



Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.



Trabajo Fin de Grado en Ingeniería Electrónica Industrial.

Procesado de señales de audio mediante FPGAs

Autor: Emilio Cano García

Tutores: Carlos Jesús Jiménez Fernández

Manuel Valencia Barrero

Departamento de Tecnología Electrónica

Septiembre 2018



Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.



RESUMEN

Procesado de señales de audio mediante FPGAs

Autor: Emilio Cano García

Tutores: Carlos Jesús Jiménez Fernández

Manuel Valencia Barrero

En este proyecto se ha implementado un sistema de procesamiento de señales de audio en una FPGA, con el fin de crear diferentes efectos de audio digitales controlados con una placa de desarrollo.

Se ha realizado un estudio sobre el procesamiento de señal digital, incluyendo las principales técnicas y herramientas para el diseño de filtros digitales. Posteriormente, se han definido estructuras basadas en sistemas retardadores mediante las cuales se consiguen distintos efectos de audio.

Mediante la herramienta Simulink se ha simulado el resultado de cada efecto y se han configurado sus parámetros de control en función de la salida de audio obtenida. Con Matlab se ha realizado un breve estudio de la utilidad de generar código HDL a partir de un modelo Simulink, para luego implementarlo en un dispositivo lógico programable.

Para optimizar el sistema resultante, se ha preferido generar manualmente los códigos VHDL para los sistemas modelados en Simulink. Estos códigos han sido sintetizados e implementados en la FPGA Artix-7 XC7A100T con placa de desarrollo Nexys4 DDR.



Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.





Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.



ABSTRACT

Procesado de señales de audio mediante FPGAs

Autor: Emilio Cano García

Tutores: Carlos Jesús Jiménez Fernández

Manuel Valencia Barrero

In this Project, an audio signal processing System has been implemented in a FPGA, in order to create different digital audio effects controlled with a development board.

A study on digital signal processing including the main techniques and tools for digital filter design has been carried out. Later, structures based on delayed systems have been defined by which different audio effects are achieved.

Using Simulink tool the result of each has been simulated and its control parameters have been configured according to the audio output. With Matlab, a brief study has been made of the utility of generating HDL code from a Simulink model, to then implement it in a programmable logic device.

To optimize the resulting System, it has been preferred to manually enter the VHDL codes to for System modelled in Simulink. These codes have been synthesized and implemented in Artix-7 XC7A100T FPGA with Nexys4 DDR development board.



Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.



Trabajo Fin de Grado en Ingeniería Electrónica Industrial.

Procesado de señales de audio mediante FPGAs

Memoria Descriptiva

Autor: Emilio Cano García

Tutores: Carlos Jesús Jiménez Fernández

Manuel Valencia Barrero



Departamento de Tecnología Electrónica

Septiembre 2018





Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.





			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página I

Índice



1. INTRODUCCIÓN.....	1
2. PROCESADO DIGITAL DE SEÑALES	3
2.1. Introducción	3
2.2. Técnicas de DSP en sistemas LTI.....	7
2.2.1. Respuesta impulsiva $h(n)$. Convolución	8
2.2.2. Respuesta en frecuencia. DFT.....	8
2.2.3. Función de transferencia en el dominio Z. Transformada Z	9
2.3. Filtros digitales.....	10
3. HERRAMIENTAS Y TECNOLOGÍAS EMPLEADAS	13
3.1. VHDL	13
3.1.1. Unidades de diseño	14
3.1.1.1. Entidad.....	14
3.1.1.2. Arquitectura.....	15
3.1.2. Objetos.....	15
3.1.3. Operadores	17
3.1.4. Sentencias concurrentes.....	17
3.1.5. Sentencias secuenciales.....	17
3.1.6. Declaración de componentes.....	18
3.2. Dispositivos lógicos programables.....	20
3.2.1. SPLDs	20
3.2.2. CPLDs.....	20
3.2.3. FPGAs.....	20
3.2.3.1. Estructura interna.....	21
3.2.3.2. Fusible/anti fusible:.....	22
3.2.3.3. Celda de memoria RAM	22
3.3. Matlab.....	24
3.3.1. Simulink.....	24
4. PROCESADO DE AUDIO	27

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página II

4.1.	Introducción	27
4.2.	Efectos de sonido basados en retardadores.....	28
4.2.1.	Sistema retardador basado en filtros peine FIR	28
4.2.2.	Sistema retardador basado en filtros de peine IIR	32
4.2.3.	Filtro reverberador.....	33
4.2.4.	Filtros IIR paso todo	33
4.2.5.	Retardos modulados en el tiempo.....	34
4.2.6.	Estructura paso todo generalizada.....	34
4.3.	Implementación de filtros mediante Simulink.....	36
4.3.1.	Configuración del modelo.....	36
4.3.2.	Bloques utilizados	38
4.3.3.	Diseño de filtros en Simulink	40
4.3.3.1.	Modelo para efecto Eco, Slapback y Doubling.....	41
4.3.3.2.	Modelo efecto vibrato	43
4.3.3.3.	Modelo efecto flanger.....	44
4.3.3.4.	Modelo efecto chorus	46
4.4.	Flujo de diseño. Simulink-VHDL	48
4.4.1.	System Generator	50
5.	DISEÑO DE CIRCUITOS	55
5.1.	Interfaz del convertidor analógico digital PmodAD1	55
5.2.	Interfaz del convertidor analógico digital PmodI ² S.....	59
5.3.	Generación de relojes del sistema.....	62
5.4.	Sistema DSP	65
5.5.	Bloques de almacenamiento de muestras.....	67
5.5.1.	RAM distribuida.....	69
5.5.2.	Bloque de RAM (BRAM)	71
5.5.3.	Métodos para incorporar módulos de memoria al diseño	72
5.5.3.1.	Mediante instanciación HDL.....	72
5.5.3.2.	Mediante un IP Core Generator	73
5.5.4.	Proceso de control de lectura y escritura en memoria	80

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página III

5.5.5.	Consumo de recursos en función del tipo de RAM escogida .	81
5.6.	Módulos de interacción con la placa de desarrollo	83
5.7.	Construcción de filtros	85
5.7.1.	Selección de filtro mediante el módulo select_filter.....	87
5.7.2.	Tiempos de retardo y flujo de datos del filtro echo	88
5.7.3.	Tiempos de retardo y flujo de datos del filtro vibrato.....	88
5.7.4.	Tiempos de retardo y flujo de datos del filtro flanger.....	91
6.	SISTEMA FÍSICO	93
6.1.	Conversión A/D.....	93
6.2.	Conversión D/A.....	93
6.3.	Construcción del sistema.....	94
6.4.	Acondicionamiento de la señal de entrada	95
6.5.	FPGA y placa de desarrollo	98
7.	CONCLUSIONES	103
8.	REFERENCIAS	105

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página IV

Índice de Figuras

Figura 1: Representación de una señal analógica.	3
Figura 2: Representación de una señal digital.	3
Figura 3: Representación de un sistema digital.	4
Figura 4: Estructura general de un sistema de procesamiento digital de señal. ...	6
Figura 5: Diagrama de polos y ceros en el plano Z.	9
Figura 6: Ejemplo diagrama de bloques forma directa tipo I.	10
Figura 7: Estructura interna de una FPGA.	21
Figura 8: Diagrama funcional para creación de efectos digitales de audio ..	27
Figura 9: Filtro peine FIR.	29
Figura 10: Respuesta en frecuencia filtro FIR.	29
Figura 11: Filtro peine FIR con retardo variable.	31
Figura 12: Filtro peine FIR sin realimentación.	31
Figura 13: Filtro peine IIR.	32
Figura 14: Filtro reverberación simple.	33
Figura 15: Filtro IIR paso todo.	34
Figura 16: Estructura filtro paso todo generalizada.	35
Figura 17: Configuración de los parámetros de simulación del modelo	37
Figura 18: Flujo de datos sample-based y frame-based de Matlab.	38
Figura 19: Bloque From Multimedia File.	38
Figura 20: Buffer del bloque Audio Device.	39
Figura 21: Bloque To Audio Device y To Multimedia File.	39
Figura 22: Bloques: Constant, Display, Scope, Time Scope.	40
Figura 23: Bloque Manual Switch.	40
Figura 24: Sistema Genérico DSP.	41
Figura 25: Modelo Simulink para el efecto eco.	42
Figura 26: Modelo Simulink para el efecto vibrato.	44
Figura 27: Modelo Simulink para el efecto flanger.	45
Figura 28: Filtro paso bajo de primer orden.	45
Figura 29: Diagrama Simulink efecto flanger con LPF de primer orden.	46
Figura 30: Configuración bloque Lowpass Filter.	47
Figura 31: Diagrama Simulink para el efecto chorus.	48
Figura 32: Configuración generación de código HDL.	49
Figura 33: Bloque System Generator.	51
Figura 34: Ventana de configuración System Generator.	51
Figura 35: Configuración del reloj System Generator.	52
Figura 36: Bloques gateways.	52
Figura 37: Modelo System Generator con RAM de doble puerto.	53
Figura 38: Diagrama del circuito PmodAD1.	55



			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página V

Figura 39: Descripción de los pines del PmodAD1	56
Figura 40: Máquina de estados del proceso adc_state_control (PmodAD1)	58
Figura 41: Simulación PmodAD1	59
Figura 42: MCLK/LRCK ratio.....	60
Figura 43: Frecuencia de muestreo y frecuencia MCLK	60
Figura 44: Diagrama temporal del PmodI ² S.....	61
Figura 45: Simulación PmodI ² S	62
Figura 46: Ventana IP Core Clocking Wizard.....	64
Figura 47: Máquina de estados del proceso sampler_fsm (sistema_dsp)....	67
Figura 48: Características por tipo de dispositivo (Artix-7 FPGA)	68
Figura 49: Recursos del CLB (Artix-7 FPGA).....	69
Figura 50: Slices de un CLB.....	69
Figura 51: Logic Resources In One CLB.....	70
Figura 52: Ejemplo de RAM distribuida de un solo puerto (RAM64X1S)	70
Figura 53: Ejemplo de RAM distribuida de doble puerto (RAM64X1D)	71
Figura 54: Bloque de RAM de 36Kb (RAMB36)	72
Figura 55: Templates para Memorias RAM.....	73
Figura 56: Directorio Block and Distributes Memory Generator	74
Figura 57: Distributed Memory Generator	74
Figura 58: Single-Port, Dual-Port Distributed RAM Primitives.....	75
Figura 59: Características temporales memoria RAM Distribuida.....	76
Figura 60: Block Memory Generator	78
Figura 61: Modo WRITE_FIRST	79
Figura 62: Modo READ_FIRST	80
Figura 63: Modo NO_CHANGE	80
Figura 64: Ejemplo de escritura y lectura en memoria	81
Figura 65: Consumo de recursos de una RAM distribuida de 192Kb.....	82
Figura 66: Consumo de recursos de una memoria BRAM de 192Kb.....	82
Figura 67: Esquema de pines del display.....	84
Figura 68: Diagrama temporal del periodo de refresco del display	84
Figura 69: Onda senoidal positiva de amplitud 20 de 50 muestras.	90
Figura 70: PmodAD1	93
Figura 71: DAC CS4344 Figura 72: PmodI ² S.....	94
Figura 73: Construcción del sistema	94
Figura 74: Esquemático del circuito de acondicionamiento de señal.	95
Figura 75: Respuesta del circuito de acondicionamiento de señal.....	96
Figura 76: Operacional TL072CP.....	96
Figura 77: Configuración de pines TL072CP.	97
Figura 78: Esquemático circuito acondicionamiento de señal.....	97
Figura 79: PCB Circuito de acondicionamiento de señal.	98







			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página VI

Figura 80: Cable conexión PmodAD1.	98
Figura 81: Ordering Information	99
Figura 82: FPGA ARTIX-7 XC7A100T-1CGS324C.....	99
Figura 83: Placa de desarrollo Nexys4 DDR.....	101

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página VII

Índice de Tablas

Tabla 1: Parámetros de configuración por tipo de efecto	35
Tabla 2: Delays en función del efecto del filtro eco	88
Tabla 3: Pintado en función del efecto del filtro eco	88
Tabla 4: Rango de entrada analógica AD7476A.	95
Tabla 5: Periféricos de la placa de desarrollo Nexys4 DDR	101

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 1 de 120

1.INTRODUCCIÓN

Para procesar señales de audio, actualmente el mecanismo fundamental es el procesamiento digital de señales. Esto se debe a la mejora que ha tenido la tecnología de circuitos integrados, que ha permitido el desarrollo de circuitos, con más capacidad de procesamiento y softwares capaces de realizar simulaciones de circuitos complejos.

En el presente trabajo se introduce la teoría básica del procesamiento digital de señales, exponiendo diferentes herramientas y técnicas ampliamente utilizadas para este propósito. Estas nociones básicas se aplican en estructuras con las que se consiguen diferentes efectos auditivos.



