UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN SOBRE FPGA DE UN PEDAL MULTI-EFECTOS DIGITAL SOBRE PROTOCOLO I2S2

EROS GARCÍA ARROYO

ENERO 2020

TRABAJO DE FIN DE GRADO

Título:	Diseño e implementación sobre FPGA de un pedal multi-efectos digital sobre protocolo I2S2
Título (inglés):	Design and implementation on FPGA of a digital multi-effects pedal on I2S2 protocol
Autor:	Eros García Arroyo
Tutor:	Samuel López Asunción
Ponente:	Pablo Ituero Herrero
Departamento:	Departamento de Ingeniería Electrónica

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería Electrónica
LSI

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN SOBRE FPGA DE UN PEDAL MULTI-EFECTOS DIGITAL SOBRE PROTOCOLO I2S2

Eros García Arroyo

Enero 2020

Resumen

Este trabajo persigue y tiene como principal finalidad obtener un pedal multiefectos, plenamente funcional a tiempo real, para guitarras eléctricas o sistemas de captación de audio, como micrófonos, aunque el objetivo principal será la guitarra eléctrica. De esta forma lo que se pretende es que, a través de una señal de entrada, consigamos hacer variarla de manera digital y obtener a su salida otra señal modificada por el efecto seleccionado dentro de la arquitectura del pedal.

Para definir los múltiples efectos del pedal debemos tener en cuenta que vamos a tratar con 3 tipos de resultados:

- Efectos basados en retardadores.
- Efectos basados en sistemas no lineales.
- Efectos basados en sistemas lineales.

El producto de los efectos retardadores tendrán como finalidad añadir una línea de retardo o de espera sobre la señal de entrada que luego se solapará junto con la señal de salida. Los efectos no lineales se encargarán de afectar a la ganancia de la señal de entrada para obtener una modificación no lineal de está a la salida y, finalmente, los efectos lineales tienen una función similar a los descritos anteriormente pero, como su categoría indica, será de forma lineal.

El proyecto partirá desde la elección de la FPGA, protocolo y algoritmo de trabajo, hasta la obtención de un pedal multiefectos perfectamente implementado y funcional como prototipo y siendo este el resultado final del proyecto.
Además, debido a las características de este proyecto y al lenguaje de programación empleado (VHDL) y el entorno de desarrollo utilizado (Vivado), podrá modificarse en versiones futuras de manera muy sencilla y contener los efectos que el usuario necesite dependiendo de la aplicación profesional que se le quiera dar.
Por tanto, el prototipo desarrollado en este trabajo fin de grado me ha servido para afianzar mis conocimientos sobre electrónica digital y tratamiento/procesado de señales de audio.

Palabras clave: Pedal multiefectos, FPGA, VHDL, I2S2, Nexys A7, Tiempo real, Línea de retardo, Umbral de ganancia, Máquina de estados finita con ruta de datos.

Abstract

This work pursues and has as its main purpose to obtain a multi-effects pedal, fully functional in real time, for electric guitars or audio capture systems, such as microphones, although the main object will be the electric guitar. In this way the aim is that, through an input signal, we can make it vary digitally and get at its output another signal modified by the selected effect within the pedal architecture.

To define the multiple effects of the pedal we must take into account that we will deal with 3 types of results:

- Retardant-based effects.
- Effects based on non-linear systems.
- Linear system-based effects.

The product of the retarding effects will aim to add a delay or wait line over the input signal which will then overlap along with the output signal. Nonlinear effects will be responsible for affecting the gain of the input signal to obtain a nonlinear modification of is to the output and eventually, linear effects have a function similar to the one described above but, as its category indicates, it will be linearly.

The project will start from the choice of FPGA, protocol and working algorithm, to obtaining a perfectly implemented and functional multi-effects pedal as a prototype and this being the final result of the project.

In addition, due to the features of this project and the programming language used (VHDL) and the development environment used (Vivado), it can be modified in future versions in a very simple way and contain the effects that the user needs depending on the professional application you want to give it.
Therefore, the prototype developed in this end-of-degree work has served me, to strengthen my knowledge of digital electronics and the treatment/processing of audio signals.

Keywords: Multi-effects pedal, FPGA, VHDL, I2S2, Nexys A7, Real-time, Delay line, Gain threshold, Finite-state machine with datapath.

Agradecimientos

Me gustaría agradecer a todos aquellos que me han apoyado y ayudado desinteresadamente durante todo este tiempo sin esperar nada a cambio: mis padres y mi hermana por haberme ayudado en los momentos difíciles cuando empecé este camino universitario, a mis abuelos paternos por haberme acogido en su casa y a mi abuela materna por haberme cuidado desde que era pequeño. También quiero hacer una mención especial a mi tutor Samuel López por haberme ayudado todas y cada una de las semanas desde que decidimos llevar a cabo el proyecto y a Pablo Ituero por haberme ofrecido la oportunidad de llevarlo a cabo.

Índice general

Resumen **I**

Abstract **III**

Agradecimientos **V**

Índice general **VII**

Lista de figuras **XI**

1. Introducción, estado del arte y motivación **1**

1.1. Introducción y estado del arte 1

1.2. Objetivos del proyecto 3

1.3. Estructura del documento 3

1.4. Metodología 4

1.5. Resultados esperados 5

2. Interfaz de sonido externa y FPGA **7**

2.1. Sistema de entrada/salida del audio 7

2.1.1. Etapa de pre-amplificación a la entrada 7

2.1.2. Etapa de amplificación a la salida 9

2.2. FPGA 10

2.3. Captación del sonido en la guitarra eléctrica 11

3. Análisis teórico de los efectos digitales **13**

3.1. Teoría sobre los efectos digitales estudiados e implementados 13

3.1.1. Efectos basados en retardadores 13

3.1.1.1. Efecto retardo 14

3.1.1.2. Efecto vibrador 15

3.1.1.3. Efecto coro 17

3.1.1.4. Efecto reverberador 19

3.1.1.5. Efecto eco 20

3.1.2. Efectos basados en sistemas no lineales 21

3.1.2.1. Efecto sobre-saturación 21

3.1.2.2. Efecto compresor 22
3.1.3. Efectos basados en sistemas lineales 23
3.1.3.1. Efecto entrada/salida 23
3.1.3.2. Efecto looper 24
3.1.3.3. Efecto banco de filtros 25
3.2. Efectos digitales estudiados pero no implementados 27
3.2.1. Efecto autowah 27
3.2.2. Efecto octavador 29

4. Implementación digital de los efectos **31**
4.1. Protocolo i2s2 (Arquitectura i2s2) 31
4.2. Arquitectura desarrollada para la FPGA y distribución del hardware 33
4.3. Matriz de diodos Leds y Displays de 7 segmentos 34
4.3.1. Vúmetro LED 34
4.3.2. Play/Pause 35
4.4. Implementación de los efectos 36
4.4.1. E/S 36
4.4.2. Looper 36
4.4.3. BankFilter 37
4.4.4. Delay 38
4.4.5. Vibrato 39
4.4.6. Chorus 39
4.4.7. Reverb 40
4.4.8. Eco 41
4.4.9. Overdrive 41
4.4.10. Compressor 41
4.5. Recursos consumidos por el código VHDL implementado 41

5. Pruebas sobre el algoritmo desarrollado **43**
5.1. Pruebas sobre el i2s2 43
5.2. Pruebas sobre los LEDs 44
5.3. Pruebas sobre los efectos desarrollados 44
5.3.1. Prueba E/S 44
5.3.2. Prueba Looper 44
5.3.3. Prueba BankFilter 44
5.3.4. Prueba Delay 44
5.3.5. Prueba Vibrato 44

5.3.6. Prueba chorus 44
5.3.7. Prueba Reverb 44
5.3.8. Prueba Eco 44
5.3.9. Prueba Overdrive 44
5.3.10. Prueba Compressor 44

6. Conclusiones y trabajo futuro **45**
6.1. Conclusiones 45
6.2. Trabajo futuro 45

A. Impact of this project **I**
A.1. Social impact I

B. Economic budget **III**
B.1. Physical resources III

C. Diagramas de bloques para VHDL **V**
C.1. Arquitectura del transceiver i2s2 V
C.2. FSMD i2s2 VI
C.3. Diagrama para la cuantificación del vúmetro VII
C.4. Lógica para la elección de ena y wea VIII
C.5. Diagrama para la lecutra y escritura de la memoria RAM IX
C.6. Implementación y cronograma del filtro FIR X
C.7. Lógica interna de las señales parametrizables XI
C.8. FSMD para el control de la saturación XII

D. Tablas para la distribución del Hardware **XIII**
D.1. Tabla de distribución de los botones XIII
D.2. Tabla de distribución de los *switches* XIV

Bibliography **XV**

Índice de figuras

1.1. Arquitectura general del proyecto. 3
1.2. Montaje al completo, visto desde arriba. 5

2.1. Características para la elección del amplificador de pre-amplificación 7
2.2. Amplificador empleado en la etapa de pre-amplificación del prototipo. . . . 8
2.3. Características para la elección del amplificador en la etapa de amplificación. 9
2.4. Amplificador empleado en la etapa de amplificación del prototipo. 10
2.5. Placa Nexys A7 de *Digilent*. 11
2.6. Recursos disponibles dentro de la plataforma Nexys A7. 11
2.7. Ejemplo de una pastilla para guitarra eléctrica. 12
2.8. Circuito electrónico interno de una guitarra eléctrica. 12

3.1. Filtro FIR de tipo peine. 14
3.2. Filtro FIR resultante para el Vibrato. 16
3.3. Filtro IIR de tipo peine. 17
3.4. Filtro IIR resultante para el efecto *chorus* 18
3.5. Filtro reverberador. 19
3.6. Diagrama general para un sistema de *overdrive*. 22
3.7. Diagrama general para un sistema de *compressor*. 23
3.8. Diagrama del efecto *E/S*. 24
3.9. Diagrama del efecto *Looper*. 25
3.10. Diagrama del filtro FIR digital. 25
3.11. Coeficientes del filtro paso bajo. 26
3.12. Respuesta en frecuencia del filtros FIR paso bajo. 26
3.13. Coeficientes del filtro paso alto. 27
3.14. Respuesta en frecuencia del filtros FIR paso alto. 27
3.15. Diagrama general del efecto *WahWah*. 28
3.16. Filtro de estado variable. 28
3.17. Diagrama de un rectificador de media onda. 29

4.1. Funcionamiento del protocolo i2s2. 32
4.2. Conexión de las señales i2s2 generadas en lógica programable hacía el puerto *pmod header JA* de la *Nexys A7*. 33

4.3. Asignación de los registros de salida *i2s2* para el *delay*. 39
4.4. Asignación de los registros de salida *i2s2* para el *vibrato*. 39
4.5. Asignación de los registros de salida *i2s2* para el *chorus*. 39
4.6. Asignación de los registros de salida *i2s2* para el *reverb* no parametrizado. . 40
4.7. Asignación de los registros de salida *i2s2* para el *reverb* parametrizado. . . 40
4.8. Señal de salida auxiliar, ganancias y umbrales para el efecto *compressor*. . . 41
4.9. Utilización de los recursos de la *Nexys A7* tras el proceso de síntesis. 42
4.10. Utilización de los recursos de la *Nexys A7* tras el proceso de implementación. 42

C.1. Arquitectura del *transceiver i2s2*. V
C.2. Máquina de estados finita con ruta de datos para la creación de las señales del protocolo i2s2. VI
C.3. Diagrama para la cuantificación del vúmetro. VII
C.4. Lógica para la elección de *ena* y *wea*. VIII
C.5. Máquina de estados finita con ruta de datos para la gestión de la memoria RAM. IX
C.6. Implementación y cronograma del filtro FIR. X
C.7. FSM para la lógica interna de las señales parametrizables. XII
C.8. FSMD para el control de la saturación de los botones. XII

CAPÍTULO 1

Introducción, estado del arte y motivación

1.1. Introducción y estado del arte

Hoy en día se vive en un mundo donde se ha desarrollado la electrónica digital en el ámbito tecnológico, pero, a su vez y en otro ámbito paralelo, la música también ha experimentado un gran avance. Gracias a ello, la creación de un pedal de efectos es una iniciativa factible ya que se puede lograr de manera eficaz.

Un pedal de efectos es un dispositivo que se encarga de tomar una señal de entrada de audio. Una vez el dispositivo recibe esta señal de audio, deberá aplicar ciertas modificaciones sobre la misma. Normalmente estos cambios suelen ir destinados a variar el timbre, el tono, el volumen o la intensidad (e incluso, todos al mismo tiempo). Fijándose detenidamente, hablando en términos musicales, los efectos del pedal afectan a los cuatro elementos sonoros fundamentales de la música. Por ello se establece una relación directa entre el mundo musical y el mundo del tratamiento digital de audio. Finalmente, el resultado que se deberá obtener será una señal de audio, transformada oportunamente por el efecto seleccionado.

Ahora bien, debido a las características de este tipo de pedales, su uso más común es sobre instrumentos de tipo electrófono. Es decir, aquellos instrumentos que se hacen valer de un sistema electrónico para producir el sonido. Por esta razón, la idea será destinar este pedal al uso de la guitarra eléctrica debido a que es un instrumento muy empleado en la actualidad y que ofrece una gran variedad de posibilidades para sus usuarios más experimentados. De esta manera se pueden lograr distintos sonidos que son capaces de dar gran matiz a las distintas obras musicales que se decida interpretar, ya sea en directo empleando el pedal como un sistema en tiempo real, ya sea en grandes post-producciones para generar arreglos sobre pistas de audio que han sido previamente grabadas por una guitarra eléctrica sin ningún tipo de efecto.

Por tanto, el objetivo perseguido en el proyecto es la creación del pedal multiefectos partiendo de cero, pero siempre atendiendo a que todos los pasos realizados han sido previamente acordados con el profesor - tutor. El trabajo propuesto no estaba ofertado por el Departamento de Ingeniería Electrónica, aunque el tutor asignado, don Pablo Ituero accedió a esta propuesta de trabajo de fin de grado tomando en consideración mis motivaciones personales, mis conocimientos musicales y mi manejo de la guitarra eléctrica.

Para la realización del prototipo emplearemos una arquitectura de FPGA Artix-7 orientada a estudiantes y que se monta sobre la plataforma Nexys A7 [1]. Esta plataforma con la FPGA se puede encontrar en *Xilinx* con la referencia *XC7A100T-1CSG324C (la sección previa al guión hace referencia al tipo de arquitectura que se monta sobre la plataforma y la sección posterior al guión hace referencia a la plataforma que sustenta todo el Hardware necesario para poder emplear dicha arquitectura).*

La elección de esta FPGA y esta plataforma está motivada en que son las que nos puede proporcionar el Departamento de Ingeniería Electrónica para eliminar costes al alumnado ya que este tipo de dispositivos suele tener un coste elevado. Además, ofrece un entorno *Hardware* muy accesible basado en 16 *Switches*, varios diodos *LEDs*, 5 botones o pulsadores, un total de ocho *displays* de 7 segmentos y lo más importante, la incorporación de los *Pmod Headers* que serán esenciales para la interconexión de estos dos elementos: La arquitectura FPGA y la guitarra eléctrica.

Por otra parte, se necesitará una etapa que se encargue de digitalizar nuestra guitarra eléctrica para conseguir llevar la señal de salida desde el instrumento musical electrófono (guitarra eléctrica) hasta la entrada de la FPGA. Para ello se usará el *Pmod i2s2* [2] de *Digilent* el cual también será proporiconado por el Departamento de Ingeniería Electrónica. Este *Pmod* incorpora *ADC*, *DAC* y los conectores mini-jack estándar de audio, que serán de ayuda para gestionar la señal de entrada y salida de nuestra guitarra. Lo único que se deberá añadir serán unos adapatadores de mini-jack a jack-estándar debido a que la guitarra electrica usa entradas de jack-estándar.

Finalmente, para capturar el sonido de la guitarra eléctrica, se empleará una etapa de preamplificación basada en un amplificador de guitarra eléctrica de baja potencia (hasta unos 15W) y, una vez procesado el efecto del pedal en baja potencia, se colocará a la salida otro amplificador de guitarra eléctrica pero de alta potencia (puede ser un amplificador de guitarra eléctrica basado en transistores de 150W, por ejemplo).

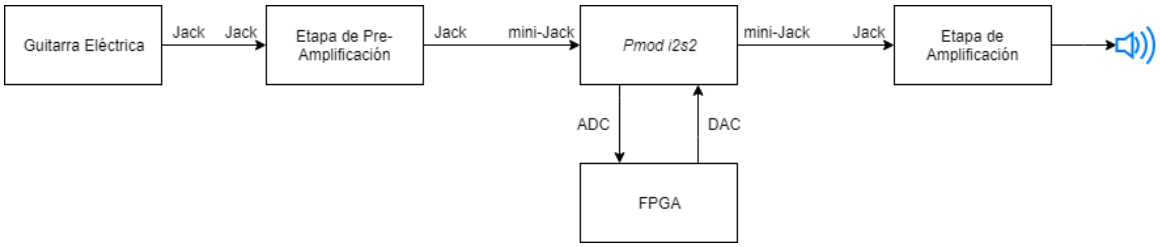


Figura 1.1: Arquitectura general del proyecto.

