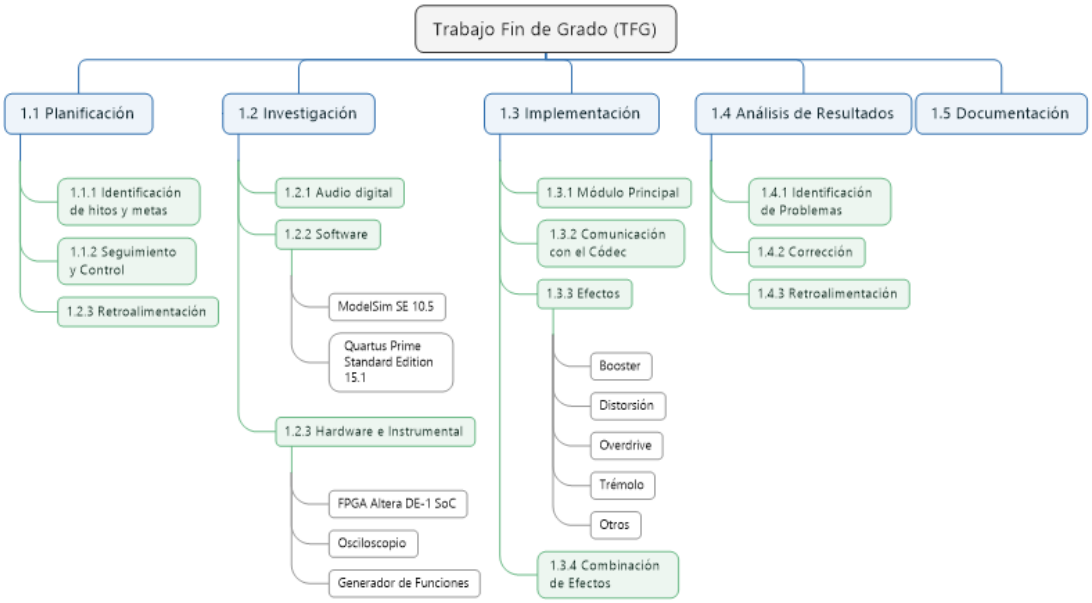


Figura 2.1 Primer EDT del proyecto.

En el mes de Abril se terminaron de diseñar los cuatro efectos propuestos en el anterior diagrama. Surgió la idea de dejar de lado el diseño de módulos de hardware y probar a diseñar otro tipo de efectos en software con el procesador Nios II. Debido a este cambio, la estructura del EDT varía un poco.

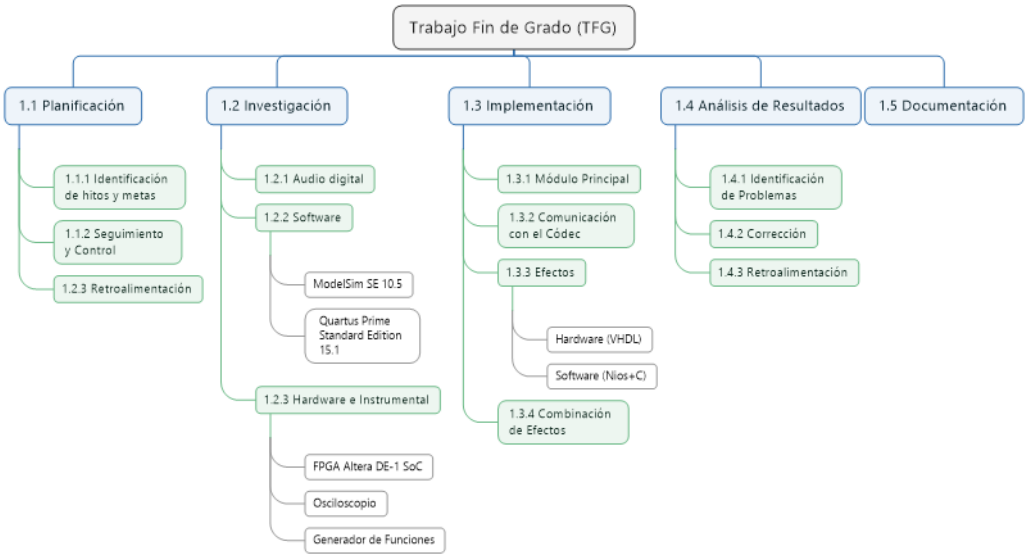


Figura 2.2 Segundo EDT del proyecto.

2.4. Diagrama de Gantt

El diagrama de Gantt es una herramienta gráfica que tiene como finalidad exponer el tiempo de dedicación previsto para realizar las tareas a lo largo de un tiempo determinado. Se muestra en la siguiente figura.

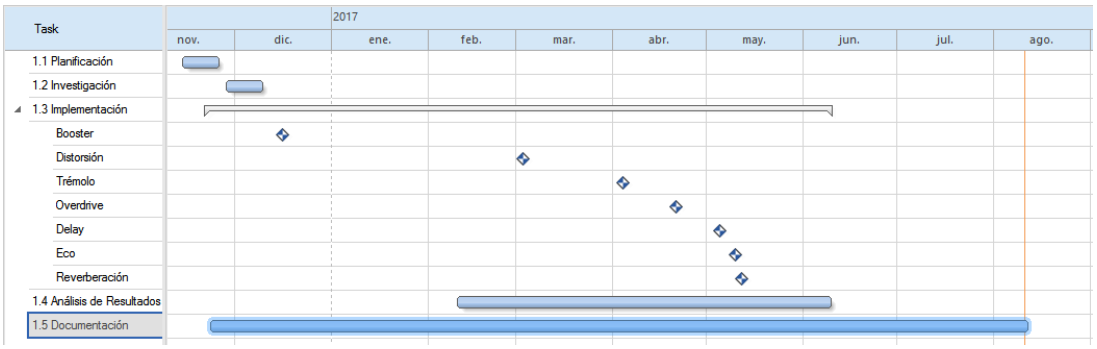


Figura 2.3 Diagrama de Gantt

La primera fase es la planificación del proyecto y hacer un estudio de viabilidad. Por otra parte, tiene que haber una investigación previa para profundizar en la materia y aprender a manejar las herramientas que se utilizarán, tanto en programas informáticos como en material de laboratorio.

La documentación abarcará todo el transcurso del proyecto. Todo lo que se haga quedará reflejado en la memoria.

La implementación de los efectos está prevista que concluya en el mes de Junio, dejando margen para que se puedan analizar los resultados y hacer correcciones o modificaciones.

El fin del proyecto está esperado para el mes de agosto, dejando también un margen para posibles correcciones antes del cierre.

Puede parecer confuso que en un diagrama de Gantt aparezcan los hitos, sin embargo, se observa como la fase de implementación tiene como subsecciones los efectos diseñados. El hito representa el momento en el que ese efecto cumple con las expectativas y por lo tanto se da por terminado.

2.5. Estimación del tiempo

El TFG se corresponde con 12 créditos, lo que equivale a 300 horas de trabajo. A grandes rasgos se ha hecho una estimación sobre el tiempo que se dedicarían a cada una de las tareas principales. Se puede observar en la siguiente tabla.

Tareas	Tiempo (h)
Planificación	4h.
Investigación	34h.
Implementación	150h.
Análisis	20h.
Documentación	80h.
TOTAL	**288h.**

Tabla 2.1 Tiempo estimado para cada conjunto de tareas.

En el capítulo de “Conclusiones”, aparecerá esta misma tabla más detallada con los tiempos reales del proyecto, no los teóricos.

2.6. Comunicación con el tutor y Sistema de Información

La comunicación con el tutor ha sido mediante reuniones y a través del correo electrónico.

Las reuniones no han sido siempre planificadas ni han tenido una periodicidad concreta, se han realizado cuando ha sido necesario, ya fuera por dudas, para analizar los resultados obtenidos o para decidir cuáles serían los pasos siguientes a dar.

Se ha utilizado el correo electrónico para concretar días y horas de reunión, aunque en otros casos, también ha existido la posibilidad de ir al despacho del tutor para solventar las dudas respecto al proyecto.

En cuanto al sistema de información para intercambiar el trabajo realizado, se decidió utilizar Google Drive. Además, se han hecho copias de seguridad periódicas de toda la información perteneciente al proyecto en distintos medios, tanto físicos como virtuales.

3

Audio Digital

Este capítulo sirve al lector como introducción al procesado digital de señales, en este caso audio. Se explican los procesos que hay que realizar para que una señal analógica sea discretizada y pueda ser reconstruida manteniendo fidelidad con respecto a la original.

3.1. Introducción al Procesado Digital de Señales

La base del proyecto es el procesamiento digital de señales, en este caso de audio (aunque también se aplica para tomar datos de los potenciómetros u otros elementos analógicos). El procesado digital de señales consiste en la modificación matemática de una señal. Se trabaja en dominios discretos, generalmente en el dominio del tiempo y de la frecuencia.

Se puede trabajar con señales analógicas, pero al tratarse de un sistema digital habrá que discretizarlas de modo que sean compatibles con el procesamiento digital. Para ello se realiza un proceso que consta de tres partes: muestreo, cuantificación y codificación.

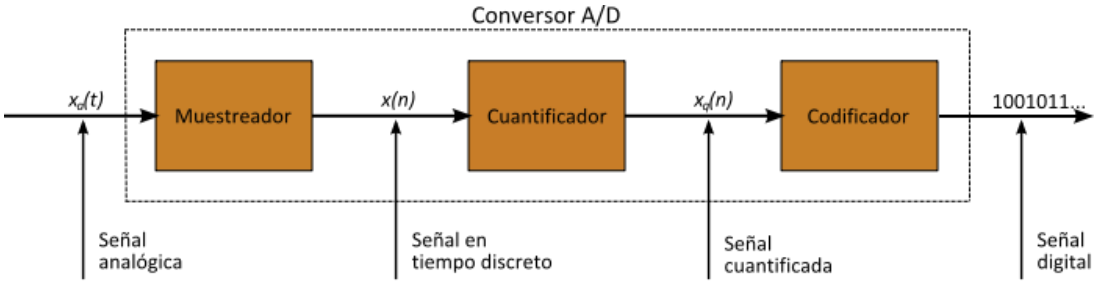


Figura 3.1 Esquema del proceso de conversión analógico-digital.

3.1.1. Muestreo

Una señal analógica puede tomar infinitos valores según la precisión con la que se quiera representar. En el mundo digital no se pueden representar con total exactitud, pero sí que se puede realizar una representación bastante precisa de la forma de la señal. El muestreo es una de las partes del proceso de digitalización de las señales. Se obtienen muestras de una señal analógica a una frecuencia constante aplicando el Teorema de Nyquist. Posteriormente se aplica la cuantificación, que asigna un valor binario a la amplitud de esa muestra.

3.1.1.1. Teorema de Nyquist

Es un teorema fundamental de la teoría de la información. Éste demuestra que la reconstrucción exacta de una señal periódica continua es matemáticamente posible si la señal está limitada en banda y la tasa de muestreo es al menos dos veces el ancho de banda (AB) de la señal analógica.

A la frecuencia *2*AB* se la denomina "razón de muestreo de Nyquist". Si ésta no se cumple, existirán frecuencias cuyo muestreo coincidirán con otras, a este fenómeno se le conoce como "aliasing" o solapamiento. El aliasing impide recuperar correctamente la señal cuando las muestras de ésta se obtienen a intervalos de tiempo demasiado largos. La forma de la onda recuperada presenta pendientes muy abruptas y aparecen frecuencias que no guardan parecido con la onda original. Éste fenómeno afecta más a las frecuencias altas, que se pierden antes, por lo que los tonos agudos se verán más afectados por el aliasing.

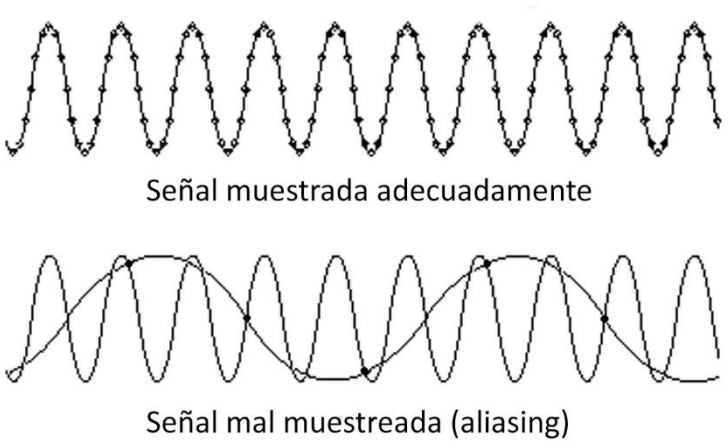


Figura 3.2 Efecto aliasing.

En la anterior imagen se observa el muestreo de una señal sinusoidal. La primera se está muestreando a una frecuencia adecuada, por lo que la forma de la señal resultante es una reconstrucción exacta. Sin embargo, en el segundo caso se está muestreando a una frecuencia insuficiente, por lo que la señal resultante dista mucho del aspecto de la original. Esto significa que no es representativa, y que está mal muestreada.

El espectro audible del oído humano tiene un rango que va desde los 20Hz hasta los 20kHz, este margen varía según la persona y se reduce con la edad. Conforme al teorema de Nyquist, muestrear la señal a 40kHz sería suficiente para reconstruir una señal de sonido aceptable al oído humano. Para este proyecto se está muestreando a 48kHz, por lo que se determina que la frecuencia elegida es más que suficiente.

3.1.2. Cuantificación

El muestreo de una señal continua-analógica da como resultado una señal discreta en el tiempo. Mediante el anterior proceso se ha discretizado el tiempo, sin embargo, las amplitudes de las muestras obtenidas pueden tener infinitos valores. El proceso de cuantificación consiste en transformar el rango continuo de las amplitudes de las muestras de la señal de entrada en un conjunto finito de amplitudes. Normalmente, este conjunto está uniformemente distribuido (incremento constante) y se habla de cuantificación uniforme.

Durante el proceso de cuantificación se mide el nivel de tensión de cada una de las muestras obtenidas, y se les atribuye un valor finito de amplitud, seleccionado por aproximación dentro de un margen de niveles previamente fijado. En este proceso uno de los factores importantes es la resolución, la cual indica el número de valores discretos que puede producir sobre el rango de valores analógicos.

El proceso de cuantificación es un proceso con pérdidas. Debido al hecho de que se le asigna el mismo valor de reconstrucción a cualquier valor de entrada que pertenezca al

mismo intervalo de cuantificación, se produce un error que depende de la señal de la señal de entrada y del valor de reconstrucción. Es un error intrínseco a la cuantificación, siempre que se introduce una etapa cuantificadora se produce este denominado "*Error de Cuantificación"*. La señal resultante tras este proceso es diferente a la señal eléctrica analógica y añade ruido. Si el ruido de cuantificación se mantiene por debajo del ruido analógico de la señal a cuantificar, el proceso cuantificador no tendrá ninguna consecuencia sobre la señal de interés.

La resolución determina la magnitud del error de cuantificación, y es que cuanto mayor sea la resolución menor será la diferencia en voltios entre un valor digital y su inmediatamente posterior. Con más resolución se consigue más precisión con respecto a la onda original.

Pese a que la señal ha sido digitalizada, aún no se ha traducido al sistema binario, esto se hace en la siguiente etapa, codificación.

3.1.3. Codificación

La codificación digital consiste en la traducción de los valores de tensión eléctrica analógicos, que ya han sido cuantificados, al sistema binario mediante códigos preestablecidos. Esta traducción es el último de los procesos de la conversión analógico-digital. El resultado es un sistema binario.

Normalmente se utilizan números en complemento a dos para representar todo el rango de valores discretos de la cuantificación uniforme, que suele ser una potencia de dos, y por ello, la resolución es expresada en bits. Para este proyecto se está usando una resolución de 16 bits, por lo que hay 65.536 valores para representar la amplitud de la onda cuantificada.

3.1.4. Procesado

Una vez que la señal ha sido digitalizada llega el turno de efectuar el procesado o modificación de la señal.

El procesado puede tratar de aplicar filtros o mejorar la señal original, pero en este caso lo que se intentará hacer en esta etapa es aplicar una modificación en la señal para que la salida sea similar a la que logran efectuar sus homólogos, los pedales analógicos.

Para este apartado hay que tener en cuenta cómo se representa la señal. Para este proyecto, la forma de representación utilizada es binario en complemento a 2, y se utiliza una precisión de 16 bits.

A la hora de hacer operaciones matemáticas y comparaciones para efectuar cambios, hay que tener esto presente. La onda de audio se representa mediante valores negativos y positivos. Y si queremos hacer una comparación, por ejemplo, hacer un corte cuando se supere un umbral determinado, hay que conocer a qué valor se corresponde. Esto hay que

tenerlo en cuenta tanto en la parte positiva, como en la negativa, es por ello que hay que conocer cuál es el sistema de representación numérica con el que se está trabajando. En la siguiente se ilustra con un ejemplo de señal sinusoidal con una precisión de 4bits.

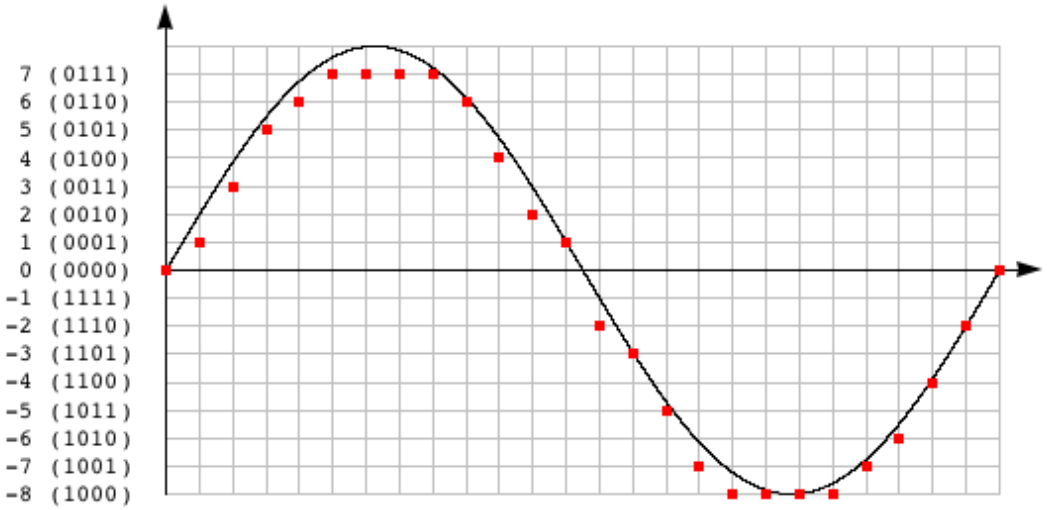


Figura 3.3 Señal sinusoidal muestreada en 4bits en complemento a dos.

3.2. Procesado de audio en Terasic DE1-SoC

Como se ha mencionado anteriormente, la placa utilizada consta de un códec que permite el procesado de audio. Para poder usar este componente, es necesario programar en la FPGA un controlador que permita la comunicación con él.

El códec tiene dos parámetros fundamentales, que son la frecuencia de muestreo (kHz) y la resolución (bits). También se puede configurar para sonido mono o estéreo. En este caso, dado que la fuente utilizada es una guitarra y su salida es mono, la configuración del códec se adaptará a ésta.

La configuración se efectúa a través de un bus I2C, y los datos se intercambian a través de líneas de datos serie. Existe un conjunto de señales de control que facilitan su uso.

Se desea efectuar tres tareas, que son la configuración del códec, la lectura de muestras para la grabación y la escritura de muestras la reproducción. Por tratarse de rutinas independientes se han dividido en tres módulos:

- AU_SETUP: Configura el códec a través del bus I2C en la inicialización del sistema. (Señales I2CSCLK y I2CSDAT)
- AU_OUT: Envía datos al códec. (Señales ADCLRC, BCLK y DACDAT)
- AU_IN: Recibe datos del códec. (Señales DACLRC, BCLK y ADCDAT)

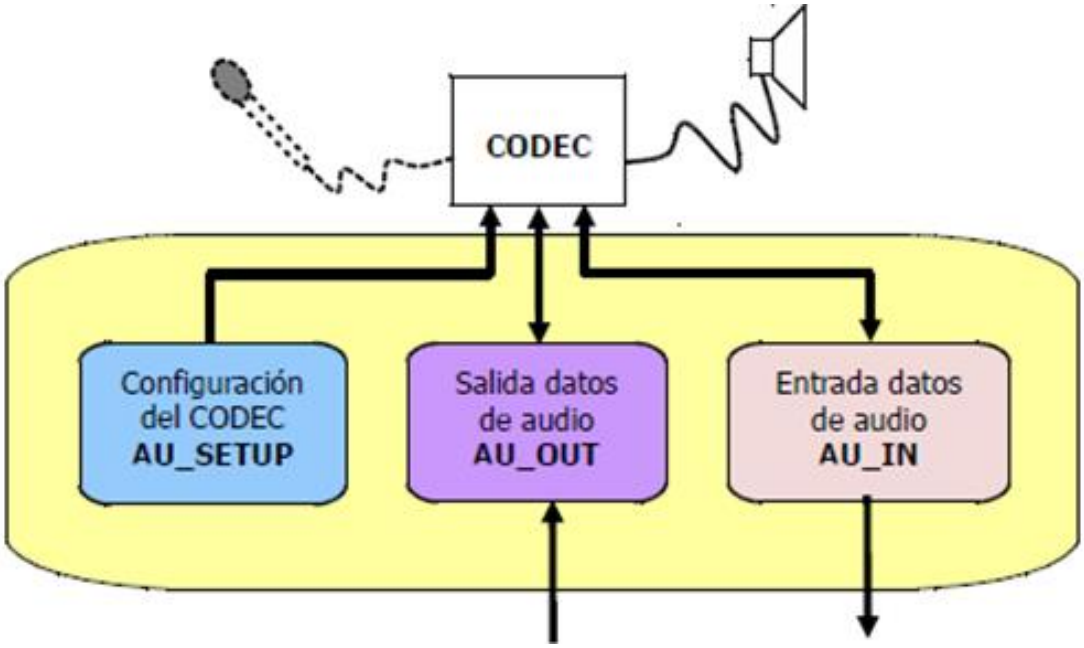


Figura 3.4 Esquema de módulos que interactúan con el códec.