Para implementar estos efectos se ha utilizado el software Simulink, con el que se han realizado simulaciones en tiempo real e interactivas de cada estructura, mediante los periféricos del PC. De esta forma, se ha comprobado la funcionalidad del filtro y se han elegido los parámetros de configuración para conseguir diferentes efectos sobre señales de audio.

La finalidad de este proyecto es, además de hacer una breve introducción del procesamiento digital de audio y de diferentes efectos auditivos, implementar en un dispositivo lógico, en este caso una FPGA, un procesamiento digital de señales de audio con las estructuras definidas en Simulink. Para ello se explican diferentes métodos para pasar código Matlab a código VHDL, eligiendo al final uno de ellos.

En la implementación del sistema digital en la FPGA, se diseña una estructura genérica formada por diferentes módulos que realizan una función específica, como por ejemplo, la interfaz con los convertidores, la placa de desarrollo y el control de lectura y escritura en memoria.

Los filtros implementados están basados en acumuladores, es decir, necesitan memorizar muestras. Para implementarlo en la FPGA se exponen los diferentes tipos de almacenamiento de este dispositivo lógico, y se realiza un estudio para llegar a la conclusión de cuál de ellos es más eficiente para la estructura implementada.

Una vez definido la estructura del procesamiento digital de audio en la FPGA y los módulos que los implementan, mediante el software de desarrollo ISE Design proporcionado por el fabricante Xilinx, se introduce, verifica e implementa el código VHDL de cada módulo del sistema.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 2 de 120



Mediante el software se realizan simulaciones temporales de cada uno de los módulos y del sistema final mediante test bench, en los cuales se controlan los parámetros de entrada del circuito, para de esta forma realizar comprobaciones del funcionamiento del sistema.

Para cada efecto se ha creado una implementación diferente en la FPGA, para realizar pruebas más fácilmente del funcionamiento de cada uno de ellos. Una vez depurado el código para cada efecto, se han implementado todos los efectos en un mismo sistema genérico, para poder así realizar todos los efectos en un mismo dispositivo.

Para que el usuario seleccione cada efecto y modifique sus parámetros de control, se programa una interfaz con los periféricos de la placa de desarrollo que implementa la FPGA, pudiendo de esta forma el usuario tener una percepción auditiva y visual de cada efecto.

Durante el diseño de los módulos, se han realizado pruebas en laboratorio con una señal de audio de un PC. Para realizar estos ensayos, se tuvo que crear una placa de circuito impreso con un acondicionador de señal, para adaptar la señal de audio del PC a los requisitos de la entrada del convertidor analógico/digital del sistema. Esta PCB se ha diseñado con el software DesignSpark y se ha construido en laboratorio.

Como resultado de este trabajo, se ha proporcionado el conocimiento para diseñar en una herramienta software, estructuras que implementen efectos de audio y diferentes métodos para implementar esos diseños en un dispositivo, en este caso una FPGA. Posteriormente se han implementado esos diseños en la FPGA Artix7 de Xilinx y se han explicado los diferentes módulos de programación para implementar un procesamiento digital genérico, para luego implementar todos los efectos de audio seleccionados.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 3 de 120

2.PROCESADO DIGITAL DE SEÑALES

2.1. Introducción

El procesamiento digital de señal (DSP, *Digital Signal Processing*) es la disciplina que estudia y desarrolla técnicas de tratamiento de señales representadas digitalmente, basándose en la estadística y en la matemática aplicada. Una de sus aplicaciones es el procesamiento de señales de audio.

Una señal de audio es de origen analógico, tanto en su generación como a la hora de ser reproducida. Una señal es una función que transporta información. Matemáticamente se representan como funciones de una o más variables independientes, en el caso del audio, por el tiempo. En este tipo de señales tanto la variable dependiente como independiente son continuas en el dominio del tiempo, es decir, presentan un valor para cada instante de tiempo y reciben el nombre de **señales analógicas** (Figura 1).

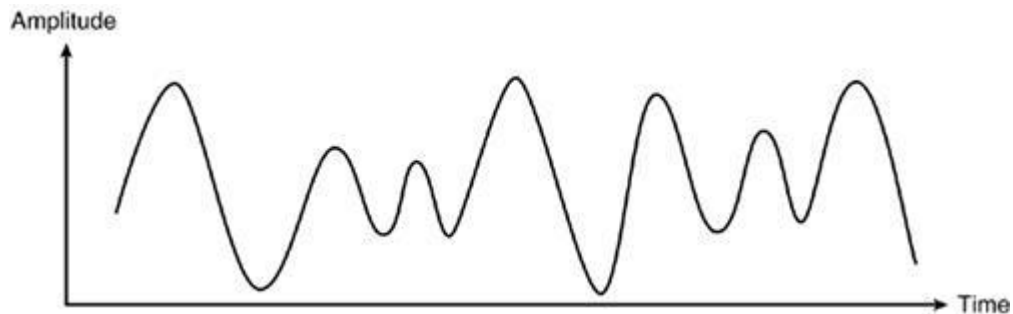


Figura 1: Representación de una señal analógica.

También existen las **señales digitales** en las cuales la variable dependiente y/o independiente son discretas, es decir, tiene un número finito de valores (Figura 2).

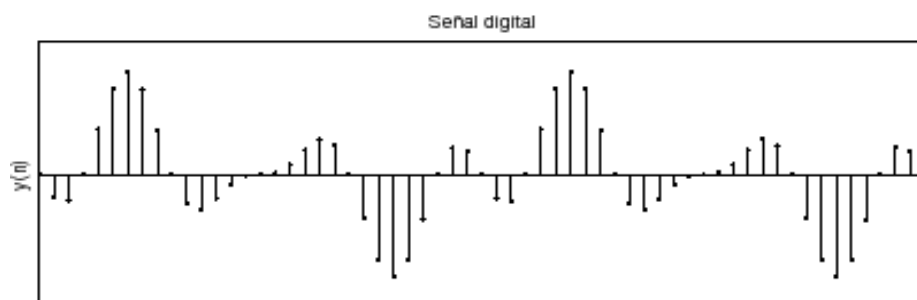




Figura 2: Representación de una señal digital.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 4 de 120

Una vez definidos los tipos de señales, se detalla qué es un **sistema**. Un sistema es una entidad que tiene unas señales como entrada, las cuales son modificadas o procesadas y da como resultado una señal de salida. Si el sistema tiene como entrada y salida señales digitales, se trata de un **sistema digital** (Figura 3).



Figura 3: Representación de un sistema digital.

Un sistema puede tener diferentes características dependiendo de la función que realice. Entre estas características destacan:

- Estabilidad BIBO: un sistema es BIBO-estable (*Bounded-Input Bounded-Output*) si y solo sí a una función de entrada acotada, le corresponde una salida acotada. Una función está acotada cuando existe un número M finito de salidas, es decir:

$$|x(n)| \leq M < \infty$$

- Memoria: esta propiedad la adquieren los sistemas que dependen de valores pasados de la señal de entrada. La “profundidad” o el tamaño de la memoria es el número de estados que tienen que ser almacenados. Por ejemplo, la siguiente función que define a un sistema, tiene esta propiedad y necesita almacenar dos estados:



$$y(n) = x(n) + x(n - 2)$$

- Causalidad: un sistema es causal cuando su salida depende solo de los valores presentes y/o pasados de la señal de entrada.

$$y(n) = x(n) + 2x(n - 3)$$

- Recursividad: sistema en el cual su salida depende de los valores y/o de la señal de entrada y de los valores pasados de la señal de salida.

$$y(n) = x(n) + 2x(n - 3) + y(n - 2)$$

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 5 de 120

- Invertibilidad: esta propiedad se cumple cuando la entrada de un sistema se puede recuperar a partir de la salida.

$$y(n) = H[x(n)]$$

$$x(n) = H^{-1}[y(n)]$$

- Invarianza en el tiempo: un sistema cumple con esta propiedad, si un retraso o adelanto en la señal de entrada produce un retraso o adelanto idéntico en la señal de salida.

$$y(n) = H[x(n)]$$

$$y(n - N) = H[x(n - N)]$$

- Linealidad: esta propiedad se cumple si el sistema satisface el principio de superposición. Es decir:

$$\text{Si } y_1(n) = H[x_1(n)] ; y_2(n) = H[x_2(n)] ; x(n) = ax_1(n) + bx_2(n)$$

$$y(n) = H[x(n)] = ay_1(n) + by_2(n)$$



Los sistemas que poseen las propiedades de linealidad e invarianza en el tiempo, se denominan LTI (*Linear Time-Invariant*). Estos sistemas son muy útiles en el procesamiento de señales digitales debido a que pueden ser caracterizados por su respuesta al impulso.

Si se quiere un sistema digital que tenga como entradas señales analógicas, éstas se tendrán que convertir en digitales antes de entrar en el sistema, y de nuevo convertirlas en analógicas a su salida, para que no pierda su naturaleza. Para la conversión analógico digital son necesarias el muestreo y la cuantización de las señales analógicas mediante convertidores analógicos/digitales, tomándose valores a una frecuencia de muestreo determinada.

En este proceso de conversión, no se pierde información de la señal, si se captan las muestras suficientes. Para ello, el teorema de Nyquist establece la limitación en la que una señal está unívocamente representada, si la frecuencia de muestreo es al menos el doble la frecuencia máxima de la señal.

$$f_s > 2f_M$$

A la salida del procesamiento digital, la señal digital debe convertirse en analógica mediante convertidores digitales/analógicos. Con esta estructura queda definido un sistema genérico de procesamiento digital, como se muestra en la figura 4.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 6 de 120

Esta estructura puede tener variaciones en función de la aplicación, por ejemplo, en el caso de que la señal ya hubiese sido almacenada, o si no se desea reproducir la señal de salida porque va a ser almacenada, lo que implicaría que los convertidores de señal no fuesen necesarios.

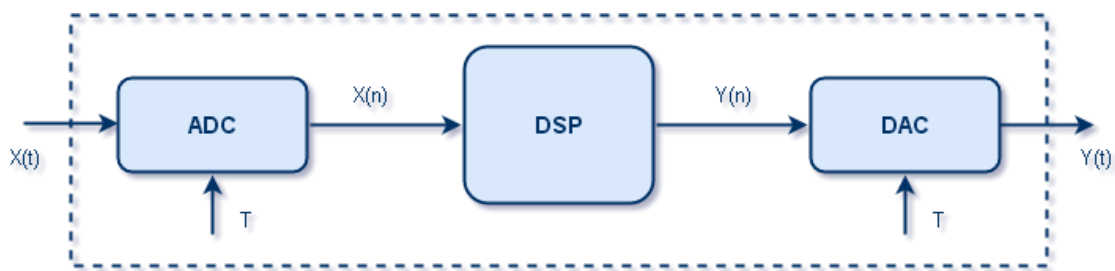




Figura 4: Estructura general de un sistema de procesamiento digital de señal.

Al igual que existe el procesamiento digital, también existe el procesamiento analógico, el cual trabaja con señales analógicas. En función de los requisitos del sistema, es más conveniente utilizar un procesamiento digital o analógico. Principalmente las ventajas de utilizar uno u otro son las siguientes:

- Mediante un procesamiento digital se pueden diseñar procesos y algoritmos más complejos que en el procesamiento analógico. La razón principal es que en los procesos analógicos las señales se van degradando conforme se van procesando, deteriorando la señal y empeorando la relación señal ruido. Esto no ocurre con las señales digitales, ya que estas conservan la relación señal ruido, pudiendo además acotar los errores de redondeo, es decir, mediante señales digitales se pueden realizar procesos complejos sin degradar la señal.
- Un proceso digital puede ser reprogramado, se puede decir que es flexible, ya que es una función que depende de unas variables y estas pueden ser modificadas. Un proceso analógico es único, por ejemplo, un filtrado o una amplificación, los cuales dependen de la estructura del circuito y de los componentes utilizados en el mismo.
- En los procesos digitales, las variables de configuración del proceso están almacenadas en una memoria, que permite conservarlas independientemente del tiempo y de las condiciones ambientales. Sin embargo, en los procesos analógicos las variables de configuración lo forman los dispositivos del circuito, los cuales son sensibles a las condiciones ambientales y pueden variar en el tiempo.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 7 de 120

A pesar de estas ventajas, el procesamiento analógico sigue siendo efectivo para funciones más sencillas en las que se requiere rapidez y bajo consumo en el proceso.

El procesamiento digital está incrementando año tras año su capacidad debido al crecimiento y la mejora de componentes, ordenadores y sistemas empujados que incorporan procesadores digitales más rápidos con un coste cada vez menor.

2.2. Técnicas de DSP en sistemas LTI

Las principales técnicas en el procesamiento de señales digitales se basan en el principio de superposición, es decir, en la descomposición de señales.

