UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

"Diseño y prototipado de un sintetizador digital basado en una arquitectura system on chip (SoC)"

TRABAJO FINAL DE GRADO

Autor/a:
Juan Carlos Rodríguez Vercher

Tutor/a:
Javier Valls Coquillat
Germán Ramos Peinado

GANDIA, 2019

Resumen

En el presente trabajo se aborda el diseño y prototipado de un sintetizador digital de audio sobre una arquitectura *System on Chip* (SoC). El diseño se plantea sobre la herramienta Simulink® y, mediante *plug-ins* para generación automática de código, se convierte el modelo visual del sistema en código C, desplegable en el procesador *software,* y en código VHDL, sintetizable para su despliegue en el chip FPGA, ambos integrados en el SoC de la placa de desarrollo utilizada.

En primer lugar, se aborda el diseño de las etapas de la cadena de síntesis, valorando diferentes opciones de diseño en cada una de ellas, con el fin de obtener las características sonoras deseadas. Tras validar el diseño inicial, se plantea su separación en dos bloques de procesado *software* y *hardware*, para posteriormente realizar la conversión a precisión finita del bloque *hardware*. Tras la comprobar que el nuevo modelo mantiene las prestaciones del modelo original, se genera automáticamente el código y se despliega en el SoC de la placa de desarrollo, donde se realiza la validación final del sistema. El resultado final es un sintetizador digital de audio implementado en un SoC, aprovechando las ventajas del procesado *software* y de la lógica programable.

Palabras clave: Síntesis digital, Procesado de digital de audio, System on Chip (SoC), Modelado visual, ARM®, FPGA, HDL, Simulink®.

Abstract

The present work approaches the design and prototyping of a digital audio synthesizer based on a System on Chip (SoC) architecture. The design is leveraged on the Simulink® tool and, by means of plug-ins for the automatic code generation, the system visual model turns into C code, deployable on the software processor, and into synthesizable VHDL code, for its deployment in the FPGA chip, both integrated in the SoC of the development board.

First, the design of the stages of the synthesis chain is carried out, evaluating different design options for each one of them, in order to achieve the desired sound characteristics. After validating the initial design, the system is partitioned into 2 blocks: a processing software block and a hardware logic block, for later carrying out the conversion to finite precision of the hardware block. After checking that the new model maintains the features of the original model, the code is automatically generated and deployed in the SoC of the development board where the final validation of the system is carried out. The final result of the work is a digital audio synthesizer implemented in a SoC, taking advantage of the processed software and the programmable logic.

Keywords: Digital synthesis, Digital Audio Processing, System on chip (SoC), Visual modeling, ARM®, FPGA, HDL, Simulink®.

***Agradecimientos*:**

Agradecer especialmente a mi familia y a mis amigos su apoyo incondicional en todas mis aventuras. Sin él, nada de esto habría sido posible.

A mis tutores, Javier y Germán, por compartir conmigo parte de su infinita sabiduría y por su apoyo e implicación en los momentos difíciles.

A David, por contagiarme su pasión por el audio, por colaborar en la concepción del trabajo y por dejarme su placa, sé que para un friki no es fácil desprenderse de sus juguetitos.

A Romina y Jesús, por su apoyo para desarrollar mis inquietudes fuera del B015.

A todos los compañeros y profesores de la EPSG que, durante estos años, han hecho que en mi vuelta a las aulas me sintiera como en casa.

¡Muchas gracias a todos!

Índice de contenido

Índice de tablas

Índice de figuras

1. Introducción y objetivos

1.1 Introducción

Desde la invención de la síntesis digital a finales de los años 50 ([1]), las técnicas y sistemas empleados para su implementación han permanecido en continua evolución, en busca de un aumento de la calidad sonora, de la complejidad de los esquemas de síntesis planteados y de la polifonía, todo ello en tiempo real.

En la actualidad existen multitud de sintetizadores digitales implementados como *plug-ins software*, que se ejecutan en ordenadores personales, y sintetizadores digitales *hardware*, basados en arquitecturas con chips DSP (*Digital Signal Processing*), todos ellos con una alta capacidad de procesamiento secuencial.

A lo largo de los últimos años, los diseños basados en FPGAs (*Field Programmable Gate Array*) han penetrado de forma contundente en el sector del audio digital, en parte gracias a la llegada de dispositivos *System on Chip* (SoC), que incorporan FPGAs integradas junto con procesadores ARM®. Estos chips posibilitan el reparto de las tareas entre ambas tecnologías *software*/*hardware*, proporcionando los mecanismos de comunicación pertinentes entre éstos para permitir su cooperación de forma eficiente. El uso de este tipo de arquitecturas se ha convertido en una opción realmente atractiva para el procesado digital del audio en tiempo real, especialmente en aquellos sistemas que requieren de un procesado en paralelo de la señal. Estos sistemas aprovechan la flexibilidad proporcionada por las FPGAs a la hora de replicar bloques del sistema para ofrecer un elevado grado de paralelismo en el procesado del audio, manteniendo siempre una latencia prácticamente nula. En la actualidad, están empezando a aparecer los primeros sintetizadores comerciales basados en este tipo de arquitecturas, que aprovechan dicho paralelismo para ofrecer un elevado nivel de polifonía, con posibilidad de disponer de un complejo esquema de síntesis en cada una de las voces, o aumentar el número de timbres independientes generables simultáneamente.

Tal y como es de esperar, los diseños basados en este tipo de arquitecturas presentan una elevada complejidad a la hora de implementarlos, puesto que requieren de una clara separación de funciones y de su implementación en diferentes lenguajes, como pueden ser los lenguajes C y Verilog/VHDL, así como un profundo conocimiento de los protocolos y técnicas de intercomunicación entre los diferentes componentes del sistema para lograr una sincronización precisa y eficiente entre ellos. Afortunadamente, en la actualidad existen herramientas de alto nivel que permiten transformar diseños de bloques en código ejecutable y sintetizable, permitiendo concentrar los esfuerzos en el diseño funcional del sistema, abstrayendo de muchas de las complejidades inherentes a la arquitectura destino, al menos en la etapa de prototipado del sistema.

A lo largo del presente documento se describe el proceso seguido durante el diseño y modelado de un sintetizador digital de audio en Simulink® y su posterior despliegue en una placa de desarrollo que incorpora un SoC y un codificador digital de audio, entre otros componentes. Para ello se han utilizado los *plug-ins* disponibles para esta herramienta, que permiten la generación automática tanto del código *software* en C como del código de descripción de *hardware* en VHDL, a partir del modelo de bloques del sistema, quedando éste dividido en dos grandes bloques: una parte de procesado *software*, que realiza la configuración inicial del

sistema y permite el ajuste interactivo de los parámetros por parte del usuario, y otra parte *hardware*, de procesado lógico, en la que se implementan los diferentes componentes del sintetizador. Una vez generado el código en VHDL, mediante la herramienta Xilinx Vivado®, se sintetiza el código y se genera el *bitstream*. Finalmente, tanto la parte *software* como la *hardware* se despliegan sobre un SoC basado en la arquitectura Zynq 7000 de Xilinx®, de forma que el resultado final del presente trabajo es un sintetizador digital de audio, parametrizable en tiempo real.

Durante la etapa de diseño se plantean diferentes opciones para implementar cada uno de los subsistemas utilizados en la cadena de síntesis, con el fin de obtener las características sonoras deseadas, y se abordan los problemas habituales inherentes a la síntesis digital, utilizando para ello técnicas de procesado digital de la señal de audio.

Para cada uno de los bloques del sintetizador se estudian sus requerimientos para su implementación como un sistema de precisión finita, ajustando los formatos de los parámetros de entrada, de salida, así como de los resultados de las operaciones intermedias. Además, también se presentarán las diferentes pruebas realizadas para validar su correcto funcionamiento.

Para finalizar el trabajo, se analizarán los resultados obtenidos a nivel de utilización de recursos, con el fin de valorar la viabilidad de este tipo de enfoque de cara al diseño de sistemas finales y determinar los límites del sistema sobre la placa de desarrollo utilizada durante el presente trabajo.

1.2 Objetivos

A lo largo del presente trabajo se abordan aspectos relacionados con la síntesis de audio, el procesado digital de la señal, el diseño de sistemas digitales de precisión finita, el diseño y validación de sistemas con Simulink® y las arquitecturas SoC que integran FPGAs.

El objetivo principal del trabajo es el estudio de cada uno de los bloques que forman habitualmente la cadena de síntesis de audio y las peculiaridades de su diseño como un sistema digital de precisión finita. Para cada bloque se plantea un diseño y, a partir del análisis de los resultados obtenidos, se analizan posibles mejoras que, en algunos casos se han llevado a cabo y en otros se han dejado para futuras líneas de trabajo, tal y como se irá detallando a lo largo del documento.

Como segundo objetivo, se evalúa el método de trabajo planteado para el diseño y prototipado del sistema sobre una arquitectura SoC con FPGA, que posteriormente puede ser adaptado para cualquier otro tipo de sistema digital. Para ello, se valora la viabilidad de su diseño utilizando únicamente las herramientas de modelado visual proporcionadas por Simulink® y su posterior conversión automática a código *software* y de descripción de *hardware*, mediante el uso de las herramientas de generación automática de código Embedded Coder® y HDL Coder™.

Por lo tanto, el objetivo del presente trabajo no es el diseño de un producto final, sino el de un prototipo que permita estudiar las peculiaridades de un sistema de síntesis digital de audio y de las arquitecturas SoC, con procesado *software* y *hardware*. Con el fin de poder continuar evolucionando el presente trabajo en el futuro, en el apartado *Futuras líneas de trabajo* del presente documento, se presentan posibles mejoras que pueden ser llevadas a cabo, tanto para mejorar la calidad del diseño planteado o como ampliar sus funcionalidades.

1.3 Estructura del documento

El presente documento está estructurado en 7 capítulos. Una vez contextualizado y definido el alcance de este trabajo, en el capítulo 2 se realiza una introducción a los sistemas basados en arquitecturas SoC, se describe el entorno de desarrollo, tanto el *hardware* utilizado como las herramientas *software*, y la metodología seguida durante el desarrollo del trabajo.

En el capítulo 3 se describen las características técnicas y funcionales del sistema desarrollado. Se parte de una descripción de las características habituales de los sistemas de audio digital y de una breve introducción a las características utilizadas habitualmente para clasificar los sintetizadores. Posteriormente, se describe la estructura de bloques del sistema desarrollado, entrando en detalle en el diseño de cada uno de los subsistemas que lo forman. Para cada subsistema se justifican las decisiones de diseño tomadas, se presentan algunos resultados de las pruebas realizadas y se proponen posibles futuras líneas de trabajo.

En el capítulo 4 se describe el proceso de generación automática de código y su posterior despliegue en el SoC, a partir del modelo de bloques diseñado. Una vez generado el código de descripción *hardware*, sintetizado e implementado, se presenta el análisis de recursos utilizados por el sistema diseñado y se valoran las opciones de ampliación del sistema.

En el capítulo 5, se proponen futuras líneas de trabajo, de cara a mejorar la calidad del diseño planteado, ampliar sus funcionalidades y explotar al máximo los recursos disponibles en el SoC. Posteriormente, en el capítulo 6 se realiza un análisis final del trabajo realizado, exponiendo las conclusiones acerca del desarrollo de este tipo de sistemas en arquitecturas SoC y del uso de herramientas de modelado visual y generación automática de código para su desarrollo.

Finalmente, en el capítulo 7 se detallan las referencias bibliográficas citadas a lo largo del documento.

1.4 Convenciones utilizadas en el documento

A lo largo del documento aparecen figuras con modelos de bloques de Simulink®, en las que aparecerán los formatos de datos de las señales, representados en el formato propio de la herramienta. Para facilitar la interpretación de los mismos, a continuación se describe brevemente su formato.

- Las señales que utilizan formatos de coma flotante están etiquetadas con los tipos de datos habituales: `float`, `double`, `int16`, `uint16`, `boolean`, etc.
- Las señales que utilizan formatos de precisión finita siguen el siguiente formato:

  `<u|s>fix<n>[_En<f>]`

  donde:

  - `u`: indica que el tipo de datos no tiene signo
  - `s`: indica que el tipo de datos sí tiene signo
  - `n`: indica el número de bits totales del tipo de datos
  - `_En<f>`: esta parte es opcional e indica que el formato tiene parte fraccional y `f` representa el número de bits utilizados para representarla.

2. Herramientas y metodología de trabajo

2.1 Diseño de sistemas basados en arquitecturas SoC con lógica programable

Los denominados *System on Chip (*SoC*)* consisten en circuitos integrados que por sí solos forman un sistema de cómputo completo en un único chip, optimizado para determinadas tareas, en función de los diferentes componentes que lo forman. Generalmente, integran un procesador principal junto otros elementos habitualmente presentes en sistemas computacionales completos, como pueden ser memoria y caché integradas, periféricos e interfaces de comunicación externa, como: GPIOs, Ethernet, USB, UART, I2C, etc. Además, integran los mecanismos de comunicación necesarios para la integración eficiente de todos los componentes.

Este tipo de sistemas está en continua evolución y en la actualidad se pueden encontrar SoCs, como los de la familia Zynq 7000 de Xilinx® ([2]), que incorporan FPGAs, permitiendo así el uso de lógica programable en el sistema, en cooperación con el resto de subsistemas.

A la hora de plantear el diseño de un sistema basado en este tipo de dispositivos, se debe tener en cuenta la división interna del SoC y repartir las tareas entre los diferentes componentes internos. En el caso de la familia Zynq 7000 de Xilinx®, se dispone de dos procesadores ARM® Cortex™ A9, que permiten ejecutar código *software* secuencial que, a su vez, puede interactuar con la lógica programable desplegada en el chip FPGA, mediante la cual se puede hacer un uso extensivo del paralelismo que ofrecen este tipo de dispositivos. La comunicación entre ambos componentes se realiza mediante el protocolo AMBA 4 de ARM ([3]), que utiliza buses AXI4/AXI4-Lite/AXI4-Stream, con un mecanismo determinado de sincronización, configurable en la etapa previa a la generación del código a desplegar en el chip.

Un aspecto determinante a la hora de diseñar este tipo de sistemas son los formatos de datos que pueden ser utilizados en cada uno de los componentes y transportados por los diferentes buses. La parte *software* del sistema permite trabajar con formatos de precisión de coma flotante mientras que la parte *hardware* utiliza de forma nativa formatos de precisión finita. Además, deben tenerse en cuenta los tamaños de los diferentes tipos de buses definidos dentro de la especificación AMBA®.

Por otro lado, además de tener en cuenta los diferentes aspectos *hardware* a la hora de determinar los formatos de datos a utilizar, el hecho de trabajar con formatos de datos de precisión finita requiere de un meticuloso análisis de todos y cada uno de los parámetros y señales utilizadas durante el procesado de la señal. Se debe ser especialmente cuidadoso a la hora de elegir el formato de datos y el tamaño de éstos, puesto que de ello dependerá la calidad final de la señal de audio. Una mala elección puede derivar en diversos problemas, como por ejemplo: saturaciones de la señal, debidas a no cubrir todo el rango dinámico de la señal con el número de bits asignados a la parte entera del formato de datos; no disponer de la precisión necesaria para representar valores muy bajos o cambios suaves entre dos niveles de amplitud de la señal contiguos (provocando ruido de cuantificación), debido a la utilización de menos bits de los necesarios para la representación de la parte fraccional de la señal; o provocar un aumento en el coste de ejecución de las operaciones, bien de tiempo o de uso de recursos, si se utiliza un número excesivo de bits para representar la información ([4,5]).

Cabe destacar también que los dispositivos de lógica programable disponen diferentes tipos de recursos y de un número limitado de cada uno de ellos, entre los que cabe destacar:

- Bloque de lógica configurable (CLBs): Lookup tables (LUTs), flip-flops, sumadores en cascada
- Bloques de memoria RAM
- Bloques DSP

A la hora de realizar el diseño del sistema debe tenerse presente en todo momento la disponibilidad de cada uno de los diferentes tipos de recursos, adaptando las decisiones de diseño a los recursos disponibles.

Un último aspecto a tener en cuenta a la hora de diseñar este tipo de sistemas es la frecuencia de trabajo a la que puede trabajar el sistema. Los formatos de datos elegidos, el uso de los diferentes recursos disponibles y las técnicas de optimización utilizadas marcarán la frecuencia de trabajo máxima a la que puede trabajar el sistema.

Dada la complejidad que implica el diseño de este tipo de sistemas, se hace necesario el uso de un entorno de desarrollo que permita realizar el desarrollo teniendo en cuenta todos los aspectos descritos anteriormente, facilitando lo máximo posible la validación de los diseños durante el desarrollo y que proporcione las herramientas necesarias para todo el ciclo de desarrollo, desde la etapa inicial de modelado del sistema al despliegue final del código en el SoC.

2.2 Placa de desarrollo utilizada

Como placa de desarrollo para la evaluación de la metodología, se utiliza la placa Avnet® Zedboard™ Zynq-7000 ([6]), que dispone, entre otros elementos, de un SoC Zynq XC7Z020, sobre el que se despliega el sistema diseñado y un convertidor digital de audio (ADAU1761 de Analog Devices [7]) conectado a las conexiones de entrada y salida de audio (Figura 1).

Siguiendo la arquitectura SoC del chip Zynq XC7Z020 incluido en la placa de desarrollo, el diseño de cada uno de los bloques del sistema se divide en dos partes: una primera parte basada en procesado *software* (*Processing System - PS*), que se encarga de leer configuración del sistema, recoger los diferentes ajustes de los parámetros actualizados por el usuario de forma interactiva y de realizar los cálculos previos al sistema *hardware*; y una segunda parte *hardware* (*Programamable Logic - PL*), en la que se implementa el diseño lógico del sistema, que toma como entradas los parámetros pre-calculados en la parte *software* y que, además, se encarga la comunicación con el códec de audio con el que se intercambiará las señales de entrada y salida.

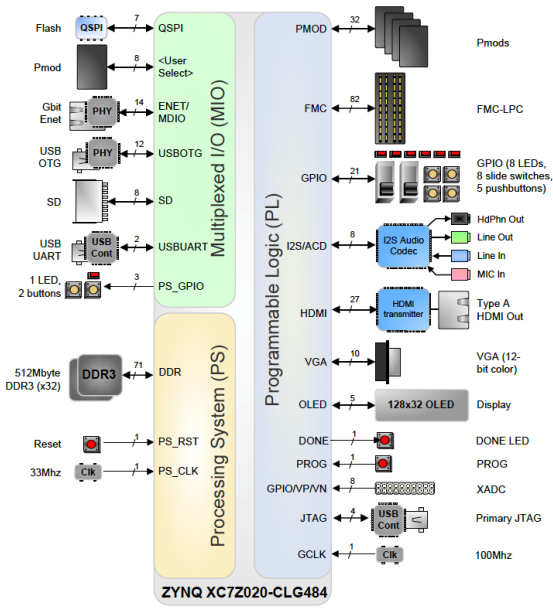


Figura 1. Diagrama de bloques interno del SoC Zynq XC7Z020 ([6])

2.3 Entorno de desarrollo

La metodología de trabajo se basa principalmente en el uso de la herramienta Simulink® de Mathworks® y sus *plug-ins* para el diseño de sistemas digitales mediante modelos visuales, generación automática de código y posterior despliegue sobre arquitecturas SoC, concretamente para arquitecturas Zynq® 7000. Estos complementos hacen uso a su vez del compilador C MinGW-w64 y la suite Vivado® HL WebPACK™, para las etapas de generación automática de código C y la síntesis e implementación del bitstream en la FPGA, respectivamente.

De los *plug-ins* de Simulink® utilizados, cabe destacar HDL Coder™ y Embedded Coder®, que son los encargados de llevar a cabo la transformación del modelo visual de Simulink® a un sistema funcional, desplegado en el SoC de la placa.

