Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería de Tecnologías de Telecomunicación

Trabajo Fin de Grado

Implementación de un sintetizador granular en FPGA

Autor: Enrique Ramo Bonet

Tutor: Ignacio Bravo Muñoz

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado de Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Implementación de un sintetizador granular en FPGA

Autor: Enrique Ramo Bonet

Tutor/es: Ignacio Bravo Muñoz

TRIBUNAL:

Presidente: Raúl Mateos Gil

Vocal 1º: Miguel Ángel García Garrido

Vocal 2º: Ignacio Bravo Muñoz

FECHA:

Índice

Índice de figuras

Índice de tablas

Resumen

En el presente trabajo se detallan los pasos seguidos para la implementación de un sintetizador granular en la tecnología FPGA (Field Programmable Gate Array). Dicha técnica de síntesis consiste en generar texturas o tonalidades únicas mediante la manipulación de pequeños fragmentos de sonido llamados granos. Su implementación en FPGA consistirá en desarrollar un manejo de memoria para obtener dichas unidades de sonido, aplicar una envolvente a cada una de ellas, y generar así un flujo de datos que será modificado mediante un bloque de filtrado y una envolvente ASR (Attack Sustain Release). Finalmente, se extraerá el flujo a través del protocolo I2S (Integrated Interchip Sound), y se permitirá al usuario modificar los parámetros internos del bloque mediante los elementos Hardware de la placa.

Palabras clave:

- FPGA
- Síntesis granular
- PMOD
- Protocolo I2S

Abstract

In the present work, the steps followed to implement a granular synthesizer on a FPGA board will be presented. This synthesis technic consists in taking small audio fragments called Grains and manipulate them in such a way that unique textures and timbres are obtained. The FPGA implementation will consist in developing a memory management block to obtain the grains, then applying an envelope to each of them, creating a data stream that will be processed by a filter and an ASR envelope (Attack Sustain Release). After that, the sound will be externalised with the I2S protocol, and the internal parameters of the synthesis will be modifiable by the user via the board Hardware.

Key words:

- FPGA
- Granular Synthesis
- PMOD
- I2S Protocol

Resumen extendido

En el siguiente trabajo se detallarán los pasos seguidos para la implementación de un sintetizador granular basado enteramente en tecnología FPGA, más específicamente la tarjeta Nexys 4 DDR de la empresa Digilent.

La técnica de síntesis granular nació en 1946 gracias al físico Dennis Gabor [1], y fue utilizada de manera musical por primera vez por el compositor Iannis Xenakis. Este método consiste en la utilización de fragmentos de entre 1 a 50 milisegundos, pudiendo llegar hasta los 150 milisegundos, provenientes de un sonido base. Dichos pedazos, llamados granos, debido a su reducido tamaño, pasan a ser indistinguibles del total que los contenía pudiendo utilizarse como elemento base para la generación de nuevas texturas y timbres sonoros, al ordenarse y combinarse mediante distintas técnicas.

Dentro de las variantes de síntesis granular que han sido planteadas, para este diseño se utilizará la llamada "síntesis granular síncrona" y basada en "samples". La primera quiere decir que la posición de los granos en el flujo de datos será determinada, no variará con el tiempo. Que esté basada en samples quiere decir que la fuente de datos será un sonido determinado precargado en memoria.

La primera fase del diseño consistirá en la creación del bloque de control de memoria. Este se comunicará con una memoria ROM (Read Only Memory) de dos puertos (utilizando la tecnología Block RAM), permitiendo así obtener dos flujos de granos que puedan superponerse. Debido a la naturaleza de este método de síntesis, una envolvente debe ser aplicada a cada uno de los granos, para evitar cambios bruscos en los límites entre unidades, que producirían lo que se considera como "Clicking". Los valores de referencia de dicha envolvente se almacenarán en otra memoria ROM de doble puerto, de la que se extraerán los datos de manera sincronizada con la extracción de granos para que, posteriormente, mediante dos multiplicadores, se obtengan los dos flujos independientes de granos, que posteriormente se sumarán y conformarán la señal a modificar por los sucesivos bloques del diseño.

Posterior a dicho bloque, se encuentra la envolvente de ASR (Attack, Sustain, Release), que se encargará de manejar la amplitud de la señal según toque la tecla el usuario, y cuando la suelte. Dicha envolvente estará regida por una simple máquina de estados, basada en la implementación [2].

El próximo bloque consiste en el apartado de filtrado del instrumento. Se utilizará un SVF (State Variable Filter) [3], muy usado en aplicaciones de audio por permitir, con un solo algoritmo, obtener un filtro paso bajo, paso alto, paso banda y banda eliminada.

Por último, se desarrollarán los bloques de interfaz de usuario. Estos serán los encargados de comunicarse con los elementos Hardware PMOD de la empresa Digilent y modificar los parámetros internos que presente el instrumento. Para la generación de la señal de audio se utilizará el PMOD I2S2 [4]. Se comunicará con dicho dispositivo mediante el protocolo I2S, y se generará la señal sonora, extraíble por salida Jack estándar y lista para ser conectada a cualquier altavoz o auricular. Además, utilizando

los elementos Hardware de la placa, además del PMOD ENC, se permitirá al usuario modificar los parámetros internos del dispositivo.

Para el análisis previo de los algoritmos y la obtención de distintos datos clave del diseño se hará uso del programa Matlab, así como su extensión Simulink. Tras dicha simulación, se implementará en el software Vivado, de Xilinx, utilizando el lenguaje VHDL.

1. Introducción

1.1 Estructura del trabajo

El presente trabajo contará con los siguientes apartados:

- Introducción: En ella se detallarán los objetivos a conseguir, así como una breve historia tanto de los sintetizadores como de la tecnología granular.
- Estado del arte: En este apartado se estudiarán los actuales sintetizadores creados mediante FPGA, así como los distintos usos que presenta la síntesis granular y su implementación en entornos Hardware, para así conocer lo ya construido y asentar las bases del desarrollo en base a lo que se va a aportar con el mismo.
- Desarrollo del sintetizador: en esta sección se explicarán las bases del desarrollo del instrumento, las consideraciones previas que se han tenido en cuenta y la teoría detrás de cada bloque, así como el funcionamiento del código desarrollado.
- Resultados: se mostrarán las distintas simulaciones realizadas sobre el sistema, para demostrar la viabilidad y el funcionamiento de cada uno de los bloques, así como del sistema completo. Además, se aportarán pruebas del funcionamiento del sistema al descargarlo en la placa.
- Conclusiones y trabajo futuro: se resumirá cual ha sido el resultado general del diseño, si se ha cumplido con las expectativas, y que se podría hacer a futuro para mejorar el dispositivo.

1.2 Objetivos por conseguir

En el siguiente trabajo, se pretende desarrollar un sintetizador granular basado en la tecnología FPGA. Para ello, se fijan los siguientes objetivos a cumplir:

- Estudiar a fondo la síntesis granular, los usos que tiene actualmente, que elementos Hardware y Software existen con esta tecnología y entender los distintos parámetros que posee para la implementación.
- Comprobar la viabilidad de esta tecnología de síntesis en dicha plataforma, al ya estar más que comprobado para otros métodos como la sustractiva o la modulación en frecuencia, y crear un instrumento que sea capaz de generar texturas sonoras atractivas al usuario.
- Desarrollar bloques presentes en la gran mayoría de sintetizadores estándar, siendo estos la envolvente ASR y el filtro de salida.
- Desarrollar un bloque que interactúe con el Hardware para permitir al usuario modificar en tiempo real, aún mientras se genera el sonido, los distintos parámetros internos de síntesis.
- Conseguir obtener audio para poder cerciorarse de los resultados de la implementación, no solo desde un punto de vista técnico, sino desde la perspectiva artística, y comprobar que el sistema al completo genera un sonido interesante, cohesionado, y, sobre todo, que el jugar con los distintos parámetros se haga ameno.

1.3 Historia de los sintetizadores

Un sintetizador se define como aquel instrumento que utiliza elementos electrónicos para la generación y procesado de señales auditivas. Se dividen en dos grandes grupos: sintetizadores analógicos y digitales. Los primeros sintetizadores de la historia pertenecían al primer tipo, como el Trautonio [5], creado en 1924, y que utilizaba lámparas de neón de bajo voltaje para generar una onda y posteriormente filtrarla, siendo uno de los primeros ejemplos de síntesis sustractiva, o la primera generación de sintetizadores Moog [6], que ya utilizaban elementos que a día de hoy se consideran estándar, como osciladores controlados por voltaje (VCO) ya basados en transistores o filtros más precisos.

Si bien hoy en día se siguen produciendo y utilizando sintetizadores de carácter analógico, los digitales han crecido exponencialmente desde su primera ideación en 1973. A día de hoy, estos instrumentos son implementados mayoritariamente en entornos Software, llegando incluso a existir programas que recrean fielmente el sonido de sintetizadores antiguos, ya sean analógicos, como el Minimoog [7], o digitales, como el programa Dexed [8], que emula al famoso DX7 de Yamaha, uno de los instrumentos más prósperos de los ochenta.

A pesar de los avances en la industria del Software, debido a las limitaciones técnicas que poseen los ordenadores portátiles, muchos músicos necesitan hoy en día sintetizadores Hardware, y poder así utilizar su computadora como un mero organizador de todos los instrumentos, dejando el procesamiento y generación del sonido a unidades dedicadas expresamente a ello, permitiéndoles así expandir sus herramientas en eventos en vivo.

1.4 Orígenes de la síntesis granular

La síntesis granular fue utilizada por primera vez para crear una pieza musical en 1997 por el músico griego Iannis Xennakis, en su obra Analogique B. Para su creación, Xennakis separó distintos fragmentos de cinta magnética con pequeños eventos musicales, para posteriormente unirlos minuciosamente, creando así la primera composición basada en la “granulación” del sonido [9].

Si bien Xennakis sería el pionero en la utilización de la técnica de síntesis granular en el ámbito musical, el físico Dennis Gabor fue el primero en definir la teoría que subyace a la granulación. Aplicó el razonamiento de la física cuántica al sonido [1], es decir, suponía que este no solo se podía descomponer en elementos frecuenciales, como definía la transformada de Fourier, cuya aplicación no era válida para señales con frecuencias variables en el tiempo, sino que se podía separar en pequeños fragmentos denominados “quanta” de mínimo 10 milisegundos, que de por sí solos no son audibles, no más que un pequeño “click” o sonido percusivo, pero son el elemento de construcción fundamental del sonido del que provienen.

El objetivo de Gabor distanciaba mucho de crear un método de síntesis musical, si no crear un método matemático que permitiese reducir el ancho de banda necesario para telecomunicaciones de voz. Es por ello por lo que una de las aplicaciones que primero ideó este físico era la de un dispositivo que permitiese modificar la frecuencia de una señal. Funcionaba de la siguiente manera: la máquina poseía una pieza rotatoria sobre la que se habían realizado unas hendiduras, que harían las veces de ventana de envolvente. Enfrentada a la pieza giratoria se encontraba una pista de audio en cinta, y una lámpara. En el centro de la pieza con hendiduras, se encontraba un fotodetector, que convertiría la pista de audio que iluminase la bombilla en audio. El útil creado por Gabor es representado en la figura 1.

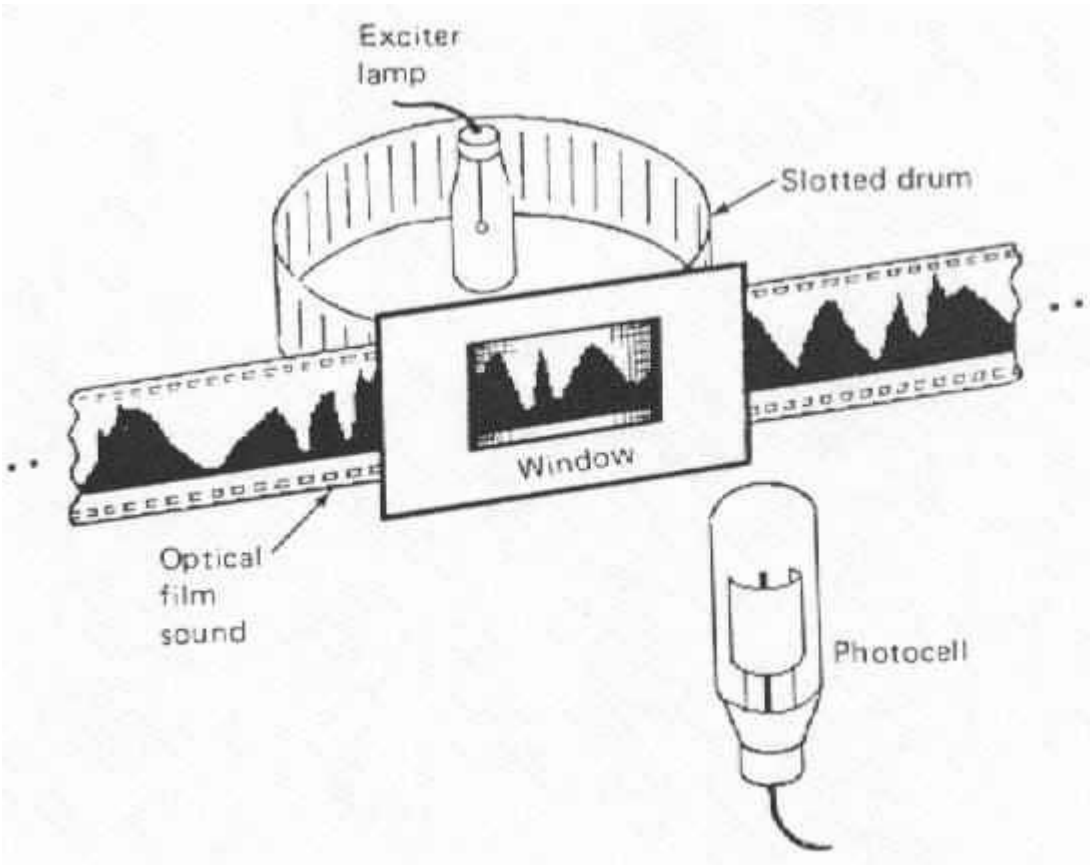


Figura 1: Útil para modificación de frecuencia de audio de Dennis Gabor [10]

Tras ajustar minuciosamente distintos parámetros, como la velocidad del tambor, o la distancia entre ranuras, Gabor constató que, para evitar artefactos y obtener una fiel reproducción del audio que se pretendía modificar, los granos creados debían tener una superposición determinada, tal y como se observa en la figura 2. Al separar con la adecuada distancia las distintas ranuras, la suma de los granos generados por cada una de ellas conseguían generar un sonido uniforme, sin artefactos, y con el mismo contenido espectral que la señal original. Sin embargo, al separarlos demasiado, se producían discontinuidades entre cada uno de los granos, perdiendo así fidelidad y capacidad de reconstrucción del audio base.

Si bien la intención de Gabor se alejaba del mundo de la música en gran medida, su máquina de variación de frecuencia supondría el primer ejemplo de "Pitch-Shifting" o cambio de tonalidad basada en síntesis granular, uno de los usos más extendidos, incluso hoy en día, de este tipo de procesado de señal, como se verá en el apartado 2.2.

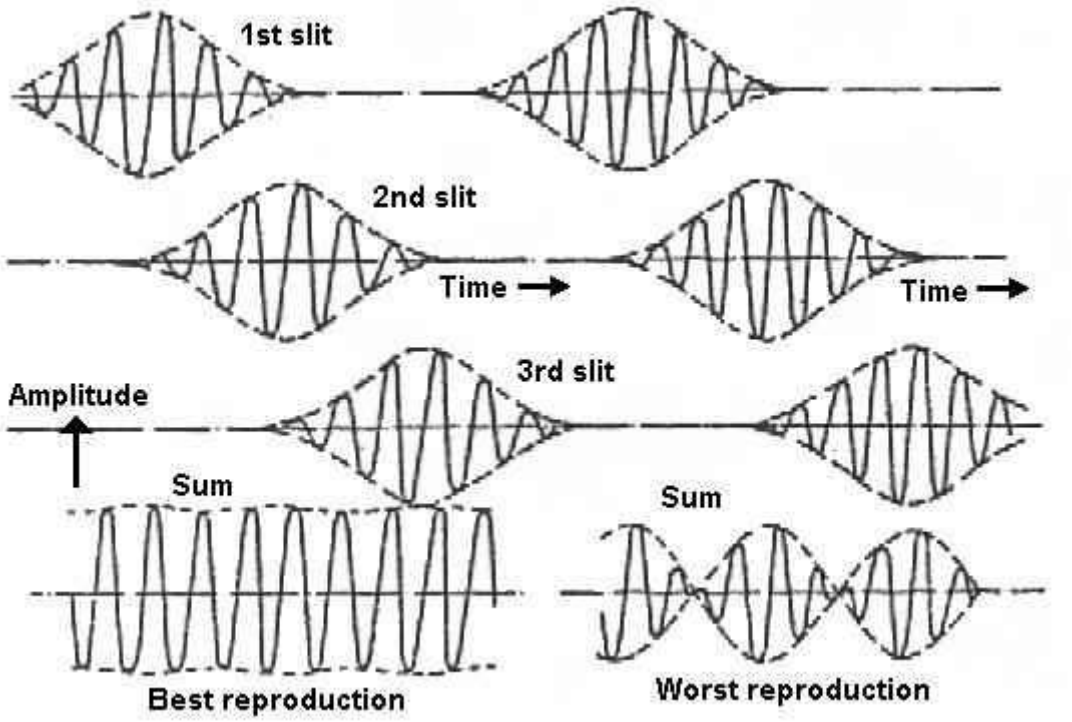


Figura 2: Diagrama de superposición de máquina de Gabor [10]

Tras Xennakis, aparecieron nuevos músicos que se lanzaron a utilizar este método de síntesis, como Curtis Roads [11], quien creó una herramienta llamada Cloud Generator, un software que servía para enseñar los principios de la granulación, o Barry Truax, quien ideó la primera implementación a tiempo real [12].

2. Estado del arte

2.1 Sintetizadores Hardware basados en FPGA

Como se comentaba en el apartado 1.1, a pesar de los avances en sintetizadores Software, el Hardware de producción musical no se queda atrás, y sigue en uso a día de hoy.

Las tarjetas FPGA (o Field Programable Gate Array) están a la orden del día en cuanto a procesamiento de señales digitales se refiere, pero no han acabado de despegar en el mundo de los sintetizadores y todavía existen pocos sistemas High-End que estén basados o que utilicen en gran medida este tipo de tecnología.

Un ejemplo de sintetizador que utiliza una FPGA es Peak, de la empresa Novation [13], que se puede ver en la figura 3. Este instrumento posee una arquitectura mixta, es decir, posee elementos analógicos, como los filtros, para aportar calidez y un control menos lineal obteniendo así un sonido más orgánico, y digitales.

Una de las razones para elegir crear este sintetizador utilizando tecnología FPGA fue aumentar considerablemente la frecuencia de muestreo utilizada para los osciladores, alcanzando los 24 MHz de reloj, tanto en su generación como en los DAC (convertidor digital-analógico) previos a la etapa analógica, evitando así cualquier grado de antialiasing que pudiese generarse ante cualquier tono, y creando así un sonido prácticamente analógico.

Los osciladores utilizan dos tecnologías distintas, controlados numéricamente (NCOs) o Wavetables. La primera tecnología, que simplemente generará las señales mediante contadores, es utilizada para crear formas de onda cuadradas, triangulares y de diente de sierra. En cambio, las Wavetables son formas de onda guardadas en memoria que son utilizadas para producir formas de onda no lineales, como las senoidales.

Figura 3: Novation Peak [13]

Además de los osciladores, los efectos también han sido implementados sobre la FPGA con una frecuencia de muestreo de 90kHz, de nuevo, para aprovechar la potencia de la FPGA y obtener un sonido lo más limpio posible tras la etapa analógica del sintetizador

Otro de los sintetizadores que abiertamente ha presumido de utilizar la tecnología FPGA como pilar fundamental de su procesado y generación de señal es Kyra, de la empresa Waldorf [14], representado en la figura 4. Si bien la empresa no detalla los aspectos técnicos detrás del instrumento, sí que lo catalogan como el primer sintetizador basado enteramente en FPGA.

Figura 4: Kyra de Waldorf [14]

En resumen, ambos instrumentos utilizan esta tecnología por su potencia de procesamiento, comparable a un ordenador de sobremesa, y por sus altas velocidades de reloj, que permiten al sintetizador no escatimar en frecuencia de muestreo, garantizando así un sonido sin artefactos en todo el rango dinámico.

Además de los sintetizadores de comerciales, existen algunos “caseros” basados en FPGA, como el Tiny Synth [2], un instrumento de síntesis sustractiva cuya implementación es una demonstración de las capacidades de la FPGA de operar al mismo nivel que los DSPs más utilizados, incluyendo el estándar de comunicación MIDI (Musical Instrument Digital Interface), una interfaz serie muy utilizada en el mundo de la producción que permite transmitir la nota que se desea tocar, la amplitud de dicha nota, y otros parámetros desde una gran variedad de teclados estándar hacia el sintetizador. Otro ejemplo de instrumento casero es el XVA1 [15], que aprovecha al máximo el paralelismo que ofrece el desarrollo en FPGA y consigue crear un sintetizador de alta gama con un Hardware muy reducido, incluyendo hasta 32 voces, sinónimo del número de notas que pueden ser reproducidas simultáneamente, y contando cada una con 4 osciladores independientes. Lo más valioso de este instrumento es su unidad de efectos, contando con hasta doce efectos, pudiendo trabajar todos en paralelo, gracias al diseño sobre FPGA. También incluye integración MIDI, lo que lo hace un instrumento muy completo.

Se puede afirmar entonces que las FPGA son una parte importante en el futuro de los sintetizadores hardware, viendo que varias empresas se han lanzado a utilizar esta tecnología en sus dispositivos de alta gama, e incluso proyectos más pequeños, hechos para funcionar en un Hardware muy reducido, alcanzan un gran número de características previamente reservadas a los instrumentos de mayor gama fabricados con DSPs de corte clásico. La tabla 1 presenta un resumen de los sintetizadores más representativos, así como sus principales características.

Sintetizador basado en fpga	**Características**
Peak de Novation	8 voces, cada una con 3 osciladores y un generador de ruido, además de 2 LFOs (Low Frequency Oscillator), todos ellos sampleados a 24MHz. Osciladores NCO y de Wavetable. Efectos con frecuencia de muestreo de 90kHz. Con interfaz MIDI.
Kyra de Waldorf	128 voces, cada una con hasta 4096 osciladores de forma de onda distintos y 8 de corte más clásico (Onda cuadrada, triangular, diente de sierra…) y 9 efectos que pueden trabajar en paralelo. Con interfaz MIDI.
Tiny Synth de Matteo Di Cosmo	Polifónico, con siete tipos de osciladores distintos por voz, filtro SVF (State Variable Filter) y dos tipos de efectos. Con interfaz MIDI.
XVA1 de René Ceballos	32 voces, con 4 osciladores por voz, más de veinte tipos distintos de filtros, dos LFOs por voz, 12 efectos que pueden trabajar simultáneamente. Con interfaz MIDI.

Tabla 1: Resumen de características de Sintetizadores basados en FPGA

2.2 Aplicaciones Software granulares

Generalmente, la técnica de síntesis granular es utilizada para la generación de texturas sonoras [16], ya sea mediante modelos estocásticos, dando lugar a sonidos más naturales, o deterministas. Si bien esto es obra de los pioneros que idearon su uso en el ámbito musical [1] [12], esta tecnología no solo se limita a ella.

Existen, por tanto, aplicaciones que tienen como eje fundamental la tecnología granular, fuera del ámbito de los sintes, como la utilización de sonidos naturales, como botellas cayendo o gotas de agua, para generar texturas “naturales” [17] o

elongar pequeñas piezas musicales hasta el extremo, en el efecto conocido como PaulStretch, que puede alargar un fragmento musical de cinco minutos hasta alcanzar una hora de duración [18].

Es este último programa el que introduce uno de los dos usos más extendidos de la síntesis granular. Estos son, "Time-Stretching" y "Pitch Shifting" [19]. Estos procesos tratan de modificar la longitud de un fragmento de audio sin modificar su tonalidad y alterar el tono de un sonido sin alargar o acortar su duración, respectivamente. Ambos algoritmos son complementarios, ya que antes de su concepción, se necesitaba modificar la frecuencia a la que se reproducía un sonido para conseguir uno de los efectos, pero inevitablemente se producían los dos simultáneamente.

