UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN SOBRE FPGA DE UN PEDAL DE EFECTOS DIGITAL

GRADO EN INGENIERÍA DE TECNOLOGÍAS
Y SERVICIOS DE TELECOMUNICACIÓN

JAVIER OTERO MARTÍNEZ
Madrid, Junio 2019

DISEÑO E IMPLEMENTACIÓN SOBRE FPGA DE UN PEDAL DE EFECTOS DIGITAL

Autor: Javier Otero Martínez

Tutor: Pablo Ituero Herrero
Departamento de Ingeniería Electrónica

Me gustaría agradecer a los que me han apoyado todo este tiempo: mis padres, mi familia, Aida e Impostor mientras trabajaba en el proyecto. También me gustaría agradecer especialmente a los profesores que me han ayudado desinteresadamente cuando he acudido a ellos con alguna pregunta: Alfredo Sanz, Jose Parera, Fernando Gonzalez y por supuesto a mi tutor, Pablo, por toda su paciencia estos meses.

Resumen

El presente proyecto se enmarca en el ámbito de la producción musical en tiempo real. En concreto, plantea el diseño, desarrollo e implementación de un pedal de efectos digital, cuyo efecto principal será el de octavador. Estos dispositivos permiten variar la señal entrante en tiempo real obteniendo a la salida diferentes efectos y modulaciones.

Un octavador es un dispositivo que entrega a la salida un señal que es idéntica (idealmente) a la señal de entrada salvo que su frecuencia está dividida por 2, añadiendo un carácter mucho más sólido y profundo al sonido resultante de la mezcla de ambos.

El proyecto plantea el proyecto desde cero, es decir, desde la idea original hasta la implementación de un prototipo. Todas las decisiones y criteros de diseño se llevan a cabo pensando en su utilización profesional, refiriéndome tanto a latencia como calidad del sonido, formato, latencia y el resto de parámetros involucrados.

En consecuencia, el resultado final ha sido una evaluación de diferentes algoritmos en base a sus diferentes propiedades y características para posteriormente realizar una propuesta de implementación en VHDL utilizando la herramienta Vivado. Estos conocimientos se han plasmado en un prototipo que me ha ayudado a afianzar estos conceptos y ha mejorado mi comprensión tanto del tratamiento de señales de audio como de la implementación de un algoritmo en FPGA.

Abstract

This project belong to the real-time music production context. Concretely, it addresses design, development and implementation of a digital effects pedal, which main feature will be an octavizer. Such devices allow us to modify the input signal in real time obtaining a variety of different effects and modulations as the outcome.

An octavizer provides a signal which is identical (ideally) to the introduced one except that its frequencies are divided by two, adding a much more solid and deep nature to the resulting mix.

This work is carried out from zero, indeed, from the original idea to the implementation of the prototype. Every design choices and criteria have been made thinking in its professional use, refereing to either latency, audio quality, layout and all other parameters involved.

Consequently, the outcome was an evaluation of different algorithms based on its properties and characteristcs to finally make an VHDL implementation proposal using Vivado tool. This acknowledgements have been put into practise via designing a prototype that helped me to ground such concepts and improved my comprenhension of both audio signal proccessing and algorithm implementation on FPGA.

Índice general

Capítulo 1

Introducción y generalidades

Un pedal de efectos es un dispositivo que se conecta entre un instrumento (normalmente electrófono) y su amplificador encargándose de modificar la señal de entrada y sus características fundamentales como pueden ser timbre, tono y volumen. En este proyecto se pretende estudiar las posibilidades de diseño e implementación de uno de estos pedales, enfocándolo a un uso con instrumentos de viento, en concreto el saxofón puesto que es el instrumento que yo toco. Generalmente, no es habitual el uso de pedales de efectos en instrumentos de viento, ya que en general se busca mantener el sonido lo más fiel posible al producido por el instrumento. No obstante, en multitud de producciones se pueden apreciar efectos añadidos en postproducción ya sea digital o analógicamente. Algunos ejemplos son el *reverb* o el *chorus*. Sin embargo, aqui se pretende implementar un efecto de octavador, el cual se describirá posteriormente.

Se ha decidido el combinar este efecto con la implementación en formato de pedal. Este formato se ha hecho muy popular desde su aparición, debido a que los intérpretes pueden activarlo con el pie pudiendo mantaner las manos en el instrumento. Aunque los intérpretes de instrumentos de viento no están acostumbrados al uso de pedales, se mantiene la idea del pedal por analogía con otros instrumentos.

Este proyecto abarca todo el proceso desde la idea inicial, diseño y montaje del prototipo final, por tanto, las especificaciones de funcionamiento que se han utilizado pretenden facilitar un uso profesional del prototipo, de forma que sea compatible con los estándares establecidos en el contexto musical e ingenieril al mismo tiempo.

El flujo de datos será el siguiente. En primer lugar se utiliza un transductor para adquirir la señal, en este caso, un micrófono convencional. Una vez que el estímulo externo es transformado en pulsos eléctricos, atravesará un etapa de entrada analógica que preamplifica la señal y la adecúa a la entrada de la FPGA.

Para el prototipo se ha utilizado la placa proporcionada por el departamento: Nexys A7. Esta placa monta una FPGA *Xilinx XC7A100T-1CSG324C* junto con varios switches, botones, leds y displays de 7 segmentos, que harán más fácil el manejo de la misma. Esta placa tiene un micrófono integrado, pero es de tan baja calidad que se opta por diseñar la etapa de entrada, analógica completamente, y conectarla con uno de los puertos del *Pmod i2s2*, también de Digilent y proporcionado por el departamento. Este módulo contiene ADC, DAC y los conectores de mini-jack estándar en formato de audio, que servirán para gestionar la señal de entrada y la de salida.

Se implementará un algoritmo que se encargará de llevar a cabo la octavación de la señal de entrada y de proporcionarla en la salida. Este algoritmo utiliza una aproximación de *Phase Vocoder* muy común en el tratamiento de señales de audio realizando una transformación al dominio de la frecuencia mediante FFT. Durante todo el proceso se priorizará el criterio de la baja latencia, dado que si no, resulta imposible operar en tiempo real. El algoritmo se describirá en profundidad en su capítulo correspondiente.

1.1. Objetivos

Para la realización de este proyecto de forma satisfactoria se ha establecido la consecución de una serie de objetivos que son los siguientes:

- Diseño de un sistema completo, a partir de una problemática definida previamente mediante el estudio del instrumento y las señales.
- Implementación sobre la FPGA proporcionada.
- Construcción de un prototipo para estudiar la problemática desde la práctica.
- Afianzar, debido a todo esto, conocimientos adquiridos durante el Grado en diversos ámbitos como el procesamiento de señal y en concreto en la especialidad de Sistemas Electrónicos como la programación hardware, el montaje de un circuito analógico y la correcta comunicación entre todos los módulos.

1.2. Metodología

- En primer lugar se seleccionará el efecto que se quieren implementar, dando prioridad al octavador.
- Como segundo paso, se estudiará la literatura existente y se probarán distintas soluciones algorítmicas empleando MATLAB.
- Estudio y elección de la interfaces de entrada y salida. Selección de micrófono, ADC y DAC.
- Desarrollo y verificación en VHDL empleando la herramienta VIVADO de Xillinx.
- Montaje de un prototipo empleando la placa Nexys A7 de Digilent, el micrófono seleccionado anteriormente y el resto de dispositivos que fueran necesarios.
- Test y depuración.

1.3. Fundamentos del sonido y teoría musical

Considero necesario hacer un pequeña introducción a algunos conceptos que tendrán importancia para el desarrollo del tranajo en relación con la física del sonido y la teoría musical, para facilitar la comprensión del documento por un público no especializado en

estos ámbitos. En primer lugar cabe destacar que como cualquier fenómeno físico, un sonido lleva asociados una serie de parámetros matemáticos que lo definen, pero tradicionalmente, estos términos han recibido otro nombre en el ámbito musical, con el que se enseñan en escuelas y conservatorios. Es importante, por tanto, tener clara la relación entre la nomenclatura musical y la equivalencia física y matemática.

- **Tono:** hace referencia a la altura de la nota en cuanto a grave o aguda. En parámetro que la define es la frecuencia, que se mide en Hz. Aunque el oído humano puede llegar a percibir señales de hasta 20kHz, las notas fundamentales rara vez sobrepasan los 2kHz.
- **Volumen:** equivale a la intensidad que posee la señal tanto eléctricamente como cuando se propaga por el aire como una onda de presión. Cuando se mide en un circuito eléctrico se utilizan unidades de tensión mientras que como onda se suelen utilizar los dB dado su carácter logarítmico, sin embargo, posee unidades de intensidad acústica: W/m^2.
- **Timbre:** es característico de cada fuente de sonido y es lo que nos permite diferenciar un instrumento de otro, o voces humanas entre si, aún produciendo la misma nota.