Entre las funcionalidades que proporciona HDL Coder™ cabe destacar las siguientes:

- Proporciona un conjunto de bloques de Simulink® para los que permite la generación automática de código de descripción de *hardware*.
- Realiza las validaciones pertinentes sobre el modelo para comprobar que los bloques utilizados y sus ajustes sean compatible con las restricciones impuestas por la posterior etapa de generación de código *hardware*.
- Facilita la instalación de diseños de referencia, encargados de realizar la conexión interna de los diferentes componentes del SoC (Figura 2) y el resto de elementos de la placa con los que éste interactúe. Incluye diseños de referencia predefinidos, como el utilizado en el presente trabajo, que posibilita la interacción con el códec de Audio.
- Conversión automática del modelo Simulink® a código VHDL o Verilog, en función de las preferencias del usuario, y generación del IP Core.
- Cuando es viable, permite aplicar optimizaciones de forma automática al modelo para mejorar el uso de recursos en la FPGA u optimizar la velocidad de ejecución.
- Realiza la síntesis, la generación del *bitstream* y la programación de la FPGA, apoyándose en el SDK de Xilinx® incluido en la suite de Vivado® HL WebPACK™.

Por otro lado, entre las funcionalidades proporcionadas por Embedded Coder®, cabe destacar:

- Permite la instalación inicial de un sistema operativo Linux en el SoC, que permitirá la comunicación con el dispositivo mediante red Ethernet, mediante conexión vía puerto de serie (protocolo UART) y mediante el protocolo JTAG.
- Generación del código *software* del modelo en lenguaje C, que se ejecutará sobre el sistema operativo previamente instalado y que permite la recepción de los ajustes de parámetros desde el modelo de Simulink® y la comunicación con la FPGA.

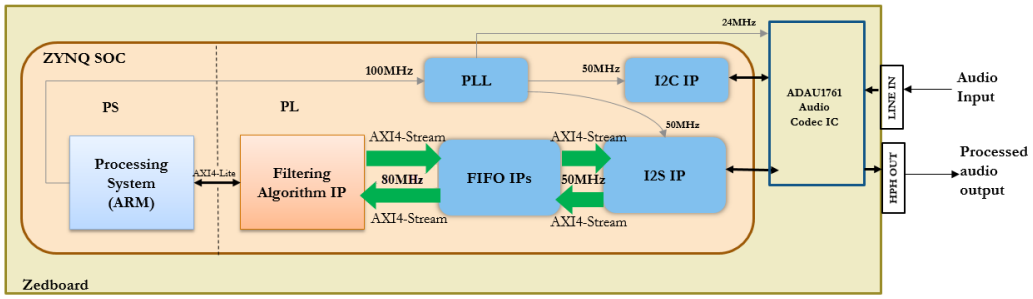


Figura 2. Visión global de los subsistemas de la solución, para el IP Core descrito en [8] *(Filtering Algorithm IP)*

Para completar la funcionalidad de los complementos HDL Coder™ y Embedded Coder®, de modo que el código se genere según las especificaciones propias de la arquitectura SoC de la familia Zynq 7000, se utilizan además los siguientes paquetes de soporte *hardware*:

- Embedded Coder® Support Package for Xilinx Zynq Platform
- HDL Coder™ Support Package for Xilinx Zynq Platform

Además de los complementos destinados a la generación automática de código y programación del dispositivo, también se han utilizado otras herramientas de Mathworks® de ayuda al desarrollo, como son: Stateflow®, el cual permite el diseño de máquinas de estado sintetizables a lógica programable, o Fixed-Point Designer™, que facilita la conversión de sistemas digitales de precisión en coma flotante a sistemas de precisión finita.

2.4 Metodología de trabajo

Una vez realizada la instalación de las herramientas *software* descritas en los apartados anteriores y realizada la configuración inicial de éstas, ya se está en disposición de empezar con el modelado de sistemas y su posterior generación de código y despliegue sobre la placa de desarrollo.

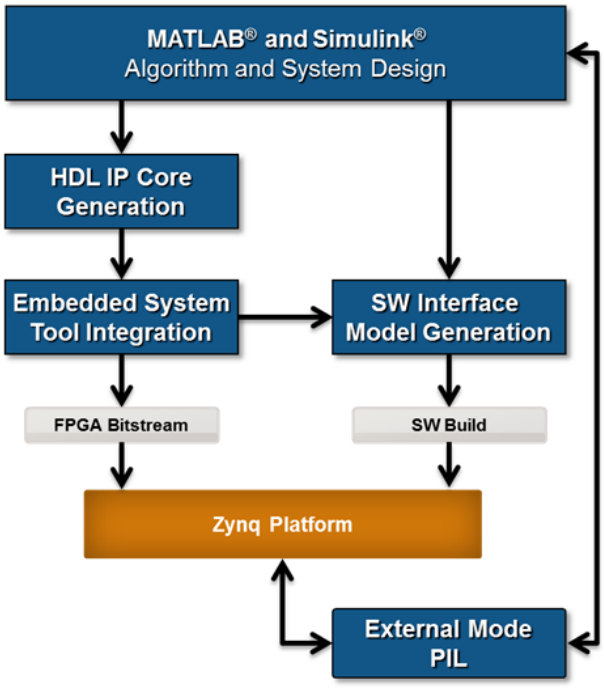


Figura 3. Esquema general de las etapas de desarrollo ([9])

En la Figura 3 se muestra de forma esquemática el ciclo de desarrollo habitual, cuyas etapas se pueden resumir como sigue:

1. Diseño y validación del modelo en Simulink® con formato de datos en coma flotante
2. Particionado del modelo separando los bloques que se vayan a ejecutar en el procesador *software* y los bloques de lógica programable que se vayan a implementar en la FPGA.
3. Conversión de los bloques de lógica programable a formato de coma fija y validación.
4. Generación del código *hardware* y programación del dispositivo mediante HDL Advisor (asistente para incluido en HDL Coder™) y Vivado® HL WebPack™

5. Generación del código *software* y programación del dispositivo mediante Embedded Coder®.
6. Validación del sistema utilizando la simulación externa de modelos de Simulink® (Figura 4).

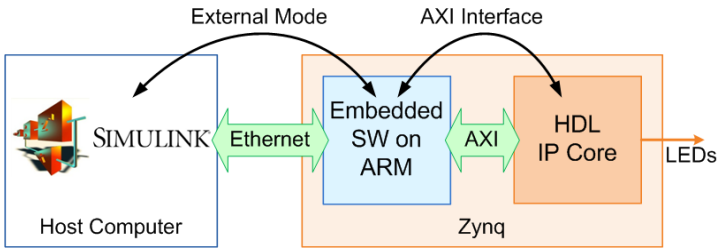


Figura 4. Esquema genérico de validación del sistema mediante simulación externa ([9])

El uso de esta metodología permite realizar todo el diseño y validación inicial del sistema desde el entorno Simulink®, utilizando para ello todos los mecanismos que la herramienta proporciona tanto para la etapa de diseño como para su validación, como son la utilización de scripts para la configuración y automatización de validaciones del modelo, la exportación de señales al *workspace* de MATLAB para su posterior análisis, el uso de monitores de señal en el tiempo o en frecuencia, el registro de señales y su posterior análisis mediante de Simulink Data Inspector o el analizador lógico de señales, etc.

Una vez ya validado el diseño en Simulink®, se procede a la generación del código *hardware* y *software* y a su despliegue en la placa. Este proceso es el más largo de todos, dependiendo del modelo puede llegar a tener una duración del orden de horas, algo habitual en el desarrollo de sistemas digitales sobre FPGAs, por lo que conviene haber realizado previamente todas las validaciones necesarias sobre el modelo en Simulink®, minimizando así el tiempo de desarrollo.

Finalmente, una vez desplegado el sistema completo en el SoC, ya se puede proceder a la validación final del sistema desplegado en la placa de desarrollo, comprobando en esta etapa final que la integración con el resto de componentes de la placa externos al SoC funciona correctamente, que los formatos de representación de datos elegidos para cada una de las señales durante el diseño son óptimos para su transporte sobre los buses dispositivo y que el uso de los relojes y señales de *enable* también se realiza de forma correcta.

3. Diseño del sintetizador

3.1 Características técnicas del sistema de audio

Todo sistema digital de audio trabaja a una determinada frecuencia de muestreo (f_s), que determina el número de muestras de la señal discreta que procesará por segundo y la frecuencia máxima a la que podrá trabajar el sistema de forma consistente, de modo que la señal digital interna se corresponda con la señal analógica generada en el proceso de conversión digital-analógica, tal y como determina el teorema de muestreo de Nyquist-Shannon ([10,11]).

En el presente proyecto se ha realizado el diseño partiendo de una frecuencia de muestreo de 48kHz. El convertidor analógico/digital incluido en la placa de desarrollo permite frecuencias de muestreo de hasta 96kHz, por lo que en un futuro sería posible realizar una revisión del diseño planteado para optimizarlo a dicha frecuencia de trabajo.

Otra de las características más importantes que caracterizan los sistemas de audio digital es el número de bits utilizado para representar la información de las muestras. El sistema diseñado trabaja con una profundidad de audio de 24 bits. El audio generado por los osciladores utiliza 24 bits para representar de forma interna la señal, concretamente se utiliza un formato de precisión finita [24,23] en complemento a 2, es decir, números con signo de 24 bits de los cuales se utiliza 1 bit para representar la parte entera y el signo, y 23 bits para representar la parte fraccional. Cabe destacar que este formato no se mantiene fijo a lo largo de la cadena de síntesis ya que, como se verá más adelante, en el caso del filtro se llegan a utilizar un mayor número de bits en algunos bloques, con el objetivo de mantener el rango dinámico y la precisión necesaria de la señal en etapas intermedias de cálculo. Finalmente, antes de enviar el flujo de audio al conversor digital se realiza una conversión final de los datos al formato [24,0] en complemento a 2.

La calidad de la señal final generada por el sistema viene determinada por las características del convertidor digital de audio incluido en la placa. En el caso de la placa de desarrollo utilizada, se incluye un convertidor ADAU1761, el cual ofrece una relación de señal ruido (SNR) de 98 dB con una profundidad de audio de 24 bits.

La placa de desarrollo utilizada dispone del siguiente conjunto de entradas y salidas conectadas al convertidor digital de audio: *Headphone out, Line out, Line in* y *Mic in*. De estas entradas y salidas, el sintetizador únicamente utiliza una señal de salida estéreo (*Headphones out*). Esta restricción viene impuesta por el diseño de referencia utilizado (*Audio System with AXI4 Stream Interface*), que viene incluido en HDL Coder™, que únicamente configura el conversor digital de audio para usar una entrada de audio externa estéreo (*Line in*) y una salida también estéreo (*Headphones out*). En un futuro, sería posible crear nuevos diseños de referencia personalizados, lo que permitiría utilizar las entradas y salidas del convertidor digital de audio al antojo del diseñador.

3.2 Características funcionales del sintetizador

Los sintetizadores de audio se pueden clasificar en función de varias de sus características, entre las más relevantes cabe destacar:

- Tipo de síntesis utilizada
- Polifonía
- Capacidad politímbrica

En la actualidad existen multitud de tipos de síntesis entre las que destacan por su uso más generalizado la síntesis sustractiva, la síntesis aditiva, la síntesis FM, la síntesis *wavetable*, la síntesis granular y los diferentes tipos de síntesis por modelado físico. El tipo de síntesis determina la forma en la que se genera la señal de audio y, por tanto, la estructura de bloques que se utilizará, tanto a la hora de determinar qué bloques forman parte de ella como a la hora de diseñar la ruta de la señal de audio y de las señales de control.

Para el presente trabajo se ha optado por un enfoque basado en la síntesis sustractiva que, de forma resumida, consiste en la generación de tonos con diferente contenido armónico mediante osciladores, que posteriormente es alterado mediante su filtrado para acentuar, atenuar o incluso eliminar completamente determinados armónicos.

La mayoría de sintetizadores sustractivos cuentan con una serie de bloques básicos que, en función de las decisiones tomadas en la etapa de diseño del sistema, ofrecen diferentes ajustes y posibilidades de integración entre ellos, con el fin de generar timbres con un carácter personalizado y dotarlos de variabilidad en el tiempo mediante el uso de envolventes y osciladores de baja frecuencia (LFO). Estos bloques suelen estar divididos en tres secciones: osciladores, filtro y amplificador.

Otro aspecto diferenciador entre distintos sintetizadores es su polifonía, la cual determina el número de voces con un mismo timbre que pueden sonar de forma simultánea, permitiendo la generación de acordes o cualquier otra forma de interpretación musical que requiera de varias notas que suenen de forma simultánea con un mismo timbre. Basándose en este aspecto, los sintetizadores se clasifican en monofónicos, parafónicos y polifónicos. Los sintetizadores monofónicos permiten la reproducción de una única nota musical de forma simultánea, con todos los recursos de la cadena de síntesis dedicados a ella. Por otro lado, los instrumentos polifónicos y parafónicos, permiten la reproducción de múltiples notas con un mismo timbre de forma simultánea, aunque existe una diferencia sustancial entre estos dos tipos: la utilización de los recursos por cada voz. Mientras que los sintetizadores polifónicos disponen de todos los elementos de la cadena de síntesis por cada voz, en los sintetizadores parafónicos únicamente los osciladores son exclusivos de cada voz, compartiendo el resto de elementos de la cadena de síntesis entre todas las voces, minimizando así el uso de recursos, a costa de una pérdida de riqueza tímbrica y de dinámica del sonido en las diferentes voces.

Aunque *a priori* puedan parecer los más limitados, los sistemas monofónicos tienen un gran éxito entre los usuarios, puesto que su diseño suele estar enfocado a un uso determinado, lo que les dota de un sonido bastante personalizado. En algunos casos, los fabricantes les añaden osciladores adicionales o permiten configurar los existentes para convertirlos en sistemas parafónicos. Finalmente, los polifónicos también son sistemas de gran éxito entre los usuarios, puesto que son los más versátiles y en algunos casos porque son los auténticos buques insignia de las marcas, con arquitecturas de síntesis realmente complejas y con unos costes realmente elevados. De las tres opciones, la menos extendida es la versión parafónica.

Otro aspecto diferenciador de los sintetizadores es su capacidad politímbrica, es decir, la capacidad para generar múltiples timbres de forma simultánea por diferentes salidas de audio. De esta forma, es posible generar varios timbres totalmente diferentes de forma simultánea, con tantas voces cada uno como se dispongan, con un único dispositivo. Esta capacidad permite que el usuario pueda interpretar de forma simultánea varias secuencias con diferentes timbres de forma simultánea, bien sea de forma manual (mediante la división del teclado por timbres) o

mediante sistemas externos de secuenciación. Esta característica resulta de gran interés para los usuarios, especialmente en los sintetizadores cuyo precio es elevado, puesto que con una única unidad se pueden llegar a emular el comportamiento de tantas unidades independientes como número de timbres simultáneos puedan generarse.

Tanto la polifonía como la capacidad politímbrica de los sintetizadores están habitualmente limitadas por la capacidad de procesado del sistema. En el presente trabajo se ha planteado un diseño monofónico, pero como se describirá en el apartado *Análisis de recursos utilizados por el sistema diseñado*, el diseño realizado podría ser ampliado para convertirse en un sistema polifónico y politímbrico.

3.3 Estructura de bloques del sintetizador

Una vez determinado el tipo de síntesis en la que se basará el diseño del sintetizador, el siguiente paso natural es el diseño de la cadena de síntesis por voz, es decir, el planteamiento de bloques y rutas de las señales de audio y de control para una única voz, como si de un diseño monofónico se tratara. A partir de este diseño, en función de las especificaciones funcionales, puede abordarse su diseño polifónico/parafónico, en función de los recursos disponibles.

En la Figura 5 se presenta la cadena de síntesis planteada para cada voz. Como se puede observar, se trata de una cadena de síntesis básica, cuyo diseño puede ser ampliado en futuras líneas de trabajo.

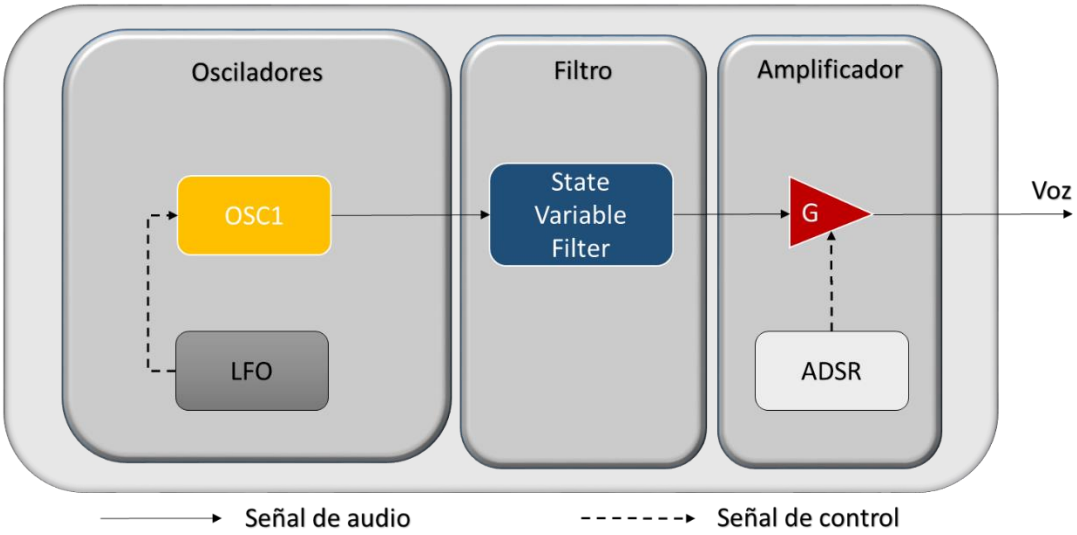


Figura 5. Cadena de síntesis planteada

A lo largo de los siguientes apartados de este capítulo, se irán describiendo en detalle los aspectos de diseño de cada uno de los bloques presentados en la Figura 5.

3.4 Los osciladores

En un sintetizador digital, los osciladores son los encargados de generar una señal discreta, periódica, a una determinada frecuencia fundamental y con unos determinados armónicos. Esta señal posteriormente irá pasando por las diferentes etapas de la cadena de síntesis, sufriendo diferentes alteraciones en cada una de ellas, con el fin de generar un timbre personalizado y un determinado comportamiento temporal.

En el diseño planteado, se han incluido diferentes tipos de osciladores, que generan diferentes formas de onda y, por tanto, diferente contenido armónico. Los diferentes tipos de osciladores implementados generan formas de onda de tipo senoidal, diente de sierra, triangular y de pulso variable (Figura 6).

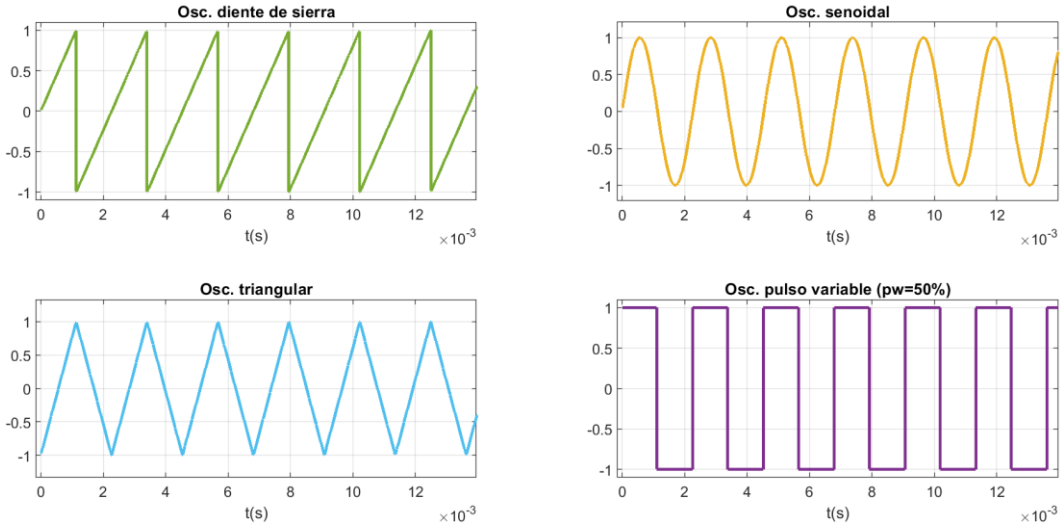


Figura 6. Osciladores ajustados a 440Hz

Tal y como se ha mencionado anteriormente, los osciladores permiten generar tonos a diferentes frecuencias fundamentales. Inicialmente, el rango de frecuencias de oscilación viene dado por las frecuencias asociadas a las diferentes notas de un piano de 88 teclas, de modo que las frecuencias de los tonos generados irán desde los 27.5 Hz correspondientes a la nota A0 hasta los 4186 Hz de la nota C8. Sin embargo, como tal y como se ha descrito anteriormente, en el diseño final del sistema la frecuencia de oscilación de los osciladores puede modularse mediante un oscilador de baja frecuencia (LFO). Como se describirá más adelante en la sección *Modulación de parámetros*, esto implica que el rango de frecuencias a cubrir por el oscilador se extiende hasta la mitad de la frecuencia mínima (una octava por debajo) y hasta el doble de la frecuencia máxima (una octava por arriba) indicadas anteriormente, es decir, desde los 13.75 Hz hasta los 8372 Hz.