El primer algoritmo consiste en separar el audio a elongar en pequeños fragmentos, es decir, los granos, y, aplicando una envolvente a cada uno de ellos, para evitar artefactos indeseados en los límites entre unidades sucesivas. Acto seguido, cada uno de los granos se repite un número prefijado de veces para conseguir alargar el audio, además de presentar cierta superposición en los límites de cada grano, para crear una continuidad a lo largo de la muestra. Al contener cada grano un contenido espectral no modificado del sonido original se consigue mantener intacta su tonalidad. Se puede ver un ejemplo de funcionamiento del algoritmo en la figura 5.

El método de "Pitch Shifting" es muy similar al anterior descrito. El primer paso, de nuevo, es fragmentar el audio a modificar en granos. A partir de aquí se puede realizar de dos maneras distintas. La primera sería alargar la muestra al igual que con "Time Stretching". Acto seguido, se muestrearía el audio con un ratio acorde a la modificación a realizar. Si por ejemplo se requiere aumentar una octava el tono, se debe muestrear al doble de frecuencia que la original. En este caso, los dos pasos seguidos pueden realizarse en orden inverso, y se obtendría un resultado muy similar. En la segunda manera, se modifica el tono de cada uno de los granos, quedando todos ellos con una longitud distinta a la original. Acto seguido, se repiten o eliminan (según se esté aumentando o disminuyendo el tono) los granos para mantener la duración original de la muestra.

Estos dos métodos son los utilizados por uno de los algoritmos más utilizados en procesamiento de audio, el Overlapp-Add, u OLA. Tiene numerosas aplicaciones, tanto en el dominio frecuencial como en el temporal. En este último es en el que se desarrolla los métodos previamente descritos. Su funcionamiento se detalla en la figura 5, en la que podemos observar cómo, para alargar la señal original, se elige adecuadamente la longitud de cada grano, para posteriormente, mediante un intervalo determinado de superposición, aunarlos y conseguir un fragmento de audio de mayor duración. Este intervalo de superposición es predeterminado en este algoritmo, lo cual puede provocar que, para según qué señales de origen, se produzcan distorsiones en dicha intersección. Para ciertas aplicaciones, como para aquellas que utilicen audio de voz, dicho efecto puede eliminar la funcionalidad del sistema al completo. Por ello, nace la evolución directa de este algoritmo, el SOLA (Synchronous OverLap and Add), que analiza la señal de origen mediante una DFT (Direct Fourier Transform) para encontrar el intervalo de superposición con mayor correlación entre ambos extremos a juntar, reduciendo así los efectos adversos comentados [20].

Tras lo descrito, se puede concluir que la funcionalidad de la síntesis granular, aun centrándose en el mundo del audio, va más allá de la generación de texturas y sonidos, siendo la base de una de las modificaciones más utilizadas por músicos día a día.

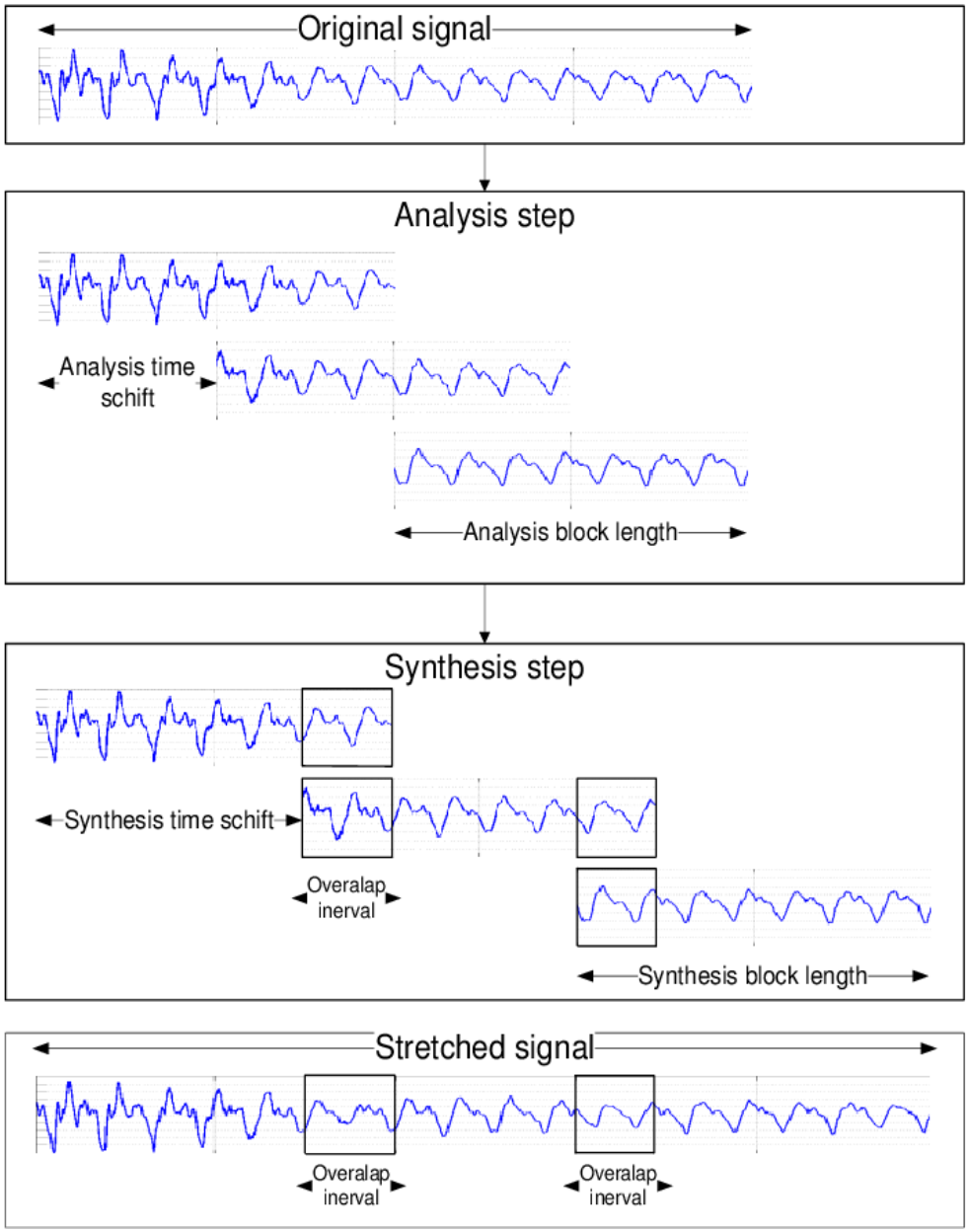


Figura 5: Demostración de "Time Stretching" con algoritmo OLA [20]

2.3 Sintetizadores hardware que utilizan síntesis granular

Hoy en día existen numerosos ejemplos de sintetizadores hardware basados íntegramente en síntesis granular. Uno de los más aclamados por la crítica es el GR-1, de Tasty Chips Electronics [21] y lo vemos en la figura 6.

Este sintetizador, implementado sobre una Raspberry Pi3, al estar íntegramente centrado en la síntesis granular, posee características que hacen brillar esta técnica, como una polifonía de 16 voces, cada una pudiendo generar hasta 128 granos por segundo, alcanzando una cifra de más de 1000 granos reproduciéndose en paralelo. Además, permite insertar distintas envolventes a los granos, permitiendo jugar con la modificación espectral que cada una de ellas supone, un aspecto muy interesante a introducir en todo sintetizador de esta índole.

Figura 6: GR-1 de Tasty Chips Electronics [21]

2.4 Conclusión

Tras analizar los distintos sistemas que existen hoy en día en el mercado de la producción musical, tanto en el ámbito de la FPGA, como en el de la síntesis granular, puede concluirse que ambos son pilares fundamentales de la misma, siendo el primero un gigante en crecimiento, y el segundo una técnica bien fundamentada y usada en un amplio abanico de aplicaciones. Es por ello por lo que en el presente trabajo se buscará implementar las bases de un sintetizador granular, quedándose en el aspecto fundamental de dicha tecnología, para explorar como puede ser implementada sobre FPGA, y, sobre todo, que aspectos de dicha plataforma pueden ser ventajosos para este tipo de síntesis en concreto.

3. Desarrollo del sintetizador

3.1 Introducción

Para la implementación del sintetizador granular en FPGA, se hará uno simplificado, que contenga los bloques básicos necesarios para la generación y procesamiento de los granos. Sus características serán:

- Síncrono: los granos estarán dispuestos de tal manera que su posición no varíe con el tiempo y se puedan superponer entre sí. Para ello, se utilizará una memoria de doble puerto, que nos permita obtener muestras de un grano aun no habiendo terminado de obtener el anterior al completo. El usuario seleccionará que nivel de superposición o distancia querrá entre los granos. Dicho parámetro se define como "Densidad", ya que, mediante este, se modificará el número de granos que se reproducen por segundo [1].
- Basado en "Samples": la fuente de los granos será un archivo almacenado en memoria Block RAM. Esto aportará una mayor riqueza sonora que otras formas de síntesis, como la basada en osciladores [22], debido a la variedad espectral que podemos obtener de muestras provenientes de instrumentos, voces o de otros sintetizadores.

El sistema se organizará según aparece en la figura 7. El primer bloque a crear será el manejador de memoria, que se encargará de seleccionar las muestras tanto del audio como de las envolventes de cada grano, todas ellas almacenadas en dos memorias de tipo Dual Block ROM. En el bloque que aunará el anterior, el granulador, se situarán dos multiplicadores que trabajarán en paralelo y generarán cada uno de los granos de los dos flujos. A la salida del bloque, se aunarán ambas señales para generar el audio base de nuestro sistema.

Posterior al bloque granulador, estará presente el módulo ASR, acrónimo de Attack, Sustain Release. Dicho bloque modificará la amplitud del flujo de audio al momento de tocar la tecla, al mantenerla y al soltarla, tal como aparece en la figura 7. Tanto la duración de la fase Attack y Release.

El siguiente bloque será el filtro SVF o filtro de variables de estado [23]. Este filtro digital trata de emular un filtro analógico. Las bondades de dicho módulo es la capacidad de obtener en una misma construcción un filtro paso bajo, paso alto, paso banda y banda eliminada. Si bien su curva de ganancia no es la más acusada, para aplicaciones de audio sus ventajas superan con creces a sus limitaciones. Permite ajustar tanto la frecuencia de corte como la resonancia fácilmente.

Finalmente, el flujo de datos alcanzará el bloque I2S, cuya función consiste en generar las señales de control y transmisión de datos del estándar I2S [24] para comunicarse con el dispositivo PMOD I2S2 de Digilent [4], que permitirá reproducir los datos generados por el resto de bloques del sistema.

Ajeno al procesamiento de datos, se encuentra el bloque de Interfaz de usuario, que se conectará al módulo PMOD ENC de Digilent [25], y junto a los elementos Hardware

integrados en la placa Nexys4 DDR [26], permitirá modificar los parámetros internos del sistema. Dichos parámetros son:

- Duración del grano: Determinará el número de muestras que contenga cada unidad sonora. Dicho parámetro podrá variar entre 10 y 150 milisegundos, en saltos de 10 cada uno.
- Inicio de toma de muestras: Se indicará a partir de qué punto del "Sample" se comienzan a generar los granos.
- "Interonset time": este término hace referencia al tiempo que transcurre entre el inicio de un evento sonoro y otro, como se muestra en la figura 8. En nuestro caso, definirá el tiempo que transcurrirá entre el inicio de sucesivos granos. Este parámetro es el que definirá la densidad del flujo de granos.
- Envolvente ASR: Se permitirá la modificación de la duración de las etapas de Attack y Release, así como el volumen que se mantendrá durante la etapa de Sustain.
- Tipo de filtro a utilizar: el usuario podrá seleccionar entre filtro paso bajo, paso alto, paso banda o banda eliminada, e incluso una opción para que dicho bloque no modifique la señal.
- Frecuencia de corte del filtro.
- Resonancia del filtro.

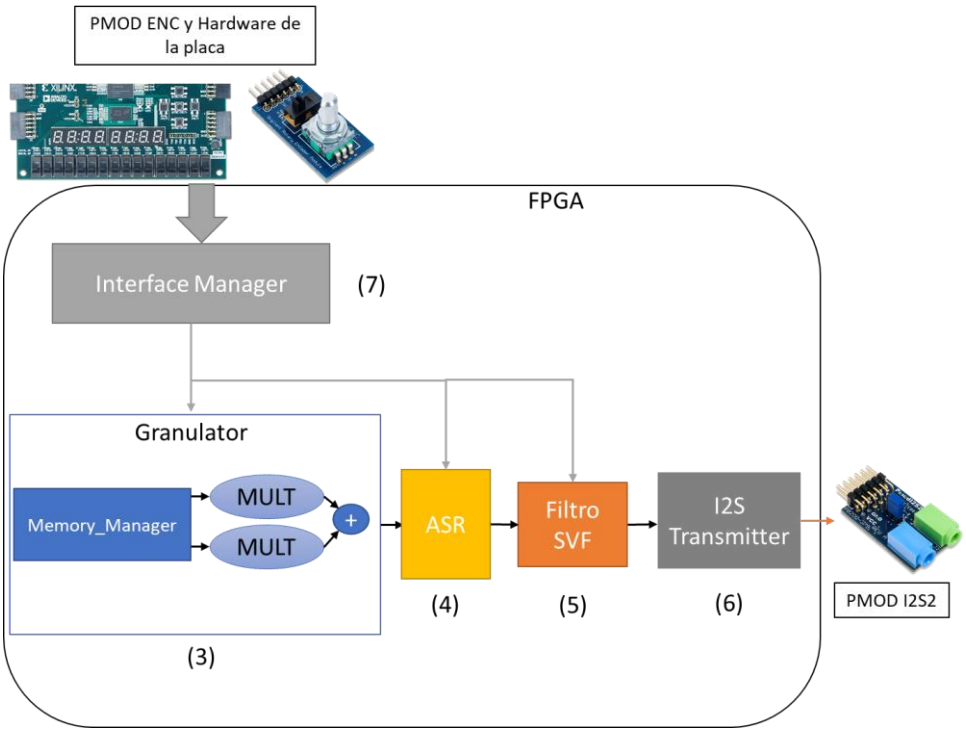


Figura 7: Esquema general del sistema

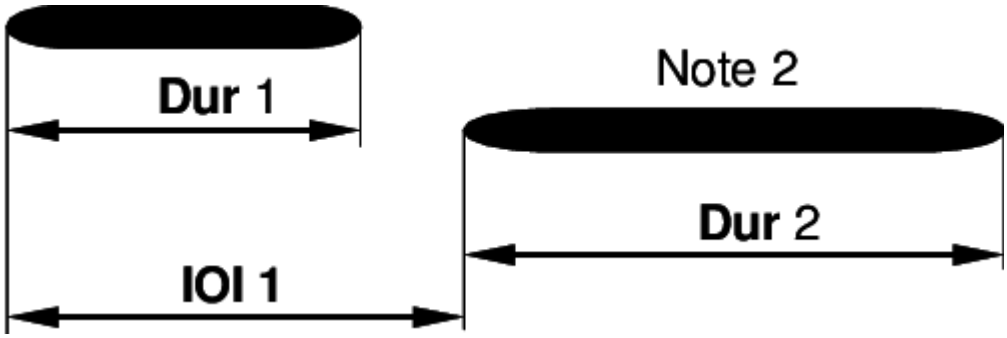


Figura 8: Definición gráfica del Interonset Time, en la imagen IOI 1 [24]

3.2 Consideraciones iniciales

Previo a la construcción de cada bloque, se determinarán aspectos fundamentales del sistema, como el tipo de datos a utilizar, la obtención de las distintas envolventes del sistema y otros aspectos.

3.2.1 Tipo de datos a utilizar

Para comenzar a diseñar cada bloque, se debe tener claro que formato tendrá el flujo de audio principal, y que frecuencia de muestreo utilizar. Debido a la naturaleza de la síntesis basada en "Samples", estos valores vendrán ya dados por el sonido que usemos de origen.

El tamaño del dato y la frecuencia de muestreo vendrán dada por el audio de origen que se elija. En este caso, se ha optado por la utilización del formato .WAV, cuya estructura es mostrada en la figura 10. Este códec de audio es muy popular entre aquellos entusiastas de la música que busquen un sonido limpio y nítido en un amplio espectro de frecuencias, ya que, además de admitir una frecuencia de muestreo de hasta 4.3 GHz, debido al tamaño de dicho dato en el formato, almacena audio no comprimido, es decir, la señal original se conserva al completo, sin importar la memoria que ocupe. Al contrario que formatos más conocidos, como .MP3, WAV mantiene intacta la señal que fue muestreada en primera instancia. Es por este formato que se ha descartado la utilización de un flujo de audio representado en coma fija, ya que los datos almacenados en él son enteros.

Además de permitir una elevada frecuencia de muestreo, el formato WAV también permite obtener tamaños de sample variados. Por delimitar uno para el diseño, se escogerán solo archivos con este formato, de 16 bits por muestra, y una velocidad de muestreo de 44.1 KHz. Para extraer los datos del formato y llevarlos a otro más sencillo de implementar dentro de VHDL, se utilizará la función *audioread* de Matlab, con la opción de "*native*" activada, para obtener así la parte de datos WAV en su formato original. De la misma manera, se utilizará el mismo programa para almacenar los datos en un archivo .TXT, que permita utilizar el audio original como fuente de muestras para las simulaciones en VHDL. En la figura 10 se muestra un esquema de la obtención de los datos de este formato.

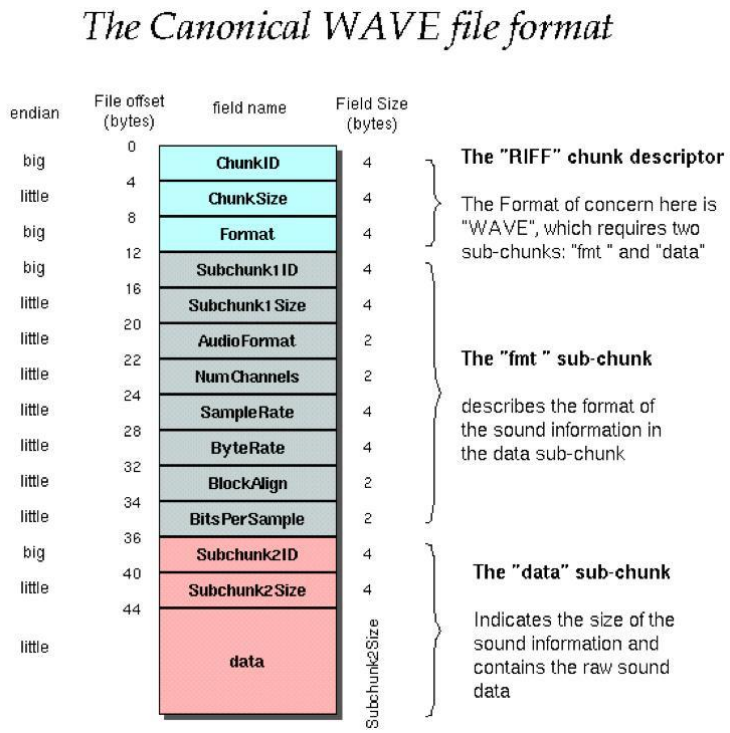


Figura 9: Estructura del formato de archivo WAV [39]

Para otros datos, como los valores de control del filtro SVF y las ventanas de grano, se ha utilizado el formato de coma fija, obtenido mediante la función *fi* de Matlab. Al ser datos que oscilan entre 0 y 1, se ha declarado una parte entera de 1 bit, más una parte decimal de 15 bits, para que su tamaño vaya a la par con el flujo de audio. Posteriormente, se almacenarán en un fichero COE, que serán explicados más tardes, o incluidos en el código del diseño. El esquema de obtención de datos mediante Matlab se detalla en la figura 10. En ella, se detallan los parámetros a seleccionar con la función *fi*, que son, respectivamente, dato sin signo, de 16 bits, con 15 dedicados a la parte decimal.

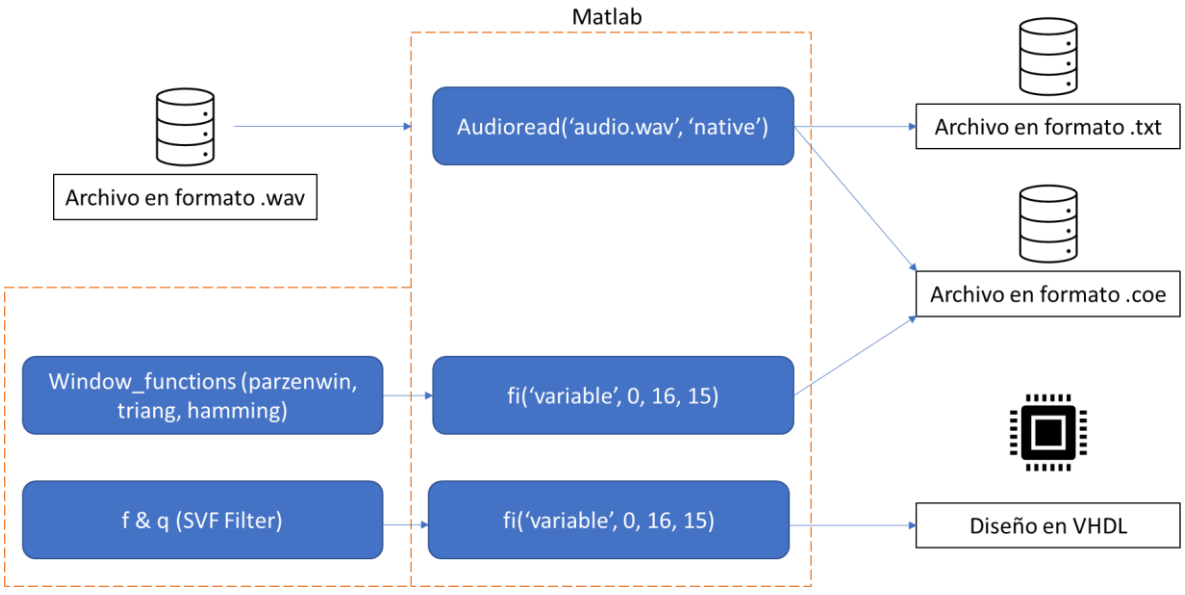


Figura 10: Esquema de obtención de datos mediante Matlab

3.2.2 Envolventes de grano

Como se detallaba al inicio, a cada grano debe aplicársele una envolvente para evitar artefactos indeseados. Si bien para ello cualquier envolvente nos valdría, también repercutirá sobre el resultado final del flujo creado, desde un punto de vista frecuencial. Es decir, si el grano es lo suficientemente reducido, la envolvente actuará incluso como un filtro de paso banda.

Para el diseño, se incluirán tres tipos de ventanas distintas: triangular, de Parzen y de Hamming. La razón para escoger cada una de ellas, es lo distintivo del análisis espectral de cada una, que se pueden observar en la figura 11. La triangular presenta unos lóbulos laterales estrechos y con relativa poca separación. La ventana de Parzen, si bien no suele ser usada en el ámbito musical, presenta unos lóbulos laterales notablemente distanciados y con considerable anchura, lo que a efectos prácticos puede aportar un timbre interesante al sonido. Por último, la ventana de Hamming, muy utilizada en filtros digitales, presenta un espectro notablemente lineal en las frecuencias laterales, lo que dará menos color al sonido, y más homogeneidad en dichas regiones.

Todas las ventanas serán generadas mediante Matlab, con un tamaño de 6615 muestras cada una, siendo este el tamaño máximo de grano que va a procesar el sistema.