Toda señal puede descomponerse en sumas de otras señales más elementales como por ejemplo, sumas de señales impulso, de señales escalón, señales senoidales.



$$x(n) = x_1 + x_2 + x_3 + \dots$$

Si dicha señal descompuesta, se aplicara en un sistema lineal e invariante en el tiempo LTI, la señal resultante sería la suma de las respuestas a cada una de las señales que la componen.

$$y(n) = H[x(n)] = H[x_1(n)] + H[x_2(n)] + H[x_3(n)] + \dots$$

En función del tipo de análisis que se realice, se optará por descomponer la señal con un tipo de señal elemental u otra. Para un análisis temporal, la señal se descompone en señales impulso o escalón y se utiliza la operación de convolución. Si por el contrario, se quiere realizar un análisis frecuencial, la señal se descompone en señales senoidales y se utilizan las transformadas de Fourier. Otras formas de representar un sistema son mediante ecuación de diferencias, que expresen la relación de la entrada con la salida, y la función de transferencia en el dominio Z.

Por lo tanto un sistema LTI se puede representar de diversas formas en función del análisis que se quiera y de la herramienta utilizada.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 8 de 120

2.2.1. Respuesta impulsiva $h(n)$. Convolución

Partiendo de que una señal se puede descomponer en sumas de impulsos y de las propiedades de un sistema LTI, se puede demostrar la denominada respuesta impulsiva del sistema $h(n)$:

$$h(n) = H[\delta(n)]$$

Conociendo la entrada del sistema $x(n)$ y la respuesta impulsiva $h(n)$, se puede obtener la salida de cualquier sistema LTI mediante la operación de convolución:

$$y(n) = \sum_{k=-\infty}^{\infty} h_{(k)}x(n - k)$$

Esta operación solo es aplicable para los sistemas donde la respuesta impulsiva se hace cero tras un número finito de muestras, es decir, cuando hablamos de sistemas FIR (*Finite Impulse Response*). Por lo tanto, la respuesta de sistema FIR se puede obtener mediante la operación de convolución:



$$y(n) = \sum_{k=0}^M h_{(k)}x(n - k)$$

2.2.2. Respuesta en frecuencia. DFT

La Transformada de Fourier (FT) es la operación matemática capaz de transformar una señal en el dominio temporal al dominio de la frecuencia. La transformación, da lugar a una función continua y periódica de periodo 2π . La función resultante continua, se puede representar de forma discreta aplicando otra operación, denominada Transformada Discreta de Fourier (DFT), la cual obtiene una secuencia de muestras discretas de la transformada de Fourier. Las expresiones en forma compleja de la DFT y de la inversa, se calculan a partir de una serie discreta de Fourier. Las expresiones son las siguientes, siendo N el número de muestras tomadas para realizar el cálculo, el cual equivale al periodo de la DFT:

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N} n} \quad 0 \leq k \leq N - 1$$

$$x[n] = \sum_{k=0}^{N-1} X[k] e^{-jk \frac{2\pi}{N} n} \quad 0 \leq k \leq N - 1$$

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 9 de 120

2.2.3. Función de transferencia en el dominio Z. Transformada Z

La transformada Z es una herramienta muy útil para analizar sistemas y señales en el dominio discreto, se asemeja a la transformada de Laplace ya que ésta es igualmente útil pero en el análisis de sistemas continuos en el tiempo. Tanto la transformada de Laplace como la transformada Z, están relacionadas con la transformada de Fourier, y ambas permiten obtener la función de transferencia $H(z)$ o $H(s)$ de un sistema definido por su respuesta impulsiva. La expresión de la transformada Z de una señal discreta $x(n)$, es la siguiente:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}; z = e^{j\omega}$$

Se puede observar como esta expresión equivale a la transformada de Fourier si sustituimos la variable z por su valor complejo:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

La mayor utilidad de utilizar la transformada Z se debe a que si se aplica en la señal de entrada y salida de un sistema discreto, se puede obtener la función de transferencia del sistema $H(z)$, mediante la siguiente expresión:

$$H(z) = \frac{Y(z)}{X(z)}$$

La transformada Z se puede representar mediante el plano Z, el cual es una circunferencia unidad donde se representan los polos y los ceros del sistema (Figura 5). El ángulo entre un punto dentro del plano y el eje real es igual a la pulsación discreta ω (rad).

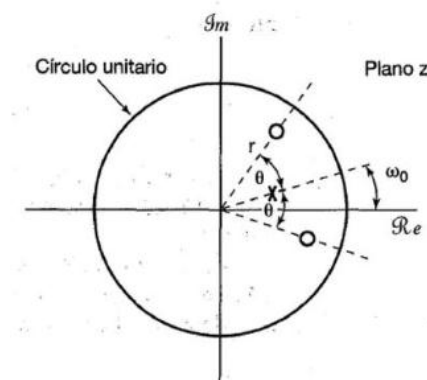




Figura 5: Diagrama de polos y ceros en el plano Z.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 10 de 120

2.3. Filtros digitales

Un filtro digital es un sistema que modifica una señal digital de entrada mediante una serie de operaciones y técnicas y obtiene una señal de salida con unas características determinadas. Los filtros digitales al ser sistemas LTI, pueden expresarse utilizando la respuesta impulsiva $H(n)$, la respuesta en frecuencia $H(j\omega)$ y la función de transferencia $z H(z)$.

Se pueden clasificar en función de su respuesta impulsiva, en filtros FIR (*Finite Impulse Response*) y filtros IIR (*Infinite Impulse Response*). Los primeros tienen una respuesta impulsional finita y los segundos una respuesta impulsional infinita.

Los filtros digitales se pueden representar mediante diagramas de bloques, los cuales contienen las operaciones elementales de multiplicación, adición y memorización de datos (retraso de muestra) y así representar la función de transferencia mediante ecuaciones de diferencias.

$$y[n] = \sum_{k=1}^N a_k y[n - K] + \sum_{k=0}^M b_k x[n - k]$$

En la figura 6 se puede visualizar un ejemplo de representación de un filtro mediante diagrama de bloques:

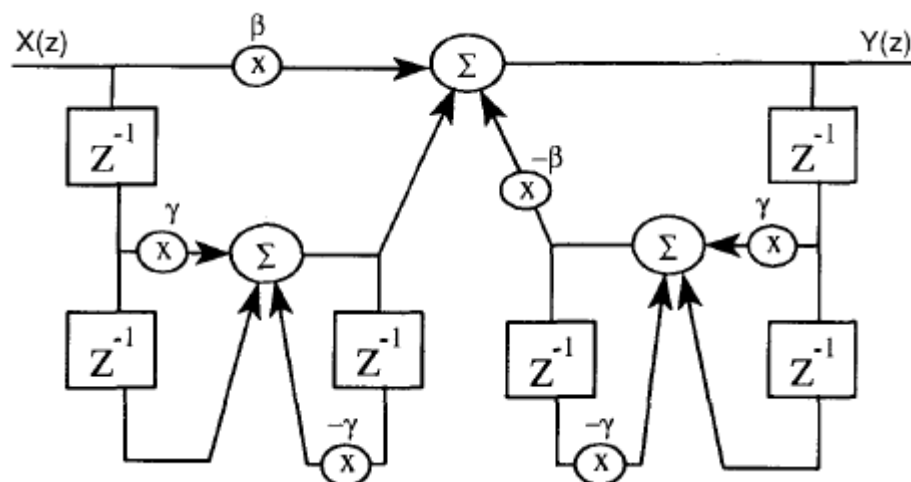




Figura 6: Ejemplo diagrama de bloques forma directa tipo I.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 11 de 120



En función del tipo de filtro que se desee diseñar, se tendrán que utilizar diferentes técnicas. Uno de los métodos de diseño para filtros IIR, se basa en aplicar a filtros analógicos transformaciones que los conviertan en digitales con las mismas características. Una de las transformaciones más utilizadas es la transformación bilineal, que consiste en obtener la función de transferencia del filtro analógico $H(s)$, para lograr así la función de transferencia $H(z)$ del filtro digital. Para el diseño de los filtros IIR se utiliza principalmente esta transformada. Los filtros IIR tienen una respuesta impulsional infinita y la señal de salida filtrada depende de valores pasados de la entrada y de la salida, es decir, la respuesta en frecuencia no es de fase lineal debido a que no se cumplen las condiciones de simetría. La ventaja de utilizar este tipo de filtro es que pueden llegar a tener características de atenuación y frecuencias de paso o atenuadas con un orden inferior al orden que requiere un filtro FIR.



Los filtros FIR se caracterizan por tener una respuesta impulsional finita y por conseguir una fase lineal en la banda paso, lo cual evita distorsiones de fase que provoca efectos audibles en la señal procesada. Estas características lo hacen muy útiles en aplicaciones de audio. La característica de que la fase sea lineal hace que el filtro no tenga distorsiones de fase a la salida en la banda de paso. Esta propiedad es uno de los principales motivos para utilizar estos filtros.

Para el diseño de estos filtros se utilizan diferentes métodos. Uno de ellos es el método de ventanas. Este método consiste en realizar la operación de convolución a la respuesta impulsional del filtro ideal $h(n)$ en M puntos, con una función que actúe como ventana $w(n)$.

$$h(n) \cdot w(n)$$

Dependiendo del resultado y de las características que se quieran tener del filtro, se aplicarán diferentes tipos de ventanas. Los resultados que se obtienen al aplicar una ventana, son disminuir el rizado de la banda de paso y de stop, y aumentar el tamaño de la banda de transición.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 12 de 120

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 13 de 120

3. HERRAMIENTAS Y TECNOLOGÍAS EMPLEADAS

Esta sección define las herramientas que se han utilizado en el trabajo para el diseño de filtros digitales y las tecnologías con las que se han implementado.

3.1. VHDL



VHDL es un lenguaje de descripción de hardware que surge a principios de los '80 de un proyecto DARPA (Departamento de Defensa de los EE.UU.) llamado VHSIC (*Very High Speed Integrated Circuits*) cuyo fin era crear un HDL (*Hardware Description Language*) estándar. También es el acrónimo de la combinación de VHSIC y HDL.

El proyecto fue cedido al IEEE (*Institute of Electrical and Electronics Engineers*) en 1987, y a partir de ese momento es un estándar abierto. En 1993 el lenguaje VHDL fue revisado y ampliado pasando a ser estándar (ANSI/IEEE 1076-1993).

VHDL permite describir sistemas digitales. A partir de estas descripciones se pueden realizar **simulaciones** para comprobar que la funcionalidad del diseño es el deseado y **sintetizarlo**, es decir, crear un circuito que funcione como el modelo en un PLD (*Programmable Logic Device*), FPGA (*Field-Programmable Gate Array*) o ASIC (*Application-Specific Integrated Circuit*).

Otra forma de diseñar circuitos digitales es crear un esquemático e implementarlo con circuitos discretos. Esta forma de diseño no es eficiente si se aumenta la complejidad del circuito, ya que la introducción del diseño no es tan sencilla y la probabilidad de errores aumenta. Estos errores deben ser encontrados antes de la introducción del diseño, por lo que debemos tener puntos de verificación que garantice un correcto funcionamiento del circuito.

Con VHDL surge la necesidad de nuevas herramientas y nuevas metodologías de diseño que proporcionen mecanismos de descripción (más productivos y seguros), que permitan verificar el diseño mediante herramientas software y que permitan conseguir objetivos no funcionales en el diseño, como reducir la frecuencia de operación, el tamaño y el consumo del circuito.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 14 de 120

3.1.1. Unidades de diseño

Una unidad hardware se visualiza como una “caja negra” con una interfaz y un funcionamiento específico. En VHDL la caja negra se denomina **entidad**, la cual describe las entradas y salidas del diseño, y la descripción del comportamiento del circuito se denomina **arquitectura**. Toda arquitectura tiene que estar asociada a una entidad y una entidad puede describirse de múltiples maneras por lo que puede tener múltiples arquitecturas.

En el modelo, se pueden definir paquetes y librerías, los cuales pueden encontrarse en el sistema o ser creados por el usuario, que nos mostrarán que tipo de señales y/o componentes, funciones, etc., podemos utilizar en la entidad y ayudarnos en diseños complejos. Para usar los paquetes y las librerías hay que indicarlo antes de declarar la entidad.

3.1.1.1. Entidad



La entidad es única para un circuito y define los puertos de entrada/salida.

Los puertos de una entidad son los canales de comunicación del circuito con el exterior. Consta de un nombre, que debe de ser único dentro de la entidad y tiene las siguientes propiedades:

- Modo: es la dirección del flujo de datos. Puede ser:
 - IN: Una señal que entra en la entidad y no sale. La señal puede ser leída pero no escrita.
 - OUT: Una señal que sale de la entidad y no es usada internamente. La señal no puede ser leída dentro de la entidad.
 - INOUT: Una señal bidireccional, entrada/salida de la entidad.
 - BUFFER: Una señal de salida que no puede leerse su contenido.
- Tipo: son los valores posibles que puede tomar el puerto. Por ejemplo: '0', '1', 'a', 'b', etc.