Los sintetizadores digitales suelen utilizar la técnica de Síntesis Digital Directa (*Direct Digital Synhtesis – DDS*) para generar las señales de los osciladores. Existen diferentes técnicas para generar señales mediante este método, de entre las que cabe destacar principalmente dos:

- La generación de la señal mediante modelos matemáticos
- La generación de la señal mediante tablas

En el diseño planteado, a excepción de la forma de onda senoidal, la generación de las señales proporcionadas por los osciladores tiene un coste computacional relativamente bajo, puesto que se trata de funciones periódicas relativamente simples que pueden generarse con operaciones como son la suma, multiplicación o la comparación de valores, por lo que se ha optado por este enfoque para su diseño. En el caso de la señal senoidal, al tener un coste computacional relativamente alto se ha optado por el uso de técnicas alternativas para su generación, como es la síntesis directa mediante tablas, que fue introducida en [12], cuyo coste computacional es muy inferior al del cómputo de la señal, aunque requiere de tablas en memoria de sólo lectura para almacenar valores de amplitud precalculados. Cabe destacar en

este punto que la FPGA incluida en el chip Zynq XC7Z020, utilizado en el presente proyecto, dispone de 53200 *Lookup Tables (*LUTs*)*, bloques sobre los que se implementa esta estrategia.

Como criterio de diseño, se establece que las señales generadas por todos los osciladores tendrán una amplitud pico a pico de 2, tomando valores de amplitud entre -1 y 1 (Figura 6), que serán representados mediante números en coma fija representados en complemento a 2 con una precisión [24, 23], es decir, números de 24 bits de los que se utiliza 1 bit para representar la parte entera y el signo, y 23 para la parte fraccional. Hay que tener en cuenta que con dicho formato precisión no es posible alcanzar el rango completo deseado, puesto que dicha representación sólo permite representar valores en el rango [-1..0.9999998807907104]. Sin embargo, se acepta el valor 0.9999998807907104 como una buena aproximación al valor 1, puesto que la alternativa pasa por utilizar 1 bit más para la parte entera y una serie de operaciones extra para controlar el rango de la señal, lo que supondría un aumento en el uso de recursos computacionales y de área, sin que derive de ello un beneficio que compense el coste extra. Al utilizar 24 bits para cuantificar la amplitud de la señal (*W*), se puede llegar a lograr una relación señal/ruido (SNR) de aproximadamente 146 dB, según (1).

$$SNR \approx 6.02W + 1.76\ dB \tag{1}$$

Como cualquier otro sistema digital, los sintetizadores digitales de audio trabajan con señales discretas, es decir, las señales únicamente tomarán valor en determinados instantes, quedando completamente representadas por una secuencia de valores que representan su amplitud en un instante dado (n).

Es importante tener en cuenta que la frecuencia de muestreo se mantiene constante durante todo el proceso de generación de la señal, siendo un parámetro que se ajusta al inicio del sistema y permanece inalterado durante el funcionamiento del mismo, puesto la mayoría de cálculos que se realizan en los diferentes bloques dependen de su valor.

3.4.1 Oscilador diente de sierra

El oscilador de formas de onda de diente de sierra es el más simple de todos los utilizados en el presente trabajo. Tal y como se puede observar en la Figura 6, este oscilador se puede implementar como una función rampa, acumulando valores en los sucesivos instantes de muestreo hasta llegar al valor máximo de amplitud en el instante correspondiente, instante que vendrá determinado por la frecuencia a la que se ajuste el oscilador.

Tal y como se irá presentando a lo largo de los siguientes apartados, este oscilador es la base sobre la que se apoyan el resto de los osciladores del sistema, por lo que se su diseño debe ser lo más preciso posible. El formato de datos utilizado en cada uno de los bloques que lo forman determinará la resolución y rango de frecuencias generables por los diferentes osciladores.

Como ya se ha descrito anteriormente, el rango de frecuencias que debe cubrir el oscilador va desde los 13.75 Hz hasta los 8272 Hz. En cuanto a la resolución, si se tratara de un piano tradicional vendría marcada por la distancia entre semitonos, es decir, una doceava de octava, que vendría dada por (2). Teniendo en cuenta que la distribución de las octavas se realiza siguiendo la escala logarítmica, el caso más restrictivo se daría para la frecuencia más baja a reproducir, es decir, 13.75 Hz. Aplicando (2) para dicha frecuencia se obtiene una resolución de 0.8176 Hz.

$$dist_{st} = f_o \cdot 2^{\frac{1}{12}} - f_o \ (Hz) \tag{2}$$

Para la implementación del oscilador se ha utilizado un acumulador de módulo 2^M, donde M es el número de bits utilizados en la parte entera del formato de precisión numérica utilizado. El formato de datos del acumulador se ajusta para que sea igual al utilizado para representar las señales del oscilador, es decir, [24,23] en complemento a 2, por lo que el valor de M en este caso es 1. En cada instante de muestreo se realizará un incremento del valor del acumulador, este incremento es conocido como paso (*step*), hasta llegar al valor máximo permitido por el formato de datos utilizado en el acumulador, instante que coincidirá con el periodo de oscilación de la señal a generar.

Asumiendo que la frecuencia de muestreo se mantiene constante y teniendo en cuenta que el valor máximo que podrá alcanzar el acumulador es de 2^M (obviando que el rango de trabajo está entre -1 y 1), en un periodo de oscilación (T_o) se cumplirá la igualdad (3).

$$T_o = \frac{2^M * T_s}{step} \tag{3}$$

A partir de (3) se deduce automáticamente (4), que determina el paso del acumulador. Debe tenerse en cuenta que el resultado de la operación se representará en un formato [24,23] en complemento a 2, por lo que se realizará el redondeo del resultado al valor representable más cercano.

$$step \ = \frac{2^M * f_o}{f_s} \tag{4}$$

$$f_o = \frac{f_s * step}{2^M} \tag{5}$$

Estos primeros cálculos del paso en función de la frecuencia de oscilación se han implementado sobre la parte de procesado *software*, dejando para la parte de lógica programable la implementación del acumulador.

Para simplificar la deducción del rango de frecuencias representable y de la precisión del oscilador, en lugar de trabajar con formatos decimales, se supondrá que el formato de datos es [24,0] sin signo, de modo que el valor máximo alcanzable por el oscilador sería de 2^{24} y el paso de 1. Teniendo en cuenta que la resolución es la diferencia entre un paso y el siguiente, partiendo de (5) se puede deducir la resolución mediante (6):

$$\Delta f_{o\_min} = f_o(step + 1) - f_o(step) = \frac{f_s}{2^M}(step + 1 - step) \tag{6}$$

$$\Delta f_{o\_min} = \frac{f_s}{2^M} \tag{7}$$

El valor mínimo de frecuencia representable por el oscilador, vendrá dado por la propia resolución, es decir, cuando se requiera de todos los bits disponibles para obtener el incremento entre dos muestras contiguas. Sustituyendo valores en (7), se obtiene un valor de resolución (Δf_{o_min}) de 0.002861022949219 Hz, óptimo para los requisitos del diseño.

En cuanto al valor máximo de frecuencia representable por el oscilador, este vendrá determinado por el teorema de muestreo de Nyquist-Shanon ([10,11]), es decir, por (8):

$$f_{o\_max} = \frac{f_s}{2} \tag{8}$$

En la Figura 7 se presenta el diagrama de bloques utilizado para la implementación del acumulador en Simulink®, cuya salida (`acc_value`) es el valor de amplitud de la señal del oscilador con forma de onda de diente de sierra en un instante dado.

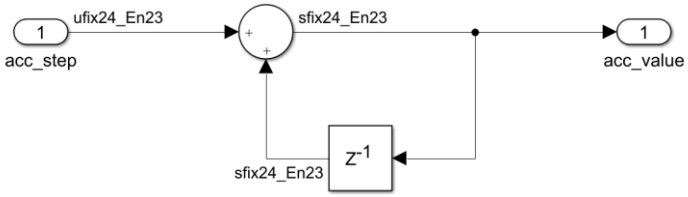
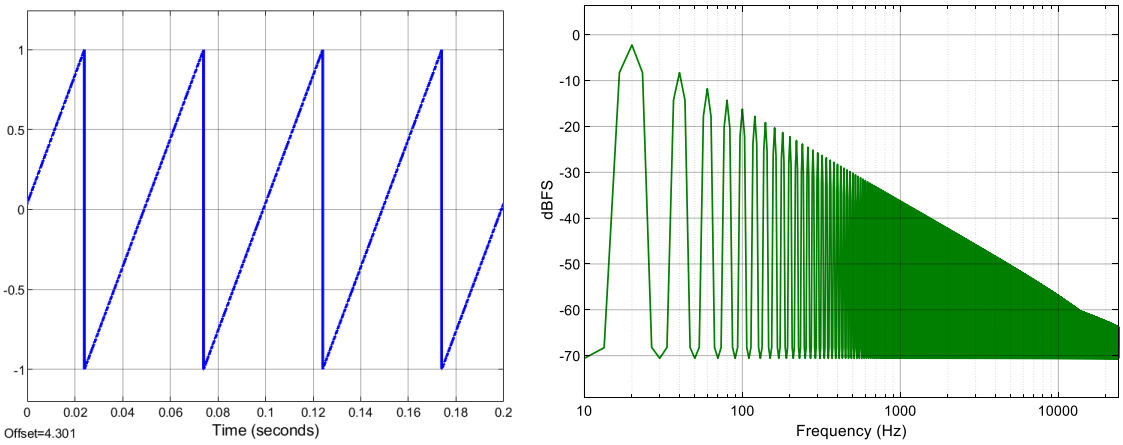


Figura 7. Acumulador módulo 2^M

En la Figura 8 y en la Figura 9 se presentan las respuestas temporales y frecuenciales del oscilador de diente de sierra para una frecuencia de oscilación de 20 Hz y de 8372 Hz, respectivamente. Como se puede observar en las representaciones temporales, a baja frecuencia la forma de onda es prácticamente idéntica a la que se obtendría de un oscilador analógico, sin embargo, a altas frecuencias se percibe el efecto de típico de los osciladores digitales, en el que el acumulador no siempre alcanza la máxima amplitud de forma exacta, sino que la sobrepasa y dicho exceso se suma al valor mínimo del acumulador, dando como resultado un valor de amplitud inferior al previsto inicialmente. Para reducir este efecto, debería utilizarse una frecuencia de muestreo superior, de modo que se utilizara un mayor número de muestras por periodo para representar la señal, aunque el efecto nunca desaparecerá completamente. Por otro lado, debe tenerse en cuenta que las representaciones temporales presentadas son capturas del sistema digital en modo *sample & hold*, por lo que una vez la señal pasa por el filtro paso bajo de reconstrucción, incluido en el convertidor digital de audio, la señal analógica generada no presentará esta forma escalonada que se percibe en las capturas, especialmente a medida que se aumenta la frecuencia de la señal.



(a) *(b)*

Figura 8. Representación temporal (a) y frecuencial (b) de la señal diente de sierra de 20 Hz

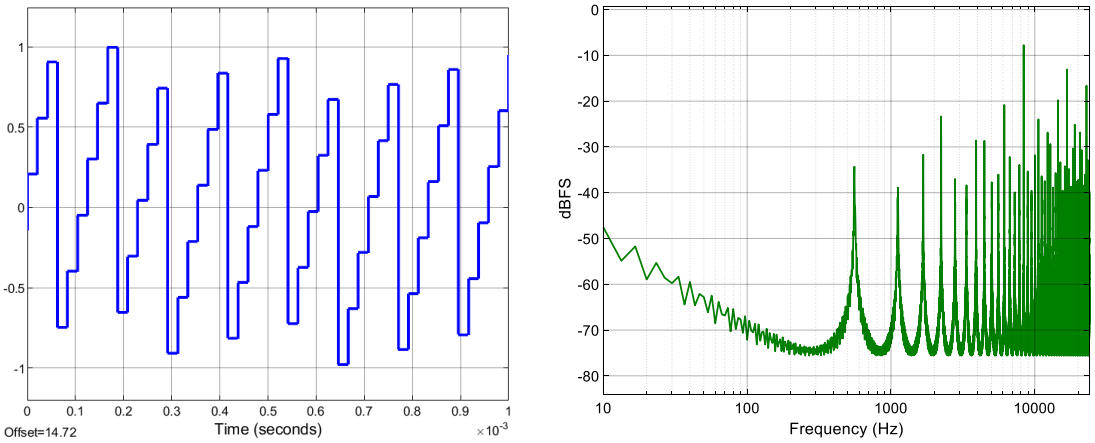


Figura 9. Representación temporal (a) y frecuencial (b) de la señal diente de sierra de 8372 Hz

Tal y como se observa en las representaciones espectrales de la señal, aparecen componentes frecuenciales que no son propias de este tipo de señales. Esto es debido al efecto del aliasing generado por los armónicos de la señal que se generan de forma ilimitada a causa de las discontinuidades de la señal. Para evitar este efecto pueden utilizarse osciladores de banda limitada, que evitan la aparición de dichos armónicos por encima de la frecuencia de muestreo, evitando de ese modo la aparición del aliasing. Varios autores han realizado numerosos estudios relacionados con técnicas para la mejora de la calidad de la señal generada por los osciladores digitales, en [13] y en el capítulo 4 de [14] se describen varias de ellas. La aplicación de estas técnicas se hace indispensable si el objetivo es el de obtener un oscilador digital que emule las características de los osciladores analógicos, sin embargo, este no es el objetivo inicial de este trabajo, por lo que su aplicación se deja para futuras líneas de trabajo.

3.4.2 Oscilador senoidal

La generación de señales senoidales mediante modelado matemático tiene un elevado coste computacional, es por ello que habitualmente se utilizan técnicas de síntesis digital por tablas para su generación. Estas tablas se conocen comúnmente como *Lookup Tables* (LUT). Esta técnica, descrita en detalle en [15] y [16], consiste en almacenar previamente un periodo de la señal a generar en una tabla, muestreada en un número determinado de instantes, y, posteriormente, recorrer las entradas de la tabla variando el incremento entre una posición y la siguiente a leer, para generar señales de diferentes frecuencias.

Esta técnica se divide en dos etapas (Figura 10): una primera consistente en la generación de la fase de la señal, denominada comúnmente *acumulador de fase,* y otra posterior, denominada comúnmente denominada *convertidor de fase a amplitud,* en la que los valores de fase generados en la primera etapa se utilizan para recorrer las entradas de la tabla, en las que se almacenan los valores de amplitud de la señal objetivo.

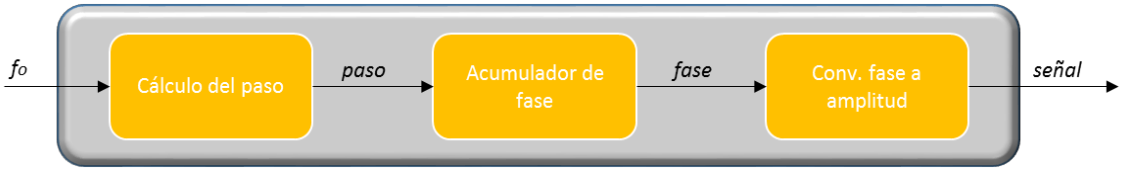


Figura 10. Etapas Síntesis Digital Directa por tablas

La señal de fase, generada por el acumulador de fase, consiste en una señal de tipo rampa con un periodo de oscilación igual que el de la señal que se quiere generar como salida tras cada recorrido completo de la LUT. Esta señal es exactamente igual a la generada en el oscilador de diente de sierra, por lo que se puede reutilizar, con el fin de optimizar recursos. En la Figura 11 se pueden observar las dos señales generadas en las diferentes etapas.

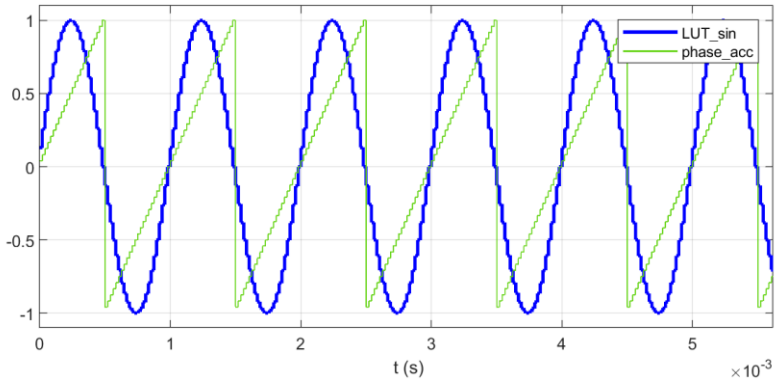


Figura 11. Señal del acumulador de fase (phase_acc) y señal senoidal generada a la salida de la LUT (LUT_sin)

Tal y como se ha descrito en el apartado anterior, a partir de la frecuencia de oscilación deseada se calcula el tamaño del paso del *acumulador de fase*. Para frecuencias bajas, este valor será más bajo, lo que implicará que se recorran las entradas de la LUT en un mayor número de instantes de muestreo. A medida que se aumente la frecuencia de oscilación, el tamaño del paso del acumulador de fase también aumentará, por lo que el recorrido de la LUT se realizará en menores instantes de muestreo. En la Figura 12 se puede observar la señal obtenida al recorrer una misma LUT con pasos (P) diferentes, cuanto mayor sea el paso, mayor será la frecuencia de la señal generada.

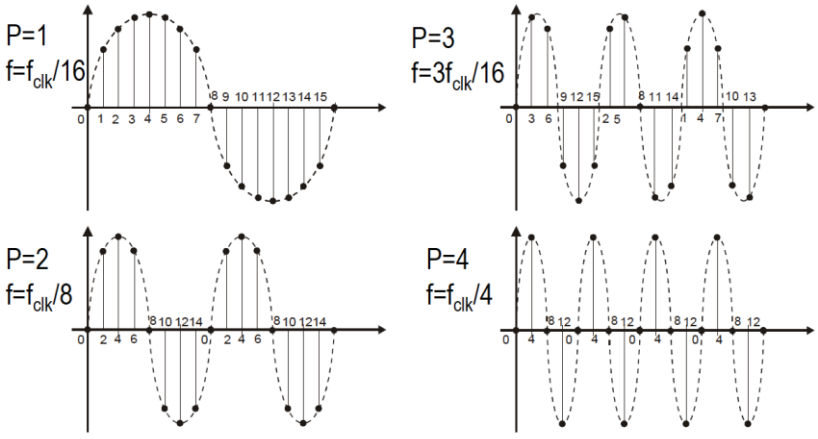


Figura 12. Ejemplo de acumulador de fase de 4 bits ([17]*)*

El tamaño de la LUT es un aspecto que debe considerarse durante el diseño del oscilador. Debe tenerse en cuenta que la LUT ocupará una determinada cantidad de memoria, que vendrá determinada por (9), donde L se corresponde con el número de bits utilizados para indexar las posiciones de la tabla y W con el número de bits utilizados para el almacenamiento de las muestras de la señal de salida en cada entrada de la tabla.

$$LUT_{mem} = 2^L \cdot W\ bits \quad (9)$$

Como se puede deducir de (9), cuanto mayor sea L más memoria se requerirá para almacenar la tabla. Si se quisieran utilizar todos los bits de la señal rampa para indexar la LUT (L=M), se requeriría una enorme cantidad de memoria, lo cual resulta totalmente inviable en un sistema como el del presente trabajo. Este es un problema habitual en la utilización de este tipo de técnicas en sistemas con recursos limitados, por lo que ya se han estudiado diferentes técnicas para lograr una reducción de la memoria requerida. Al utilizar estas técnicas, se debe buscar un equilibrio que reduzca lo máximo posible la cantidad de memoria requerida a la vez que se garantiza una calidad óptima de la señal. Un primer planteamiento para reducir el tamaño de la tabla consiste en utilizar un número de bits L < M, es decir, realizar un truncamiento de la fase. De este modo se almacenarán un menor número de muestras en la LUT. Esta técnica provocará que, para determinadas frecuencias, se mantenga el mismo valor de fase en varios instantes de muestreo consecutivos, lo que derivará en la aparición de componentes frecuenciales adicionales en la señal de salida, denominadas espurios (*spurious*). Este efecto se hará más latente cuanto menor sea la frecuencia de oscilación y, por tanto, el paso. La forma de cuantificar este efecto viene dada por la relación SFDR (*Spurious Free Dynamic Range*), según (10).

$$SFDR \approx\ 6.02L - 3.92\ dB \quad (10)$$

En [16], se describen algunas técnicas para la reducción de este efecto no deseado, que pueden ser implementadas en futuras líneas de trabajo, si se desea disponer de osciladores senoidales con un rango dinámico libre de espurios superior.