En la imagen se observa el esquema de los tres módulos. En el apartado de implementación se explicará en más profundidad el funcionamiento de estos tres módulos y cómo interactúan con los efectos diseñados.

3.3. Procesado de señales analógicas en Terasic DE1-SoC

La inmensa mayoría de pedales analógicos cuenta con potenciómetros que permiten interactuar con éste hasta obtener el sonido deseado, otorgando al músico una mayor versatilidad. En este proyecto se pretende diseñar réplicas digitales de dichos efectos, por lo que será necesario utilizar otras señales analógicas, además de la de audio.

La placa utilizada consta de un Conversor Analógico-Digital (CAD) de 8 canales, con una resolución de 12 bits y una frecuencia de hasta un millón de muestras por segundo. Para este proyecto, las entradas analógicas van a ser un número de voltajes que se controlan a través de potenciómetros, aunque un CAD se puede utilizar para recibir información de otras fuentes, como podrían ser sensores.

El rango de voltaje para la entrada analógica puede ir de 0V a 2,5V o de 0V a 5V, dependiendo del rango que se seleccione. Las ocho entradas van conectadas a un cabezal de 2x5 pines situado en la placa. En el apartado de implementación se explicará más en profundidad tanto el conexionado como el funcionamiento.

4

Adaptación de Impedancias

Durante la realización del proyecto se ha encontrado una pérdida de matices tras el procesado. Esto se debía a un problema de adaptación de impedancias. En este capítulo se explica qué es la impedancia, qué problema existe y cómo se ha solventado.

4.1. La impedancia

La impedancia (Z) es una medida de oposición que presenta un circuito a una corriente cuando se aplica una tensión. La impedancia extiende el concepto de resistencia a los circuitos de corriente alterna (CA), y posee tanto magnitud como fase, a diferencia de la resistencia, que sólo tiene magnitud.

Por definición, la impedancia es la relación (cociente) entre el fasor tensión y el fasor intensidad de corriente:

$$Z = \frac{V}{I}$$

Donde Z es la impedancia, V es el fasor tensión e I corresponde al fasor intensidad.

4.1.1. Problema inicial

En este proyecto se va a usar la entrada de audio Line-in de la placa DE1-SoC. En el manual de usuario de la placa no aparece el dato de la impedancia que tiene la entrada de audio, sin embargo, se ha consultado con Terasic a través de correo electrónico y comunican que el valor de ésta es de 9,4K Ohmios.

Consultando la ficha técnica del códec WM8731 se observa que, en el apartado dedicado a esta entrada de audio, es un componente diseñado especialmente para recibir señales a *nivel de línea*. Esto significa que está preparado para recibir señales con más fuerza que las que, en este caso, genera un instrumento musical como la guitarra. Estas señales también tendrán una impedancia más baja.

Cuando se envía la señal de un instrumento a un dispositivo que requiere de una entrada de nivel de línea se obtiene un sonido débil, con un procesado inadecuado y existe ruido cuando se trata de amplificarlo.

Antes se ha mencionado la fuerza de la señal, esta se puede medir como un voltaje en Corriente Alterna, sin embargo, la forma más utilizada es el decibelio (dB). En la siguiente tabla se muestra una comparativa entre el nivel de línea y de instrumento.

	dbV	Voltios (RMS)
Nivel de línea	0dBV	1V
Nivel de instrumento	-22dBV	0,08V

Tabla 4.1 Comparativa entre nivel de línea y de instrumento.

La unidad dBV es un ratio logarítmico que tiene como referencia el voltaje de V_0=1V=0dBV.

Aunque el nivel de instrumento es una estimación, se observa que la diferencia entre ambos es significativa. Además, los instrumentos que usan pastillas electromagnéticas para transformar la vibración de las cuerdas en señales eléctricas, tienen una impedancia elevada, en torno a los 7-15K Ohm de media. Cuando se toca un instrumento electroacústico, las pastillas generan una señal que se corresponde exactamente con las notas que están siendo tocadas. Debido a las propiedades de la bobina, hay frecuencias que

son más fáciles de crear que otras. La resistencia de una bobina depende de las frecuencias de la señal que la recorre, y a esta interdependencia se le denomina impedancia. Esto significa que algunas frecuencias oponen menos resistencia que otras, es decir, la impedancia varía en función de la frecuencia, dependiendo de las notas que sean tocadas en cada momento.

Volviendo de nuevo al problema original, la FPGA tiene una entrada de audio cuya impedancia es pequeña en relación a la impedancia de salida que ofrece un instrumento musical. Esto se traduce en que las frecuencias más altas de la guitarra se van a perder, dando como resultado un sonido pobre, sin respuesta en agudos y apagado.

4.1.2. Solución

La solución al problema anteriormente planteado es acoplar la señal, pero no de cualquier forma. En telecomunicaciones se usaba el acoplamiento de impedancias del mismo valor porque, cuando la impedancia de salida de un circuito es igual a la de entrada de otro, se produce la máxima transmisión de potencia. En ese caso se trataba de llevar una señal a largas distancias, por lo que la potencia era un factor importante. El caso actual es diferente, lo que se desea es no atenuar el voltaje de la señal original, es decir, que la señal que reciba la FPGA para procesar sea lo más parecida a la original.

Para ello se recurre al “bridging” o “puente” de impedancias. En audio esta técnica consiste en una conexión en la que la impedancia de entrada es mucho más alta que la impedancia de salida de la fuente, como mínimo la impedancia de entrada será 10 veces más alta que la de la fuente. El puente sirve para minimizar la pérdida de corriente y maximizar el voltaje de la señal a través de la carga.

En el caso de la guitarra no se cumple esta situación de forma natural, (salvo en guitarras con pastillas activas, cuya impedancia es de alrededor de los 100 Ohmios), así que hay que recurrir a un puente de impedancias y manipular la señal para obtener una impedancia baja de salida que se acople perfectamente a la entrada de la FPGA.

Una forma de conseguir que el instrumento esté a nivel de línea es mediante el uso de cajas de inyección directa. Suelen ser usadas en estudios de grabación, pero tienen el inconveniente de ser caras y el circuito tiene cierta complicación. Otra forma es mediante un buffer, tienen una electrónica más sencilla y no suponen mucho coste.

4.2. Adaptación de impedancias

Tras el análisis anterior se ha decidido utilizar un buffer. Para ello se ha planteado un circuito, que ha sido simulado con la herramienta NI Multisim, y tras observar su funcionamiento teórico se ha construido. En las siguientes páginas se encuentra todo el proceso detallado.

4.2.1. El buffer

Un buffer es un dispositivo electrónico que sirve para hacer adaptación de impedancias entre circuitos. Se utilizará para transferir una tensión desde la guitarra (alta impedancia) a la FPGA (baja impedancia). El buffer impide que el segundo circuito cargue demasiado al primero, provocando un funcionamiento incorrecto. Un buffer ideal tiene una resistencia de entrada infinita y la resistencia de salida es cero. La ganancia es de 1, por lo que el voltaje de entrada no varía respecto al de salida.

Para llevar a cabo esta implementación se ha decidido trabajar con amplificadores operacionales, aunque otra opción habría sido usar transistores de tipo JFET. Los operacionales funcionan en corriente continua y son capaces de amplificar una señal. Son adecuados para esta práctica debido a su alta impedancia de entrada y a su baja impedancia de salida.

El esquema más básico de un amplificador de ganancia 1 es mediante una retroalimentación negativa. Se puede ver en la siguiente figura.

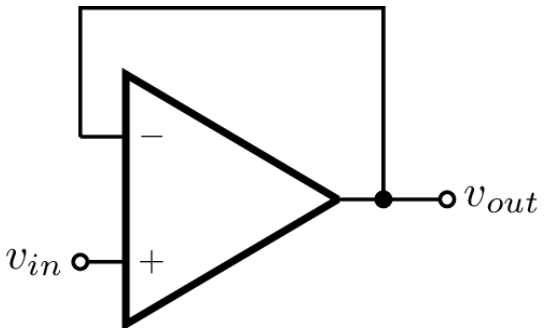


Figura 4.1 Amplificador operacional de ganancia 1.

Basado en estos principios teóricos, se ha realizado el diseño de un circuito que solucione el problema planteado. En los siguientes apartados se facilita el seguimiento del trabajo realizado, acompañado de los esquemas y modificaciones que han surgido.

4.2.2. Circuito 1

Se ha construido un buffer utilizando el amplificador operacional TL071. El amplificador operacional está indicado para aplicaciones de amplificación de audio de alta calidad, admitiendo señales de alta impedancia y dando como resultado una salida de baja impedancia.

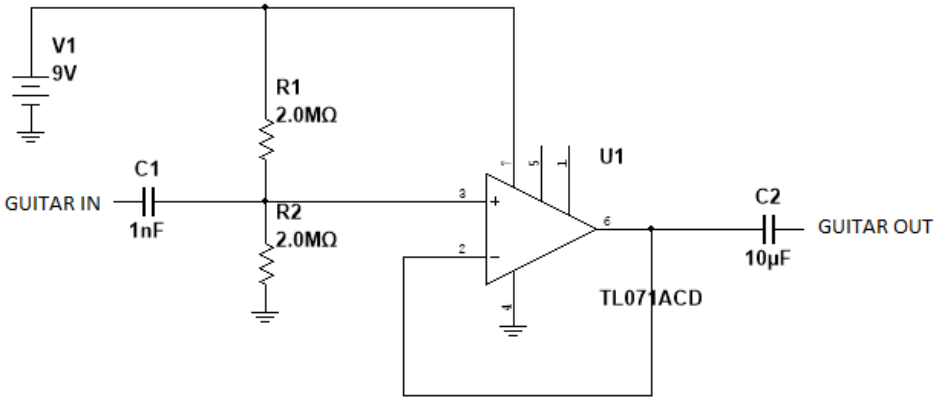


Figura 4.2 Adaptador de impedancia.

El circuito tiene por un lado la alimentación, y por otro lado el procesado de la señal. En la entrada lleva un condensador cerámico no polarizado de 1nF, se usa para filtrar las posibles señales de corriente continua que pudieran interferir con la señal de entrada. Lo mismo ocurre con el condensador situado a la salida, el amplificador operacional es otra etapa del circuito, y también hay que filtrarla ante la posibilidad de que introduzca frecuencias no deseadas. Los condensadores están actuando como filtros pasa-alto.

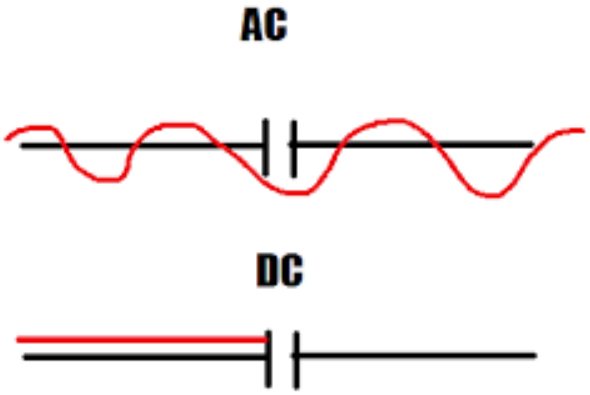


Figura 4.3 Condensador filtrando una señal de corriente continua.

Un condensador filtra la corriente continua de la forma en la que se observa en la imagen. La corriente alterna, que es la señal de audio, puede atravesar el condensador, sin embargo, la corriente continua no.

Las dos resistencias actúan como un divisor de voltaje, y al usar un valor de 2MΩ se obtiene una impedancia de entrada de 1MΩ. Como se ha mencionado con anterioridad, la impedancia de entrada de un circuito debe ser como mínimo diez veces mayor que la impedancia de salida del circuito que lo precede, por lo que en este caso cumple con creces.

Antes de montar el circuito sobre una placa se ha hecho una simulación con el software Multisim, de National Instruments. Se observó que la señal de salida no era alterada al atravesar el circuito, por lo que se validó el funcionamiento teórico.

El montaje del circuito se hizo sobre una placa perforada de fibra de vidrio. Una vez construido el circuito se hicieron pruebas reales:

- **Amplificador de potencia:** El circuito construido debería de funcionar bien en un amplificador diseñado para guitarra. Tras la conexión se confirmó que efectivamente funcionaba como era debido.
- **FPGA:** Dado que funcionaba correctamente en el amplificador, la situación esperada era que también funcionara de igual manera en la FPGA, pero no fue así. El sonido era distorsionado. En un primer lugar se pensó que podría ser que la ganancia del buffer fuera superior a 1, y que hubiera una amplificación por encima de lo que soporta el códec. Por ello se pasó a analizar la señal con un osciloscopio.
- **Osciloscopio:** El resultado esperado hubiera sido el mencionado tras la prueba anterior, pero no fue así. Tal vez en la imagen no se aprecie correctamente, pero hay dos señales, la original y la de salida del buffer. Ambas son representadas al mismo tiempo y hay un solapamiento, es decir, la amplitud de ambas señales coincide. Se confirma que la ganancia es 1.

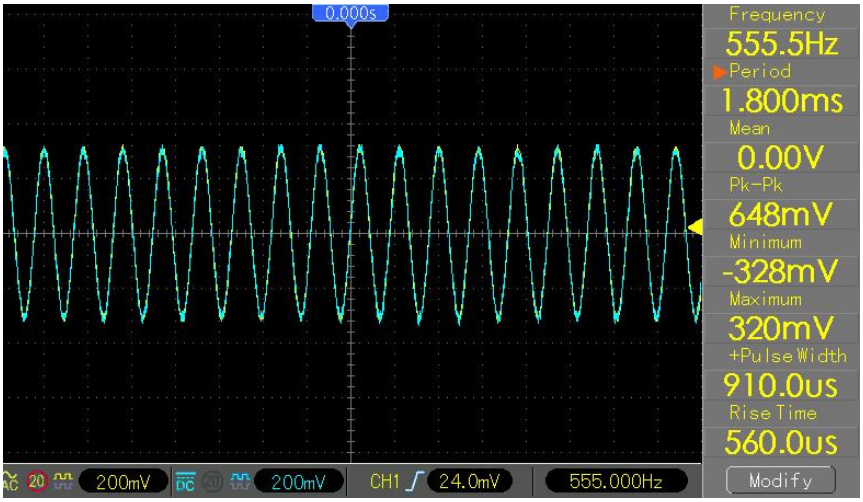


Figura 4.4 Señal original y señal de salida del adaptador de impedancia.

- **PC:** Por último, para confirmar que no hubiera ningún problema en la placa, se hizo la misma prueba en un PC. A través de la línea de entrada de audio se grabó una muestra utilizando el buffer. El resultado obtenido fue el mismo que en la FPGA, una señal con un sonido distorsionado.

La conclusión de este primer circuito es que no cumple para el objetivo que se desea utilizar, aunque sí que es válido para usarlo en equipos analógicos.

4.2.3. Circuito 2

Dado que el anterior circuito no satisfacía las condiciones, se han efectuado algunas modificaciones para mejorarlo y hacer que la distorsión que se presenta en el anterior diseño desaparezca.

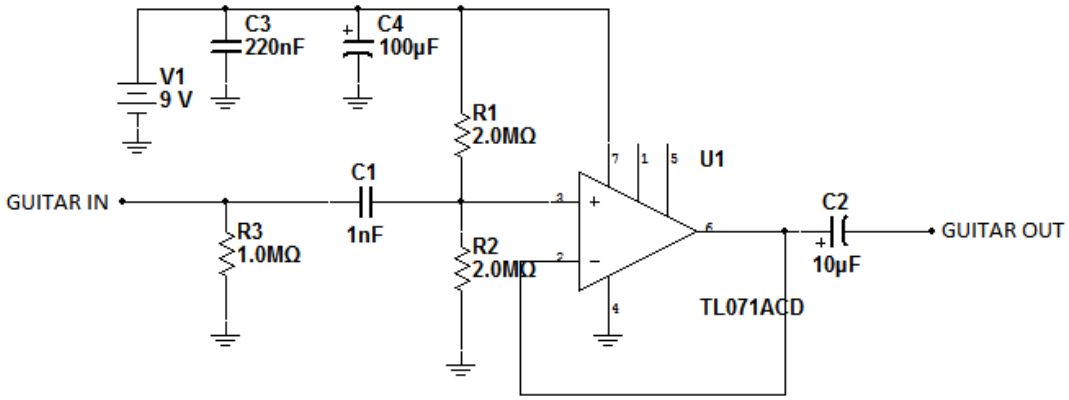


Figura 4.5 Adaptador de impedancia mejorado.

Se han añadido dos condensadores entre la fuente y el divisor de tensión. El primero es de unos 220nF y el segundo es un electrolítico de 100µF. Esta modificación permite filtrar la fuente de alimentación, ya que parte de la corriente alterna que proviene del condensador C1 atraviesa R1 y acaba en la fuente.

También se ha añadido la resistencia R3, situada en paralelo al inicio de la línea de entrada. Esta tiene un valor de 1MΩ y debería adaptar la impedancia y capturar el máximo de la señal originaria de la guitarra.

Se han efectuado las mismas pruebas que antes y, aunque el resultado mejora, sigue habiendo una pequeña distorsión. Esto puede ser debido a que el códec tiene un sistema integrado de reducción de ruido. Si hay una señal que esté por debajo de un umbral de ruido, ésta se recorta.

Por último, se ha probado a utilizar diferentes amplificadores operacionales, y sustituyendo el TL071 por un LM741 se puede decir que el resultado es satisfactorio. Se obtiene un sonido limpio y amplio en matices. A diferencia de antes, ahora la señal tiene un conjunto más amplio de frecuencias, lo que enriquece el sonido, y no hay distorsión.

4.2.4. Construcción del circuito y encapsulado

Para construir el buffer se han obtenido los componentes electrónicos que aparecen en la imagen del circuito. El ensamblado se ha realizado sobre una placa perforada y se ha soldado utilizando estaño. El resultado ha sido el siguiente:

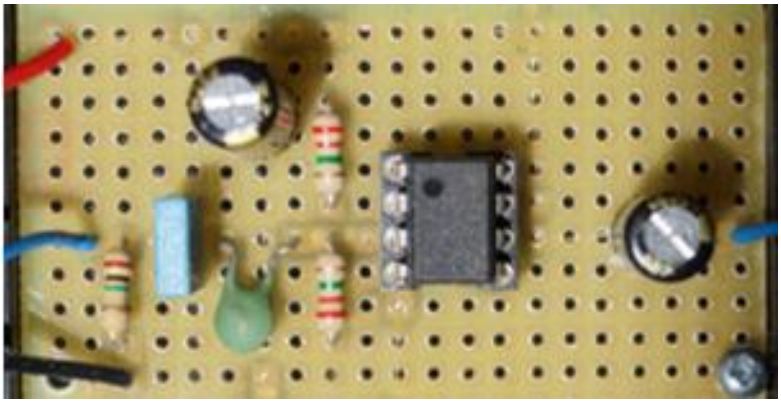


Figura 4.6 Adaptador de impedancia montado sobre placa de puntos.

En la imagen anterior se ven cuatro cables. Dos de ellos son para la alimentación, el cable azul de la izquierda es para la entrada de audio, y, por último, el de la derecha es para la salida de audio. El dispositivo se ha instalado en una caja de plástico, y se ha acomodado de la siguiente manera:

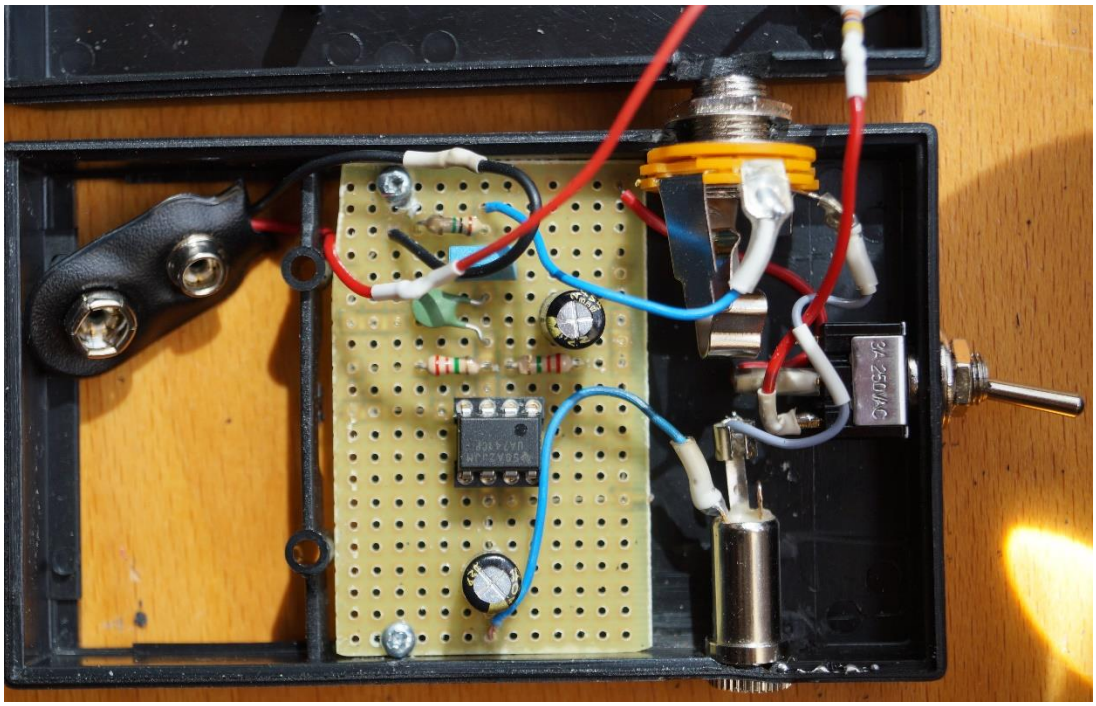


Figura 4.7 Encapsulando el adaptador de impedancia.