1.2. Objetivos del proyecto

Los objetivos que este Trabajo Fin de Grado pretende lograr son los siguientes:

- Diseñar la creación de un pedal multiefectos en su totalidad partiendo desde cero y con la finalidad de transformar diferentes sonidos.
- Comprobar y asegurar el correcto funcionamiento de cada uno de los módulos *Hardware* desarrollados a lo largo del proyecto en formato *.vhd*.
- Construir y probar el prototipo completo atendiendo a la arquitectura presentada en la figura 1.1 del punto anterior.
- Consolidar los conocimientos impartidos en los estudios de Grado de Ingeniería en Tecnologías y Servicios de la Telecomunicación en los siguientes temas: tratamiento y procesado de la señal de audio, programación de lenguaje *Hardware* en tiempo real e integración de distintos sub-módulos que engloban entre todos un prototipo funcional.

1.3. Estructura del documento

En este apartado vamos a hacer un recorrido por el desarrollo que se ha seguido para llevar a cabo el prototipo (el pedal de efectos). Para ello comenzaremos hablando del estado del arte y las motivaciones personales en este Capítulo 1.

A continuación, en el Capítulo 2, trataremos la captación de sonido de la guitarra eléctrica y cómo llevar este sonido hasta la FPGA y, una vez allí, cómo reproducirlo con una buena calidad de audio.

Seguidamente, analizaremos los efectos que vamos a implementar dentro del pedal multi-efectos. Por ello, en el Capítulo 3 se llevará a cabo un estudio teórico donde explicaremos la base teórica de todos los efectos desarrollados dentro del proyecto y a qué categoría corresponden (efectos basados en retardadores, efectos basados en sistemas no lineales y efectos basados en sistemas lineales) para luego en el Capítulo 4 explicar cómo hemos implementado estos efectos dentro del lenguaje VHDL junto con el protocolo *i2s2*, necesario para hacer funcionar correctamente el *Pmod i2s2* junto con su *ADC* y *DAC*.

Finalmente, en el capítulo 5 se describirán todas las pruebas llevadas a cabo para comprobar la veracidad, el funcionamiento y la implementación de cada uno de los efectos realizados atendiendo a sus criterios de diseño junto con una breve conclusión final.

1.4. Metodología

La metodología que vamos a llevar a cabo en este trabajo será la siguiente:

- En primer lugar seleccionar la categoría a la que corresponde el efecto que se quiere implementar (efecto basado en retardadores, efecto basado en un sistemas no lineales o efecto basado en un sistemas lineales). Una vez elegida la categoría del efecto que se quiere desarrollar, estudiar sobre el algoritmo que implemente la función del efecto que se desea realizar. Para esto último, será de gran ayuda la referencia [3] de la bibliografía.
- El segundo paso será buscar o desarrollar un diagrama de bloques que permita implementar tanto el efecto seleccionado como las especificaciones que el protocolo *i2s2* requiere para funcionar correctamente.
- En tercer lugar buscar dos amplificadores para la conexión de las etapas de pre-amplificación y amplificación para el montaje final.
- El cuarto paso consistirá en conseguir integrar todos los efectos con sus correspondientes algoritmos en el programa *Vivado* con el lenguaje VHDL. Todos estos efectos deberán ser capaces de convivir entre ellos dentro de la arquitectura desarrollada y se pondrá algún ejemplo de varios efectos funcionando simultáneamente.
- Finalmente, se hará uso de la herramienta de simulación de Vivado para comprobar el correcto funcionamiento de los efectos desarrollados, para ello se utilizará: un *test-bench* avanzado, la herramienta MATLAB y el propio debuging sobre la FPGA (es decir, escuchando con nuestros propios oídos el resultado obtenido).

Así bien, siguiendo todos estos pasos se debe ser capaz de montar el prototipo final entorno a la Nexys A7.

1.5. Resultados esperados

La duración del proyecto ha sido de aproximadamente 6 meses. La idea surgió a finales del mes de Junio de 2019 y se comenzó a llevar a cabo en septiembre de ese mismo año de manera práctica ya que, el mes de Julio fue destinado a estudiar y preparar el proyecto presentado.
Durante este tiempo y teniendo su finalización en Enero de 2020 se puede afirmar que se han alcanzado las diferentes metas acordadas al inicio del proyecto: **se dispone de un prototipo plenamente funcional y probado en salas de ensayo con mi propia guitarra eléctrica y pudiendo comprobar que este mismo cumple con el resultado acordado previamente en Junio de 2019**.

Figura 1.2: Montaje al completo, visto desde arriba.

De esta forma, se puede asegurar que debido al desarrollo llevado acabo, a día de hoy se presenta el pedal con una serie de efectos finitos y previamente seleccionados, pero no habría ningún inconveniente en generar un banco de efectos y cada vez que se quiera usar el pedal, cargar los efectos que el usuario desee de una manera fácil. Todo esto se explicará más adelante cuando se trate de cómo se ha llevado a cabo la programación en el Capítulo 4 del presente documento. Cabe destacar que todo lo que se recogerá en el Capítulo 4 supone aproximadamente el 65 % del total de las horas que se han destinado a este trabajo mientras que el otro 35 % se ha destinado a la compresión de los algoritmos, redacción del documento y pruebas sobre los resultados obtenidos.

Así bien, la estimación de horas dedicadas al proyecto se estima que han sido unas 20 horas por cada una de las semanas. Si se tiene en cuenta que el proyecto se empezó a llevar a cabo de manera efectiva en septiembre de 2019 y que solamente ha habido una semana de descanso en todo este tiempo, se calcula que aproximadamente se han trabajado 18 semanas.

En total, un computo aproximado de 360 horas.

CAPÍTULO 2

Interfaz de sonido externa y FPGA

Como se ha mencionado anteriormente, en este capítulo se tratará todo lo referido a la interconexión del *Hardware* externo que permitirá el correcto funcionamiento del pedal de efectos. Se detallará la elección de los amplificadores y se hablará un poco más a fondo de la plataforma Nexys A7.

2.1. Sistema de entrada/salida del audio

2.1.1. Etapa de pre-amplificación a la entrada

La función primordial que tendrá esta etapa de pre-amplificación será captar en su entrada la señal eléctrica producida por la guitarra eléctrica y en su salida aportar una señal adaptada y válida para ser procesada por el *Pmod i2s2*. Para ello se hará uso de los dos canales jack tipo hembra de los que dispone el amplificador; el canal de entrada conectado a la guitarra eléctrica, y el canal de *Phones* utilizado como medio de salida para conducir la señal acondicionada y adaptada a la entrada del *Pmod i2s2*.

Figura 2.1: Características para la elección del amplificador de pre-amplificación

En una primera instancia se planteó realizar una pequeña etapa de *Hardware* externa donde se pudiera conectar la guitarra a través de un conector tipo jack-XLR, evitando tener que colocar este amplificador, pero rápidamente fue descartada por varios motivos:

- No cumple la característica primordial de este proyecto (tener todo integrado dentro de la FPGA).
- Las posibles implementaciones de este tipo de etapas requieren fuentes de alimentación complejas que rara vez se podrán encontrar en locales de ensayos o salas de conciertos, ya que su uso está más orientado a aplicaciones de laboratorio.
- En congruencia con la razón anterior, se plantea que es mejor emplear un amplificador previo de baja potencia que será más común en los lugares previamente mencionados.

Así bien, el amplificador elegido para esta aplicación será el *Yamaha GA15* que cuenta con una potencia total de 15W. En la referencia [4] se pueden ver las características de este de forma más detalladas.

Figura 2.2: Amplificador empleado en la etapa de pre-amplificación del prototipo.

Fijándose en la figura 2.2 se puede ver que cumple con las características mostradas en la figura 2.1:

- Dispone de la entrada de *Input* general.
- Cuenta con tres controles para modificar el nivel de potencia que tendrán los sonidos agudos, medios y graves que se enviarán a la FPGA y con el controlador de volumen para regular la ganancia de señal de audio que se quiera a la entrada de la FPGA.
- Tiene la salida auxiliar *Phones* la cual enviará la señal acondicionada a la FPGA.
- Es de baja potencia (15W).

Por lo tanto, para finalizar esta sección, se puede llegar a la conclusión siguiente: debido a las especificaciones con las que se desarrollará el prototipo y en los lugares donde se utilizará, es mejor ayudarse de un pequeño amplificador y emplear el conector auxiliar de *Phones* para enviar la señal de audio hasta la FPGA (a pesar de no utilizar el cono amplificador para reproducir el sonido), ya que lo que se hace es **emplear los componentes analógicos internos del amplificador como etapa acondicionadora** siendo esta capaz de alimentarse a través de la red eléctrica común sin necesidad de fuentes de alimentación complejas.
Además, este tipo de amplificadores no suelen ser muy molestos de transportar y para la aplicación necesaría (en este caso, proporcionar la señal adaptada y acondicionada a la FPGA) resultan ser bastante buenos, ya que no trabajan en potencias muy elevadas.

2.1.2. Etapa de amplificación a la salida

La principal función que tendrá esta etapa de amplificación será tomar la señal modificada por el pedal de efectos y reproducirla a un volumen adecuado y óptimo para el entorno en el que se encuentre el pedal multiefectos montado. Para ello se hará uso del canal jack de entrada tipo hembra del que disponga dicho amplificador y, en este mismo canal, se conectará la señal de salida proviniente de la FPGA. Así bien, a diferencia de la sección anterior, ahora sí que se hará uso del cono amplificador para desencapsular el audio contenido dentro de todo el sistema y hacerlo sonar hacia el exterior.

Figura 2.3: Características para la elección del amplificador en la etapa de amplificación.

Así bien, el amplificador elegido para esta aplicación será el *Combo Line 6 Spider III 75W x2*. El motivo de la elección de este se debe a que dispone de dos conos de amplificación y por lo tanto reproduce cada una de las señales que componen el estéreo del audio por cada uno de los conos. En la referencia [5] se pueden ver las características de forma más detallada.

Figura 2.4: Amplificador empleado en la etapa de amplificación del prototipo.

Fijándose en la figura 2.4 se puede ver que cumple con las características mostradas en la figura 2.3:

- Dispone de la entrada de *Input* general.
- Cuenta con el controlador de volumen para regular la magnitud de señal que queremos a la salida.
- Es de alta potencia (75W x2).

Por consiguiente, para finalizar esta sección, se puede llegar a la conclusión siguiente: es **necesario disponer de un amplificador que tenga dos conos de amplificación y sea estéreo**. Si esto último no fuera posible y se tuviera que montar una etapa con solo un cono puede que se lleguen a experimentar ciertos problemas con la calidad del audio debido a que el *Pmod i2s2* trabaja en estéreo.

2.2. FPGA

Tal y como ha sido comentado, la placa que se va a emplear para montar todo el diseño del pedal multiefectos será la plataforma *Nexys A7* de *Digilent*.

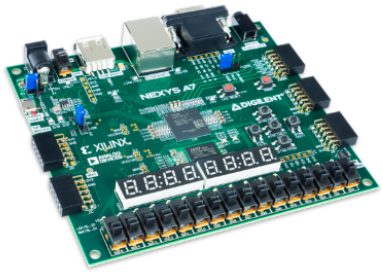


Figura 2.5: Placa Nexys A7 de *Digilent*.

La arquitectura FPGA que incorporá esta plataforma [2.5] es una FPGA *Artix-7*. Además, esta plataforma incorporá gran cantidad de recursos internos dentro de la FPGA.
Para estudiar el número de recursos del que se dispone, se pueden seguir dos métodos:

- Emplear la herramienta Vivado [6] (antes de iniciar el proyecto, Vivado permite seleccionar previamente con qué plataforma y arquitectura FPGA se desea trabajar y por lo tanto podemos ver dentro del programa la cantidad de recursos totales).


Search: xc7a100tcsg324-1 (1 match)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers	GTXE2 Transceivers	GTHE2 Transceivers	GTZE2 Transceivers	GTHE3 Transceivers	GTYE3 Transceivers	GTHE4 Transceivers	GTYE4 Transceivers	PCIe	MMCMs
xc7a100tcsg324-1	324	210	63400	126800	135	0	240	0	0	0	0	0	0	0	0	0	1	6

Figura 2.6: Recursos disponibles dentro de la plataforma Nexys A7.

- Analizar la documentación ofrecidada por *Digilent*. [7]

Así bien, tal como se puede ver en la figura 2.6 o accediendo a la documentación de *Digilent*, los recursos disponibles serán: **63400 LUTs (*Look-up Table*) de las cuales 19000 son de tipo LUTRAMs, 126800 FF (*Flip-Flops*), 135 BRAMs (*Block RAM*), 240 DSPs Slices (*Demand Side Platform*) y 6 MMCMs (*Clock Management Tiles*)**.

2.3. Captación del sonido en la guitarra eléctrica

La guitarra eléctrica, como ya se ha mencionado, es un instrumento electrófono [8] y por lo tanto emplea elementos electrónicos capaces de capturar y producir el sonido del instrumento en cuestión. El elemento electrónico que emplea la guitarra eléctrica, concretamente, es una **pastilla electromagnética** [8] (en la figura 2.7 se puede ver un ejemplo de este tipo de pastillas).

Figura 2.7: Ejemplo de una pastilla para guitarra eléctrica.

Estas pastillas electromagnéticas están formadas por un imán permanente rodeado por un bobinado de alambre de cobre. Por lo tanto, cuando un cuerpo metálico ferromagnético (las cuerdas) se mueve dentro del campo magnético de un imán permanente, se produce una corriente inducida en el bobinado de frecuencia igual a la de la oscilación del objeto y proporcional a la amplitud del movimiento. Esta corriente es muy débil, por ello el cableado interno del instrumento y el cableado hasta la amplificación, debe estar perfectamente apantallado para reducir y evitar en todo los posibles ruidos parásitos. [9]

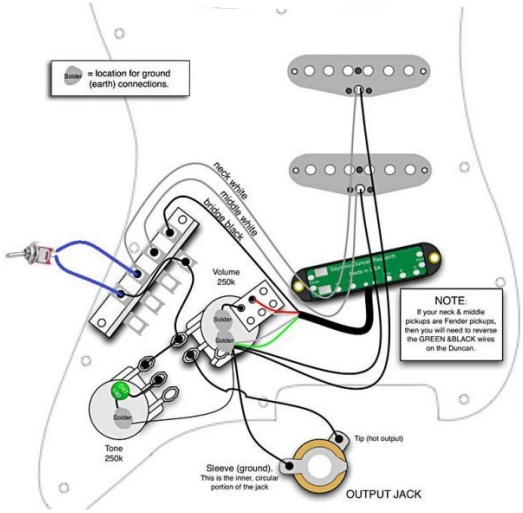


Figura 2.8: Circuito electrónico interno de una guitarra eléctrica.

Fijándose en la figura 2.8 se puede ver el lugar donde iría colocada la pastilla y que la manera de procesar la señal de audio hacia el exterior de la guitarra eléctrica será a través un conector Jack (tal y como se podía ver con anterioridad en la figura 1.1).

CAPÍTULO 3

Análisis teórico de los efectos digitales

El objetivo de este capítulo será exponer el fundamento teórico de cada uno de los efectos estudiados a lo largo de estos meses. El capítulo se segmentará en 2 partes: por un lado, los efectos digitales que se han implementado dentro de la arquitectura FPGA para esta versión del prototipo y por otro lado, los efectos que han sido estudiados pero no han sido incluidos dentro de la arquitectura.

3.1. Teoría sobre los efectos digitales estudiados e implementados

En esta primera parte del capítulo se hará un recorrido por cada una de las categorías a las que puede pertenecer un efecto digital de audio. Estas tres categorías de efectos son las siguientes:

3.1.1. Efectos basados en retardadores

Los efectos basados en retardadores serán aquellos que se encarguen de **añadir al audio original una muestra de este mismo sonido (o varias muestras) un tiempo después del audio original** [10]. Cuando se habla de un efecto retardador, hay que tener en cuenta que la señal demorada se verá afectada por hasta 3 parámetros:

- **Tiempo del retardo** o **línea de retardo**. Este parámetro hará referencia al tiempo que existe entre el audio original y la primera muestra retardada. Este valor se dará en milisegundos.
- **Número de retardos**. Esta variable cuantificará el número de veces que se retarda el sonido original (dicho de otra forma, la cantidad de versiones que habrá a la salida del audio original). El valor estará acotado entre 1 e infinito.
- **Atenuación**. Determina cuánto disminuye la intensidad sonora con respecto a la muestra principal de audio.

Dependiendo del valor que tome cada uno de estos parámetros se definirá un tipo de retardador concreto. Este retardador será el encargado de dar nombre a nuestro efecto digital. A continuación se muestra la lista de los retardadores implementados con su correspondiente explicación:

3.1.1.1. Efecto retardo

Es el efecto más básico dentro de esta categoría. Su retardador esta basado en un único retardo, es decir, el **número de retardos es igual a uno**. La **atenuación suele ser de un medio** y por ello, la muestra retardada se escuchará con la mitad de intensidad que la muestra original (pese a mencionar estos valores, no quiere decir que deba ser siempre así ya que se pueden llegar a aplicar atenuaciones de un cuarto o simplemente mantener el mismo nivel de intensidad sonora entre la muestra retardada y la muestra original y por lo tanto tener una atenuación nula. Así bien, el nivel de atenuación variará con respecto la aplicación que se quiera desarrollar). El **tiempo de retardo suele ser más flexible e indeterminado** puesto que depende de la aplicación para la que se vaya a usar pero siempre tendrá un valor **constante**; por ejemplo si este tiempo es inferior a $10ms$ se puede considerar el retardador como un *doubling* y si, está entre $25ms$ y 50 ms se considera un *slapback*. Más allá de este rango de valores simplemente será un *delay*.