En la Figura 13 se presenta el diagrama de bloques utilizado para la implementación de la etapa de conversión de fase a amplitud del oscilador.

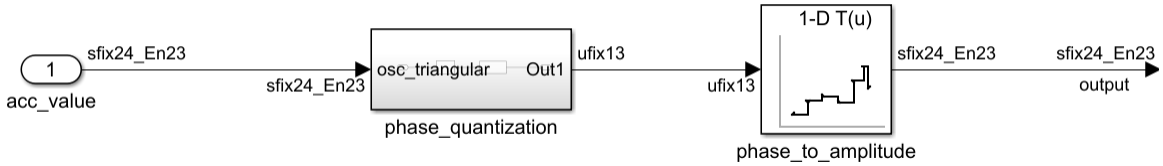
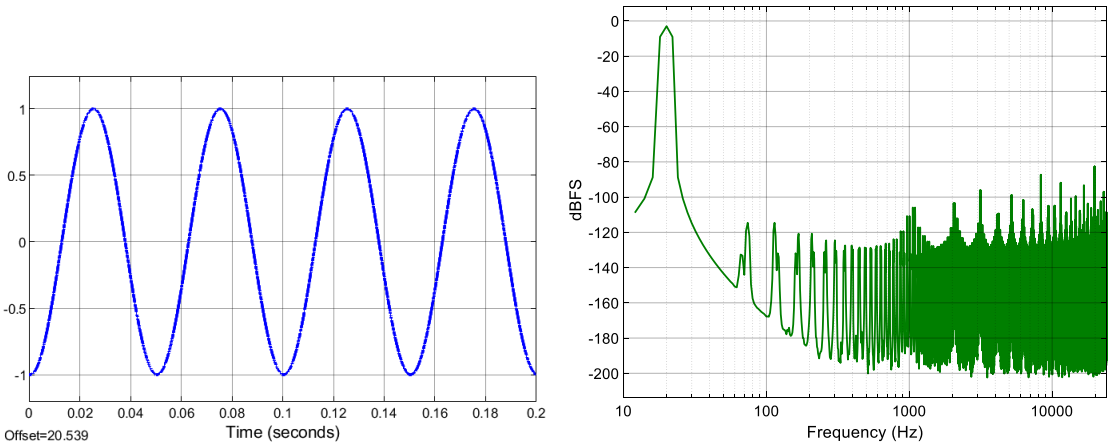


Figura 13. Diagrama de bloques de la conversión de fase a amplitud

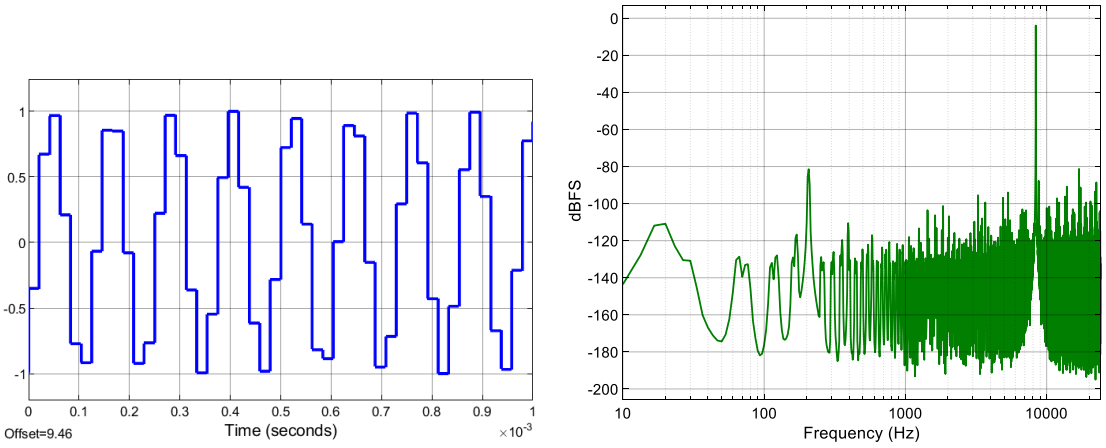
En la Figura 14 y en la Figura 15 se presentan las respuestas temporales y frecuenciales del oscilador senoidal para una frecuencia de oscilación de 20 Hz y de 8372 Hz, respectivamente. Nuevamente, en la representación temporal de señales de alta frecuencia se percibe el efecto de los osciladores digitales, a medida que la frecuencia de oscilación se aproxima a la de muestreo.

En las representaciones frecuenciales se puede observar el efecto de los espurios descritos anteriormente, quedando una zona libre de espurios de alrededor de 75 dB, tal y como corresponde según (10) al utilizar 13 bits para la cuantificación de la fase.



(a) (b)

Figura 14. Representación temporal (a) y frecuencial (b) de la señal senoidal de 20 Hz



(a) (b)

Figura 15. Representación temporal (a) y frecuencial (b) de la señal senoidal de 8372 Hz

3.4.3 Oscilador triangular

El oscilador de forma de onda triangular, tal como su nombre indica, genera una onda cuyo periodo describe una forma triangular empezando y acabando por su nivel más bajo y tomando su nivel más alto a mitad de periodo. Teniendo en cuenta el criterio de diseño establecido por el que todos los osciladores se implementan de forma bipolar, la señal tomará valores de amplitud entre -1 y 1.

Nuevamente, la generación de la señal se basa en la señal rampa generada para el oscilador de diente de sierra. En este caso se utilizarán operaciones matemáticas para convertir la señal con forma de diente de sierra en una señal triangular. El primer paso consiste en convertir los valores negativos de la señal rampa en positivos, para lo que se utiliza la función de valor absoluto, con lo que se obtendrá una señal triangular de amplitud pico a pico de 1. Para conseguir la amplitud pico a pico de 2, establecida como criterio de diseño de los osciladores, se multiplica el valor obtenido por 2. Finalmente, para obtener la señal bipolar entre -1 y 1, se resta 1 al valor obtenido. En la Figura 16 se presenta el diagrama de bloques utilizado para dicha transformación.

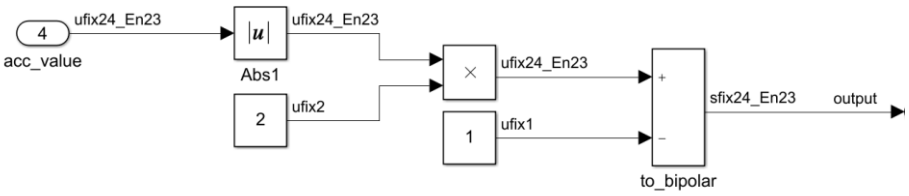
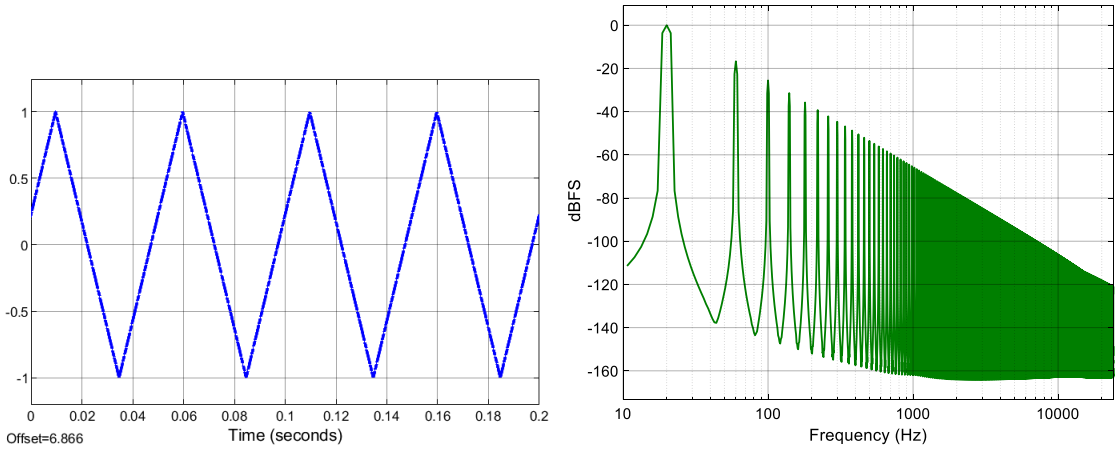


Figura 16. Diagrama de bloques para la generación de la señal triangular

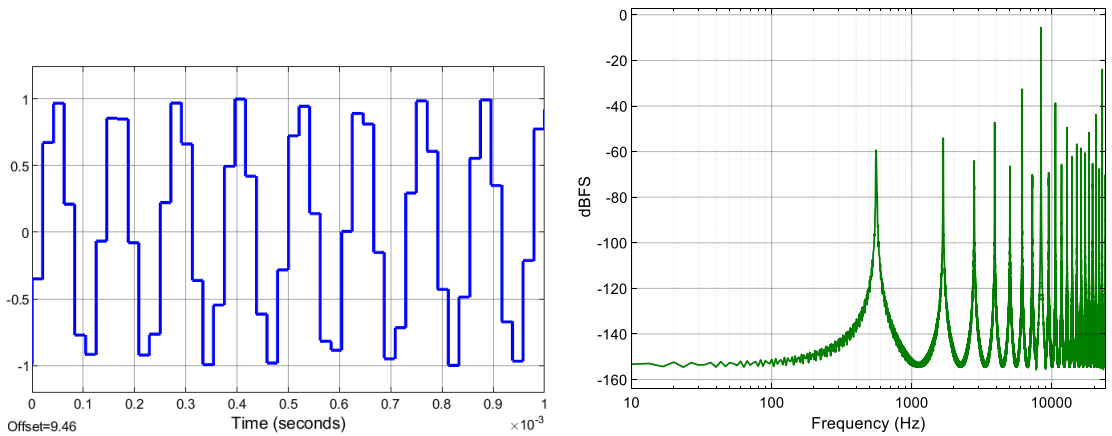
En la Figura 17 y en la Figura 18 se presentan las respuestas temporales y frecuenciales del oscilador senoidal para una frecuencia de oscilación de 20 Hz y de 8372 Hz, respectivamente. Una vez más, en la representación temporal de señales de alta frecuencia se percibe el efecto típico de los osciladores digitales al aproximarse a la frecuencia de muestreo.

En la representación frecuencial se puede apreciar cómo el efecto del aliasing introduce nuevas componentes de alta frecuencia. Tal y como ya se ha descrito para el oscilador de diente de sierra, este efecto puede evitarse si se diseñan osciladores de banda limitada.



(a) *(b)*

Figura 17. Representación temporal (a) y frecuencial (b) de la señal triangular de 20 Hz



(a) *(b)*

Figura 18. Representación temporal (a) y frecuencial (b) de la señal triangular de 8372 Hz

Si se analiza el espectro presentado en la Figura 19, que corresponde a una frecuencia de oscilación de 440 Hz, se pueden apreciar los armónicos impares característicos de este tipo de señales. Además, también se percibe claramente el efecto provocado por el aliasing en altas frecuencias, descrito ya anteriormente.

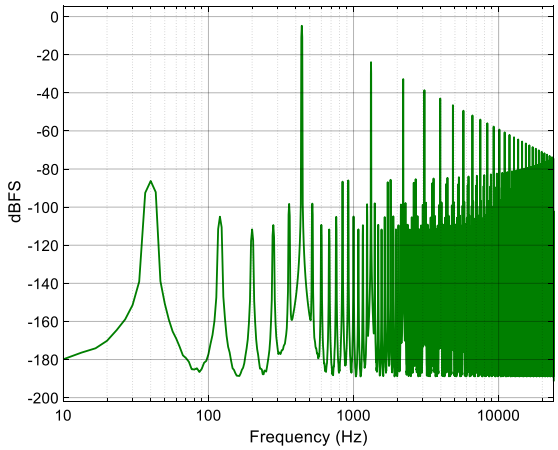


Figura 19. Representación frecuencial de señal triangular de 440 Hz

3.4.4 Oscilador de pulso variable

En este tipo de osciladores, la señal de salida puede tomar únicamente dos valores: pulso activo o pulso inactivo. Teniendo en cuenta el criterio de diseño establecido por el que todos los osciladores se implementan de forma bipolar, la señal tomará valores de amplitud entre -1 y 1. De todos los osciladores implementados, éste es el único que no utiliza la señal rampa generada por el oscilador de diente de sierra.

Al tratarse de un oscilador de pulso variable, se debe implementar un mecanismo para el control del tamaño del pulso. Para ello, en primer lugar, se calcula el número de muestras de cada periodo de oscilación (`smp_osc`) y de la parte del periodo en el que la señal se encuentra activa (`smp_pw`), a partir de los ajustes de frecuencia de oscilación (f_o) y anchura del pulso (`pulse_width`) realizados por el usuario. Estos cálculos se realizan en la parte de procesado *software* del SoC, utilizando el diagrama de bloques presentado en la Figura 20.

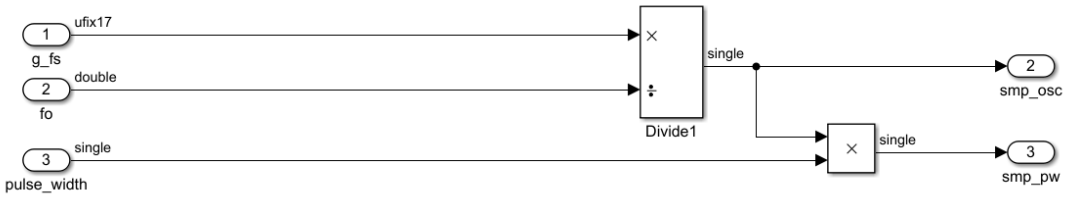


Figura 20. Diagrama de bloques para el control del tamaño del pulso

Ya en la parte de procesado lógico, se implementa un *contador de fase* (Figura 21), que no se debe confundir el *acumulador de fase* descrito anteriormente y que únicamente realizara el conteo de instantes de muestreo transcurridos. Este valor de fase se compara con el valor calculado del periodo de oscilación (`smp_osc`) y se obtiene un contador de fase relativo al periodo de la señal (phase_counter).

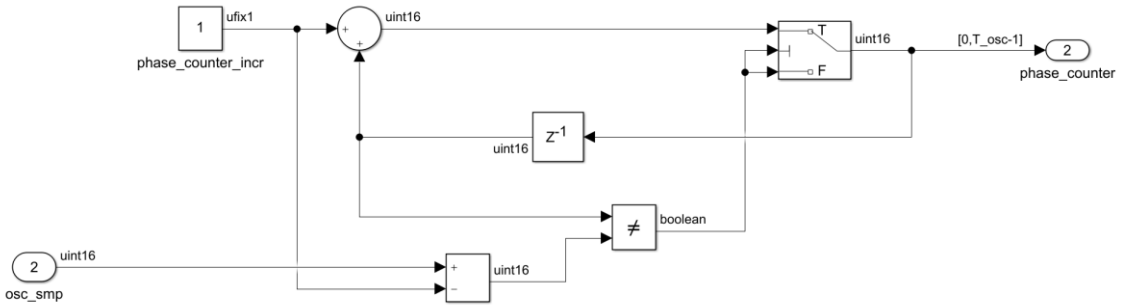


Figura 21. Diagrama de bloques para la generación del contador de fase (phase_counter)

Posteriormente, en la etapa de *conversión de fase a amplitud* se utiliza el contador de fase (`phase_counter`) y el número de muestras en las que el pulso está activo (`smp_pw`) y se determina si el pulso de la señal debe estar activo o inactivo en ese instante, tal y como se muestra en la Figura 22.

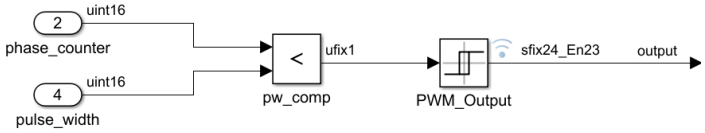


Figura 22. Diagrama de bloques para la generación de la señal de pulso variable

En la Figura 23 y en la Figura 24 se presenta el resultado obtenido para un ajuste del oscilador de pulso variable con un ancho de pulso del 50% y una frecuencia de oscilación de 20 Hz y 8372 Hz, respectivamente.

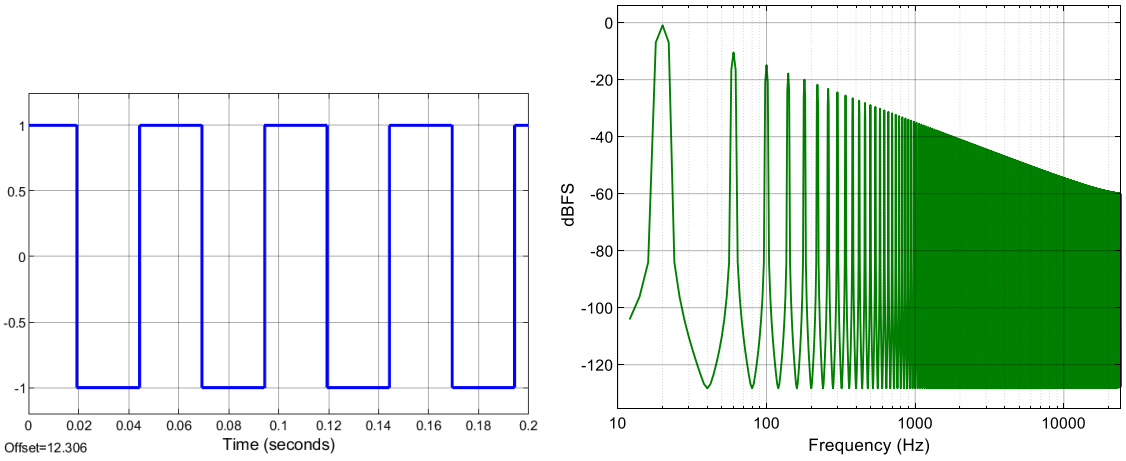


Figura 23. Representación temporal (a) y frecuencial (b) de la señal de pulso variable (pw = 50%) de 20 Hz

En este caso, la forma señal en el dominio del tiempo no se ve afectada por el efecto de los osciladores digitales al aproximarse a la frecuencia de muestreo, puesto que únicamente puede tener dos valores que se establecen de forma manual (-1 y 1), pero sí se ve afectado su periodo y, por tanto, su frecuencia de oscilación. En este ocasión, este efecto es debido a que al dividir la frecuencia de muestreo entre la frecuencia de oscilación, el número de periodos sólo será entero en las frecuencias de oscilación que sean múltiplo de la de muestreo. En el resto de casos, habrá que hacer un redondeo para determinar en qué punto pasa la señal de su nivel mínimo al

máximo, provocando que la frecuencia de oscilación no coincida exactamente con la establecida. Este mismo efecto se produce también a la hora de dividir el periodo entre la zona de nivel alto y de nivel bajo, según se haya establecido el ancho de pulso. Nuevamente, una forma de minimizar este efecto podría ser el aumento de la frecuencia de muestreo.