Los puertos, por lo tanto, son una clase especial de señales, que adicionalmente al tipo de señal añaden el modo.

```
entity entity_name is
  port(
    X, Y:      in tipo;
    Z:        out tipo);
end nombre;
```

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 15 de 120

3.1.1.2. Arquitectura

La arquitectura está asociada a una entidad y describe la estructura y el comportamiento del circuito. La combinación de la arquitectura con la entidad describe un diseño del circuito completo. La descripción VHDL de la arquitectura tiene dos partes:

- Parte declarativa: se declaran todas las señales y componentes que se van a utilizar en el diseño.
- Cuerpo: se describe la funcionalidad del circuito con asignaciones, instanciaciones y procesos.

```

architecture architecture_name of entity_name is
  Parte declarativa:
  Señales
  Declaración de componentes
begin
  Cuerpo de la arquitectura:
  Instrucciones concurrentes
    Asignaciones de señales
    Colocación de componentes
    Procesos
    Instrucciones secuenciales
end architecture_name;

```



Las sentencias que hay en el cuerpo de la arquitectura se ejecutan de forma concurrente, es decir, cuando cambian sus entradas y de forma simultánea.

Para realizar poder realizar instrucciones secuenciales se utilizan los procesos, que son instrucciones concurrentes dentro del cuerpo de la arquitectura.

3.1.2. Objetos

Se utilizan para almacenar datos de cualquier tipo, que van a ser utilizados en nuestro diseño. Existen cuatro tipos de objetos: contantes, variables, señales y archivos.



- Constante: se inicializa con un valor en un punto determinado del diseño. Este valor permanece fijo y no puede ser alterado. Las constantes se pueden declarar en la entidad, en la arquitectura o en el proceso.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 16 de 120

- **constant** nombre: tipo:=valor;
- Variables: el contenido de una variable puede ser modificado en cualquier momento y también se le puede asignar un valor inicial. Se usan y declaran dentro de un proceso o un subprograma. Las variables declaradas en dos procesos diferentes, son independientes.
 - **variable** nombre: tipo:=valor;
- Señales: representan las conexiones reales en el circuito, es decir, tiene un sentido físico. Pueden contener un valor o una secuencia de valores y se le puede asignar un valor inicial. Las señales tienen una historia pasada de valores, un valor presente y valores futuros. Solamente se pueden modificar los valores futuros de una señal mediante una asignación. Se declaran en la parte declarativa de la arquitectura y son visibles en todos los procesos y bloques de la misma, es decir, se pueden utilizar en cualquier parte del programa.
 - **signal** nombre: tipo;
- Archivo: contiene una lista de valores, los cuales pueden ser leídos o escritos durante el programa.

Los datos de los objetos, tienen que tener un tipo definido. Los tipos predefinidos en VHDL pueden ser de tipo escalar, fichero o compuestos. Los que vienen definidos según el estándar son los siguientes:

- Boolean (False, True)
- Bit ('0', '1')
- Character (a,b, ...)
- Integer (número entero)
- Natural (número natural)
- Real
- Bit_vector (arrays de bits)
- Time

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 17 de 120

3.1.3. Operadores

Estos son los tipos de operadores definidos en VHDL:

- Lógicos: NOT, AND, NAND, OR, NOR, XOR, XNOR
- Relacionales: =, /=, <, <=, >, >=. Devuelven un valor booleano.
- Adición: +, -, &. El símbolo & realiza la operación de concatenación de vectores. La dimensión del vector resultante será igual a la suma de las dimensiones de los vectores sobre los que opera.
- Multiplicativos: *, /, rem (resto), mod (módulo).
- Misceláneos: abs. (valor absoluto), ** (elevar un número a una potencia entera, mod (módulo), rem (resto).
- Desplazamiento:
 - SLL, SRL: desplazamiento lógico a izquierda y derecha
 - SLA, SRA: Desplazamiento aritmético a izquierda y derecha. Conserva el signo (el valor que tuviera el bit más significativo)
 - ROL, ROR: rotación a izquierda y derecha.

3.1.4. Sentencias concurrentes



Como se ha explicado anteriormente, las instrucciones concurrentes van dentro de la parte declarativa de la arquitectura. Este tipo de instrucciones se ejecuta de forma paralela, es decir, se ejecutan todas al mismo tiempo. Por lo tanto, podemos decir que el orden de las instrucciones concurrentes dentro de la arquitectura es indiferente.

Este tipo de sentencias pueden ser de tres tipos:

- Asignación de señales y variables, las cuales pueden ser asignaciones directas, directas pero con retraso o asignaciones condicionales. Las formas básicas son:
 - $Y \leq B$
 - WHEN-ELSE
 - WITH-SELECT-WHEN
- Inserción de componentes.
- Procesos

3.1.5. Sentencias secuenciales

Se realizan mediante procesos. Un proceso contiene un conjunto de instrucciones que se ejecutan de forma secuencial y se comunica con el resto del diseño mediante señales o variables declaradas fuera del proceso. Los procesos al ser instrucciones concurrentes, se declaran dentro de la parte

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 18 de 120

declarativa de la arquitectura y se ejecutan en paralelo. Un proceso, además de realizar lógica secuencial, también puede realizar lógica combinacional (como por ejemplo una asignación de señal).

En los procesos no se pueden declarar señales, pero sí se pueden utilizar, se pueden declarar variables y tener una lista de sensibilidad, la cual indica cuando se activa el proceso.

La estructura de los procesos es la siguiente:



```
process (lista de sensibilidad)
    Variable declaración;
begin
    Instrucciones secuenciales; asignaciones;
end process;
```

Las instrucciones secuenciales que se pueden realizar dentro de un proceso son las siguientes:

- **Wait**: esta instrucción provoca una pausa hasta que ocurre el evento o condición.
- Asignaciones de señales y variables.
- Control de la secuencia por medio de ejecución condicional (**if** y **case**), bucles o repeticiones (**for...loop**, **while**, **until**) o saltos (**next** y **exit**).
- Subprogramas que definen algoritmos secuenciales que se usan repetitivamente en un diseño (**procedure** y **function**).
- **Null**: instrucción que indica que no hay acción a realizar.

3.1.6. Declaración de componentes

Un componente es una entidad ya definida, con una arquitectura que realiza una función específica. Dentro de una arquitectura se pueden declarar componentes, y así poder utilizar la funcionalidad de ese componente en esa arquitectura. En la declaración del componente solo se suministra la información del identificador del componente y de los terminales, es decir, se declara la entidad del componente. Dentro de la arquitectura, se realiza la inserción del componente, donde se pueden realizar asignaciones de señales a los puertos del componente.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 19 de 120

architecture architecture_name **of** entity_name **is**

Declaración del componente:

component component_name **is**

port(X, Y: **in** tipo;

 Z: **out** tipo);

end component;



begin

 module_name : component_name

port map (asignaciones a cada puerto);

end architecture name;

Utilizando componentes, se puede utilizar la metodología de diseño bottom-up, la cual se caracteriza por describir primero los diseños más bajos de la jerarquía, para luego terminar con un diseño o construcción de alto nivel.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 20 de 120

3.2. Dispositivos lógicos programables

Un dispositivo lógico programable es cualquier sistema digital cuya función está especificada por el usuario, después de haberse fabricado el dispositivo. Actualmente, debido a la creación de nuevas estructuras y al aumento de la integración en los circuitos, se puede programar cualquier función booleana en un PLD, desde el funcionamiento de una puerta lógica, hasta procesos más complejos como procesadores digitales. Estos dispositivos destacan por estar armónicamente estructurados, tanto en su hardware como en el software utilizado para su programación.

Los dispositivos lógicos programables se pueden clasificar en tres grandes grupos: SPLDs, CPLDs, FPGAs.

3.2.1. SPLDs

Los dispositivos lógicos programables simples, presentan estructuras matriciales lógicas basadas en planos AND-OR. En función de qué plano sea programable o fijo, existen los siguientes tipos:

- PLA (*Programmable Logic Array*): tanto el plano de puertas AND, como el plano de puertas OR son programables.
- PROM (*Programmable Read Only Memory*): solamente el plano de puertas OR es programable.
- PAL (*Programmable Array Logic*): solamente el plano de puertas AND es programable.

3.2.2. CPLDs

Los dispositivos lógicos programables complejos utilizan SPLDs interconectados mediante una matriz de conmutación fija o programable. Estos componentes proporcionan una mayor densidad de integración de puertas, las cuales pueden superar ampliamente las 20000. Una de las razones de uso de este de PLD, es precisamente que integran muchos SPLDs en un mismo dispositivo, esto proporciona costes más bajos de implementación, ciclos de desarrollo más cortos y una temporización más simple y determinista. Además, la estructura de este dispositivo facilita la interconexión con otros dispositivos y proporciona herramientas de diseño simples.

3.2.3. FPGAs

La traducción de sus siglas al castellano se corresponde con Red de Puertas Programables en campo. Dentro de la familia de los PLD, las FPGAs son los dispositivos que tienen más densidad de puertas lógicas superando el millón de

puertas lógicas, esto permite al usuario programar proyectos que involucren la necesidad de redes de compuertas programables muy densas, mediante herramientas técnicas económicas y sencillas para el usuario. Estas características hacen que estos dispositivos se utilicen en el campo de la investigación, en laboratorios, en ámbito educativo e incluso en pequeñas empresas, debido a su alta versatilidad, alta capacidad y a su programación fácil y económica.

3.2.3.1. Estructura interna

La estructura interna de una FPGA está compuesta por bloques de entrada y salida, bloques lógicos configurables y por una red de interconexión (Figura 7).

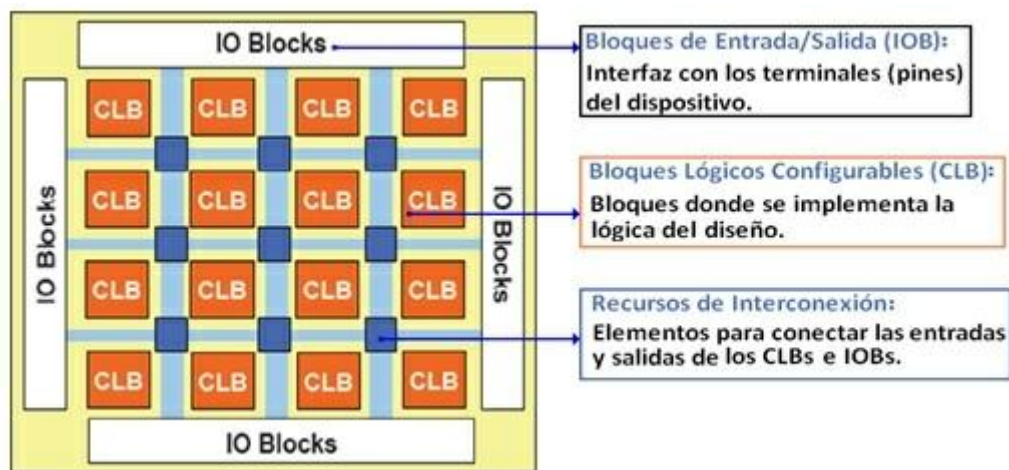




Figura 7: Estructura interna de una FPGA.

Los bloques lógicos configurables o CLBs, son elementos funcionales y ampliamente configurables, que hace posible al usuario programar cualquier tipo de función lógica. Su estructura interna se basa en dos biestables tipo D, un bloque lógico combinacional puro, también llamado LUT (*Look-up tables*) y varios multiplexores programables, que permiten seleccionar los diferentes modos de funcionamiento del bloque.

La red de conexión de la FPGA permite interconectar los CLBs del dispositivo. Estas conexiones se pueden realizar de tres formas diferentes, en función de la longitud y del destino de las uniones. Pueden ser:

- Interconexiones de propósito general.
- Interconexiones directas.
- Interconexiones largas.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 22 de 120

Por último, los bloques de entrada y salida permiten interconectar otros dispositivos a los terminales de la FPGA.

Las FPGAs se pueden clasificar en función de la tecnología de programación utilizada, en Fusible/anti fusible, Celda de memoria no volátil y Celda de memoria RAM estática:

3.2.3.2. Fusible/anti fusible:

Este tipo de FPGAs se caracteriza por la utilización de anti fusibles en las interconexiones de las celdas lógicas. Este tipo de interconexión eléctrica no es reversible, lo que significa que estas FPGAs no se pueden reprogramar y que su programación es definitiva. Sin embargo, este tipo de interconexión ocupa estructuralmente poco espacio, lo que permite conseguir densidades de integración muy altas.

Para realizar test de funcionamiento, estos dispositivos tienen un modo de prueba, que permite al desarrollador comprobar el funcionamiento interno a través de un hardware y un software especial.



3.2.3.3. Celda de memoria RAM

La estructura interna de este tipo de FPGA la forman una matriz de CLBs, la cual la recorren una red de líneas verticales y horizontales. Las conexiones a esta red se realizan mediante transistores CMOS, los cuales están controlados mediante memorias RAM (*Random Access Memory*), es decir, en la RAM se configuran las interconexiones de la matriz.