Para la implementación de este retardador se hará uso de un filtro FIR [11]. Sin embargo, para esta aplicación especial se usará concretamente el **filtro FIR de tipo peine**.

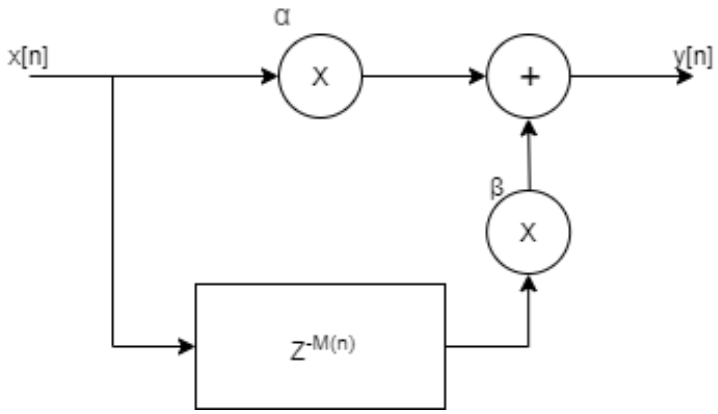


Figura 3.1: Filtro FIR de tipo peine.

Observando la figura 3.1 se deduce su ecuación en diferencias (3.1) donde *M(n)* será la función matemática de la línea de retardo. Además se incluye la ecuación (3.2) para calcular el valor numérico del retardo M entre las muestras donde τ será la constante que definirá el retardo y f_m la frecuencia de muestreo:

$$y[n] = \alpha x[n] + \beta x[n - M(n)] \tag{3.1}$$

$$M = \tau / f_m \tag{3.2}$$

Centrándose en analizar la ecuación en diferencias obtenida a través del filtro y la teoría conocida sobre el retardador a implementar, se observará que α representa la ganancia de la muestra principal del audio y β la ganancia de la muestra retardada del audio. Por lo tanto, **el parámetro de la atenuación se verá reflejado sobre estas dos variables (α y β)** que proporciona el filtro. Los valores elegidos para el prototipo serán $\alpha = 1$ y $\beta = 0.5$.

Por otro lado, analizando la línea de retardo se puede ver que será un valor constante, esto implica que $M(n) = M$. El valor de la constante de retardo elegida para el prototipo será $\tau = 4000$ y dado que la frecuencia de muestreo es de $f_m = 17.640KHz$ (en el capítulo 4 se hablará de cómo se obtiene este valor) el valor numérico del retardo será $M = 226.75ms$ (es decir, la muestra retardada se oirá $226.75ms$ después que la muestra original).

Finalmente, se puede observar que el número de retardos es uno (hay un único lazo con $Z^{-M(n)}$). Esto supone que la señal de salida tendrá un retardo único (como ya se ha comentado anteriormente).

Con todas estas consideraciones, la ecuación en diferencias resultante y el valor numérico del retardo serán los siguientes:

$$y[n] = x[n] + 0{,}5x[n - 0{,}22675] \tag{3.3}$$

$$M = 4000/17{,}640KHz = 226{,}75ms \tag{3.4}$$

3.1.1.2. Efecto vibrador

O mejor conocido como efecto *Vibrato*. Este retardador tiene su fundamento en introducir variaciones casi periódicas en las frecuencias del sonido. La finalidad es conseguir un sonido similar al que se produce cuando en un instrumento de cuerda se hace temblar el dedo que esta presionando la cuerda. El retardador de este efecto está basado en un único retardo, al

igual que lo que ocurría con el efecto *Delay* anterior. Por lo tanto, el **número de retardos es igual a uno**. En este efecto **la atenuación suele ser nula** ya que **la muestra original de audio se elimina**. Finalmente, el **tiempo de retardo debe estar acotado entre $5ms$ y $10ms$**.

Para la implementación de este se hará uso del filtro FIR de la figura 3.1 pero con una serie de modificaciones que se exponen a continuación:

- Dado que la atenuación del camino directo es nula y que la señal retardada no recibe atenuación tendremos los valores siguientes: $\alpha = 0$ y $\beta = 1$.
- En este caso la línea de retardo es variable. Más concretamente, la función matemática que genera este retardo será una señal sinusoidal con una frecuencia de $f_{sinV} = 5.1Hz$ y una amplitud de $A_{sinV} = 2^7$. Esto implica que $\mathrm{M(n)} = 2^7\mathrm{sin}(2\pi 5.1\mathrm{n})$. El valor de la constante de retardo elegida para el prototipo será $\tau = 500$ y como la frecuencia de muestreo es $f_m = 17.640KHz$ el valor numérico del retardo estará acotado entre $M_{max} = 28.34ms$ y $M_{min} = 21.15ms$. Cabe destacar que en el prototipo no se cumplirá que el tiempo de retardo este acotado entre $5ms$ y $10ms$. Se ha tomado la decisión de aumentar estos tiempos para que el efecto sea más notorio. Si se quisiera que estuviera dentro de estas especificaciones lo que habría que hacer sería disminuir τ y A_{sinV} teniendo en cuenta que siempre se debe cumplir que $\tau > A_{sinV}$. Unos valores correctos serían por ejemplo $\tau = 100$ y $A_{sinV} = 2^5$.
- El número de retardos es uno ya que solo hay un lazo con $Z^{-M(n)}$.

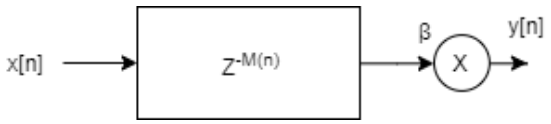


Figura 3.2: Filtro FIR resultante para el Vibrato.

Con todas estas consideraciones, la ecuación en diferencias resultante para el efecto *Vibrato* y el valor numérico del retardo serán los siguientes:

$$y[n] = x[n - (500 - 2^7 sin(2\pi 5{,}1n))/17640] \tag{3.5}$$

$$M_{max} = (500 - 0)/17{,}640KHz = 28{,}34ms \tag{3.6}$$

$$M_{min} = (500 - 127)/17{,}640KHz = 21{,}15ms \tag{3.7}$$

3.1.1.3. Efecto coro

También conocido por su nombre en inglés *chorus*. Este retardador consiste en añadir copias del audio original retardadas entre un intervalo de tiempo acotado con pequeños cambios aleatorios en el tiempo de retardo. La finalidad es conseguir, como el nombre del retardador indica, un coro que acompañe al audio original. El retardador de este efecto está basado en varios retardos. Por lo tanto, el **número de retardos está acotado entre uno e infinito**. **La atenuación suele ser de un medio** pero este valor puede variar dependiendo de la aplicación (tal y como sucedía con el efecto *Delay*). Finalmente, el **tiempo de retardo debe estar acotado entre 10ms y 25ms**.

Para la implementación de este retardador se hará uso de un filtro IIR [11]. Sin embargo, para esta aplicación especial se usará concretamente el **filtro IIR de tipo peine**.

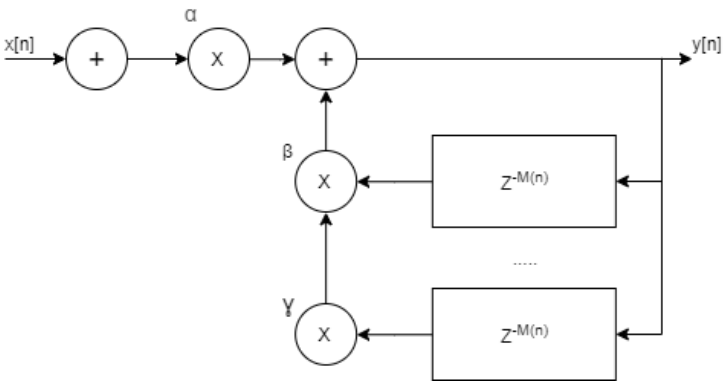


Figura 3.3: Filtro IIR de tipo peine.

Observando la figura 3.3 se deduce su ecuación en diferencias (3.8) donde *M(n)* será la función matemática de la línea de retardo. Además, se incluye la ecuación (3.9) para calcular el valor numérico del retardo M entre las muestras donde τ será la constante que definirá el retardo y f_m la frecuencia de muestreo:

$$y[n] = \alpha x[n] + \beta y[n - M(n)] + ... + \gamma y[n - M(n)] \tag{3.8}$$

$$M = \tau / f_m \tag{3.9}$$

Ahora bien, analizando la ecuación en diferencias resultante del filtro y la teoría expuesta sobre el retardador a diseñar, se observará que α representa la ganancia de la muestra principal del audio y β, γ, ... representan la ganancia de la muestra retardada del audio de

salida que realimenta al filtro. Por lo tanto, **el parámetro de la atenuación se verá reflejado sobre estas variables (α, β, γ, ...)**. Los valores elegidos para el prototipo serán $\alpha = 1$, $\beta = 0.5$, $\gamma = 0$.

Siguiendo con el análisis, la función matemática que modulará este retardo será una señal **aleatoria** con un filtrado de tipo paso bajo. Esto implica que M(n) = *Random Signal + LPF*. **Para nuestro prototipo vamos a emplear directamente una señal sinusoidal pero aleatoria**, es decir, se generará la señal sinusoidal pero los valores que se tomarán para la línea de retardo serán aleatorios. De esta forma se logrará aproximar de manera práctica lo que se pretende hacer con el filtrado paso bajo, ya que lo que se busca con este tipo de filtrado es suavizar los picos de las señales aleatorias para evitar saturar la realimentación. La señal senoidal que se utilizará como modulación aleatoria para el prototipo dispondrá de las siguientes características: $f_{sinC} = 2.55Hz$ y $A_{sinC} = 2^7$. El valor de la constante de retardo elegida para el prototipo será $\tau = 1000$ y como $f_m = 17.640KHz$ el valor numérico del retardo estará acotado entre $M_{max} = 56.69ms$ y $M_{min} = 49.49ms$ (cabe destacar que ocurre lo mismo que lo que sucedía con el efecto *Vibrato*, se ha decidido aumentar estos valores para que el resultado del retardador sea más notable. Si se desease que estuviera dentro de las especificaciones, hay que disminuir τ y A_{sinC} cumpliendo que $\tau > A_{sinC}$. Unos valores acordes a la especificación serían por ejemplo $\tau = 425$ y $A_{sinC} = 2^7$.).

El número de retardos es uno ya que solo hay un lazo con $Z^{-M(n)}$ (debido a que solamente hemos dado valor distinto de cero a α y β).

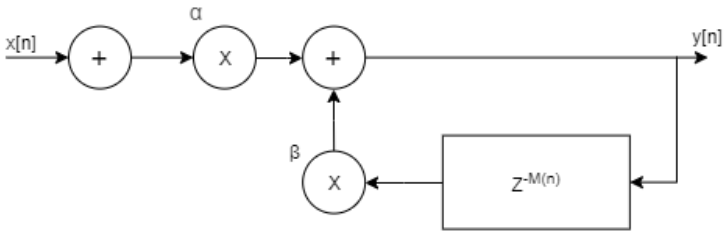


Figura 3.4: Filtro IIR resultante para el efecto *chorus*

Con todas las consideraciones planteadas, la ecuación en diferencias resultante para el efecto *Chorus* y el valor numérico del retardo serán los siguientes:

$$y[n] = x[n] + 0{,}5y[n - (1000 - 2^7 sin(2\pi 2{,}55n))/17640] \tag{3.10}$$

$$M_{max} = (1000 - 0)/17{,}640KHz = 56{,}69ms \tag{3.11}$$

$$M_{min} = (1000 - 127)/17{,}640KHz = 49{,}49ms \tag{3.12}$$

3.1.1.4. Efecto reverberador

El efecto *Reverb* tiene como objetivo recrear la reverberación natural generada por las reflexiones acústicas que el oyente percibe cuando el sonido original todavía está reproduciéndose. Este efecto estará formado por múltiples versiones retardadas del audio original. Por lo tanto el **número de retardos será infinito** y **el tiempo de retardo debe ser menor a 50*ms*** ya que para tiempos superiores nos estaremos acercando más a un efecto de eco (este tipo de efecto se explicará en la sección 3.1.1.5). Las atenuaciones serán de gran importancia en este efecto ya que, dependiendo de la atenuación que reciban los retardadores, se lograrán reverberaciones más "profundas". Por consiguiente, **no hay un valor concreto para la atenuación de este efecto**.

Para la implementación de este retardador se hará uso de un filtro reverberador [12]. Este tipo de filtro cuenta con una combinación de filtro peine FIR e IIR:

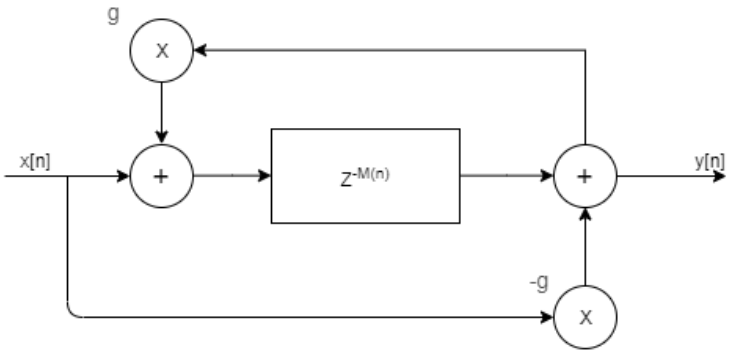


Figura 3.5: Filtro reverberador.

Con este tipo de filtros 3.5 se logrará crear reverberaciones artificiales en el audio de entrada y en consecuencia se puede afirmar que el efecto de reverberación aportará calidez al sonido.

Observando la figura 3.5 se deduce su ecuación en diferencias (3.13) donde *M(n)* será la función matemática de la línea de retardo. Además, se incluye la ecuación (3.14) para calcular el valor numérico del retardo M entre las muestras donde τ será la constante que definirá el retardo y f_m la frecuencia de muestreo:

$$y[n] = -gx[n] + x[n - M(n)] + gy[n - M(n)] \tag{3.13}$$

$$M = \tau / f_m \tag{3.14}$$

Para el prototipo se ha decidido implementar dos versiones de este efecto *Reverb*:

- **Una versión no parametrizable** con unos parámetros de $g = 0.5$, $\tau = 500$, M(n) $= M$, $f_m = 17.640KHz$ y $M = 28.34ms$. El resultado de la ecuación en diferencias para este caso será la siguiente:

$$y[n] = -0{,}5x[n] + x[n - 0{,}02834] + 0{,}5y[n - 0{,}02834] \tag{3.15}$$

$$M = 500/17{,}640KHz = 28{,}34ms \tag{3.16}$$

- **Una versión parametrizable** a través del *Hardware* que nos ofrece la plataforma *Nexys A7* para modificar los diferentes parámetros del filtro (línea de retardo, ganancia de la señal de audio a la entrada, ganancia de la señal retardada de entrada y finalmente la ganancia de la señal retardada de salida que realimenta el sistema). En el capítulo 4 se describirá con más detalle cómo se ha llevado a cabo todo este proceso de parametrización para las variables *i, j, k, l*. La ecuación en diferencias resultante para esta versión del reverberador será la siguiente: ($\tau = 1500$, M(n) $= M_l$ y $f_m = 17.640KHz$)

$$y[n] = -g_{1i}x[n] + g_{3j}x[n - M_l] + g_{2k}y[n - M_l] \tag{3.17}$$

$$M1 = (1500 - 0)/17{,}640KHz = 85{,}03ms \tag{3.18}$$

$$M2 = (1500 - 500)/17{,}640KHz = 56{,}69ms \tag{3.19}$$

$$M3 = (1500 - 1000)/17{,}640KHz = 28{,}34ms \tag{3.20}$$

$$M4 = (1500 - 1499)/17{,}640KHz = 5{,}67ms \tag{3.21}$$

$$g11 = 1; g12 = 0{,}5; g13 = 0{,}33; g14 = 0{,}25; \tag{3.22}$$

$$g31 = 1; g32 = 0{,}5; g33 = 0{,}33; g34 = 0{,}25; \tag{3.23}$$

$$g21 = 0{,}5; g22 = 0{,}25; g23 = 0{,}125; g24 = 0{,}0625; \tag{3.24}$$

3.1.1.5. Efecto eco

El efecto eco está basado en añadir al audio de entrada versiones múltiples retardadas y atenuadas, es decir, intenta imitar el eco original que genera la naturaleza. Este efecto se puede generar de varias formas pero su característica fundamental es que el **tiempo de retardo debe ser superior a 50*ms***. Una de las maneras más habituales de crear este

efecto es empleando un filtro IIR, como el de la figura 3.3, pero si se realiza de esta forma quedará poco consistente ya que las señales que realimentan el filtro se atenuaran rápidamente. Por este motivo, es mejor realizar el eco a través de un filtro reverberador como el presentado en la figura 3.5 ya que empleando este tipo de filtrado se añade un refuerzo con las señales de entrada retardadas y se pueden conseguir ecos más "profundos".