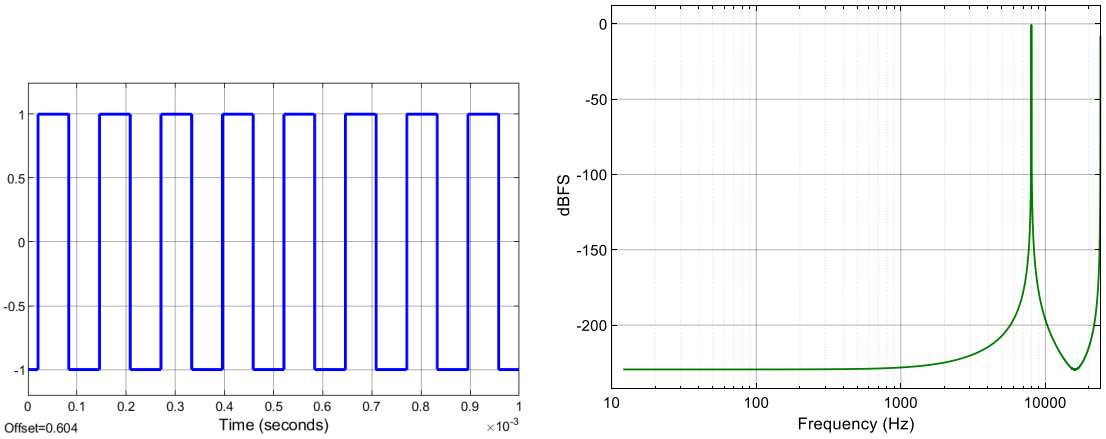


Figura 24. Representación temporal (a) y frecuencial (b) de la señal de pulso variable (pw = 50%) de 8372 Hz

En la Figura 25 se puede observar la representación frecuencial de la señal de pulso variable con ancho de pulso del 50% y frecuencia de oscilación de 440 Hz. En esta figura se aprecian claramente tanto los armónicos impares, característicos de las señales cuadradas (pw = 50%), como el efecto del aliasing ya descrito anteriormente.

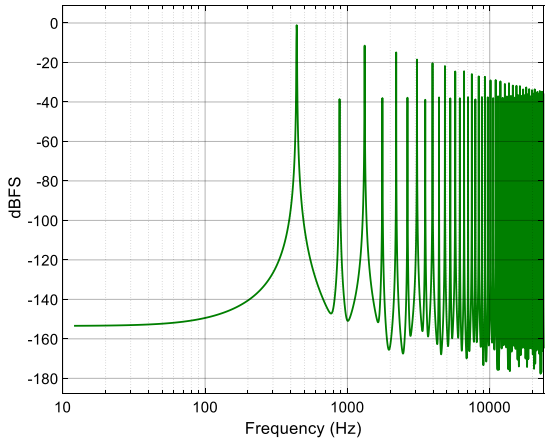


Figura 25. Representación frecuencial de señal de pulso variable (pw = 50%) de 440 Hz

En la Figura 26 se presentan varias señales de pulso variable con una frecuencia de oscilación de 440 Hz y diferentes anchos de pulso.

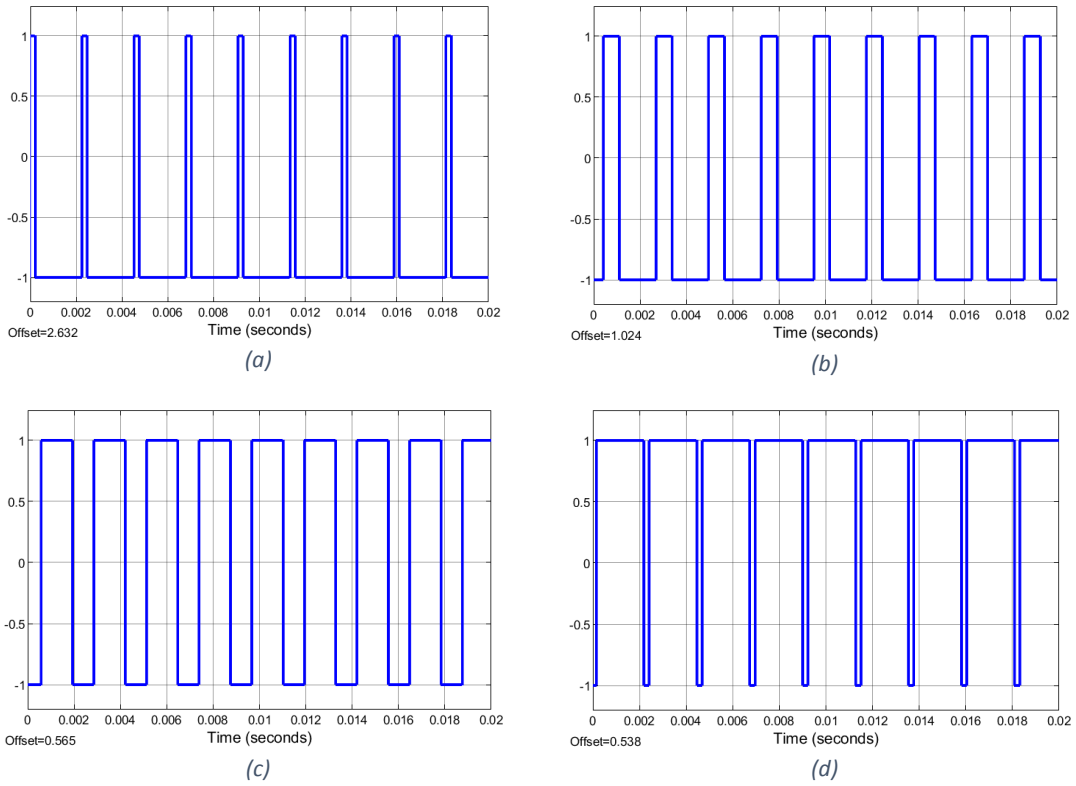


Figura 26. Representación frecuencial de la señal de pulso variable de 440 Hz, con ancho de pulso del 10% (a), 30% (b), 60% (c) y 90% (d)

3.5 El filtro

En los sintetizadores, los filtros tienen la función de alterar el contenido espectral de la señal generada por los osciladores, mediante la acentuación, atenuación o, incluso, el filtrado completo de determinados armónicos. Para ello se utilizan filtros resonantes, con frecuencia de corte y factor de calidad (Q, en adelante, resonancia) parametrizables por el usuario de forma interactiva. Este último aspecto es determinante a la hora de elegir el tipo de filtro a utilizar para el diseño del sintetizador. Debe tratarse de un filtro variable que permita el ajuste de la frecuencia central y del nivel de resonancia, puesto que en función la frecuencia y el tipo de señal generada por el oscilador, se ajustarán estos parámetros de forma diferente para obtener el timbre deseado. El ajuste de los parámetros puede ser realizado de forma manual, por el usuario, o mediante modulaciones, como se verá más adelante, lo que dota de un alto grado de dinamismo al sonido generado.

Para poder llevar a cabo de forma eficiente los ajustes en tiempo real, el cálculo de los coeficientes del filtro en función de los parámetros ajustados por el usuario no debe implicar un número elevado de operaciones. Otro criterio de diseño deseable relacionado con los parámetros del filtro es que éstos sean independientes el uno del otro, aunque internamente puedan utilizarse conjuntamente para el cálculo de los coeficientes del filtro. De este modo, el usuario podrá ajustar ambos parámetros de forma individual sin que el cambio de uno provoque una modificación en el otro.

Dependiendo del tipo de diseño empleado, en los sintetizadores se suelen encontrar filtros que funcionan en diferentes modos: paso bajo, paso alto, paso banda o elimina banda. De todos

ellos, el más habitual es el paso bajo, que suele estar presente en la mayoría de diseños. El orden de estos filtros suele ir desde orden 2 (12 dB/oct) hasta orden 4 (24 dB/oct).

A la hora de diseñar este tipo de filtros o de integrarlos en un diseño ya existente, también se debe tener en cuenta el efecto que producen los valores elevados del parámetro de control de la resonancia. A medida que se aumenta el valor del parámetro Q, se produce un aumento de la ganancia de la señal en la frecuencia de resonancia, pudiendo llegar incluso a incrementos superiores a +20 dB. Este aumento de la ganancia en la frecuencia de resonancia del filtro implica un incremento de la amplitud de la señal y, por tanto, un aumento del rango dinámico de la señal, lo que requiere un mayor número de bits en la parte entera del formato numérico utilizado para representar la señal. Existen varias opciones para afrontar este problema, que van desde la atenuación a medida que se aumenta la resonancia, hasta el aumento del rango dinámico de la señal de salida para permitir elevados valores de amplitud, pasando por el uso de saturadores, bien en el propio diseño del filtro o a su salida. Debe tenerse en cuenta que el uso de saturadores introducirá nuevos armónicos en la señal, lo cual no tiene por qué considerarse necesariamente un efecto negativo, puesto que en algunos casos se trata de un efecto buscado. Otra alternativa, con un enfoque más complejo, consiste en ligar la ganancia del filtro al valor de la resonancia, con el fin de mantener controlado el rango dinámico de la señal, tal y como se hace en la implementación del filtro de escalera de Moog ([[18]]).

Finalmente, un último aspecto deseable en los filtros empleados en síntesis de audio es su estabilidad en todo el rango audible aunque, en ocasiones, este rango puede considerarse reducido al rango de trabajo del dispositivo.

Con el fin de cumplir con los requisitos expuestos, en síntesis digital de audio suelen utilizarse los filtros denominados *Virtual Analog (VA)*, que consisten en emulaciones digitales de diseños de filtros analógicos.

Para el presente trabajo se ha optado por utilizar el diseño de filtro en variables de estado (*state variable filter*), que fue introducido por primera vez en [19] y cuya versión digital fue planteada inicialmente por Hal Camberlin en [20] y analizada posteriormente por múltiples autores ([21–23]), quienes han ido proponiendo ligeros cambios sobre el diseño inicial o en los rangos de valores de los coeficientes. Estos cambios han ido orientados a mejorar la respuesta en frecuencia y la estabilidad del filtro dentro del rango audible.

Además del filtro de estado variable, existen otros diseños que cumplen también con los requisitos expuestos y que han sido utilizados en diferentes sintetizadores de gran éxito, como por ejemplo: el filtro del Oberheim SEM, el filtro en escalera de Moog (*Moog ladder filter*), el filtro Sallen-Key Korg35, etc. En [14] se realiza una breve descripción de ellos y en su bibliografía pueden encontrarse referencias a descripciones más detalladas de cada uno de los diseños.

3.5.1 Filtro digital en variables de estado

Tal y como se describe en [20], el filtro analógico de estado variable cumple con las características deseables para utilizarse como filtro en un sintetizador de audio. En este caso, se trata de un filtro de segundo orden que, además, permite obtener las diferentes salidas de filtro paso alto, paso bajo y paso banda con un único circuito, sin necesidad de cálculos adicionales. También es posible obtener la salida de filtro rechaza banda con la incorporación de un sumador adicional, a partir de las salidas de los filtros paso alto y paso bajo.

Este filtro tiene una respuesta en frecuencia con una caída de 12 dB/oct a partir de la frecuencia de central para las salidas paso bajo y paso alto, y una caída de 6 dB/oct para la pasa banda.

El esquema de la versión digital de dicho filtro se presenta en la Figura 27. Como se puede observar en la esquema del filtro, los únicos coeficientes que tiene el circuito son F_c y Q_c. El primero de ellos está relacionado con la frecuencia central del filtro y el segundo con la resonancia.

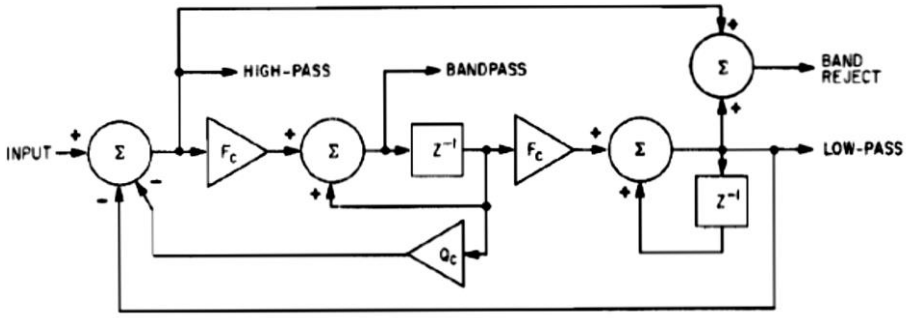


Figura 27. Versión digital del filtro de estado variable original (Chamberlin, 1985)

Con el fin de emular de la forma más fiel posible el comportamiento de la versión analógica del filtro y su estabilidad, en la medida de lo posible, los valores de estos coeficientes quedan limitados a un determinado rango. El valor de Q_c debe estar en el rango de valores [0..0.5], siendo los valores inferiores los que provocarán una mayor resonancia. Su relación con el factor de calidad del filtro (Q) viene dada por (11).

$$Q_c = \frac{1}{Q} \tag{11}$$

El segundo coeficiente que permite el ajuste del filtro es F_c que queda definido por (12), donde f_c se corresponde con la frecuencia central a la que se quiera ajustar el filtro.

$$F_c = 2 \cdot sin\left(\frac{\pi f_c}{fs}\right) \tag{12}$$

Sin embargo, el diseño original planteado por Chamberlin tiene algunas limitaciones de estabilidad. Se ha podido comprobar que, para una frecuencia de muestreo de 48 kHz y un valor de $Q = 4$, el filtro deja de ser estable a partir de una frecuencia central de unos 12 kHz, aproximadamente. Es decir, el filtro deja de ser estable a partir de $f_c = \frac{f_s}{4}$.

Para evitar este problema, existen varias alternativas. De entre ellas, para el presente trabajo se ha optado por mantener el diseño del circuito original y realizar un ajuste en el cálculo de los coeficientes, planteado en [23].

Para evitar confusiones a la hora de consultar la bibliografía, a partir de este punto se utilizarán las letras F y D para denotar los coeficientes del filtro (hasta ahora F_c y Q_c, respectivamente). El coeficiente D mantendrá el valor definido para Q_c definido en (11), mientras que el valor del coeficiente F se calculará a partir del valor de F_c definido en (12) y del valor de D, según (13).

$$F = F_c(1.85 - 0.85 \cdot D \cdot F_c) \tag{13}$$

En la Figura 28 se puede observar el modelo del filtro implementado en Simulink® ya con la nueva nomenclatura.

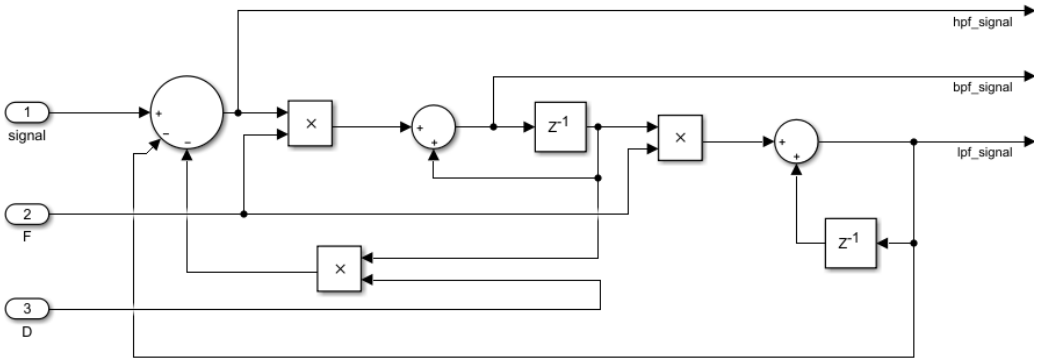


Figura 28. Diseño en Simulink® del filtro de estado variable

De las tres salidas proporcionadas por el filtro, en el presente trabajo se analiza únicamente la salida del filtro paso bajo, quedando pendiente el análisis análogo del resto de salidas del filtro para futuras líneas de trabajo.

Gracias al ajuste en el cálculo del coeficiente relacionado con la frecuencia central del filtro, planteado en [23], se consigue controlar el comportamiento del filtro paso bajo a altas frecuencias (Figura 29 y Figura 30). Como se puede observar en la Figura 29, para un valor constante de $Q = 4$, se produce un ligero aumento en la ganancia a medida que se aumenta la frecuencia. En la Figura 30 se observa que el filtro mantiene un desfase de 90° a medida que se aumenta la frecuencia central.

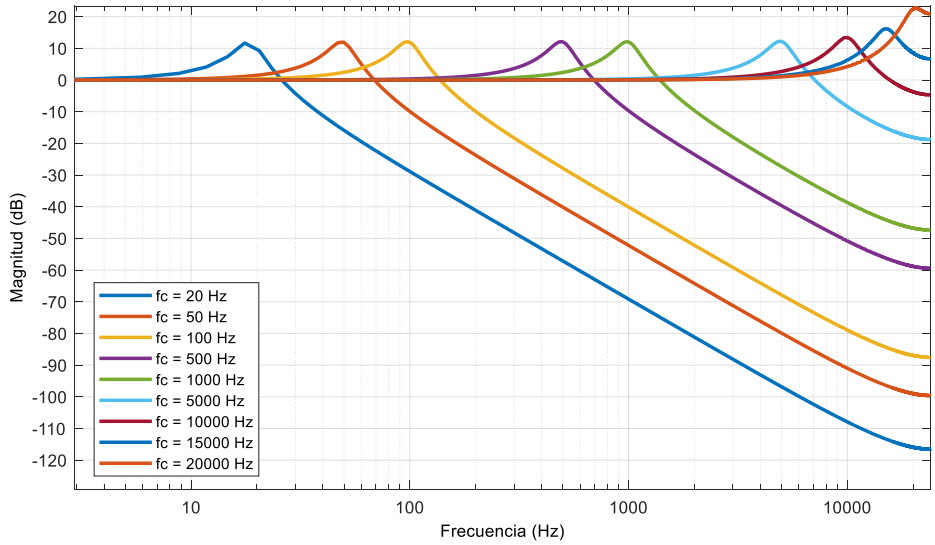


Figura 29. Respuesta en magnitud al impulso del filtro paso bajo para Q = 4

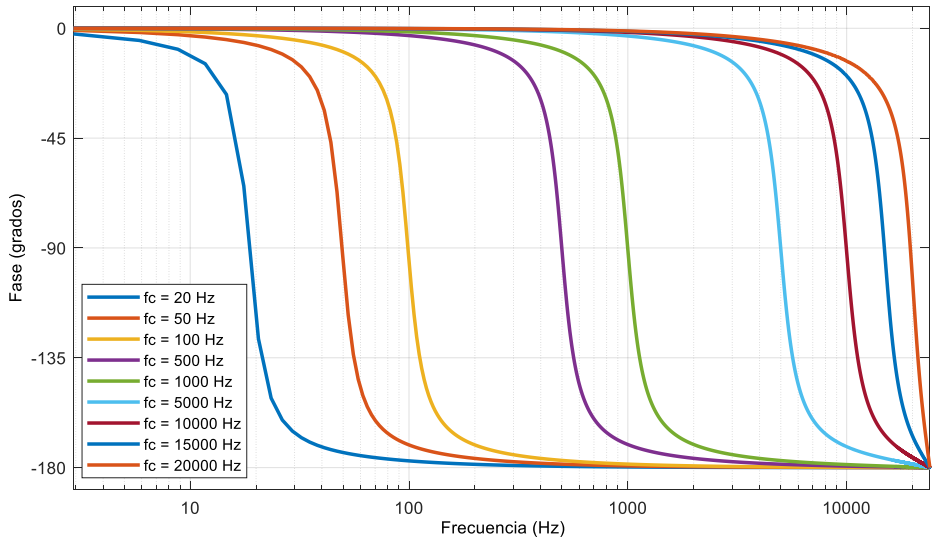


Figura 30. Fase de la respuesta al impulso del filtro paso bajo para Q = 4

Como se puede observar en la respuesta al impulso del filtro de la Figura 31, a medida que se aumenta el valor del parámetro Q, aumenta la ganancia en la frecuencia central. En un sistema de precisión finita, este aumento debe controlarse para evitar pérdidas de precisión por asignar demasiados bits a la parte entera de la señal.

Para esta primera versión del sistema no se han aplicado ninguna de las propuestas para el control de ganancia del filtro descritas en [6], [12] o [14], quedando pendiente su implementación para futuras líneas de trabajo. Como alternativa, se ha limitado el rango del parámetro Q a un máximo de 4, el cual acaba generando una señal con una amplitud de pico cercana a 5 en el peor caso, que se da cuando la señal del oscilador es una senoidal con $f_o = f_c$ y f_o tome su máximo valor posible, que en este caso será de 8372 Hz. Por tanto, a la salida del filtro la señal quedará representada en un tipo de datos con 4 bits para representar la parte entera de su amplitud, con su correspondiente signo.