En las memorias RAM se puede escribir y reescribir ilimitadamente, de forma rápida, es decir, direccionando el acceso a memoria y teniendo en cuenta el tiempo de acceso. Tampoco necesitan tensiones de programación altas o especiales, solamente necesitan la tensión de alimentación del circuito. Por lo tanto, se puede decir que estas propiedades las adquiere este tipo de FPGA siendo esta borrable y programable eléctricamente.

El problema del uso de memorias RAM es que estas pierden su información cuando se les corta la energía, es decir, son volátiles. Para subsanar este inconveniente se crearon dos posibles soluciones:

- Mantener alimentada la memoria RAM mediante una pila de litio. La pila podría durar años ya que el consumo de la memoria es bajo. Incluso se podría utilizar una pila recargable para evitar el riesgo de descarga de la pila.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 23 de 120

- Utilizar una memoria PROM, para almacenar los datos de programación inicial o de configuración, con un circuito secuencial que vuelque los datos en la memoria RAM cuando vuelva la alimentación al circuito. Otra forma sería guardar la información de configuración en la memoria de un microprocesador, y del mismo modo transfiera la información ante cualquier pérdida de datos.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 24 de 120

3.3. Matlab

Es una herramienta de software matemático, sus siglas significan laboratorio de matrices (*Matrix Laboratory*). Fue creada por Cleve Moler, el presidente del departamento de informática de la Universidad de Nuevo México a finales de la década de 1970. Su idea fue crear una herramienta para sus alumnos que les dieran acceso a LINPACK y EISPACK (bibliotecas de software para realizar álgebra lineal numérica en computadoras digitales escritas en lenguaje Fortran) sin que tuvieran que aprender el lenguaje Fortran.

En 1984 Cleve Moler se unió a Jack Little y Steve Bangert, reescribieron MATLAB en C y fundaron Mathworks. Estas bibliotecas reescritas se conocen como JACKPAC. En los años 2000 MATLAB se reescribió para utilizar unas bibliotecas más nuevas para la manipulación de matrices, LAPACK.



MATLAB tiene un lenguaje de programación propio, llamado lenguaje M, y utiliza un entorno de desarrollo integrado IDE, que facilita al usuario el desarrollo software. Es compatible con la mayoría de los sistemas operativos como Windows, Unix, GNU/Linux y MacOS.

Este software entre sus utilidades más usadas, permite operar con matrices, representar datos y funciones, implementar algoritmos, conectar programas escritos en un lenguaje diferente y dispositivos hardware y crear interfaces de usuario GUI (*Graphical User Interface*). Además en los últimos años ha mejorado sus prestaciones, como por ejemplo, añadiendo la opción de programar directamente procesadores digitales de señal o la **creación de código VHDL** de un diseño.

MATLAB dispone de dos herramientas más que expanden su capacidad, Simulink, una herramienta de simulación y GUIDE, un editor de interfaces de usuarios. Además MATLAB permite aumentar su capacidad de procesamiento, añadiendo cajas de herramientas específicas (*toolboxes*) y en Simulink con los paquetes de bloques (*block sets*).

3.3.1. Simulink

Simulink es una herramienta que utiliza una programación visual de alto nivel. Permite relacionar bloques que implementan una determinada función, creando un flujo de datos, construyendo así, un diagrama de bloques secuencial.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 25 de 120

Las funciones que implementan los bloques que conforman las librerías de Simulink, son funciones desarrolladas en Matlab. Esto permite realizar interacciones entre Simulink y Matlab.

Simulink tiene multitud de librerías destinadas a diferentes aplicaciones, por ejemplo, procesamiento de señales, sistemas de comunicación, redes neuronales, sistemas eléctricos, etc., además tiene la opción de crear nuevas librerías destinadas a otras aplicaciones. Por todo ello, Simulink se considera una herramienta muy potente en muchos campos, siendo fundamental a niveles educativos y tecnológicos.



Esta herramienta también tiene la opción de interactuar con dispositivos hardware externos al ordenador, como un micrófono, un altavoz, un procesador digital de señales, Arduino, etc., permitiendo así ejecutar los modelos construidos en Simulink en placas compatibles o interactuar con esos dispositivos hardware.



En resumen, se puede decir que Simulink es una herramienta muy útil y muy utilizada debido a las siguientes ventajas:

- Utiliza una programación por bloques, la cual es fácil, intuitiva y cómoda.
- Se pueden hacer simulaciones en tiempo real mediante flujo de datos y de esta forma visualizar resultados en tiempo real del modelo.
- Permite interactuar con el entorno Matlab.
- Proporciona una programación rápida en funciones complicadas, evitando la programación mediante código.
- Es posible conectarse a dispositivos hardware externos.
- Se pueden crear librerías y bloques nuevos mediante código MATLAB y C para aplicaciones específicas.
- Es más fácil e intuitivo interpretar los diseños frente a un archivo de código, debido a la visualización en diagrama de bloques.

Sin embargo, Simulink también presenta las siguientes desventajas:

- El tiempo de ejecución del código es más lento que en otros como el lenguaje de programación en C/C++.
- En función de la aplicación que vaya a tener el diseño, el nivel de programación puede ser más alto del requerido lo que obligará a recurrir a la programación en el entorno de Matlab.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 26 de 120

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 27 de 120

4.PROCESADO DE AUDIO

4.1. Introducción

En el procesamiento de señales digitales de audio existe el concepto de efecto de audio. Un efecto de audio hace referencia a cualquier modificación que se efectúa sobre una señal de audio, que posteriormente es audible. Existen multitud de tipos efectos, desde eliminar ruido de fondo hasta un efecto de distorsión en una guitarra eléctrica.

Dependiendo del efecto sonoro que se quiera obtener, se tendrán que utilizar diferentes estructuras y configuraciones, hasta conformar un sistema digital de audio que pueda implementar ese tipo de efecto (Figura 8). El efecto digital tiene unos parámetros de control, que permiten realizar modificaciones del resultado. Para ajustarlos, se necesita obtener información de la señal de entrada y de salida, para que así el controlador pueda modificar los parámetros a sus necesidades. Esta información puede ser acústica (salidas de audio) o visual (barras led).

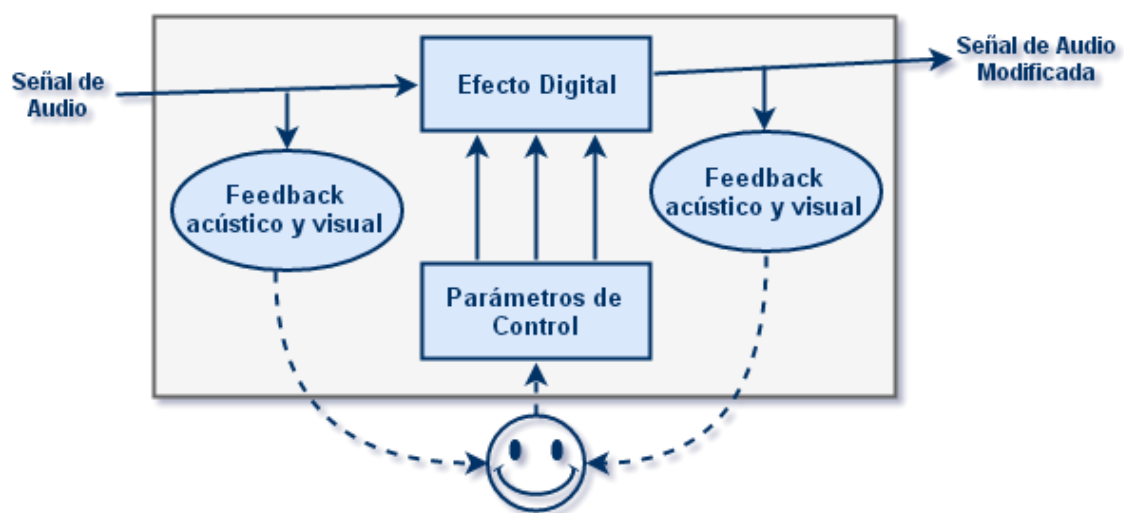




Figura 8: Diagrama funcional para creación de efectos digitales de audio

Los efectos de audio están presentes en toda la cadena de producción de cualquier música y hay multitud de variantes, esto hace complicada una

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 28 de 120

clasificación, ya que se pueden clasificar por ejemplo, por la finalidad del efecto (de interpretación, de producción etc.), por la percepción del efecto, etc.

Otra posible clasificación sería por el tipo de sistema digital utilizado en el diseño del efecto. Los principales sistemas digitales que se utilizan son:

- Basados en retardadores: mediante estos sistemas se pueden generar los efectos de eco, chorus, reverberación, vibrato, flanging, doubling, slapback.
- Sistemas moduladores: efectos como Vibrato, chorus, flanging, trémolo, phasing, Wah-wah
- Sistemas lineales: puede generar efectos como distorsión, overdrive, fuzz, octavador.

4.2. Efectos de sonido basados en retardadores

Un sistema retardador añade a la señal de entrada, otras señales del mismo origen, pero retrasadas en el tiempo y atenuadas o amplificadas.

Los parámetros de control de un sistema retardador que definen los tipos de efecto, son:

- Tiempo de retardo: es el tiempo que transcurre entre la señal original y la primera versión retardada.
- Número de retardos: es el número de veces que se le aplica un retardo a la señal original.
- Atenuación: es la pérdida de energía que se le aplica a cada una de las versiones retardadas.

En función de los parámetros de control y de las estructuras del sistema podemos obtener los siguientes efectos.

4.2.1. Sistema retardador basado en filtros peine FIR

Este tipo de filtro se utiliza para añadir solamente una versión a la señal original, es decir, con un retardo. La atenuación es constante, y el tiempo de retardo puede ser constante o variable, en función del efecto deseado.

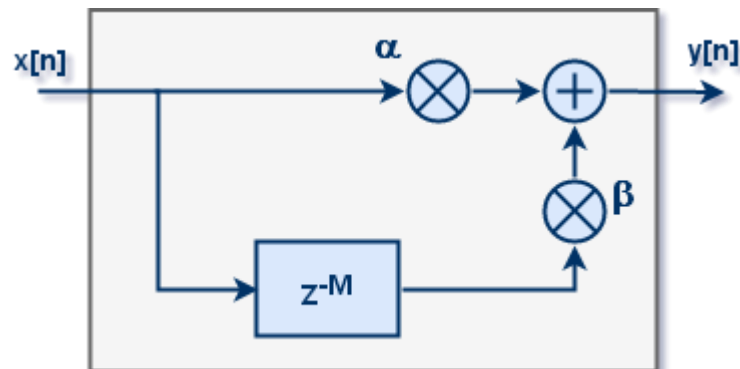


Figura 9: Filtro peine FIR

En el diagrama de la figura 9 se observa como el sistema almacena un número de muestras (M) a una determinada frecuencia de muestreo, luego las atenúa multiplicándola por un factor β y posteriormente la añade a la señal original. El tiempo de retardo se representa por la letra τ y es igual al número de muestras almacenadas por la frecuencia de muestreo, es decir, el tiempo que se tarda en almacenar las muestras.

La ecuación de diferencias del sistema es la siguiente:

$$y[n] = \alpha x[n] + \beta x[n - M]$$

En la respuesta en frecuencia del sistema (figura 10) se observa como el filtro amplifica todas las frecuencias que son múltiplos a la inversa del tiempo de retardo y atenúa las frecuencias intermedias. Este filtro recibe el nombre de peine debido a la similitud de las púas del peine con el diagrama frecuencial.

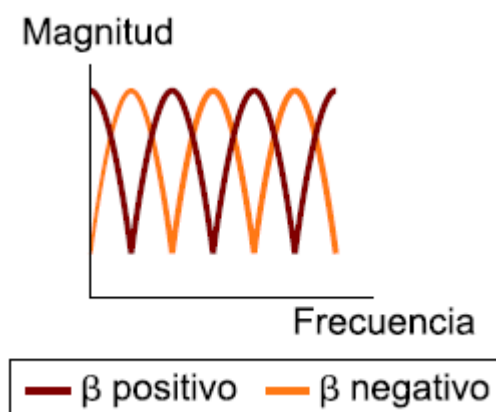




Figura 10: Respuesta en frecuencia filtro FIR

La función de transferencia de este filtro es igual a:

$$H(z) = \alpha + \beta z^{-M}$$

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 30 de 120

Los efectos posibles utilizando esta estructura dependen del tiempo de retardo y de si este es constante o variable. Para el tiempo de retardo contante, se puede tener diferentes efectos, dependiendo del valor del tiempo de retardo:

- Eco: si se aplica un tiempo de retardo mayor de 50 segundos entre la señal original y la versión retardada, se obtendrá este efecto. La percepción audible es como de un eco natural, pero de solo un retardo, es decir, como si se estuviera en una habitación vacía y se emitiera un sonido, que rebotara una vez y volviera a nuestro oído. Si se aumenta el tiempo de retardo, la distancia audible entre el sonido original y la versión será mayor.
- Doubling: este efecto añade a la señal original, la versión retardada con muy poco tiempo de retraso, como máximo de 10ms. El oído humano no es capaz de percibir este retardo, por lo que a efectos audibles no se diferencian la señal original entre la retardada, es como si fueran la misma. Este efecto se utiliza para dar cuerpo a los sonidos, por ejemplo, si se aplica a una voz humana, el efecto resultante parecerá dos voces humanas, simulando un coro.
- Slapback: en este efecto el tiempo de retardo es un poco mayor al efecto doubling, entre 25ms y 100ms. El slapback si es perceptible para el oído humano, y el efecto audible es como una repetición muy rápida del sonido original.