La placa de puntos se ha atornillado a la caja para que quede fija y no se mueva al transportarla. Se ha instalado un interruptor para poder encender y apagar el dispositivo. Además, para maximizar la compatibilidad entre la guitarra y la FPGA, se ha utilizado un Jack de audio mono de 6,3mm, para la entrada, y un Jack de 3,5mm para la salida. De este modo no es necesario utilizar adaptadores. La alimentación se realiza a través de una pila de 9V, debido a que el consumo no es elevado, la pila será lo suficientemente duradera como para que no sea necesario el haber añadido un conector DC, que hubiera posibilitado utilizar un conversor externo de 9V con una intensidad de 300mA.

Figura 4.8 Resultado final del adaptador de impedancia.

El resultado final de este dispositivo es el que se observa en la imagen superior. Para saber cuándo está encendido o apagado se le ha incorporado un diodo led. Por razones de estética se ha hecho un pequeño diseño descriptivo.

5

Implementación

En este capítulo se recogerá toda la información relativa a la implementación del proyecto. Se analizarán los diseños planteados a través de diagramas y esquemas, y se explicarán los efectos que se han modelado.

5.1. Introducción: Primeros Pasos

Este es un proyecto en el que intervienen diversos componentes de la placa. Se trabaja con un códec de audio, un conversor analógico-digital y, posteriormente, se verá que también se trabaja con el procesador Nios, para diseñar otro tipo de efectos. En definitiva, para la comunicación y el control son necesarios varios módulos, de los cuales, algunos de ellos, funcionarán a modo de controlador o driver.

Para este proyecto han sido proporcionados de antemano cuatro módulos. Por un lado, están los tres que se encargan de la configuración del códec y de la recepción y envío de muestras de audio (AU_SETUP, AU_IN y AU_OUT), y por último el que se encarga de configurar el conversor analógico digital y hacer llegar la información (ADC).

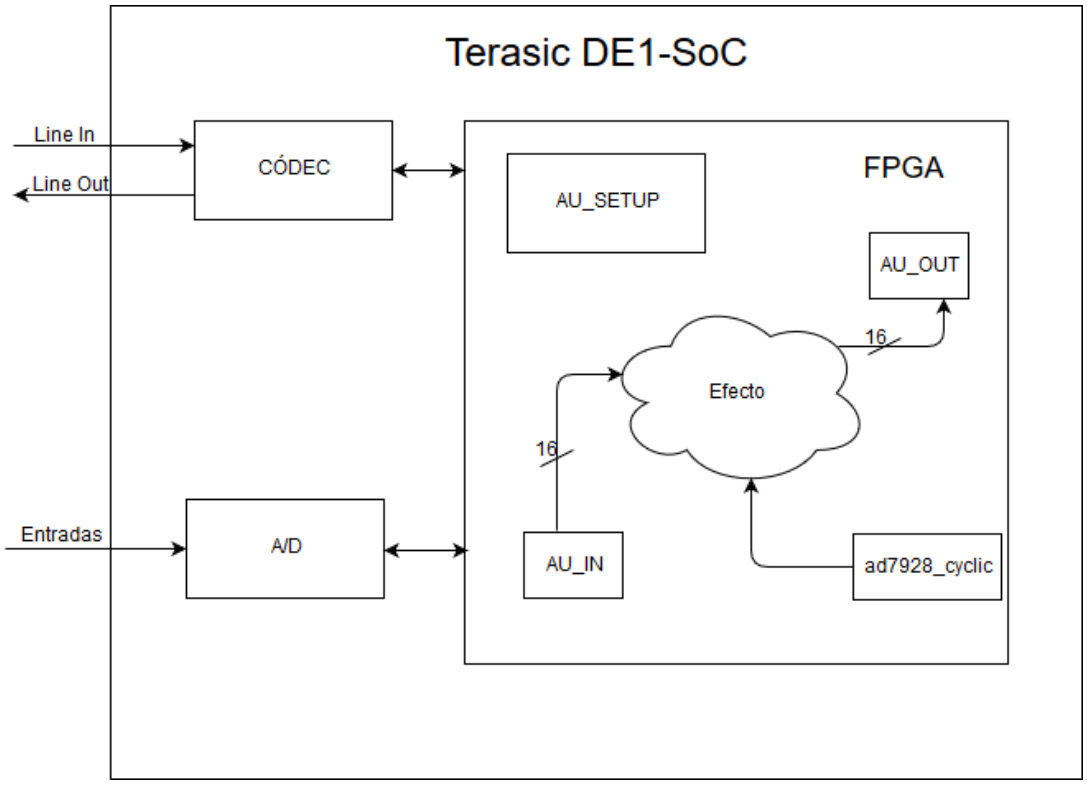


Figura 5.1 Esquema de la comunicación entre componentes y módulos.

La operación más simple es un “bypass”, es decir, que el audio de salida sea el mismo que el audio de entrada, en este caso no es necesario el CAD de señales, pero sí el códec de audio. Para poder realizar dicho proceso, es necesario un programa principal que instancie los módulos AU_SETUP, AU_IN y AU_OUT.

La instanciación del primer módulo configurará el códec para que empiece a funcionar con los valores deseados (posteriormente se explicarán las posibilidades que ofrece), por otro lado, está el módulo AU_IN que se encarga de coger una muestra de audio (en caso de haber un efecto, esta muestra sería alterada), y por último está el módulo AU_OUT dónde se escribe la muestra para retornarla (en este caso es la misma, pero podría ser modificada).

5.1.1. Configuración del Códec (AU_SETUP)

Este módulo se encarga de la inicialización del códec de audio tras la activación de la señal de "reset". Suministra una señal de reloj (XCK) de la que dependerá la frecuencia de muestreo utilizada. La frecuencia de esta señal se obtiene de la división de la frecuencia del reloj del sistema (50Mhz) entre 4, lo que proporciona un reloj para el códec de 12,5MHz.

Para la configuración del códec se ha decidido utilizar una codificación de 16 bits en complemento a dos, una transmisión de datos serie, con señales de sincronización, y una frecuencia de muestreo de 48kHz.

La frecuencia de muestreo se puede cambiar mediante el parámetro "SAMPLE_RATE", en la siguiente tabla se muestran las diferentes posibilidades.

Frecuencia deseada	Parámetro (SAMPLE_RATE)	Frecuencia obtenida
8kHz	1	XCK/1536=8138.0208
32 kHz	2	XCK/384=32552.0833
48 kHz	3	XCK/256=48828.125

Tabla 5.1 Posibles frecuencias de muestreo disponibles.

El esquema del módulo AU_SETUP se puede ver en la siguiente imagen:

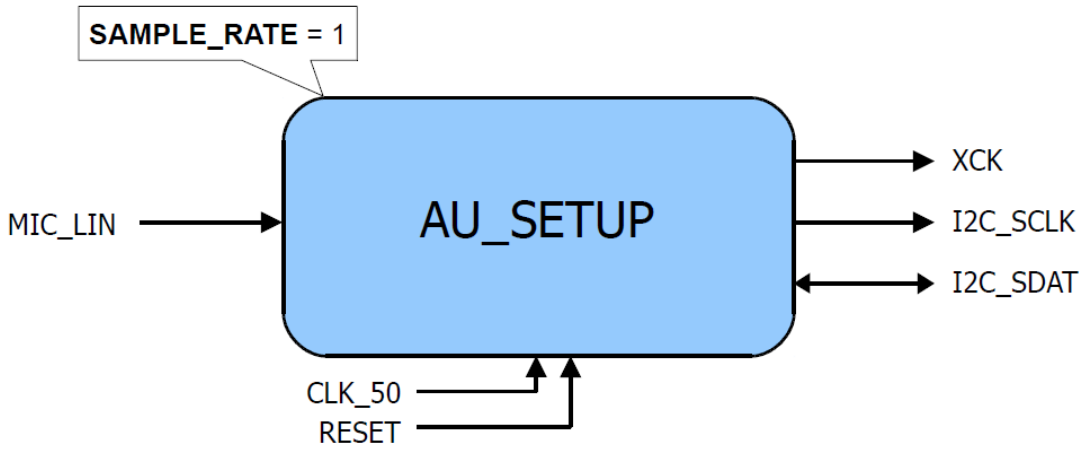


Figura 5.2 Esquema del módulo AU_SETUP.

Mediante el parámetro SAMPLE_RATE se selecciona la frecuencia y mediante el parámetro MIC_LIN se selecciona la entrada de audio (0: Entrada Line In, 1: Entrada de micrófono). Al otro lado hay dos señales de salida, que se corresponden a las señales de reloj para el códec y para el bus I2C, y una señal bidireccional que se corresponde con la línea de datos del bus I2C.

5.1.2. Lectura de muestras desde el Códec (AU_IN)

El módulo AU_IN es el encargado de la recepción de muestras de audio desde el códec. La recepción de datos se realiza a través de una línea serie (ADCDAT). Se alternan muestras del canal izquierdo y del derecho, pero en este caso, al estar trabajando con sonido mono, basta con leer únicamente uno de ellos, ya que los dos canales son iguales. Existen señales de control, generadas por el códec, que indican cuando es el momento de leer los bits y a qué canal se corresponden (BCLK y ADCLRC).

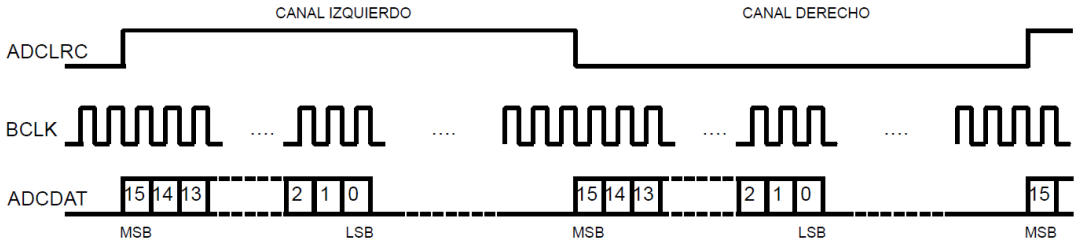


Figura 5.3 Recepción de datos del códec.

La señal ADCLRC indica si se trata de una muestra del canal izquierdo o derecho. Por ejemplo, basado en la figura anterior, para obtener una muestra del canal izquierdo habrá que esperar al flanco de subida de ADCLRC, y recoger la muestra en los 16 flancos de subida siguientes de la señal BCLK, que indica cuando hay que leer el bit en la señal ADCDAT.

En la siguiente imagen se presenta el módulo AU_IN, con sus entradas, salidas y señales de control:

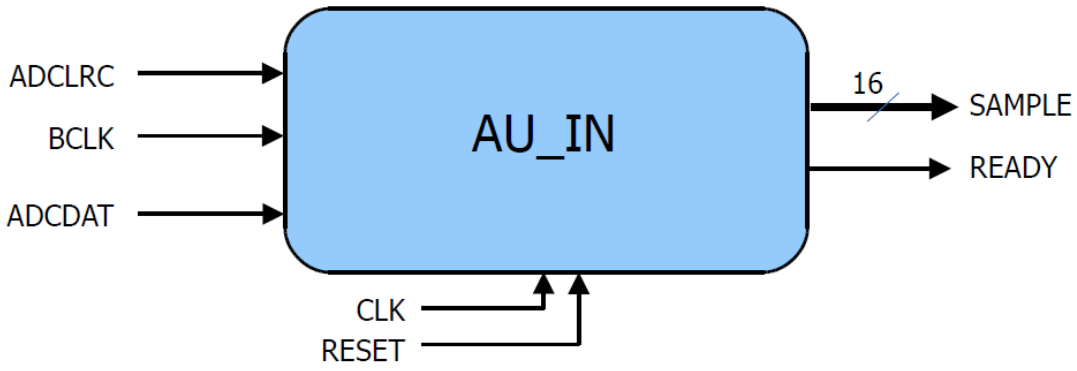


Figura 5.4 Esquema del módulo AU_IN.

Como se ha comentado anteriormente, las entradas son ADCLRC (indica a qué canal se corresponde la muestra recibida), BCLK (indica cuando leer los bits de la línea de serie) y ADCDAT (línea de serie de entrada). Las salidas de este módulo son SAMPLE (es la muestra recibida de 16 bits en complemento a 2) y la señal READY (que se activa durante un ciclo de reloj al terminar de recibir una muestra, es decir, indica cuando una nueva muestra está disponible).

5.1.3. Escritura de muestras desde el Códec (AU_OUT)

El envío de muestras de audio al códec se efectúa a través de una línea serie, y es el módulo AU_OUT el encargado de ello. La escritura de muestras se facilita con señales de sincronización generadas por el códec, que son BCLK y DACLRC. Al igual que en el módulo anterior, estas señales tienen el mismo cometido, indicar cuando leer un bit y confirmar a qué canal pertenece dicha muestra. Hay que generar la señal DACDAT manteniendo cada bit entre dos flancos de bajada, para que sea estable en los flancos de subida, que es cuando lo lee el códec. Anteriormente se ha mencionado que se trabaja con sonido mono, aun así, la muestra hay que replicarla para los dos canales, ya que, de lo contrario, al ser la salida del códec en estéreo, de utilizar dos altavoces sólo se oiría a través de uno de ellos.

En la siguiente imagen se presenta el módulo AU_OUT, con sus entradas, salidas y señales de control:

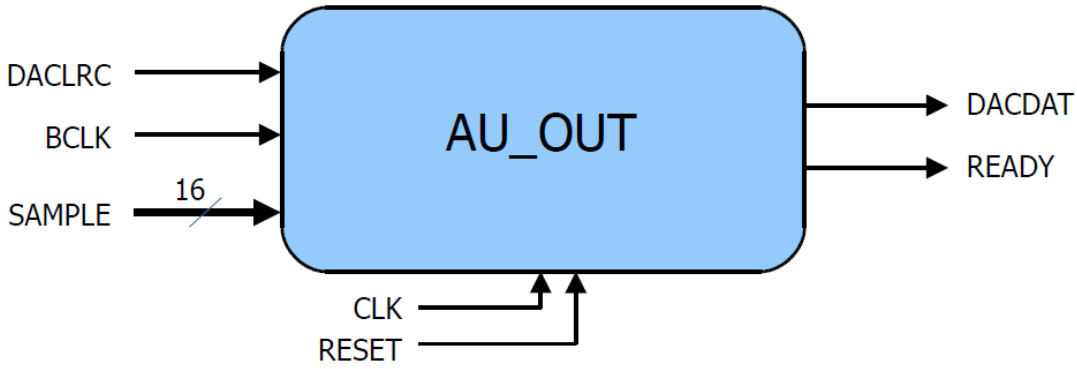


Figura 5.5 Esquema del módulo AU_OUT.

Entre las entradas están las señales DACLRC y BCLK, que se han explicado con anterioridad, y la señal SAMPLE (la muestra en complemento a 2 que debe ser enviada). En la salida está la señal DACDAT (línea de serie de salida) y la seña de control READY (que se activa durante un ciclo de reloj después de leer la muestra que va a ser enviada. Ésta indica que la muestra enviada ya se ha recogido y se puede cambiar el valor en la entrada SAMPLE para proceder con el envío de la siguiente muestra).

5.2. Entradas analógicas de control

Se desea que los efectos no sean fijos, si no que el músico tenga la posibilidad de ajustar ciertos valores para que cumpla con sus gustos o necesidades. Es por ello que se necesitan de unas entradas externas que permitan controlar los efectos. A continuación, se explica cómo se hace, para ello se utiliza el CAD que tiene la placa Terasic DE1-SoC.

5.2.1. CAD y el cabezal 2x5

En la placa se encuentra un cabezal de 2x5 pines que se corresponde con las entradas analógicas y su correspondiente alimentación. En la siguiente imagen se puede ver cómo es el conexionado:

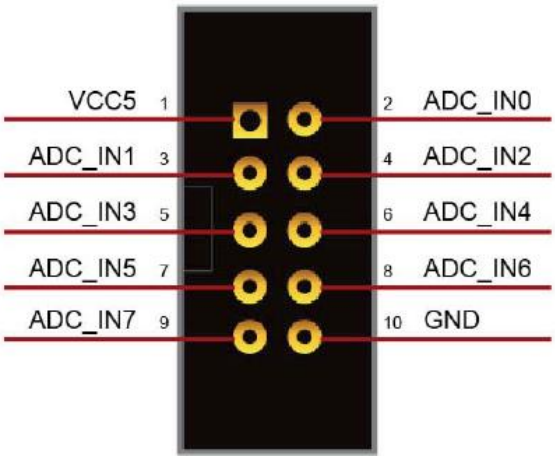


Figura 5.6 Patillaje del cabezal 2x5 correspondiente al CAD.

En este cabezal se pueden conectar hasta un máximo de 8 entradas analógicas, para el proyecto actual, eso es más que suficiente. En la siguiente imagen se observa cómo se comunica el CAD con los elementos externos conectados a los pines, y cómo se comunica con la FPGA, la cual tiene el módulo ad7928_cyclic, que se encarga de la configuración y de la recepción de las señales.

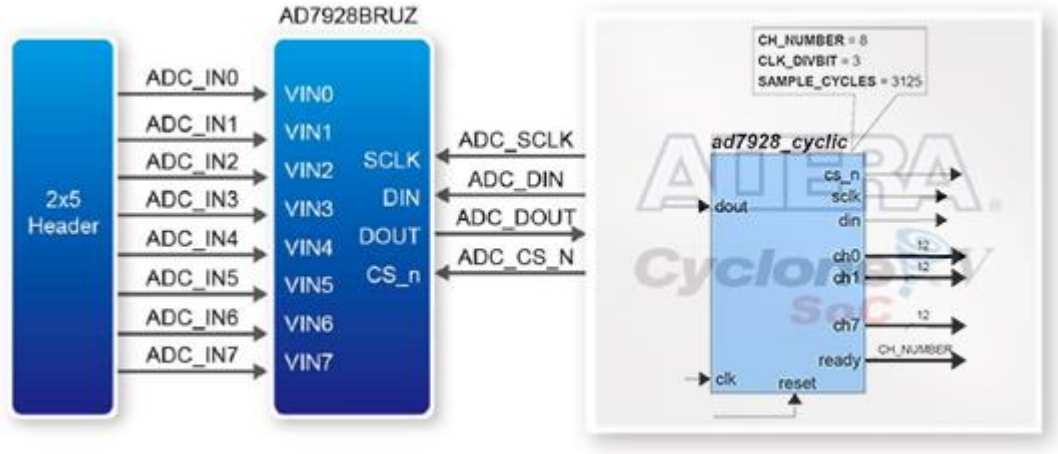


Figura 5.7 Conexiones entre la FPGA, el cabezal y el CAD.

La imagen superior representa, por un lado, las entradas analógicas del cabezal, y por otro lado se observa el CAD intercambiando señales con el módulo ad7928_cyclic, que actúa como controlador. Las entradas son las siguientes, ADC_SCLK (le llega una señal de reloj), ADC_DIN (entrada de datos digital), ADC_CS_N (señal de control para la selección de canal).

La señal de salida es ADC_OUT (salida de datos digital). En la siguiente imagen se observa a mayor escala el módulo que actúa como controlador, presente en la FPGA.

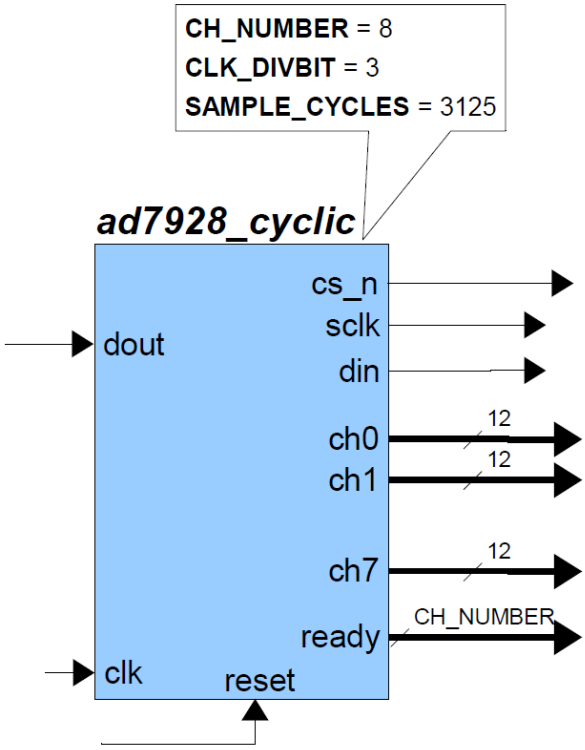


Figura 5.8 Esquema del módulo ad7928_cyclic.