La lista de aqui arriba muestra los tres parámetros fundamentales de un sonido musical, sin embargo, no queda claro como funciona el mecanismo matemático que rige el funcionamiento del timbre.

1.3.1. Timbre, armónicos y serie armónica

Cualquier estudiente de ingeniería ha estudiado la generación de armónicos en el contexto de la física y las vibraciones. Sin embargo, puede que no tantos se hayan preguntado como se *oyen* estos modos o si se pueden percibir. La respuesta es sí, y además con mucha claridad. En lo que se refiere a la música, tanto los instrumentos musicales como la voz humana son estructuras complejas que producen sonidos a muchas frecuencias diferentes al mismo tiempo. Nuestro cerebro, cuando oye un sonido se encarga de interpretar la señal recibida para producir un sonido identificable.

Es inevitable establecer una analogía entre este funcionamiento y el terorema de Fourier, que dice que podemos descomponer cualquier señal como suma de una o varias componentes sinusoidales. Esta poderosa afirmación explica como funciona la generación de cualquier sonido musical.

Así pues, cuando se produce una nota con un instrumento, se generan varias vibraciones a distintas frecuencias, en consecuencia, es la relación entre estas componentes lo que modifica el timbre de la nota generada y nos permite identificarla. La trampa está en que las frecuencias que se producen no son aleatorias, son una serie de notas que reciben el nombre de *armónicos*. El conjunto de estos armónicos se llama *serie armónica*.

Así pues, podemos concluir con que cada nota está a su vez formada por varios armónicos presentándose en diferente proporción entre ellos. El armónico que define el sonido es el primero de la serie y recibe el nombre de fundamental, el resto se obtienen multiplicando esta frecuencia por números enteros. En la imagen 1.1 vemos como una guitarra, un fagot y un saxofón generan una mezcla de armónicos diferentes cuando interpretan la misma nota.

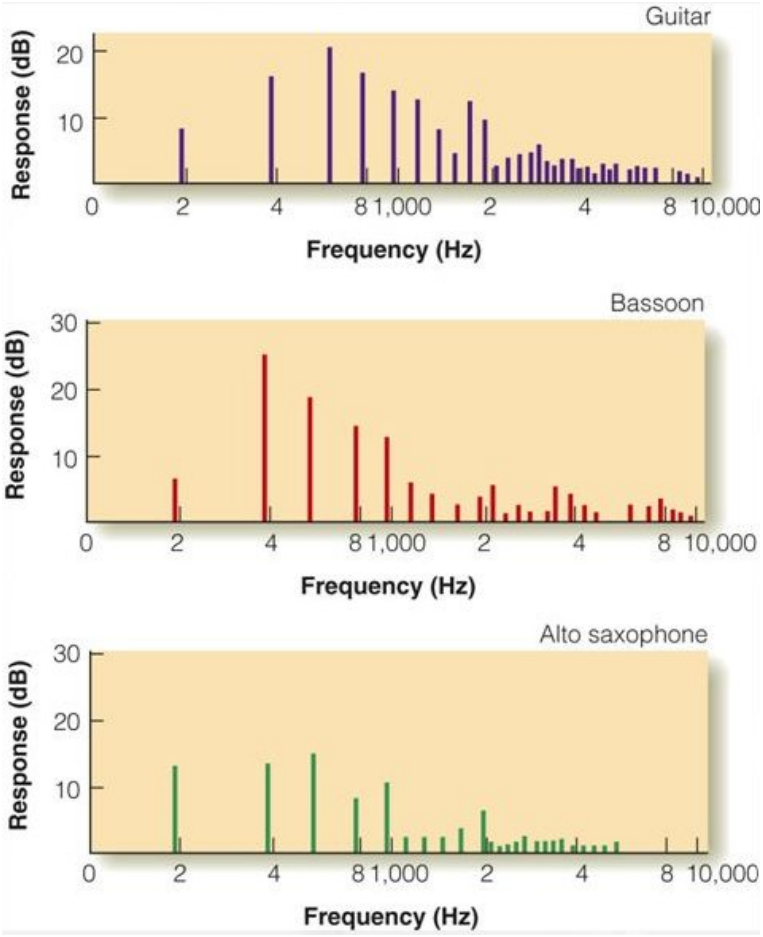


Figura 1.1: Diferentes instrumentos interpretando una misma nota

Las operaciones que vamos realizar en la señal de entrada tienen como objetivo modificar el tono de una nota introducida mientras mantenemos inalterados su volumen y su timbre. En la práctica veremos como tanto el algoritmo como la FPGA introducen desviaciones y no idealidades que nos permiten elaborar y clasificar diferentes métodos y operativas.

Capítulo 2

Entrada: etapa analógica

El pedal de efectos tendrá un estructura definida de la siguiente manera: la señal captada atravesará una etapa a la entrada que se encarga de amplificarla y adecuarla para introducirla en la FPGA, donde se procesará. Por último, el resultado será reconvertido en analógico y amplificado para permitir su reproducción.

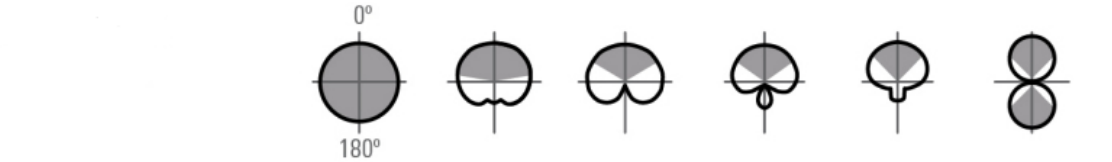
2.1. Selección de micrófono

Como normalmente se utilizan pedales de efectos en instrumentos electrófonos, la señal de salida del instrumento ya viaja por un cable de camino a la amplificador por lo que el pedal actúa como un intermediario entre ambos. Sin embargo, en instrumentos de viento, es necesario utilizar algún transductor que sea capaz de convertir la señal acústica consistente en ondas de presión en una serie de impulsos eléctricos que puedan ser procesados posteriormente.

La solución más sencilla consistiría en utilizar el micrófono que viene integrado con la placa Nexys A7: modelo *ADMP421* de Analog Devices [Nex]. No obstante, la utilización de este micrófono plantea los siguientes dos problemas.

En primer lugar, es inmediato pensar que incluso en el caso de un prototipo, si se plantea usar como pedal, no resulta nada recomendable colocar el micrófono encargado de recoger todo el sonido en el suelo. Además de estar lejos de la fuente sonora, capturaría el sonido resultante de la manipulación de los controles suponiendo una bajada en la calidad del sonido que pudiese proporcionar el dispositivo. Por tanto, es mejor utilizar un micrófono no integrado en la propia placa.

En segundo lugar, pero no menos importante, conviene tener en cuenta que la respuesta de sensibilidad del micrófono integrado es omnidireccional (ver figura 2.1), es decir, captará todos los sonidos sin importar la dirección de dónde vengan. Este tipo de transductores se usan principalmente en radio y televisión, donde puede haber varias personas hablando en el mismo micrófono o para la grabación de orquestas o agrupaciones en localizaciones cerradas determinadas. Estos micrófonos son capaces de captar tanto el sonido proveniente de la fuente directamente como los ecos y reflexiones característicos del espacio, dando una sensación de amplitud al oyente como la que produciría su escucha en esa misma localización. Pero mientras que en estos casos se trata de un efecto deseado, resulta poco



	Omni-directional	Sub-cardioid	Cardioid	Hyper-cardioid	Line	Bidirectional
Acceptance Angle (3 dB down)	—	170°	120°	100°	90°	90°
Null (angle of minimum sensitivity)	none	160°	180°	110°	120°	90°
Distance Factor (DF)	1.0	1.2	1.7	2.0	2.5	1.7

Figura 2.1: Tipos de micrófonos según su diagrama polar [Aud]

agradable captar estas reflexiones en un ambiente no preparado para ello y encima cercano al suelo, siendo más conveniente utilizar micrófonos de tipo cardioide.

Estos micrófonos son muy utilizados con instrumentos de viento, ya que el sonido suele provenir de un punto concreto. Es preciso matizar que en el caso del saxofón, contra la creencia popular, el sonido no sale siempre por la campana del instrumento. El sonido sale, por el contrario, por la primera llave abierta más próxima a la embocadura, que suele ser en torno al centro del instrumento. En consecuencia, no resulta conveniente acercar el micrófono mucho a la campana descuidando otras llaves del instrumento. La figura 2.2 muestra un saxofón alto aunque hay varios instrumentos de la misma familia.