En cambio, si el tiempo de retardo es variable, es decir, el número de muestras almacenadas varía con el tiempo, se pueden obtener los siguientes efectos:

- Flanging: este efecto añade a la señal original una única versión retardada, pero con el tiempo de retardo variable en cada una de los retardos. Generalmente, el retardo es una función senoidal o triangular de una frecuencia de 0.1Hz a 1Hz y con un tiempo de retardo de 0 a 15 ms. El efecto audible se asemeja a un sonido metálico oscilante sobre la señal original. Este efecto en la era analógica se realizaba con dos cintas que reproducían el mismo sonido pero con las ruedas de giro (*flange*, en inglés) desplazadas alternativamente, creando así retardos entre ellas. La función $M(n)$ senoidal o triangular que se utiliza para el efecto flanging es:

$$M(n) = M_0(1 + A \sin(2\pi f n T))$$

La estructura para implementar efectos con el tiempo de retardo variable, se observa en la figura 11.

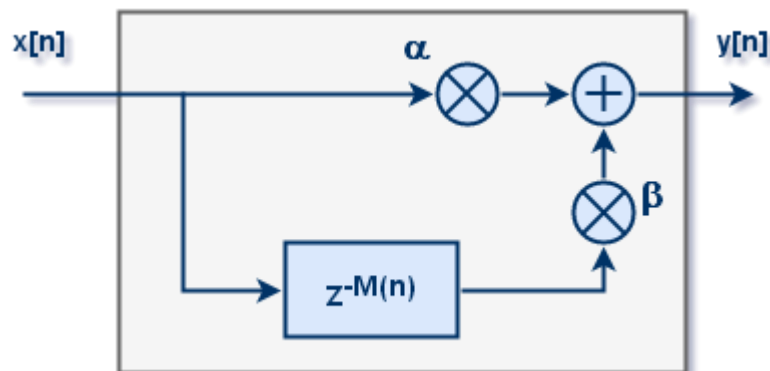


Figura 11: Filtro peine FIR con retardo variable

- **Vibrato:** En este efecto se realizan variaciones periódicas en la frecuencia de la señal original. La percepción audible es como una oscilación del sonido original, por ejemplo, la técnica de vibrato que se utiliza en los instrumentos de cuerda, consiste en oscilar el movimiento al presionar la cuerda. Este efecto se puede realizar con moduladores, pero también con un filtro retardador, simulando el efecto Doppler. Para ello, se retarda la señal con un tiempo de retardo entre 0ms a 1ms, y este tiempo se modula mediante una onda senoidal de frecuencia entre 5Hz a 14Hz. De esta forma estamos variando el tiempo de retardo periódicamente, lo que produce una variación periódica en la frecuencia percibida.

$$y[n] = \beta x[n - M(n)]$$

En la figura 12, se visualiza como la estructura para este efecto no tiene lazo de realimentación.

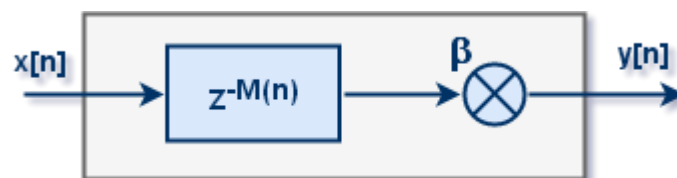




Figura 12: Filtro peine FIR sin realimentación

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 32 de 120

4.2.2. Sistema retardador basado en filtros de peine IIR

En este tipo de filtros se le añade a la señal original, infinitas versiones retardadas.

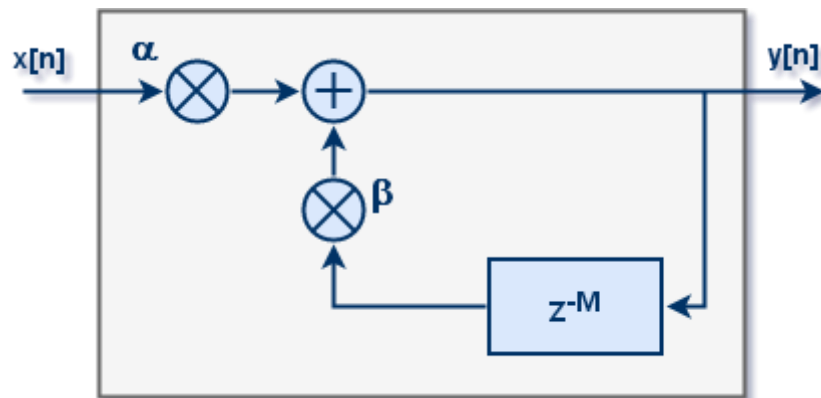


Figura 13: Filtro peine IIR

En el diagrama de la figura 13 se observa cómo se almacenan un número de muestras (M) de la señal de salida a una determinada frecuencia de muestreo, las atenúa y las añade a la nueva señal de entrada.

La ecuación de diferencias y la función de transferencia son las siguientes:

$$y[n] = \alpha x[n] + \beta y[n - M]$$



$$H(z) = \frac{\alpha}{(1 - \beta z^{-M})}$$

En este filtro, se modifica la amplitud de cada versión retardada de la señal original por un factor de α^p , donde p es el número de veces que la señal ha sido retardada. Para garantizar la estabilidad del filtro, el valor de β debe ser menor que 1, porque si no, la señal de salida crecería sin límite.

En la respuesta en frecuencia del filtro se observa como el sistema amplifica todas las frecuencias que son múltiples al tiempo de retardo y atenúa las frecuencias intermedias. La diferencia respecto a los filtros peine FIR, es que en los filtros peine IIR la amplitud de los picos se estrecha conforme α se va acercando a 1.

Con estos filtros podemos diseñar los siguientes efectos:

- Eco: Es el mismo efecto que se genera con el filtro FIR, pero en este caso se le añade a la señal original múltiples versiones retardadas y atenuadas.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 33 de 120

- Chorus: En este efecto se le añade a la señal original múltiples versiones retardadas con el tiempo de retardo de entre 10ms y 25ms y con cambios aleatorios en el tiempo de retardo. El efecto audible es como si al sonido original se le añadiese un coro.

4.2.3. Filtro reverberador

El efecto reverberador natural se produce por la reflexión del sonido, que llega al receptor antes de que se haya extinguido el sonido original. Esto quiere decir que este efecto está formado por múltiples versiones retardadas que se añaden a la versión original. Cada una de las versiones retardadas, no deben de superar los 50ms de tiempo de retraso, ya que para valores mayores el efecto percibido se asemeja más al efecto eco.

La estructura más simple para un filtro reverberador se puede realizar mediante la combinación de filtros peine FIR y filtros peine IIR (Figura 14).

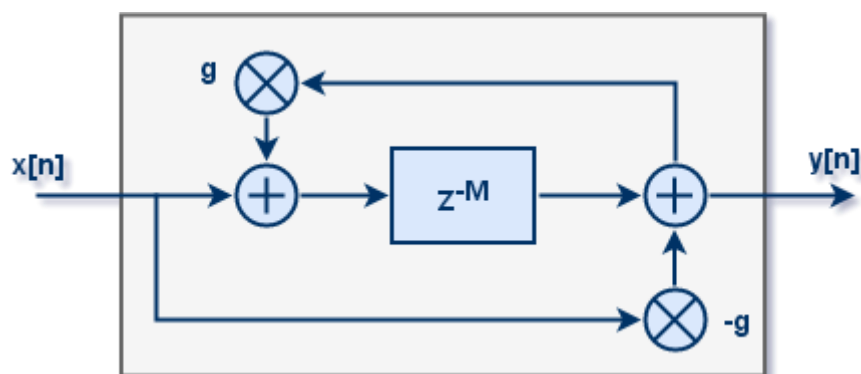


Figura 14: Filtro reverberación simple



La ecuación de diferencias del diagrama de la figura 14 es la siguiente:

$$y[n] = -gx[n] + x[n - M] + gy[n - M]$$

Los sistemas para implementar estos filtros pueden ser muy sofisticados y complejos, teniendo como fin obtener una reverberación artificial. Este efecto a efectos audibles aporta calidez al sonido.

4.2.4. Filtros IIR paso todo

Como se indicó anteriormente, los filtros IIR amplifican las frecuencias múltiples al tiempo de retardo, estrechando la amplitud de los picos conforme se aumenta la ganancia de la señal de entrada, creando lo que se llama una coloración excesiva en esas frecuencias. Los filtros IIR paso todo, disminuyen este efecto y además permiten añadir una gran densidad de ecos a la señal de entrada. La ecuación de diferencias de este filtro y función de transferencia es:

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 34 de 120

$$y(n) = -ax(n) + x(n - D) + ay(n - D)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{-a + z^{-D}}{1 - az^{-D}}$$

La estructura de este filtro se visualiza en la figura 15:

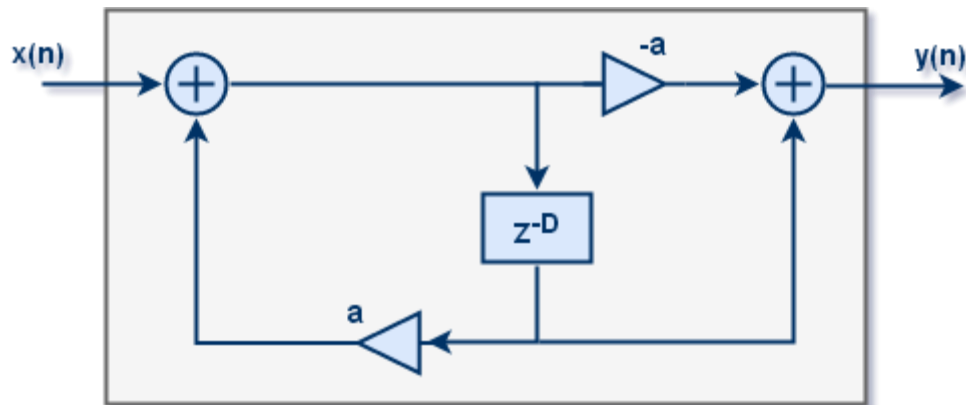


Figura 15: Filtro IIR paso todo

4.2.5. Retardos modulados en el tiempo

Los efectos de audio flanger, vibrato y chorus para su implementación utilizan un sistema de retardos modulados por el tiempo. Estos retardos son valores no enteros, lo que provoca que no puedan ser aplicados en un sistema de señal discreta.

Una de las posibles soluciones a este problema es aplicar el método de interpolación, el cual calcula cada tiempo de retraso entre dos muestras adyacentes. Para aplicar la interpolación existen diferentes técnicas, la utilizada en los filtros diseñados utiliza la interpolación lineal.

4.2.6. Estructura paso todo generalizada

Existe una estructura genérica que en función de los parámetros de configuración se pueden conseguir todos los efectos explicados anteriormente. Este sistema se denomina estructura paso todo generalizada, y consiste en un filtro IIR paso todo.

El diagrama de esta estructura se puede visualizar en el la figura 16.

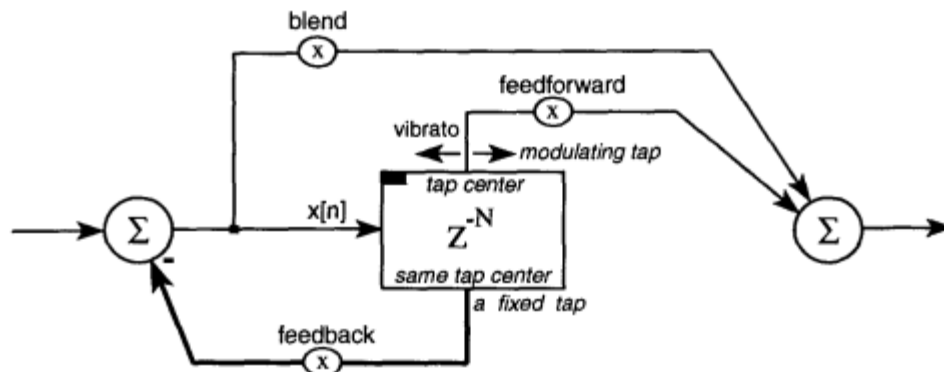




Figura 16: Estructura filtro paso todo generalizada

Se observa como cambiando los parámetros se puede conseguir una estructura u otra. Por ejemplo, si se anula el lazo de realimentación del acumulador se obtendría un filtro FIR utilizado para obtener el efecto eco, en cambio, si además se anula el lazo de realimentación de la señal de entrada se conseguiría el efecto vibrato.