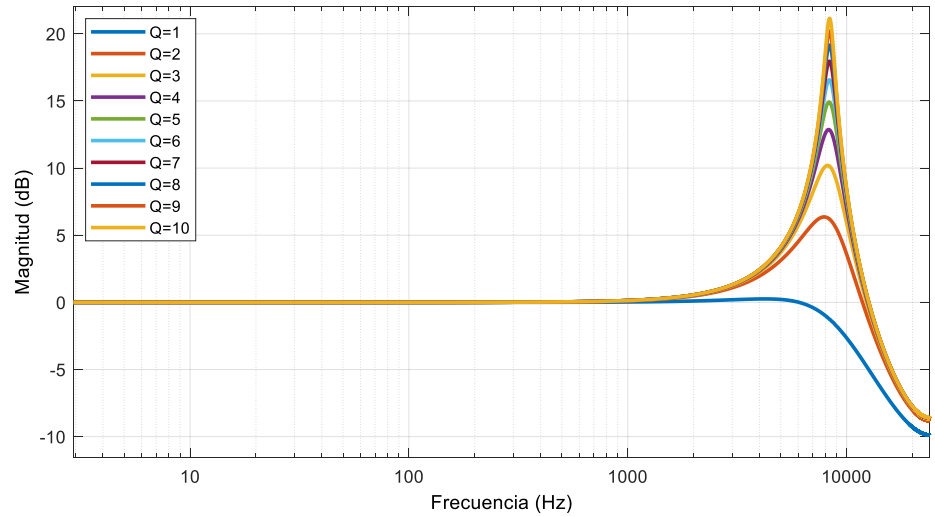


Figura 31. Magnitud de la respuesta al impulso del filtro paso bajo con f_c = 8372 Hz

3.5.2 Modelo del filtro en precisión finita

Una vez analizado el comportamiento del filtro, a lo largo de este apartado se describen los detalles de implementación del modelo que, como el resto de subsistemas, se ha particionado en una parte de procesado *software* y otra de lógica programable.

Todos los cálculos relativos a la conversión de los valores de los parámetros, ajustados de forma interactiva por el usuario, a coeficientes del filtro, se realizan en coma flotante en la parte de procesado *software* del SoC. Por otro lado, el modelo del filtro presentado en la Figura 28, que realiza el procesado de la señal de audio, se ha implementado en la FPGA, por lo que requiere de una conversión a formatos de datos de precisión finita.

Tal y como se describe en [21], tanto la cuantificación de los coeficientes del filtro como la de todos los operadores en la ruta de la señal procesada puede producir efectos no deseables. Una mala cuantificación de los coeficientes puede provocar un error a la hora de ubicar los polos y los ceros del filtro, generando una salida que no se corresponda con la del modelo en coma flotante. También puede verse afectada la amplitud de la señal, debido a las multiplicaciones internas del filtro utilizando valores redondeados de los coeficientes. Por otro lado, la cuantificación de la señal interna determinará el rango dinámico máximo de la señal y la cantidad de ruido que pueda llegar a introducir el filtro. En [5] se puede consultar un análisis en detalle de estos efectos.

Tal y como ya se ha descrito anteriormente, el filtro tiene únicamente dos coeficientes. De los dos coeficientes, el rango de valores que puede tomar el coeficiente D es el más limitado. Los valores típicos que puede tomar el parámetro Q suelen ir entre 1 y 10, lo que genera el siguiente conjunto de valores para D:

```
D = [1.0000, 0.5000, 0.3333, 0.2500, 0.2000, 0.1667, 0.1429, 0.1250, 0.1111, 0.1000]
```

Utilizando el comando `fi` de Matlab®, se determina que, para que no haya pérdida de precisión al cuantificar los posibles valores de D, debe usarse una precisión de [15,14] sin signo:

```
>> fi (D, 0, 15, 14)

    1.0000  0.5000  0.3333  0.2500  0.2000  0.1667  0.1429  0.1250  0.1111  0.1000
```

Para el coeficiente F, el rango de valores es bastante más amplio, por lo que se decide utilizar un método alternativo, que consiste en la evaluación de la respuesta en frecuencia del filtro.

Para comprobar el efecto de la cuantificación de los coeficientes, se simula el modelo del filtro en coma flotante y se compara la respuesta en frecuencia obtenida utilizando coeficientes en coma flotante de doble precisión con la obtenida utilizando coeficientes en precisión finita.

Como se puede observar en la Figura 32 y en la Figura 33, con 18 bits para el tamaño de palabra del coeficiente F, la respuesta del filtro es prácticamente idéntica a la obtenida utilizando coeficientes en coma flotante. Teniendo en cuenta que los multiplicadores de la FPGA son de 18x25 bits, se da por bueno este formato, por lo que se decide utilizar una precisión [18,17] sin signo para el coeficiente F. Se ha comprobado que, con un número de bits inferior, la respuesta en frecuencia del filtro se ve alterada (Figura 32a).

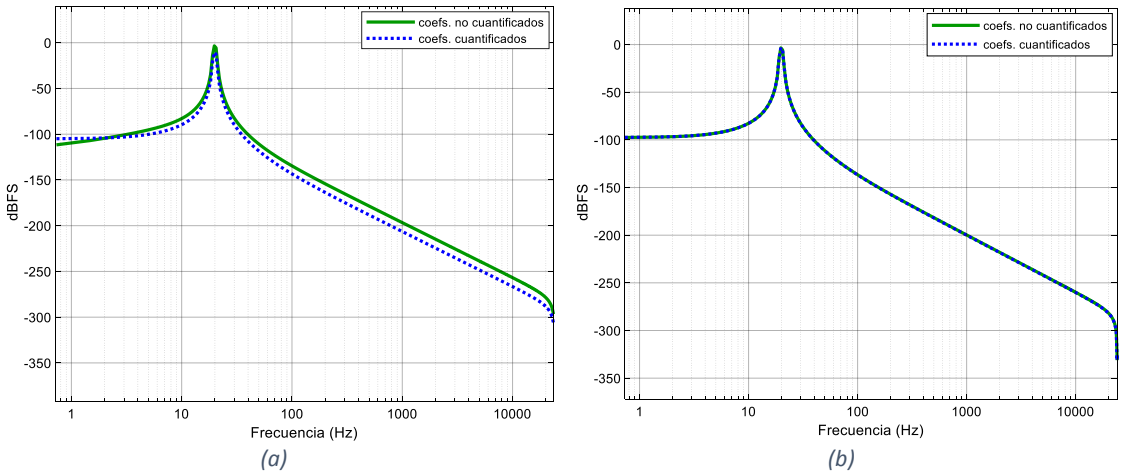


Figura 32. Efecto de la cuantificación de los coeficientes: (a) coeficiente F cuantificado con 10 bits; (b) coeficiente F cuantificado con 18 bits

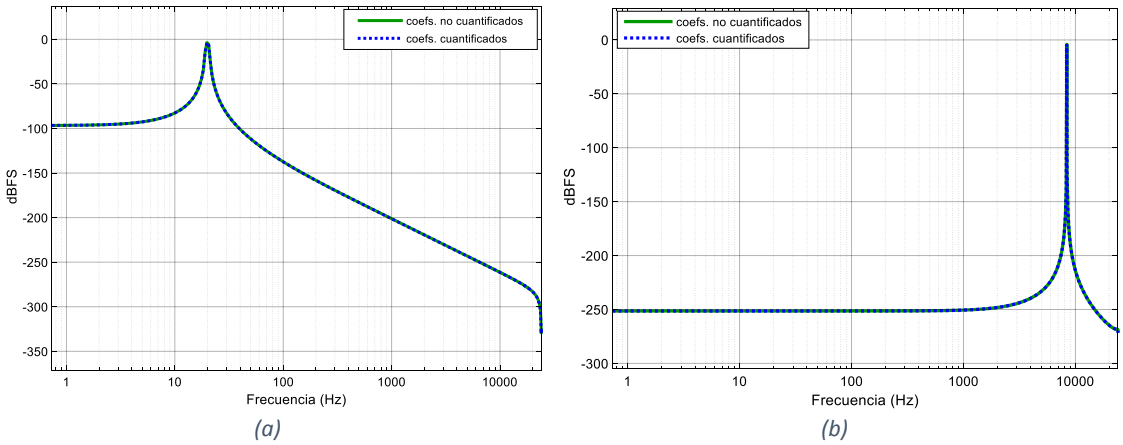


Figura 33. Respuesta del filtro para señal senoidal: (a) fo = fc = 20Hz y Q =1; (b) fo = fc = 8372 Hz y Q = 4

Una vez determinados los tamaños de palabra de los coeficientes, se procede a la conversión del modelo del filtro a precisión finita. Para ello, se hace uso de la herramienta Fixed-Point Designer™, que permite realizar la recolección de rangos del modelo y, a partir de dicha recolección, sugiere los formatos de datos a aplicar a los distintos operadores que componen el modelo.

Tras realizar diferentes pruebas con distintos tamaños de palabra, se llega a una configuración en la que se utilizan 48 bits para representar la señal internamente en el filtro. En la Figura 34 se puede observar que con dicho tipos de datos, se cubren los rangos simulados. Esta herramienta, además comprueba que se cubran los rangos deducidos a partir de los valores máximos y mínimos establecidos para las señales de entrada y salida. Durante la recolección de rangos, se utilizan los tipos de datos definidos anteriormente para los coeficientes F y D.

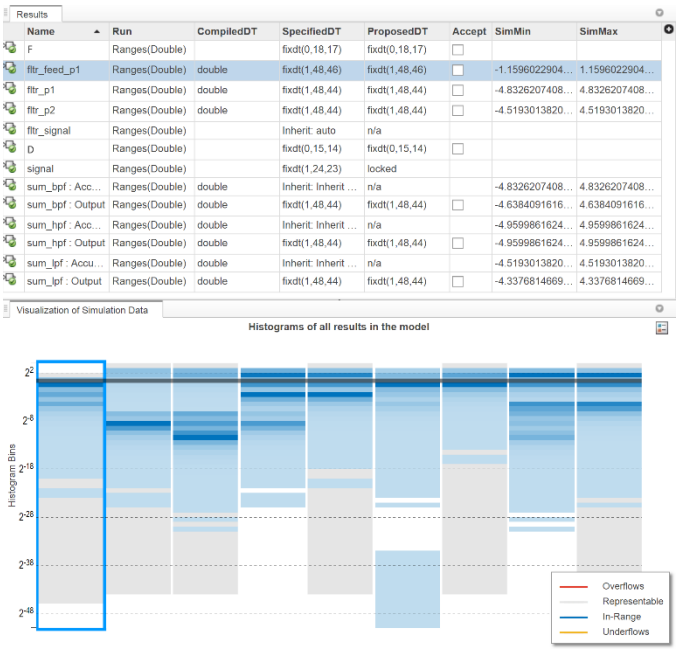


Figura 34. Recolección y propuesta de tipos de datos con Fixed-point Designer™

En la Figura 35 se presenta el modelo con el tipo de datos asociado a cada operador y a las señales de entrada.

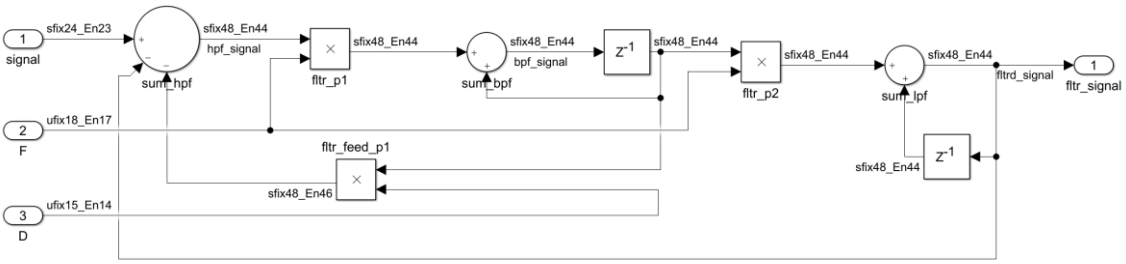


Figura 35. Modelo del filtro convertido a precisión finita

Una vez generado el modelo en precisión finita, se procede a la validación del mismo. Para ello se ha comprobado que el nivel de ruido introducido por el filtro, utilizando precisión finita, sea lo más próximo al obtenido con el modelado en coma flotante de doble precisión y, en cualquier caso, no sea perceptible por el usuario. Para comprobar el ruido introducido por el filtro, se utiliza como señal excitadora del filtro una señal senoidal en las frecuencias extremo del rango en estudio (27.5 Hz y 8372 Hz) y en una frecuencia intermedia de 880 Hz. En cada una de las frecuencias analizadas se analiza la respuesta con valores de Q en ambos extremos (1 y 4).

En la Figura 36 se presentan varias respuestas del filtro ante una señal de entrada senoidal a 27.5 Hz y el filtro ajustado con una frecuencia central igual a la de oscilación y la resonancia en sus valores extremos. Se puede apreciar la diferencia de la respuesta obtenida en función del formato de representación de los datos y cómo un tamaño de palabra de 30 bits podría servir, a priori, para emular el funcionamiento del filtro en coma flotante de precisión simple, pero para obtener una respuesta equivalente a la obtenida con coma flotante de doble precisión se requieren un mayor número de bits.

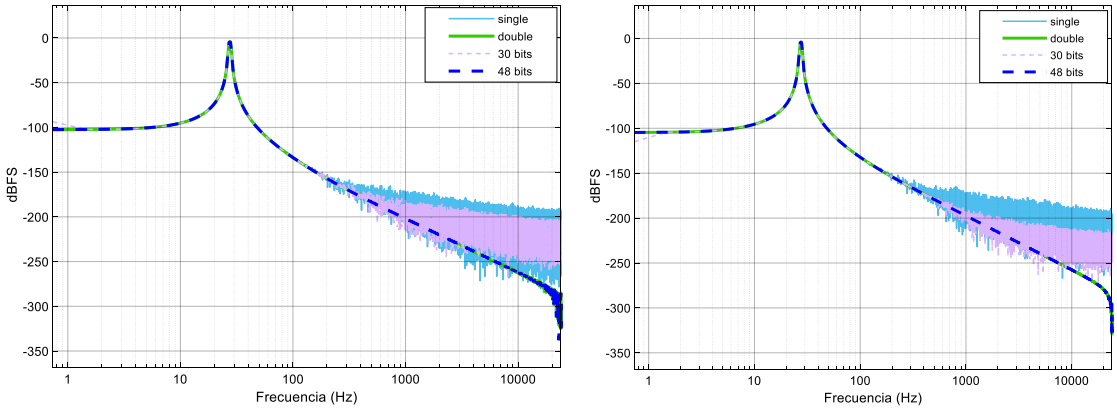


Figura 36. Comparativa de modelos con diferente precisión para $f_o = f_c = 27.5$ Hz: (a) Q = 1; (b) Q = 4

Con el fin de facilitar la interpretación del resultado, en las siguientes figuras se comparan únicamente el resultado obtenido con la solución propuesta, en precisión finita con un tamaño de palabra de 48 bits, y el obtenido utilizando coma flotante de doble precisión. También se incluye la señal original sin filtrar como referencia.

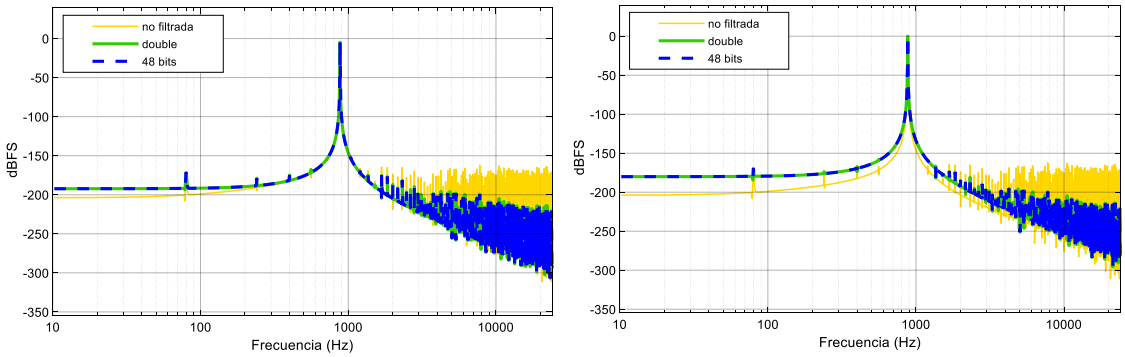


Figura 37. Comparativa entre coma flotante y precisión finita para $f_o = f_c = 880$ Hz: (a) Q = 1; (b) Q = 4

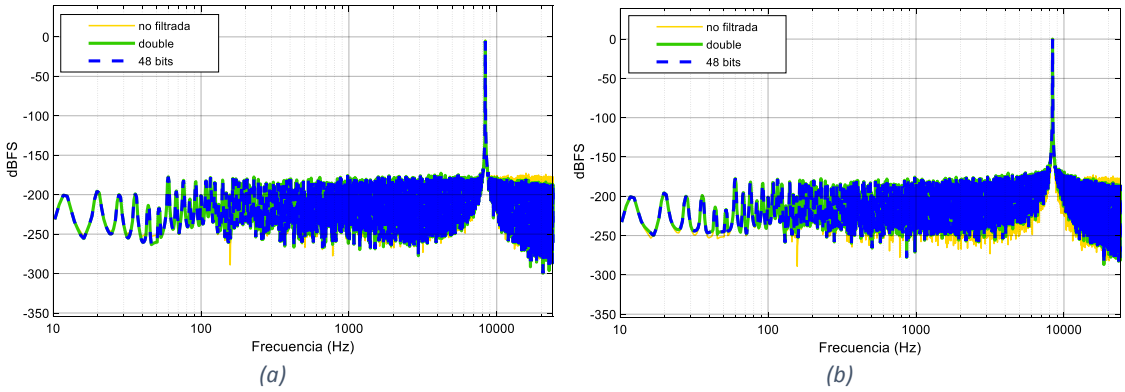


Figura 38. Comparativa entre coma flotante y precisión finita para fo = fc = 8372 Hz: (a) Q = 1; (b) Q = 4

En todos los casos se ha obtenido una relación señal/ruido (SNR) alrededor los 120 dB. Se ha comprobado que para valores altos de frecuencia central y de resonancia, se obtienen las SNR más bajas, estando en 117 dB para el caso en que $f_o = f_c$ 8372 Hz y $Q = 4$, frente a los 126 dB

de SNR de la señal original, tanto para el modelo en coma flotante como para el de precisión finita. En todos los casos, la SNR de la señal obtenida mediante precisión finita es prácticamente idéntica a la obtenida utilizando coma flotante de doble precisión, por lo que la solución propuesta se considera válida.

3.5.3 Ganancia del filtro

Una vez convertido el filtro a precisión finita y validado, queda un último aspecto por resolver. El filtro requiere de un formato de datos con 4 bits en la parte entera, para poder representar todo el rango dinámico de la señal. Sin embargo, el codificador de audio espera un dato de 24 bits en formato [24,0] y a la hora de realizar la conversión de la señal, interpreta 23 bits como parte fraccional, por lo que queda un único bit para representar la parte entera de la señal. Para entregar al codificador de audio una señal en formato [24,23], se requiere un escalado de la misma a la salida del filtro. Para ello, se plantean dos opciones: realiza un desplazamiento de 3 bits hacia la derecha, quedando la señal en el rango esperado por el codificador ([-1,1]) o realizar un desplazamiento de 4 bits y utilizar un saturador para evitar que la señal exceda el rango de [-1,1]. La primera opción presenta el inconveniente de realizar un escalado excesivo para la mayoría de casos, en los que la amplitud de la señal no lo requeriría, lo que supone un desaprovechamiento del rango dinámico disponible y una atenuación considerable de la señal en la mayoría de los casos. La segunda opción, realizará un escalado más ajustado de la señal, sin embargo, en los casos extremos planteados anteriormente, además, requerirá de una saturación de la señal para mantenerla dentro de los límites del codificador de audio, lo que introducirá nuevos armónicos en la señal. Puesto que esta situación sólo se dará en situaciones en las que se fuercen los valores límites, se opta por esta opción como primera aproximación.