Figura 2.2: Saxofón alto

Además de tener en cuenta el diagrama polar para la elección de micrófono, resulta imprescindible conocer los posibles tipos en cuanto a fabricación y funcionamiento. En este sentido tenemos los dos tipos más comunes: los micrófonos de condensador y los dinámicos.

Los micrófonos de condensador reciben su nombre del condensador que poseen en el interior de su cápsula. El diafragma es la pieza encargada de vibrar cuando las ondas de presión del sonido lo atraviesan pero, a su vez, se une esta pieza con una de las placas del condensador. El resultado es que con cada movimiento del diafragma, varía la distancia entre las placas y en consecuencia, se modifica la capacidad de manera inversamente proporcional. Estos cambios modifican una señal eléctrica en la que quedan registradas las ondas de sonido recibidas. Estos micrófonos poseen una buena sensibilidad pero necesitan de alimentación *phantom* de entre 24 y 48 V, que se realiza desde la mesa de mezclas por el mismo cable de señal.

Los micrófonos dinámicos por el contrario, no requieren de alimentación y además poseen buena robustez y son más baratos que los anteriores. Estos funcionan gracias a una bobina unida al diafragma que se mueve conforme a las ondas de presión recibidas del sonido dentro de un campo electromagnético. Por la acción de la inducción electromagnética, este movimiento genera una corriente que atravesará la bobina proporcionalmente al estímulo entrante. La principal desventaja de estos micrófonos es que su respuesta no es del todo lineal con la frecuancia, produciendo mayor o menor ganancia en función del rango del espectro en el que se encuentre el sonido de entrada. Para arreglar esto se suele usar ecualización posterior o diferentes diafragmas para cada rango del espectro para luego reconstruir la señal a base de sumas de los diferentes tramos.

Teniendo en cuenta su uso en el diseño, he escogido un micrófono dinámico por la robustez que presenta que además evitará tener que preocuparse de la alimentación. Además, estos micrófonos funcionan de manera muy similar entre sí, lo que resultará muy conveniente para mantener la flexibilidad del proyecto. El micrófono elegido finalmente será un *MODELO DEL MICRO QUE USE* cortesía del Club Musical Delta. Este se colocará para capturar los sonidos en un pie en la ubicación más adecuada para el instrumento y se conectará a la etapa siguiente ubicada en el suelo mediante un cable.

En cualquier caso, es necesario añadir una etapa posterior de *pre-amplificación o preamp* que adecúe la señal eléctrica del micrófono a la que pueda interpretar la FPGA.

2.2. Preamp

Las principales funciones que va a realizar la etapa de preamplificación será la de amplificar la señal entrante y transformar la señal balanceada proveniente del micrófono en una sin balancear. Se realizará de manera puramente analógica dado que el esquema está muy desarrollado en la literatura sobre el tema. En este caso he implementado el modelo propuesto por P.Allison en [PAm][1] al que he añadido una fuente de alimentación lineal estándar.

Típicamente, todos los modelos de micrófonos comerciales para aplicaciones de música, utilizan conexiones balanceadas para proporcionar su señal a la salida. El formato de estos cables de 3 hilos recibe el nombre de *XLR* pero también se pueden utilizar conectores de

[1]Sugerido por el profesor Alfredo Sanz Hervás de la UPM al que agradezco su referencia

tipo *Jack de 6.35 mm*[2] adecuados a este tipo de señales, ilustrados en la figura 2.3.

Figura 2.3: Conector tipo XLR hembra y jack no balanceado macho de 6.35 mm

El fundamento del par balanceado consiste en enviar la señal por dos conductores entrelazados con la polaridad invertida entre sí, envueltos de un tercer conductor que actúa de barrera frente a interferencias electromagnéticas externas. De esta forma, si se produce una interferencia, afectará a ambos conductores de igual manera. Posteriormente, se introduce la señal en destino en un *Amplificador de diferencia*[3] que se encargará de amplificar únicamente la diferencia entre ambas señales rechazando la interferencia sufrida. Esta idea muy extendida en los diseños analógicos, permite enviar la señal por largos cables de manera robusta a cambio de introducir un tercer conductor.

Así pues, el diseño de pre-amplificación constará de un amplificador de diferencia mediante amplificador operacional junto con una etapa de amplificación mediante transistores BJT, tal y como se puede ver en el esquema del mismo de la figura 2.4. El potenciómetro permite controlar la ganancia del sistema, es decir, el nivel de señal a la salida del mismo.

2.3. Alimentación

[2]Este tipo de conector pero sin balancear, es el utilizado en guitarras y bajos eléctricos. Para los auriculares se utiliza jack de 3.5 mm que recibe el nombre de "minijack".

[3]Del inglés *Difference Amplifier*, la traducción al castellano puede inducir a error

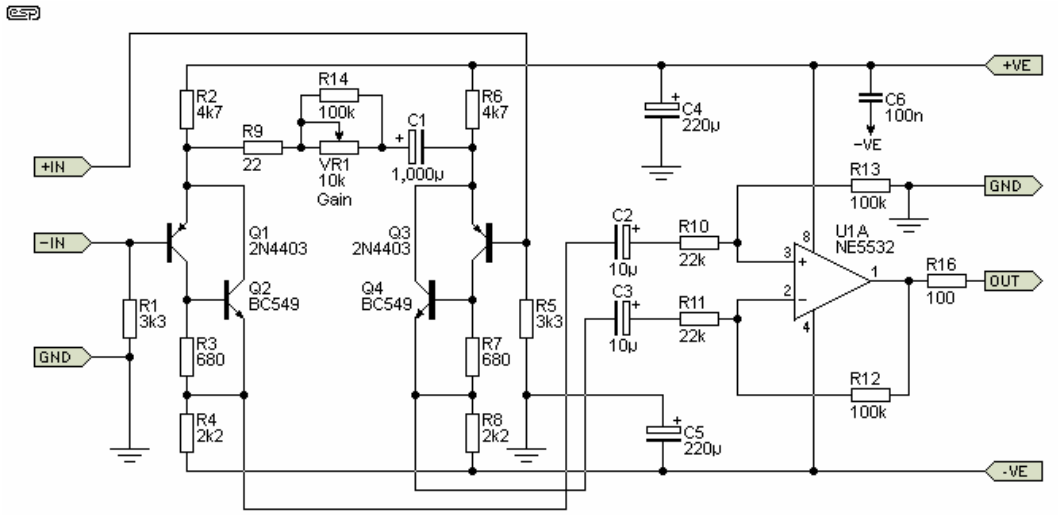


Figura 2.4: Esquema del circuito de pre-amplificación implementado

Capítulo 3

El algoritmo

Como ya hemos visto, el propósito de un octavador es proporcionar una señal una octava inferior a la señal introducida. En consecuencia el algoritmo debe dividir la frecuencia de cada muestra entrante por dos. Lo primero que salta a la vista es la manifiesta relación con la transformada de Fourier para operar en el dominio de la frecuencia, sin embargo, el coste computacional y temporal de implementar esta operación matemática es elevado. Es por ello que se estudian diferentes posibilidades que pudiesen simplificar la arquitectura.

3.1. Aproximaciones al problema

El primer pensamiento es que se trata de algo sencillo: basta con eliminar una de cada dos muestras en el espectro, es decir, un diezmado en frecuencia, para obtener a la salida una señal con las frecuencias divididas por dos. Sin embargo las propiedades de la transformación hacen imposible realizar esta operación: como las frecuencias de entrada son numerosas y variables no basta este diezmado, ya que el desajuste en la fase produce un desplazamiento circular en la señal de salida (ecuación 3.1).

$$Si \quad \mathscr{F}(\{x_n\})_k = X_k \quad Entonces \quad \mathscr{F}(\{x_n e^{\frac{2\pi i}{N} nm}\})_k = X_{k-m} \tag{3.1}$$

La consecuencia es que si la señal de entrada no es una única nota invariante, la salida resulta irreconocible, por lo que hay que descartar inmediatamente este proceder. [GRAFICA DE LA ENTRADA/SALIDA EN ESTE CASO MATLAB] Otra consideración que no hay que dejar de lado es la pretensión de operar en tiempo real. Esto supone que se debe tener en cuenta un mecanismo que permita trocear la señal en conjuntos finitos para poder aplicar la transformación de Fourier. Este proceso, junto con la propia implementación de la transformada va a complicar en gran medida la arquitectura, es por ello que se decide en primer lugar evaluar los algoritmos que no precisan de esta operativa.