Por lo tanto, dependiendo de los parámetros de configuración y del tipo de modulación del retraso se conseguirá un efecto u otro. La tabla 1 indica esta relación para cada efecto:

Tabla 1: Parámetros de configuración por tipo de efecto

Efecto	Tiempo de retardo	Modulación	Blend	FeedForward	FeedBack
Doubling	< 10ms	-	0.7	0.7	0
Slapback	25ms - 50ms	-	0.7	0.3	0
Eco	> 50ms	-	0.7	0.3	0
Flanging	0ms - 15 ms	Sinusoidal (0.1Hz - 1Hz)	0.7	0.7	-0.7
Vibrato	0ms - 1ms	Sinusoidal (5Hz - 14Hz)	0	1	0
Chorus	10ms - 25ms	Aleatoria	0.7	1	-0.7

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 36 de 120

4.3. Implementación de filtros mediante Simulink

Para la implementación y el diseño de los filtros el software Simulink es una herramienta muy útil. Mediante esta herramienta, se pueden crear diseños con diagramas de bloques, y realizar simulaciones reales del filtro.

Simulink además proporciona una librería DSP (procesado digital de señal) llamada *System Toolbox*, con bloques que realizan funciones específicas que se utilizan en el procesamiento de señales y en el diseño de filtros. También proporciona bloques para conectar dispositivos hardware como un micrófono o un altavoz. De esta forma se pueden realizar simulaciones reales del filtro, para percibir si el efecto es el deseado y cambiar los parámetros de configuración si fuera necesario.



4.3.1. Configuración del modelo

El primer paso para el diseño de un modelo en Simulink es configurar los parámetros de simulación del modelo. Para ello, en el panel superior, ventana *Model Configuration Parameters*, vienen todas las opciones para configurar la simulación.

Primero se tiene que especificar si se trata de un modelo de pasos fijo (*fixed-step*) o pasos variables (*variable-step*), es decir, Simulink puede ejecutar el modelo con intervalos de un escalón fijo o variable. En el caso de las señales de audio, como se va a trabajar con una frecuencia fija de muestreo, se elegirá la opción *fixed-step*. Posteriormente tiene la opción de elegir el periodo de muestreo de forma manual o de forma automática. Cuando la frecuencia de muestreo es automática, Simulink escoge la frecuencia en función de las señales de entrada y salida del modelo.

Por último, se tiene que indicar si nuestro modelo es un sistema continuo o discreto. En los diseños creados, se ha utilizado una entrada de muestras discretas de señales de audio a una frecuencia de muestreo de 44.1kHz.

En la figura 17 se puede visualizar la ventana de configuración, *Model Configuration Parameters*.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 37 de 120

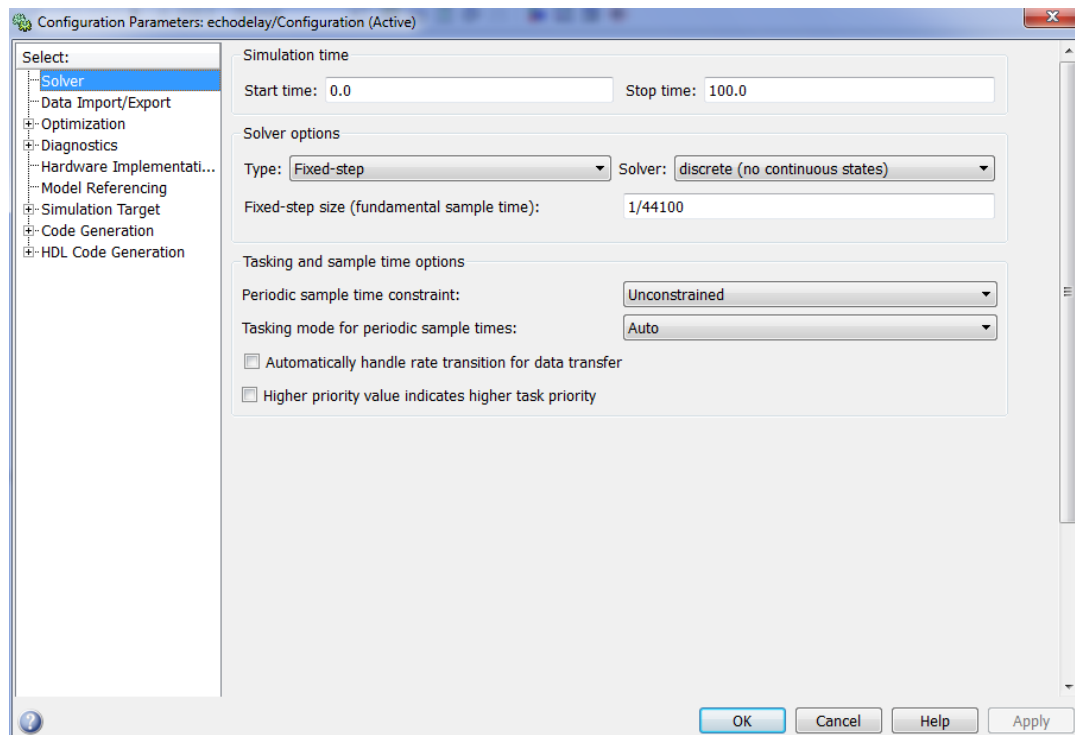




Figura 17: Configuración de los parámetros de simulación del modelo

En Simulink existen dos tipos de flujos de datos, los cuales se pueden elegir para cada bloque del modelo. El flujo puede ser muestra a muestra (*Sample-based*) o por un conjunto de muestras (*Frame-based*). Los modelos que utilizan el tipo de flujo *Frame-based* se utilizan en simulaciones en tiempo real que necesiten un buen rendimiento y un proceso rápido. Esto se consigue mediante el proceso de interrupción al adquirir datos. En un flujo *Sample-based*, el sistema de adquisición de datos se interrumpe en cada muestra, la muestra es procesada por el bloque y después de un tiempo de procesamiento, sale por la señal de salida. Si se utiliza un flujo por *frames*, el sistema se interrumpirá menos veces haciendo el sistema más rápido. Sin embargo, si los *frames* contienen un número de muestras altos, puede provocar tiempo de latencias elevadas.

En la figura 18 se puede observar los dos tipos de flujo.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 38 de 120

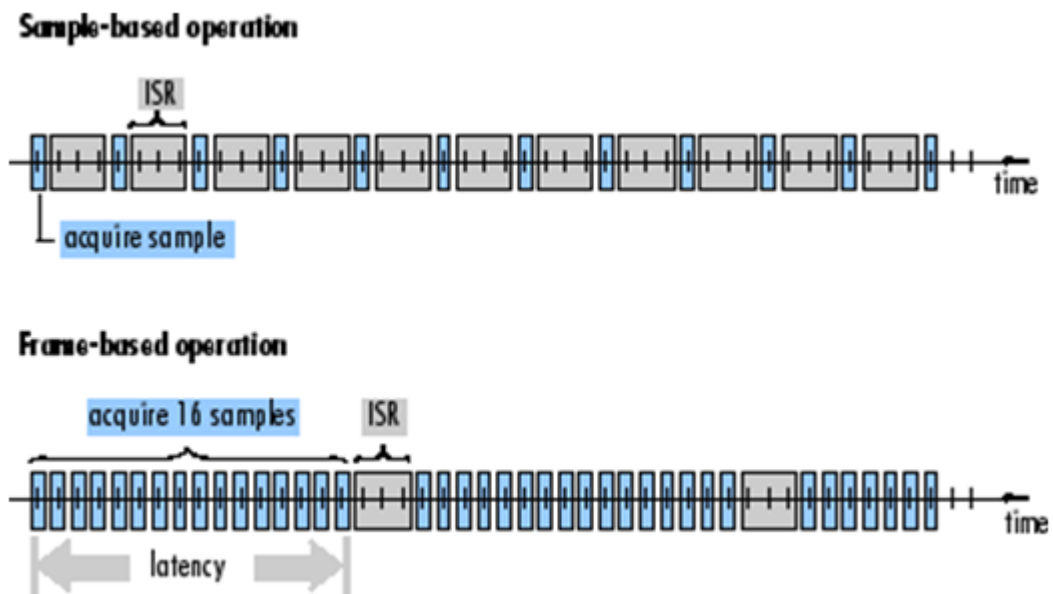


Figura 18: Flujo de datos sample-based y frame-based de Matlab

4.3.2. Bloques utilizados

En el diseño del modelo se colocarán los bloques de izquierda a derecha en función de la dirección del flujo de datos, estando a la izquierda la señal de entrada y a la derecha la señal de salida.

Para la simulación de los filtros, se utiliza una señal de audio de prueba en formato .mp3 reproducida por el bloque **From Multimedia File** de la librería *DSP System Toolboxes / Sources* (Figura 19). Este bloque permite reproducir archivos multimedia almacenados en el PC. En sus parámetros de configuración se puede seleccionar el tipo de dato de la muestra, la frecuencia de muestreo del bloque, la cual puede ser la misma que la del archivo o una específica y el tipo de flujo de datos. En los diseños se han utilizado muestras de 16bit, a la misma frecuencia de la señal de entrada (44,1kHz) y el flujo de datos ha variado dependiendo de la estructura.

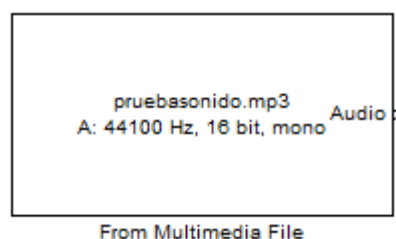




Figura 19: Bloque From Multimedia File

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 39 de 120

De la misma librería, pero en la carpeta *Sinks*, se ha utilizado el bloque **To Audio Device** para reproducir la señal de salida del sistema en un dispositivo hardware de audio (Figura 21). En los parámetros de configuración del bloque, te permite elegir el dispositivo hardware disponible, el tipo de dato, la frecuencia de muestreo, el tamaño del buffer en muestras y el tiempo de espera (*queue duration*). De esta forma se puede escuchar en tiempo real la señal de salida y así percibir el efecto del filtro. El parámetro *queue duration*, indica el tiempo en el que el sistema permanece en silencio y almacenando muestras en un buffer. Transcurrido ese tiempo, el buffer empieza a enviar las muestras de audio al dispositivo hardware (Figura 20). Este parámetro es necesario en sistemas donde se requiere mayor velocidad de procesado.

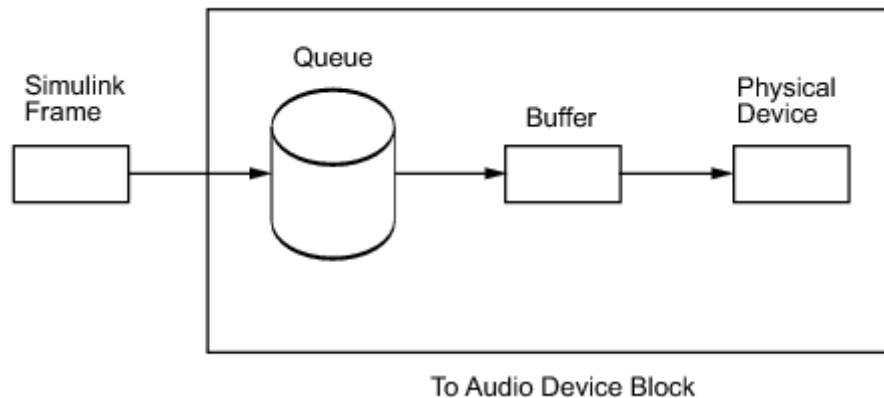


Figura 20: Buffer del bloque Audio Device

También se tiene la opción de utilizar el bloque **To Multimedia File** para guardar la señal de salida en un archivo multimedia, y así poder reproducirlo posteriormente (Figura 21).

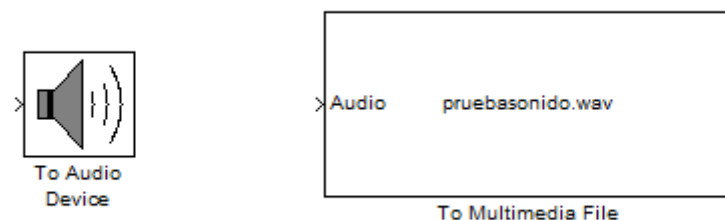




Figura 21: Bloque To Audio Device y To Multimedia File

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 40 de 120

También se utilizan los bloques de la librería por defecto de Simulink, como el bloque **constant** para señales de entrada con un valor constante, bloques matemáticos como **Gain** y **Sum**, los cuales te permiten amplificar señales y sumar varias señales, y bloques para ver gráficamente las señales como el **Display** y **Scope** (Figura 22).