Los parámetros para configurar el CAD son CH_NUMBER (número de canales que se van a utilizar), CLK_DIVBIT (el factor por el que se divide la frecuencia de CLK, al generar SCLK. Es un valor entre 1 y 11) y SAMPLE_CYCLES (Periodo de muestreo, en ciclos de SCLK).

En la entrada está la señal dout, (la línea de entrada), y en la salida están las señales cs_n (el flanco de bajada que indica el comienzo de una captura), sclk (reloj para el CAD), din (línea serie de salida), ch*N* (último dato capturado en el canal *N*) y ready (indica que se ha terminado de capturar el número de canales seleccionados por el parámetro CH_NUMBER).

5.2.2. Uso y conexionado del cabezal 2x5

Se ha configurado el CAD para que trabaje con un voltaje de entre 0V y 5V. Los elementos que se han conectado a los pines del cabezal han sido cuatro resistencias variables o potenciómetros con un valor de 47kΩ. Debido a que en algunos efectos se esperan diferentes comportamientos, se han utilizado dos potenciómetros lineales y otros dos logarítmicos. En esta imagen se muestra la diferencia entre ambos:

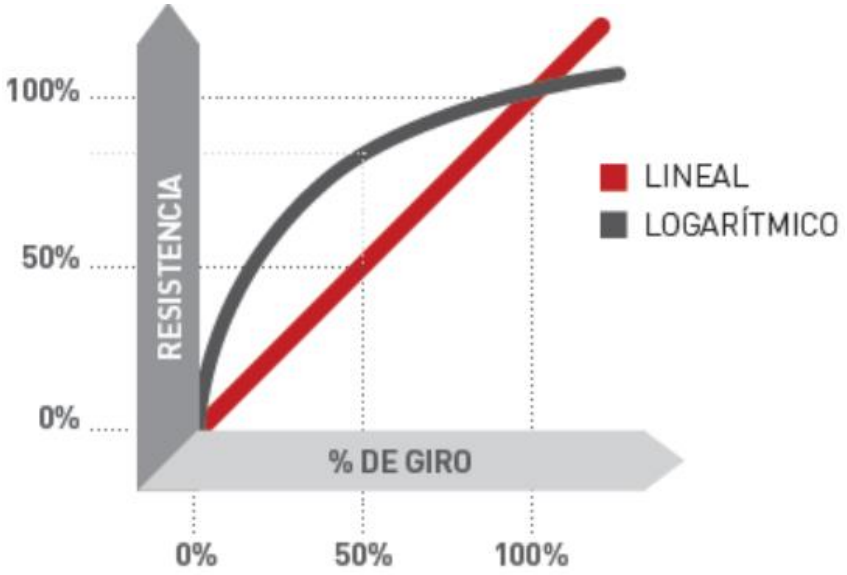


Figura 5.9 Diferencia entre la curva de un potenciómetro lineal y otro logarítmico.

El esquema que se ha seguido para la conexión de los potenciómetros es el siguiente:

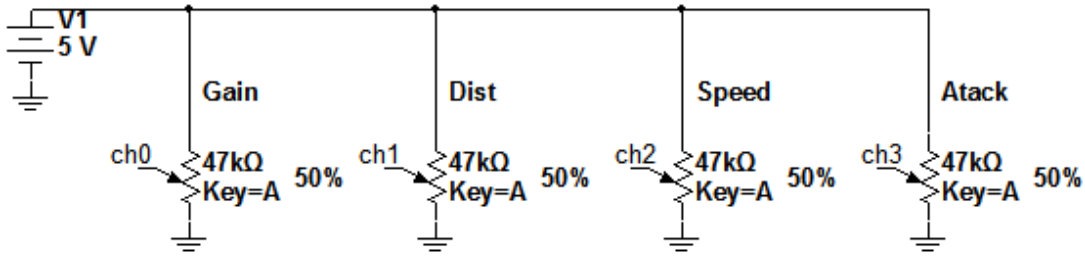


Figura 5.10 Esquema de conexionado de los potenciómetros al CAD.

El propio CAD ofrece una fuente de tensión de 5, es la que se usa en el circuito. Los potenciómetros están colocados en paralelo para que todos tengan la misma tensión. El dato del voltaje que el potenciómetro deja pasar es el que se envía al CAD, que lo convierte en un dato binario de 12 bits.

En una primera instancia, el conexionado para realizar las pruebas se hizo sobre una placa de prototipado, sin embargo, se ha desarrollado una versión mejorada sobre una placa de puntos.

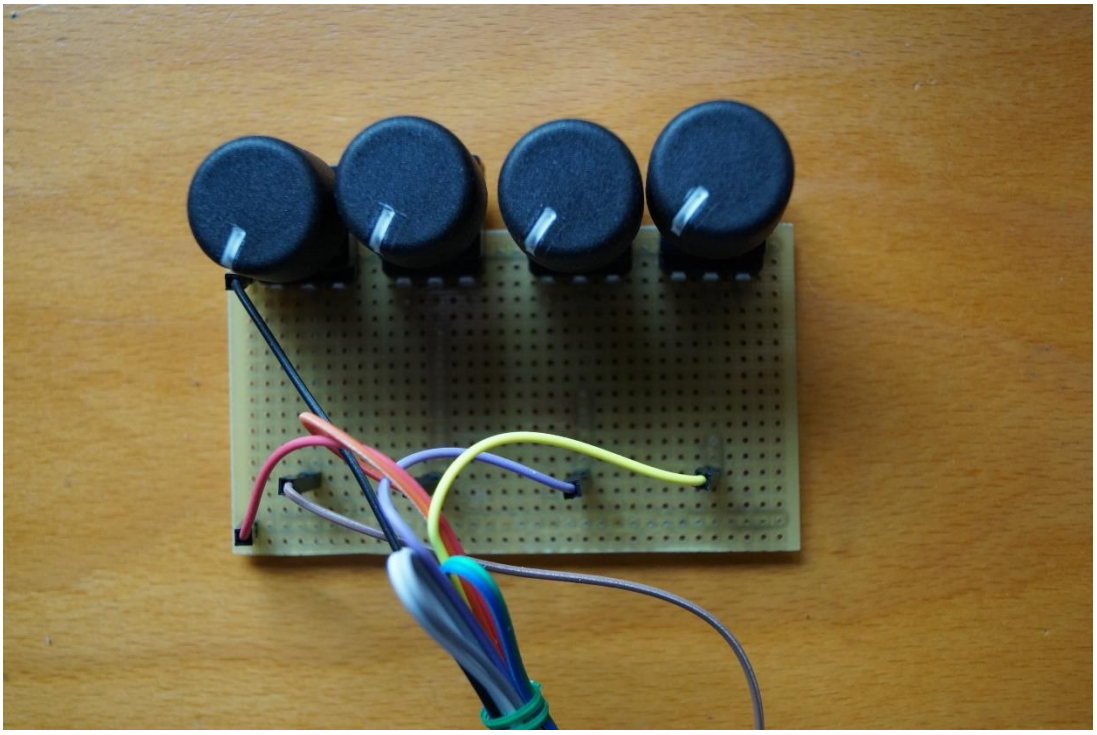


Figura 5.11 Prototipo del controlador externo del pedal.

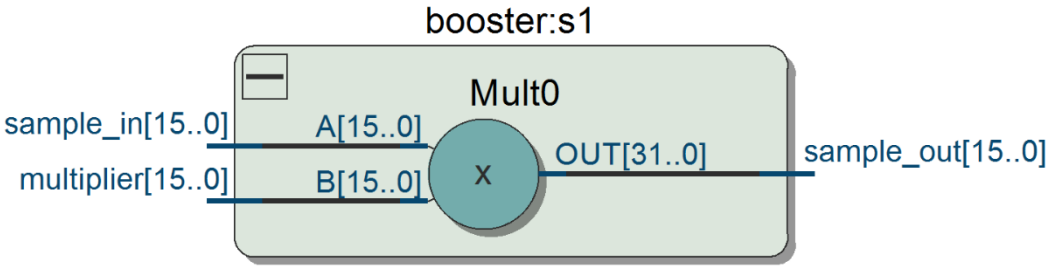
Los controles permiten ajustar el sonido de una forma fácil y cómoda. Además, el cable utilizado está totalmente adaptado para que se puede conectar al patillaje fácilmente y sin peligro de dañar los pines, cosa que podría suceder en caso de conectar los cables de forma individual.

5.3. Implementación de Efectos I (Hardware)

Para este proyecto se ha querido replicar de forma digital algunos efectos de guitarra. Para ello se han definido de forma individual, en forma de componentes. Al igual que en una pedalera convencional, se podrán instanciar en cualquier momento.

5.3.1. Booster

Se trata de una amplificación lineal limpia, es decir, permite incrementar el volumen respetando el carácter y el tono de la guitarra, sin coloración de la señal. Se utiliza por los músicos en momentos puntuales para tener un sonido por encima del resto de miembros de



la banda, por ejemplo, para que la guitarra destaque en un solo.

Figura 5.12 Diseño del módulo “booster”.

A este bloque le llega una señal que es “sample_in”. Se trata de la muestra en binario que ha capturado el Códec de audio. Por otro lado, está la señal “multiplier”, es el valor que se obtiene a través de una resistencia variable y que se digitaliza mediante un Conversor Analógico-Digital (CAD).

Con las dos señales de entrada se efectúa una operación de multiplicación, obteniendo una señal equivalente, pero con una mayor amplitud. La multiplicación de dos vectores de 16 bits da como resultado una salida de 32 bits. Dentro del módulo se adapta para conseguir que la señal sea de 16 bits, para ello se descartan los 7 bits menos significativos y los 9 más significativos para controlar que el volumen de la salida no sea excesivo.

El nuevo valor que ha sido obtenido va hacia la salida “sample_out”. Puede ir directamente hacia el códec de audio para ser reproducida por unos altavoces o puede ser la entrada de otro módulo que aplique un efecto diferente. Es decir, los efectos pueden ser concatenados.

5.3.1.1. Booster: Resultados

El resultado ha sido el esperado. Se ha conseguido una amplificación limpia, y el control del volumen es el adecuado para que no sea excesivo ni sobrepase los límites del códec, lo cual ocasionaría distorsión.

En las siguientes imágenes obtenidas a través de un osciloscopio se observan las diferencias entre una onda sinusoidal natural y la forma que obtiene después de aplicar el efecto.

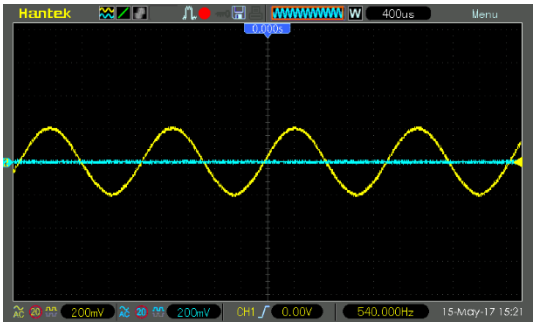


Figura 5.13 Señal atenuada.

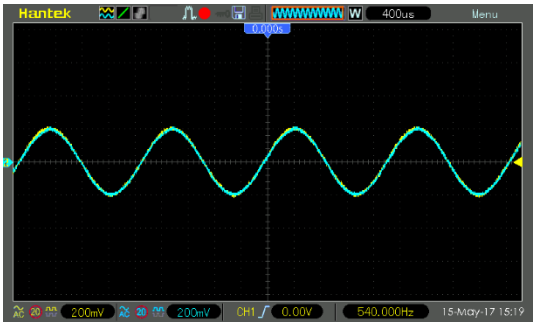


Figura 5.14 Señal de igual ganancia.

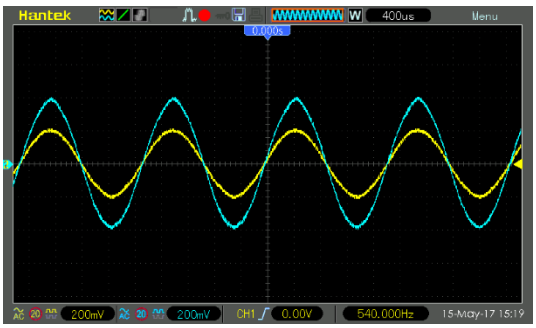


Figura 5.15 Señal amplificada.

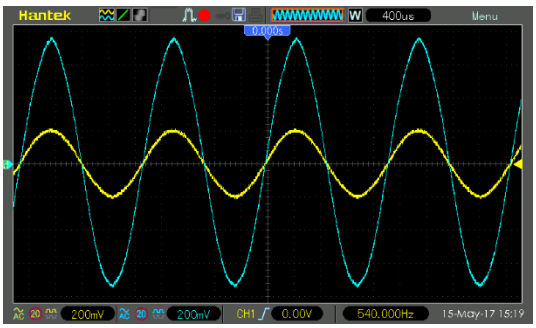


Figura 5.16 Señal de amplitud máxima.

En las imágenes anteriores se observan algunas de las variaciones en la amplitud que se pueden obtener con este efecto. El abanico de posibilidades va desde la atenuación completa de la señal hasta su amplitud máxima.

En las figuras también se observa que no hay desfase ni deformaciones en la señal, y en las pruebas sonoras se comprueba también que el tono del instrumento no varía, sólo cambia su volumen. Así que con estas pruebas se confirma que se ha diseñado un amplificador lineal y que el sonido se mantiene limpio, sin ningún tipo de coloración ni distorsión.

5.3.2. Distorsión/Fuzz

La distorsión diseñada ha sido de tipo “hard-clipping” o de “recorte duro”. El recorte es una forma de distorsión que limita la señal una vez excedido un umbral. En este caso, al ser dura, implica que está estrictamente limitada, es decir, que llegado a ese límite se produce un

corte plano. Si fuera un recorte suave, significaría que la señal continúa haciendo una forma similar a la original, pero con menos ganancia, lo que en el mundo de la música se le conoce como efecto de “overdrive” o saturación. En la siguiente figura se observan las diferencias:

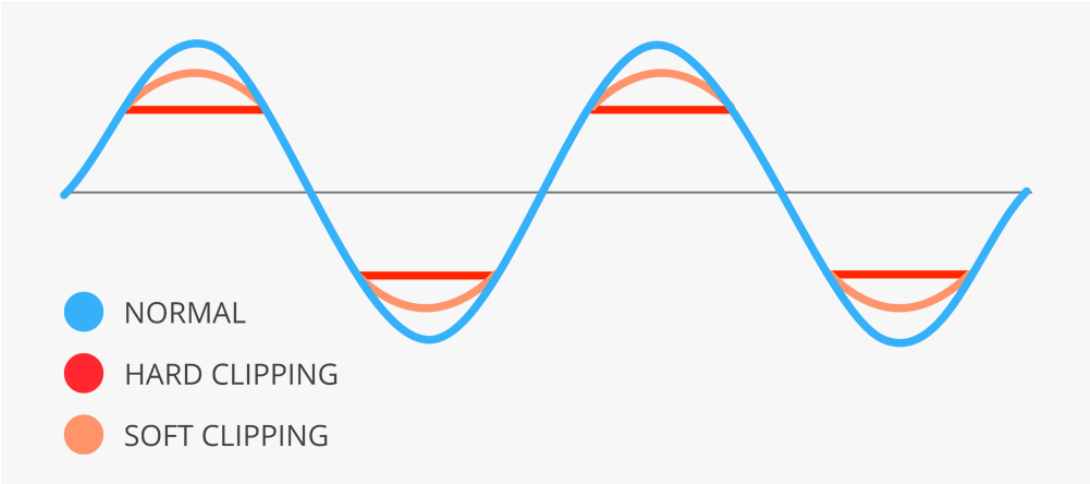


Figura 5.17 Dos tipos de distorsión.

En el dominio de la frecuencia se producen fuertes armónicos en el rango de alta frecuencia, sobre todo cuando la forma de la onda se aproxima a una onda cuadrada.

Al igual que otra distorsión analógica del mismo tipo, este efecto cuenta con dos controles, uno para el volumen y otro para la distorsión. Este último establece el umbral a partir del cual se corta la onda original. El resultado que se obtiene es una distorsión que va desde un estilo agresivo y primitivo, hasta una sutil distorsión que es casi limpia. La onda generada se asemeja a la que produce un pedal de distorsión de tipo “Fuzz”.

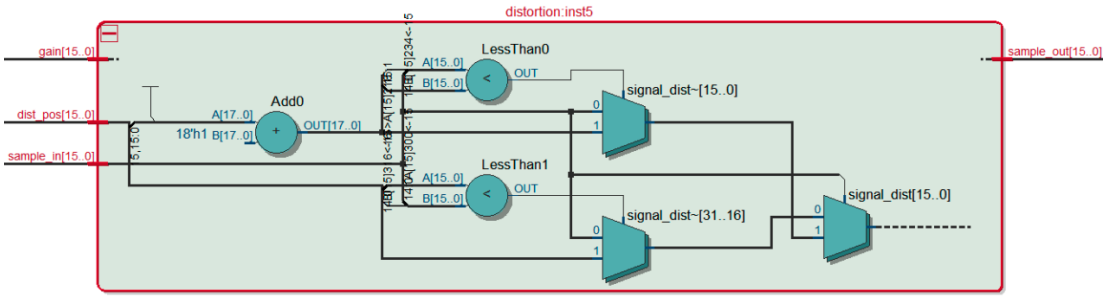


Figura 5.18 Diseño del módulo “distortion”.

5.3.2.1. Distorsión: Resultados

Aunque en una primera instancia el efecto pueda parecer ruidoso, éste se usa a menudo en algunos estilos musicales como el rock ácido o el métal. Es característico por su tono crudo y sin comprimir.

En la siguiente figura se observa cómo se produce el recorte. Éste es el caso básico, en el que no se juega con una posterior amplificación de la señal. Se puede ver como la señal

original es sinusoidal, y como la señal modificada es recortada cuando el valor de la señal supera el umbral máximo, controlado a través de un potenciómetro.

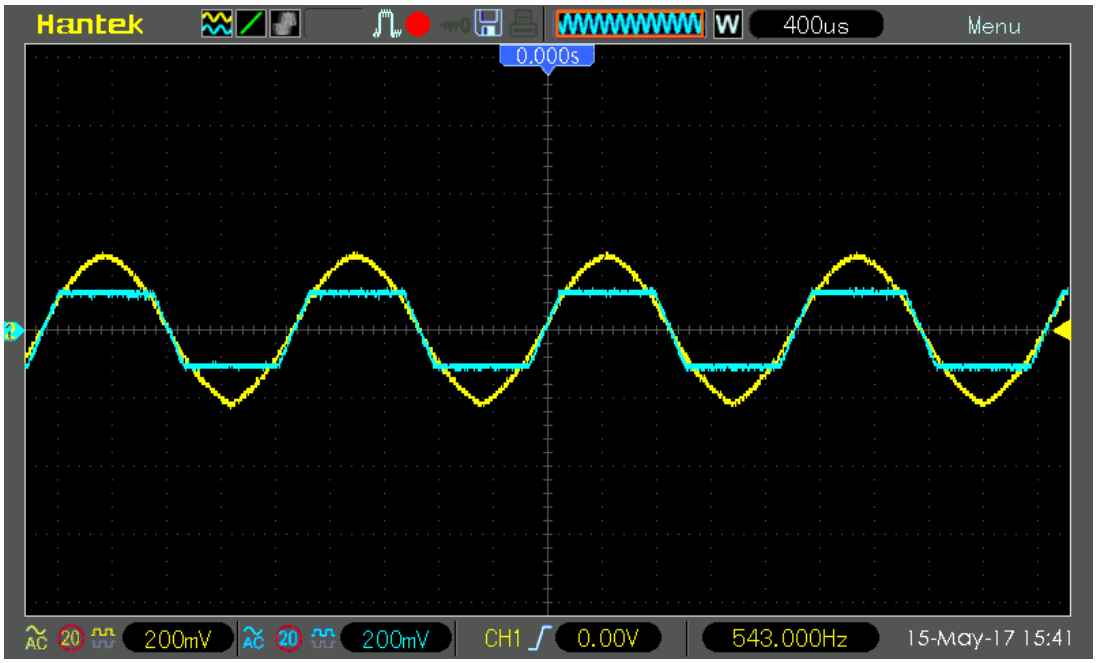


Figura 5.19 Señal sinusoidal original y señal modificada.

Las ondas sinusoidales son modificadas y según los ajustes pueden llegar a tener el aspecto de una onda cuadrada. Este efecto se caracteriza por su versatilidad, como se puede comprobar en las siguientes figuras.

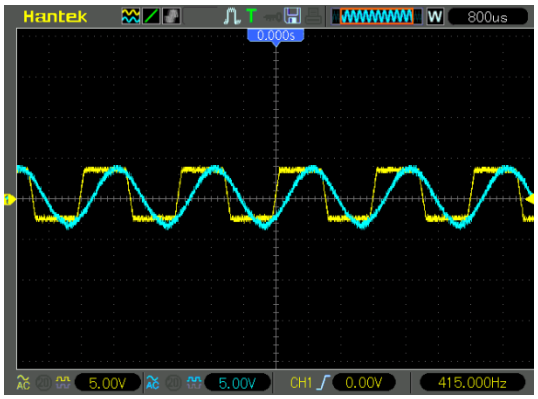


Figura 5.20 Misma amplitud, forma cuadrada.