3.1.1. Prescindiendo de Fourier

De cara a obtener un pedal de efectos para un instrumento más grave, se podría haber pensado en optar por un octavador ascendente. Esto, aunque parece que plantea los mismos problemas, resulta mucho más sencillo de implementar precisamente por ser 2 el factor de

multiplicación. La fácil solución consiste en elevar cada muestra al cuadrado y realizar un filtrado pertinente que aísle las componentes que nos interesan, de forma que $x_{out} = x_{in}^2$, según vemos en el gráfico [INSERTAR GRAFIO MATLAB]. Como se puede apreciar, se produce un aumento llamativo de los armónicos produciendo una ligera variación en el timbre que podría resultar o no conveniente. Este tipo de efectos recibe el nombre de *enhancer*[1] pudiendo modificar el timbre de forma significativa. Si añadiésemos a este otros efectos como tremolo, reverb o delay, podríamos realizar una aproximación muy válida a un pedal comercial, sin embargo, he preferido mantenerme fiel a la intención original de realizar la octavación descendente.

No obstante, merece la pena probar que ocurre si realizamos la operación opuesta, $x_{out} = \sqrt{x_{in}}$, de forma que se octavara la señal de forma descendente. El resultado es menos halagüeño de lo que pudiera parecer, en primer lugar está el inconveniente de tener que lidiar con las muestras de valor negativo[2], lo que resulta una molestia de cara al flujo de datos. Además, debido a que la entrada está limitada en banda, al reducir de forma cuadrática el valor de los armónicos más agudos, se produce una modificación en el timbre que provoca un sonido *robótico* o *artificial* por lo que se descarta también esta idea.

La última de estas operativas *sencillas* consiste en utilizar las propiedades de la multiplicación por coseno para modular la señal a la altura deseada. Aunque la idea parece sencilla, resulta muy complicado de llevar a la práctica por que habría que implementar un algoritmo que detectara los picos de frecuencia y los modulara utilizando un coseno de valor $f_{pico}/2$. La detección de picos en frecuencia ya supone otra vez la vuelta al dominio de Fourier sin contar con que la gestión de la anchura de esos picos se hace muy compleja. No obstante, el algoritmo de baja latencia que se describe posteriormente en la sección 3.1.1.2 propone algo similar.

3.1.1.1. Algoritmo NFC-TSM

En este momento descubrí gracias una vez más a la charla con José Parera, la existencia de *dafx.com*, que se trata de una página dónde se publican anualmente un gran número de estudios relacionados con los efectos digitales de audio y de donde he obtenido la mitad de las referencias bibliográficas. De la investigación en esta página descubrí un algoritmo que realizaba la octavación descendente sin llevar a cabo la transformada de Fourier, descrito en [TSM].

Este documento propone un ingenioso algoritmo al que llaman Modificación de Escala Temporal por Correlación Normalizada y Filtrada o por sus siglas en inglés *NFC-TSM*. El esquema de funcionamiento es el siguiente; primeramente se lleva a cabo un remuestreo con la tasa deseada $f_{s_original}/f_{s_replay}$ (para realizar la octavación debería ser 2:1). En segundo lugar tiene lugar el proceso de la modificación de la escala temporal que vuelve a variar la escala para obtener una salida de igual tamaño que la entrada.

En las propias palabras de los autores, la idea consiste en descartar y repetir algunos segmentos de la señal para comprimir o expandir la longitud del audio resultante. Utilizan para ello un sistema de buffer circular con dos punteros que se mueven a diferente

[1] Esta operativa me fue propuesta por José Parera, al que agradezco mucho el tiempo que me dedicó desinteresadamente

[2] Como se verá más adelante, el método de entrada en la FPGA devuelve las muestras normalizadas en el rango (-1,1)

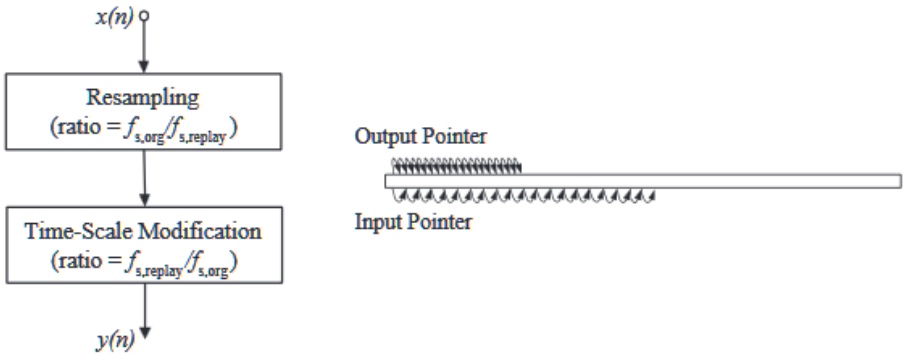


Figura 3.1: Esquema del funcionamiento NFC-TSM en octavación descendiente

velocidad añadiendo algunas variaciones para evitar que ambos colisionen. En consecuencia, los saltos que realiza el puntero más rápido tienen longitud variable y pueden ser en cualquier dirección, puesto que la distancia entre este y el otro puntero no es fija. Además se calcula el mejor punto para realizar el salto mediate AMDF (ecuación 3.2) para evitar una discontinuidad demasiado abrupta que resulte percetible al oído.

$$D(m) = \sum_{k=0}^{L-1} \mid x(k+m) - x(k) \mid \tag{3.2}$$

La función AMDF se utiliza para ajustar el salto del puntero de manera que no se rompa la periodicidad de la señal, la cual podría no mantenerse si el punto de salto fuera aleatorio. Para ello, se compara una ventana de correlación correspondiente a donde se ubica el puntero con un área de búsqueda determinada. El punto de salto se ubica en el mínimo de la función AMDF. La longitud de las ventanas así como su ubicación son parámetros ajustables del algoritmo, tal y como aclaran los autores.

En resumen, este método combina varias operativas para lograr un algoritmo versátil que pueda ajustar al factor de octavación deseado con una calidad relativamente buena en todos. Sin embargo, para llevar a cabo la ejecución de este algoritmo es necesaria mucha capacidad de cálculo en cada instante ya que la AMDF requiere de bastantes operaciones[3], haciéndolo poco deseable para la FPGA. La siguiente solución presentada propone un mismo concepto de cálculo sencillo repetido en muchas ocasiones, lo cual resulta mucho más deseable a la hora de introducirlo en la placa.

3.1.1.2. Algortimo de baja latencia

Para comprender la base de esta operativa, descrita en profundidad en [LLA], hay que diferenciar dos maneras de abordar la problemática del cambio de afinación a grandes rasgos ya que la nomenclatura induce a error:

- **Desplazamiento de tono:** o *pitch shifting*, se basa en que cada frecuencia se **multiplica** por una constante. Este el caso de los algoritmos descritos antes que pretendían dividir todas las frecuencias entre 2.

[3]Estas se detallan en el artículo [TSM] para varios casos distintos

- **Desplazamiento de frecuencia:** o *frequency shifting*, en este caso a cada frecuencia se le **suma** (o resta) una constante definida. Estas técnicas no se han aplicado anteriormente en algoritmos de cambios de afinación porque se rompen las relaciones armónicas entre una fundamental y sus componentes. Sin embargo, esta va a ser la solución que proponen los autores para construir el algoritmo de baja latencia.

De forma equivalente a algunos métodos anteriores, si a cada frecuencia le restamos su frecuencia mitad, $f_{out} = f_{in} - f_{in}/2$, obtenemos también la división por 2. El algoritmo se construye sobre esta idea, la cual, para que esta idea funcione, debe *fijar* las frecuencias entrantes de alguna manera, ya que si no, no podríamos saber a priori qué constante hay que utilizar en cada momento. La solución es utilizar un banco de filtros IIR lo suficientemente estrechos como para que se distingan correctamente dos notas sucesivas. De esta forma, se realiza la resta inmediatamente después de haber hecho el filtrado, como se muestra en 3.2. Conociendo la afinación del saxofón, se pueden conocer de antemano las frecuencias de entrada, por lo que habrá que centrar cada filtro con una de las notas del registro. El resto del espectro correspondiente a los armónicos se cubre con filtros equiespaciados siempre en escala logarítmica.