La ecuación en diferencias que definirá el comportamiento de este efecto será la ecuación correspondiente al filtro de la figura 3.5 (3.13). Para el prototipo se emplearán los parámetros que se han definido en la sección 3.1.1.4 pero cambiaremos el valor de la constante de retardo. Anteriormente tomaba un valor de $\tau = 500$ y ahora pasará a tomar un valor de $\underline{\tau = 5000}$. Esto implica que el tiempo de retardo será de $\underline{M = 283.44ms}$. La ecuación en diferencias resultante implementada en el prototipo será la siguiente:

$$y[n] = -0{,}5x[n] + x[n - 0{,}28344] + 0{,}5y[n - 0{,}28344] \tag{3.25}$$

$$M = 5000/17{,}640KHz = 283{,}44ms \tag{3.26}$$

3.1.2. Efectos basados en sistemas no lineales

Los efectos basados en sistemas no lineales **añadirán modificaciones no lineales a la señal de audio de salida**. En esta categoría de efectos se verá que, a partir de la señal de audio de entrada y un sistema auxiliar no lineal que modifique esta señal de audio de entrada, obtendremos una señal de audio de salida que ha sufrido estos efectos no lineales. Dependiendo de la función matemática que tome cada uno de estos sistemas auxiliares se definirá un tipo de efecto concreto que afectará a la forma de onda de nuestro audio mediante unos **umbrales de ganancia**. A continuación se muestra la lista de los sistemas no lineales implementados con su correspondiente explicación:

3.1.2.1. Efecto sobre-saturación

El efecto sobre-saturación u *overdrive* consiste en establecer un umbral de saturación positivo y un umbral de saturación negativo. Cuando la señal de audio de entrada pasa por este sistema no lineal pueden ocurrir dos cosas:

- Si la **señal de entrada cuenta con una ganancia inferior a los umbrales de saturación** ($|x[n]| < |h[n]|$), la señal de entrada pasa directamente a la salida sin sufrir ningun tipo de modificación y se puede decir que se mantiene dentro del régimen sistema del lineal auxiliar.

$$y[n] = x[n] \tag{3.27}$$

- Si la **señal de entrada supera los umbrales de saturación** ($|x[n]| \geq |h[n]|$), el valor que toma la señal de audio de salida será la correspondiente al umbral que se haya establecido.

$$y[n] = h[n] \tag{3.28}$$

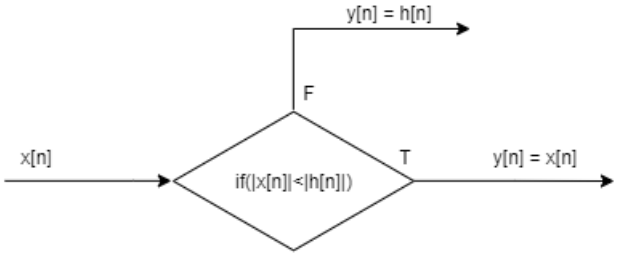


Figura 3.6: Diagrama general para un sistema de *overdrive.*

Para el prototipo desarrollado se hará uso de un umbral de saturación postivo $h_{pos}[n] = 0{,}375$ y un umbral de saturación negativo $h_{neg}[n] = -0{,}25$. Hay que tener en cuenta que estos umbrales están normalizados a un formato $< 1, 15 >$ donde se dispondrá de 16 bits, un bit de signo y 15 bits para la parte decimal. En el Capítulo 4 se hablará con más detalle sobre esta normalización de los valores y de dónde proviene.

Las ecuaciones que definirán el comportamiento de este efecto (tal y como se puede ver en la figura 3.6) para nuestro prototipo serán las siguientes:

$$if(x[n] \geq 0{,}375) \implies y[n] = 0{,}375 \tag{3.29}$$

$$if(-0{,}25 < x[n] < 0{,}375) \implies y[n] = x[n] \tag{3.30}$$

$$if(x[n] \leq -0{,}25) \implies y[n] = -0{,}25 \tag{3.31}$$

3.1.2.2. Efecto compresor

O también conocido como *compressor*. El objetivo de este sistema es atenuar las muestras de audio que superen cierto nivel de ganancia y aquellas que no superen cierto umbral mantenerlas de manera lineal a la salida del sistema. En definitiva, este efecto lo que busca es crear una **compensación en todo el rango de ganancias que se puedan encontrar a la entrada del sistema**.

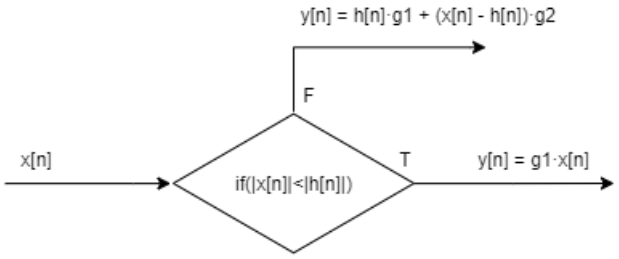


Figura 3.7: Diagrama general para un sistema de *compressor*.

Una condición que se debe cumplir es que **la ganancia de la zona no lineal** $g2$ (o zona superior al umbral de ganancia$\equiv h[n]$) **deberá ser menor que la ganancia de la zona lineal** $g1$. Esta condición se puede resumir como $g1 > g2$.

Para el prototipo, los valores de las ganancias que se utilizarán y el umbral empleado serán los siguientes: $g1 = 0{,}625$, $g2 = 0{,}125$, $h_{pos}[n] = 0{,}75$, $h_{neg}[n] = -0{,}75$

Finalmente, las ecuaciones que modelarán el comportamiento de este efecto compresor dentro del prototipo (tal y como se puede ver en la figura 3.7) serán las siguientes:

$$if(x[n] \geq h_{pos}) \implies y[n] = h_{pos}[n]g_1 + (x[n] - h_{pos})g_2 = 0{,}46875 + (x[n] - 0{,}75)0{,}125 \quad (3.32)$$

$$if(0 < x[n] < h_{pos}) \implies y[n] = g1x[n] = 0{,}675x[n] \quad (3.33)$$

$$if(h_{neg} < x[n] \leq 0) \implies y[n] = g1x[n] = 0{,}675x[n] \quad (3.34)$$

$$if(x[n] \leq h_{neg}) \implies y[n] = h_{neg}[n]g_1 + (x[n] - h_{neg})g_2 = 0{,}46875 + (x[n] + 0{,}75)0{,}125 \quad (3.35)$$

3.1.3. Efectos basados en sistemas lineales

Los efectos basados en sistemas lineales serán aquellos que se encarguen de **añadir modificaciones lineales en la señal de audio de salida**. Dentro de esta categoría de efectos lo que se encontrará serán sistemas auxiliares de tipo lineal que se encargarán de añadir modificaciones a la señal de audio de salida.

3.1.3.1. Efecto entrada/salida

Pese a parecer irreleante, en el mundo de la música existe el efecto entrada-salida. Este efecto consiste en reproducir la señal de entrada a la salida sin ningún tipo de modificación. Para su implementación en el prototipo lo que se deberá hacer es realizar la interconexión entre las señales de entrada y salida. Este efecto será de gran ayuda para conseguir probar

el correcto funcionamiento del protocolo *i2s2* ya que permitirá asegurar que los conversores estén perfectamente configurados.

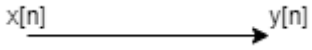


Figura 3.8: Diagrama del efecto *E/S*.

Observando la figura 3.8 podemos ver que en este caso particular el sistema lineal auxiliar equivale a una interconexión directa entre las señales x[n] e y[n]. Por lo tanto la ecuación en diferencias que se puede deducir es la siguiente:

$$y[n] = x[n] \tag{3.36}$$

3.1.3.2. Efecto looper

El efecto *Looper* es un efecto muy reclamado a día de hoy. El motivo de que este efecto haya tomado fuerza en los últimos años es que permite grabar la señal de entrada en un instante de tiempo predeterminado y posteriormente reproducir a tiempo real las pistas que han sido grabadas mientras se superpone por encima otra señal de audio diferente a la que se ha grabado. De manera más técnica se puede decir que el fundamento de este efecto es ser capaz de solapar en la señal de salida la señal de entrada del audio a tiempo real y la señal de audio que ha sido previamente grabada y almacenada. **El sistema lineal auxiliar que emplea este efecto, habitualmente, es un controlador de memoria que permite trabajar con la grabación y reproducción de las diversas señales de entrada de audio que lleguen al sistema**.

Los pedales que incorporan efectos *Looper* más sofisticados suelen emplear dos tipos de memorias: RAM y ROM de manera simultánea (se utiliza la memoria RAM para el momento de la grabación y reproducción y posteriormente se almacena en una memoria ROM para dejar libre esta RAM y poder seguir tomando nuevas muestras) o solamente la memoria RAM. Para nuestro prototipo vamos a emplear solamente una memoria RAM debido a la dificultad que requiere el desarrollo del sistema lineal auxiliar.

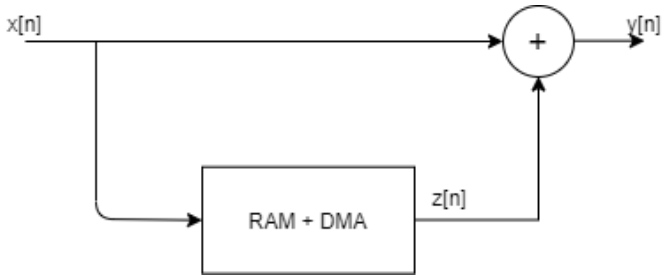


Figura 3.9: Diagrama del efecto *Looper*.

Observando la figura 3.9 se deduce su ecuación en diferencias (3.28):

$$y[n] = x[n] + z[n] \tag{3.37}$$

Para el prototipo desarrollado se utilizará una memoria **RAM de puerto simple con un profundidad de palabra de 2^{19} bits y un tamaño de palabra de entrada y salida de 2^8 bits**. Con estas características en la memoria RAM y sabiendo que $f_m = 17.640KHz$ podemos ver que el tiempo que podrá grabar audio nuestro *Looper* será el siguiente:

$$T_{grabacion} = 2^{19}/17{,}640KHz = 29{,}72s \simeq 30s \tag{3.38}$$

3.1.3.3. Efecto banco de filtros

Tal y como el propio nombre indica, la idea de este efecto es crear un banco de filtros. Es decir, crear una **base de datos donde haya coeficientes normalizados que permitan recrear la estructura de un filtro FIR digital**. Estos bancos de filtros se suelen emplear para ecualizar la señal de salida del audio. Para llevar a cabo estos filtros se debe crear una estructura FIR de N etapas como la que se muestra en la figura 3.10:

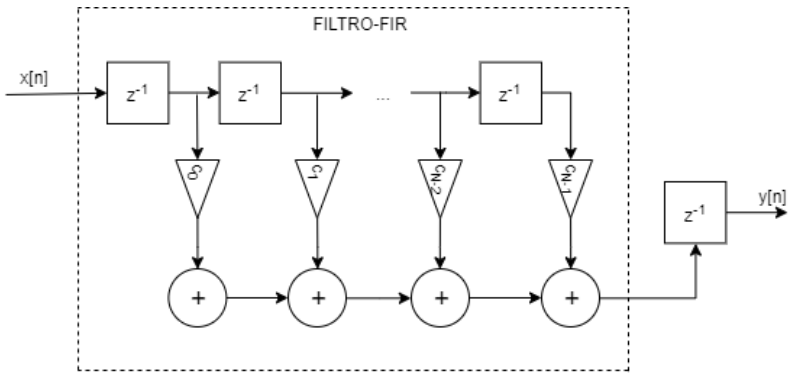


Figura 3.10: Diagrama del filtro FIR digital.

Observando la figura 3.10 podemos deducir que la función de transferencia del filtro FIR que se desea implementar tendrá la siguiente función de transferencia:

$$H_{FIR} = \sum_{n=0}^{N-1} C_n Z^{-n} \tag{3.39}$$

Para el filtro que se diseñará en el prototipo se hará uso de un total de **16 coeficientes**, es decir, $N = 16$. Los filtros que se podran hayar dentro del banco implementado dentro del prototipo serán dos [13]:

- Un **filtro FIR paso bajo simétrico con frecuencia de inicio a 200*Hz* y una banda de transición a 1.764*KHz* con una atenuación de -26.02dB**. Los coeficientes para este filtro serán los siguientes: $c_0 = c_{15}$ = -0.001, $c_1 = c_{14} = 0.023$, $c_2 = c_{13} = 0.047$, $c_3 = c_{12} = 0.053$, $c_4 = c_{11} = 0.08$, $c_5 = c_{10} = 0.094$, $c_6 = c_9 = 0.11$, $c_7 = c_8 = 0.116$.

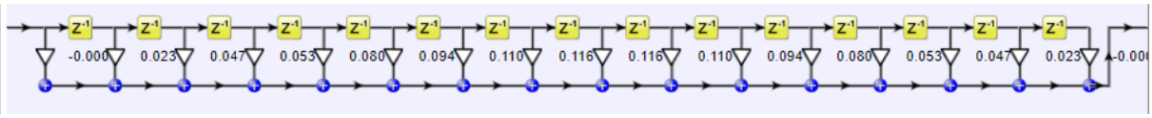


Figura 3.11: Coeficientes del filtro paso bajo.

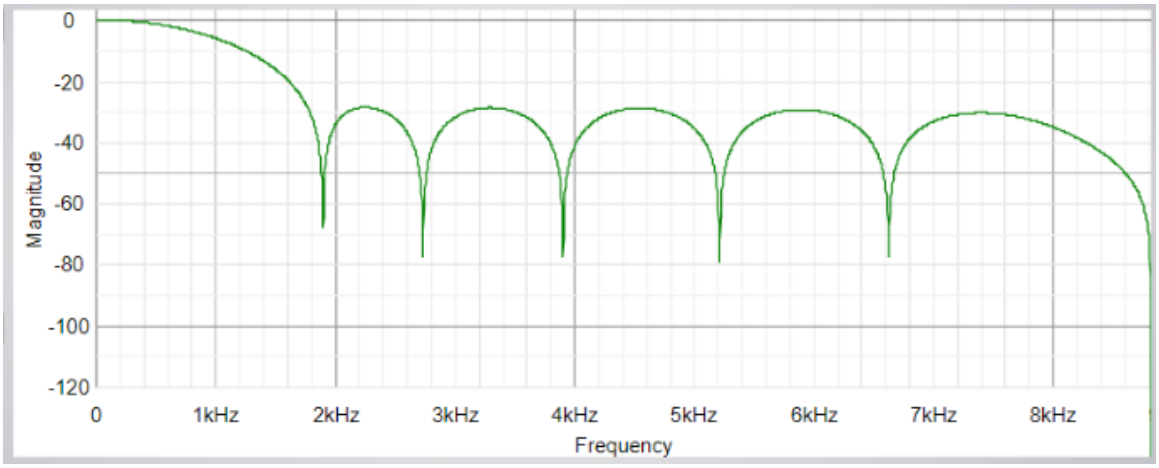


Figura 3.12: Respuesta en frecuencia del filtros FIR paso bajo.

- Un **filtro FIR paso alto simétrico con frecuencia de corte a 2*KHz* y una banda de transición a 1.5*KHz* con una atenuación de -26.02dB**. Los coeficientes para este filtro serán los siguientes: $c_0 = c_{15} = 0.01$, $c_1 = c_{14} = 0.023$, $c_2 = c_{13}$ = -0.097, $c_3 = c_{12}$ = -0.003, $c_4 = c_{11}$ = -0.167, $c_5 = c_{10} = 0.015$, $c_6 = c_9$ = -0.33, $c_7 = c_8 = 0.504$.

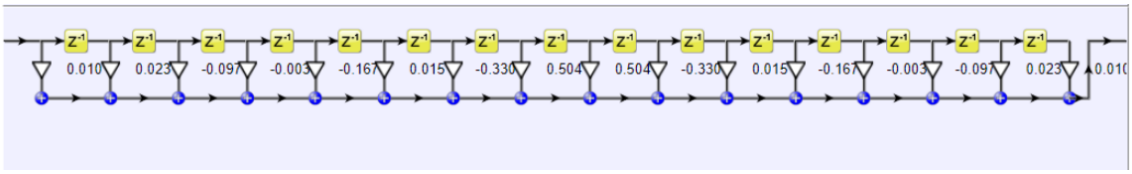


Figura 3.13: Coeficientes del filtro paso alto.

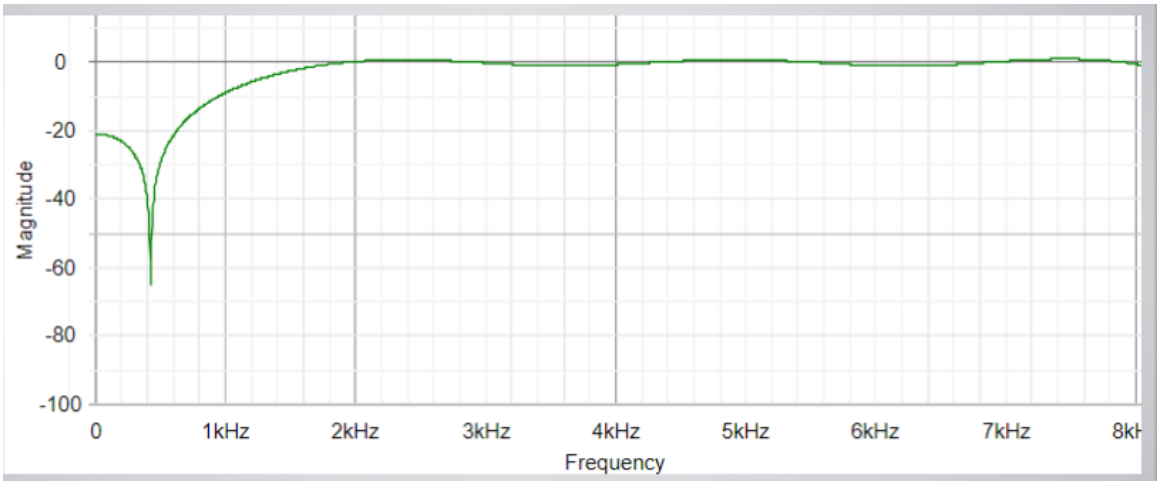


Figura 3.14: Respuesta en frecuencia del filtros FIR paso alto.