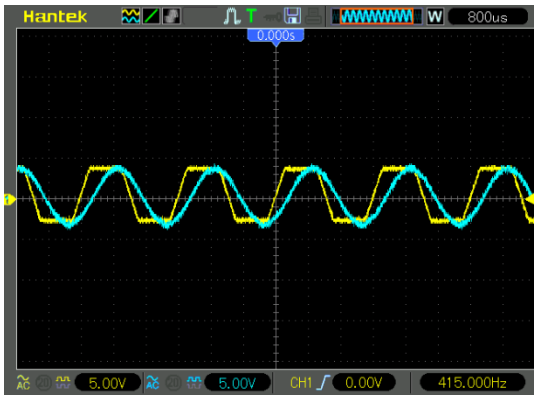


Figura 5.21 Misma amplitud, forma trapezoidal.

Lo que se observa en estas imágenes es que jugando con los potenciómetros de volumen y distorsión se pueden conseguir señales de la misma amplitud, pero con diferente forma de onda.

En la figura de la izquierda se observa que los flancos de subida y de bajada tienen una mayor verticalidad, es decir, pasan de un estado a otro de manera más rápida. Esto es porque el umbral de distorsión se ha situado en una posición muy baja, pero luego ha sido amplificado por la etapa de ganancia (booster). Por el contrario, en la figura de la derecha, la transición de un estado a su opuesto se hace de una forma más lineal y menos brusca, la

señal es más triangular, por lo que la distorsión es menos notoria pero la amplitud es idéntica.

En esta otra imagen se aprecia como aún se puede ajustar más la señal resultante a la original, teniendo el recorte la misma forma plana:

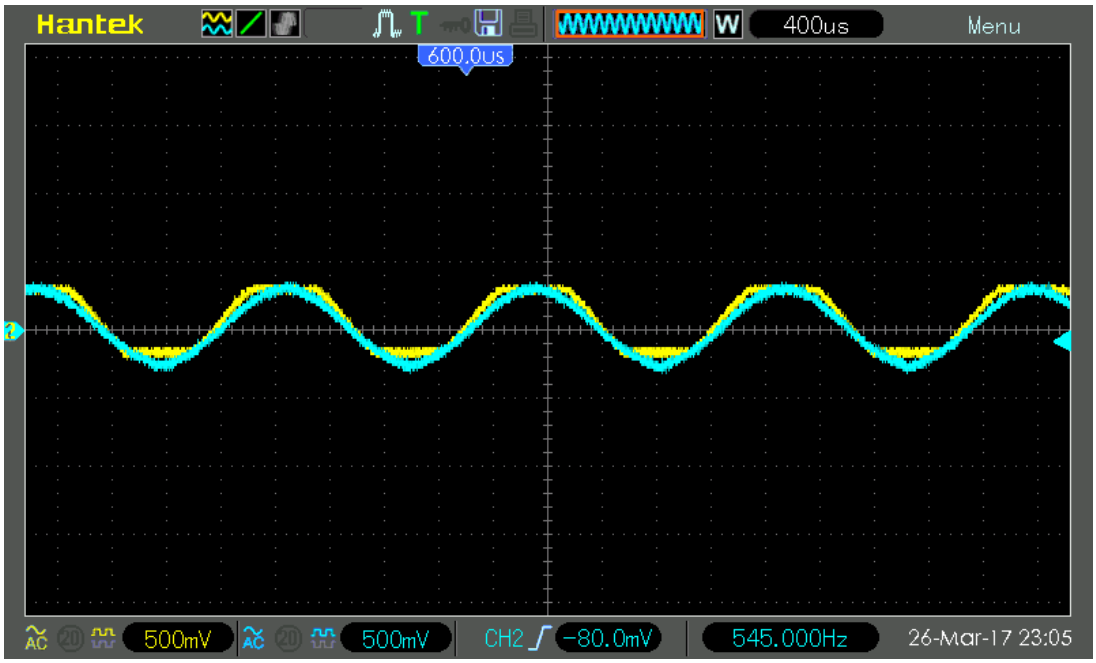


Figura 5.22 Otra posible configuración del efecto.

Ajustando también los controles de tono y volumen de la guitarra, el abanico de posibilidades se abre aún más.

5.3.3. Saturación (Overdrive)

En el anterior apartado se ha explicado el efecto de distorsión basado en “hard-clipping”. Este segundo efecto aplica una distorsión de tipo “soft-clipping”, que es más suave que la anterior, debido a que la señal no se recorta drásticamente, si no que se limita su amplitud una vez alcanzado el umbral máximo, manteniendo una forma similar a la original en vez de cuadrada.

El efecto de “overdrive” trata de replicar la forma que adquiere una onda en un amplificador de válvulas cuando la etapa de ganancia se sobrecarga de tal modo, que el nivel de la señal excede su nivel de salida. En ese caso la señal de salida adquiere una forma diferente, las ondas de mayor amplitud son contenidas, es decir, son más bajas de lo que debieran ser idealmente. Este cambio en la forma de la onda provoca también un cambio en el sonido, resultando en una distorsión suave, de tonos cálidos a volúmenes bajos y llegando a una distorsión más severa a medida que aumenta la ganancia.

Respecto a la distorsión, este efecto de saturación provoca un menor cambio en la tonalidad y genera menos armónicos. El “overdrive” es preferible si se desea obtener un sonido más limpio

Para la implementación de la saturación se ha seguido un patrón muy similar al de la distorsión. Se ha establecido el mismo corte, pero con un pequeño matiz. Cuando la señal está por encima del umbral máximo se calcula la diferencia entre el umbral y la posición de la señal, posteriormente se divide esa diferencia entre un valor predeterminado y se le suma al umbral máximo previamente definido. Lo que se consigue es que la forma sinusoidal de la onda se mantenga, pero que su amplitud esté controlada.

5.3.3.1. Overdrive: Resultados

Se trata de una distorsión suave, con tonos cálidos que varían en función del recorte que se produce y de la ganancia. A diferencia del anterior efecto, éste es más polivalente, se usa prácticamente en todos los géneros, desde el blues o el rockabilly a otros más pesados como el hard-rock o el heavy metal.

Con este efecto se escucha como se obtiene un sonido "comprimido", ya que hay un umbral máximo de amplitud que se está limitando, y también se percibe algo de "sustain". Si se pone un osciloscopio para ver la señal resultante se obtiene el siguiente resultado:

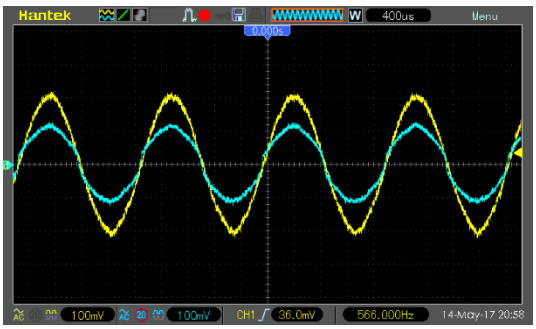


Figura 5.23 Señal atenuada con "soft-clipping".

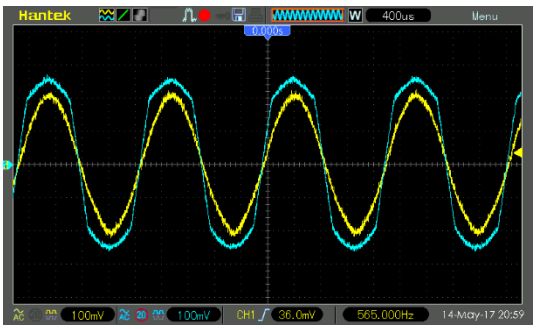


Figura 5.24 Señal amplificada con "soft-clipping".

Se observa como la onda es recortada en sus extremos, pero, sin embargo, se conserva una forma similar a la original. En el caso de la distorsión del efecto anterior, ésta era dura, se recortaba totalmente de una forma plana, perdiendo así su forma sinusoidal y asemejándose más a una onda cuadrada.

5.3.4. Trémolo

Trémolo es un adjetivo de origen italiano que significa tembloroso. En música, esta palabra describe la variación periódica en la amplitud de una onda acústica, manteniendo la frecuencia constante. A veces se confunde trémolo con vibrato, este último sí que afecta a la frecuencia.

Los trémolos más sencillos a nivel comercial suelen tener dos parámetros para ajustar el sonido, que son la velocidad de repetición (speed o rate) y la profundidad (depth), es decir, la diferencia entre la mayor y la menor amplitud. Otros más completos permiten modificar el tipo de onda.

Para esta implementación se ha modelado un efecto digital que consta de tres parámetros que el usuario puede controlar.

- **Rate:** Determina la velocidad del Oscilador de Baja Frecuencia (LFO). Se sitúa entre 10Hz y 1,5Hz.
- **Depth:** Delimita la diferencia que existe entre el pico de mayor amplitud y el mínimo.
- **Wave:** Especifica la forma de la onda. Se han establecido tres diferentes. Cuadrada, diente de sierra y triangular.

Los dos primeros controles se modifican mediante dos potenciómetros independientes. El tipo de onda se ajusta mediante el uso de los interruptores presentes en la FPGA. Para el correcto funcionamiento se han implementado dos módulos, el primero se llama "tremolo" y es el encargado de recibir las señales del exterior. Dentro de este módulo también se ha diseñado un LFO, que controla la frecuencia con la que se hace la variación de la amplitud. Este componente efectúa un intercambio de información con el segundo módulo, que se llama "wave_setup". Este último se encarga de, a través de los parámetros que recibe, variar la amplitud de la onda en función del tiempo. Existen tres casos diferentes, dependiendo de la forma de la onda que se desee generar como salida:

- Onda cuadrada: Es la más sencilla de todas. Sólo hay dos amplitudes posibles, la máxima y la mínima, el cambio que se hace al pasar de una a otra es brusco. Para ello se tiene en cuenta el valor del "rate" y el número de muestras que han sido recibidas. Durante la mitad del ciclo, la amplitud de la señal está atenuada, y durante la otra mitad es amplificada.
- Onda de diente de sierra: Se emplea un divisor y un contador para calcular cada cuántos ciclos hay que incrementar o disminuir el valor de la amplitud. De este modo la onda generada varía su amplitud de forma lineal y constante, y cuando se completa el ciclo vuelve a su posición inicial. Se han creado dos tipos diferentes, una en la que dado un valor inicial la amplitud es creciente y otro en el que es decreciente.
- Onda triangular: Al igual que en la anterior forma, se emplea un divisor y un contador. En este caso, no hay ningún corte brusco durante el ciclo, si no que la amplitud de la onda se incrementa hasta llegar a un punto máximo, y a partir de ahí empieza a disminuir hasta llegar a su mínimo. Todo ello se hace de forma gradual durante el tiempo.

El módulo "wave_setup" tiene como salida un valor binario, que es un multiplicador que será usado por el módulo "booster", que ya ha sido descrito con anterioridad.

5.3.4.1. Trémolo: Resultados

Este es un efecto que tiene muy buen comportamiento. Respecto a sus homólogos analógicos se podría decir que es muy completo y que no le falta casi nada.

El valor añadido que posee este trémolo digital es se puede variar la forma de la onda que se genera. Es una función que en el mercado de los efectos analógicos sólo la suelen tener los trémolos de gama superior.

El efecto de esta implementación se puede decir que es sobresaliente. Tiene muchas funciones, proporciona al músico un amplio abanico de configuraciones y además, suena muy bien.

Se ha puesto a prueba el diseño con un generador de señales y se recogen los resultados con la ayuda de un osciloscopio. En las siguientes imágenes se presentan las pruebas realizadas y su comportamiento.

Caso 1: Onda cuadrada

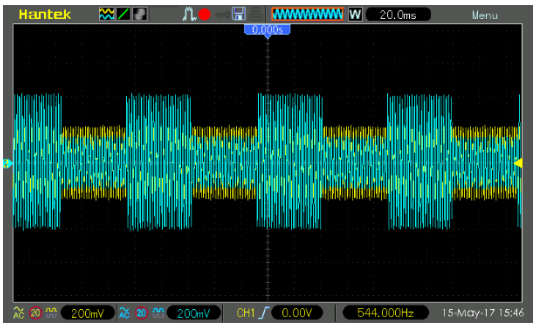


Figura 5.25 Alternancia de amplitudes.

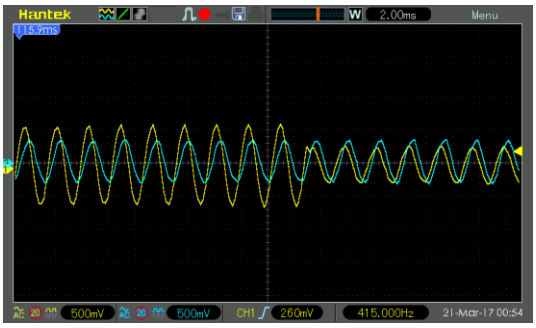


Figura 5.26 Momento de transición de amplitud.

En estas imágenes se observa la transición de un estado, en el que la señal está amplificada, a un segundo estado en el que la señal se mantiene al natural. Se observa como en los dos estados la amplitud es constante hasta que se produce el cambio.

El resultado es el esperado, hay dos valores de amplitud constante que van alternando en función del tiempo. Los cortes son bruscos, pero según el estilo de música que se toque, es esperado que así se desee que sea.

Caso 2: Onda de diente de sierra

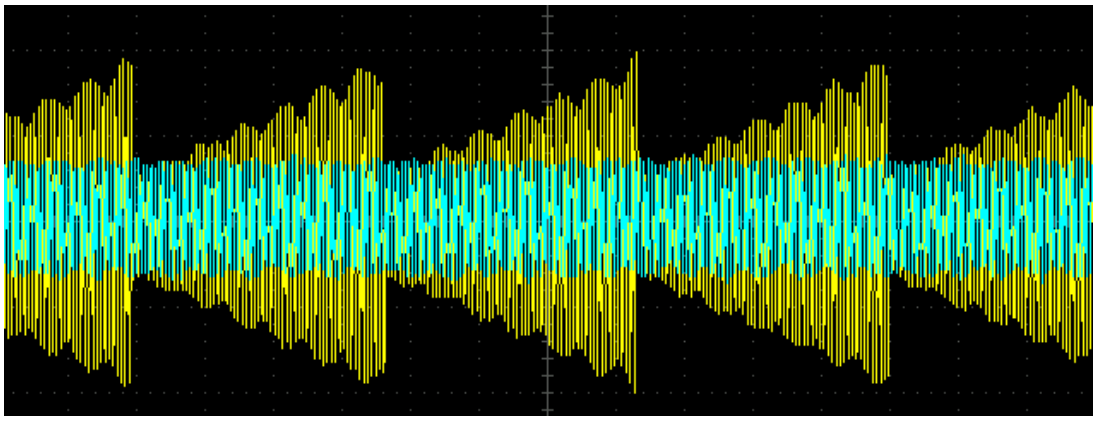


Figura 5.27 Incremento de la amplitud en relación al tiempo (1).

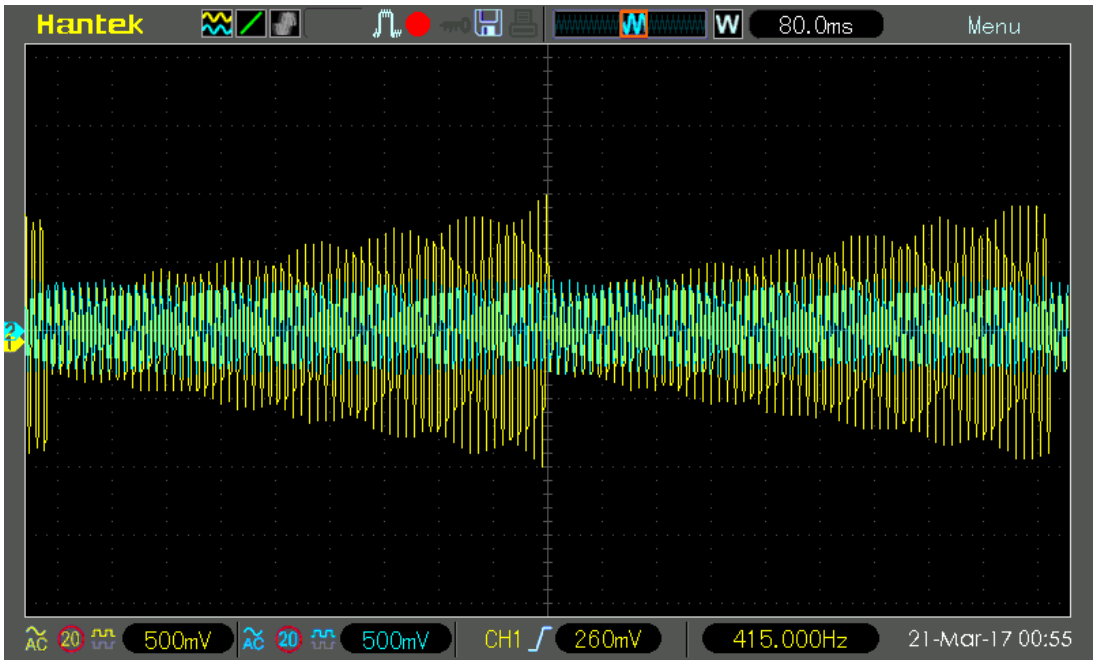


Figura 5.28 Incremento de la amplitud en relación al tiempo (2).

Se aprecia en las imágenes X e Y que la onda tiene un crecimiento constante durante un intervalo, y que una vez pasado vuelve a su posición inicial. En este caso se presenta de forma creciente, también se ha hecho la implementación en forma decreciente.

Caso 3: Onda triangular

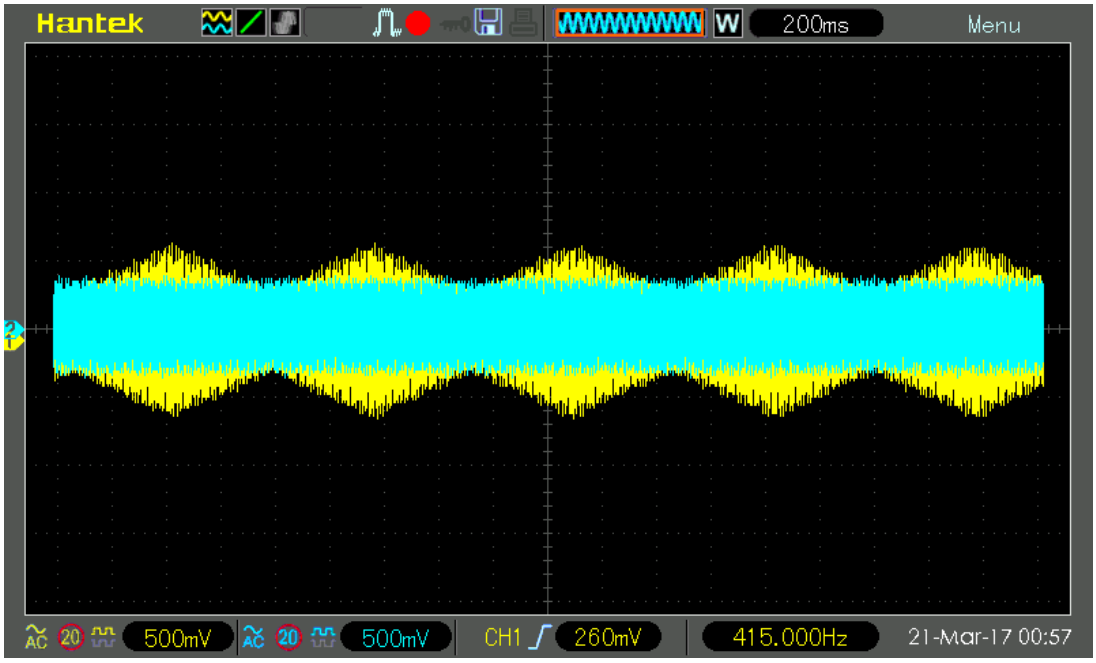


Figura 5.29 Crecimiento y decrecimiento de la amplitud de onda en función del tiempo.

En este último caso se ve como las ondas de la figura X forman en conjunto una forma triangular. La amplitud va variando en función del tiempo desde su valor inicial hasta un máximo, y vuelve a decrecer hasta alcanzar de nuevo su amplitud mínima.

Podría compararse con el primero, pero la transición del estado de mínima amplitud al de máxima se hace de una forma progresiva, por lo que resulta más natural. Esta forma de onda es la más común en los trémolos.

5.4. Implementación de Efectos II (Nios II)

Además de definir módulos hardware que modifiquen el sonido de entrada, en este proyecto también se ha decidido hacer efectos por software. Para ello se utilizará el procesador Nios II. Se trata de un "soft-core", es decir, un procesador que ha sido diseñado por Altera utilizando un lenguaje de descripción de hardware. Este procesador sigue los principios básicos de una arquitectura RISC (Reduced Instruction Set Computer) y usa un conjunto de instrucciones pequeño y optimizado.