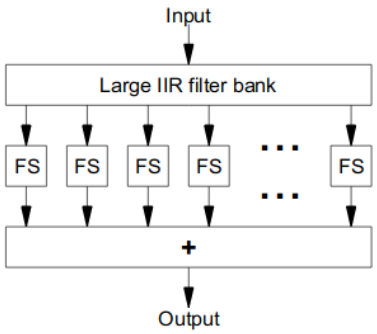


Figura 3.2: Esquema del algoritmo de baja latencia. FS equivale a cada resta de frecuencia

Tal y como proponen los autores es necesario que el ancho de banda de cada filtro vaya aumentando con la frecuencia pero estableciendo un mínimo de 50Hz para las frecuencias más bajas. En esto radica uno de los problemas de este algoritmo y es que, produce inevitablemente una ligera desafinación que se puede acentuar si se pierde la relación entera con los armónicos superiores. Esta desafinación es más pronunciada en las frecuencias inferiores debido al ancho de banda mínimo establecido ya que cada filtro coge una o dos notas.

Para realizar cada etapa de restado, son necesarios dos osciladores a 90º, una transformada de Hilbert y otros pocos componentes más, como ilustra la figura 3.3. Adicionalmente, se puede sustituir la etapa de la transformada de Hilbert por filtros IIR, reduciendo aún más el coste computacional total. No obstante, es cierto que el elevado número de módulos, aunque sencillos, tienen un coste de área grande en la FPGA aunque no resulta crítico.

La razón para no implementar este algoritmo de baja latencia ha sido puramente personal, ya que he priorizado eliminar el desafinamiento a reducir la latencia. En la fuente bibliográfica ([LLA]) hay muestras de audio de buena calidad comparando diferentes métodos, pero ninguno de ellos estaba implementado en FPGA. Es evidente que no se

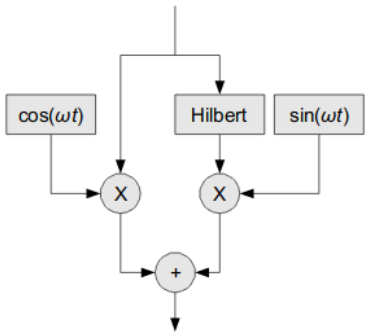


Figura 3.3: Esquema de cada etapa de restado en el dominio del tiempo

puede comparar de igual manera ya que las pérdidas que se producen debido al tratamiento de la señal muestra a muestra en la FPGA no se producen en un procesador. Dado que la implementación no va a ser óptima en ninguno de los casos, me ha resultado preferible implementar un algoritmo más sencillo que reduzca el número de puntos críticos en el mismo.

3.1.2. Retorno a la transformada

Así pues, se va a plantear una arquitectura basada en la idea del *Vocoder de fase* que se presenta en los epígrafes sucesivos.

3.2. Vocoder

Durante todo el siglo XX se han ido desarrollando diferentes técnicas de tratamiento y codificación para la voz, conforme iba la tecnología en aumento. La consecuencia de ello es la aparición de diversos algoritmos que permiten este tipo de operaciones con una carga computacional relativamente baja.

Un *Vocoder*[4] es generalemente cualquier aparato que analiza y/o sintetiza la voz humana para lograr algún objetivo concreto, como compresión de datos, multiplexación o encriptación en la mayoría de los casos.

El Vocoder de canal[5], desarrollado por los famosos *Bell Labs* en 1928, utilizaba varios filtros multibanda seguidos por detectores de envolvente cuyas señales de control se transmitían al decodificador del receptor. Estas señales de control son mucho más lentas que la señal original a transmitir, por lo que se puede reducir el ancho de banda permitiendo a un mismo medio de transmisión soportar un mayor número de canales, ya sea por radio o cable. Finalmente, el decodificador amplifica estas señales de control y las introduce en los filtros correspondientes a cada banda para poder sintetizar de nuevo la señal original. Además de las ventajas sobre el ancho de banda, también se ayuda a proteger la señal para que no se pueda interceptar. Encriptando las señales de control y modificando los parámetros de los filtros, se puede hacer muy difícil su correcta reinterpretación si no se

[4]Del inglés voice (voz) junto a encoder (codificador)
[5]Del inglés çhannel vocoder"

sincronizan el codificador y el decodificador. Esto popularizó su uso durante la Segunda Guerra Mundial en el bando aliado, patentándose diversos diseños.

El concepto se ha mantenido constante durante todo el siglo hasta nuestros días, donde podemos ver implementaciones modernas de la misma idea, por lo que se ha desarrollado una estandarización. La voz humana posee un rango de frecuencias de entre 200 y 3400 Hz típicamente, por lo que se optó por una frecuencia de muestreo de 8 kHz. Es común que se utilice una codificación con 16 bit por muestra por analogía con el estandar CD, pero con utilizar al menos 12 la mayoría de los receptores será capaz reproducir la señal con una fidelidad razonable. Citando un ejemplo, los codificadores según la norma ITU G.729, que son utilizados en telefonía comercial, tienen una buenísima calidad con una tasa binaria de 8 kbps. Actualmente también se utilizan para desarrollar tecnologías relacionadas con la lingüistica, la física y la neurociencia.

3.2.1. Vocoder en la música

Paralelamente a su utilización en comunicaciones, el vocoder se comenzó a popularizar durante la década de los 70 como método de síntesis, ya que esta estaba muy de moda en la época. Cabe mencionar, que durante esta década, surge un gran interés en los músicos por experimentar con diferentes timbres y sonidos en instrumentos conocidos o experimentales. Para aplicaciones musicales, se utiliza una frecuencia portadora proveniente de un instrumento en lugar de extraer la frecuencia fundamental del sonido que se esta grabando. El resultado es una deformación del sonido capturado que, por estar afinado en una nota adecuada, produce un resultado agradable al oído. Fue el primer fabricante de sintetizadores y pionero de la música electrónica, Robert Moog, el que desarrolló un prototipo llamado *Farad* en 1968, pero no fue hasta 1970 cuando unieron el funcionamiento de esta máquina con el sintetizador modular *Moog* que había lanzado previamente al mercado. Quedaba ya conformada la esencia de utilizar la señal proveniente de un micrófono como moduladora y la proveniente de sintetizador como portadora para modularla. Algunos ejemplos tempranos de músicos reconocidos que utilizaron estos dispositivos fueron Phil Collins, Mike Oldfield, Stevie Wonder, Herbie Hancock o Michael Jackson.

Estos vocoder proporcionaban sonidos a los que el público estaba poco acostumbrado pero realmente no mantenían una fidelidad tímbrica respecto el sonido que captaban. Por ello se empezaron a utilizar los vocoder de fase los cuales permiten llevar a cabo expansión o compresión en el tiempo y *Pitch Shifting*, en otras palabras, modificar la altura musical del sonido o afinación sin cambiar la forma de onda que proporciona el timbre característico.

El método para hacerlo es el siguiente. En primer lugar se lleva a cabo una transformada mediante STFT (Short Time Fourier Transform) para posteriormente modificar la afinación mediante sub y sobremuestro. Este proceso hace que el audio resultante no resulte reconocible, por lo que es necesario ajustar el valor de la fase de cada muestra para mantener la coherencia entre ellas, de ahí el nombre de vocoder de fase. Una vez calculadas las muestras, se transforman de vuelta al dominio del tiempo, donde se rellena con ceros para obtener la misma duración que la señal entrante. A continuación se explican en detalle estas etapas.

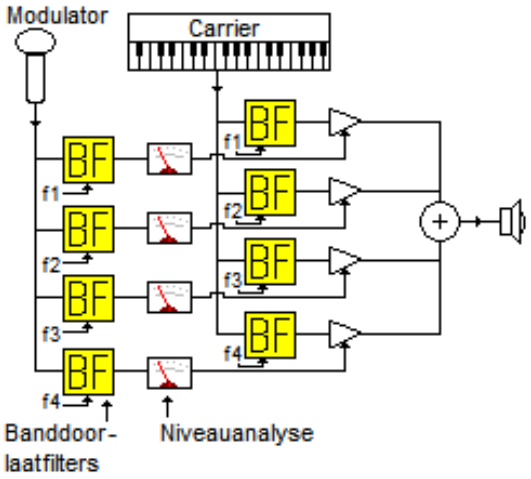


Figura 3.4: Esquema del funcionamiento de un vocoder musical[TRADUCIR]

3.3. Transformación a frecuencia: STFT

Una STFT se usa para determinar el módulo y fase de muestras próximas de una señal mientras cambia con el tiempo, haciéndola muy adecuada para aplicaciones en tiempo real. Para ello, se divide la señal en segmentos más cortos de la misma longitud y se calcula la transformada de Fourier de cada uno de ellos por separado. El método para calcular la transformada es indiferente pero para obtener una latencia baja conviene decantarse por el algoritmo de la Transformada Rápida de Fourier o FFT.