3.2. Efectos digitales estudiados pero no implementados

En esta segunda parte del capítulo se hará un recorrido por los efectos que se han estudiado a lo largo de estos meses. Sin embargo, debido a la ausencia de recursos, tiempo y debido a otras dificultades no han podido llegar a verse implementados dentro del prototipo. Pese a ello, en un futuro se podrán ver implementados dentro del prototipo ya que se seguirá trabajando sobre este mismo. Por estos motivos expuestos, los dos efectos que se encontrarán en las secciones 3.2.1 y 3.2.2 se considerarán como mejoras inmediatas a implementar en las siguientes versiones del prototipo:

3.2.1. Efecto autowah

El efecto *WahWah* es un efecto de la categoría "Efectos basados en sistemas lineales"3.1.3. Este efecto se puede lograr utilizando filtros paso banda ya que la idea es aplicar a la señal de audio de entrada un filtro paso banda con una frecuencia central variable en el tiempo y con un ancho de banda estrecho.

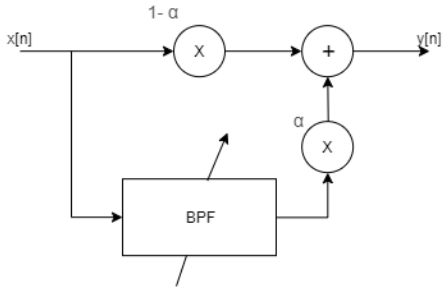


Figura 3.15: Diagrama general del efecto *WahWah*.

Normalmente, para variar esta frecuencia central se emplean sistemas *Hardware* externos (tales como potenciómetros). Sin embargo, para la implementación de este efecto se entra en conflicto con la especificación de que todo debe estar integrado dentro de la *Nexys A7* y por ello en vez de realizar un *WahWah* se llevará a cabo un *autoWah*. El fundamento teórico es el mismo pero ahora la frecuencia de corte del filtro variará de manera periódica en el tiempo a través de un sistema lineal auxiliar. Por tanto, la idea de este sistema lineal auxiliar será **recrear un oscilador de baja frecuencia que se encargue de generar variaciones de la frecuencia central de manera periódica y automática**.

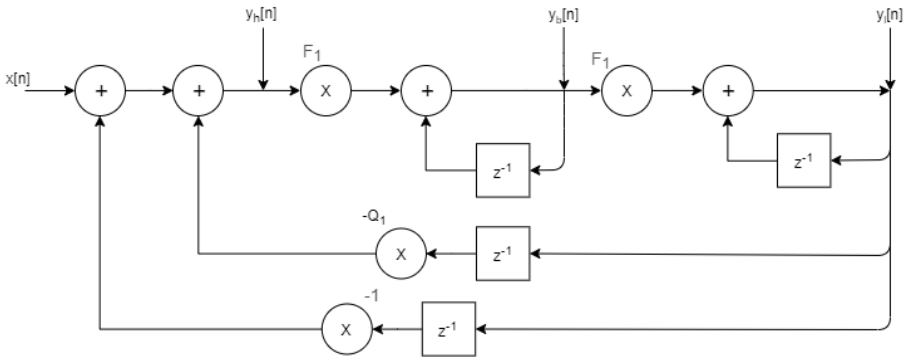


Figura 3.16: Filtro de estado variable.

Para la implementación del *autoWah* haremos uso de un filtro de estado variable [14] como el que se puede ver en la figura 3.16 donde *x[n]* hace referencia a la señal de audio de entrada, y_h corresponde a la señal de salida paso alto, y_b será la señal de salida paso banda y y_b se asociará a la señal de salida paso bajo, f_c es la frecuencia de corte del filtro, f_s la frecuencia de muestreo del sistema global y finalmente, d es un factor constante conocido como *damping*. Sus ecuaciones en diferencias se muestran a continuación:

$$y_l[n] = F_1 y_b + y_l[n-1] \tag{3.40}$$

$$y_b[n] = F_1 y_h + y_b[n-1] \tag{3.41}$$

$$y_h[n] = x[n] - y_l[n-1] - Q_1 y_b[n-1] \tag{3.42}$$

$$F_1 = 2sin(\pi f_c / f_s) \tag{3.43}$$

$$Q_1 = 2d \tag{3.44}$$

3.2.2. Efecto octavador

El efecto octavador consiste en **incorporar al sonido original de la entrada de audio armónicos de los tonos originales** simulando que en la salida suena de manera simultánea la señal de entrada junto con el tono octavado generado. En la salida por lo tanto ocurrirá un efecto similar a lo que ocurría cuando se hablaba del efecto *looper* 3.1.3.2, pero ahora la señal auxiliar que se superpondrá será un armónico de la señal de entrada. Para realizar este efecto existen diversas formas, una de ellas es intentar aproximar de manera digital este efecto al *vocoder* [15]. Otra posible implementación sería emplear rectificadores de media onda.

Cuando se proceda a implementar este tipo de efecto lo más seguro es que se realice empleando rectificadores de media onda. Estos rectificadores consisten en un sistema no lineal en el que **a la salida se tendrá solamente la parte positiva de la señal de entrada**, es decir, si la señal de entrada *x[n]* es positiva se realiza una interconexión directa con la señal de salida *y[n]* y si la señal de entrada es menor o igual que cero (x[n] es negativa) directamente el valor de la señal de salida será cero. Esta teoría se puede plasmar sobre la siguiente función matemática (3.45)(3.46):

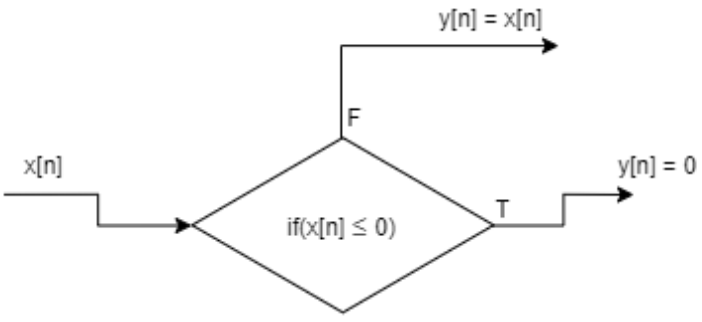


Figura 3.17: Diagrama de un rectificador de media onda.

$$if(x[n] \leq 0) \implies y[n] = 0 \tag{3.45}$$

$$else \implies y[n] = x[n] \tag{3.46}$$

Observando detenidamente esta función matemática y teniendo en cuenta que las señales de audio suelen tener forma similar a una sinusoide, **se puede afirmar que el espectro de salida no solo contendrá la frecuencia de entrada, sino que además también aparecerán todas las frecuencias armónicas de orden par**.

CAPÍTULO 4

Implementación digital de los efectos

En el capítulo presente se hablará de cómo se ha implementado cada uno de los efectos presentados a lo largo del capítulo 3. Para ello será necesario configurar el *Pmod i2s2* (habrá que integrar el protocolo i2s2 dentro de nuestro prototipo). Una vez configurado el protocolo, se pasará a hacer una estimación de los recursos *Hardware* que se necesitarán para conseguir implementar todos los efectos y facilitar el uso del prototipo. Para finalizar, se hablará de cómo se han implementado en lenguaje VHDL el algoritmo de cada uno de los efectos detallados en la sección 3.1. Todo el código desarrollado en lenguaje VHDL se puede encontrar en el siguiente enlace de GitHub: https://github.com/ErosHeavy666/Multi-effect-pedal-in-FPGA-with-i2s-protocol.

4.1. Protocolo i2s2 (Arquitectura i2s2)

Este protocolo es un estándar eléctrico de bus serial para interconectar circuitos de audio digital. La característica más importante de este protocolo es que es capaz de separar las señales de datos y de reloj y, por lo tanto, el *jitter* es menor que con respecto a otros protocolos. El protocolo consiste en, al menos, tres líneas [16] (las cuales debemos generar dentro de la FPGA e implementar en lenguaje VHDL):

- Reloj de bit (*SCLK*).
- Reloj de palabra (*WS o LRCLK*).
- Línea de datos multiplexados a la entrada (*SDATA*).

De manera opcional se puede añadir también:

- Reloj maestro (*MCLK*).
- *Enables* (*en_rx* y *en_tx*).
- Línea de datos multiplexados a la salida (*SDOUT*)

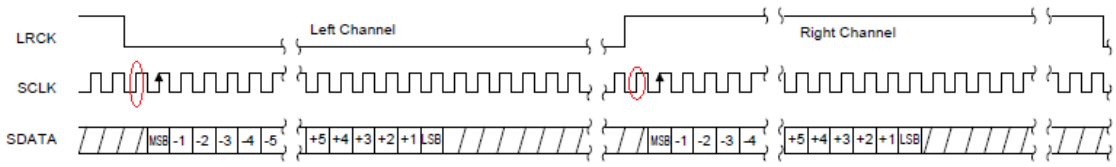


Figura 4.1: Funcionamiento del protocolo i2s2.

Para comprender cómo funcionan los relojes y las líneas de datos que componen el protocolo lo mejor será analizar el diagrama que se puede ver en la figura 4.1:

- **El reloj serie se encarga de muestrear la línea de datos multiplexados de entrada**.
- **Los bits muestreados se guardan en dos registros asociados al reloj de palabra, estos registros tiene un tamaño de *d_width***.
- **Una vez lleno el registro, se conmuta el reloj de palabra y se procede a llenar el otro registro siguiendo el mismo procedimiento anterior**.

Ahora bien, para nuestro protocolo a implementar dentro de la arquitectura FPGA se hará uso de los tres relojes (*MCLK*, *SCLK* y *WS*), *enables* de entrada y salida (*en_rx* y *en_tx*) y la línea de datos multiplexados de entrada y salida (*SDATA* y *SDOUT*). **El tamaño de los registros será de 16 bits normalizados a un bit de signo y quince bits para la parte decimal** $< 1, 15 >$ **(*d_width* = 16 bits)**. El rango de valores con el que se trabajará será <**-1 , 0.999969482421875**> y la resolución de nuestro protocolo será de $\mathbf{2^{-15}}$.

Los valores numéricos para generar cada una de las señales serán los siguientes:

- El reloj maestro será de $11.29MHz$ [2]. Este se obtendrá a partir de un PLL que se generará con la interfaz *IP* de Vivado [17]. Este PLL tomará a la entrada el reloj interno de 100 MHz de la FPGA y generará un reloj a la salida de 11.29MHz.
- El reloj serie será de $1.41125MHz$. Este reloj **no** se puede generar a partir de la *IP* que ofrece Vivado y, por lo tanto, se deberá generar un contador a partir de *MCLK* que nos permitá recrear esta frecuencia de reloj. Observando la documentación del *Pmod i2s2* se puede ver que para un tamaño de palabra en los registros de $d_{width} = 16$ se requiere que *SCLK* sea *MCLK/8*.

- El reloj de palabra será de $17.64KHz$. Con este reloj ocurre lo mismo que lo que sucedía con *SCLK*, no se puede generar a través de la *IP* de Vivado y, por lo tanto, habrá que generarlo a partir de *MCLK* usando un contador. Observando la documentación del *Pmod i2s2* se puede ver que para un tamaño de palabra en los registros de $d_{width} = 16$ se requiere que *WS* sea *MCLK/64*. **Este reloj será el que defina la frecuencia de muestreo de nuestro sistema global**.
- El enable de entrada se pondrá activo a nivel alto durante un ciclo de reloj de *SCLK* cada vez que *WS* complete un ciclo completo. **Cuando en_rx = 1 los registros de *WS* que contienen la señal muestreada pasarán a la arquitectura que contiene todos los efectos digitales**.

En el apéndice C se pueden encontrar dos imágenes: una de ellas representa la arquitectura del *transceiver i2s2* (C.1) implementado para el prototipo y la otra un diagrma FSMD (C.2) que indica cómo se ha desarrollado la lógica programable de dicho *transceiver*.

Para concluir este apartado, se deberán llevar las señales del *transceiver* al puerto del *pmod header JA* que posee la *Nexys A7*. Para ello lo mejor será recurrir a la documentación de *pmod i2s2* de *Digilent* [18]. El resultado de esta interconexión se deberá realizar en el fichero de asociaciones de la *Nexys A7* dentro de *Vivado* tal y como se muestra en la figura 4.2.

```
set_property -dict {PACKAGE_PIN C17 IOSTANDARD LVCMOS33} [get_ports mclk[1]]
set_property -dict {PACKAGE_PIN D18 IOSTANDARD LVCMOS33} [get_ports ws[1]]
set_property -dict {PACKAGE_PIN E18 IOSTANDARD LVCMOS33} [get_ports sclk[1]]
set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports sd_out]
set_property -dict {PACKAGE_PIN D17 IOSTANDARD LVCMOS33} [get_ports mclk[0]]
set_property -dict {PACKAGE_PIN E17 IOSTANDARD LVCMOS33} [get_ports ws[0]]
set_property -dict {PACKAGE_PIN F18 IOSTANDARD LVCMOS33} [get_ports sclk[0]]
set_property -dict {PACKAGE_PIN G18 IOSTANDARD LVCMOS33} [get_ports sd_in]
```

Figura 4.2: Conexión de las señales i2s2 generadas en lógica programable hacía el puerto *pmod header JA* de la *Nexys A7*.

4.2. Arquitectura desarrollada para la FPGA y distribución del hardware

En este apartado se hablará muy brevemente de la función que tiene cada uno de los dieciséis *switches* y los cinco botones de la *Nexys A7*. Se debe tener en cuenta que el sistema implementado dentro de la FPGA cuenta con 4 arquitecturas distintas:

- Arquitectura global (engloba todas las siguientes).
- Arquitectura *i2s2* (ya ha sido explicada en el apartado 4.1).
- Arquitectura *LED/Displays* (se hablará de ella en el apartado 4.3).
- Arquitectura de los efectos digitales diseñados e implementados (se explicará en la sección 4.4).

Ahora bien, de estos dieciséis *switches* se destinarán tres de ellos a elementos de control del sistema (ya sea para la arquitectura global, para la arquitectura de los efectos o para la arquitectura del *transceiver*). Y los otros trece *switches* resultantes se emplearán para el control de los efectos (dentro de la arquitectura que almacena cada uno de los efectos implementados). En el caso de los botones, se destinará un botóon a la arquitectura global del sistema y el resto se usarán para generar señales auxiliares dentro de la arquitectura de los efectos.

En el apéndice D se pueden encontrar dos tablas, una de ellas con la aplicación de los *switches* y a qué arquitectura va destinado cada *switch* (D.2); la otra tabla representará lo mismo pero con los botones (D.1).

4.3. Matriz de diodos Leds y Displays de 7 segmentos

En esta sección del capítulo 3 se hablará sobre la arquitectura que se ha desarrollado para los dieciséis *LEDs* y los *Displays* de 7 segmentos:

4.3.1. Vúmetro LED

Un vúmetro *LED* es un sistema que a partir de una matriz de diodos *LED* es capaz de representar un nivel de intensidad, es decir, a partir de una señal de entrada se puede cuantificar su nivel de ganancia o potencia encendiendo o apagando los diodos *LED* de manera sucesiva [19]. **Para nuestro pedal implementaremos un vúmetro digital que nos permitirá saber cuándo la señal de audio que está entrando a la FPGA se encuentra saturada y por consiguiente poder regular con mayor eficacia la etapa de pre-amplificación**.

Para desarrollar este vúmetro se han empleado los dieciséis diodos *LEDs* que dispone la *Nexys A7*. Si el sistema está funcionando ($SW15 = 1$) el vúmetro se activa y por lo tanto

toma un duplicado del registro de 16 *bits* que genera el protocolo i2s2. Este registro se lee y, dependiendo del valor que tome dicho registro, se encenderán los *LEDs* que correspondan a ese valor registrado. Como se ha mencionado, siempre se encenderán de manera sucesiva desde el LED_0 hasta el LED_{15}. Si el sistema está pausado ($SW15 = 0$) se encenderán los *LEDs* 15, 12, 11, 9, 6, 4, 3 y 0 y el resto permanecerán apagados.