Como futura línea de trabajo, se plantea el control de ganancia del filtro, evitando tener que realizar el escalado de la señal antes o después del filtro y la saturación sólo para algunos casos. Existen otros diseños de filtros que incorporan dicho control en la ruta de la señal, evitando así el exceso de ganancia provocado por niveles elevados del parámetro Q, como por ejemplo el filtro de en escalera de Moog.

Otro aspecto susceptible de optimizar en futuras líneas de trabajo es el análisis exhaustivo de los tipos de datos utilizados en cada uno de los operadores, minimizando al máximo el número de bits utilizado en cada uno de ellos, con el fin de reducir al máximo el uso de recursos de la FPGA.

3.6 El amplificador

El amplificador es la última etapa de la cadena de síntesis y se encarga de controlar la amplitud final de la señal generada por los osciladores y tratada por los filtros. En el caso de la síntesis digital, con el fin de no disminuir la resolución de la señal de audio asignando más bits a la parte entera del formato de datos, en lugar de amplificar la señal, se obtiene una mayor precisión diseñando internamente el control de ampliación como un atenuador. Tal y como se ha visto hasta ahora, la señal se genera en la etapa de los osciladores y posteriormente se ve amplificada en la etapa de filtrado, debido a los valores altos de resonancia. Por tanto, se considera que la señal que sale de la etapa de filtrado se corresponde con la máxima amplificación y, a partir de ahí, se ofrece al usuario la posibilidad de atenuarla, aunque externamente se perciba como una etapa de amplificación.

Para ello, se proporciona un control con un rango de valores entre 0 y 1, cuyo valor se multiplicará por la señal de salida del filtro. De este modo, el valor máximo (1) se corresponderá con la señal original generada y el resto de valores propiciarán la atenuación de dicha señal.

En esta etapa también es posible utilizar señales de modulación, que se describirán en la sección *Modulación de parámetros*. El uso del generador de envolvente aplicado a la amplitud de la señal permite moldear la evolución temporal del sonido, permitiendo generar con un mismo oscilador desde sonidos de carácter más percusivo, con un ataque y un tiempo de caída muy cortos, a sonidos que emulen el comportamiento de instrumentos de cuerda con tiempos de ataque y de *release* muy largos, por ejemplo. Por otro lado, si se aplica el oscilador de baja frecuencia a la amplitud de la señal, se obtendrá el efecto conocido como *tremolo*, que consiste en una modulación periódica en amplitud de la señal.

En el diseño planteado en el presente trabajo únicamente se hace uso del generador de envolvente como modulador de la señal de salida. En futuras líneas de trabajo, se propone la inclusión del LFO como fuente de modulación.

3.7 Modulación de parámetros

Tal y como se ha ido describiendo a lo largo del presente documento, los diferentes parámetros de ajuste de los distintos bloques del sintetizador pueden ser modulados mediante señales de control, con el fin de dotar de un mayor dinamismo al sonido generado. En síntesis de audio suelen emplearse principalmente dos tipos de moduladores: los generadores de envolvente (*EG*) y los osciladores de baja frecuencia (*LFO*). El funcionamiento de cada uno de ellos y sus detalles de implementación se describen en los siguientes apartados.

3.7.1 Generador de envolvente

Los generadores de envolvente permiten definir modulaciones temporales de los parámetros a los que se asignen, dotándoles de una evolución temporal a partir de un determinado evento (habitualmente el inicio de una nota) y de duración limitada, que coinciden con la forma de la envolvente configurada. Existen multitud de tipos de envolventes, cada una de ellas cuenta con unas determinadas etapas y con diferentes funciones de transición entre ellas, siendo las más comunes las lineales y las exponenciales. Los parámetros que permiten ajustar las diferentes etapas de una envolvente suelen ser de tipo temporal o de ganancia (Figura 39).

En el presente trabajo se considerará que la envolvente se iniciará al comienzo de cada nota, lo que se considerará como el evento `note_on`, que durará hasta que finalice dicha nota. Mientras dicho evento se mantenga activo, la envolvente irá evolucionando atravesando sus diferentes etapas. En función de su diseño, puede tener alguna etapa final que se inicie una vez el evento deje de estar presente, lo cual es bastante habitual.

En este caso se ha optado por implementar una envolvente de tipo ADSR (*Attack*, *Decay*, *Release* y *Sustain*), que cuenta con las cuatro etapas que su propio nombre indica. Cada una de estas etapas se ajusta mediante los siguientes parámetros:

- Tiempo de ataque (*attack*): tiempo que tarda el parámetro modulado en alcanzar el valor al cual se ha ajustado por el usuario. Su rango de valores se ha limitado entre 0.1 y 10 segundos, con una precisión de 1 ms.

- Tiempo de *decay*: tiempo que tarda el parámetro modulado en pasar del valor establecido por el usuario al nivel de *sustain*, definido en la envolvente. Su rango de valores se ha limitado entre 0.1 y 10 segundos, con una precisión de 1 ms.
- Nivel de *sustain*: es el factor por el que se multiplicará el valor ajustado por el usuario para el parámetro modulado, una vez haya pasado el tiempo de ataque y de *decay*, siempre y cuando se mantenga la pulsación de la tecla. Su rango de valores va desde 0 a 1.
- Tiempo de *release*: tiempo empleado para que el valor del parámetro modulado pase del nivel en el que se encuentra en el momento en que deja de estar presente el evento note_on hasta alcanzar el valor 0. Su rango de valores se ha limitado entre 0.1 y 10 segundos, con una precisión de 1 ms.

De las cuatro etapas, las tres primeras pueden ser interrumpidas si el evento `note_on` deja de estar presente durante el transcurso de alguna de ellas. En caso de darse dicha situación, se pasará directamente a la última etapa. Esta última etapa también puede ser interrumpida, si se produce nuevamente un evento `note_on` antes de que se cumpla el tiempo de *release* establecido, lo que provocaría un nuevo inicio del ciclo de la envolvente.

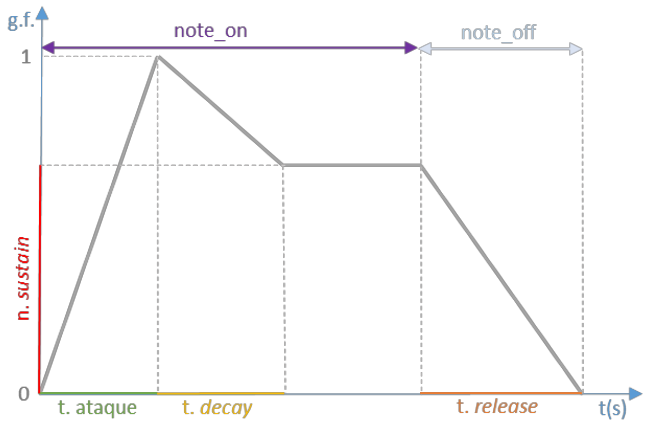


Figura 39. Envolvente ADSR

El comportamiento descrito para la envolvente se ha modelado mediante una máquina de estados de Moore, tal y como se puede observar en la Figura 40. Mediante la máquina de estados se calculan los factores de ganancia (etiquetados con el sufijo `_gf`) de cada una de las etapas de la envolvente. Para ello se utilizan como condiciones en las transiciones entre estados el evento `note_on` y el número de muestras que le corresponde a cada estado (etiquetados con sufijo _s), cuyo valor se calcula previamente en la parte de procesado *software* del SoC, a partir de los parámetros de la envolvente, ajustados por el usuario, y la frecuencia de muestreo del sistema.

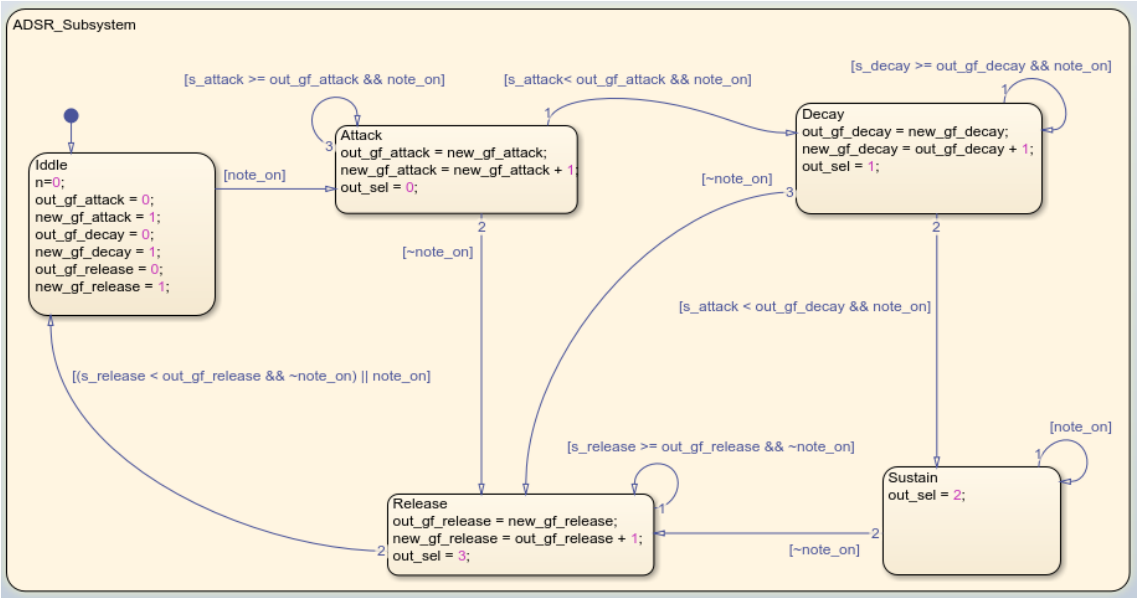


Figura 40. Máquina de estados de Moore para implementación de la envolvente ADSR

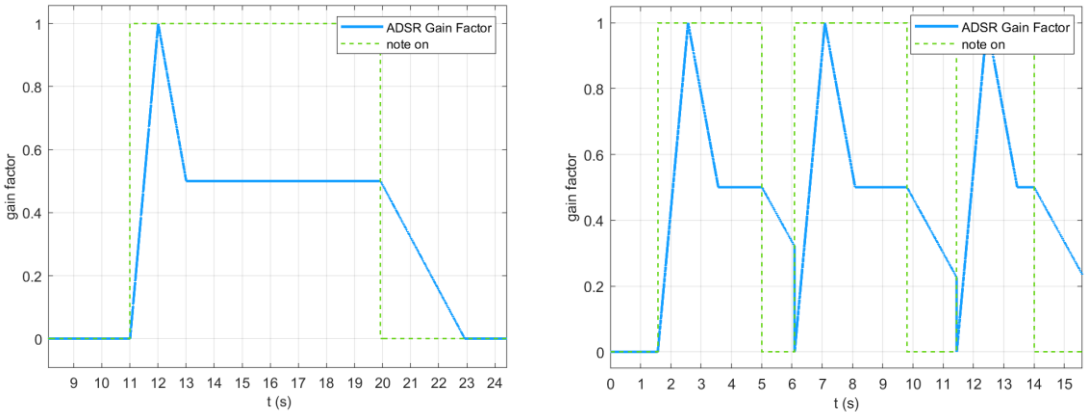


Figura 41. Simulaciones de la envolvente. (a) Envolvente complete; (b) Varias envolventes consecutivas con diferentes duraciones de notas

3.7.2 Oscilador de baja frecuencia (LFO)

Los osciladores de baja frecuencia son osciladores que suelen trabajar por debajo del rango audible, entre 0 y 20 Hz, y se utilizan para modular parámetros de forma periódica. De forma análoga a los osciladores utilizados para general la señal de audio, estos osciladores pueden generar diferentes formas de onda.

A diferencia de los osciladores utilizados para generar la señal de audio, en el caso de los LFO el análisis espectral de la señal no es tan relevante como el temporal, puesto que se trata de señales de control, no audibles. Esto implica algunas diferencias en su diseño con respecto a los osciladores de señal de audio. En primer lugar, al tratarse de osciladores de muy baja frecuencia, la etapa de generación de fase requiere de un mayor número de bits, para así poder generar señales próximas a los 0 Hz. En cambio, una vez generada correctamente la fase, la etapa de conversión de fase a amplitud requerirá de un número inferior de bits, puesto que en este caso

la precisión ya no es tan crítica y la posible aparición de espurios, debidos al truncamiento de la fase, no afectará a la señal de control.

A la hora de aplicar los LFOs se suelen proporcionar tres controles que, para el presente trabajo, tendrán las siguientes características:

- Forma de onda: se utilizan las mismas que en los osciladores de señal de audio (diente de sierra, senoidal, triangular o de pulso variable).
- Frecuencia del LFO: este parámetro habitualmente se conoce como *rate* y determina la velocidad a la que irá modulándose el parámetro destino entre los valores máximo y mínimo. Su rango de valores va desde 0.1 a 20 Hz, con incrementos de 0.1 Hz.
- Profundidad del LFO: este parámetro habitualmente se conoce como *depth* y determina el rango entre el que oscilará el parámetro destino. Su rango de posibles valores es desde 0 a 1, con incrementos de 0.05.

Para lograr que el LFO oscile a una frecuencia de 0.1 Hz, se ha utilizado un acumulador de 28 bits, mientras que para la etapa de conversión de fase a amplitud de los osciladores internos, se han utilizado tablas de 8 bits.

Cuando el LFO se utiliza para modular un parámetro de frecuencia, bien sea la frecuencia de oscilación, la frecuencia de resonancia del filtro o cualquier otra presente en el diseño, el efecto del LFO sobre dicha frecuencia se suele aplicar de forma exponencial, es decir, en lugar de modular la frecuencia de forma lineal, la variación va desde la octava inferior a la superior de la frecuencia que se esté modulando. Para ello, el rango de valores de salida del LFO debe estar entre 0.5 y 2, de modo que al multiplicarlo por el valor de la frecuencia modulada se obtenga el efecto deseado.

Para llevar a cabo dicha conversión, la salida de los diferentes tipos de osciladores del LFO pasa por una LUT que se encarga de mapear los valores de la señal de entrada a su valor exponencial. En la Figura 42 se presenta el diagrama de bloques utilizado para generar la salida exponencial del LFO. A partir de la señal generada por los osciladores (`lfo_osc_out`), se le aplica el parámetro *depth* y posteriormente se genera la salida exponencial (`lfo_out`) mediante el uso de una LUT.

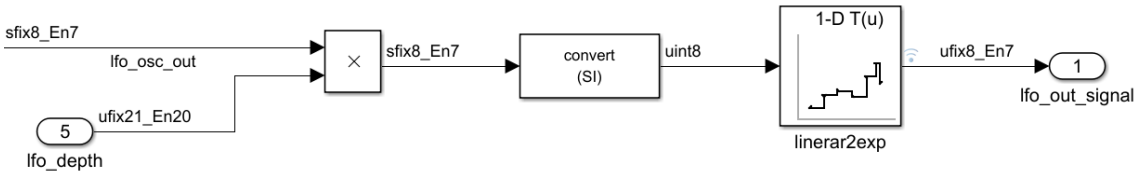


Figura 42. Conversión de la salida del oscilador del LFO a exponencial

En la Figura *43* se presenta la salida exponencial del LFO para las diferentes formas de onda, con un valor del parámetro *depth* de 1. Como puede observarse, los valores de amplitud de la señal generada del oscilador interno van desde -1 a 1, mientras que los de la señal de salida se encuentran en el intervalo entre 0.5 y 2, siguiendo una evolución exponencial. En el caso de la señal de pulso variable, los cambios provocados por el LFO siempre pasarán del valor mínimo al valor máximo.

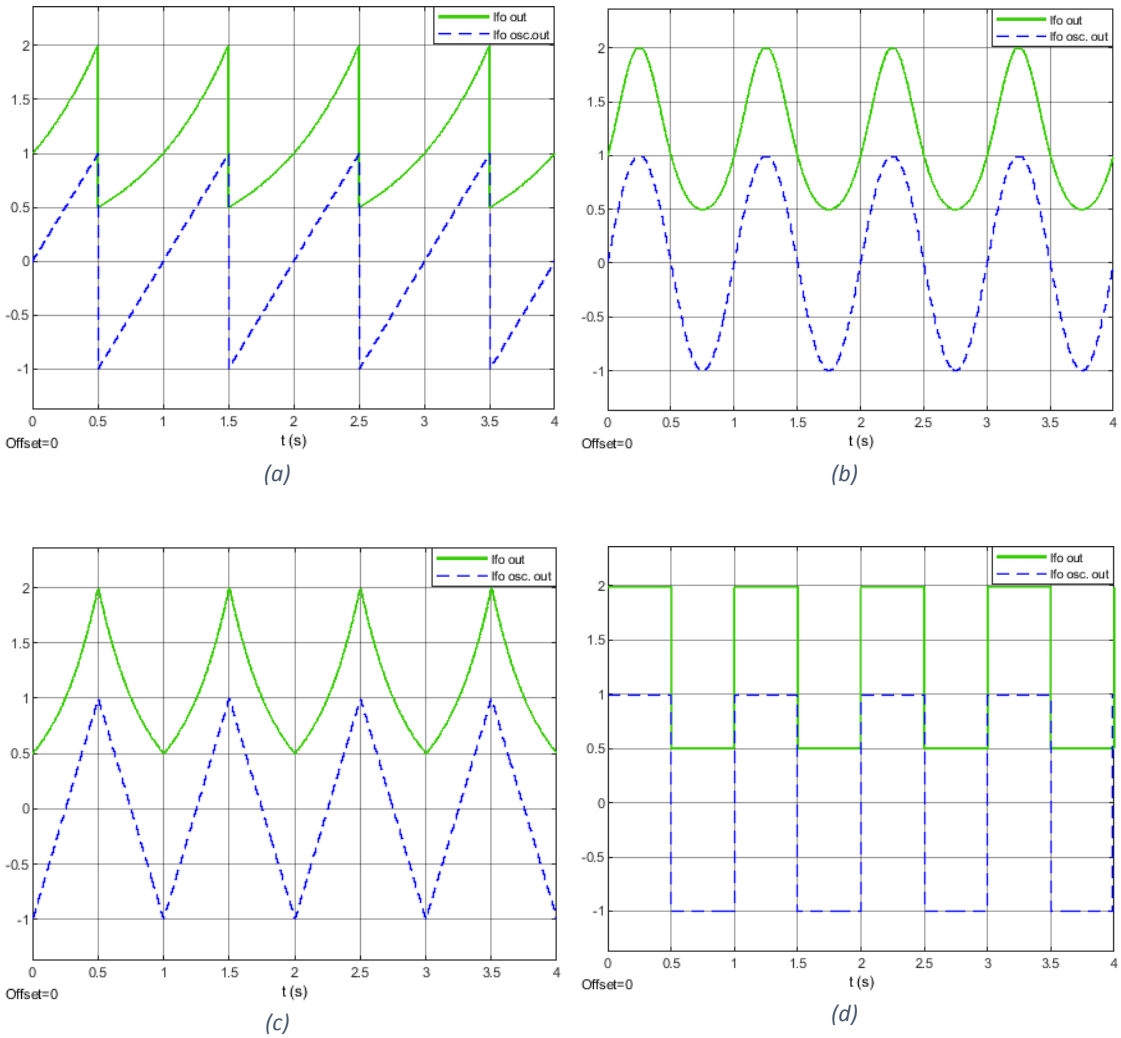


Figura 43. Salida del LFO para las diferentes formas de onda: (a) diente de sierra; (b) senoidal; (c) triangular; (d) pulso variable (50%)

A la hora de aplicar el LFO sobre cualquier parámetro del sistema, basta con multiplicar la salida del mismo por el parámetro a modular y éste comenzará a oscilar entre los valores límite del LFO.

4. Generación y despliegue del prototipo

4.1 Generación del código VHDL y despliegue del bitstream en la FPGA del SoC

Una vez completado el diseño y validados cada uno de los bloques de forma individual y en su conjunto, se procede a la generación del código VHDL mediante HDL Coder™. Para ello se utiliza el asistente HDL Workflow Advisor (Figura 44).