3.3.1. Transformada de Fourier: FFT e iFFT

Para realizar la transformación al dominio de la frecuencia, la opción más adecuada es sin duda el algoritmo de la Transformada Rápida de Fourier o FFT. Este algoritmo calcula la Transformada de Fourier en Tiempo Discreto o DFT descomponiendo la señal original de longitud N en fragmentos de tamaño $N/2$ como muestra la figura 3.5 y posteriormente multiplicarlo por los términos W_n calculados previamente. Nótese que en los bloques de $N/2$ se puede volver a aplicar el mismo principio de forma recursiva. Esto consigue reducir el tiempo de cálculo porque la transformada propiamente dicha se calcula para una longitud mucho menor. En la ecuación 3.3 correspondiente la DFT genérica podemos ver como la complejidad depende cuadráticamente de la longitud de la entrada $O(n^2)$ mientras que la FFT lo realiza únicamente con $O(n * \log(n))$.

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-2\pi kni/N} \qquad Donde \quad k = 0, 1...., N-1 \tag{3.3}$$

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X_k e^{-2\pi kni/N} \qquad Donde \quad n = 0, 1...., N-1 \tag{3.4}$$

El caso de la transformada inversa es totalmente análogo, el algoritmo de la FFT se puede aplicar de la misma forma para realizar iDFT de forma más rápida, lo que se conoce

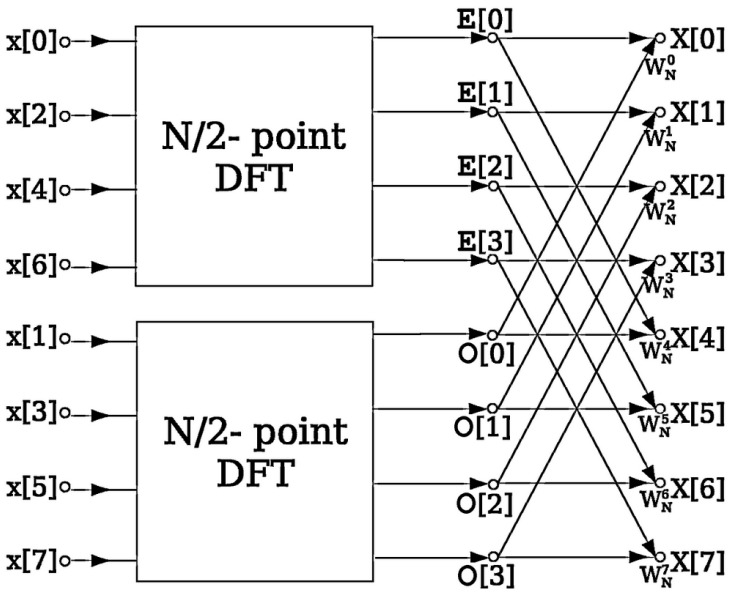


Figura 3.5: Esquema del algoritmo para la realización de la FFT

como iFFT. La ecuacion 3.4 muestra la expresión genérica de la iDFT para un señal de N muestras. Cada uno de los parámetros que se utilizan para realizar las transformaciones se encuentra explicado en el apartado de implementacion HACER LA CITA.

3.3.2. Solapamiento y enventanado

Dividir la señal entrante en sucesivas tramas es un proceso sencillo, únicamente se almacenan las muestras en una memoria para introducirlas posteriormente en el módulo que realiza la FFT. El problema entonces reside en la propia naturaleza de la FFT, ya que esta funciona perfectamente para señales periódicas, pero al trocear la señal en pequeñas tramas, no se garantiza que estas tramas contengan un número entero de periodos. Esto se agudiza especialmente cuando la señal es variante con el tiempo y el número de elementos por trama es independiente de la frecuencia de la señal de entrada.

En la figura 3.6 se ilustra el problema del recortado arbitrario descrito y su clara mejora al aplicarle enventanado. Frente al caso ideal, el primero, el corte introduce componentes en otras frecuencias que se traducirán como un ruido molesto al final de cada trama, conocido en en inglés como *clipping*, al escuchar la transformada inversa. Podemos comprobar como al aplicar una ventana a la señal de entrada, este se hace menor y afecta a menos muestras. Para este ejemplo se ha utilizado una ventana de Hann como la que se utilizará en el prototipo.

Junto al enventanado, se suele aplicar cierto *solapamiento*, es decir, en lugar de empezar a construir una trama a continuación de la anterior, la empezamos a llenar antes de que se haya acabado de llenar la anterior, repitiendo una misma muestra en una o varias trama sucesivas. De esta forma, las tramas están enlazadas entre ellas evitando una discontinuidad abrupta.

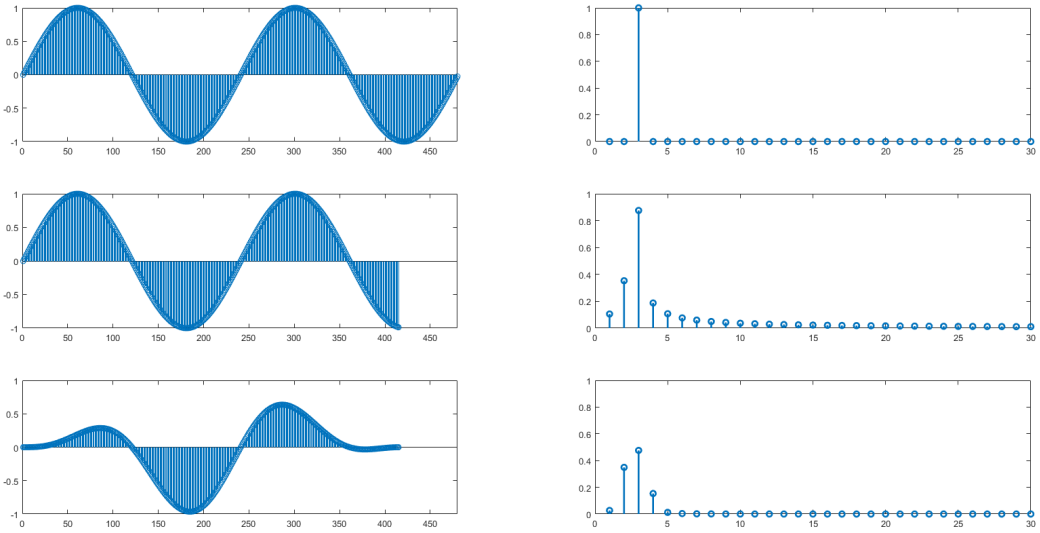


Figura 3.6: Señal sinoidal y su FFT con periodos enteros, recortada y enventanada

Generalmente se cuantifica este proceso mediante un *factor de solapamiento, fs,* expresado en tanto por ciento. Si una trama t de longitud $n = 100$ muestras tiene un solapamiento de 15 %, las primeras $n * 15\,\% = 15$ muestras de t son idénticas a las 15 últimas de la trama anterior, t-1, y así sucesivamente.

Lógicamente, cuando aumentamos el factor de solapamiento más muestras procesamos y en consecuencia, menos eficiente es el algoritmo, puesto que se procesa información redundante. Este desperdicio de recursos en el procesado supone un compromiso doble con las prestaciones. Por un lado, cuanto más disminuya esta eficiencia, más aumentará la latencia, ya que habrá que esperar al cálculo de la siguiente trama para poder finalizar la construcción de la trama presente. Por otro lado, resulta mucho más complejo de cara a la temporización en su implementación.

Como conclusión, debemos elegir un factor de solapamiento $fs > 50$ para que resulte

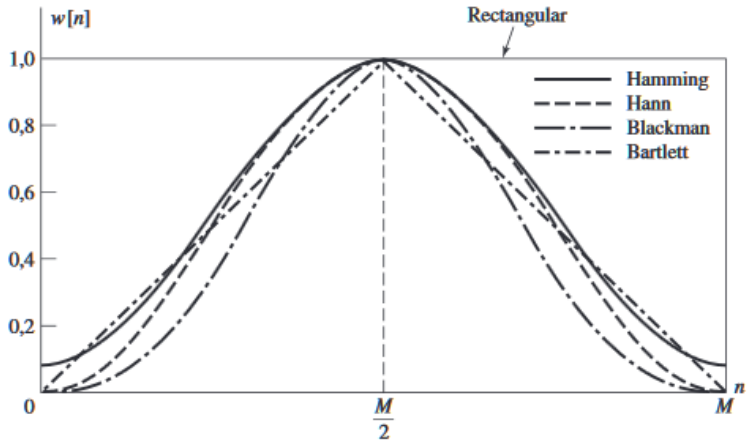


Figura 3.7: Comparativa de las ventanas más utilizadas

práctico, ya que si no el efecto es demasiado sutil como para que merezca la pena dedicar esfuerzo a su implementación en la arquitectura. Tras un modelado en Matlab, he implementado finalmente un valor de $fs = 75\,\%$ tal y como recomienda Ellis [Ell] en su versión del vocoder de fase.