En el apéndice C.3 se puede encontrar la lógica combinacional con la que se ha implementado el cuantificador para la señal de entrada del vúmetro y, en consecuencia, los *LEDs* que se encienden para cada una de las señales de entrada. Por tanto, la expresión matemática que define el cuantificador será la siguiente (4.1):

$$Cuantificador = \sum_{i=1}^{16} \frac{2^{16}}{2^i} \tag{4.1}$$

4.3.2. Play/Pause

Los *displays* de 7 segmentos que tiene la *Nexys A7* se emplearán para realizar una pequeña aplicación que nos indicará si el pedal está funcionando o está en *stand-by*. Simplemente, habrá que fijarse en el estado de la señal del transceiver *reset_s*. Esta señal se conecta al SW15 de la plataforma (tal y como se puede ver en la tabla D.2). Ahora bien, si esta a nivel alto, el transceiver estará activo y por lo tanto el sistema funciona. En este caso, se podrá ver a través de los *displays* la palabra "PLAY". Por el contrario, si estuviera a nivel bajo la señal de *reset_s* se representará la palabra "PAUSE".

Para conseguir que estos *displays* estén activos se debe sumistrar la alimentación oportuna a través de los pines correspondientes, en este caso, serán los asociados a los puertos *an*. Además, se deben activar también los siete segmentos de cada *display*. La idea del funcionamiento de estos *displays* es crear un reloj de frecuencia superior a $60Hz$ que se encargue de refrescar las alimentaciones de cada uno de estos *displays* y a través de una *look-up table* hacer un mapeado de los segmentos que se deben iluminar y cuáles no. **Es muy importante tener en cuenta que el funcionamiento es activo a nivel bajo**.

En el prototipo diseñado, el fichero *display_counter*, será el que se encargue de generar el refresco de los *displays*. Para ello se hará uso del reloj maestro del sistema global y de un pequeño contador que permita fraccionar la frecuencia de dicho reloj maestro. Por otro lado, el fichero *display_interface* dispondrá de una pequeña máquina de estados controlada por *reset_s*. Dependiendo del estado en el que se encuentre dicha máquina se aplicará un vector de segmento a cada uno de los *displays*.

4.4. Implementación de los efectos

Para la implementación de los efectos se cuenta con un fichero llamado *digital_effects*. Este fichero se encargará de **soportar toda la arquitectura asociada a los efectos que han implementado**. Dentro de este fichero se podrán encontrar todos los efectos de manera independiente y dependiendo del *Hardware* externo que esté activo en un cierto instante de tiempo, este fichero deberá de ser capaz de conmutar entre los efectos que tiene alamcenado a tiempo real. Además, este mismo se debe encargar de generar toda la lógica programable que permite la conmutación entre los distintos efectos y gestionar la entrada/salida de los registros del *transceiver i2s2*.

Tal como se ha comentado, este fichero tendrá todos los efectos digitales que se han implementado. **La manera de incorporar los efectos será tener cada uno de ellos como un *component* en Vivado e instanciarlo mediante las señales de la lógica programable que genera el propio fichero *digital_effects* junto con las señales provinientes del *transceiver i2s2***:

4.4.1. E/S

Para este efecto, simplemente lo que se hará dentro de Vivado será detectar que el registro de entrada está lleno ($en\_rx = 1$) y si hay un flanco de subida del reloj maestro. Si esto es así, se conectarán los **registros de entrada directamente a la salida**.

La condición de que el *enable* esté a nivel alto ($en\_rx = 1$) y se detecte un flanco de subida del reloj maestro será fundamental para que todos los efectos que vienen a continuación envíen el efecto procesado a los registros de salida *i2s2*.

4.4.2. Looper

En este efecto se deberá crear una máquina de estados con ruta de datos que se encargue de gestionar los ciclos de escritura y lectura de la memoria RAM a partir de los puertos de control de la memoria RAM (*ena, rsta, wea y addra*) [20]. A continuación se detallará que aporta cada uno de estos puertos:

- *ena* será la señal de *enable* de la memoria RAM. Si está a nivel alto la memoria RAM está activa, si está a nivel bajo la memoria RAM estará deshabilitada.
- *wea* se encarga de seleccionar el tipo de funcionamiento de la memoria RAM. Si está a nivel bajo, la memoria RAM está en modo lectura, si esta a nivel alto la memoria RAM

estará configurada en modo escritura.

- *rsta* es el reset de la memoria RAM. Si está activo, en el puerto de salida de la memoria RAM no habrá ningún tipo de señal, pero todo lo que previamente haya sido guardado en la memoria RAM se mantendrá.
- *addra* es la dirección de memoria de la RAM. Indica el lugar donde se almacena una palabra concreta.

Los requisitos que debe cumplir esta máquina de estados serán los siguientes:

- Detectar el estado de los *switches 5 y 6* para determinar el valor de *ena* y *wea* (C.4).
- Gestionar la grabación y reproducción de las señales de audio teniendo control sobre la posición de memoria. Por tanto, cuando la grabación acabe de reproducirse deberá volver a empezar desde el principio. Si se quisiera volver a grabar, habiendo grabado previamente, se deberá continuar grabando sin sobreescribir lo que estaba ya antes grabado (salvo en el caso de que las direcciones de memoria se acaben ya que en ese caso sí se podrá sobreescribir) (C.5).

Una consideración muy importante a tener en cuenta es que la memoria RAM tiene un tamaño de palabra de 8 bits y nuestros registros son de 16 bits, por lo tanto habrá que hacer un casteo de nuestro registro a un formato $< 1,7 >$ cuando se vaya a conectar el registro al puerto de escritura de la RAM (*dina*) y cuando se vaya a conectar el puerto de salida de la memoria RAM (*douta*) a los registros de salida habrá que aplicar un *zero padding* de 8 bits en la zona de los bits menos significativos de los registros de salida.

Finalmente, dado que gracias al protocolo empleado, se puede trabajar en estéreo, el canal derecho será el que se escriba/lea en la memoria RAM y el canal izquierdo se mantendrá funcionando a tiempo real como si de un efecto entrada/salida se tratase.

4.4.3. BankFilter

La implementación de este efecto requiere del filtro FIR que se presentó en la sección 3.1.3.3. Para implementar esta estructura en VHDL se seguirá el procedimiento que se llevó a cabo en la asignatura *Diseño de Sistemas Electrónicos Digitales* [21]:

- El primer paso será detallar la cantidad de recursos que se quieren destinar a la implementación del filtro. **En nuestro caso se empleará un sumador, dos medios**

multiplicadores y 16 registros.

- En segundo lugar se realizará la planificación temporal junto con la vinculación, análisis de tiempo de vida de variables y asignación de registros.
- En tercer lugar se hace un análisis de conexiones para la extracción de multiplexores.
- En cuarto lugar se realizará la implementación junto con su cronograma.
- Finalmente, una vez realizada la implementación se necesitará hacer una cuantificación de las señales auxiliares que son necesarias para llevar a cabo la implementación. **En esta cuantificación habrá que tener en cuenta los desbordamientos que pueden ocasionar las multiplicaciones y las sumas que se van a realizar**. Pese a ello, **a la salida del filtro se debe tener una señal en formato** $< 1, 15 >$.

En el apéndice C.6 se puede ver el formato que debe tener cada una de las señales auxiliares para tener en cuenta los desbordamientos junto con el diagrama general de la implementación del filtro FIR y su cronograma de tiempos.

Una vez obtenidos los diagramas de la implementación y el cronograma será mucho más fácil recrear el filtro en lenguage VHDL ya que lo único que hará falta será un fichero que tenga la ruta de datos de la lógica combinacional desarrollada junto con los coeficientes que definen el filtro FIR (*Ruta_datos_FIR_bankfilter*) y un controlador que recree la lógica programable del cronograma obtenido (*controlador_fir_bankfilter*). Finalmente, estos dos sub-ficheros se instancian y se consigue tener la arquitectura del filtro FIR perfectamente integrada para poder usarse con los registros provinientes desde la arquitectura *i2s2*.

4.4.4. Delay

Para la implementación del efecto *delay* se deberá hacer uso de la estructura que fue presentada en la figura 3.1. A la hora de realizar la implementación en VHDL, se debe suministrar a los registros *i2s2* de salida la señal de entrada que no está retardada y la señal que se ha retardado. La manera de retardar esta señal será crear un bucle *for* que tenga como índice el valor de la constante de retardo y las señales de entrada de audio, en este caso $\tau = 4000$. Una vez que la señal termine de procesarse dentro del bucle *for* estará lista para enviarse a los registros de salida pero esta se debe colocar en el registro de salida como un desplazamiento lógico a la derecha y aplicarle el correspondiente nivel de atenuación detallando el número de posiciones que queremos que se desplace a la derecha.

```
l_data_out <= std_logic_vector(signed(l_data_in) + shift_right(l_data_reg(n-1), 1));
r_data_out <= std_logic_vector(signed(r_data_in) + shift_right(r_data_reg(n-1), 1));
```

Figura 4.3: Asignación de los registros de salida *i2s2* para el *delay*.

4.4.5. Vibrato

En la implementación del *vibrato* hay que ayudarse del código VHDL que se realizó para el *delay*. En este caso, cuando se asignen las señales de salida hay que eliminar *l_data_in* y *r_data_in* y dejar solamente la señal que ofrece el bucle *for* con el retardo correspondiente.

La carecterística importante que aparece aquí es que se deberá crear una señal sinusoidal auxiliar, para ello se empleará el paquete *sine_package.all* [22]. Así pues, con la amplitud de esta señal sinusoidal se modulará el retardo dentro del desplazamiento lógico hacia la derecha.

```
l_data_out <= std_logic_vector(shift_right(l_data_reg(n-to_integer(unsigned(wave_out_retard))-1),1));
r_data_out <= std_logic_vector(shift_right(r_data_reg(n-to_integer(unsigned(wave_out_retard))-1),1));
```

Figura 4.4: Asignación de los registros de salida *i2s2* para el *vibrato*.

4.4.6. Chorus

Para este efecto habrá que fijarse en el código VHDL desarrollado en el *delay* y en el *vibrato*. La consideración a tener en cuenta ahora es que la señal que debe entrar al bucle *for* no es la señal de entrada del audio, sino que debe ser la señal de salida que se crea a partir de la señal de entrada junto con la señal retardada como se explicaba en la figura 3.4. Para realizar esta función habrá que ayudarse de una señal interna debido a que el lenguaje VHDL no permite realizar una conexión de tipo *out* sobre otra de tipo *out*. Otra posible solución a este problema sería emplear señales *inout* pero su uso está desaconsejado ya que en el proceso de síntesis se pueden producir *latches*.

De nuevo se empleará el paquete *sine_package.all* para modular el retardo.

```
l_data_out_aux <= std_logic_vector(signed(l_data_in) + shift_right(l_data_reg_aux(n-to_integer(unsigned(wave_out_retard))-1), 1));
r_data_out_aux <= std_logic_vector(signed(r_data_in) + shift_right(r_data_reg_aux(n-to_integer(unsigned(wave_out_retard))-1), 1));
l_data_out <= l_data_out_aux;
r_data_out <= r_data_out_aux;
```

Figura 4.5: Asignación de los registros de salida *i2s2* para el *chorus*.

4.4.7. Reverb

Cuando se vaya a implementar el *reverb* en VHDL, el mejor punto de partida será ayudarse del código desarrollado para el *delay* y el *chorus*. En este caso habrá que disponer de dos bucles *for*: uno para la señal de entrada de audio y otro para la señal de salida de audio que realimenta el sistema. Ahora bien, a la salida lo único que se debe hacer es colocar las tres señales con las que se trabaja.

En el caso del *reverb* sin parametrización, simplemente se colocan los valores que se detallaron en el capítulo 3 en la sección 3.1.1.4.

```
l_data_out_aux <= std_logic_vector(-signed(l_data_in)/2 + shift_right(l_data_reg(n-1),0) + shift_right(l_data_reg_aux(n-1),1));
r_data_out_aux <= std_logic_vector(-signed(r_data_in)/2 + shift_right(r_data_reg(n-1),0) + shift_right(r_data_reg_aux(n-1),1));
l_data_out <= l_data_out_aux;
r_data_out <= r_data_out_aux;
```

Figura 4.6: Asignación de los registros de salida *i2s2* para el *reverb* no parametrizado.

En el caso del *reverb* parametrizado se debe hacer uso de señales auxiliares. Estas señales se crearán dentro de la lógica interna programable del efecto a través de una máquina de estados finita. En el apéndice C.7 se puede ver el diagrama de las máquinas de estados desarrolladas para cada una de las señales auxiliares.

Una consideración importante para el diseño de estas señales auxiliares es que su modulador está asociado a los botones *hardware* de la *Nexys A7* y por lo tanto hay que implementar un *software* de control para no saturar el pulsador. Este *software* se realiza mediante máquinas de estados finitas con ruta de datos. En el apéndice C.8 se puede ver el diagrama de estas máquinas de estados.

```
if(enable_in = '1' and enable_n1 = '1') then
    l_data_out_aux1 <= std_logic_vector(-signed(l_data_in)/g1 + shift_right(l_data_reg1(n1-1),g3) + shift_right(l_data_reg_aux1(n1-1),g2));
    r_data_out_aux1 <= std_logic_vector(-signed(r_data_in)/g1 + shift_right(r_data_reg1(n1-1),g3) + shift_right(r_data_reg_aux1(n1-1),g2));
elsif(enable_in = '1' and enable_n2 = '1') then
    l_data_out_aux1 <= std_logic_vector(-signed(l_data_in)/g1 + shift_right(l_data_reg1(n1-501),g3) + shift_right(l_data_reg_aux1(n1-501),g2));
    r_data_out_aux1 <= std_logic_vector(-signed(r_data_in)/g1 + shift_right(r_data_reg1(n1-501),g3) + shift_right(r_data_reg_aux1(n1-501),g2));
elsif(enable_in = '1' and enable_n3 = '1') then
    l_data_out_aux1 <= std_logic_vector(-signed(l_data_in)/g1 + shift_right(l_data_reg1(n1-1001),g3) + shift_right(l_data_reg_aux1(n1-1001),g2));
    r_data_out_aux1 <= std_logic_vector(-signed(r_data_in)/g1 + shift_right(r_data_reg1(n1-1001),g3) + shift_right(r_data_reg_aux1(n1-1001),g2));
elsif(enable_in = '1' and enable_n4 = '1') then
    l_data_out_aux1 <= std_logic_vector(-signed(l_data_in)/g1 + shift_right(l_data_reg1(n1-1499),g3) + shift_right(l_data_reg_aux1(n1-1499),g2));
    r_data_out_aux1 <= std_logic_vector(-signed(r_data_in)/g1 + shift_right(r_data_reg1(n1-1499),g3) + shift_right(r_data_reg_aux1(n1-1499),g2));
end if;
```

Figura 4.7: Asignación de los registros de salida *i2s2* para el *reverb* parametrizado.

4.4.8. Eco

La implementación del eco tendrá el mismo código VHDL que la versión del *reverb* no parametrizado. El único parametro que hay que modificar dentro de este código es el valor de la constante de retardo (antes tenía un valor de $\tau = 500$ y ahora tomará un valor de $\tau = 5000$).

4.4.9. Overdrive

La programación de este efecto en VHDL se puede hacer rápidamente incorporando una señal interna auxiliar junto con dos constantes. Estas dos constantes serán los umbrales positivo y negativo por los que se hará pasar la señal de los registros de entrada del protocolo *i2s2*. Lo único que habrá que hacer será interconectar la señal auxiliar a los registros de salida del protocolo *i2s2* y dentro de un *process* asignar a esta señal auxiliar el valor de los registros de entrada o de los umbrales definidos siguiendo la estrucutra que se explicó en la figura 3.6.

4.4.10. Compressor

Para llevar a cabo la programación de este efecto en VHDL se tomará como referencia el código desarrollado para el efecto *overdrive*. Lo que diferencia este efecto es que en este caso se deberán añadir otras dos constantes que corresponden con la ganancia de la zona lineal y la ganancia de la zona no lineal y que la estructura que se debe seguir ahora en el *process* será la mostrada en la figura 3.7.

```
signal l_data_out_aux, r_data_out_aux: signed ((d_width*3/2-1) downto 0);
signal Vth_negative : signed (d_width-1  downto 0) := "1001111111111111"; --Umbral de la zona no lineal negativa
signal Vth_positive : signed (d_width-1  downto 0) := "0110000000000000"; --Umbral de la zona no lineal positiva
signal g1 : signed ((d_width/2 - 1) downto 0) := "01010000"; --Ganancia para zona lineal
signal g2 : signed ((d_width/2 - 1) downto 0) := "00010000"; --Ganancia para zona no lineal
```

Figura 4.8: Señal de salida auxiliar, ganancias y umbrales para el efecto *compressor*.

4.5. Recursos consumidos por el código VHDL implementado

Una vez realizada toda la programación de los efectos comentados en la sección 4.4 se deberá proceder a realizar el proceso de síntesis, implementación y *bitstream*. Vivado será el encargado de realizar estos procesos.

En el proceso de síntesis, el código implementado será analizado por Vivado y paso a paso lo irá transformando en lógica digital. Una vez este proceso finalice se debe comprobar si

ha sido capaz de transformar ese código sin exceder los recursos disponibles.

Resource	Estimation	Available	Utilization %
LUT	27977	63400	44.13
LUTRAM	18246	19000	96.03
FF	34253	126800	27.01
DSP	4	240	1.67
IO	61	210	29.05

Figura 4.9: Utilización de los recursos de la *Nexys A7* tras el proceso de síntesis.