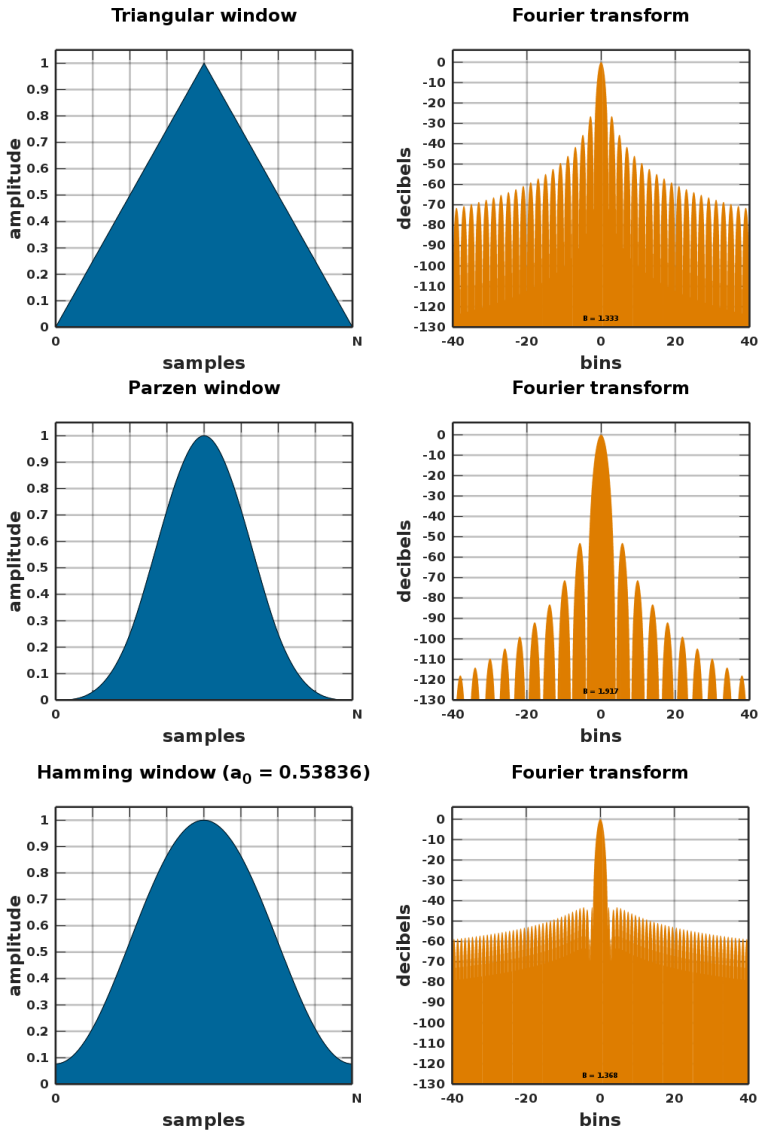


Figura 11: Ventanas utilizadas en el diseño y su respuesta frecuencial [27]

3.2.3 Generación de archivos COE

Para la inicialización de los bloques de memoria, se utilizarán archivos de formato COE, que siguen el formato que se observa en la figura 12 [28]. En la primera línea se especifica el número de bits por muestra. En la segunda se introducen los datos con los que se inicializará la memoria.

Para la creación de estos archivos, se utilizará el Script de Matlab que aparece en la figura 13. En él, previo al grabado de los datos, se transformarán en formato de coma fija mediante la función *fi,* con las especificaciones requeridas sobre parte decimal y parte entera. Dicha conversión de datos se detalla en la figura 10. Acto seguido, en un bucle se irán escribiendo, tras la inicialización requerida, los datos a grabar en el archivo COE.

```
memory_initialization_radix=16;
memory_initialization_vector=
004b,00c7,00cf,004b,ff95,ff19,ff1b,ffaa,00b5,015d,0136,0079,0018,000b,ffd0,ffab,
41,00ad,01a5,0108,ff9d,fed9,fe61,fef1,0071,0186,0150,00e5,008b,ffa4,fe88,fe25,fe
,00ed,021e,0209,0107,0015,ff73,ffa7,007a,0130,0132,0036,000f,fff3,ff0a,fe7e,fe84
154,0255,0231,00e9,fedb,fd69,fe3b,ffeb,00a0,0010,ff5e,ff1e,fee8,ff00,ff68,ffb5,f
5,011a,0065,ffc4,fedc,fe15,fe3f,ffb3,01bc,034b,03a3,031a,01f0,0045,feda,fe6a,ff0
00f3,ff72,fe1b,fd2f,fca5,fc46,fd0d,fefb,0105,0252,0213,008e,feff,fe94,feec,ff5c,
6b,ff2f,ff1a,0098,0253,02b1,021f,fff3,fec8,ffc2,0105,0078,ffaa,ff93,ff80,ff55,ff
,fc8c,ff82,01d7,0107,ff4d,fc90,fbd3,fb91,fc60,ffc1,0281,03c5,022f,ff73,fca7,fc1e
43a,0450,02d1,027c,01fc,0182,0073,ffd0,0094,01a1,03c8,0452,0275,0023,ffc9,ffa5,f
c,fc69,fb37,fbf3,fd1c,fe5d,fef8,fda6,fc81,fcb0,fe91,0015,0150,028a,0300,025c,ffd
01ab,0128,018b,0223,024e,00d7,ffa9,ffb6,fff0,0012,00fa,0155,00f9,0070,0036,0085,
```

Figura 12: Formato de archivo COE

```
triang_window=triang(150*44.1);
parzen_window=parzenwin(150*44.1);
hamming_window=hamming(150*44.1);
array_window=vertcat(triang_window, parzen_window, hamming_window);
array_window_fp= fi(array_window, 0, 16, 15);
array_window_fp_hex=array_window_fp.hex;

fid = fopen('windows.coe', 'w');
fprintf(fid, 'memory_initialization_radix=16;\n');
fprintf(fid, 'memory_initialization_vector=\n');

for row = 1 : size(array_window_fp_hex,1)
    for col = 1:size(array_window_fp_hex,2)
        if col == size(array_window_fp_hex,2)
        fprintf(fid, '%s,', array_window_fp_hex(row,end));

        elseif row == size(array_window_fp_hex,1) && col == size(array_window_fp_hex,2)
        fprintf(fid, '%s;', array_window_fp_hex(row,end));

        else
        fprintf(fid, '%s', array_window_fp_hex(row,col));
        end
    end
end
fclose(fid);
```

Figura 13: Código de generación de archivos COE de ventanas

3.3 Bloque granulador

Este bloque, que consistirá, como se observa en la figura 7 (bloque 1), del manejador de memoria, dos multiplicadores y un sumador, será el encargado de generar el flujo inicial de granos para alimentar el resto de los bloques.

3.3.1 Manejador de memoria

El presente módulo será el encargado de, según las especificaciones recibidas por la interfaz de usuario, generar cuatro señales distintas, dos para granos, y dos para envolventes, extraídas de dos memorias de tipo ROM de doble puerto. Su estructura es la observada en la figura 14, mientras que sus puertos quedan descritos en la tabla 2.

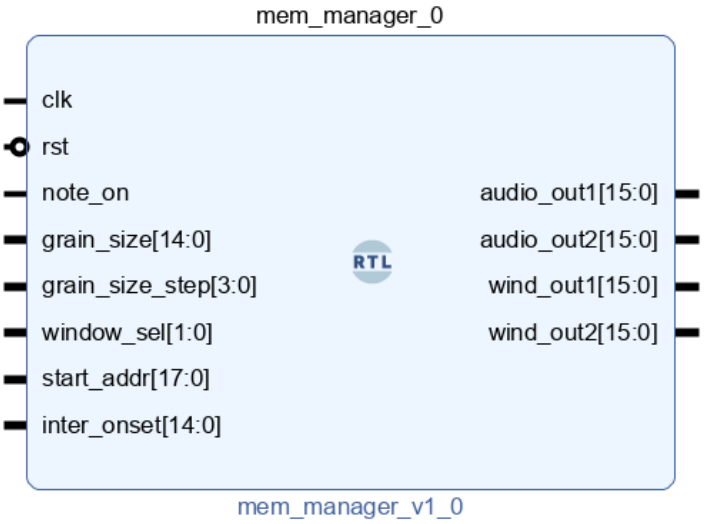


Figura 14: Estructura del bloque manejador de memoria

Para la creación de las memorias ROM, se utilizará el IP Block Memory Generator [28], de Xilinx, cuyo asistente nos permitirá configurar los distintos aspectos del almacenamiento. En la figura 15 se observa la pantalla inicial del IP en cuestión. En ella se seleccionará el tipo de memoria que se requiere crear, en este caso, ROM de doble puerto, y el algoritmo de creación de memoria. Dicho parámetro determinará las distintas primitivas, o bloques fundamentales, que seleccionará el asistente a la hora de generar el bloque. Se distinguen tres casos:

- Minimum Area: Con este algoritmo, se creará la memoria con la configuración que ocupe el mínimo área posible, sin importar las primitivas que se usen, tal y como se muestra en la figura 16.
- Low Power: Con esta opción activada, la memoria generada será la que menos primitivas active en cada operación de lectura, y, por tanto, la que menos energía consumirá. Un ejemplo de distribución de memoria generado con este método se muestra en la figura 17.
- Fixed Primitives: Con el último algoritmo, se creará la memoria usando exclusivamente las primitivas fijadas por el usuario. Un ejemplo de este tipo de memoria es mostrado en la figura 18.

Puerto	Tipo	Descripción
clk	Entrada	Entrada de reloj
rst	Entrada	Entrada de señal de reset
note_on	Entrada	Señal de tecla pulsada
grain_size	Entrada	Numero de muestras del grano
grain_size_step	Entrada	Tamaño del grano expresado del 1 al 15
window_sel	Entrada	Array de selección de ventana a usar
start_add	Entrada	Dirección de inicio para la toma de muestras
inter_onset	Entrada	Numero de muestras entre granos sucesivos
audio_out1	Salida	Salida de audio
audio_out2	Salida	Salida de audio
wind_out1	Salida	Salida de datos de ventana
wind_out2	Salida	Salida de datos de ventana

Tabla 2: Descripción de puertos de bloque manejador de memoria

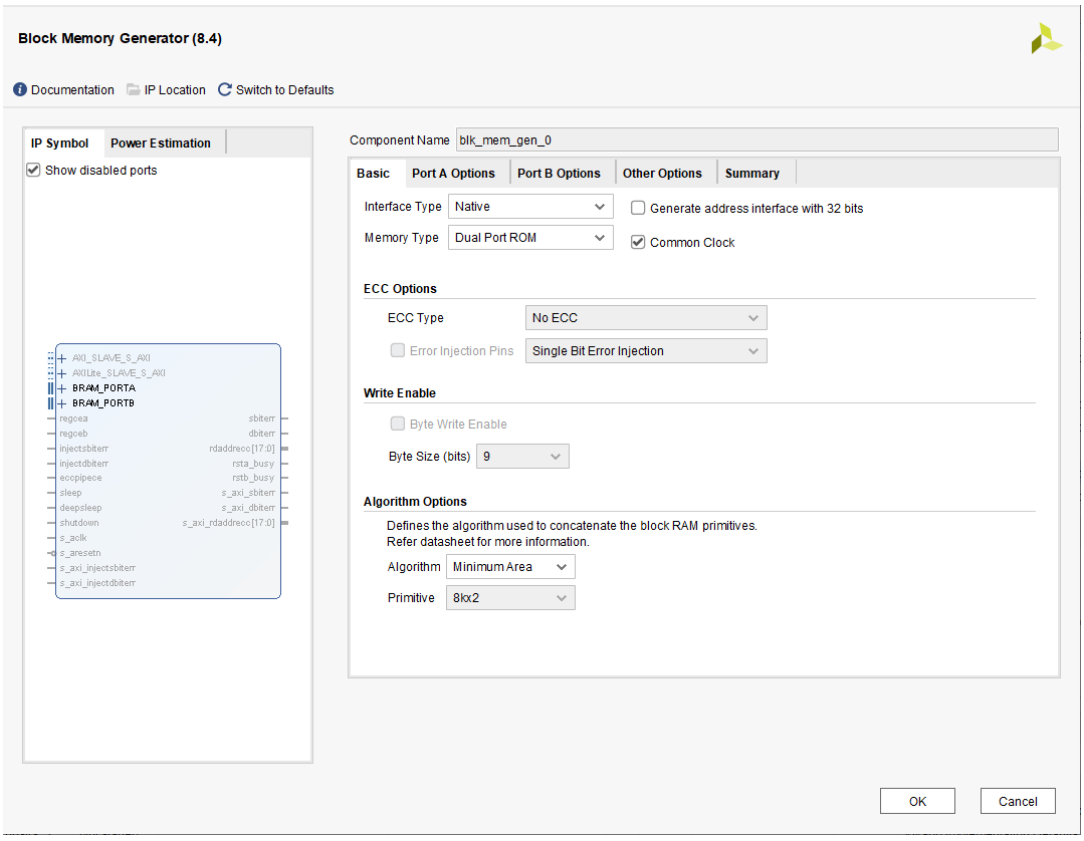


Figura 15: Primera pantalla de asistente Block Memory Generator

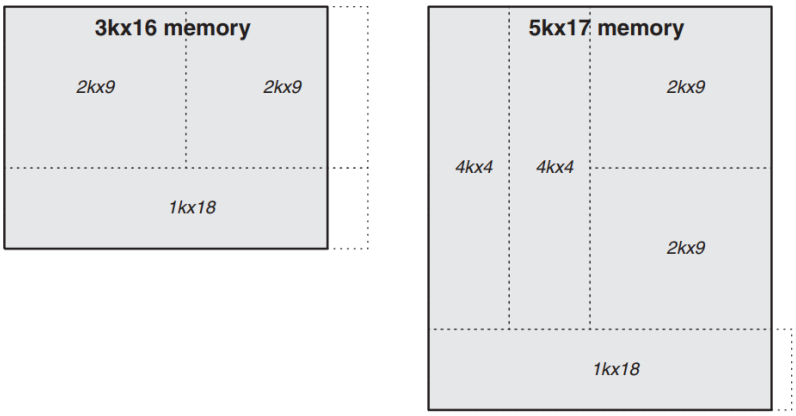


Figura 16: Ejemplo de memorias generadas con algoritmo Minimum Area

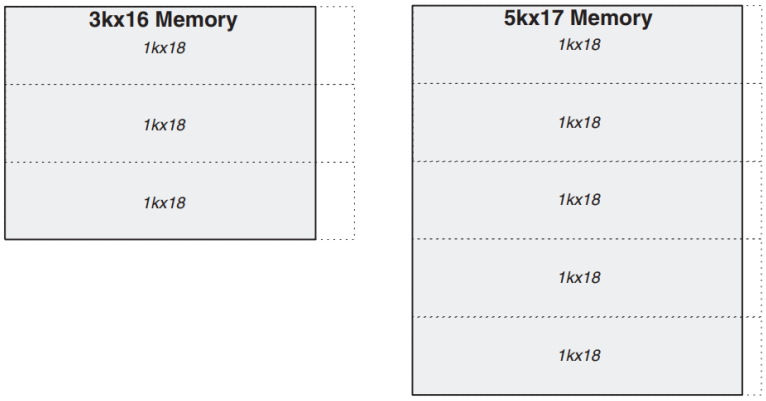


Figura 17: Ejemplo de memorias generadas con algoritmo Low Power

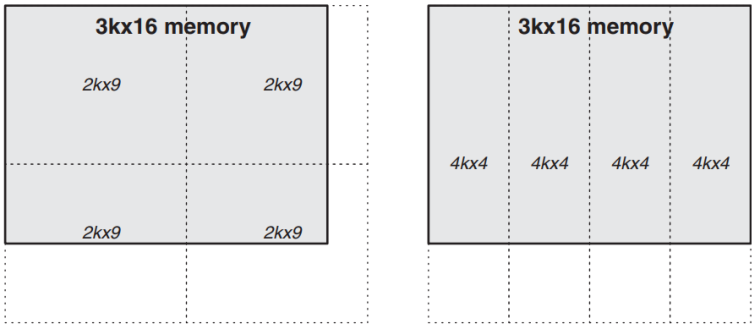


Figura 18: Ejemplo de memorias generadas con algoritmo Fixed Primitives

Para el presente diseño, se ha seleccionado la opción de área mínima como la óptima, debido a la memoria que presenta la FPGA utilizada, que es de 4860 kilobits dedicada a Block RAM [29], más que suficiente para alojar el audio seleccionado y las muestras de ventana, así como otro bloque que requiera de almacenamiento de este tipo.

En la siguiente pantalla del asistente, mostrada en la figura 19, seleccionaremos el ancho del puerto A, de 16 bits, así como el número de muestras que almacenará. Se ha tomado como máximo un valor que podrá alojar hasta 260000 muestras, que, según el formato utilizado, supondrían unos 6 segundos de audio. La memoria que alojará las ventanas, en cambio, almacenará 19845 muestras exactas. En esta misma sección se seleccionará si se requiere de una salida registrada. Si bien una latencia de uno o dos ciclos más de reloj no supondría ningún perjuicio a la hora de la generación de audio, se deseleccionarán dichas opciones, lo que hará que la salida de datos se produzca en un solo ciclo.

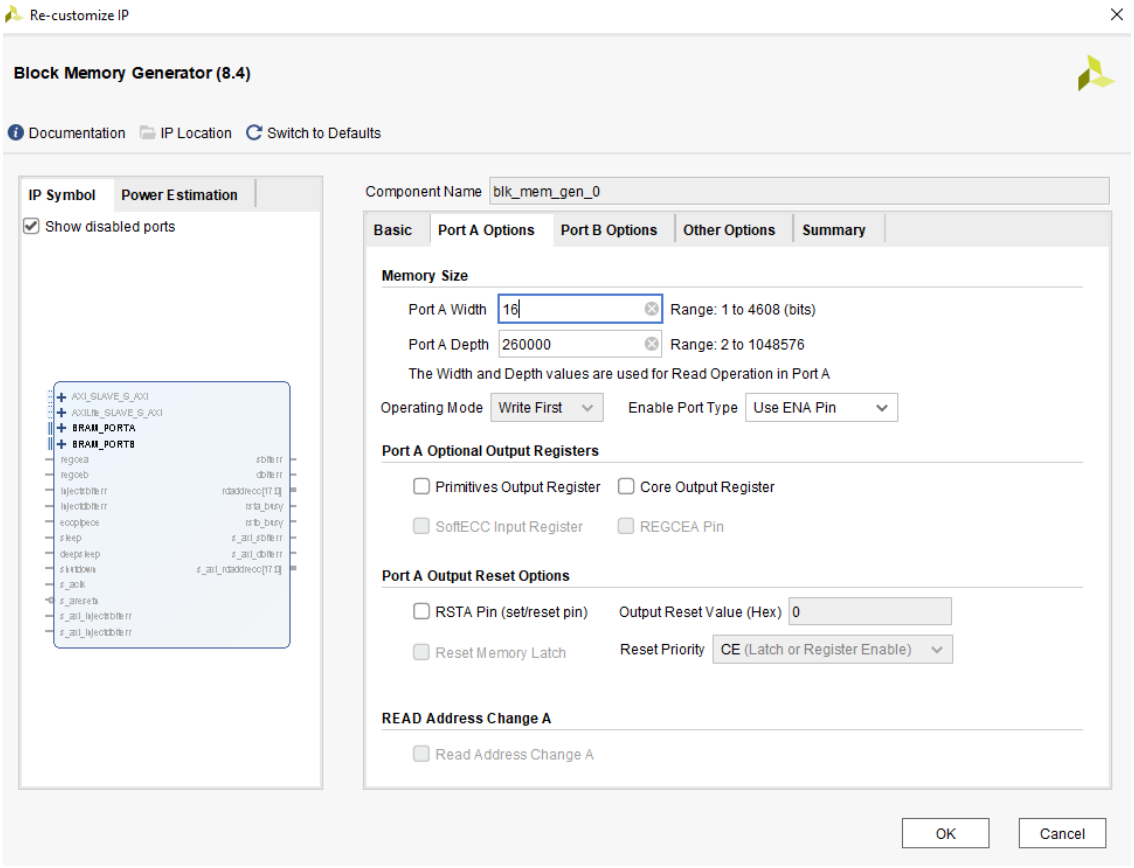


Figura 19: Segunda pantalla de asistente Block Memory Generator

Tras la generación de ambas memorias, se detallará el código que las maneje. El sistema cuenta con tres procesos distintos. El primero de todos será el encargado de obtener las muestras de ambas memorias. Su funcionamiento, representado en la figura 20, se rige por cuatro sentencias *if*, dos para cada uno de los flujos de datos, y dos para las ventanas. En estos dos primeros, se extraerán las muestras de la memoria mientras un contador interno se mantenga inferior al número establecido por *grain_size*. Cuando se alcance dicho límite, se comprobará si dicho contador supera dos veces el valor de *inter_onset*. De no ser así, la salida de audio se mantendrá a cero. Cumplida esta condición, se reiniciará el contador, y se llevará la dirección de memoria al valor indicado por *start_add*, para comenzar de nuevo el ciclo.

Para que existan diferencias entre la salida de datos A y B, este último, tanto al reiniciar, como al volver la señal *note_on* a cero, se inicializará con el contador igual a *inter_onset*, y la dirección de memoria a *start_add* más *inter_onset*. Esto conseguirá

generar la distancia entre granos que se requiere. Por lo demás, el código de control de la salida de datos de B es idéntico a A.

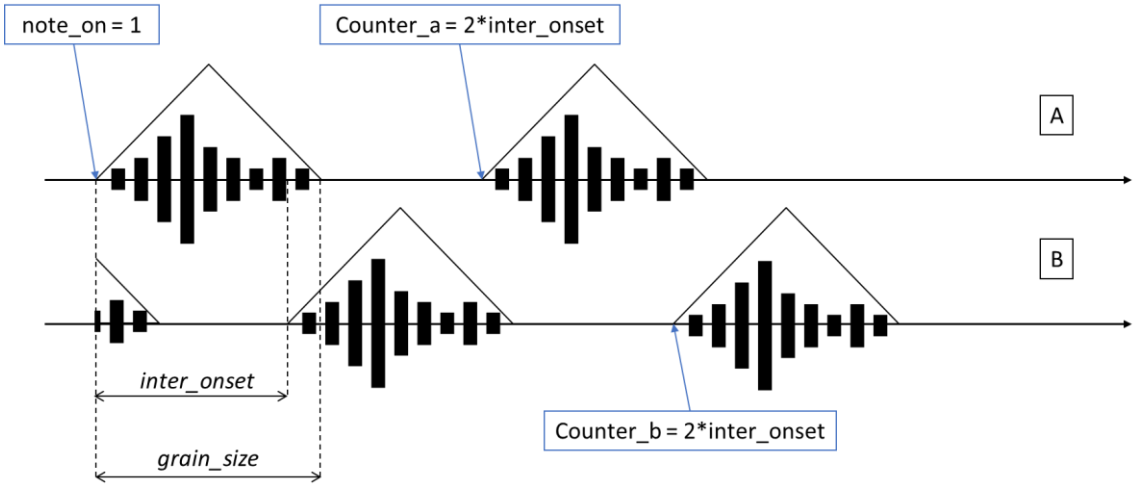


Figura 20: Diagrama de funcionamiento del bloque manejador de memoria

Las dos restantes sentencias *if* servirán para extraer los valores de la ventana seleccionada. La ventana tiene un número de muestras igual al máximo grano posible. Esto hace que haya que saltarse un número determinado de muestras determinado para alcanzar el tamaño del grano. Para ello, definimos la relación que existe entre el máximo y el mínimo número de muestras:

$$window\ scaling\ constant = \frac{\min window\ length}{\max window\ length} = \frac{441}{6615} = \frac{1}{15}$$

Este valor dictamina que para obtener una ventana de tamaño 441, partiendo de la almacenada en memoria, se deberán saltar 14 de cada 15 valores de la misma. Como se indicó al inicio, el tamaño de los granos se modificará en saltos de diez milisegundos. Para sucesivos tamaños de grano, se deberá por tanto saltar un número de $15 - grain\_size\_step$ granos cada quince muestras, siendo esta última variable un dato de valor entre 1 y 15, que variará en consonancia con el tamaño de grano seleccionado.

La salida de muestras de ventana se verá controlada de la misma manera que las de audio, por los mismos contadores. Además, existirá un segundo contador, que seguirá el control del número de muestras a saltar. Cuando este contador alcance el valor de *grain_size_step,* se reiniciará y posicionará el valor de direcciones de la memoria al siguiente valor:

$$address_{wind} = 15 - grain\_size\_step\ + 1$$

El resto de la sentencia sigue la misma dinámica que la extracción de muestras de audio.