Altera permite elegir hasta tres configuraciones diferentes para el componente Nios II. La opción más sencilla es la que se encuentra en la versión gratuita, se trata del Nios II/e, el núcleo económico. Está diseñado para un tamaño óptimo, pero no tiene ni arquitectura pipeline ni caché. Las otras dos versiones son más completas y añaden funcionalidades adicionales, sin embargo, sólo están disponibles en la versión de pago. La versión Nios II/s es el núcleo estándar, está diseñado para tener un tamaño pequeño a la vez que se mantiene

un buen rendimiento. Consta de un pipeline de cinco etapas, caché de instrucciones y predictor de saltos estático. Por último, la versión más completa es la Nios II/f. Es el núcleo más rápido, diseñado para obtener el mejor rendimiento. Tiene un pipeline de seis etapas, caché de instrucciones y de datos y predicción dinámica de saltos. En la siguiente imagen se observa una tabla con las diferencias:

Nios II/e	Nios II/s	Nios II/f
RISC **32-bit**	RISC 32-bit **Instruction Cache** **Branch Prediction** **Hardware Multiply** **Hardware Divide**	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide **Barrel Shifter** **Data Cache** **Dynamic Branch Prediction**
Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache

Tabla 5.2 Comparativa entre configuraciones del procesador Nios II.

El procesador Nios II puede ser utilizado con otros componentes para completar un sistema más grande. A través de la herramienta Qsys se puede diseñar un sistema completo. Altera ofrece distintos módulos predefinidos llamados “IP cores”. Algunos ejemplos pueden ser una CPU, memorias, UART o incluso otros elementos de la FPGA, como controladores de red o interfaces PCI. También existe la posibilidad de usar módulos previamente diseñados por el programador.

5.4.1. Bypass

La primera fase consiste en utilizar el procesador Nios II para que dada una señal de entrada se obtenga una salida idéntica. Para ello se utilizarán módulos previamente diseñados, en este caso son el módulo de configuración del códec y los módulos de entrada y salida de audio. El sistema es el siguiente:

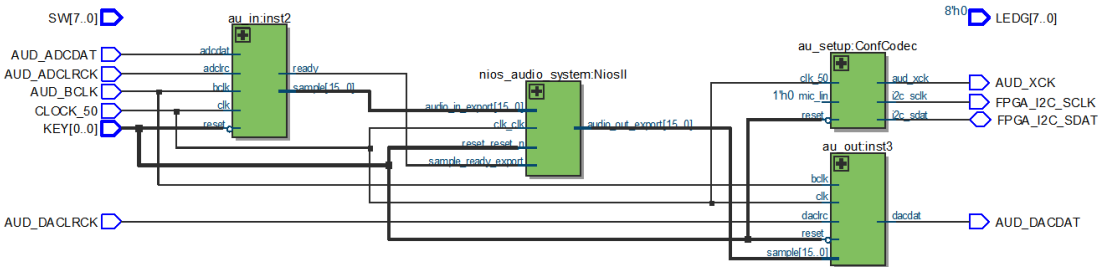


Figura 5.30 Esquema de conexionado de los módulos de configuración del códec y entrada y salida de audio con el procesador Nios II.

La utilidad de este diseño se encuentra en que es el primer paso a dar para poder diseñar futuros efectos. Se desea comprobar que a través de Nios se puede leer de una posición de memoria y escribir en otra posición de memoria, es decir, que los datos de entrada y salida de audio se pueden manipular utilizando el "soft-core".

5.4.1.1. Bypass: Resultados

El resultado ha sido satisfactorio. Se ha escrito un pequeño programa en C que permite la lectura y escritura de las muestras de sonido. La calidad del sonido es buena.

Se han analizado las señales de entrada y salida a través de un osciloscopio, el resultado es el siguiente:

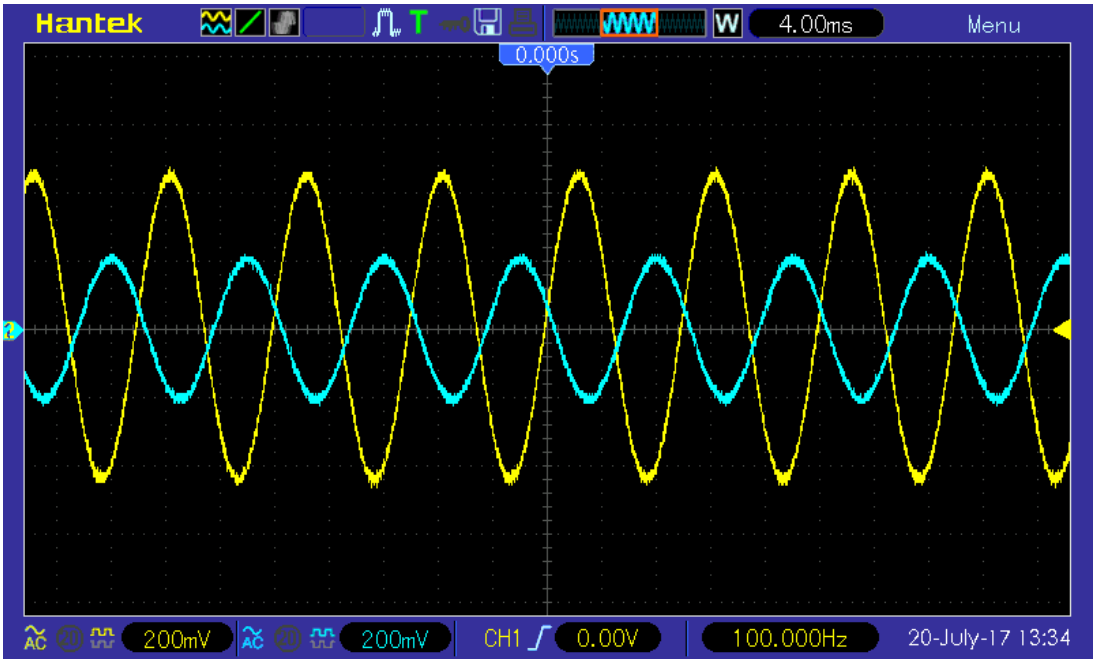


Figura 5.31 Captura de la onda original y la que atraviesa la placa.

En la captura se observa como existe una pequeña atenuación de la onda y como existe un ligero desfase. Sin embargo, éste no es significativo, el oído humano no va a percibir un retardo entre la nota tocada en el instrumento y el sonido que salga a través de la FPGA.

5.4.2. Delay

El efecto de "delay" o retardo consiste en reproducir la muestra de audio actual añadiéndole una muestra retrasada en el tiempo a una distancia determinada.

Al igual que con el bypass, éste es un punto importante, ya que pese a ser un ejemplo aparentemente sencillo, conseguir este hito, abre la posibilidad de poder trabajar con efectos de sonido basados en el tiempo. En la primera parte del proyecto, en la que se

trabajó diseñando módulos en hardware, sólo se llevaron a cabo efectos que cambiaban la forma de la onda, el software permite diseñar nuevos efectos de forma sencilla.

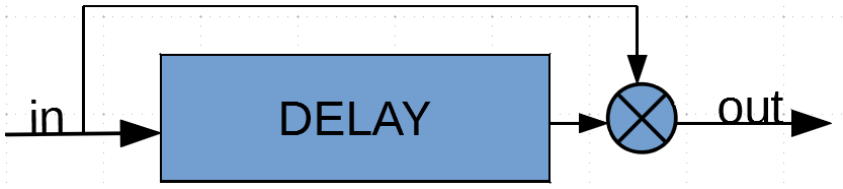


Figura 5.32 Esquema de un “delay” sin retroalimentación.

En una primera instancia se siguió el esquema de la figura anterior, sin embargo, el delay ofrece más posibilidades, y se puede utilizar también con retroalimentación.

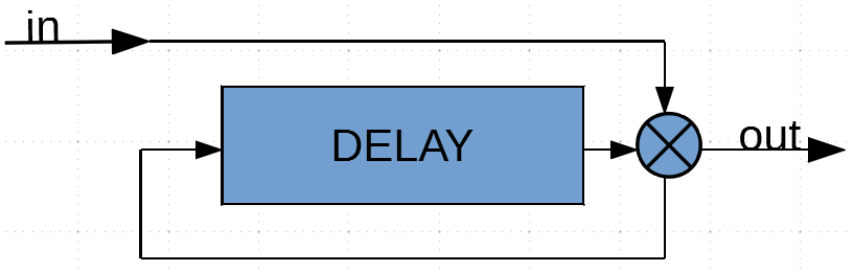


Figura 5.33 Esquema de un “delay” con retroalimentación.

Aunque sólo se ha implementado en su forma más básica, el delay con retroalimentación ofrece más posibilidades. Se podría configurar a qué periodo se empieza a repetir el sonido generado, con qué amplitud de onda (mayor o menor que la señal original generada por el músico) e incluso cuantas veces se repite. Aplicando estas opciones se consigue dar la apariencia acústica de que hay más de un músico tocando.

5.4.2.1. Delay: Resultados

El efecto se ha probado con una fuente de sonido y funciona correctamente. Existe un buffer en el que se almacena el último segundo de audio que llega a través del puerto Line-In, posteriormente, para obtener el resultado final, se suman la muestra actual con la muestra de hace un segundo que estaba guardada. La muestra utilizada se sustituye con la última muestra que llegó.

Como se hace una operación de suma, para evitar el overflow y para que se distinga el sonido original del repetido, las muestras que se toman del buffer son divididas entre dos.

En la siguiente imagen se puede observar el efecto. Para tomar las muestras se ha generado una señal de 100Hz y durante el muestreo se ha eliminado para ver qué es lo que sucedía en la salida cuando ya no había señal de entrada.

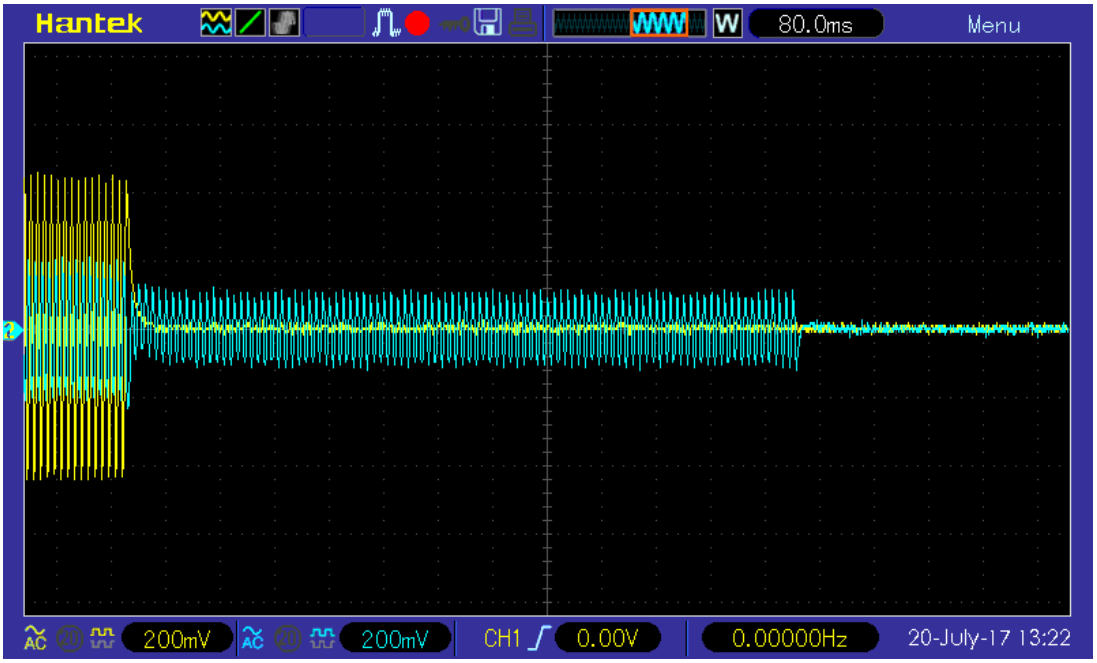


Figura 5.34 Captura del funcionamiento del “delay”.

Se aprecia en amarillo la señal de entrada. Esta desaparece y durante un segundo sigue habiendo señal de salida. Pasado ese tiempo ya no hay señal de salida.

También se observa como al desaparecer la señal de entrada la amplitud de la onda de salida se reduce. Esto ocurre por lo anteriormente explicado, en el algoritmo se hace una división para atenuar el sonido que va con retardo.

5.4.3. Eco

El eco es un fenómeno acústico que se produce cuando una onda se refleja y regresa hacia su emisor. En este caso se trata del efecto producido por una onda acústica. Para que el oído humano lo perciba tiene que superar la persistencia acústica.

La persistencia acústica es un fenómeno por el cual el cerebro interpreta como un único sonido dos sonidos diferentes recibidos en un corto espacio de tiempo. Para que esto ocurra tiene que haber un retraso entre los dos sonidos de entre 70 y 100 milisegundos. Es decir, el eco se percibe cuando la diferencia entre el primer sonido y el segundo es mayor a 100ms.

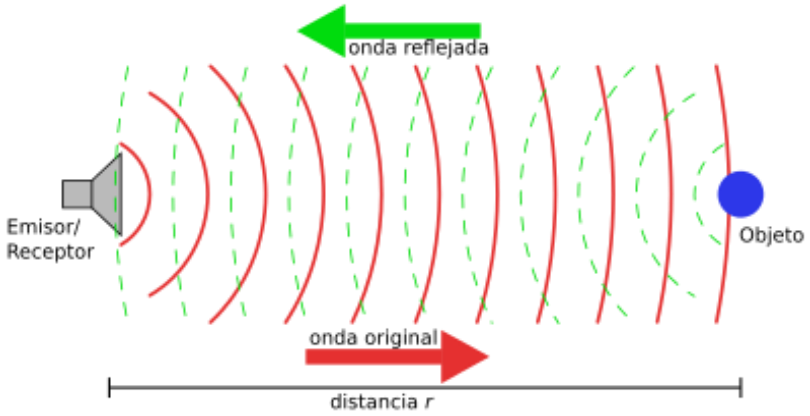


Figura 5.35 Fenómeno de eco.

En la implementación se están capturando muestras de audio a una frecuencia de 48kHz, por lo que para el efecto de eco se tienen que repetir muestras una vez que hayan pasado los 100ms de retardo, es decir, hay que almacenar más de 4.800 muestras para poder repetirlas posteriormente.

A diferencia del efecto de "delay", en el eco se le suma a la señal original una señal anterior atenuada, y existe una retroalimentación, por lo que las muestras más viejas siguen presentes pero cada vez cobran menos notoriedad. El esquema del algoritmo es el siguiente:

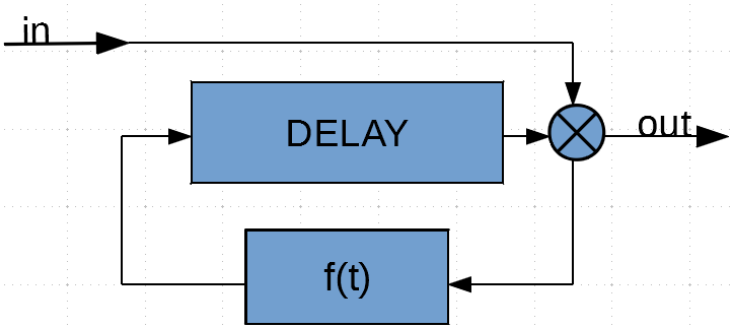


Figura 5.36 Esquema de un efecto de eco.

5.4.3.1. Eco: Resultados

La implementación del efecto de eco ha sido satisfactoria. Al efectuar la prueba con una guitarra se escucha la señal original y después de un tiempo se aprecia el eco que sigue repitiéndose. El efecto es resultante es bastante natural.

Al igual que con los anteriores efectos, se han efectuado pruebas con la ayuda de un osciloscopio y un generador de señales. En la siguiente imagen se muestra el resultado obtenido.

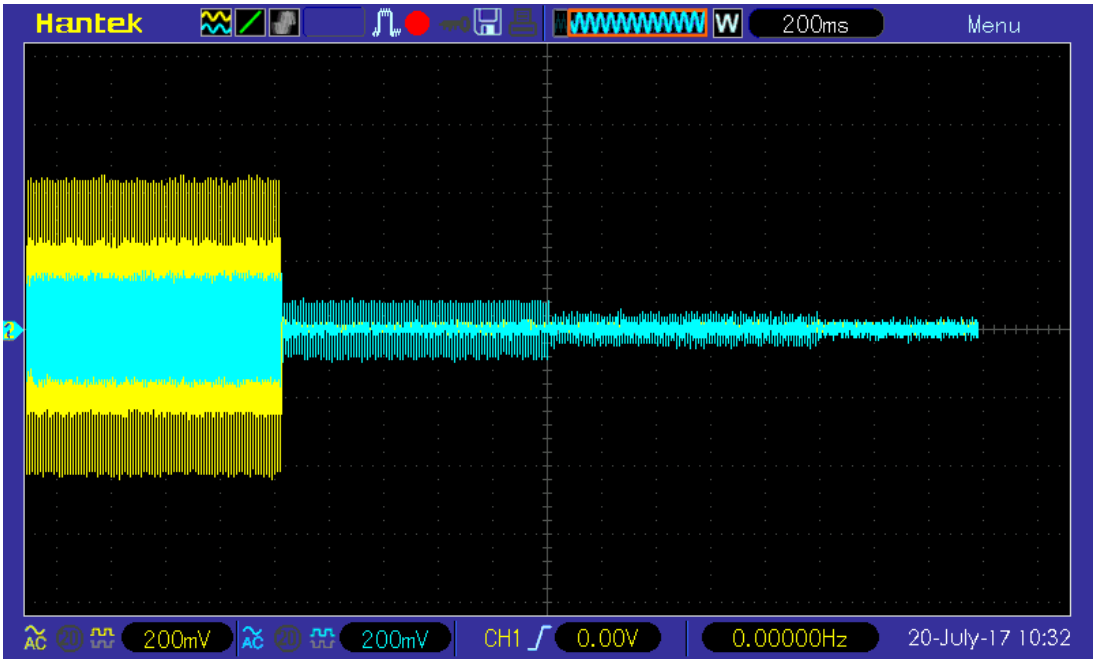


Figura 5.37 Captura del funcionamiento del efecto de eco.

En la imagen se observa que al inicio existe una señal de entrada. Cuando esta señal desaparece, en la salida sigue existiendo una señal. Ésta es el efecto de eco. Se ve como, tras eliminar la señal de entrada, hay una señal que sigue siendo la misma que la anterior pero atenuada, y en cada intervalo de un segundo, vuelve a aparecer la misma señal aún más atenuada. Al final de la captura debería de seguir apareciendo audio, pero debido a que la captura es en tiempo real y el osciloscopio realiza un barrido, aparece en negro.

Con el osciloscopio y el generador de señales no se puede apreciar, debido a que se usa una frecuencia constante, sin embargo, en este efecto existe una retroalimentación que puede durar varios segundos en el tiempo, pero las muestras más antiguas cada vez tienen una atenuación mayor, hasta que dejan de ser inteligibles.

5.4.4. Reverberación

La reverberación es un fenómeno que guarda cierta similitud con el eco. Es producido por la reflexión del sonido y consiste en una ligera permanencia de éste una vez que la fuente original ha dejado de emitirlo.

La principal diferencia está en que el eco es inteligible como un segundo sonido, pero la reverberación se percibe como una adición que modifica el sonido original. Esto es debido a la persistencia acústica, que se ha explicado en el anterior apartado. La reverberación es una reflexión que se percibe en el oído antes de que pasen 100ms.

En la siguiente imagen se observa como en una sala se produce el fenómeno de la reverberación. Se observa como hay una fuente original que genera un sonido directo hacia

los oyentes, y, por otro lado, al haber paredes que hacen que las ondas acústicas se reflejen, llega un sonido reflejado.

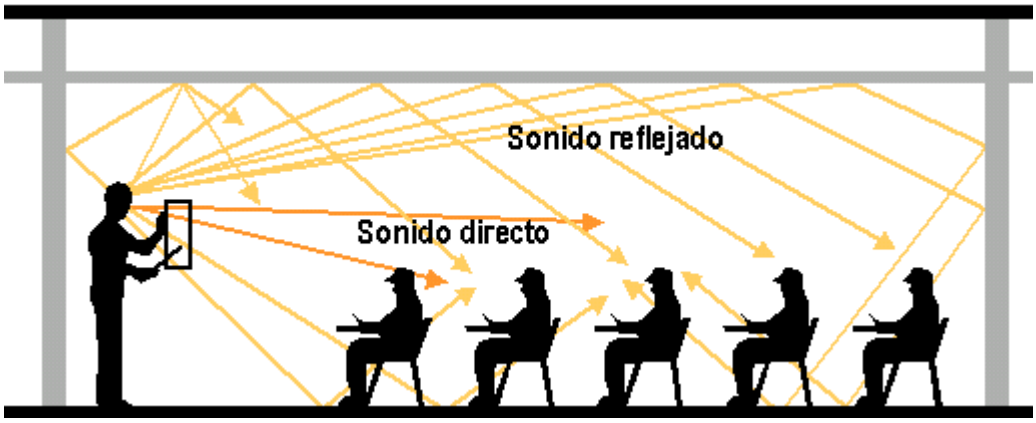


Figura 5.38 Fenómeno de la reverberación.

La reverberación no siempre es igual, depende del lugar. Por ejemplo, el sonido no se refleja igual en una sala que en un teatro o una catedral. En el mercado de los pedales analógicos se ha intentado recrear diferentes tipos de reverberación, imitando diferentes lugares.