Tal como se puede ver en la figura 4.9, el proceso de síntesis se pasa con éxito pero habrá que tener cuidado con los recursos *LUTRAM*. Es decir, hay que tener cuidado con los valores de las constantes de retardo ya que, si se aumenta el valor de estas constantes, es posible que se agoten los recursos.

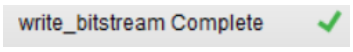
Ahora bien, en el proceso de implementación, se tomará la lógica digital que ha generado la síntesis del programa y se encargará en primer lugar de realizar un mapeado de ella. Una vez concluya el mapeado se procederá al rutado de toda la lógica que ha sido mapeada en el interior de la FPGA.

Resource	Utilization	Available	Utilization %
LUT	27952	63400	44.09
LUTRAM	18246	19000	96.03
FF	34267	126800	27.02
BRAM	128	135	94.81
DSP	4	240	1.67
IO	61	210	29.05
BUFG	3	32	9.38
PLL	1	6	16.67

Figura 4.10: Utilización de los recursos de la *Nexys A7* tras el proceso de implementación.

Fijándose ahora en la figura 4.10, se puede ver que el proceso de implementación se pasa correctamente. El motivo de que la *BRAM* esté casi rozando el límite de su utilización se debe a que en el efecto *looper* se ha ajustado su programación para poder grabar el mayor tiempo posible (siendo finalmente este tiempo de aproximadamente 30 segundos).

Finalmente, una vez realizado el proceso de implementación, se generará el *bitstream*. En este proceso Vivado generará un fichero codificado en notación binaria que servirá de ejecutable para la *Nexys A7*. Para comprobar que se dispone de ese *bitstream* se debe tener en la esquina superior derecha de Vivado la siguiente confirmación:



CAPÍTULO 5

Pruebas sobre el algoritmo desarrollado

Este capítulo consistirá en demostrar que el prototipo que se ha implementado funciona perfectamente atendiendo a las especifícaciones que se exigían para el mismo. Para probar que todo el sistema funciona lo mejor será definir pruebas concretas para cada una de las arquitecturas implementadas.

Los recursos de los que se dispone para realizar las pruebas son 3:

- *Test-Benches*. Es un fichero de simulación en lenguaje VHDL que emplea el entorno de simulación de Vivado para estudiar el funcionamiento de la lógica programable desarrollada.
- *Matlab*. Permitirá tratar con ficheros de audio *.wav*
- *Bitstream*. Se carga el programa desarrollado en la *Nexys A7* y se comprueba de manera visual y auditiva su funcionamiento.

Así bien, dependiendo del tipo de arquitectura que se este probando se empleará el método que mejor respuesta (o más información) permita obtener. En algunos casos es muy probable que se lleguen a emplear hasta dos métodos para mejor la compresión de la arquitectura que se ha implementado:

5.1. Pruebas sobre el i2s2

Para probar el funcionamiento sobre la arquitectura *i2s2* se empleara un *test-bench*. Dentro de este *test-bench* se deberá instanciar todo el diseño del *transceiver i2s2* que se explico en el capítulo 4 en la sección 4.1. Dentro de este *test-bench* se debe estudiar que todas las señales que se crean para el funcionamiento del protocolo esten perfectamente diseñadas y sincronizadas con el funcionamiento deseado.

Además se puede realizar una

5.2. Pruebas sobre los LEDs

5.3. Pruebas sobre los efectos desarrollados

5.3.1. Prueba E/S

5.3.2. Prueba Looper

5.3.3. Prueba BankFilter

5.3.4. Prueba Delay

5.3.5. Prueba Vibrato

5.3.6. Prueba chorus

5.3.7. Prueba Reverb

5.3.8. Prueba Eco

5.3.9. Prueba Overdrive

5.3.10. Prueba Compressor

CAPÍTULO 6

Conclusiones y trabajo futuro

6.1. Conclusiones

6.2. Trabajo futuro

APÉNDICE A

Impact of this project

This appendix reflects, quantitatively or qualitatively, on the possible impact...

A.1. Social impact

APÉNDICE B

Economic budget

This appendix details an adequate budget to bring about the project...

B.1. Physical resources

APÉNDICE C

Diagramas de bloques para VHDL

C.1. Arquitectura del transceiver i2s2

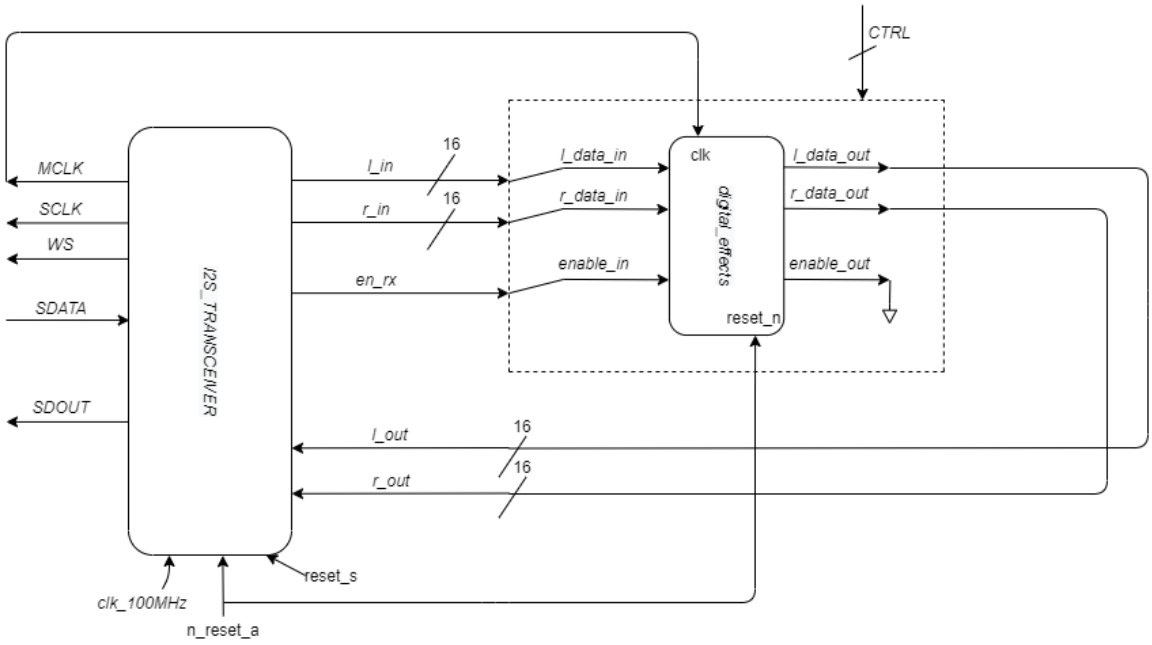


Figura C.1: Arquitectura del *transceiver i2s2*.

C.2. FSMD i2s2

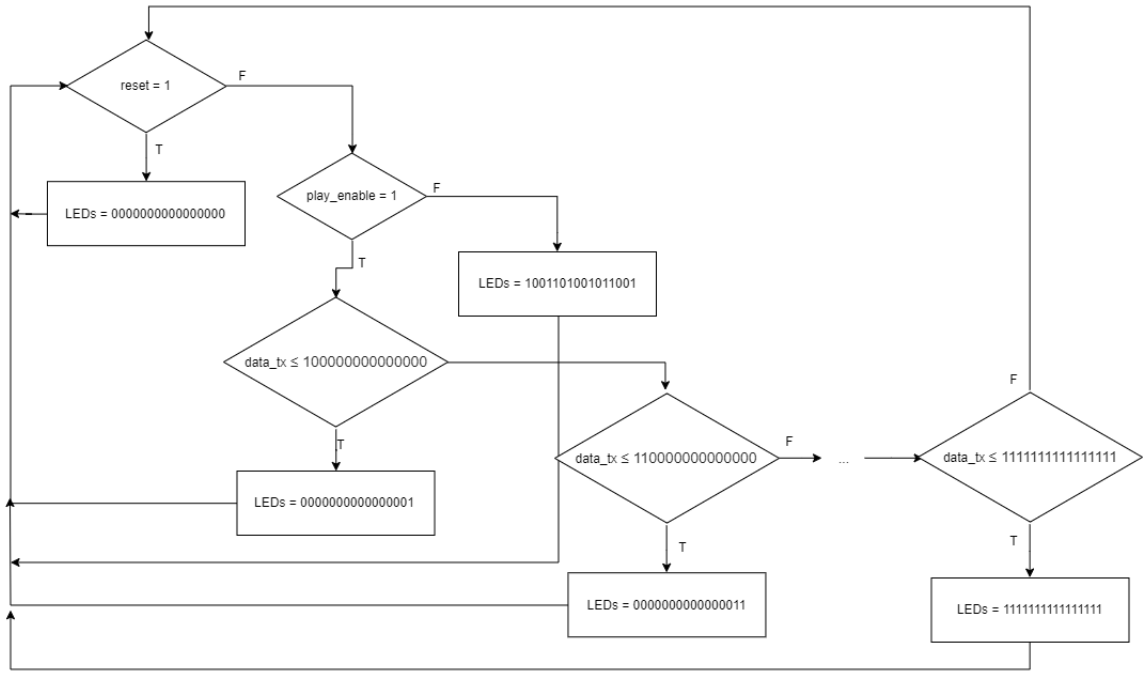


Figura C.2: Máquina de estados finita con ruta de datos para la creación de las señales del protocolo i2s2.

C.3. Diagrama para la cuantificación del vúmetro

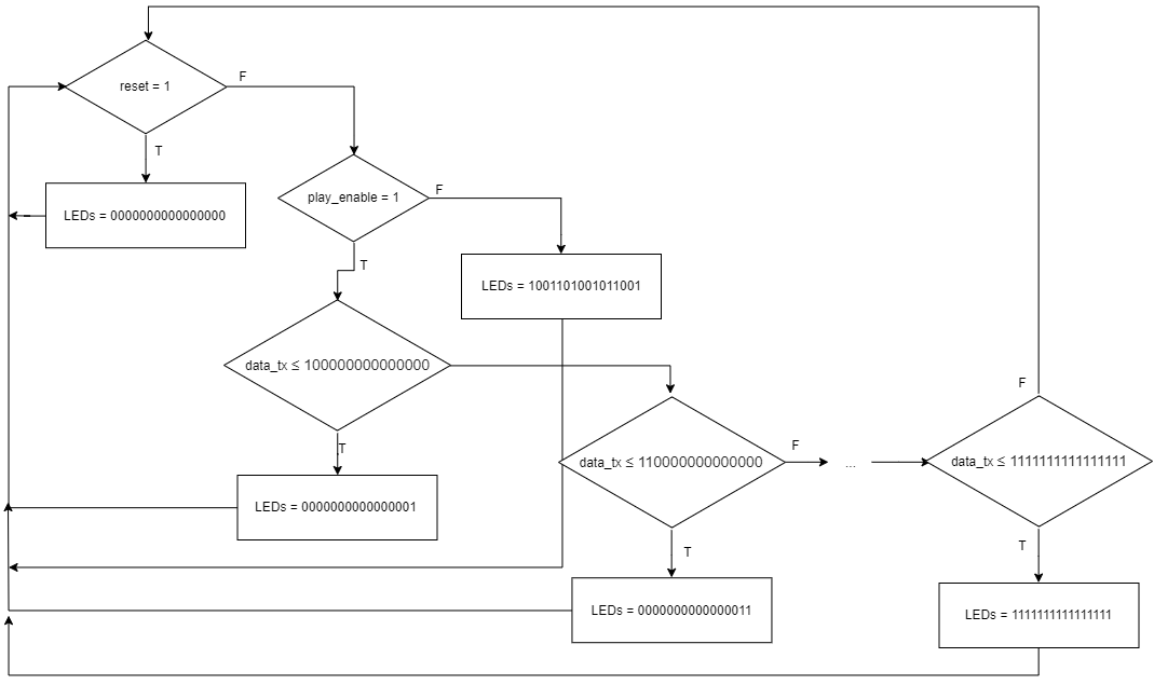


Figura C.3: Diagrama para la cuantificación del vúmetro.

C.4. Lógica para la elección de ena y wea

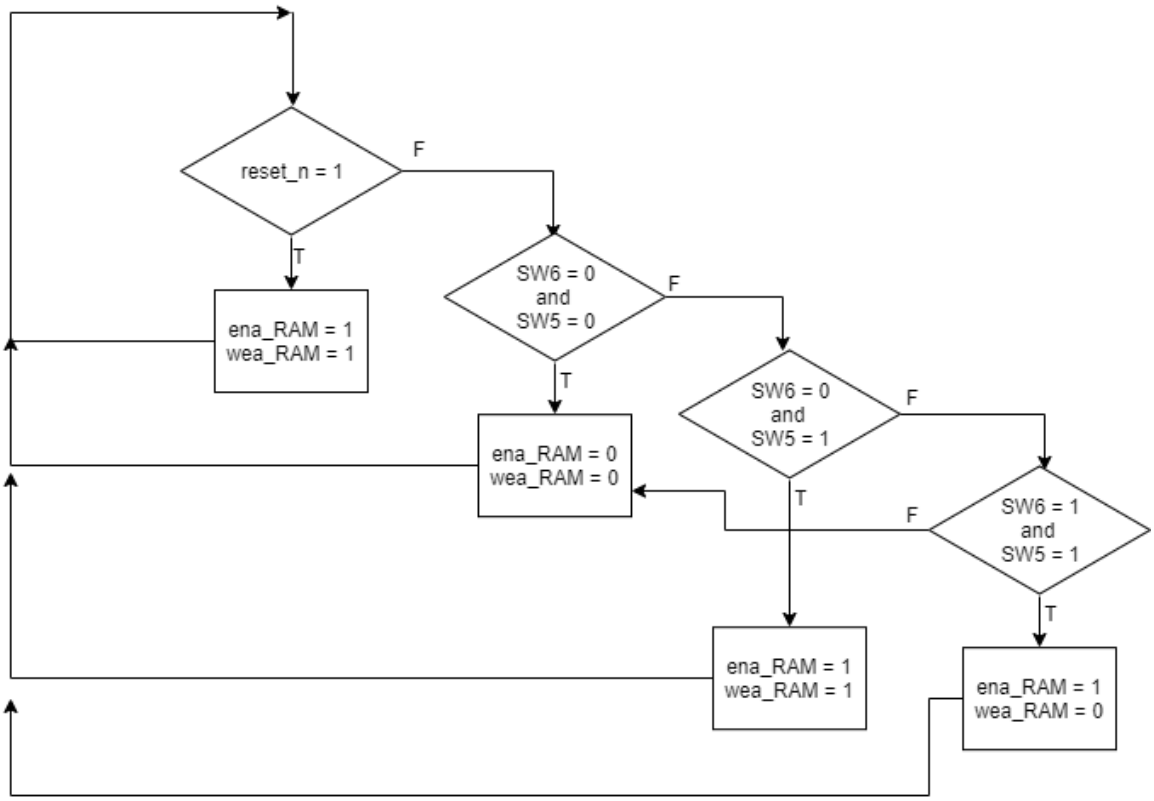


Figura C.4: Lógica para la elección de *ena* y *wea*.