Para finalizar, queda por detallar el funcionamiento de los dos proceso restantes. El primero de ellos se trata de una máquina de estados simple, cuyo único objetivo consiste en controlar si existe o no superposición de granos, comparando el tamaño

entre el tamaño de estos últimos e *inter_onset*. El bit que activa este proceso se usará como condición para conocer a que dirección debe inicializarse el puntero de memoria de B. Si existe superposición se deberá inicializar con un valor igual a *inter_onset* más la dirección de inicio de recogida de muestras. De no existir solapamiento, comenzará con el valor de esta última únicamente. El siguiente proceso se trata de otra simple máquina de estados que establecerá la dirección inicial de la ventana según el valor de *window_sel.*

3.3.2 Granulador

Conteniendo al bloque anterior se encuentra el granulador, cuya estructura se refleja en la figura 21. Tal y como se muestra en la figura 7, también cuenta con dos multiplicadores y un sumador. El primer elemento será implementado con otro IP que ofrece Xilinx, llamado Multiplier. Este permite generar multiplicadores configurables. La pantalla inicial del asistente, que se muestra en la figura 21, permite definir el tamaño y tipo de datos. En este caso, el audio es de 16 bits y con signo, mientras que la ventana será del mismo tamaño, pero sin signo. Así mismo, se puede seleccionar el tipo de construcción que se quiere para el multiplicador. Al ser uno relativamente simple, se utilizará la opción Use Mults, que utilizará un solo bloque interno de la FPGA, denominado DSP48, cuyo esquema interno se puede apreciar en la figura 22. Si bien no utilizamos todo el potencial de dicha célula, la tarjeta que se utiliza en el diseño cuenta con 240 unidades de la misma, por lo que podremos utilizarla para no ocupar unidades de LUTs (Look Up Table), célula mucho más usada en operaciones comunes.

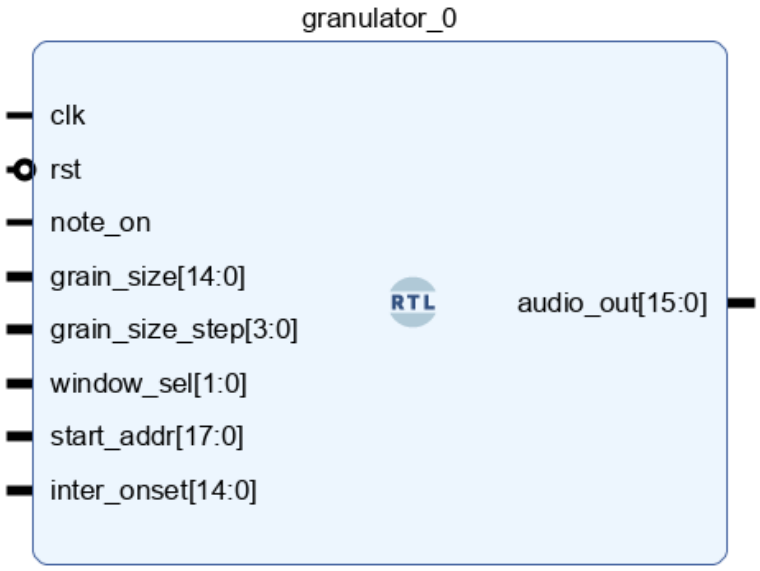


Figura 21: Estructura del bloque granulador

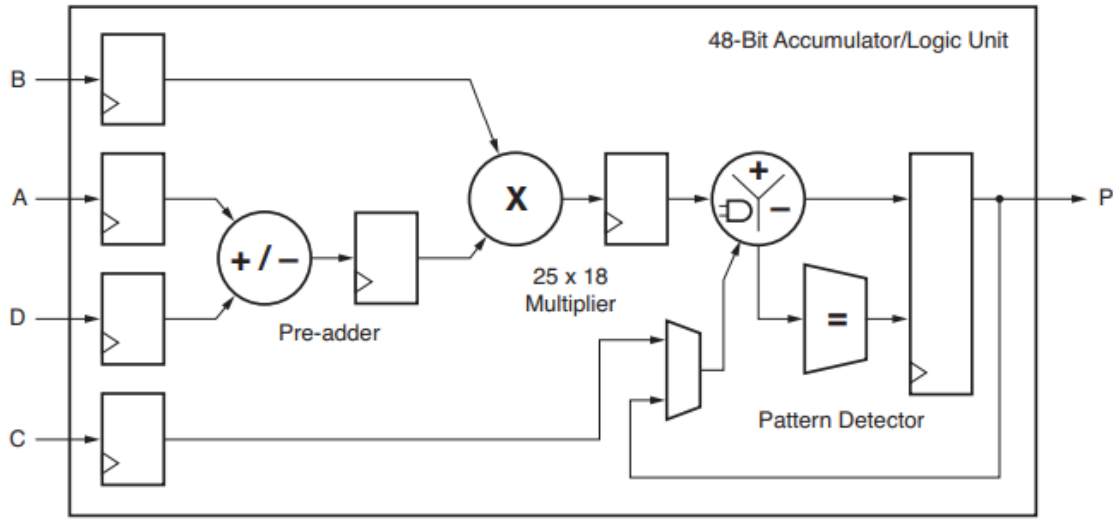


Figura 22: Estructura simplificada de la célula DSP48 [30]

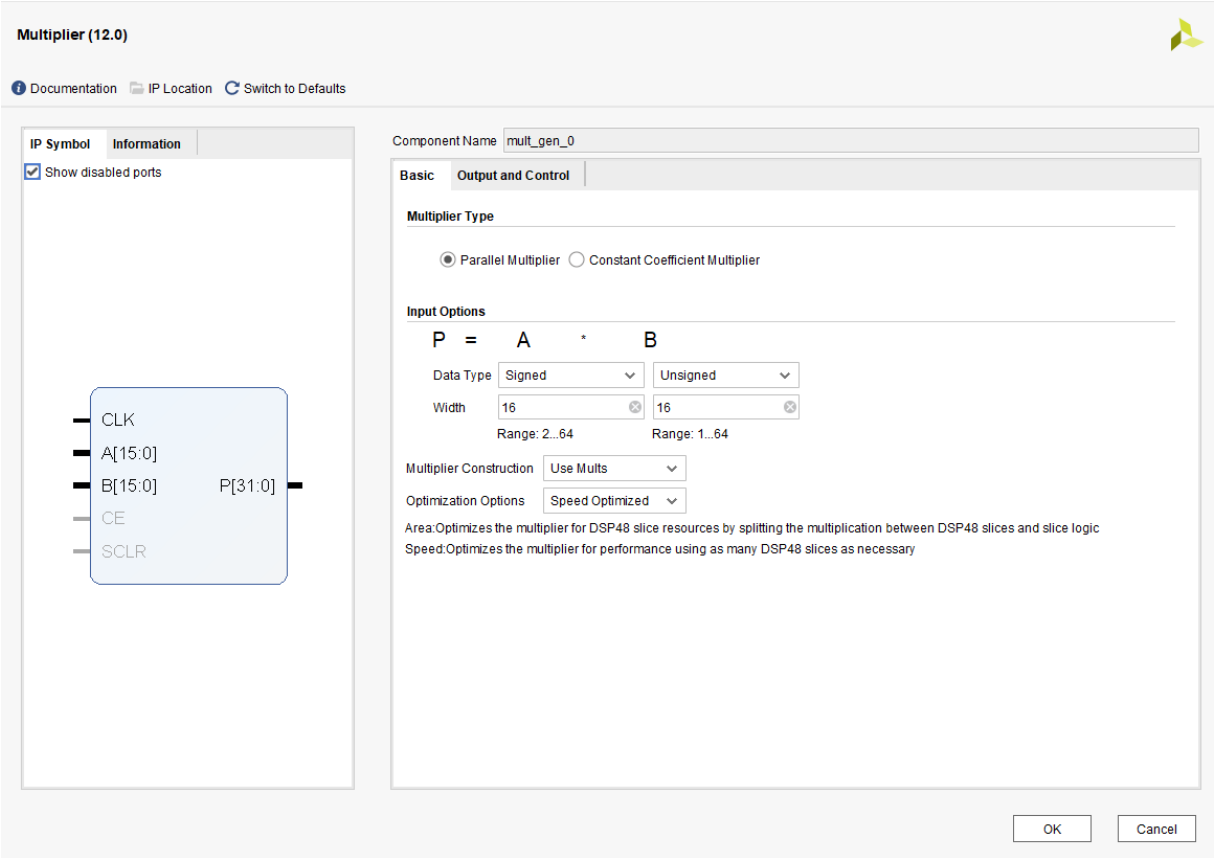


Figura 23: Pantalla inicial del asistente de Multiplier

El funcionamiento de este bloque es simple. Contiene las conexiones entre el manejador y los multiplicadores. Además, en cada ciclo de reloj, realizará la suma de los 16 bits de mayor peso del resultado de cada multiplicador, y lo llevará a la salida *audio_out* obteniendo finalmente el flujo inicial de granos.

3.4 Bloque ASR

El siguiente bloque se trata de la envolvente ASR, cuya función es la de aportar cierta sensación de evolución al sonido al pulsar la nota, mantenerla presionada y soltarla. Este bloque, fundamental en cualquier sintetizador, nace de la idea de obtener cierta modulación sobre la amplitud en relación con el tiempo, ya que, por lo contrario, un sintetizador sería estático en este aspecto.

El funcionamiento queda descrito por el diagrama de la figura 24. En él, se representa como evoluciona la amplitud de la señal de audio en el momento en el que se toca la tecla, correspondiente a la parte de "attack", a qué nivel se mantiene tras alcanzar el máximo, es decir, el "sustain", y como decae tras levantar la tecla, es decir, la "release".

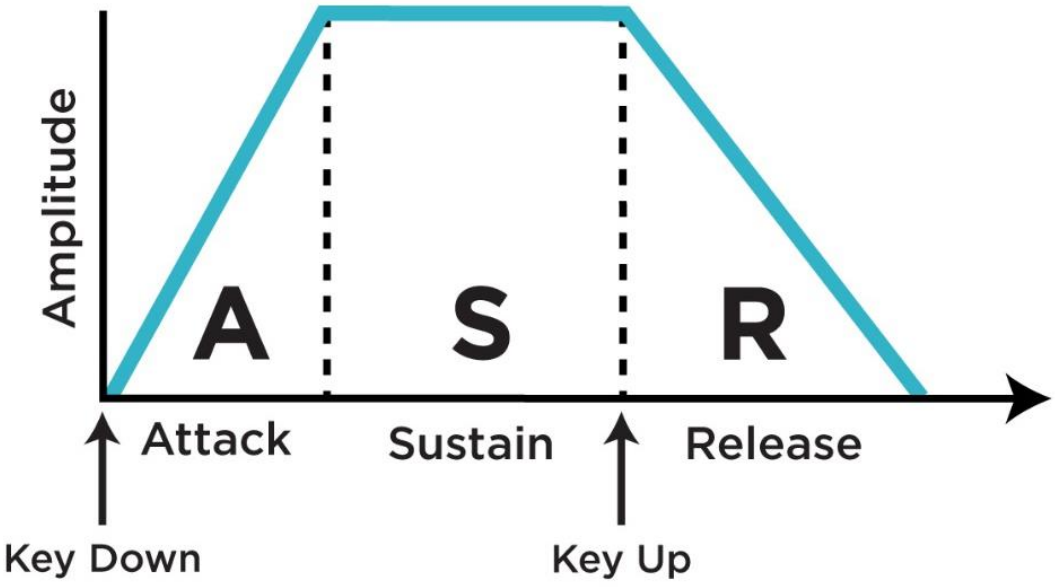


Figura 24: Funcionamiento de envolvente ASR [31]

En el bloque, cuya estructura se muestra en la figura 25 y la descripción de sus puertos en la tabla 3, una variable de control, denominada env_ASR, junto a las muestras de entrada al bloque, pasarán por un multiplicador, implementado de la misma manera que en el bloque Granulator, descrito en el apartado 3.3.2. Al modificarse env_ASR, variará la amplitud de la señal de audio resultante.

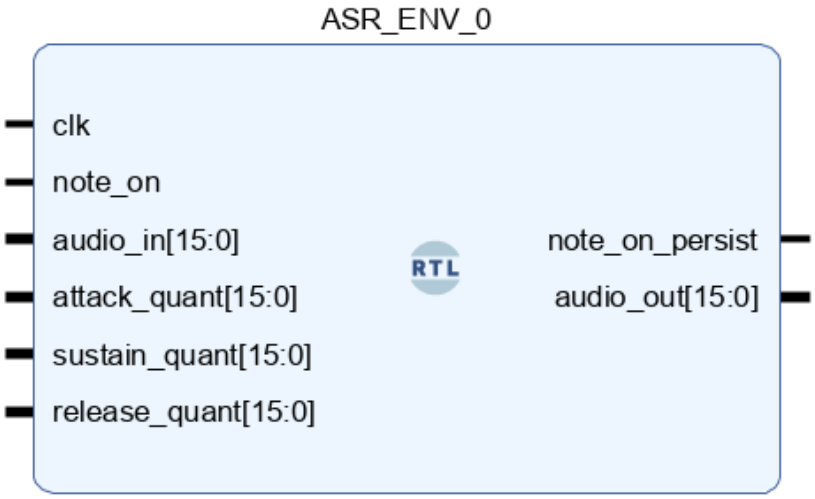


Figura 25: Estructura del bloque ASR

Puerto	**Tipo**	**Descripción**
Clk	Entrada	Señal de reloj.
Note_on	Entrada	Señal de botón de activación.
Audio_in	Entrada	Entrada de señal de audio.
Attack_quant	Entrada	Variable temporal de pendiente en el estado “attack”.
Sustain_quant	Entrada	Variable de amplitud en el estado “Sustain”.
Release_quant	Entrada	Variable temporal de pendiente en el estado “Release”.
Note_on_persist	Salida	Indicador de nota activada que persiste en el estado “Release”.
Audio_out	Salida	Salida de audio.

Tabla 3: Descripción de puertos bloque ASR

El valor de la señal *env_ASR* será controlado mediante una máquina de estados, aquella representada en la figura 26. Consta de 4 posiciones distintas: idle, attack, sustain y release. La máquina se iniciará en idle, y permanecerá en dicho estado hasta que se accione el botón de reproducción. En dicho estado, se registrarán los valores de las tres señales de control para que la envolvente no pueda ser modificada en el transcurso de una nota. Tras activarse el pulsador, se pasará al estado attack, en el que se irá incrementando la variable *env_ASR* en un valor de *attack_quant* en cada ciclo hasta que alcance o supere el valor de *sustain_quant*. De no llegar a cumplirse y desactivarse el pulsador, se avanzará al estado release directamente. De lo contrario, se pasará al estado sustain. En dicha etapa, simplemente se mantendrá la señal de salida al volumen de amplitud estipulado por *sustain_quant*, hasta que el botón de pulsador sea desactivado, momento en el cual se pasará al estado reléase. En esta

última posición, se irá decrementando la señal *env_ASR* en un valor de *reléase_quant* en cada ciclo de reloj, hasta que se alcance el cero.

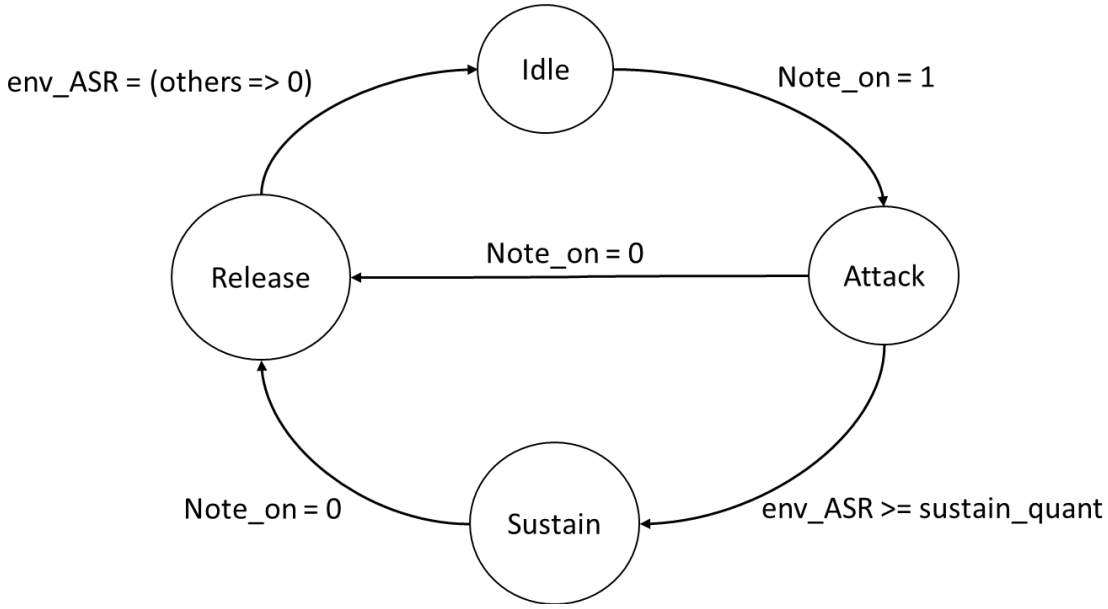


Figura 26: Máquina de estados de ASR

Durante los estados attack, sustain y release, se mantiene a nivel alto la salida *note_on_persist*, cuya función es actuar como indicador para el resto de los bloques de que, aun habiendo sido desactivado el pulsador de nota, se deben seguir desplazando muestras hasta que se salga del estado release.

3.5 Filtro SVF

El filtro de nuestro sistema se denomina SVF, acrónimo de filtro de estados variables en castellano. Se trata de un filtro FIR de segundo orden que permite obtener en la misma construcción un filtro paso bajo, paso alto, paso banda y banda eliminada, además de poder controlar la frecuencia de corte y la resonancia con solo dos parámetros. La frecuencia de corte oscilará entre los 100 y los 700 Hz, mientras que el factor de resonancia Q, variará entre 0.5 y 10.

3.5.1 Principio básico de funcionamiento

SVF se trata de un filtro que emula el funcionamiento de su homónimo en el ámbito analógico [23], cuyo esquema aparece en la figura 27. Los bloques constructores de este circuito son varios amplificadores, que contienen los valores que permiten modificar la frecuencia de corte, en este caso A, y la resonancia del filtro, siendo este B. Estos bloques, al igual que el sumador de entrada y de salida, tienen un equivalente directo en el ámbito digital. Los dos bloques restantes, sin embargo, son dos integradores, cuyo equivalente digital sería el mostrado en la figura 28. Por tanto, el filtro a implementar sigue la estructura mostrada en la figura 29.

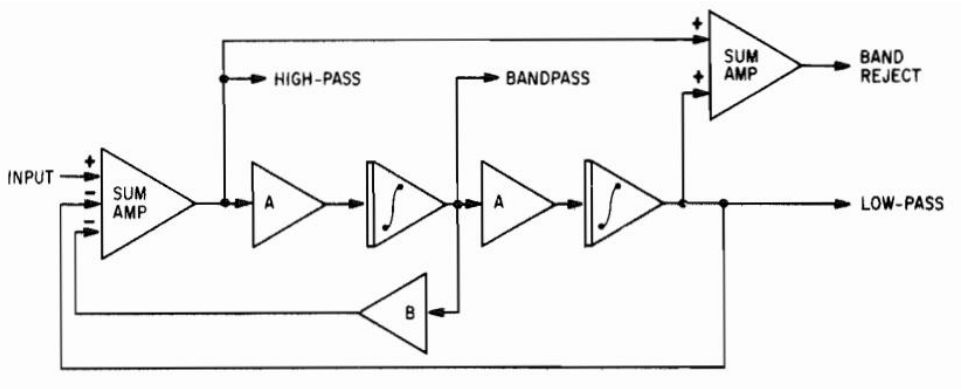


Figura 27: Filtro de variables de estado analógico [23]

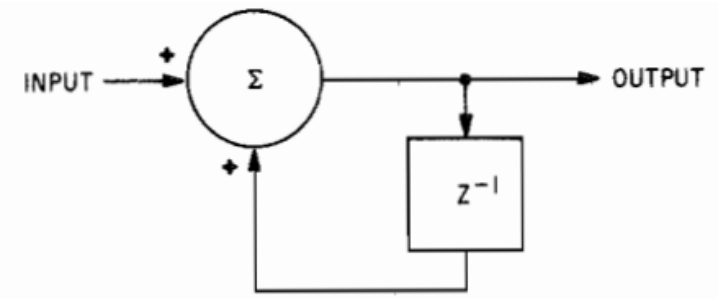


Figura 28: Derivador digital [23]

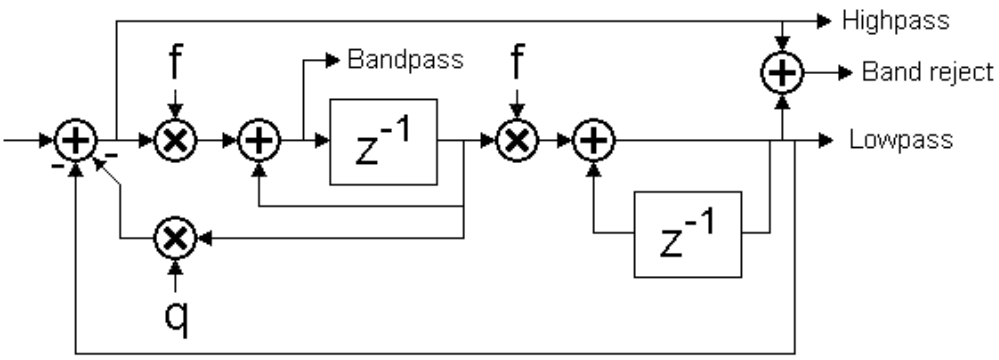


Figura 29: Filtro SVF Digital [32]

La variable *f* originalmente depende de la siguiente fórmula, siendo F la frecuencia de corte querida y Fs la frecuencia de muestreo:

$$f = 2\left(\frac{\pi F}{F_s}\right) \text{[23]}$$

En el filtro analógico equivalente, este valor dependía del valor de un resistor y un capacitor, pero, al pasarlo a formato digital, esta fórmula produce un error a medida que la frecuencia de corte se aproxima a la mitad de la de muestreo. Si bien en este diseño no se alcanzará este valor, se trabajará con la fórmula que incorpora una corrección de dicho error:

$$f = 2\sin\left(\frac{\pi F}{F_s}\right) \text{[23]}$$

La presente fórmula contiene una función no lineal, pero, para el rango de frecuencias que se utilizarán, se puede aproximar sin problemas a una recta, como se muestra en la figura 30.

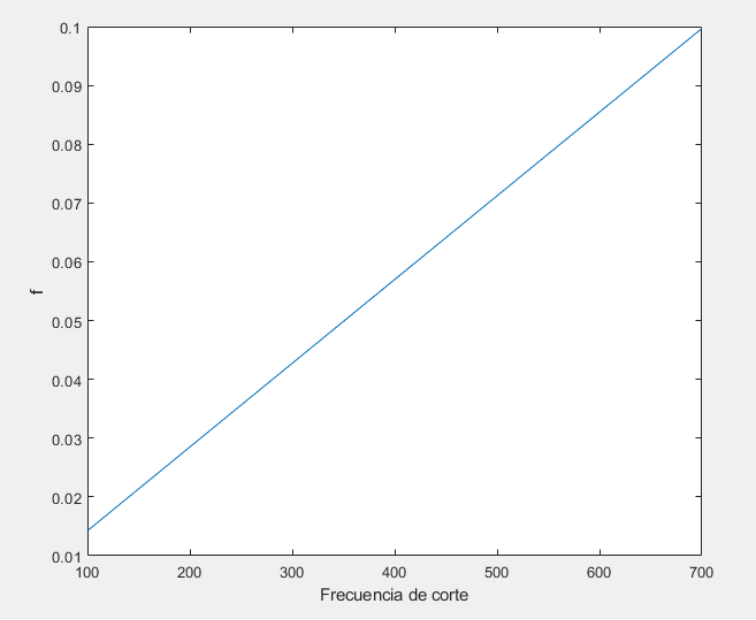


Figura 30: Recta de valores de variable f frente a la frecuencia de corte

El valor de la variable q será la inversa de Q y controlará el nivel de resonancia del filtro, es decir, cuanto se acentuarán las frecuencias más próximas a la de corte, como se muestra en la figura 31.