En primer lugar, se configura el modelo de placa destino, la herramienta de síntesis (Xilinx Vivado®) y se selecciona el diseño de referencia a utilizar (*Audio System with AXI4 Stream Interface*). A continuación, se definen las interfaces de entrada y los *pins* de conexión de la FPGA, para cada uno de los datos de entrada.

Una vez realizada la configuración inicial, la herramienta realiza la comprobación del modelo, para comprobar que éste sea compatible con la generación de código automática.

A partir de este punto, se establecen las opciones de optimización y, finalmente, se procede a la generación del código VHDL y del *bitstream* que contiene la implementación del código VHDL sintetizado.

Llegados a este punto, si no surge ningún problema, se procede a desplegar el código en la FPGA.

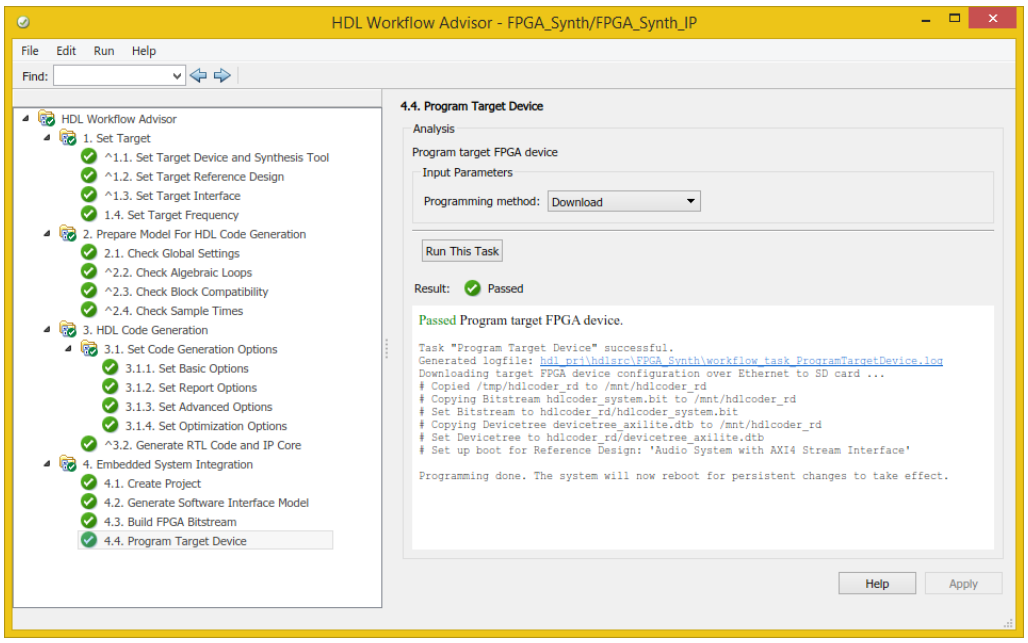


Figura 44. Etapas del asistente HDL Workflow Advisor

4.2 Generación del código C y despliegue los ARM del SoC

Durante la ejecución del asistente HDL Workflow Advisor, se genera un modelo de Simulink que actúa de interfaz *software*, permitiendo controlar el sistema desplegado en la FPGA del SoC.

Al ejecutar el modelo generado de forma externa, Embedded Coder® generará el código C correspondiente a la parte *software* del modelo, lo desplegará sobre los procesadores ARM® y lanzará su ejecución en el SoC, a través del sistema linux preinstalado durante la configuración inicial de Embedded Coder®.

Si no surge ningún problema durante el proceso, el sistema completo quedará desplegado sobre el SoC e integrado con el codificador de audio de la placa de desarrollo. A partir de este momento, el sintetizador diseñado podrá controlarse desde el modelo de Simulink generado por HDL Coder™ mientras se ejecuta en la placa de desarrollo.

4.3 Análisis de recursos utilizados por el sistema diseñado

En el diseño del sistema planteado, la mayor carga de procesamiento se encuentra en la parte de lógica programable del SoC, que es la encargada de realizar el procesado de la señal. La parte de procesado *software* se limita únicamente a realizar cálculos de los parámetros de entrada del bloque FPGA, a partir de los ajustes realizados por el usuario, por lo que no supone ningún cuello de botella del sistema.

La capacidad de procesamiento de la FPGA viene determinada por el aprovechamiento que se haga de los ciclos de reloj y de los recursos disponibles en ésta. El primer aspecto vendrá limitado por el camino crítico del sistema diseñado y el segundo por elementos utilizados para diseñar el sistema (LUTs, multiplicadores, memoria, etc.) y por los formatos de datos utilizados.

Tras generar el modelo diseñado, se ha obtenido un camino crítico de 28.33 ns, lo que implica una frecuencia máxima de trabajo de la FPGA de 35 MHz. Si se tiene en cuenta que la frecuencia de muestreo de trabajo del sistema es de 48 kHz, existe un amplio margen de mejora para optimizar el sistema mediante técnicas avanzadas, como el pipeline interleaving. El uso de estas técnicas permitiría aumentar las capacidades de procesado secuencial del sistema, pudiendo procesar un elevado de voces en cada periodo de muestreo de audio.

En cuanto a los recursos utilizados por el sistema, en la Tabla 1 se presenta el resumen obtenido mediante el *software* Xilinx Vivado®, una vez generado el *bitstream* del sistema. En la Tabla 2, partiendo del número total de recursos de la FPGA ([24]), de los recursos utilizados por el resto de subsistemas incluidos en el diseño de referencia y de los recursos utilizados por el sintetizador diseñado, se obtienen los recursos disponibles tras la implementación de todo el sistema en la FPGA y se calcula un factor de réplica del sistema. Este factor de réplica del sistema se ha obtenido dividiendo el número total de recursos disponibles tras la implementación del sistema en la FPGA entre el número de recursos utilizados por el sistema diseñado. Tomando el valor mínimo obtenido de entre los diferentes tipos de recursos, se obtiene un factor de 10, lo que implica que podría replicarse la estructura de bloques diseñada hasta 10 veces y el sistema podría ser implementado en la FPGA.

Como se ha podido comprobar, el sistema dispone de un amplio margen para la ampliación de sus capacidades, considerando su implementación sobre el SoC utilizado para el presente trabajo. Este aumento en sus capacidades, podría traducirse de forma directa en un aumento considerablemente de su polifonía y su capacidad politímbrica.

Name	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as Logic	LUT as Memory	LUT Flip Flop Pairs	Block RAM Tile	DSPs
system_top_wrapper	3042	3644	38	8	1553	2823	219	1391	11,0	19
system_top_i (system_top)	3042	3644	38	8	1553	2823	219	1391	11,0	19
FPGA_Synt_ip_0 (system_top_FPGA_Synt_ip_0_0)	1742	1670	0	0	791	1622	120	722	9,5	19
FPGA_Synt_ip	1723	1665	0	0	785	1603	120	722	9,5	19
FPGA_Synt_ip_dut	1419	1009	0	0	543	1333	86	606	9,5	19
FPGA_Synt_ip_src_FPGA_Synth_IP	1419	1009	0	0	543	1333	86	606	9,5	19
FPGA_Synt_ip_src_svf	514	190	0	0	155	479	35	168	0,0	9
FPGA_Synt_ip_src_oscillator	180	121	0	0	85	175	5	57	7,5	0
FPGA_Synt_ip_src_lfo	114	138	0	0	73	114	0	46	2,0	1
FPGA_Synt_ip_src_adsr	441	334	0	0	185	441	0	198	0,0	7
FPGA_Synt_ip_axi_lite	149	399	0	0	187	149	0	18	0,0	0
FPGA_Synt_ip_axi4_stream_slave	6	7	0	0	4	6	0	4	0,0	0
FPGA_Synt_ip_axi4_stream_master	149	250	0	0	133	115	34	94	0,0	0

Tabla 1. Resumen de recursos utilizados por el sintetizador

	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as Logic	LUT as Memory	LUT Flip Flop Pairs	Block RAM Tile	DSPs
Recursos totales	53200	106400	26600	13300	13300	53200	17400	53200	140	220
Recursos utilizados por resto de susbsistemas	1300	1974	38	8	762	1201	99	669	1,5	0
Recursos utilizados por el sintetizador de audio	1742	1670	0	0	791	1622	120	722	9,5	19
Recursos disponibles	50158	102756	26562	13292	11747	50377	17181	51809	129,0	201
Factor de réplica del sistema	28,79	61,53	inf	inf	14,85	31,06	143,18	71,76	13,58	10,58

Tabla 2. Disponibilidad de recursos para ampliación del sistema

5. Futuras líneas de trabajo

A lo largo del presente trabajo se ha abordado el diseño de los diferentes módulos que habitualmente forman un sintetizador. Debido a la gran cantidad de aspectos tratados y al tiempo limitado para este tipo de trabajos, han quedado algunos aspectos susceptibles de completarse en un futuro, que se han ido detallando a lo largo del documento. Además, partiendo del sistema diseñado, se puede continuar su desarrollo, incorporando nuevas funcionalidades, y realizar una optimización de las capacidades técnicas del mismo.

Se han clasificado las futuras líneas de trabajo en tres grandes bloques, que se describen a continuación:

- Mejora de la calidad de la señal:
  - Implementación de técnicas de reducción del aliasing de los osciladores.
  - Control de la ganancia de los filtros provocada cuando se establecen niveles altos de resonancia.
  - Aumento de la frecuencia de trabajo a 96 kHz para mejorar la resolución del audio generado.
- Nuevas funcionalidades:
  - Aumento del número de osciladores para cada voz.
  - Implementación de la polifonía.
  - Utilización de la salida estéreo como dos salidas mono independientes, permitiendo así la generación simultánea de dos timbres diferentes.
  - Inclusión de un generador de ruido blanco y rosa en el bloque de osciladores.
  - Posibilidad de carga de ficheros de audio para su carga en un oscilador Wavetable.
  - Implementación de diferentes filtros (Moog, Korg, etc.).
  - Implementación de protocolo MIDI.
  - Inclusión de un secuenciador y un arpegiador internos.
- Aumento de las capacidades técnicas del sistema:
  - Utilización de técnicas de optimización de área para minimizar al máximo el uso de recursos del sistema.
  - Utilización de técnicas de *pipelining* para aprovechar al máximo la velocidad del chip FPGA, incrementando el número de señales procesadas de forma secuencial.
  - Implementación de un diseño de referencia propio que permita la utilización de todas las entradas y salidas de la placa.

6. Conclusiones

A lo largo del presente trabajo se ha abordado de forma satisfactoria el diseño de un sintetizador digital de audio sobre una arquitectura SoC con lógica programable, mediante técnicas de modelado visual y generación automática de código.

Durante este proceso se han adquirido una gran cantidad de conocimientos de diferentes campos, como por ejemplo: diseño de sistemas de audio digital, diseño de sistemas digitales basados en precisión finita o aspectos relacionados con la síntesis de audio. También se ha profundizado en el uso de la herramienta Simulink® y algunas otras herramientas de Mathworks®, como son Fixed-Point Designer™ y Stateflow®. Todo ello ha permitido el diseño del sistema de una forma totalmente visual, sin necesidad de llegar a escribir ninguna línea de código.

Se ha comprobado que el uso de este tipo de metodología y, en concreto, de la solución basada en productos de Mathworks, tiene algunas ventajas, entre las que cabe destacar las siguientes:

- Permite el diseño completamente visual del sistema mediante diagramas de bloques y generación automática del código.
- Evaluación automatizada del diseño en coma flotante para la conversión asistida a diseño de precisión finita.
- Generación de código automatizada para máquinas de estado, tanto de Mealy como de Moore.
- Disponibilidad de un enorme conjunto de herramientas del entorno Matlab®/Simulink® para la validación del sistema desarrollado.
- Integración transparente con herramienta de síntesis de Vivado®.
- Comparado con herramientas de modelado visual proporcionadas por los fabricantes de chips, que restringen su uso únicamente a sus chips, tiene la ventaja de que los modelos pueden ser generados para aquellos fabricantes para los que exista.

Sin embargo, la metodología planteada también presenta algunas restricciones que deben considerarse, como son:

- Sistema dependiente de Simulink® y los *plug-ins* utilizados para su diseño, con el correspondiente coste en licencias que ello conlleva.
- Limitación en los modelos de placas y fabricantes para los que se realiza la generación automática de código, estando disponible únicamente *plug-ins* para las principales marcas.
- La validación de los modelos en Simulink® requiere de unos elevados recursos *hardware*.
- El diseño de sistemas mediante modelado visual dificulta el seguimiento de cambios en los modelos.
- Escasa información cuando se producen errores durante la ejecución externa de los modelos generados.

La utilización de este tipo de herramientas simplifica enormemente el proceso de diseño del sistema, pero no proporciona una abstracción total del sistema sobre el que finalmente se va a implementar el diseño. Es más, requiere del conocimiento de las problemáticas habituales que

se presentan en el diseño de este tipo de sistemas para así poder identificar los problemas que surjan durante el desarrollo del sistema, puesto que no siempre son identificados por la herramienta, ni tampoco ésta optimiza de forma automática los diseños para evitar que algunos de estos problemas surjan. Aunque esto no supone un impedimento para empezar a trabajar con este tipo de metodología, sí puede llegar a serlo cuando se aborden problemas complejos. En este aspecto cabe destacar que, a lo largo de las diferentes versiones que se han ido publicando de las herramientas de generación automática de código, algunos de los problemas más habituales se han ido corrigiendo.

Sí que es cierto que es perfectamente viable el diseño e implementación de prototipos sin tener conocimiento alguno de los lenguajes utilizados para programar tanto los chips ARM (lenguaje de programación C) como la FPGA (Verilog/VHDL). A lo largo de todo el proyecto no ha sido necesaria la programación de ninguna línea de código para realizar el diseño y hacerlo funcionar en la placa de desarrollo, lo cual puede suponer un aspecto muy favorable de cara al uso de este tipo de herramientas. Sin embargo, el ahorro en la programación de código implica una inversión inicial de tiempo en el conocimiento del diseño de sistemas mediante Simulink® y de la multitud de opciones que presentan los *plug-ins* para la generación automática de código. Este último aspecto es determinante a la hora de afrontar los problemas que pueden ir surgiendo durante el desarrollo. Por tanto, dependiendo de las habilidades de la persona encargada del desarrollo del sistema, está metodología puede resultar más o menos útil o costosa.

En general, se puede considerar que el enfoque planteado es viable y puede resultar útil tanto a nivel educacional, para abordar las diferentes problemáticas vinculadas al diseño de sistemas basados en arquitecturas SoC sin necesidad de entrar en detalles de implementación de código, como para equipos de trabajo en los que pueda resultar útil el prototipado rápido de soluciones, aunque en este último caso deben tenerse en cuenta las limitaciones existentes relacionadas con los modelos de placas de desarrollo compatibles con las herramientas de generación automática de código, lo que puede llegar a dificultar el paso del prototipo a un sistema listo para su producción.

Se ha podido comprobar también cómo este tipo de arquitecturas SoC pueden resultar óptimas para el diseño de sistemas digitales de audio que requieran elevados niveles de paralelismo. Se han podido comprobar de primera mano las problemáticas habituales de los diseños de sistemas digitales en precisión finita y cómo afrontarlas para el diseño de sistemas de audio digital, comprobando que su uso no tiene por qué suponer una merma en la calidad de la señal, en comparación con los sistemas basados en coma flotante. Se ha comprobado la dificultad y el enorme conocimiento que se requiere para realizar diseños que cumplan con los estándares de calidad del mercado y se ha creado un sistema que supone un punto de partida para la futura aplicación de mejoras tanto en el procesado de la señal como en la maximización del uso de recursos ofrecidos por la placa de desarrollo, así como la creación de nuevos diseños de sintetizadores con esquemas de síntesis totalmente diferentes.

A partir del trabajo realizado se pretende seguir completando el diseño para que, además de profundizar en las diferentes técnicas relacionadas con la síntesis digital de audio y las arquitecturas basadas en SoCs, pueda servir como herramienta educacional para aquellas personas que deseen iniciarse en el diseño de sistemas de síntesis digital de audio o simplemente conocer las peculiaridades de estos sistemas.

7. Referencias

1. Roads, C. *The computer music tutorial*; MIT Press, 1996.
2. Zynq-7000 SoC. Available online: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html (accessed 17 Febrero 2019).
3. Architecture | AMBA 4 – Arm Developer. Available online: https://developer.arm.com/products/architecture/system-architectures/amba/amba-4 (accessed 17 Febrero 2019).
4. Chen, W. Performance of Cascade and Parallel IIR Filters. *J. Audio Eng. Soc.* **1996**, *44* (3), 148-158.
5. Llorente, D. Implementación de un procesador digital de audio basado en FPGA, Escuela Politécnica Superior de Gandia - UPV, 2013.
6. Avnet. ZedBoard Hardware User's Guide 2.2. Available online: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.
7. Analog Devices. ADAU1761 Datasheet and Product Info | Analog Devices. Available online: https://www.analog.com/en/products/adau1761.html# (accessed 18 Febrero 2019).
8. Authoring a reference design for audio system on a Zynq board - MathWorks Benelux. Available online: https://nl.mathworks.com/examples/matlab-hdl-coder/mw/hdlcoder_product-hdlcoder_ref_design_audio-authoring-a-reference-design-for-audio-system-on-a-zynq-board (accessed 13 Febrero 2019).
9. Getting Started with Hardware-Software Co-Design Workflow for Xilinx Zynq Platform - MATLAB & Simulink - MathWorks España. Available online: https://es.mathworks.com/help/hdlcoder/examples/getting-started-with-hardware-software-codesign-workflow-for-xilinx-zynq-platform.html (accessed 18 Enero 2019).
10. Nyquist, H. Certain topics in telegraph transmission theory - Proceedings of the IEEE - Nyquist.pdf. *Trans. A. I. E. E* **1928**, *47*, 617-644.
11. Shannon, C. E. Communication in the Presence of Noise. *Proc. IRE* **1949**, *37* (1), 10-21. https://doi.org/10.1109/JRPROC.1949.232969.
12. Tierney, J.; Rader, C.; Gold, B. A digital frequency synthesizer. *IEEE Trans. Audio Electroacoust.* **1971**, *19* (1), 48-57. https://doi.org/10.1109/TAU.1971.1162151.
13. Stilson, T.; Smith, J. Alias-Free Digital Synthesis of Classic Analog Waveforms Why Simple Discrete-Time Pulse Trains are Aliased. *Int. Comput. Music Conf. Proc.* **1996**, *1996*, 332-335. https://doi.org/10.1144/GSL.QJEG.1978.011.02.04.
14. Pirkle, W. *Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units*; Focal Press, 2014; Vol. 3.
15. Cordesses, L. Direct Digital Synthesis: A Tool for Periodic Wave Generation (Part 1). *IEEE Signal Process. Mag.* **2004**, N.º JULY, 50-54. https://doi.org/10.1109/MSP.2004.1311140.
16. Cordesses, L. Direct Digital Synthesis: A Tool for Periodic Wave Generation (Part 2). *IEEE Signal Process. Mag.* **2004**, N.º September, 110-117. https://doi.org/10.1109/MSP.2004.1328096.

17. Valls, J. Síntesis digital directa y mezclado. En *Apuntes de la asignatura Implementación de Sistemas de Comunicaciones*; 2018.

18. Stilson, T.; Smith, J. Analyzing the moog vcf for digital implementation. *Proc. Int. Comput. Music Conf.* **1996**, 1-11.

19. Kerwin, W. J.; Huelsman, L. P.; Newcomb, R. W. State-variable synthesis for intensive integrated-circuit transfer functions. *IEEE J. Solid-State Circuits* **1967**, *SC-2* (3), 87-92. https://doi.org/10.1109/JSSC.1967.1049825.

20. Chamberlin, H. *Musical Applications of Microprocessors*, Second Ed.; Hayden Books, 1985.

21. Zölzer, U.; Amatriain, X. *DAFX : digital audio effects*; Wiley, 2002.

22. Wise, D. K. The Modified Chamberlin and Zölzer Filter Structures. *Audio* **2006**, N.º 3, 53-56.

23. Frei, B. Digital Sound Generation - Part 2. Available online: https://www.zhdk.ch/file/live/33/33792f805231039a5d7dc0115b943af952ff3c4f/digital_sound_generation_2.pdf (accessed 8 Febrero 2019).

24. Xilinx Inc. Zynq-7000 SoC Data Sheet: Overview. 2018, pp 1-21.