En la implementación digital de este efecto se han diseñado dos variantes. Por un lado, se ha construido una reverberación sencilla, que en esencia es parecido al efecto de eco, lo único que cambia es que únicamente se guardan las muestras de los últimos 100ms y se reproducen. El efecto sería similar a que sólo hubiera una fuente de reverberación. Por otro lado, se ha diseñado un segundo efecto que trata de imitar la reflexión de una onda en dos paredes a diferente distancia. Pese a ser ambos efectos de reverberación, se encuentran matices diferentes.

5.4.4.1. Reverberación: Resultados

Al igual que en los anteriores efectos, las pruebas efectuadas sobre este efecto también han sido satisfactorias.

Como se ha explicado anteriormente, se han diseñado dos tipos de reverberación. Son parecidas, pero cuando se prueba con audio, ambas tienen diferentes matices. Se han hecho pruebas con el osciloscopio para las dos situaciones, sin embargo, con este instrumental de medición los resultados son muy similares y es difícil de apreciar matices entre ambos casos. Por esta razón se tratará la reverberación como efecto único, dejando de lado las diferentes variantes diseñadas.

En las siguientes imágenes se muestran los resultados obtenidos:

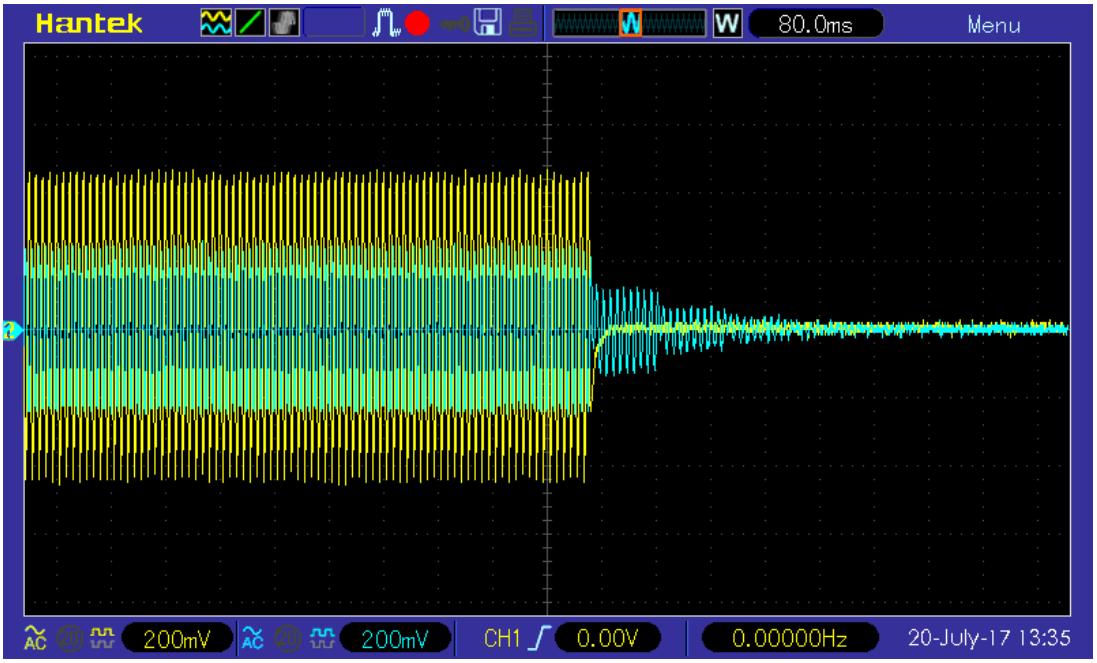


Figura 5.39 Captura del efecto de reverberación.

Las pruebas se han hecho del mismo modo que las anteriores. Se observa como al desaparecer la señal de entrada sigue habiendo una señal de salida. A diferencia del eco, esta señal es mucho más corta y se atenúa más rápidamente.

Figura 5.40 Captura ampliada del efecto de reverberación.

En esta segunda imagen se muestra lo mismo, pero en más profundidad. Se aprecia como la onda sigue teniendo una réplica pero que no puede compararse con la de un eco. Es la suficiente distancia como para se perciba como parte del mismo sonido y no como un eco.

6

Conclusiones

En este último capítulo se hará un análisis y una auto-evaluación del trabajo realizado. Servirá para explicar las complicaciones que han surgido durante el desarrollo del proyecto, así como las lecciones aprendidas que podrían ser útiles para futuros proyectos.

6.1. Conclusiones del proyecto

La idea principal del proyecto se ha completado satisfactoriamente. Se ha diseñado un sistema, sobre una FPGA, que permite la modificación de señales de audio, obteniendo una señal de salida con el efecto de audio seleccionado. En el anterior capítulo se han mostrado los diferentes efectos de audio diseñados, sin embargo, el proyecto tiene potencial para crecer con nuevos efectos y mejoras.

En el alcance del proyecto se especificó llegar a diseñar cuatro efectos de sonido mediante lenguaje de descripción de hardware. Estos efectos han sido realizados y se han efectuado pruebas sobre ellos de forma satisfactoria. Además, una vez cumplidos dichos objetivos, se ha trabajado con el procesador Nios, algo que ha sido nuevo para mí, ya que no había trabajado con ello con anterioridad. Finalmente se ha conseguido añadir otros tres nuevos efectos.

6.2. Conclusiones personales

El desarrollo de este proyecto ha sido muy enriquecedor. Por un lado, ha hecho que me adentre más aún en el tratamiento de audio. Hasta la fecha sólo había trabajado con electrónica analógica para modificar señales de audio, este trabajo me ha abierto las puertas hacia el procesado digital de señales. Por otra parte, también ha conseguido que me interese por dispositivos como las FPGA's, y he descubierto el potencial que tienen en muchos ámbitos.

Ha sido un proyecto con el que, pese a las complicaciones que a veces surgían, he disfrutado. El inicio del proyecto fue la parte más complicada, ya que había conceptos que desconocía y en las pruebas obtenía mucho ruido. Pasada esa primera etapa de "toma de contacto" la situación mejoró. El final del proyecto también tuvo su complicación, ya que el uso del procesador Nios fue una novedad para mí.

La posibilidad de construir un sistema digital relacionado con el sonido y la música me pareció una buena idea, y una oportunidad que no debía dejar pasar. Antes de empezar con este proyecto había hecho varios circuitos analógicos, y había trabajado incluso con válvulas de vacío. Sin embargo, llevaba tiempo planteándome dar el salto a lo digital y, con la ayuda y disponibilidad de mi tutor, la idea ha salido adelante.

Antes de empezar el proyecto tenía ciertos conocimientos sobre cómo se representaba la información analógica en forma digital, pero este trabajo ha conseguido que, en primer lugar, investigue y aprenda nuevas cosas sobre el sonido y su física, y, en segundo lugar, pero no menos importante, aplicar dichos conocimientos y trabajar con señales digitalizadas.

6.3. Dedicación

En el capítulo 2, dedicado a la planificación, se hizo una estimación de las horas que serían necesarias para la realización de este proyecto. En este apartado se mostrarán los tiempos reales consumidos y las diferencias con los tiempos estimados.

Tareas			T. Real (h)	T. Estimado (h)
Planificación			4h.	4h.
Investigación	Audio Digital	10h.	49h.	34
	VHDL	4h.		
	Impedancia	15h.		
	Nios II	20h.		
Implementación	Booster	15h.	113h.	150
	Distorsión	30h.		
	Overdrive	8h.		
	Trémolo	12h.		
	Adapt. Impedancia	18h.		
	Delay	24h.		
	Eco	5h.		
	Reverb	1h.		
Análisis y Resolución de Problemas			36h.	20
Documentación			90h.	80
TOTAL			**292h.**	**288h.**

Tabla 6.1 Comparativa entre planificación y tiempo real.

Se observa como hay algunas desviaciones dentro de los apartados, sin embargo, se han repartido entre otras áreas del proyecto, haciendo que la desviación en el tiempo total sea muy pequeña y siga entrando dentro de lo esperado.

Bibliografía

Libros consultados:

[1] *Transmisión Digital.* D. Martinez, P.J. Reche, N. Ruiz, P. Vera.
Universidad de Jaén (2009)
ISBN: 978-84-8439-490-7

[2] *Introducción al audio digital*. John Watkinson (1994)
ISBN: 84-932844-9-1

[3] *Tratamiento digital de señales*. John G. Proakis, Dimitris G. Manolakis.
Pearson (2007)
ISBN: 978-84-8322-347-5

[4] *Circuitos eléctricos. Primer contacto*. J.L. Galván y J.M. Salcedo.
Anaya (2005)
ISBN: 84-667-4392-8

[5] *Fundamentos físicos de la ingeniería*. J.V. Míguez, F. Mur, M.A. Castro y J. Carpo.
McGraw Hill (2010)
ISBN: 978-84-481-7498-9

[6] *Embedded SoPC design with Nios II processor and VHDL examples*. Pong P. Chu.
Wiley (2011)
ISBN: 978-1-118-00888-1

Enlaces de interés:

[1] Tutoriales de Altera:
Introduction to the Qsys System Integration Tool

[2] Edaboard
Foro especializado en electrónica.

[3] StackExchange
Foro de preguntas y respuestas. Los siguientes subforos han resultado útiles:

- Signal Processing
- Electrical Engineering
- StackOverflow

Anexo A: Glosario y acrónimos

A

Aliasing. Efecto que causa que señales continuas distintas se tornen indistinguibles cuando se muestrean digitalmente. La señal original no puede volver a ser reconstruida.

Audio. Representación del sonido, usualmente como un voltaje eléctrico.

Audio Digital. Codificación de una señal que representa una onda sonora. Se utilizan valores discretizados.

B

Buffer (electrónica). Dispositivo electrónico que sirve para hacer adaptación de impedancias entre circuitos. No afecta a la señal.

Buffer (informática). Espacio de memoria en el que se almacenan datos de forma temporal.

C

CAD. Conversor Analógico-Digital. Dispositivo electrónico capaz de convertir una señal analógica en una señal digital.

CDA. Conversor Digital-Analógico. Convierte una señal digital en una señal analógica.

Códec. Codificador/Decodificador. Hardware capaz de codificar o decodificar una señal o flujo de datos digitales.

Compilación. Convertir un programa en lenguaje máquina a partir de otro programa de computadora escrito en otro lenguaje.

D

Discretización. Técnica que transformar valores continuos en valores finitos mediante una aproximación.

Distorsión. Deformación de una señal.

E

Entorno de simulación. Conjunto de herramientas que permiten probar el comportamiento de un programa o circuito antes de implementarlo.

Espectro. Conjunto de frecuencias que es significativo.

Estéreo. Sonido grabado y reproducido por dos canales.

F

Faradio (F). Unidad del SI de capacidad eléctrica.

FPGA. Field Programmable Gate Array. Dispositivo programable que contiene bloques de lógica cuya interconexión puede ser configurada mediante un lenguaje de descripción especializado.

Frecuencia. Número de periodos por segundo. (1/T)

G

Generador de funciones. Dispositivo electrónico de laboratorio que genera patrones de señales periódicas o no periódicas, tanto analógicas como digitales.

I

I2C. Bus serie de datos. Se utiliza para la comunicación entre diferentes partes de un circuito, por ejemplo, entre un controlador y circuitos periféricos integrados.

Impedancia (Z). Es una medida de oposición que presenta un circuito a una corriente cuando se aplica una tensión.

M

Mono. Sonido que sólo está definido por un canal.

N

Nios II. Procesador embebido con arquitectura de 32 bits diseñado específicamente para la familia de FPGAs de Altera.

O

Ohmio (Ω). Unidad derivada de resistencia eléctrica en el SI.

Osciloscopio. Instrumento de visualización electrónico para la representación gráfica de señales eléctricas que pueden variar en el tiempo.

P

Pedal. Dispositivo electrónico utilizado para alterar el sonido de una fuente, generalmente un instrumento musical.

R

Ruido. Perturbación aleatoria de una señal.

S

Señal. Variación de una magnitud física que lleva información.

Señal Analógica. Señal generada por algún tipo de fenómeno electromagnético y que es representable por una función matemática continua, en la que es variable su amplitud y periodo en función del tiempo.

Señal Digital. Señal en que cada signo que codifica el contenido de la misma puede ser analizado en término de algunas magnitudes que representan valores discretos, en lugar de valores dentro de un cierto rango.

V

VHDL. Lenguaje de Descripción de Hardware utilizado para la automatización de diseño electrónico.

Anexo B: Código y Explicaciones

En este Anexo B se pretende mostrar con más detalle cómo se han elaborado los efectos de sonido, es decir, cómo es su estructura interna. Para ello se mostrará el código y se harán las explicaciones pertinentes.

1. Módulos VHDL (Hardware)

El siguiente análisis se hará sobre los módulos escritos en lenguaje VHDL, es decir, se corresponde a los módulos diseñados en hardware.

1.1. Booster

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity booster is
    port(
        --Datos:
        enable: in std_logic;
        sample_in  : in std_logic_vector(15 downto 0);
        sample_out : out std_logic_vector(15 downto 0);
        --Parametros:
        multiplier : in std_logic_vector(15 downto 0)
    );
end entity booster;

architecture a of booster is
    signal sign_unnorm: std_logic_vector(31 downto 0);
begin

    sign_unnorm <= signed(sample_in)*signed(multiplier);
    boost: process(sign_unnorm)
    begin
        if enable='1' then
            sample_out <= sign_unnorm (31) & sign_unnorm (22 downto 8);
        else
            sample_out<=sample_in;
        end if;
    end process;

end architecture a;
```

El “booster”, para entrar en funcionamiento, se activa con una señal de enable. Tiene una señal de entrada y otra de salida. Por otro lado, también recibe el parámetro “multiplier”, que es el factor que indica la ganancia del efecto de amplificación.

Dado que hay que multiplicar la señal original por un factor, se utiliza una variable auxiliar de mayor tamaño de bits (sign_unnorm), esto evita el desbordamiento de bits.

Se comprueba si la señal “enable” está activa o no, en caso negativo se hace un “bypass”, es decir, la señal de salida es la misma que la de entrada, dejando al sonido pasar sin modificaciones.

En el tercer capítulo se explicaba como la representación de la onda puede tener en un punto determinado un valor positivo o negativo. Por este motivo se coge el primer bit de “sign_unnorm”, ya que expresa el signo y es significativo. Para que la ganancia no sea excesiva, se desprecian los primeros nueve bits después del bit de signo.

1.2. Distortion

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity distortion is
    port(
        --Datos:
        enable: in std_logic;
        sample_in  : in std_logic_vector(15 downto 0);
        sample_out : out std_logic_vector(15 downto 0);
        --Parametros:
        gain : in std_logic_vector(15 downto 0);
        dist_pos : in std_logic_vector(15 downto 0)
    );
end entity distortion;

architecture a of distortion is

    component booster
        port (
            enable : in std_logic;
            sample_in : instd_logic_vector(15 downto 0);
            multiplier : in  std_logic_vector(15 downto 0);
            sample out : out std logic vector(15 downto 0)
        );
    end component;

    signal dist_neg: std_logic_vector(15 downto 0);
    signal signal_dist: std_logic_vector(15 downto 0);
begin
    dist_neg<= -dist_pos;
    boost: process(dist_pos, dist_neg)
    begin
        if enable='1' then
            if (sample_in(15) = '1') then --resultado negativo
                if (signed(sample_in(15 downto 0))<signed(dist_neg(15 downto 0))) then
                    signal_dist <= dist_neg;
                else
                    signal_dist <= sample_in;
                end if;
            else --resultado positivo
                if (signed(sample_in(15 downto 0))>signed(dist_pos(15 downto 0))) then
                    signal_dist <= dist_pos;
                else
                    signal_dist <= sample_in;
                end if;
            end if;
        else
            sample_out<=sample_in;
        end if;
    end process;

    --La distorsion recorta la amplitud de onda, quiero amplificarla.
```

```
s1 : booster
        port map (
                enable => '1',
                sample_in => signal_dist,
                multiplier => gain,
                sample_out => sample_out
        );
end architecture a;
```

El módulo de distorsión, además de tener la entrada y salida de audio acompañada del bit de “enable”, tiene otras dos entradas, que se corresponden con la ganancia y la distorsión.

Este módulo tiene integrado el módulo “booster”, comentado en el punto anterior. Se ha decidido incorporarlo porque la distorsión recorta las frecuencias que están por encima de un umbral, y esto provoca una atenuación.

El algoritmo de la distorsión consiste en obtener a través de un potenciómetro el umbral a partir del cual se va a cortar la señal. Dado que un punto de una onda puede corresponder con un valor positivo o negativo se hacen dos diferenciaciones, y se obtiene el umbral positivo y el negativo.

Si un punto concreto supera la amplitud máxima permitida, se establece el valor máximo posible, en caso contrario, no se modifica.

Por último, la muestra se envía al módulo “booster” para que sea amplificada.

1.3. Overdrive

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity overdrive is
    port(
        enable: in std_logic;
        --Datos:
        sample_in  : in std_logic_vector(15 downto 0);
        sample_out : out std_logic_vector(15 downto 0);
        --Parametros:
        gain : in std_logic_vector(15 downto 0);
        dist pos : in std logic vector(15 downto 0)
    );
end entity overdrive;
```

```
architecture a of overdrive is

    component booster
        port (
            enable: in std_logic;
            sample_in : instd_logic_vector(15 downto 0);
            multiplier : in   std_logic_vector(15 downto 0);
            sample_out : out std_logic_vector(15 downto 0)
        );
    end component;

    signal dist_neg   : std_logic_vector(15 downto 0);
    signal overdrive: std_logic_vector(15 downto 0);
    signal signal_dist: std_logic_vector(15 downto 0);
begin
    dist_neg<= std_logic_vector(-signed(dist_pos));
    boost: process(dist_pos, dist_neg)
    begin
        if enable='1' then
            if (sample_in(15) = '1') then --resultado negativo
                if (signed(sample_in(15 downto 0))<signed(dist_neg(15 downto 0))) then
                    overdrive <= std_logic_vector(signed(sample_in) /4);
                    signal_dist <= dist_neg+overdrive;
                else
                    signal_dist <= sample_in;
                end if;
            else --resultado positivo
                if (signed(sample_in(15 downto 0))>signed(dist_pos(15 downto 0))) then
                    overdrive <= std_logic_vector(signed(sample_in) /4);
                    signal_dist <= dist_pos+overdrive;
                else
                    signal_dist <= sample_in;
                end if;
            end if;
        else
            sample_out<=sample_in;
        end if;
    end process;

    --La distorsion recorta la amplitud de onda, quiero amplificarla.
    --s1 : booster
        port map ( sample_in => signal_dist,
                multiplier => gain,
                sample out => sample out
        );
end architecture a;
```

La explicación de este módulo es prácticamente idéntica a la del anterior, ya que el efecto es una ligera modificación del anterior. Utiliza las mismas señales, también tiene dos controles, uno para la ganancia y otro para el nivel de saturación e incorpora también una conexión con el módulo “booster” para incrementar la amplitud de la señal.

La diferencia de este efecto con la distorsión es la fórmula matemática que se aplica a las muestras. En el anterior efecto, si la muestra superaba un umbral determinado, a ésta se le aplicaba el valor máximo. En el caso del overdrive, no se le aplica ese umbral, sino que, en caso de sobrepasarlo, se le hace un recorte. Se observa en el código las siguientes sentencias:

```
overdrive <= std_logic_vector(signed(sample_in) /4);
signal_dist <= dist_pos+overdrive;
```

La diferencia reside en que ahora la muestra es la suma del umbral máximo y el valor de la muestra dividido entre 4. De este modo la onda sigue conservando una forma parecida, pero se atenúa a partir de un umbral cambiando el sonido.

1.4. Trémolo

El efecto de trémolo consta de dos ficheros que se explicarán a continuación. Por un lado, está el fichero “tremolo.vhd”, que se encarga de ir contando el tiempo que transcurre a través del “rate” (o de la velocidad), que se ajusta a través de un potenciómetro. Por otra parte, está el fichero “wave_setup.vhd”. Éste recibe unos parámetros y se encarga de calcular y enviar por su salida el valor de la ganancia. Además, tiene varias configuraciones para obtener distintas formas de onda.