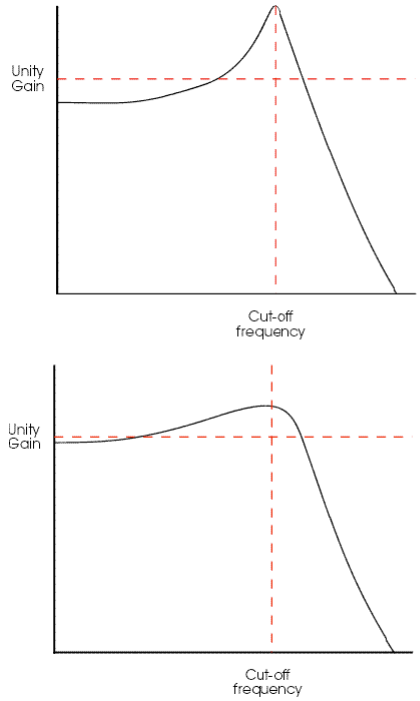


Figure 13: A typical resonant low-pass filter response.

Figure 14: Resonant low-pass filter with low Q.

Figura 31: Representación de filtro con alto y bajo valor de Q, respectivamente [33]

Si bien es un filtro que por su naturaleza IIR puede presentar inestabilidad para ciertos valores de sus parámetros de control, es muy utilizado en aplicaciones de audio por su independencia de control entre la frecuencia de corte y la resonancia, además de poder obtener altos valores de este último, tal y como se puede ver en la figura 32, en la que se representa la transformada DFT (Digital Fourier Transform) del filtro, enfrentada a dos filtros FIR. Se observa que, ante dos filtros FIR de orden 5 y 10, el filtro SVF, de solo orden 2, tiene una respuesta frecuencial más concorde a aquellas requeridas para aplicaciones de audio, sin pérdida de amplitud en la banda de paso, no requiriendo por tanto un amplificador a la etapa de salida.

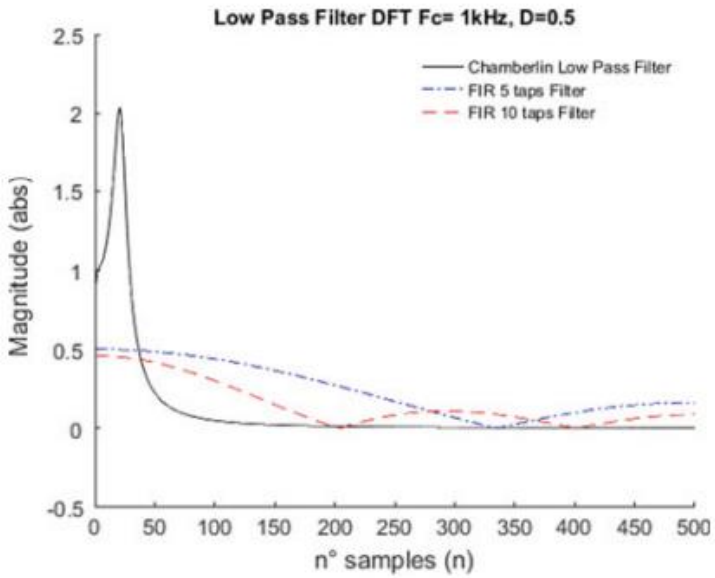


Figura 32: Respuesta en frecuencia del filtro SVF [34]

3.5.2 Implementación en VHDL

El bloque diseñado en VHDL es el representado en la figura 33, y cuyos puertos se describen en la tabla 4.

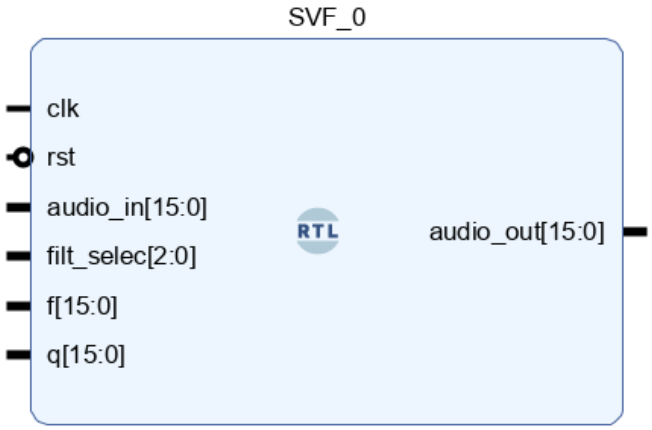


Figura 33: Estructura del bloque Filtro SVF

Puerto	Tipo	Descripción
Clk	Entrada	Reloj del filtro de frecuencia 8*Fs
Rst	Entrada	Señal de reset
Audio_in	Entrada	Entrada de datos de audio
Filt_select	Entrada	Selector del tipo de filtro
f	Entrada	Señal de control de frecuencia de corte
q	Entrada	Señal de control de resonancia
Audio_out	Salida	Salida de datos de audio

Tabla 4: Descripción de puertos Filtro SVF

Este bloque cuenta con una señal de reloj con una frecuencia ocho veces mayor que la del resto de bloques. Esto se debe a la construcción interna del mismo, ya que, en el tiempo que una muestra está presente a su entrada, deben realizarse ocho cálculos internos. La construcción de esta señal de reloj será descrita en el apartado 3.8.1.

El sistema cuenta con dos procesos distintos. El primero de ellos controla que tipo de filtro se aplica en cada vez. Gracias a la construcción de este bloque, este cambio no modifica ninguna parte de la estructura o de los parámetros internos, si no que consiste en modificar de que punto del filtro se extraen las muestras. Se extraerá el dato cada ocho ciclos de reloj.

El segundo proceso se trata del filtro en sí. Aprovechando la velocidad de procesamiento de datos del resto de bloques y las capacidades de la FPGA, se implementará el filtro en una estructura de Pipelining, es decir, se requerirán de ocho pasos distintos para la extracción del resultado final. En vez de realizar todos los cálculos en paralelo, que requeriría de utilizar 3 multiplicadores, se utilizará únicamente uno, implementado de la misma manera que la descrita en el apartado 3.3.2, pero utilizando un valor de Pipelining de 1. Dicho parámetro ajusta que número de registros se situarán a la salida del multiplicador, y, por tanto, determinarán el número de ciclos de reloj que se tardará en obtener el resultado. Para el resto de los multiplicadores, este valor se ha dejado por defecto (de 3), pero en el filtro, se requiere de un control preciso sobre la latencia.
Se crearán las siguientes variables internas para el procesado del filtro, todas ellas con un tamaño de 32 bits, para así poder almacenar y procesar los resultados de las multiplicaciones:

- Z1: Se trata de la salida del primer retardador.
- Sum1_aux1 & sum1_aux2: Variables intermedias donde se almacenará la primera suma del bloque.
- Z2: Almacenará la salida del segundo retardador.
- mA, mB & mP: Señales conectadas directamente al multiplicador, respectivamente, a las dos entradas y al resultado.

La estructura del proceso es la siguiente:

- 1º: Se sitúa a la entrada del multiplicador la variable q y z1. Además, en sum1_aux1 se situará en los bits de mayor peso la muestra de audio de entrada.
- 2º: Se espera un ciclo de reloj hasta la obtención del resultado de la multiplicación.
- 3º: Se introduce en sum1_aux2 la resta entre sum1_aux1, el resultado del multiplicador y z2, obteniendo así el resultado del primer sumador.
- 4º: Se sitúa a la entrada del multiplicador los 16 bits de mayor peso de sum1_aux2 y la variable f.
- 5º: Se espera un ciclo de reloj.
- 6º: Se introducen los 16 bits de mayor peso de z1 y f en el multiplicador, y se registrará sobre la primera la suma entre su anterior valor y el resultado del producto anterior.

- 7º: Se espera un ciclo de reloj.
- 8º: Se almacena en z2 la suma entre su anterior valor y el resultado del producto anterior.

Tras estos ocho ciclos de reloj, el resultado requerido, sea del filtro que sea, se podrá llevar al bloque de salida.

3.6 Bloque I2S

El bloque I2S será el encargado de convertir el flujo de muestras del sintetizador en audio. Para ello, se utilizará el PMOD I2S2 de la empresa Digilentic y el protocolo de comunicaciones I2S.

3.6.1 Protocolo I2S

I2S, también conocido como Integrated Interchip Sound, creado en 1986 por Philips [24], se trata de un protocolo de comunicaciones serie diseñado para transmisión de audio digital. Dicho protocolo consta normalmente de las siguientes señales:

- Reloj maestro: es la señal de mayor frecuencia. Será aquella que mantenga el resto de las señales del bus sincronizadas, en aquellos dispositivos que la requieran. A partir de ahora la definiremos como MCLK.
- Reloj serie o de bit: será el encargado de dictar a qué velocidad saldrá cada bit a transmitir por la línea de datos. A esta señal nos referiremos por SCLK.
- Reloj de palabra o "Left - Right": es el encargado de determinar si la transmisión llevada a cabo en un semiperiodo de este pertenece al canal derecho o izquierdo de la señal estéreo. Este reloj tendrá por nombre LRCLK.
- Línea de datos serie: será la línea a través de la que se multiplexará el dato a transmitir.

No todas las líneas se usarán siempre, y dependerá de la configuración del transmisor y receptor del bus. Un ejemplo de comunicación con este protocolo, solo contando con las señales fundamentales, se muestra en la figura 34. En ella SCLK se le llama SCK.

La configuración de las frecuencias de cada señal de reloj dependerá de la frecuencia de muestreo que utilicemos, el tamaño de palabra y el microchip que implemente el protocolo.

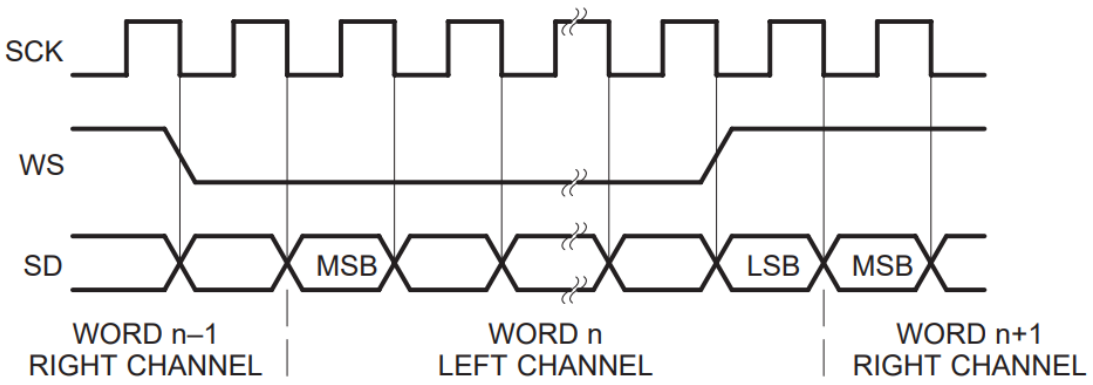


Figura 34: Diagrama temporal del protocolo I2S [24]

3.6.2 PMOD I2S2

El PMOD I2S2, de la empresa Digilentic [4], contiene tanto un conversor analógico-digital como uno digital-analógico, ambos de 24 bits, que permiten, en el mismo dispositivo, muestrear y reproducir audio digital. Contiene dos circuitos integrados, el CS5343, encargado de la conversión A/D, y el CS4244, el que lleva a cabo la conversión D/A, y el que se usará en este trabajo. Ambos ICs reciben y envían los datos mediante el protocolo I2S. El PMOD incluye el circuito necesario para adaptar la señal de tensión necesaria para tanto la lectura como su reproducción en cualquier dispositivo de línea típico, ya sean unos altavoces o auriculares, ya que el fabricante no especifica una impedancia específica a la salida.

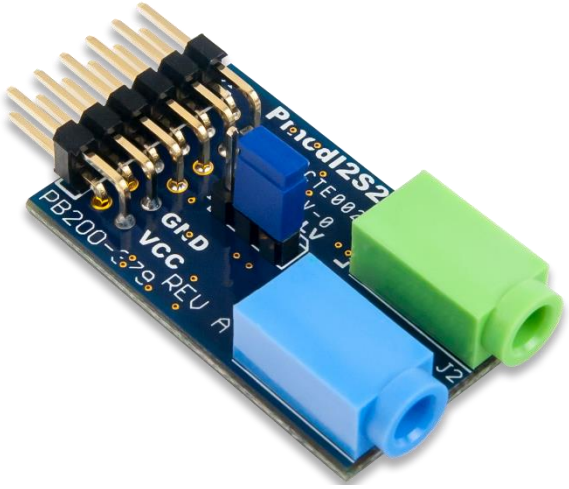


Figura 35: PMOD I2S2

El integrado CS4244 permite dos modos de operación muy similares: maestro o esclavo. En el modo maestro, el chip recibirá la señal de reloj maestra y la de selección de palabra, y calculará la frecuencia más adecuada para la señal de reloj serie, generándola internamente. En el modo esclavo, el convertidor recibirá las tres señales de reloj, que deberán seguir una cierta relación para la correcta operación del dispositivo. Este último será el utilizado en la presente implementación.

El reloj que definirá al resto será el de selección de palabra, WS o LRCLK, ya que su frecuencia deberá corresponderse con la frecuencia de muestreo del sample original. En nuestro caso, esta será de 44.1 KHz. El fabricante del convertidor especifica una serie de relaciones que deben seguir las frecuencias de cada uno de los relojes para el funcionamiento correcto, definido en la tabla 5. Se observa que, para una frecuencia de LRCLK de 44.1 KHz, existen diversas relaciones a elegir. Para el presente diseño, se escogerá una señal de reloj maestra de 11.83 MHz.

Si bien el fabricante no especifica una relación para SCLK respecto al resto de señales, sí que indica dos factores importantes:

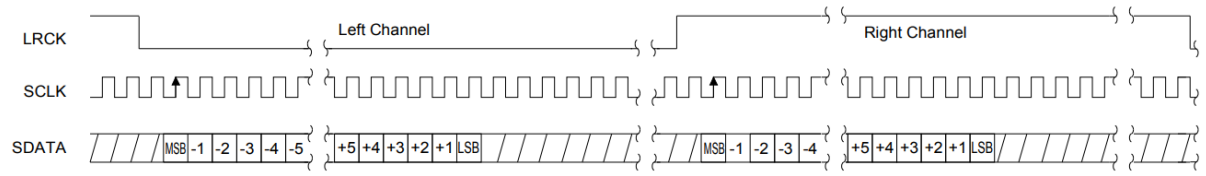
- La relación que sigue internamente para su generación cuando funciona en modo maestro es, como se muestra en la figura 36, de 64 veces la frecuencia de muestreo, es decir, la frecuencia de LRCLK.
- Aquellos bits que no entren dentro de los 24 bits que admite el dispositivo no corromperán la trama, simplemente serán descartados.

Por tanto, se escogerá la relación interna que sigue el componente, resultando en un valor de frecuencia de reloj SCLK de 2.82 MHz:

$$f_{SCLK} = f_{LRCLK} * 64 = 2.82MHz$$

LRCK (kHz)	MCLK (MHz)									
	64x	96x	128x	192x	256x	384x	512x	768x	1024x	1152x
32	-	-	-	-	8.1920	12.2880	-	-	32.7680	36.8640
44.1	-	-	-	-	11.2896	16.9344	22.5792	33.8680	45.1580	-
48	-	-	-	-	12.2880	18.4320	24.5760	36.8640	49.1520	-
64	-	-	8.1920	12.2880	-	-	32.7680	49.1520	-	-
88.2	-	-	11.2896	16.9344	22.5792	33.8680	-	-	-	-
96	-	-	12.2880	18.4320	24.5760	36.8640	-	-	-	-
128	8.1920	12.2880	-	-	32.7680	49.1520	-	-	-	-
176.4	11.2896	16.9344	22.5792	33.8680	-	-	-	-	-	-
192	12.2880	18.4320	24.5760	36.8640	-	-	-	-	-	-
Mode	QSM				DSM		SSM			

Tabla 5: Relación entre MCLK y LRCLK y las frecuencias resultantes en CS4344 [40]



Internal SCLK Mode	External SCLK Mode
I²S, 16-Bit data and INT SCLK = 32 Fs if MCLK/LRCK = 1024, 512, 256, 128, or 64 I²S, Up to 24-Bit data and INT SCLK = 48 Fs if MCLK/LRCK = 768, 384, 192, or 96 I²S, Up to 24-Bit data and INT SCLK = 72 Fs if MCLK/LRCK = 1152	I²S, up to 24-Bit Data Data Valid on Rising Edge of SCLK

Figura 36: Muestra de transmisión y relaciones de SCLK en CS4344 [40]

3.6.3 Implementación en VHDL

El bloque de control I2S y salida de audio, con la forma representada en la figura 37, es realizada en base a un diseño transceptor de I2S [35], es decir, un módulo que recibe muestras por el puerto de entrada del PMOD y las envía a la salida sin modificarlas. Se mantendrá únicamente el aspecto de salida de audio. Los puertos se describen en la tabla 6.

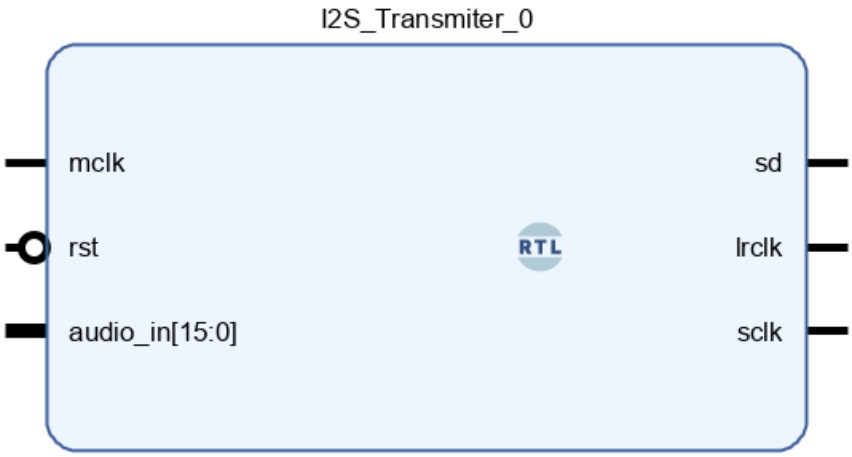


Figura 37: Bloque VHDL de I2S_Transmiter

Puerto	Tipo	Descripción
Mclk	Entrada	Entrada del reloj maestro, de 11.83MHz.
rst	Entrada	Entrada de reset.
Audio_in	Entrada	Entrada de muestras de audio.
sd	Salida	Salida de datos serie de I2S.
lrclk	Salida	Salida de reloj Left-Right, de 44.1KHz.
sclk	Salida	Salida de reloj serie, de 2.82 MHz.

Tabla 6: Puertos del componente I2S_Transmiter_0

Para obtener las distintas señales de reloj a partir de MCLK, se calcularán las relaciones que deben tener MCLK respecto a SCLK:

$$MCLK\_to\_SCLK\_Ratio = \frac{f_{mclk}}{f_{sclk}} = \frac{11.83\ MHz}{2.82\ MHz} \approx 4$$

La relación entre SCLK y LRCLK es la utilizada por el fabricante, como se constataba en el anterior apartado.

El bloque funciona de la siguiente manera: mediante contadores, y siguiendo la relaciones constatadas previamente, se generarán los relojes SCLK y LRCLK, en ese orden. Con cada ciclo de SCLK, se tomará el bit de mayor peso del valor registrado de Audio_in. Tras esto, el mismo registro es desplazado hacia la izquierda, para modificar el bit de mayor peso de cara al siguiente ciclo. Tras pasar los 16 bits, se transmitirán ceros hasta alcanzar las 64 transmisiones. Como se explicaba en el apartado previo, el convertidor solo tomará los 24 primeros bits, teniendo en cuenta el ciclo vacío que debe dejarse tras el cambio de LRCLK, como se muestra en la figura 36.

3.7 Bloque Interfaz de usuario

Este bloque permite al usuario modificar los parámetros internos del sistema mediante el uso de los interruptores que contiene la Nexys4 DDR, y el PMOD ENC [25], de la empresa Digilent.

3.7.1 PMOD ENC

El dispositivo PMOD ENC, de la empresa Digilent [25], se trata de un codificador de eje rotativo, que permite, mediante solo dos señales, comunicar cuantos ”clicks” ha rotado el mando o “knob”. Además, posee un botón, activado al presionar el rotador.

Internamente, el “encoder” contiene dos pulsadores, denominados A y B, normalmente abiertos, con estructura pull-up, es decir, por defecto se lee un nivel alto y situados a 90 grados el uno del otro, tal y como se muestra en la figura 38. Al rotar el “knob”, dependiendo de la dirección, se activará un pulsador antes que el otro. Si se detecta cual de ellos se ha activado antes, puede saberse la dirección de giro.

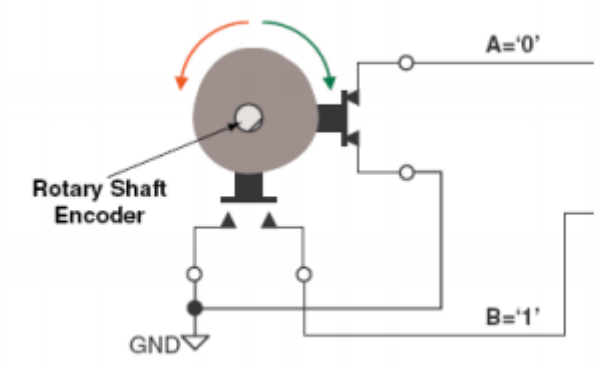


Figura 38: Esquema interno del PMOD ENC

Mediante cada “click” del encoder, se modificará el valor de la variable interna seleccionada una cantidad prefijada. Para la lectura del dispositivo se utilizará como base un código de ejemplo aportado por Digilent, que será explicado más tarde.

3.7.2 Elementos de la Nexys4 DDR

Además del PMOD ENC, se utilizarán los elementos que incluye la propia tarjeta de desarrollo sobre la que se trabaja, la Nexys4 DDR, mostrada en la figura 39, junto a una descripción de sus características en la tabla 7.

Para seleccionar que parámetro se quiere modificar en cada momento mediante el encoder, se utilizarán 10 de los interruptores presentes en la placa. Para activar y desactivar la salida de audio, se usará uno de los pulsadores de propósito general. Por último, se utilizará el botón de reset incluido en la tarjeta.

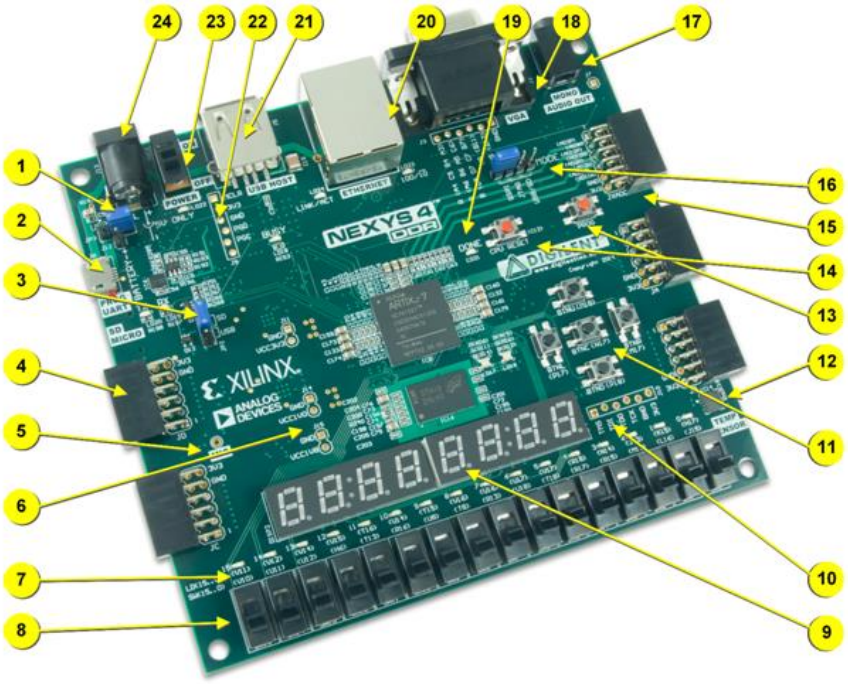


Figura 39: Placa Nexys4 DDR

Número	Component Description	Callout	Component Description
1	Pines de selección de alimentación.	**13**	Botón de reset de configuración de FPGA
2	Puerto UART/JTAG sobre USB	**14**	Botón de reset de la CPU
3	Pines de selección de configuración	**15**	Puerto PMOD para señales analógicas
4	Puertos para PMOD	**16**	Pines de selección del modo de programación
5	Microfono	**17**	Conector de audio
6	Puntos de prueba de alimentación	**18**	Conector VGA
7	LEDs (16)	**19**	LED fin de programación de FPGA
8	Switches	**20**	Conector Ethernet
9	Display de 7 segmentos	**21**	Conector USB
10	Puerto JTAG para cable externo	**22**	Puerto de programación
11	Pulsadores	**23**	Interruptor de alimentación
12	Sensor de temperatura	**24**	Entrada de alimentación

Tabla 7: Características de la placa Nexys4 DDR

3.7.3 Bloque Encoder

El siguiente bloque, cuya estructura y descripción de sus puertos se encuentran en la figura 40 y la tabla 8, respectivamente, se basará en un programa de ejemplo de Digilent [36]. Este presenta en su interior una máquina de estados que controlará la detección de la dirección de giro del encoder, y es representado en la figura 41.