Como ya se ha visto antes, el solapamiento se utilizará junto con un enventanado de Hann cuya expresión se recoge en 3.5. Esta ecuación resulta sencilla de implementar y su uso está muy extendido para aplicaciones de audio en tiempo real frente a algunas de sus alternativas vistas en la figura 3.7. El propio Ellis utiliza en su algoritmo [Ell] una ventana de estas características.

$$H(n) = 0{,}5 * \left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \tag{3.5}$$

Estos mismos conceptos se utilizarán de forma completamente análoga en la etapa de la transformada inversa, tras introducir las muestras en el módulo iSTFT. La única excepción es que para mantener la amplitud de las muestras en la salida igual que las de la entrada, hay que aplicar un factor de escala de 2/3. En la práctica, lo que se hará será aplicar este escalado a los valores previos quedando una ventana de Hann reducida por el factor mencionado, permitiéndonos ahorrar una multiplicación y acelerar el flujo de datos.

3.4. Operaciones sobre la fase

A grandes rasgos este algoritmo opera realizando un diezmado en frecuencia de factor 2, es decir, eliminando una de cada dos frecuencias entrantes, y posteriormente modifica la fase para evitar el desplazamiento circular mencionado en 3.1.1. El procesado se realiza por *tramas consecutivas pares*: dos tramas se agrupan para formar una pareja de la que

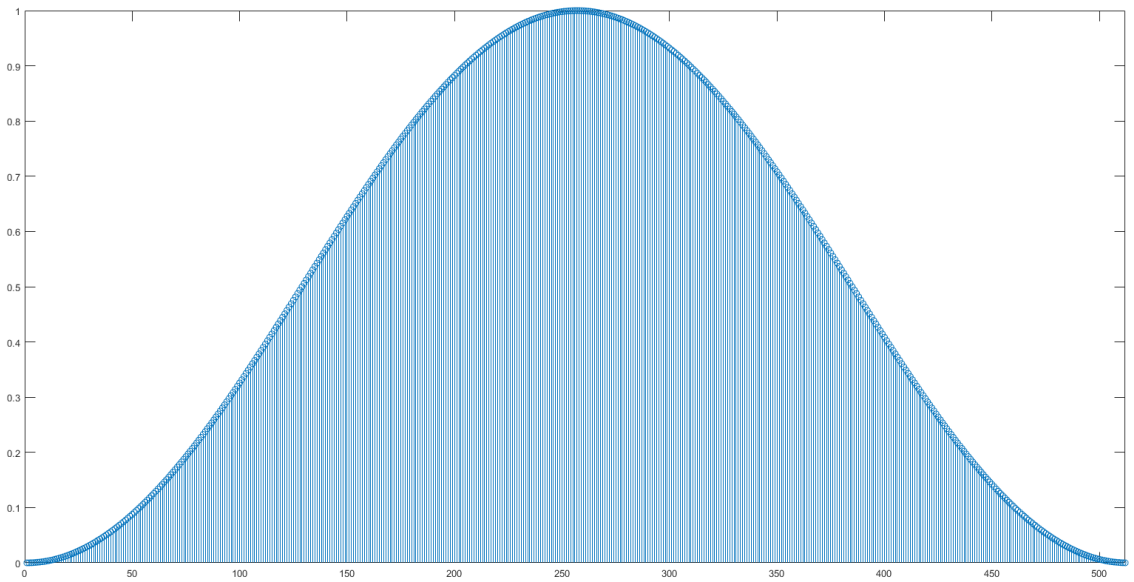


Figura 3.8: Ventana de Hann para $N = 512$ en Matlab

únicamente se va a poder calcular el valor de una sola trama de salida. De esta forma, la primera trama se agrupa con la segunda, la tercera con la cuarta y así sucesivamente.

3.4.1. Cálculo del módulo

El primer paso consiste en obtener el módulo de las muestras. Tras repasar el algoritmo, se puede ver que solo es necesario calcular este valor para una de cada dos muestras, ya que la otra es la que será diezmada y por tanto, no merece la pena malgastar recursos en esta operación.

El cálculo del módulo a partir de la forma binomial es sencillo: $mod = \sqrt{re^2 + im^2}$ donde re es la parte real e im la imaginaria. Sin embargo, la implementación en FPGA de una raíz cuadrada no resulta inmediata. Una opción es utilizar un módulo de cálculo que procese los datos y devuelva la función raíz mientras que otra es utilizar *lookuptables* para consultar el resultado de unas entradas previamente definidas. Este último método no es práctico ya que el tamaño de la memoria ROM que almacenaría esos datos tendría que ser demasiado grande, además Vivado proporciona algunos módulos IP que realizan estas operaciones matemáticas, entre otras, a cambio de algunos ciclos de procesado.

En este caso, resulta más práctico realizar una aproximación que permita reducir el tamaño en área de la implementación así como la latencia de la misma, tal y como señala el doctor P.Chu en [Chu]. Si aproximamos por tanto según 3.6, *donde* $x = max\{|a|, |b|\}, y = min\{|a|, |b|\}$, logramos simplificar en gran medida esta operación utilizando solo operaciones sencillas.

$$\sqrt{a^2 + b^2} \approx max\{((x - 0,125x) + 0,5y), x\} \tag{3.6}$$

3.4.2. Recalculando la fase

Una vez tenemos el módulo calculado, podemos calcular la nueva fase de cada muestra teniendo en cuenta que necesitamos la información de una trama t y de la siguiente, $t+1$. El método para calcular la fase partiendo de la forma binomial es $\arctan(im/re)$ por lo que en este caso, si compensa utilizar un core IP de Vivado que realice esta operación, ya que ésta y otras operaciones trigonométricas serán necesarias más adelante.

Una vez calculada la fase de cada muestra entrante n_t, procedemos a obtener la fase de salida que se aplicará a cada muestra de salida n'_t. Para lo que calculamos una variable, dp, que se irá acumulando con n según 3.7.

$$dp = fase(n_{t+1}) - fase(n_t) - dphi \tag{3.7}$$

En esta fórmula se puede observar la existencia de una constante $dphi$ que habrá de valer $dphi = (\frac{n\pi}{2})^6$ que se calcula aparte y se introduce en una ROM para utilizarla a lo largo del procesado. Aunque pueda parecer que el valor resultante puede llegar a ser muy elevado debido al carácter de esta constante, el siguiente paso será reducirlo al intervalo $(-\pi, \pi)$, eliminando el problema.

[6] Esta n también se refiere al número de muestra dentro de una trama t. Hay que recordar que $0 \leq n \leq$ *puntos de la transformada*, 511 en este caso

Esta constante dp va a servir para actualizar el valor de la fase **de la muestra siguiente** $n+1$ operando según 3.8, de forma que la fase de cada muestra depende de la anterior dentro de la misma trama.

$$fase(n'_t + 1) = fase(n_t) + dp + dphi \qquad con\ fase(n_t) = 0 \ \ si\ n = 0 \tag{3.8}$$

Tras este proceso, se puede preparar la muestra para iniciar la transformada inversa y posteriormente reenventanar la señal de salida cuando se procesen el resto de las muestras necesarias, como se ilustra en [INCLUIR DIAGRAMA DE BLOQUES DE LA PARTE TRANSFORMADA].

Para concluir, podemos afirmar que el algoritmo que se va a implementar es una adaptación del vocoder de fase propuesto por Ellis ([Ell]), de forma que este orientado a la octavación y sea lo más adecuado posible para introducirlo en la FPGA. El código de Matlab que se ha utilizado para probar este algoritmo se encuentra adjunto a este documento como anexo.

Capítulo 4

Implementación

Una vez definido el algoritmo, es necesario definir la implementación que se va a llevar a cabo teniendo en cuenta las ventajas y limitaciones de la arquitectura. En este caso, orientar el diseño a una FPGA va a condicionar en gran medida las decisiones que se van a tomar llegado a este punto.