```
--tremolo.vhd--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity tremolo is
    port(
        --Datos:
        sample_in  : in std_logic_vector(15 downto 0);
        sample_out : out std_logic_vector(15 downto 0);
        LD_Sample  : in std_logic;
        clk        : in std_logic;
        cl     : in std_logic;
        --Parametros:
        rate    : in std_logic_vector(15 downto 0);--Velocidad del tremolo
        attack : in std_logic_vector(15 downto 0);--Diferencia entre ampl min y max.
        wave    : in std_logic_vector(1 downto 0)  --Forma de la onda.
    );
end entity tremolo;


architecture a of tremolo is

    component wave_setup
    port(
            --Datos:
            rate    : in std_logic_vector(15 downto 0);
            wave    : in std_logic_vector(1 downto 0);
            attack : in std_logic_vector(15 downto 0);
            gain    : out std_logic_vector(15 downto 0);
            --Control:
            counter: in std_logic_vector(18 downto 0); --Se necesita saber en que
posicion esta el cl_counter del tremolo.
            clk_Samp: in std_logic; --clk_sample. Se necesita saber cuando ha llegado
una muestra. En el otro modulo, LD_Sample.
            clk : in std_logic;
            cl  : in std logic
        );
    end component;

    component booster
        port (
            sample in : instd logic vector(15 downto 0);
            multiplier : in   std_logic_vector(15 downto 0);
            sample_out : out std_logic_vector(15 downto 0)
        );
    end component;

    signal amplitude  : std_logic_vector(15 downto 0);
    signal cls_counter    : std_logic_vector(18 downto 0);
    signal doble      : std_logic_vector(18 downto 0);
    constant mult_por_dos: std_logic_vector(2 downto 0):="010"; --Multiplicador por
dos, mejor que dividir.
    constant cero     : std_logic_vector(18 downto 0):="0000000000000000000";
```

```
--Valores que toma la amplitud
    constant low_gain     : std_logic_vector(15 downto 0):="0000000010000001";
    constant high_gain    : std_logic_vector(15 downto 0):="0000001111111111";
    --Valores que toma la frecuencia
    signal ratio      : std_logic_vector(15 downto 0);--El valor que almaceno de lo
que llega del ADC (Pot)
    signal ataque     : std_logic_vector(15 downto 0);--El valor que almaceno de lo
que llega del ADC (Pot)
    constant max_freq     : std_logic_vector(15 downto 0):="0000100101100000"; --10Hz
(Subida y bajada en 100ms)
    constant min_freq     : std_logic_vector(15 downto 0):="0100011001010000"; --1.5Hz

begin
    --Quiero definir un ratio minimo y maximo. El ratio SOLO quiero que se actualice
cuando haya terminado de contar.
    ratio<= max_freq when (rate<max_freq and cls_counter=cero) else --10Hz
        min_freq when (rate>min_freq and cls_counter=cero) else --1.5Hz
        rate when (cls_counter=cero);

    --Lo mismo para el ataque. Solo quiero que se actualice en cada ciclo.
    ataque<= attack when (cls_counter=cero);

    tremolo: process(clk)
    begin
        doble<=signed(ratio)*signed(mult_por_dos);
        if(cl='1')then
            cls_counter<=cero;
        elsif(clk'event and clk='1') then
            if(LD_Sample='1') then
                if (cls_counter=doble)then
                    cls_counter<=cero;
                else
                    cls_counter<=cls_counter+'1';
                end if;
            end if;
        end if;
    end process;

    --s0   Instanciar un componente que me diga que amplitud usar.
    s0 : wave_setup
        port map ( rate   => ratio,
                wave   => wave,
                attack => ataque,
                gain   => amplitude,
                counter => cls_counter,
                clk_Samp=> LD_Sample,
                clk => clk,
                cl  => cl
            );

    --s1   Tengo la amplitud. Instanciar al componente que me cambia la amplitud.
    s1 : booster
        port map ( sample_in => sample_in,
                multiplier => amplitude,
                sample_out => sample_out
        );

end architecture a;
```

Éste es el fichero “tremolo.vhd”. El módulo tiene las señales de entrada y salida de audio. Además, como es un efecto que varía en el dominio del tiempo, es necesario que tenga las señales de reloj (clk), llegada de muestra (LD_Sample) y borrado (cl). Cómo parámetros externos, recibe tres tipos. Por un lado, está el “rate”, que define la velocidad o frecuencia a la que oscilará el trémolo. En segundo lugar, está el “attack”, esto significa con qué fuerza se percibe el sonido, es decir, la diferencia entre la mínima y la máxima amplitud. Cuanta mayor diferencia halla, se dirá que el “ataque” es mayor. Por último, existe el parámetro “wave”, que define la forma de la onda, es decir, si es cuadrada, triangular o de sierra.

Luego hay otras constantes definidas. Las más importantes son los límites, tanto de ganancia como de frecuencia. El volumen está acotado para que no haya un cambio excesivo, y la frecuencia se ha acotado del mismo modo que en un pedal analógico, para que vaya desde 1,5 a 10Hz.

El fichero "tremolo.vhd" lo que hace en su algoritmo es que dado un valor de repetición a través del potenciómetro (rate), va contando cuanto tiempo pasa, para eso utiliza la señal "LD_Sample", que informa cuando llega una nueva muestra y se sabe que la frecuencia de muestreo es de 48kHz, por lo tanto, un segundo se compone de 48.000 muestras.

Éste contador se envía al módulo "wave_setup", que es el encargado de calcular el multiplicador para cambiar la amplitud de la onda. Cuando el módulo "tremolo" reciba esta información, enviará la muestra y la amplitud al "booster", que será el que finalmente altere el volumen del sonido.

```
--wave_setup.vhd--

library IEEE;
use IEEE.std_logic_1164.all;
--use IEEE.std_logic_arith.all; --Obsoleta. No tiene division. numeric_std sustituye.
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity wave setup is
    port(
        --Datos:
        rate   : in std_logic_vector(15 downto 0);
        wave   : in std_logic_vector(1 downto 0);
        attack : in std_logic_vector(15 downto 0);
        gain   : out std_logic_vector(15 downto 0);
        --Control:
        counter: in std_logic_vector(18 downto 0); --Se necesita saber en que
posicion esta el cl_counter del tremolo.
        clk_Samp: in std_logic; --clk_sample. Necesito saber cuando ha llegado una
muestra. En el otro modulo, LD_Sample.
        clk : in std_logic;
        cl  : in std_logic
    );
end entity wave_setup;

architecture a of wave_setup is

    constant cero : std_logic_vector(15 downto 0):="0000000000000000";
    --Valores que toma la amplitud
    signal min_gain   : std_logic_vector(15 downto 0):="0000000010000001"; --
Inicializo.
    constant max_gain     : std_logic_vector(15 downto 0):="0000001111111111";
    --Contador
    signal gain_counter   :std_logic_vector(15 downto 0); --Cociente de la division.
Cada cuantos pasos hay que incrementar (sumar uno a) la ganancia.

    --otras:
    signal gain_cl_counter   :std_logic_vector(7 downto 0);
    signal gain_aux       :std_logic_vector(15 downto 0);
    signal diferencia :unsigned(15 downto 0);
    signal cociente       :unsigned(15 downto 0);
```

```
begin

    set_amplitude: process(rate, wave, counter, clk, cl)
    begin
        if(cl='1')then
            gain_counter<=cero;
            gain_cl_counter<="00000000";
        elsif(clk'event and clk='1') then
            --Configuro cual va a ser el ataque
            min_gain<=std_logic_vector(unsigned(attack)/integer(33));
            if (wave="00") then --Onda Cuadrada
                if (counter(15 downto 0)<rate) then
                    gain<=min_gain;
                else
                    gain<=max_gain;
                end if;
            elsif(wave="01") then --Onda de Sierra
                if(clk_Samp='1')then
                    diferencia<=(unsigned(max_gain)-unsigned(min_gain));
                    cociente<=unsigned(rate)/diferencia;
                    gain_counter<=std_logic_vector(cociente);
                    if(counter(15 downto 0)=cero)then
                        gain_aux<=min_gain;
                    else
                        if(gain_cl_counter<gain_counter(7 downto 0)) then
                            gain_cl_counter<=gain_cl_counter+'1';
                        else
                            gain_aux<=gain_aux+'1';
                            gain_cl_counter<="00000000";
                        end if;
                    end if;
                    gain<=gain_aux;
                end if;
            elsif(wave="10") then --Onda triangular
                if(clk_Samp='1')then
                    diferencia<=(unsigned(max_gain)-unsigned(min_gain));
                    cociente<=unsigned(rate)/diferencia;
                    gain counter<=std logic vector(cociente);
                    if(counter(15 downto 0)=cero)then
                        gain_aux<=min_gain;
                    elsif (counter(15 downto 0)<rate) then
                        if(gain_cl_counter<gain_counter(7 downto 0)) then
                            gain_cl_counter<=gain_cl_counter+'1';
                        else
                            gain_aux<=gain_aux+'1';
                            gain_cl_counter<="00000000";
                        end if;
                    else
                        if(gain_cl_counter<gain_counter(7 downto 0)) then
                            gain cl counter<=gain cl counter+'1';
                        else
                            gain_aux<=gain_aux-'1';
                            gain_cl_counter<="00000000";
                        end if;
                    end if;
                    gain<=gain_aux;
                end if;
            else
                gain<="0000000000000000";
            end if;
        end if;
    end process;

end architecture a;
```

A través del parámetro “attack” se configura la ganancia mínima “min_gain”. A partir de ahí, dependiendo del tipo de forma de onda seleccionado, se hacen unas operaciones u otras. Por ejemplo, en el caso de onda cuadrada, se alterna la salida entre “min_gain” y “max_gain”, mientras que en el caso de sierra. Se va incrementando una variable auxiliar.

2. Efectos en C (Software)

En este apartado se explican aquellos efectos que han sido realizados por software, utilizando el procesador Nios II y el lenguaje de programación C.

2.1. Bypass

```
#define au_in (volatile short *) 0x0081050  //Dirección de memoria de entrada.
#define au_out (volatile short *) 0x0081040
#define samp_rdy (volatile int *) 0x0081030 //Indica si una nueva muestra ha llegado.

void bypass() //BYPASS
{ while (1)
    if((*(samp_rdy + 0x3)==0x1)){
        *au_out = *au_in;
        *(samp_rdy + 0x3)=0x0;
    }
}
```

El primer ejemplo es el “bypass”. Es la base para comprender los siguientes casos. El objetivo del “bypass” es conseguir que el sonido de entrada sea el mismo que el de salida, pero usando el procesador Nios.

Se observa que hay que definir las direcciones de memoria donde llegan los datos. La función bypass es un bucle infinito en el que, cada vez que la dirección de memoria asociada al puntero “*samp_rdy” cambia, entonces se asocia el valor del puntero “*au_in” a “*au_out”. Posteriormente se vuelve a poner “samp_rdy” a 0, para que, cuando vuelva a haber otra muestra disponible, se pueda apreciar el cambio de valor.

2.2. Delay

```
#define au_in (volatile short *) 0x0081050
#define au_out (volatile short *) 0x0081040
#define samp_rdy (volatile int *) 0x0081030
#define BUFF_SIZE (48000)

void delay(){ //REPLAY
    static short buff[BUFF_SIZE];
    for(int i=0; i<BUFF_SIZE;){
        if(*(samp_rdy + 0x3)){ //Si hay una nueva muestra
           buff[i]= (short)*au_in;
           i++;
           *(samp_rdy + 0x3)= 0;
        }
    }

    while (1){
        for(int i=0; i<BUFF_SIZE;){
            if(*(samp_rdy + 0x3)){
              *au_out = (buff[i]/2) + (*au_in);   //Suena muestra actual + atrasada.
              buff[i] = (short)*au_in;            //Reemplaza la muestra en el array.
              i++;
              *(samp_rdy + 0x3)= 0;
            }
        }
    }
}
```

El efecto de “delay” lo que hace es reproducir un sonido actual, mientras que de fondo suena un sonido que sonó hace un segundo a un volumen más bajo.

Para ello se observa que hay una nueva constante, que define el tamaño de un array. El valor es de 48.000 debido a que son las muestras necesarias para almacenar un segundo de audio.

El algoritmo cuenta de un bucle inicial, el cual sirve para almacenar el primer segundo de audio. Cada vez que llega una muestra nueva, ésta se almacena en un array.

Posteriormente hay un bucle infinito. En este bucle se toma la muestra actual y se le suma la muestra atenuada de lo que se guardó hace un segundo en el array. Esa posición del array es reemplazada con la muestra actual, sin ser mezclada con ninguna otra muestra. Para hacer esto se tiene siempre en cuenta la señal “samp_rdy”, que informa de cuándo ha llegado una nueva muestra.

2.3. Eco

```
#define au_in (volatile short *) 0x0081050
#define au_out (volatile short *) 0x0081040
#define samp_rdy (volatile int *) 0x0081030
#define BUFF_SIZE (48000)

void echo(){ //ECHO
    static short buff[BUFF_SIZE];
    for(int i=0; i<BUFF_SIZE;){
        if(*(samp_rdy + 0x3)){
       buff[i]= (short)*au_in;
           i++;
       *(samp_rdy + 0x3)= 0;
        }
    }

    while (1){
        for(int i=0; i<BUFF_SIZE;){
            if(*(samp_rdy + 0x3)){
              *au_out = (buff[i]/2) + (*au_in);   //Suena muestra actual + atrasada.
              buff[i] = (short)*au_out;       //Reemplaza la muestra con la mezcla.
              i++;
              *(samp_rdy + 0x3)= 0;
            }
        }
    }
}
```

El funcionamiento es casi idéntico al del “delay”. El cambio en este caso es que existe la retroalimentación, es decir, se guarda en el array la muestra modificada, no la original. La diferencia se puede ver en estas dos sentencias:

Delay: `buff[i] = (short)*au_in;`

Eco: `buff[i] = (short)*au_out;`

2.4. Reverb

```
#define au_in (volatile short *) 0x0081050
#define au_out (volatile short *) 0x0081040
#define samp_rdy (volatile int *) 0x0081030
#define REV_SIZE (4800)


void reverb(){ //Reverb
    static short buff[REV_SIZE];
    for(int i=0; i<REV_SIZE;){
        if(*(samp_rdy + 0x3)){
            buff[i]= (short)*au_in;
          i++;
            *(samp_rdy + 0x3)= 0;
        }
    }

    while (1){
        for(int i=0; i<REV_SIZE;){
            if(*(samp_rdy + 0x3)){
                *au_out = (buff[i]/2) + (*au_in);
                buff[i] = (short)*au_out;
              i++;
                *(samp_rdy + 0x3)= 0;
            }
        }
    }
}
```

Es idéntico al eco, pero se cambia el tamaño del array. Cómo se explicó en el capítulo de implementación, la reverberación es un eco muy corto que no supera los 100ms, así que lo que se ha hecho es poner el tamaño del array a 4.800, ya que con una frecuencia de muestreo de 48kHz, 4.800 muestras equivalen a 100ms.

2.5. Reverb Cathedral (Otra modificación)

```
#define au_in (volatile short *) 0x0081050
#define au_out (volatile short *) 0x0081040
#define samp_rdy (volatile int *) 0x0081030
#define REV_SIZE (4800)


void rev_cath(){ //Reverb Cathedral
    static short buff[REV_SIZE];
   int j=REV_SIZE-2000;
    for(int i=0; i<REV_SIZE;){
        if(*(samp_rdy + 0x3)){
            buff[i]= (short)*au_in;
          i++;
            *(samp_rdy + 0x3)= 0;
        }
    }

    while (1){
        for(int i=0; i<REV_SIZE;){
            if(*(samp_rdy + 0x3)){
              *au_out = (buff[i]/4) +(buff[j]/2)+ ((*au_in)/2);
              buff[i] = (short)*au_out;
              i++;
              if(j==REV_SIZE){
                  j=0;
              }else{
```

```
                    j++;
                }
                  *(samp_rdy + 0x3)= 0;
            }
        }
    }
}
```

Es una modificación de la anterior reverberación. Se supone que en una habitación cerrada el sonido puede reverberar y colisionar contra más de una barrera. En esta implementación se pretende simular dos choques a distinta distancia. Uno reverberará pasados 100ms y otro cerca de los 60ms, produciéndose una reverberación más rica en matices.

Para eso se han hecho dos contadores para sumar las dos reverberaciones. Uno va de 0 a 4.800 y el otro de 0 a 2.800.

Anexo C: Recogida de muestras

En este anexo se explicará con más detalle cómo se ha realizado la recogida de muestras, tanto en el caso de usar el osciloscopio, como para obtener fragmentos de audio reales.

1. Muestras Ideales

Para hacer pruebas y obtener capturas de cómo afectan los diferentes efectos a una señal sinusoidal se han empleado dos herramientas, el generador de funciones y el osciloscopio. A continuación, se hace una breve explicación de su funcionamiento.

El generador de funciones es un dispositivo electrónico que produce una señal (en este caso utiliza corriente alterna) con una amplitud y frecuencia predeterminada por el usuario. Lo bueno de este aparato es que permite tener una onda periódica casi perfecta. Si la fuente en vez de ser un generador de funciones fuera la propia guitarra, sería mucho más difícil ver en un osciloscopio la alteración de la señal.

El osciloscopio es un dispositivo de visualización gráfica que muestra señales eléctricas variables en el tiempo. El eje vertical (Y) representa el voltaje y el horizontal (X) representa el tiempo. Con un osciloscopio se puede determinar directamente el periodo y el voltaje de una señal, y, de forma indirecta la frecuencia de una señal, así como la fase entre dos señales. Además, el osciloscopio también permite determinar qué parte de la señal es ruido y cómo varía este con el tiempo. También es útil como herramienta de depuración, para localizar problemas en un circuito.

En la siguiente imagen se muestra un esquema del conexionado de los tres elementos para realizar la recogida de muestras:

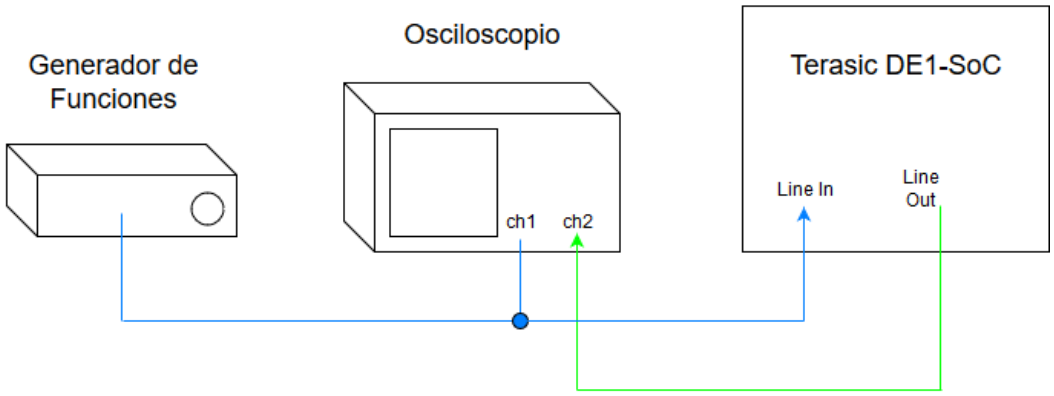


Figura C.1 Esquema del conexionado para el análisis del circuito.

En el generador de funciones se ha utilizado generalmente una onda de baja frecuencia, con una amplitud menor a un voltio de pico a pico ($1V_{pp}$). La señal del generador se ha introducido a la placa con la ayuda del chasis de un Jack de 3,5mm. En ese mismo chasis se ha conectado el canal 1 del osciloscopio para obtener muestras de la señal limpia. En la salida de audio de la placa se ha conectado otro chasis, y a éste, se le ha conectado el canal 2 del osciloscopio.

Utilizando un osciloscopio de dos canales se puede ver en la pantalla al mismo tiempo la onda original y la onda modificada después de haber pasado por el circuito de la FPGA. Las capturas de osciloscopio que aparecen en la memoria han sido recogidas de esta forma.

2. Muestras de Audio

Además de las capturas del osciloscopio, se ha decidido acompañar a esta memoria con fragmentos reales de audio. Para ello, en primer lugar, se han grabado pequeños fragmentos musicales con una guitarra, el adaptador de impedancias y un PC.

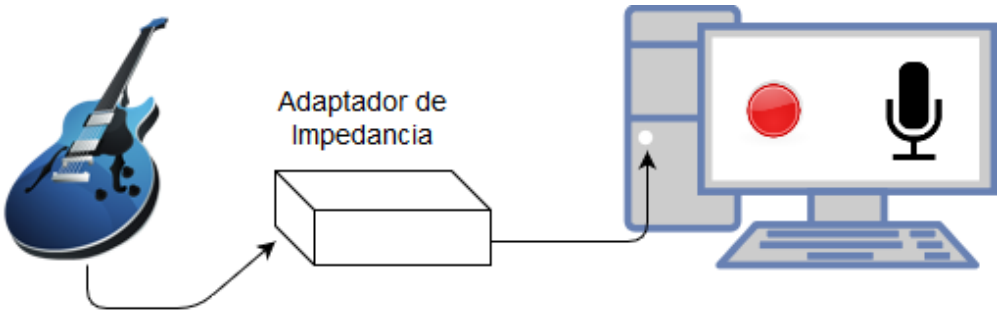


Figura C.2 Esquema del conexionado para la grabación de audio limpio.

En segundo lugar, se han pasado los archivos de audio a un dispositivo móvil con salida de audio y se ha conectado a la placa. La salida de audio de la placa va también conectada a un PC, que será el encargado de grabar de nuevo el sonido con el efecto aplicado.

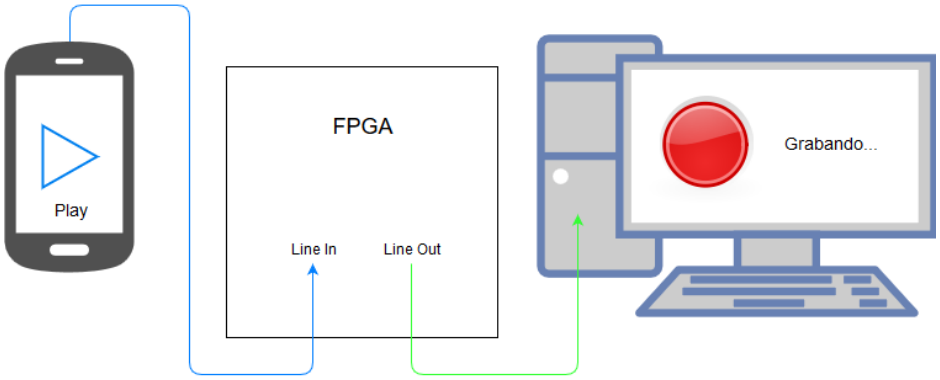


Figura C.3 Esquema del conexionado para la grabación de audio modificado.