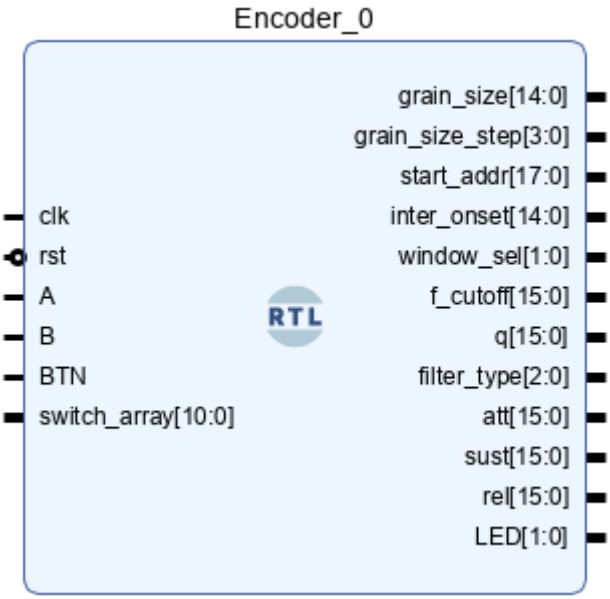


Figura 40: Estructura del bloque Encoder

El proceso en el que se sitúa esta máquina de estados es secuencial y depende de los valores de A y B, las señales del encoder. Un ejemplo de la aplicación de este proceso es mostrado en la figura 41. En ella, según el orden en el que han pasado a nivel bajo, y posteriormente, a nivel alto A y B, se determina que se está desplazando el encoder hacia la izquierda, en este caso correspondiente al estado substract. Además, también se activarán dos leds de la placa, uno para cada dirección, en cada click del encoder.

En otro proceso, ya secuencial, se comprueba la máquina de estados. Si durante un ciclo de reloj el estado es add o substract, se modificará la señal seleccionada según se ha indicado a través del encoder. Si en cambio, se pulsa el botón, la variable volverá a un valor por defecto.

El último proceso modificará que parámetro ha seleccionado el usuario a través de los interruptores de la placa, cuya señal alcanza el bloque “Encoder” a través de la señal switch_array.

Puerto	Tipo	Descripción
Clk	Entrada	Reloj de entrada
Rst	Entrada	Señal de reset
A	Entrada	Señal del encoder
B	Entrada	Señal del encoder
BTN	Entrada	Boton del encoder
Switch_array	Entrada	Señales de interruptores de placa
Grain_size	Salida	Tamaño del grano
Grain_size_step	Salida	Tamaño del grano
Start_addr	Salida	Dirección de inicio de toma de muestras
Inter_onset	Salida	Distancia entre granos
Window_sel	Salida	Selección del tipo de ventana
F_cutoff	Salida	Variable de control de frecuencia de corte del filtro
q	Salida	Variable de control de resonancia del filtro
Filter_type	Salida	Tipo de filtro aplicado
Att	Salida	Atack de envolvente ASR
sust	Salida	Sustain de envolvente ASR
Rel	Salida	Release de envolvente ASR
LED	Salida	Led indicador de dirección de giro

Tabla 8: Descripción de puertos de bloque Encoder

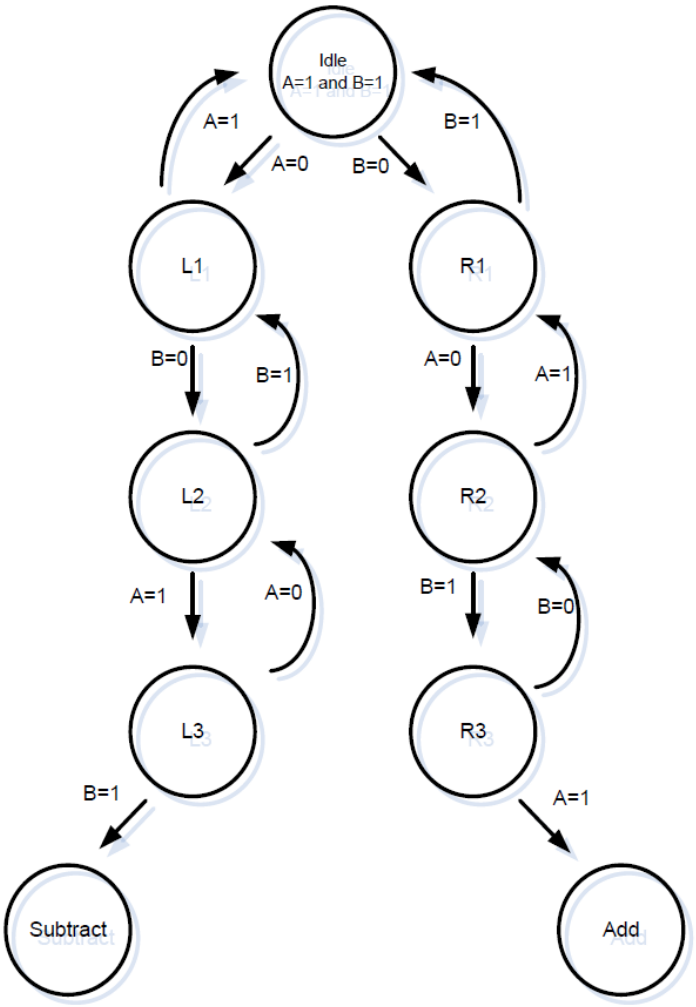


Figura 41: Máquina de estados de Encoder [36]

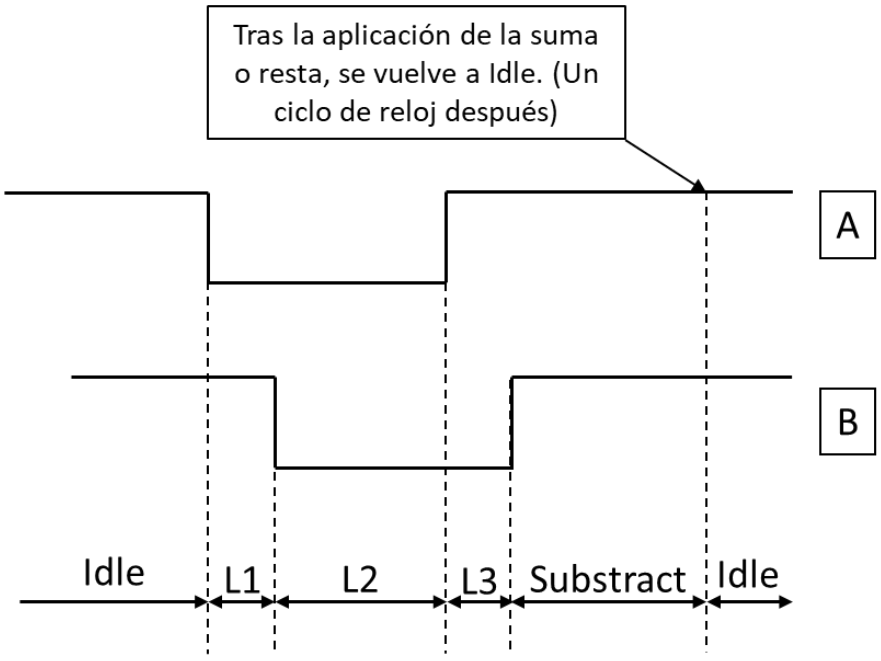


Figura 42: Ejemplo de máquina de estados de Encoder

3.7.4 Bloque antirrebotes

Además del bloque de control del encoder, el proyecto del que se extrae el manejo del PMOD contiene un antirrebotes. Su funcionalidad persigue eliminar el ruido que se produce en la conmutación de los botones internos del dispositivo.

Su funcionamiento consiste en muestrear las señales provenientes del encoder mediante un preescalado de cien cuentas sobre la frecuencia de reloj general del sistema. Con ello, se consigue que la señal que alcance al bloque "Encoder" no tenga conmutaciones residuales, y, por tanto, produzca cuentas y activaciones innecesarias.

3.8 Bloque Top Granular Synth

El último bloque consiste en la unión de todos los diseños individuales creados. Además, incluye un proceso y un bloque IP, que son los encargados de la creación y control de los relojes del sistema.

3.8.1 Generación de señales de reloj

Para este trabajo, como se anticipaba en el apartado del filtro SVF, se requieren de distintas señales de reloj, es decir, de varios dominios de reloj.

Se denomina dominio de reloj a todos los bloques que utilizan la misma señal de reloj [37]. En este sintetizador existirán hasta tres dominios de reloj.

El primero será el del reloj maestro, que será de 11.29 MHz. Esta velocidad es elegida por su facilidad para obtener el resto de las señales de reloj que se requieren, según lo visto en el apartado 3.6.2. Para la generación de dicha frecuencia, se utilizará un IP de Xilinx denominado Clocking Wizard. Se seleccionarán todos los parámetros por defecto, realizando el bloque más simple posible, con una entrada, que será la señal de reloj estándar de la placa, de 100MHz, y la salida con la frecuencia seleccionada. El único bloque del diseño que utiliza esta señal será el de interfaz de usuario.

Los dos relojes restantes son de 44.1 KHz y de 352.8 KHz. El primero será el que utilicen todos los bloques de procesamiento y generación de datos a excepción del filtro SVF, cuyo reloj será el de 352.8 KHz, ocho veces más rápido. La idea de desplazar el flujo de datos al ritmo de la frecuencia de muestreo viene dada por la naturaleza de la síntesis musical. En otras aplicaciones podría interesar una mayor velocidad de procesamiento de datos si no se contempla un flujo continuo, pudiendo situar por tanto una memoria FIFO (First In First Out) en los límites de ambos dominios de reloj [37]. En el presente trabajo, en cambio, no se sabe de primera mano cuanto tiempo estará activa la generación de datos, por lo que podría llegar a desbordarse dicha memoria. Es por esta razón que estos dos últimos relojes deben estar perfectamente sincronizados, evitando así discontinuidades en el audio.

4. Resultados

En este apartado, se detallarán las simulaciones realizadas de cada uno de los bloques para comprobar su funcionalidad.

4.1 Simulación temporal del bloque granulador

4.1.1 Manejador de memoria

Los parámetros de esta primera simulación son los siguientes:

- Tamaño de grano: 70 milisegundos.
- Tiempo entre muestras (Inter Onset): 2646 muestras, o 60 milisegundos. Es decir, existe solapamiento entre los granos.
- Ventana de grano: Parzen.
- Frecuencia de reloj: 44.1kHz.

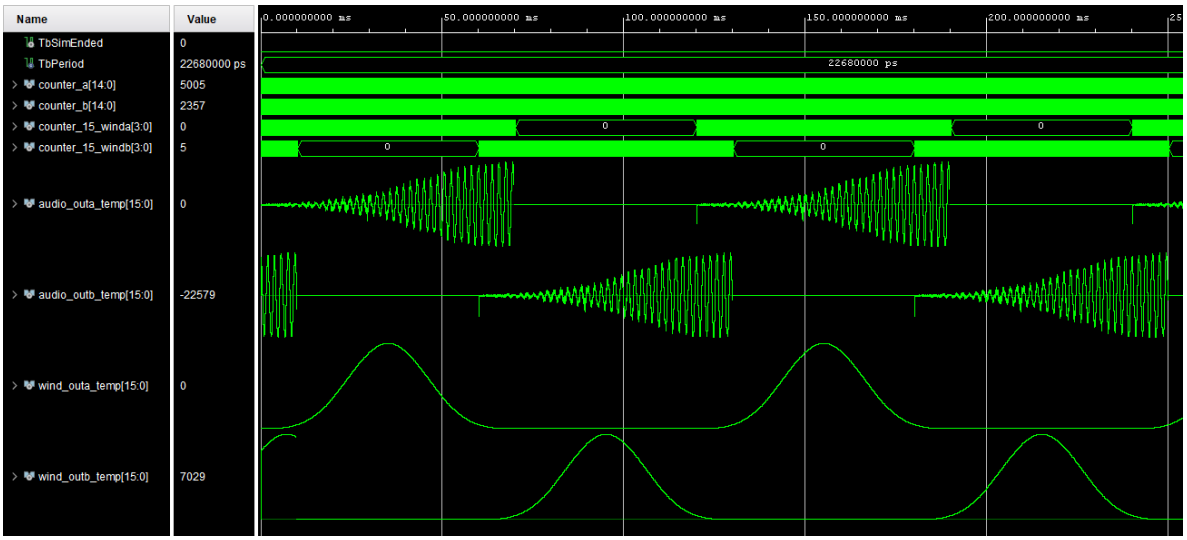


Figura 43: Simulación 1 Manejador de memoria

Tal como se puede observar en la figura 43, el audio se genera correctamente tanto por el canal A como por el B. Al haber solapamiento, se observa como B se inicia en un punto determinado del grano para alcanzar el sincronismo necesario entre ambos flujos. Por último, se observa cómo, en el primer grano de B, la ventana no tiene la forma que debería. Esto se debe a que, al hacer un reescalado del tamaño de la ventana, es complejo calcular exactamente donde debería iniciarse. Esto no supone ningún problema, ya que solo ocurrirá al inicio del flujo, apenas provocará artefactos audibles, y el resto de las ventanas se extraen de manera correcta. Por lo demás, la simulación muestra el comportamiento esperado.

A continuación, se observará un ejemplo sin solapamiento:

- Tamaño de grano: 10 milisegundos.
- Tiempo entre muestras (Inter Onset): 2646 muestras, o 60 milisegundos. Es decir, no existe solapamiento entre los granos.
- Ventana de grano: Triangular.
- Frecuencia de reloj: 44.1kHz.

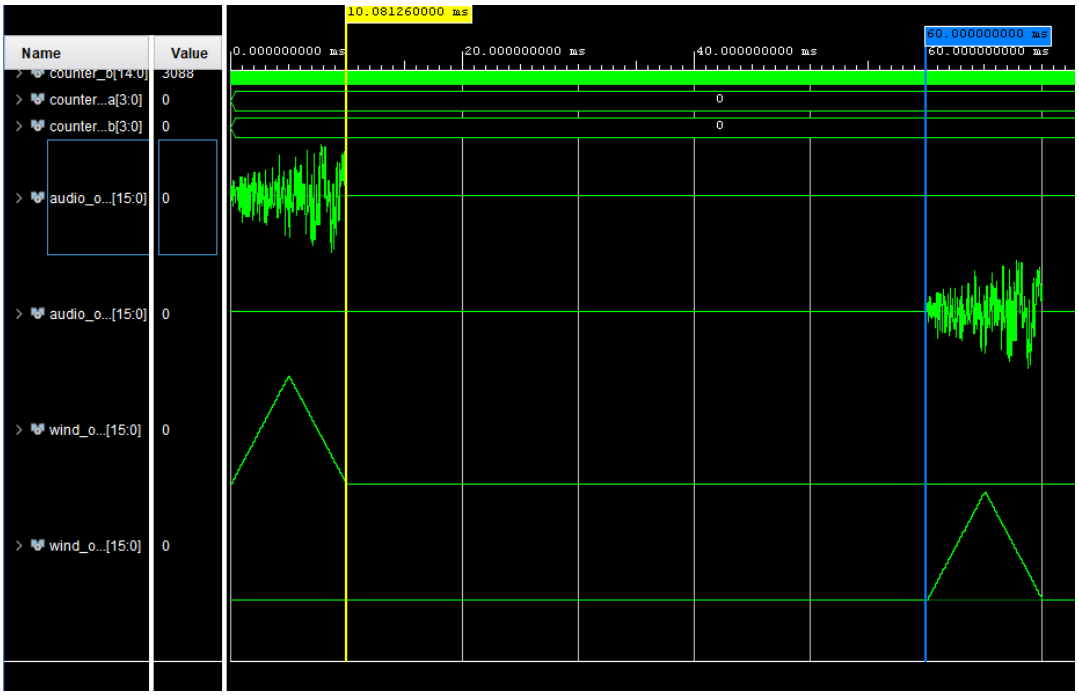


Figura 44: Simulación 2 manejador de memoria

Se puede observar en la figura 44 como en este caso B no comienza a la vez que A, al no haber solapamiento. Además, en este caso, se han incluido marcadores para comprobar que, tanto el tamaño del grano como el tiempo entre muestras se cumple correctamente. Por último, se puede comprobar cómo, aun teniendo el tamaño más pequeño posible, la ventana se reescala correctamente, sin observarse ninguna discontinuidad. De nuevo, se obtienen los resultados esperados según lo diseñado.

Por último, se simulará el funcionamiento del bloque si, en medio de la generación del flujo, se modifica un parámetro.

- Tamaño de grano: 70 milisegundos y pasará a 10 milisegundos.
- Tiempo entre muestras (Inter Onset): 2646 muestras, o 60 milisegundos.
- Ventana de grano: Parzen.
- Frecuencia de reloj: 44.1kHz.

Como se observa en la figura 45, el cambio durante la nota corta por completo el grano de A sobre el que cae, pero, al estar trabajando con elementos de un tiempo muy reducido, apenas degradará el audio que se genere. Tras esto, se concluye que el funcionamiento del bloque es el esperado, a excepción de los dos detalles comentados.

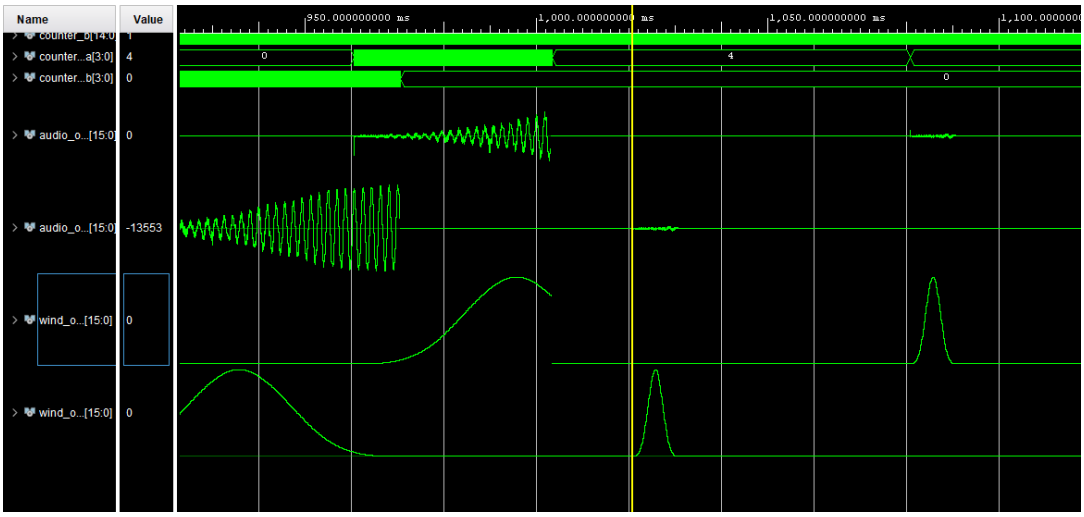


Figura 45: Simulación 3 manejador de memoria

4.1.2 Granulador

Al comprobar el funcionamiento correcto del manejador de memoria, solo se realizará una simulación del granulador, ya que apenas contiene lógica interna y la complejidad recae en el bloque previo.

- Tamaño de grano: 70 milisegundos.
- Tiempo entre muestras (Inter Onset): 1764 muestras, o 40 milisegundos.
- Ventana de grano: Parzen.
- Frecuencia de reloj: 44.1kHz.

En la figura 46, se muestran las salidas de los multiplicadores (mult_out_1 y mult_out_2), sus respectivas entradas de audio, y el resultado final de la suma. Como se puede comprobar, existe el error al inicio de la nota que se comentaba previamente, más el resto de la trama transcurre correctamente y con el solapamiento correcto, además de observar que la aplicación de la ventana es correcta en todos los granos.

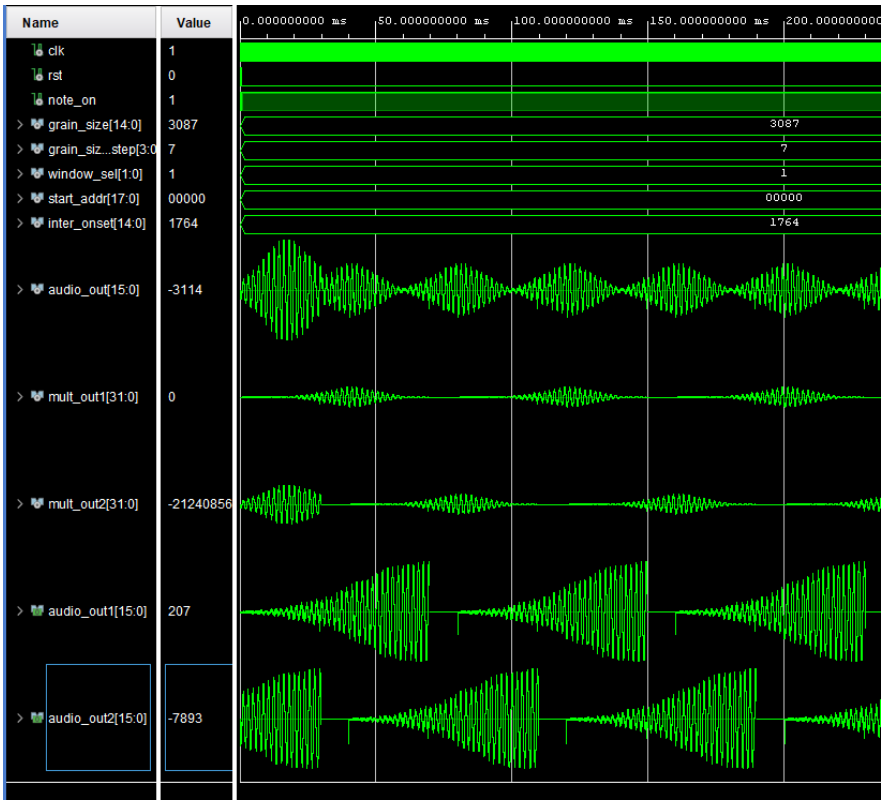


Figura 46: Simulación Granulador

4.2 Simulación temporal del bloque de envolvente ASR

Para la primera simulación de este bloque, se utilizarán los siguientes parámetros:

- Attack: Con valor 1 (lo que implica un tiempo de 371 ms).
- Sustain: 16383.
- Release: Con valor 1 (lo que implica un tiempo de 371 ms).
- Frecuencia de reloj: 44.1 KHz.

El resultado de la simulación es mostrado en la figura 47. Como se anticipó en la descripción del bloque, los tiempos de attack y reléase, son relativos al valor de sustain. La envolvente tiene la forma que se mostraba en la figura 24, por lo que se obtiene el resultado esperado. Cabe destacar que la salida de audio no es mostrada con los límites máximos y mínimos representables por 16 bits, ya que, de ser así, apenas se alcanzaría a ver la onda.

Para la segunda simulación se introducen los siguientes parámetros

- Attack: Con valor 3 (lo que implica un tiempo de 495 ms).
- Sustain: 65535 (el valor máximo).
- Release: Con valor 1 (lo que implica un tiempo de 495 ms).
- Frecuencia de reloj: 44.1 KHz.

En esta ocasión, se ha limitado la visualización de la salida de audio para poder observar con mayor claridad los efectos de la envolvente. Tras esta simulación, se determina que el bloque cumple su función según lo esperado.