C.5. Diagrama para la lecutra y escritura de la memoria RAM

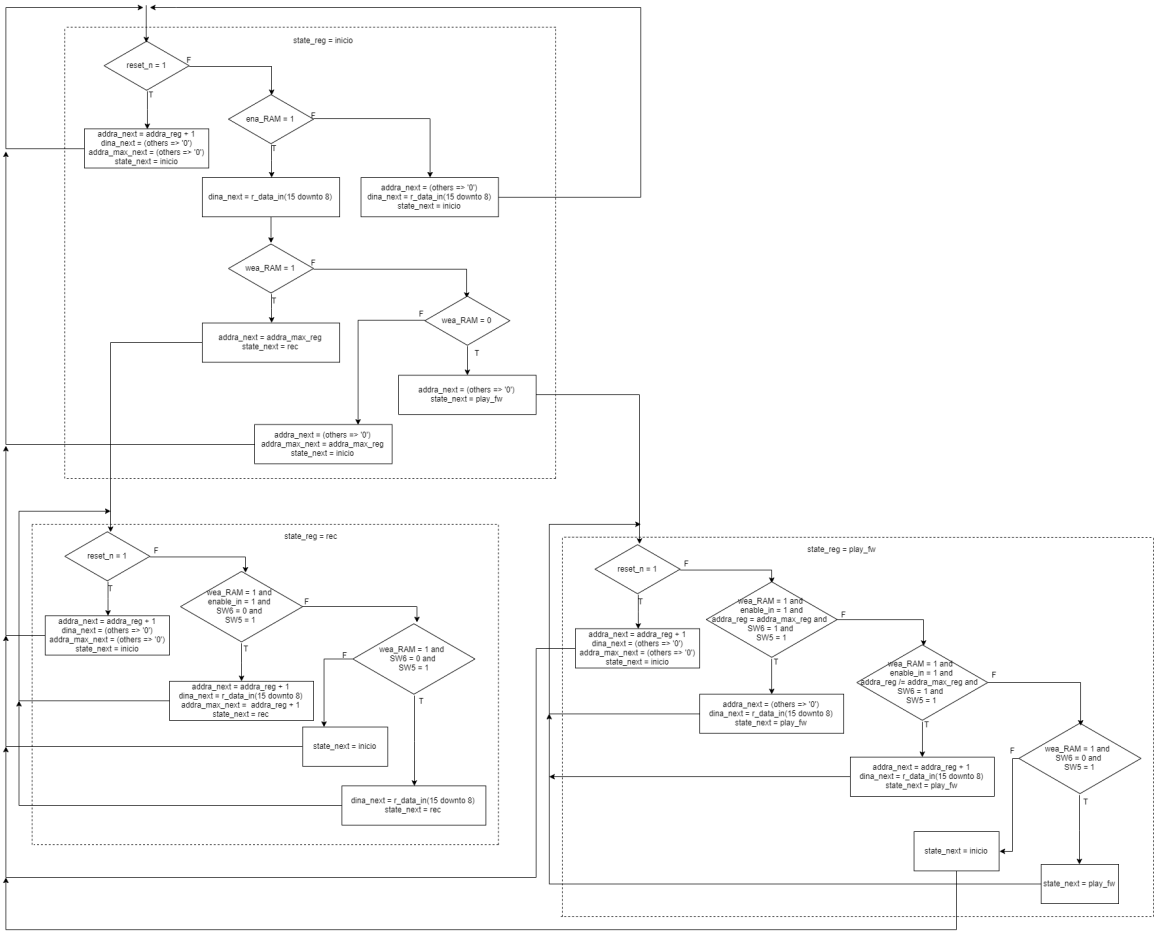
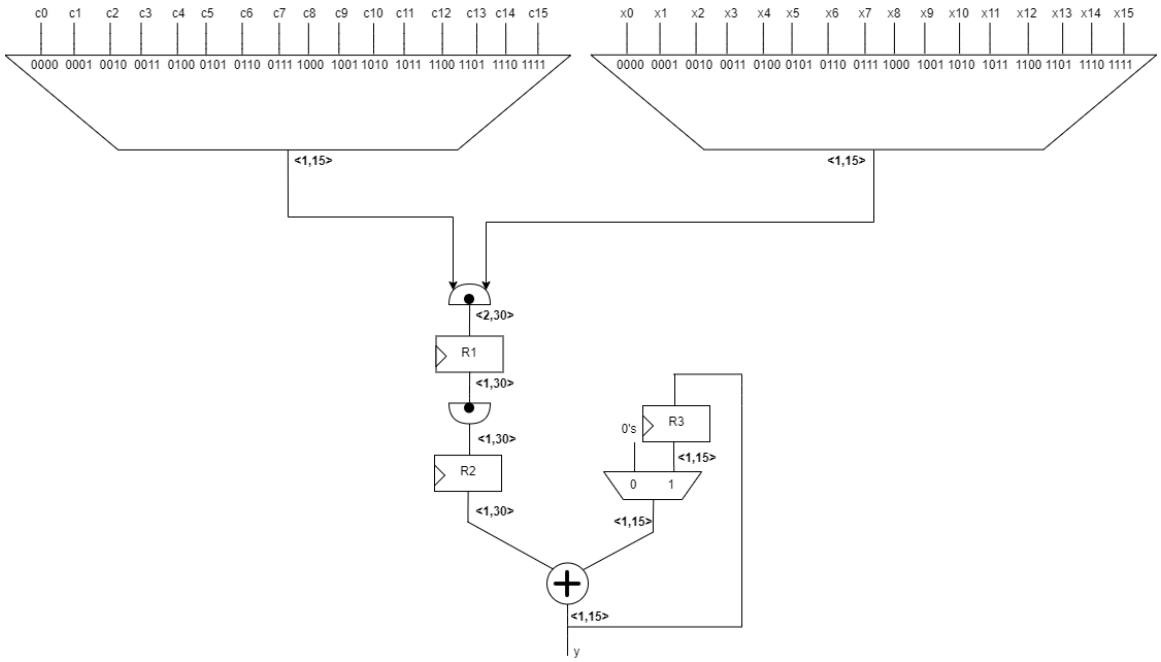


Figura C.5: Máquina de estados finita con ruta de datos para la gestión de la memoria RAM.

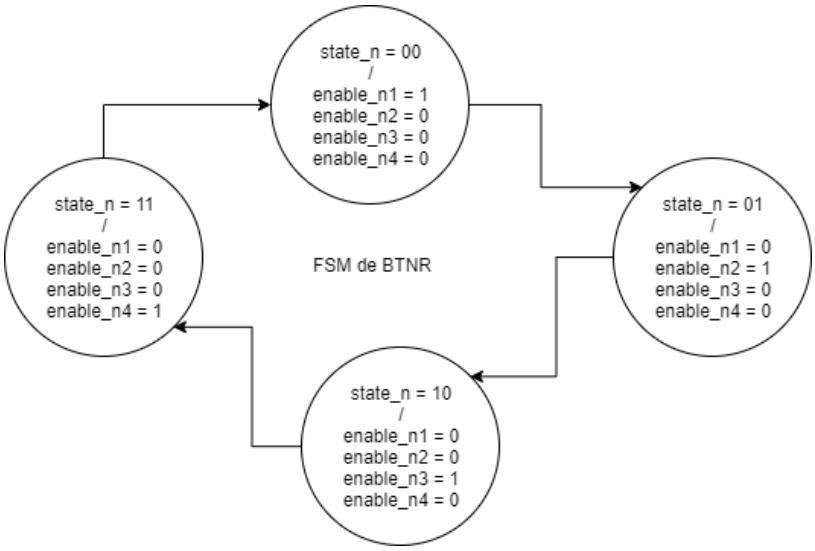
C.6. Implementación y cronograma del filtro FIR

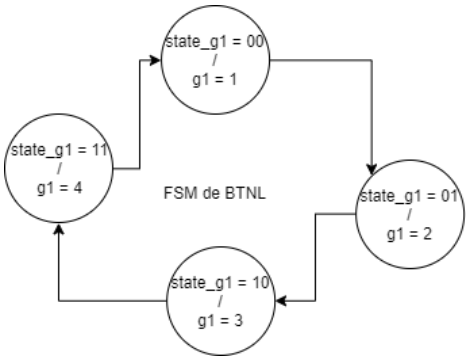


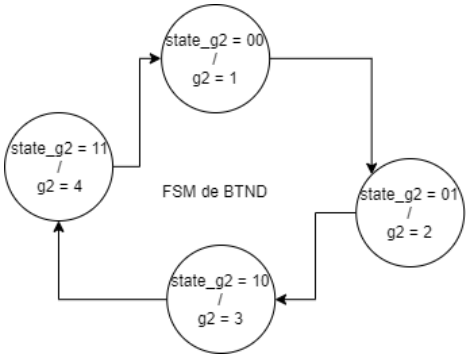
Cronograma	T=0	T=1	T=2	T=3	T=4	T=5	T=6	T=7
M1	c0 (0000)	c1 (0001)	c2 (0010)	c3 (0011)	c4 (0100)	c5 (0101)	c6 (0110)	c7 (0111)
M2	x0 (0000)	x1 (0001)	x2 (0010)	x3 (0011)	x4 (0100)	x5 (0101)	x6 (0110)	x7 (0111)
M3	0	0	0	1	1	1	1	1
Cronograma	T=8	T=9	T=10	T=11	T=12	T=13	T=14	T=15
M1	c8 (1000)	c9 (1001)	c10 (1010)	c11 (1011)	c12 (1100)	c13 (1101)	c14 (1110)	c15 (1111)
M2	x8 (1000)	x9 (1001)	x10 (1010)	x11 (1011)	x12 (1100)	x13 (1101)	x14 (1110)	x15 (1111)
M3	1	1	1	1	1	1	1	1

Figura C.6: Implementación y cronograma del filtro FIR.

C.7. Lógica interna de las señales parametrizables







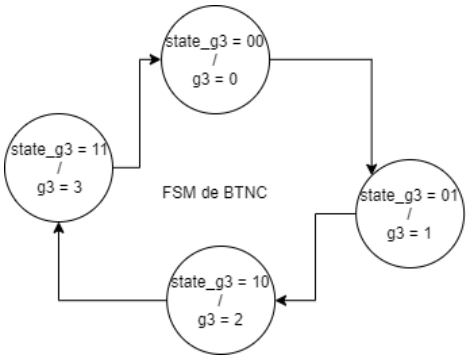


Figura C.7: FSM para la lógica interna de las señales parametrizables.

C.8. FSMD para el control de la saturación

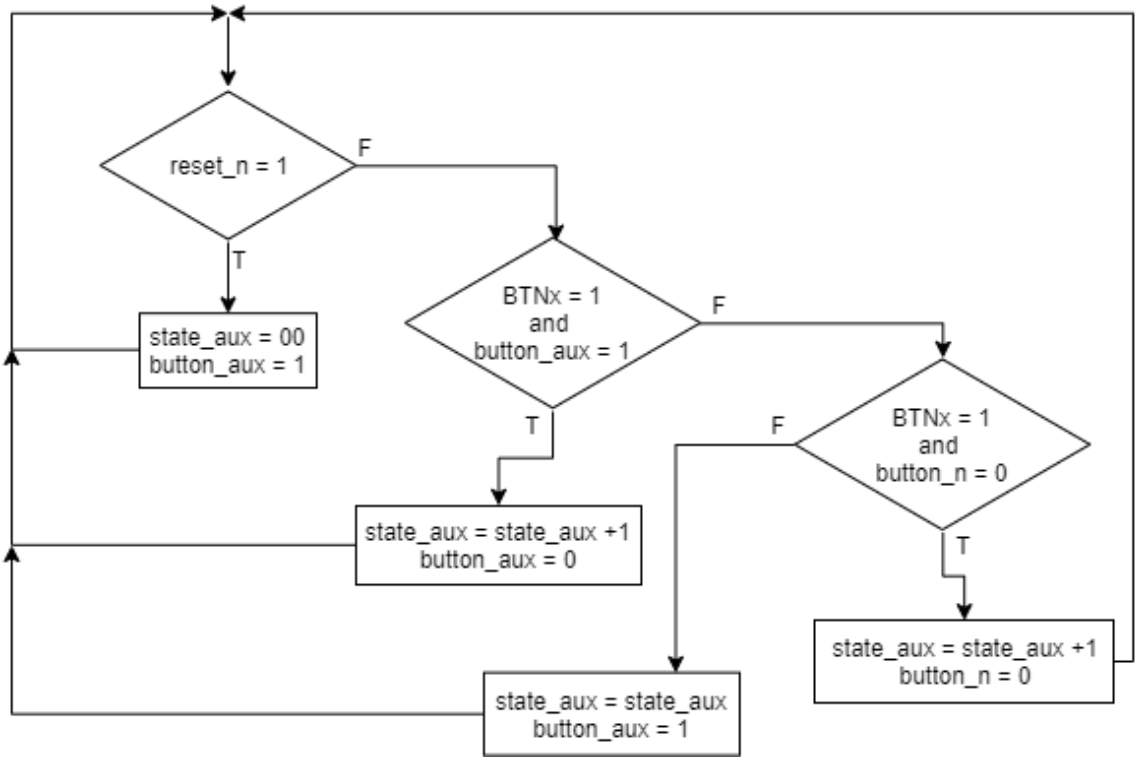


Figura C.8: FSMD para el control de la saturación de los botones.

APÉNDICE D

Tablas para la distribución del Hardware

D.1. Tabla de distribución de los botones

BTNU	*Reset asíncrono general*	arquitectura global
BTNR	*Control de la línea de retardo*	*arquitectura de los efectos (reverb_params)*
BTNC	*Volumen muestra retardada de salida*	*arquitectura de los efectos (reverb_params)*
BTNL	*Volumen muestra original*	*arquitectura de los efectos (reverb_params)*
BTND	*Volumen muestra retardada de entrada*	*arquitectura de los efectos (reverb_params)*

D.2. Tabla de distribución de los switches

SW0	*delay*	arquitectura de efectos
SW1	*chorus*	arquitectura de efectos
SW2	*reverb*	arquitectura de efectos
SW3	*eco*	arquitectura de efectos
SW4	*vibrato*	arquitectura de efectos
SW5	*looper_write*	arquitectura de efectos
SW6	*looper_read*	arquitectura de efectos
SW7	*bankfilter*	arquitectura de efectos
SW8	*overdrive*	arquitectura de efectos
SW9	*compressor*	arquitectura de efectos
SW10	*reverb_params*	arquitectura de efectos
SW11	*overdrive+delay*	arquitectura de efectos
SW12	*compressor+looper*	arquitectura de efectos
SW13	*looper muted*	arquitectura de efectos
SW14	*filter_select*	arquitectura de efectos
SW15	*play_enable*	arquitectura global

Bibliografía

[1] Digilent. Nexys4 DDR FPGA Board Reference Manual, Abril 2016.

[2] Electronicavm. I2s Pmod Quick Start (VHDL).
Accedido en 28-9-2019 a https://electronicavm.wordpress.com/2011/05/28/vumetro-monofonico-a-base-de-leds/, Octubre 2019.

[3] Udo Zölzer. DAFX: Digital Audio Effects. Second Edition. 2011.

[4] Yamaha. Especificaciones amplificador yamaha GA15.
Accedido en 9-11-2019 a https://es.yamaha.com/es/products/musical_instruments/guitars_basses/amps_accessories/ga15ii/index.html.

[5] Line6. Spider III User Manual.
Accedido en 9-11-2019 a https://line6.com/legacy/spideriii75/.

[6] Xilinx. Vivado Design Suite.
Accedido en 23-9-2019 a https://www.xilinx.com/products/design-tools/vivado.html, 2019.

[7] Digilent. Nexys A7 Reference Manual.
Accedido en 18-9-2019 a https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual.

[8] Enciclopedia. Instrumento electrófono.
Accedido en 28-12-2019 a http://enciclopedia.us.es/index.php/Instrumento_electr%C3%B3fono, Febrero 2017.

[9] Wikipedia. Pastilla (micrófono).
Accedido en 28-12-2019 a https://es.wikipedia.org/wiki/Pastilla_(micr%C3%B3fono), Septiembre 2012.

[10] Helenca Duxans Barrobés & Marta Ruiz Costa-jussà. Efectos digitales de la señal de audio. Accedido por última vez en 27-12-2019 a https://www.exabyteinformatica.com/uoc/Audio/Procesamiento_de_audio/Procesamiento_de_audio_(Modulo_5).pdf.

[11] EUM. Introducción a los filtros digitales-Clase 10. Accedido en 27-11-2019 a https://www.eumus.edu.uy/eme/ensenanza/electivas/dsp/presentaciones/clase10.pdf, 2011.

[12] John Michael Espinosa Durán & Pedro P. Liévano Torres & Claudia P. Rentería Mejía & Jaime Velasco Medina. Diseño De Un Microsistema Programable Para Efectos De Audio Digital Usando FPGAs. Accedido en 16-9-2019 a http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1794-12372014000200011, Diciembre 2014.

[13] Micromodeler. Micromodeler DSP. Accedido en 26-12-2019 a https://www.micromodeler.com/dsp/.

[14] Cardiff University. Digital Audio Effects.
Accedido en 2-2-2019 a http://users.cs.cf.ac.uk/Dave.Marshall/CM0268/PDF/10_CM0268_Audio_FX.pdf.

[15] Javier Otero. Diseño e implementación sobre fpga de un pedal de efectos digital. Trabajo fin de grado, Universidad Politécnica de Madrid, ETSI Telecomunicación, June 2019.

[16] Philips Semiconductors. I2S bus specification.
Accedido en 8-10-2019 a https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf, Febrero 1986.

[17] Xilinx. Clocking Wizard v6.0.
Accedido en 2-10-2019 a https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf, Febrero 2019.

[18] Digilent. Pmod I2S2.
Accedido en 18-10-2019 a https://reference.digilentinc.com/reference/pmod/pmodi2s2/start, Octubre 2019.

[19] Digilent Scott Larson. Vúmetro a base de leds.
Accedido en 8-10-2019 a https://www.digikey.com/eewiki/pages/viewpage.action?pageId=85295576#I2SPmodQuickStart(VHDL)-CodeDownload, Mayo 2015.

[20] Xilinx. AXI Block RAM (BRAM) Controller v4.1.
Accedido en 16-11-2019 a https://www.xilinx.com/support/documentation/ip_documentation/axi_bram_ctrl/v4_1/pg078-axi-bram-ctrl.pdf, Mayo 2019.

[21] Eros García Arroyo & Daniel Payno Zarceño. Sistema de grabación, tratamiento y reproducción de audio-Versión 4.1. Diciembre 2018.

[22] Doulos Ltd. Synthesisable Sine Wave Generator. Accedido en 26-11-2019 a https://www.doulos.com/knowhow/vhdl_designers_guide/models/sine_wave_generator/, Agosto 2003.

[23] Examples of VHDL Conversions. Accedido en 24-10-2019 a https://www.nandland.com/vhdl/tips/tip-convert-numeric-std-logic-vector-to-integer.html.

[24] Como generar números aleatorios en vhdl con un LFSR. Accedido en 30-10-2019 a https://vhdl.es/lfsr-vhdl/.

[25] Wikipedia. Diseño de filtros de respuesta finita al impulso.
Accedido en 27-11-2019 a https://es.wikipedia.org/wiki/Dise%C3%B1o_de_Filtros_de_Respuesta_Finita_al_Impulso, Septiembre 2019.

[26] Timur. Conversion of fractional numbers between numeral systems. Accedido en 26-12-2019 a https://planetcalc.com/862/, 2011.