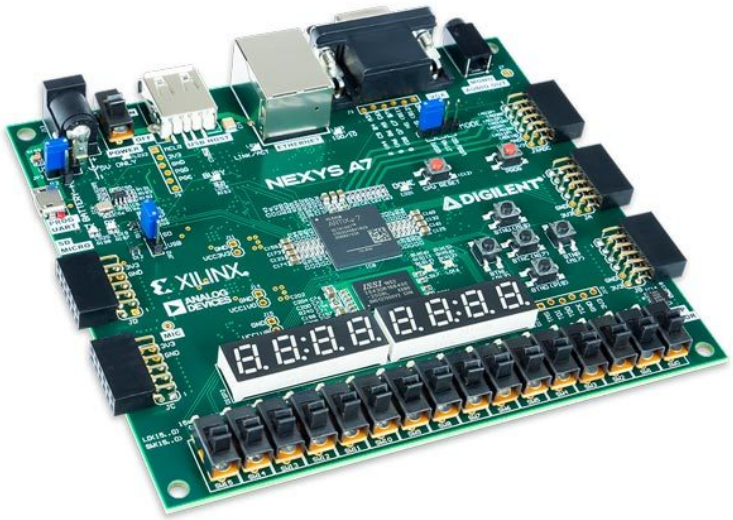


Figura 4.1: Placa Nexys A7

Para el montaje de prototipo, se va a utilizar la placa proporcionada por el departamento: Nexys A7 de Digilent (figura 4.1). Esta placa monta una FPGA de la familia Artix-7 modelo *XC7A100T-1CSG324C* junto con múltiples opciones para conectividad, como son los puertos VGA, mini USB y Ethernet y para la entrada y salida de datos, como el micrófono, slot para microSD y sensor de temperatura y acelerómetro. Además incluye varios LED, que junto a los botones y conmutadores, permiten un cómodo manejo del conjunto de la placa. El conjunto de las prestaciones será más que suficiente para probar más adelante el funcionamiento de las características del prototipo.

De especial trascendencia para la puesta en marcha del conjunto, serán las bahías de pines que se ubican en ambos laterales de la Nexys, puesto que permiten conectar y leer

los voltajes de entrada y salida que se van a emplear. La documentación proporcionada por Digilent [Nex] ha resultado muchas veces insuficiente, pero fundamental a la hora de poner en marcha el prototipo.

4.1. Gestión entrada-salida

Siguiendo el flujo de datos desde la etapa analógica, el siguiente paso consiste en introducir los mismos en la FPGA. Para ello, es imprescindible convertir el voltaje analógico en digital mediante el uso de un *Conversor Analógico a Digital*, en lo sucesivo ADC. La Nexys A7 no incorpora ninguno integrado por lo que habrá que conectarlo externamente. Adelantando los acontecimientos, será necesario también un *Conversor Digital a Analógico*, o DAC, para reconvertir a tensión analógica la salida de audio procesado. Por ello, se decide buscar ambos conversores conjuntamente, ya que se simplifica en gran medida la implemtación, ya que si son del mismo fabricante, suelen tener especificaciones de funcionamiento similares.

Exiten infinidad de este tipo de componentes en el mercado, todos ellos ideados para operar en diferentes condiciones de trabajo, en diferentes formatos y a un precio muy asequible. En el caso de las señales de audio, la mayor especificación que deben de cumplir es el compromiso de la tasa de muestreo t_s (o más frecuentemente su inversa, la *frecuencia de muestreo* F_s) para no corromper la señal entrante y preservar su calidad. Típicamente se utilizan algunos valores ya estandarizados por las grandes compañías de audio a lo largo del siglo XX:

- $F_s = 8kHz$**:** Utilizada especialmente para telefonía comercial pero no para componentes de audio de carácter musical.
- $F_s = 22050Hz$**:** Frecuencia de muestreo típica de la radio que permite reproducir señales con componentes máximas de hasta 10kHz.
- $F_s = 32kHz$**:** Se utiliza no tan frecuentemente en varios formatos de video digital, como el miniDV.
- $F_s = 44,1kHz$**:** La más extendida en formatos como MP3, MPEG y CD por razones tanto históricas como prácticas. Puesto que un oído joven es capaz de percibir tonos de hasta 20kHz aproximadamente, esta tasa se estableció tras aplicar el criterio de Nyquist dejando un margen suficiente para compensar las etapas de filtrado posteriores.
- $F_s = 48kHz$**:** También muy utilizada en televisión digital, DVD y audio profesional.
- $F_s = 96o192,4kHz$**:** Pensada especialmente para audio de alta definición en formatos como HD-DVD y Blue-Ray Disc

Tras analizar las posibilidades, resulta evidente que para una aplicación de audio musical conviene establecer la frecuencia de muestro en 44,1 o 48 kHz de forma que conserve cierta similitud con los equipos comerciales y profesionales del mercado.

Así, aprovechando su reciente adquisición por parte del departamento, se va utilizar un componente que integre tanto ADC como DAC y que permita trabajar a estas velocidades: el *Pmod i2s2* también de Digilent.

4.1.1. Pmod i2s2

Este componente trae, en definitiva todo lo necesario para este proyecto: junto a los ADC y DAC incorpora dos puertos para mini-jack estéreo hembra[1] y una serie de pines dispuestos de tal forma que la conexión con la placa es inmediata. Se pueden observar estas características en la imagen 4.2.

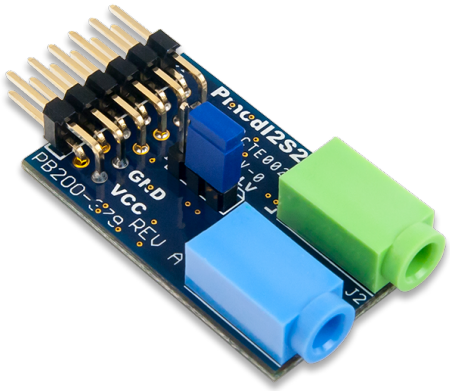


Figura 4.2: Detalle del Pmod i2s2

Lógicamente, al ser ambos placa y *plug-in* fabricados por Digilent, no hay ningún problema para conectarlos entre sí, basta con conectarlo a una de las bahías de pines que tiene la FPGA. Además, las conexiones de alimentación, Vcc y GND, se encuentran hechas de serie en la Nexys, por lo que no es necesario añadirlas en el fichero de conexiones o *constraints*.

Tanto el modelo de ADC, *Cirrus CS5345*, como el de DAC, *Cirrus CS4344* Hablar de las caracteristicas de ADC y DAC

4.1.2. Consideraciones de implementación

hablar de los relojes del vivado y sus problemas archivo de constraints

4.2. Controlador de datos

h

[1] Lo más extendido para audio, puesto que son los conectores que llevan móviles, auriculares. ordenadores, etc...

4.2.1. Bancos de memorias

h

4.2.2. Core FFT

h

4.3. Controlador global

h

4.3.1. Displays

h

Capítulo 5

Pruebas y depuración

Pruebas y tal y cual

Capítulo 6

Conclusiones y trabajo futuro

Apéndices

Apéndice A

Código del algoritmo en MATLAB

codigo aqui

Apéndice B

codigo aqui

Bibliografía

[Ell] D.Ellis, *A Phase Vocoder in Matlab*, LabROSA at Columbia University, Marzo 2003: http://www.ee.columbia.edu/d̃pwe/LabROSA/matlab/pvoc/

[Opp] A.V.Oppenheim, R.W.Schafer, *Discrete-Time Signal Processing.* Pearson Education, 2011

[TSM] A.Haghparast, H.Penttinen, V.Välimäki, *Real-Time Pitch-Shifting Of Musical Signals By A Time-Varying Factor Using Normalized Filtered Correlation Time-Scale Modification (NFC-TSM).* Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07), Bordeaux, France, September 10-15, 2007

[Nex] Digilent, *Nexys A7 Reference Manual.* https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/ reference-manual

[ADC] Cirrus Logic, *CS5343-44.* https://www.cirrus.com/products/cs5343-44/?_ga=2.264355407.117367827.1557737954-871020677.1554307733

[DAC] Cirrus Logic, *CS4344/45/48.* https://www.cirrus.com/products/cs4344-45-48/?_ga=2.264355407.117367827.1557737954-871020677.1554307733

[Aud] Audio Technica, *What's the pattern?* https://www.audio-technica.com/cms/ site/aa901ccabf1dfc6b/index.html

[LLA] N.Juillerat, S.Schubiger-Banz, S.Müller Arisona, *Low Latency Audio Pitch Shifting in the Time Domain* ICALIP 2008 - Proc. of the IEEE International Conference on Audio, Language and Image Processing, Shanghai, China; pp.29 - 35.

[PAm] P.Allison, *Low Noise Balanced Microphone Preamp.* Edited by R.Elliot for Elliot Sound Products ESP, 2008. http://sound.whsites.net/project66.htm

[Chu] P.P.Chu, *RTL Hardware Design Using VHDL.* Ed. Wiley Interscience, 2006