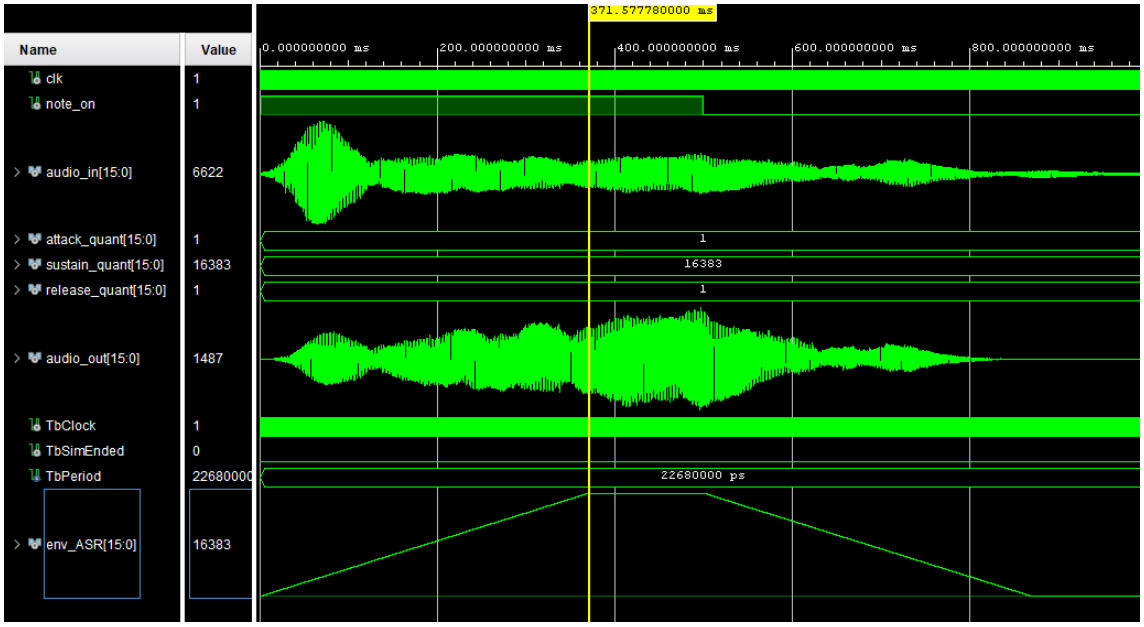


Figura 47: Simulación 1 Envolvente ASR

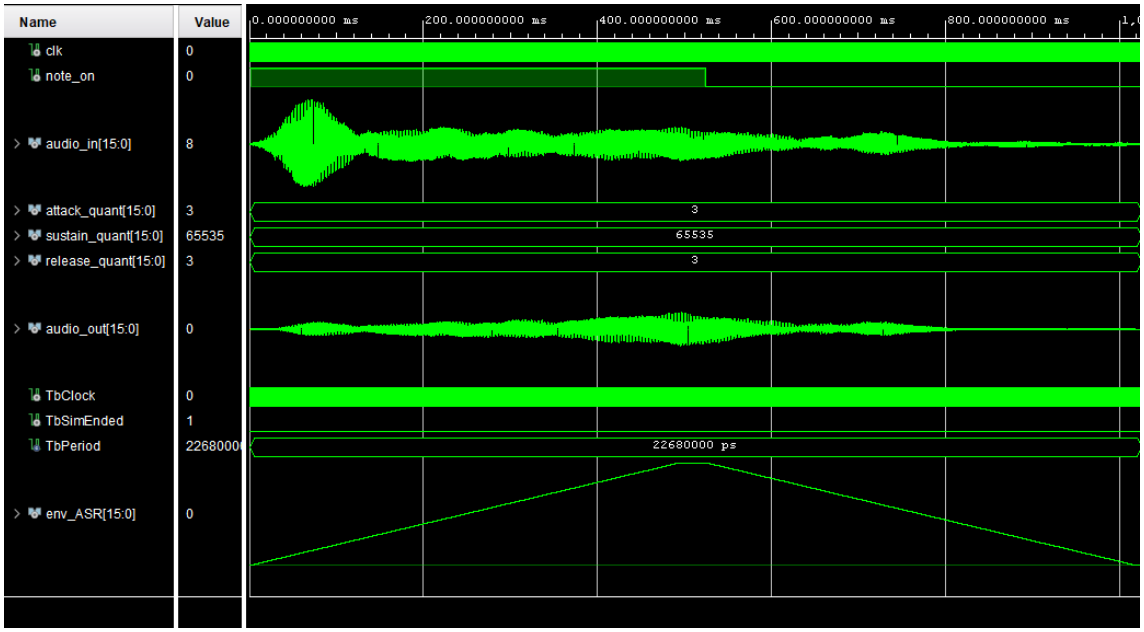


Figura 48: Simulación 2 Envolvente ASR

4.3 Simulación del bloque de filtro SVF

4.3.1 Simulación en Matlab

Previo al diseño en VHDL, el filtro SVF ha sido creado en Matlab, sobre el entorno Simulink, quedando tal y como se muestra en la figura 49.

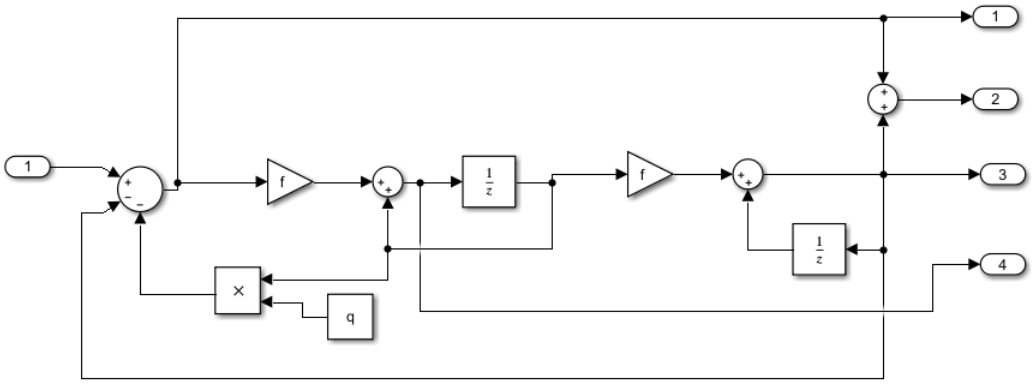


Figura 49: Filtro SVF implementado en Matlab

Para poder comparar las simulaciones realizadas entre Matlab y Vivado, se utilizarán los mismos parámetros:

- Frecuencia de corte: 800 Hz.
- Q: 1.
- Señal de entrada: Senoidal de 200Hz

Los resultados de la simulación se muestran en la figura 50. En ella, se representan, en orden, la entrada al filtro, la salida paso alto, paso bajo, y sobre el mismo recuadro, la salida paso banda y banda eliminada. Se puede comprobar que el funcionamiento del filtro es el esperado, al haber una notable reducción de la señal en paso alto y paso banda, y una leve amplificación en paso bajo y banda eliminada.

Además de la simulación del propio filtro, también se han realizado cálculos de error entre un filtro funcionando con valores en formato de coma flotante y otro con enteros de 16 bits. El resultado es mostrado en la figura 51, en el que se representan los errores entre las salidas de paso alto y paso bajo, respectivamente, de los dos sistemas. El error de cuantificación tiene un valor máximo de 0.7, valor más que asumible para el sistema, y más al estar utilizando valores enteros en el diseño en VHDL.

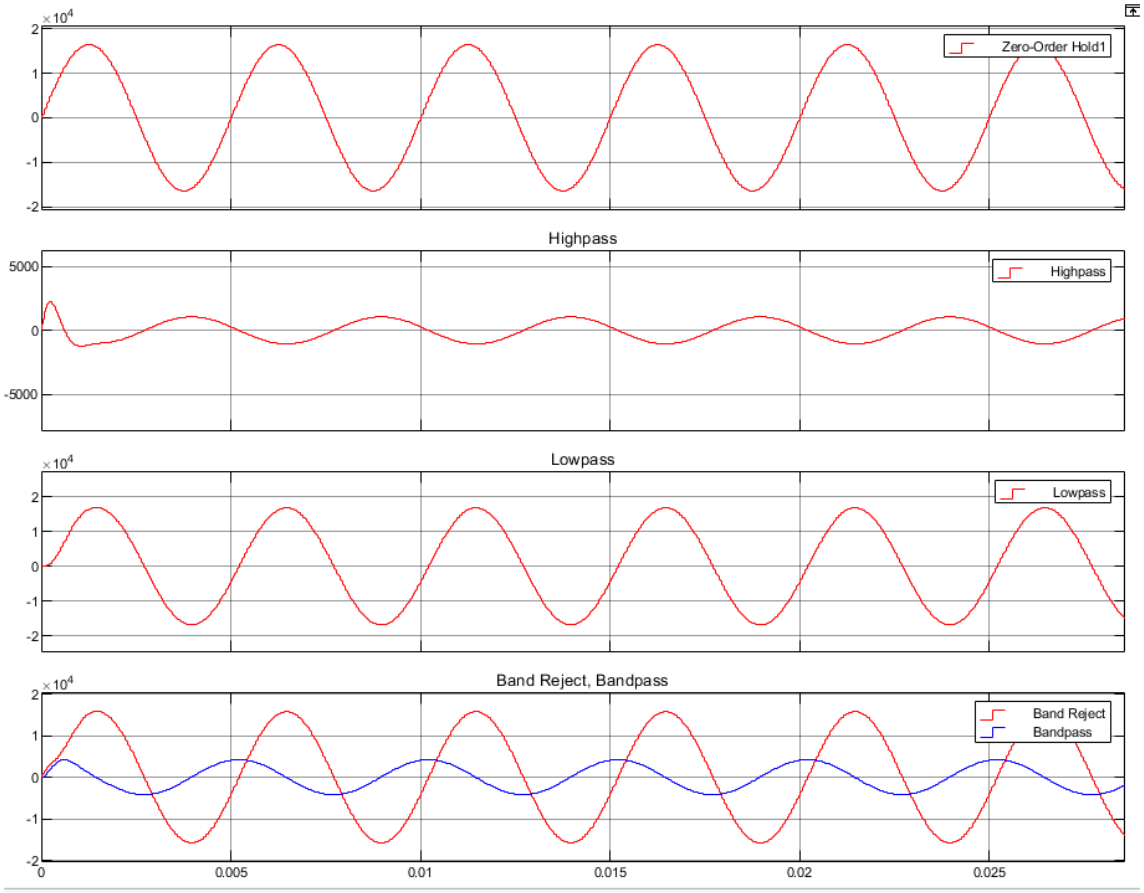


Figura 50: Simulación filtro SVF en Simulink

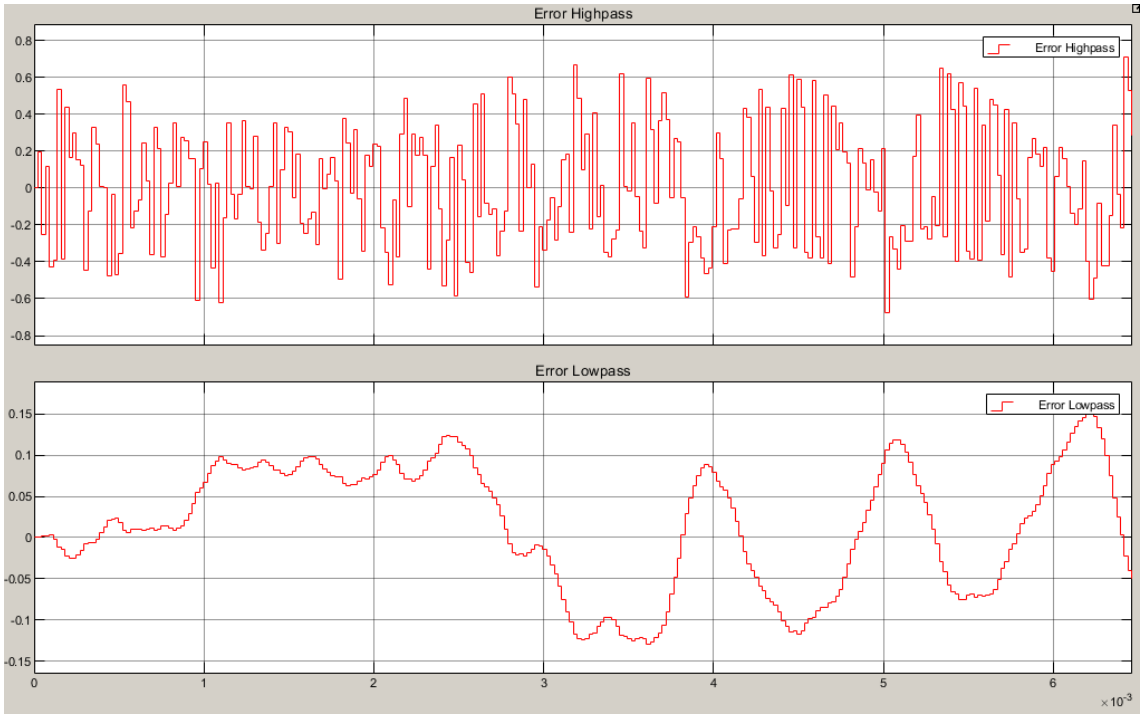


Figura 51: Error cometido en simulación Filtro SVF en Simulink

4.3.2 Simulación en VHDL

Tras variadas pruebas, se ha constatado que, a pesar de haber elegido el presente tipo de filtro por su capacidad de manejar la frecuencia de corte y la resonancia, por la aritmética interna elegida se producen saturaciones para una Q mayor a 1.04, teniendo a la entrada una señal con la mitad de la amplitud máxima permitida. Esto quiere decir que, si el sustain se mantiene bajo, es decir, la amplitud de la señal previa al filtro se recorta, existirá un mayor juego que poder hacer con la resonancia. De lo contrario, al aumentar mínimamente dicho valor se saturará la señal. No obstante, este efecto puede dar un grado más de juego en la capacidad de síntesis del instrumento.

Para la simulación del filtro, se generará mediante Matlab una señal senoidal de 200 Hz, tal y como se explicó en el apartado 3.2.1.

En la primera simulación, se configurará el filtro de la siguiente manera:

- Frecuencia de corte: 800 Hz.
- Q: 1.
- Tipo de filtro: Según simulación.

En la figura 52 se representa el sistema configurado con la salida a paso bajo. Se puede observar como la señal senoidal de salida no se ve disminuida, incluso levemente amplificada. Estos resultados coinciden en gran medida con lo observado en Matlab, si bien la amplificación es menos notable, puede deberse a trabajar enteramente con elementos enteros.

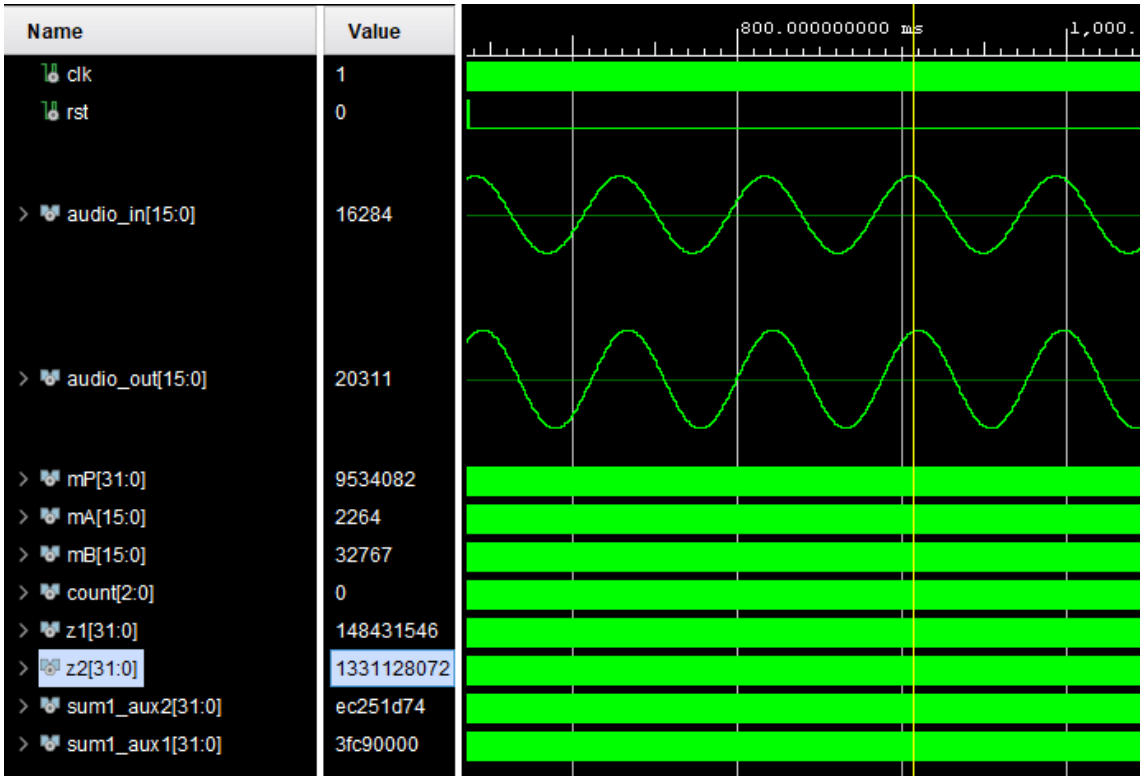


Figura 52: Simulación Paso bajo FIltro SVF

Configurada la salida en paso alto, tal y como se muestra en la figura 53, la señal seno se ve mermada en gran medida. En esta ocasión, el resultado se asemeja más a lo esperado, comparándolo con la simulación ideal en el modelo de Simulink.

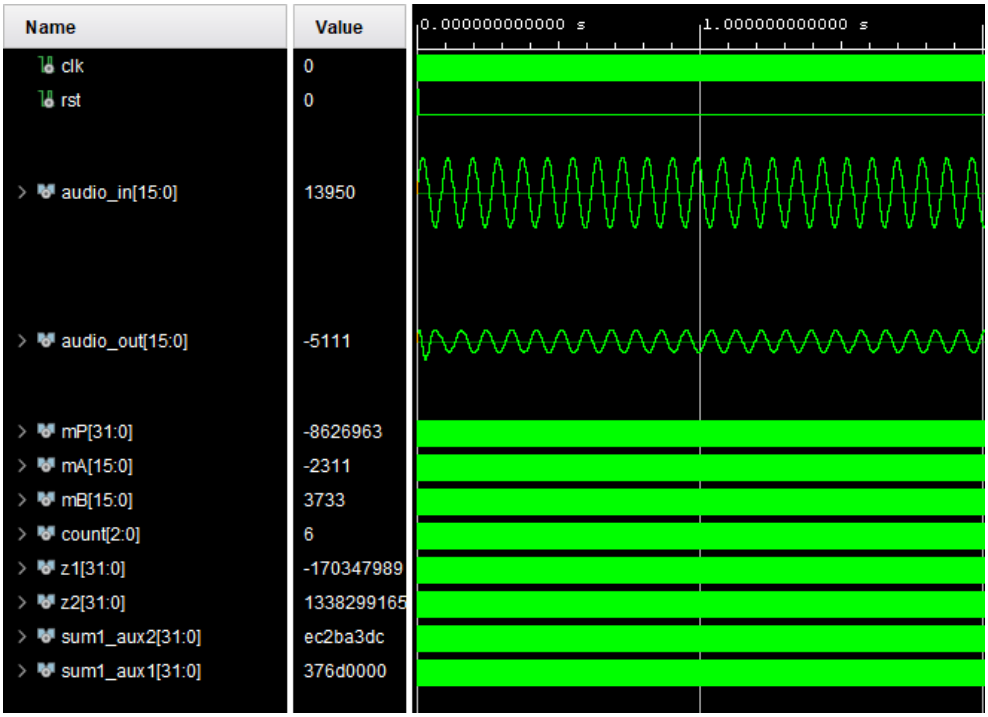


Figura 53: Simulación Paso alto Filtro SVF

En la figura 54 se observa la simulación de salida en paso banda. La reducción no se observa tan claramente como en Simulink, quizás por el fallo introducido por la resonancia que se comentaba al inicio del apartado. A pesar de esta discrepancia, se puede considerar como correcto el funcionamiento de esta salida.

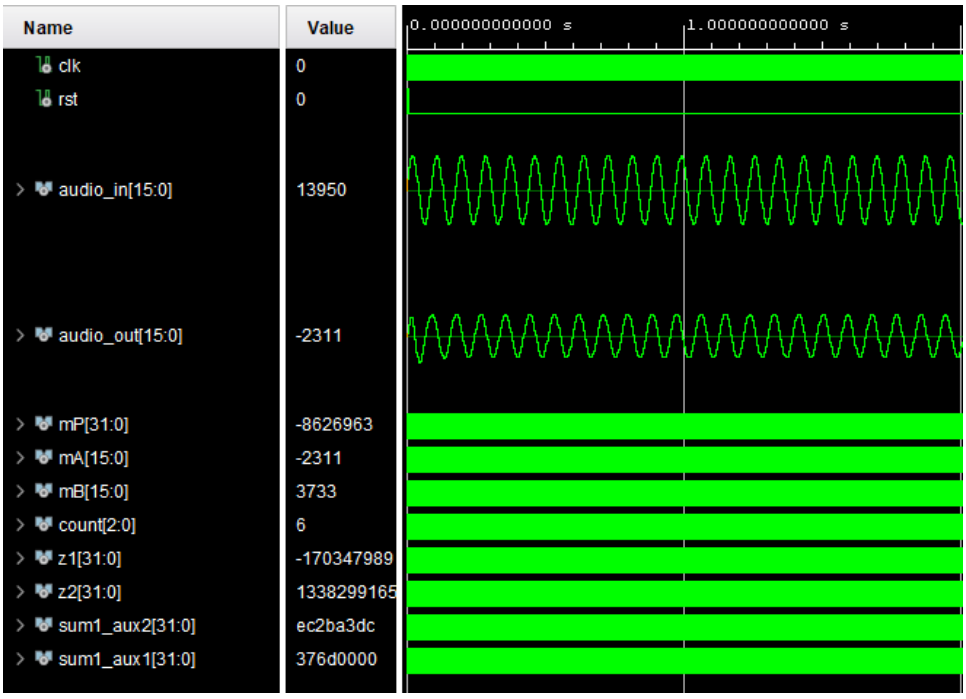


Figura 54: Simulación Paso Banda Filtro SVF

Por último, la salida en banda eliminada, que se puede observar en la figura 55, la señal se ve ligeramente reducida, al igual que se observaba en la simulación en Simulink, por lo que esta salida se toma como validada.

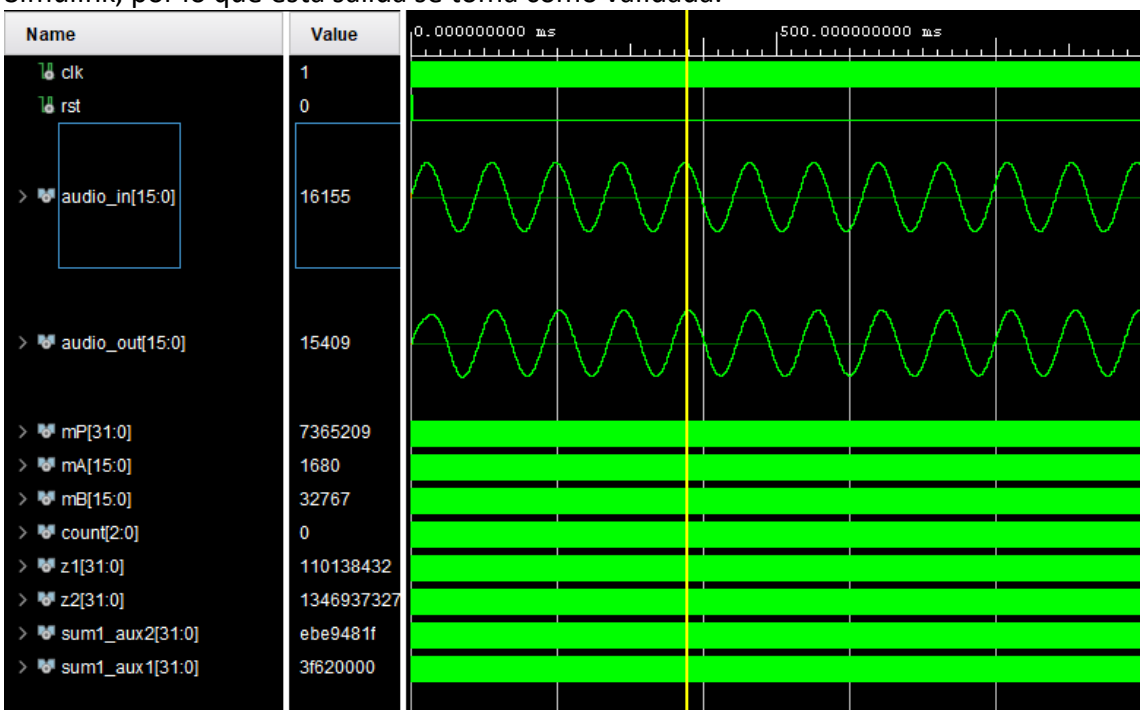


Figura 55: Simulación Banda Eliminada Filtro SVF

Tras lo visto, se puede concluir diciendo que, a pesar de los problemas que puede acarrear la distorsión ocasionada por la saturación, el filtro es totalmente funcional y cumple en gran medida con lo simulado en Matlab.

4.4 Simulación del bloque de transmisión de I2S.

Para la simulación del bloque de transmisión de I2S, se ha utilizado una memoria de Block ROM de prueba. Se segmentará en varias figuras la misma simulación. Debido a la naturaleza del bloque I2S, el único parámetro de entrada es el reloj maestro, que es de 11.29MHz, como se dijo en el apartado 3.6.2.

En la figura 56 se representa una transmisión completa del protocolo I2S. Las señales representadas son las siguientes:

- JA (3): SD, o el bit de transmisión de datos serie.
- JA (2): SCLK, o reloj serie, que dictamina el ritmo de salida de los bits.
- JA (1): LRCLK, o reloj de palabra.
- JA (0): MCLK, o reloj maestro.

Debido a ser el diseño realizado en mono, para ambos flancos del LRCLK, la muestra a transmitir será la misma.

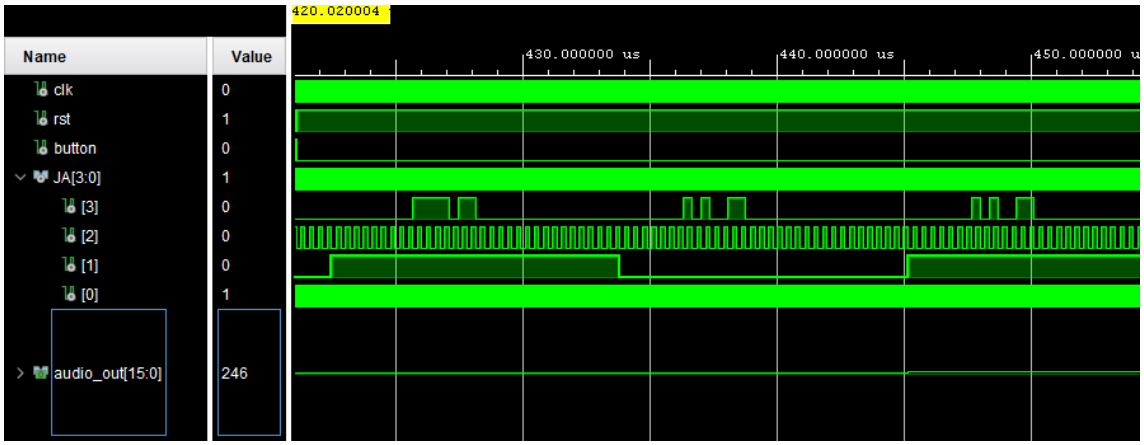


Figura 56: Simulación de Transmisor I2S

Para demostrar con seguridad el funcionamiento correcto del transmisor, se buscará la transmisión de un dato negativo. Esta se muestra en la figura 57. En ella se puede observar cómo, previo a la transmisión del bit de mayor peso, tras el flanco del LRCLK, se espera un ciclo de SCLK, como se explicó en el apartado 3.6.3. Acto seguido, se transmite, desde el bit de mayor peso al menor, los datos de la muestra, dejando vacíos los bits hasta cumplir la cuenta de 32 ciclos de SCLK. Los resultados coinciden con lo esperado según la trama habitual de protocolo I2S, mostrada en la figura 36, por lo que este bloque queda validado.

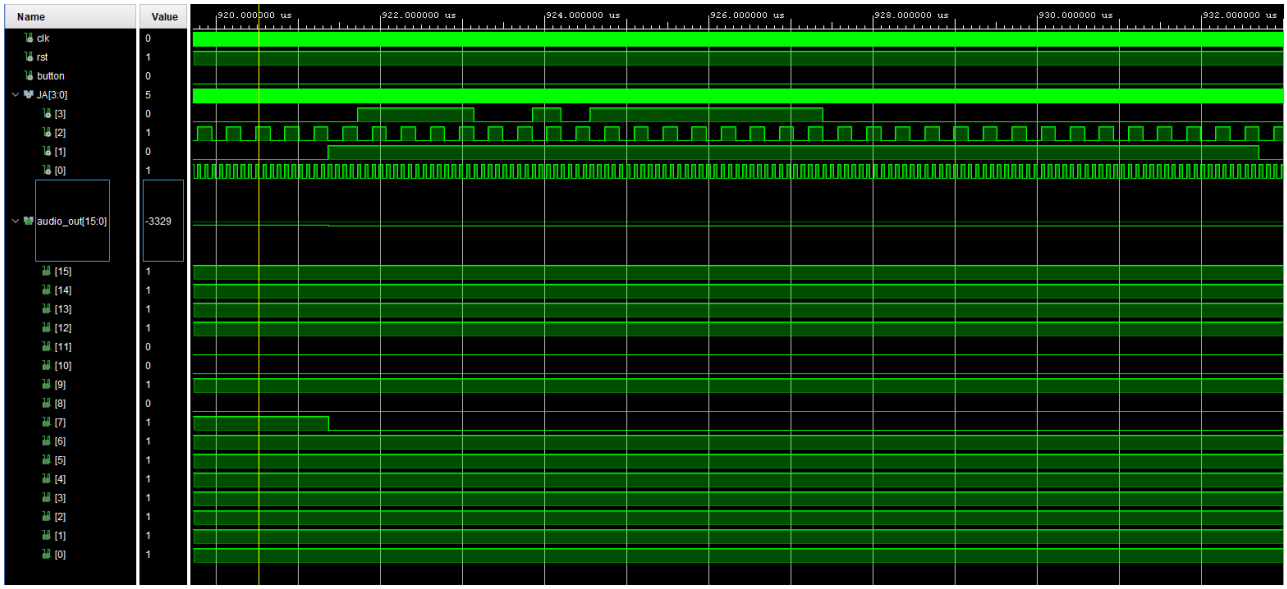


Figura 57: Detalle de simulación de Transmisor I2S

4.4 Simulación del módulo TOP

Previo a la descarga en placa, se realizará una simulación del bloque maestro para comprobar dos aspectos: las señales de los distintos dominios de reloj se generan correctamente y están sincronizadas, y las muestras recorren todo el camino correctamente y se transmiten mediante el protocolo I2S.

En la figura 58 se muestra como estos dos relojes están sincronizados, habiendo 8 ciclos de reloj de clk_svf por cada ciclo de clk_44. Existe el sincronismo necesario para que los distintos bloques trabajen de manera correcta, por lo que este comportamiento es el correcto.

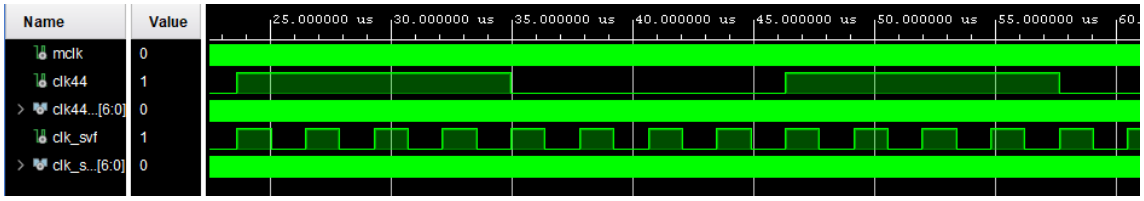


Figura 58: Simulación de módulo TOP Sincronismo de reloj

Para finalizar, en la figura 59 se puede visualizar como los datos van pasando de módulo a módulo correctamente, y como se genera correctamente la señal de note_on_persist, tan importante para que el sistema funcione correctamente de acuerdo con lo especificado en el apartado 3.4. Tras lo observado, se puede determinar que el funcionamiento del sistema completo en simulación es el esperado-

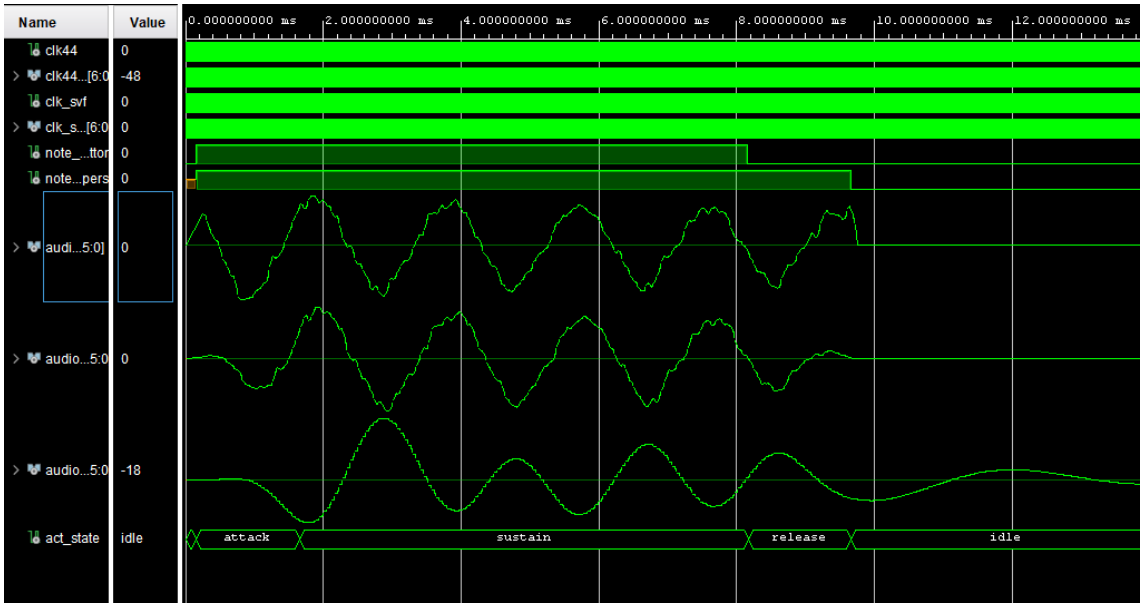


Figura 59: Simulación de módulo TOP Flujo de datos

4.5 Descarga en placa

Tras cerciorarse de que el funcionamiento del sistema, bajo simulación, es el correcto, se procederá a descargarlo en placa y así comprobar si físicamente cumple con las expectativas.

Para la colocación de los dispositivos PMOD, se ha escogido el puerto JA para el PMOD I2S2, mientras que el JB alojará el PMOD ENC. En la figura 60 se muestra el sistema montado.

Previo a la generación y descarga en placa del fichero .bit, debe añadirse al proyecto el fichero de restricciones .xdc, proporcionado por Digilent [38]. En dicho fichero se realizarán las interconexiones entre los puertos del módulo de más alto nivel con los elementos Hardware de la placa.

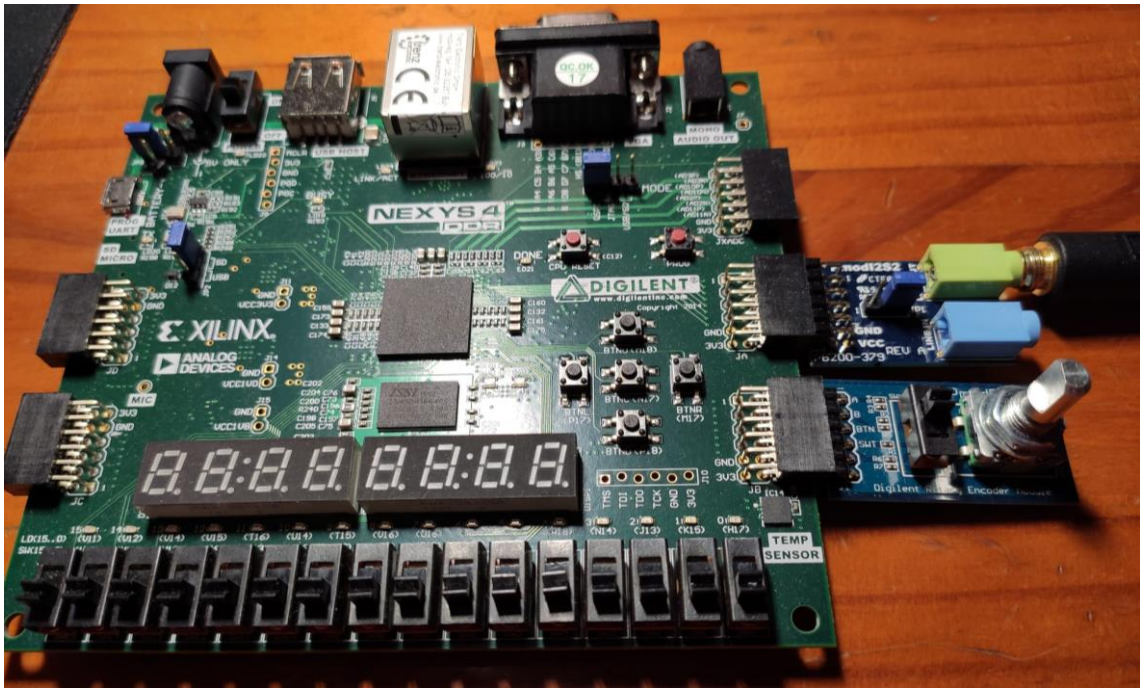


Figura 60: Tarjeta con los PMOD incorporados

Para comprobar el funcionamiento, se utilizarán auriculares conectados a la salida del PMOD I2S2. Tras realizar distintas pruebas, se determina que la modificación de los distintos parámetros mediante los interruptores y el encoder es correcta, si bien para algunos es demasiado lenta, como la frecuencia de corte o la resonancia del filtro, por lo que convendría modificar la cantidad que se suma o resta en cada "click" del PMOD ENC. El audio obtenido es el esperado, pero para las distintas ventanas de grano no se observa diferencia alguna. Los parámetros de control de la generación de granos se modifican correctamente, incluso durante un evento de pulsación. La envolvente ASR se aplica correctamente, apreciándose la modulación en amplitud que la misma ejerce. Por último, el filtro, como se comentaba, cuesta notar el cambio de frecuencia de corte y resonancia, pero el cambio de tipo de filtro es el correcto, observándose una diferencia esperable entre las distintas configuraciones. Finalmente, al estar

obteniendo audio, se puede asumir que la transmisión bajo el protocolo I2S es correcta.

En relación con la utilización de recursos, en casi todos los aspectos la Nexys4 DDR puede alojar el diseño sin ninguna clase de problemas, como se muestra en la figura 61. Ahora bien, al haber utilizado memoria del tipo BRAM para todos los datos que debía albergar el trabajo, casi se ha llegado a utilizar la totalidad de los recursos de este almacenamiento. Es por ello por lo que se reitera en la idea de, en un futuro, la prioridad para mejorar el presente sistema es implementar una memoria externa que permita alojar mayores cantidades de samples.

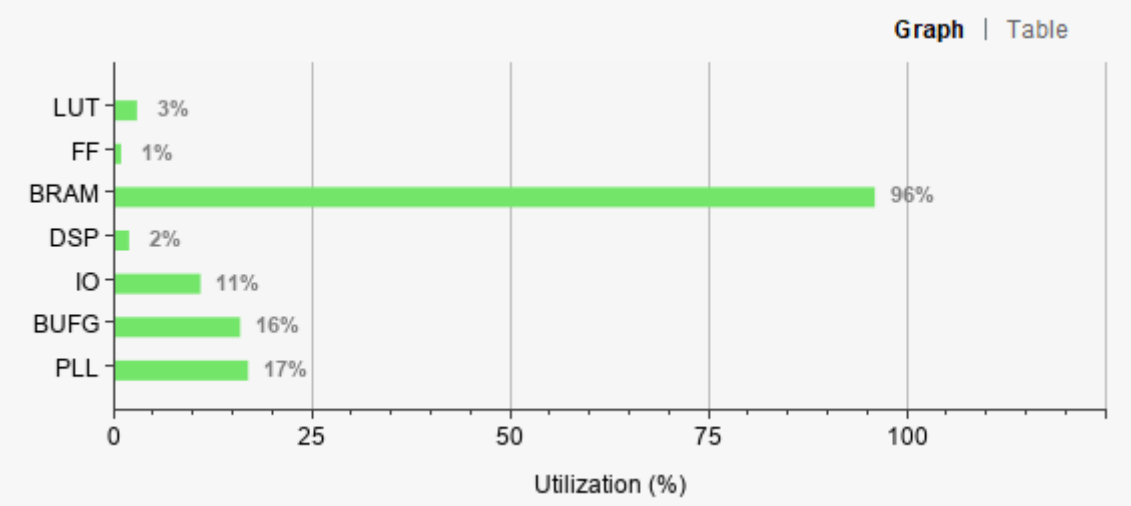


Figura 61: Recursos utilizados en diseño final

5. Conclusiones y trabajo futuro

En el presente trabajo se ha desarrollado un sistema capaz de utilizar la tecnología de síntesis granular para generar sonido, además de incluir elementos altamente usados en cualquier sintetizador, como son el filtro y la envolvente ASR. Los resultados en simulación han sido satisfactorios, a excepción del problema con el desbordamiento de la señal en el filtro, y al probarlo en placa se comprueba que el sistema cumple con las expectativas. Si bien hay ciertos aspectos a mejorar, ya sea añadiendo más envolventes de grano, ya que apenas se nota la diferencia entre ellas, y aumentar el tamaño de la aritmética interna del filtro, para así evitar saturaciones indeseadas, se puede concluir con que el sistema es perfectamente funcional.

En un futuro, este sistema podría mejorarse en aspectos como puede ser añadiendo polifonía, es decir, la capacidad de reproducción de más de una nota simultáneamente, incluyendo soporte para utilizar el formato MIDI, un estándar muy utilizado en la industria musical para interconexión de instrumentos, o añadiendo un banco de efectos variados, que le darían una mayor capacidad de creación al dispositivo. Por último, para evitar quedarse sin espacio de Block RAM, se podría implementar un control de memoria DDR2, para así poder tener incorporados más samples directamente, o bien permitir descargar en dicha memoria desde una tarjeta SD, dándole al usuario la capacidad de modificar los samples de manera sencilla.

6. Bibliografía

[1] Timothy Opie, "Sound in a Nutshell:
Granular Synthesis." , La Trobe University, 1999.

[2] Matteo Di Cosmo, "The Tiny Synth," 2019. Available:
https://thetinysynth.wordpress.com/technical-details/.

[3] (Mar 2,). *The digital state variable filter*. Available:
http://www.earlevel.com/main/2003/03/02/the-digital-state-variable-filter/.

[4] Anonymous (). *Pmod I2S2: Stereo Audio Input and Output*. Available:
https://digilent.com/pmod-i2s2-stereo-audio-input-and-output/.

[5] (Jul 25,). *Trautonium*. Available: http://musiki.org.ar/Trautonium.

[6] (). *Moog Synthesizer 1c/2c/3c*. Available:
http://www.vintagesynth.com/moog/modular.php.

[7] Moog, "Minimoog Model D App," .

[8] Raph Levien *et al*, "Dexed - FM Plugin Synth," vol. 0.9.6, .

[9] Iannis Xennakis, "Analogique B," 1997.

[10] D. Gabor, "Theory of communication. Part 3: Frequency compression and expansion," *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering,* vol. 93, *(26),* pp. 445-457, 1946. . DOI: 10.1049/ji-3-2.1946.0076.

[11] C. Roads, "Introduction to Granular Synthesis," *Computer Music Journal,* vol. 12, *(2),* pp. 11-13, 1988. Available: http://www.jstor.org/stable/3679937. DOI: 10.2307/3679937.

[12] B. Truax, "Real-Time Granular Synthesis with a Digital Signal Processor," *Computer Music Journal,* vol. 12, *(2),* pp. 14-26, 1988. Available:
http://www.jstor.org/stable/3679938. DOI: 10.2307/3679938.

[13] (). *Peak Explained*. Available: https://novationmusic.com/es/node/104.

[14] (). *Kyra Features*. Available: https://waldorfmusic.com/en/kyra-features.

[15] René Ceballos. (). *The XVA1 Synthesizer Module*. Available:
https://www.futur3soundz.com.

[16] D. Schwarz, "State of the art in sound texture synthesis," in September 2011, Available: https://hal.archives-ouvertes.fr/hal-01161296.

[17] D. Keller and B. Truax, "Ecologically-based granular synthesis," in *Icmc,* 1998, .

[18] Paul Nasca. (). *Algorithms created by me - Paul Nasca*. Available:
http://www.paulnasca.com/algorithms-created-by-me.

[19] Griffin Brown. (Feb 5,). *The Basics of Granular Synthesis*. Available: https://www.izotope.com/en/learn/the-basics-of-granular-synthesis.html.

[20] A. Kupryjanow and A. Czyzewski, "Time-scale modification of speech signals for supporting hearing impaired schoolchildren," in October 26, 2009, .

[21] Tasty Chips Electronics. (Mayo 8,). *GR-1 Granular synthesizer*. Available: https://www.tastychips.nl/product/gr-1-granular-synthesizer/.

[22] Ross Bencina, *Implementing Real-Time Granular Synthesis.* 2006Available: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=bGrVr5UAAAAJ&citation_for_view=bGrVr5UAAAAJ:Tyk-4Ss8FVUC.

[23] H. Chamberlin, *Musical Applications of Microprocessors.* Sams, 1980.

[24] "I2S Bus Specification,"Philips Semiconductors, , FebruaryPhilips Semiconductors, "I2S Bus Specification,", February, 1986.

[25] (). *Pmod ENC*. Available: https://digilent.com/reference/pmod/pmodenc/start.

[26] (). *Nexys4 DDR*. Available: https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual?s[]=nexys4&s[]=ddr.

[27] Anonymous (-09-10T15:23:06Z). *Window function*. Available: https://en.wikipedia.org/w/index.php?title=Window_function&oldid=1043530672.

[28] Xilinx, "Block Memory
Generator v8.4 LogiCORE IP Product Guide," Aug 6, 2021.

[29] Xilinx, "7 Series FPGAs Data Sheet: Overview," 8 Sep, 2020.

[30] Xilinx, "7 Series DSP48E1 Slice User Guide (UG479)," Mar 27, 2018.

[31] (03/28/19). *Learning Synthesis: Envelopes
Part One: ADSR, AD, AR, & Friends* . Available: https://www.perfectcircuit.com/signal/learning-synthesis-envelopes-1.

[32] (Sep 26,). *SVF: Digital State Variable Filter*. Available: http://www.fpga.synth.net/pmwiki/pmwiki.php?n=FPGASynth.SVF.

[33] Gerwin de Groot, "biquad - Design a Filter with Resonance," 2014. Available: https://dsp.stackexchange.com/questions/19262/design-a-filter-with-resonance.

[34] A. Ricci *et al*, "Chamberlin state-variable filter structure in FPGA for musical applications," in *Applications in Electronics Pervading Industry, Environment And,* 2019, .

[35] Scott Larson, "I2S Transceiver (VHDL)," Mar 19, 2019.

[36] Michelle Yu, "PMOD ENC Demo Project," vol. 0.01, Jun 17, 2011.

[37] (). *Crossing Clock Domains in an FPGA*. Available: https://www.nandland.com/articles/crossing-clock-domains-in-an-fpga.html.

[38] (). *XDC Files*. Available: https://github.com/Digilent/digilent-xdc/.

[39] (). *Hex Editing for Archivists (Part 2)*
How to Read & Edit Binary Data. Available: http://www.av-rd.com/knowhow/data/hexedit_archivists-2.html.

[40] Cirrus Logic, ""10-Pin, 24-Bit, 192 kHz Stereo D/A Converter" CS4344/5/8," Jul, 2013.

Universidad de Alcalá
Escuela Politécnica Superior

Universidad
de Alcalá