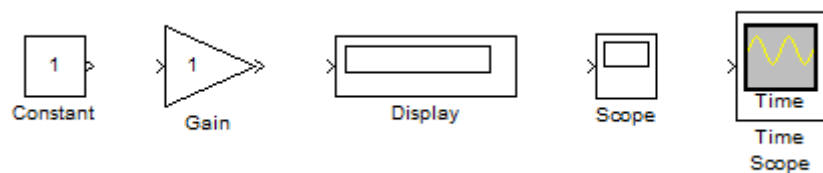


Figura 22: Bloques: Constant, Display, Scope, Time Scope

Por último, se utiliza un interruptor llamado **Manual Switch** (Figura 23) definido en la librería *Simulink/Signal Routing*, mediante el cual se interactúa con el modelo, para así activar el filtro o por el contrario no activarlo y escuchar la señal de salida sin efecto. De esta forma se puede percibir la señal de entrada con efecto o sin efecto, en tiempo real.

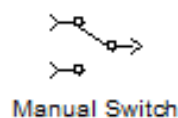




Figura 23: Bloque Manual Switch

Se utilizan otros bloques más específicos en función del filtro que se haya diseñado, los cuales serán explicados en el siguiente punto.

4.3.3. Diseño de filtros en Simulink

Una vez configurados los parámetros de simulación y definidos los bloques básicos de nuestro modelo, eligiendo la librería correspondiente, se crea una estructura única para todos los filtros.

Esta estructura se basa en una **señal de entrada de audio**, la cual es reproducida de forma digital, pero no se puede olvidar que el audio es de naturaleza analógica y ha tenido que tener una conversión analógico digital mediante un convertidor; un **DSP**, el cual realizará una serie de operaciones a la señal digital de entrada para lograr el efecto, este bloque estará formado por bloques específicos que en su conjunto lograrán una señal de salida digital con el efecto determinado. Por último, la **señal digital de salida de audio** del DSP, se reproducirá en un dispositivo hardware, realizando una conversión digital

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 41 de 120

analógica mediante un convertidor. Además, para realizar pruebas a tiempo real del modelo, la señal de entrada también irá directamente a la salida sin pasar por el DSP y mediante un *switch* se podrá conmutar para reproducir la señal sin efecto y con efecto. En la figura 24 se puede visualizar la estructura genérica.

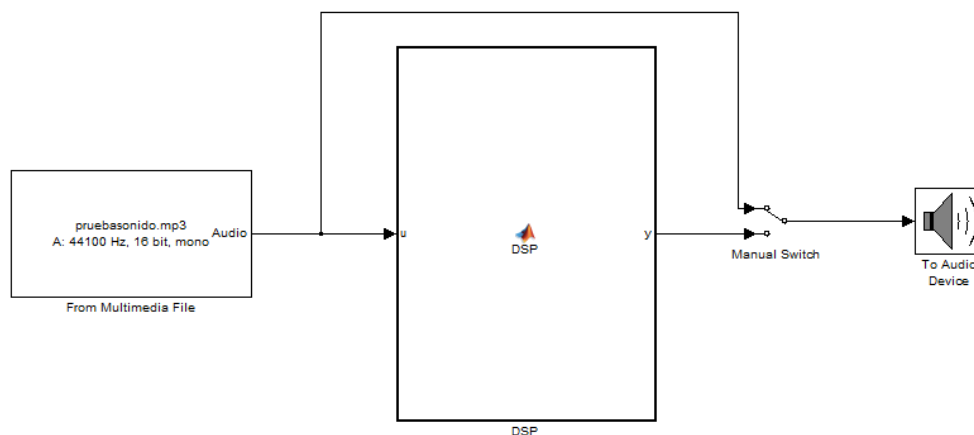




Figura 24: Sistema Genérico DSP

Los DSP diseñados crean efectos de sonido basado en retardadores, utilizando filtros FIR o IIR. Cada DSP utiliza unos bloques específicos, y en función de la estructura y de la configuración de las variables de control, se consigue un efecto u otro. Los modelos diseñados son los siguientes:

4.3.3.1. Modelo para efecto Eco, Slapback y Doubling

En estos efectos se utiliza un filtro FIR con un tiempo de retardo y atenuación constante. Se tendrá por lo tanto, un bloque que retrase una versión de la señal un tiempo constante, que luego suma a la señal de entrada. Los parámetros de control de este filtro son el tiempo de retardo de la señal y la atenuación de la señal retardada.

El bloque de retardo utilizado, es el bloque **Variable Fractional Delay**, el cual se encuentra en la librería *DSP System Toolbox* carpeta *Signal Operations*. Este bloque permite como entrada retardos fraccionados, pudiendo elegir de esta forma el modo de interpolación. En su configuración, se tiene que definir el número máximo de retardos, el modo de interpolación y el tipo de dato de entrada.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 42 de 120

Este bloque tiene dos puertos de entrada, el primero correspondiente a la señal de entrada y el segundo correspondiente al número de muestras que se quieren retrasar, entrada *delay*.

Para estos efectos el tiempo de retraso (τ) va de 10ms a más de 100ms. Como la entrada *delay* corresponde al número de muestras, tenemos que convertir el tiempo de retardo, en muestras. El tiempo de retardo en muestras corresponde, al producto del período de muestreo de cada muestra, con el número de muestras (M):

$$\tau = T \cdot M$$

$$M = \frac{\tau}{T} = \tau \cdot f_m$$

En nuestro modelo, la frecuencia de muestreo de la señal de entrada es de 44,1kHz. Por lo tanto, si se quiere retardar la señal 100ms, se necesitará almacenar 4410 muestras:

$$M = 0,1(s) \cdot 44100(Hz) = 4410 \text{ muestras}$$

Para realizar esta operación, se utiliza un bloque *constant*, que contiene el tiempo de retraso en ms y un bloque *gain*, que multiplica el tiempo de retraso por la frecuencia de muestreo en kHz. Por último, se coloca un bloque *display* para visualizar en tiempo real, el resultado de la operación, que será igual al número de muestras que se van a retrasar en el bloque *Variable Fractional Delay*.

En la figura 25 se puede visualizar el modelo diseñado en Simulink para el efecto eco, *slapback* y *doubling*.

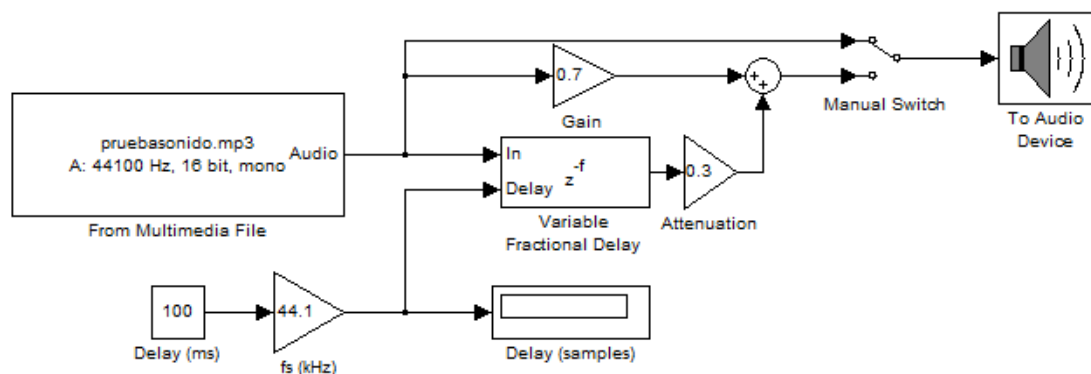




Figura 25: Modelo Simulink para el efecto eco.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 43 de 120

En función del tiempo de retardo, el efecto será diferente. En la prueba realizada con un retraso de 10 ms (441 muestras), el efecto que se percibe es el llamado *Doubling*. En este efecto, se percibe como le da más cuerpo a la señal de entrada, como si por ejemplo hablaran dos personas al mismo tiempo.

Con un tiempo de retardo de 50ms (2205 muestras) se aprecia una repetición rápida de la señal original. Este es el efecto slapback, el cual es muy utilizado en las partes vocales del *rock&roll* de los años 50 y en instrumentos de percusión.

Las siguientes pruebas se han realizado con un tiempo de retraso de 100ms (4410 muestras) y 200ms (8820 muestras) en el que se aprecia una repetición del audio original menos rápida. Este es el efecto Eco de un retardo, que por ejemplo se produce cuando hablamos en una habitación vacía.



4.3.3.2. Modelo efecto vibrato

Para este efecto también se utiliza un filtro FIR, pero con un tiempo de retardo variable y sin sumar la señal retardada al efecto original. En este diseño se utiliza también el bloque *Variable Fractional Delay*, pero en vez de asignarle en el puerto de entrada *delay* un tiempo de retardo constante, se le asigna un tiempo variable.

El tiempo de retardo para este efecto va desde 0ms a 1ms y se hace variable modulándolo con una señal sinusoidal de una frecuencia baja, entre 5 a 14Hz. Para ello se utiliza el bloque *Sine Wave* de la librería *DSP System Toolbox* carpeta *sources*. Este bloque te permite crear una señal sinusoidal discreta de una frecuencia y amplitud determinada, los cuales se pueden modificar en la configuración del bloque.

En el modelo utiliza una señal sinusoidal discreta de amplitud unidad y de frecuencia entre 5Hz a 25Hz, esta señal se multiplica al tiempo de retraso, para así tener un tiempo de retraso variable. Por ejemplo, si se utiliza un tiempo de 1ms, este va a variar desde 0ms a 1ms con la frecuencia que determine la señal sinusoidal.

En el vibrato, la señal retardada no se le suma a la señal original. El modelo para crear este efecto se visualiza en la figura 26.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 44 de 120

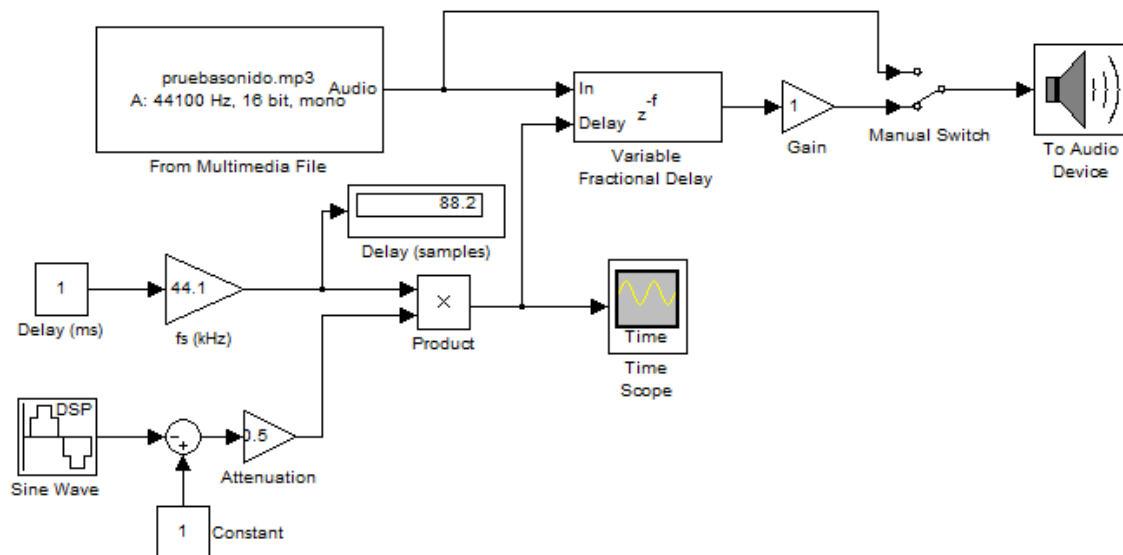




Figura 26: Modelo Simulink para el efecto vibrato.

En las pruebas realizadas, se ha ido variando la frecuencia moduladora entre 5 y 25Hz incrementándose 5Hz, y se ha utilizado 0.5ms y 1ms de delay, con una entrada de audio de voz.

A partir de 10Hz se aprecia como hay una vibración en la voz, asemejándose a un temblor. Conforme se aumenta la frecuencia este temblor aumenta. En la prueba con 1ms de delay se aprecia más el efecto, ya que la distancia entre muestras es mayor, sin embargo, si aumentamos el delay a 2ms, la distancia entre muestras es demasiado grande y provoca un efecto diferente al vibrato, cambiando el timbre de la voz y percibiendo unas fluctuaciones indeseables.

4.3.3.3. Modelo efecto flanger

En este efecto se utiliza el mismo modelo que el vibrato, pero añadiéndole a la señal retardada, la señal original. Para percibir este efecto la frecuencia de la señal sinusoidal es de 0,1Hz a 1Hz y el tiempo de retardo de 0ms a 15ms. Los bloques que se han utilizado son los mismos que el efecto anterior (Figura 27).

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 45 de 120

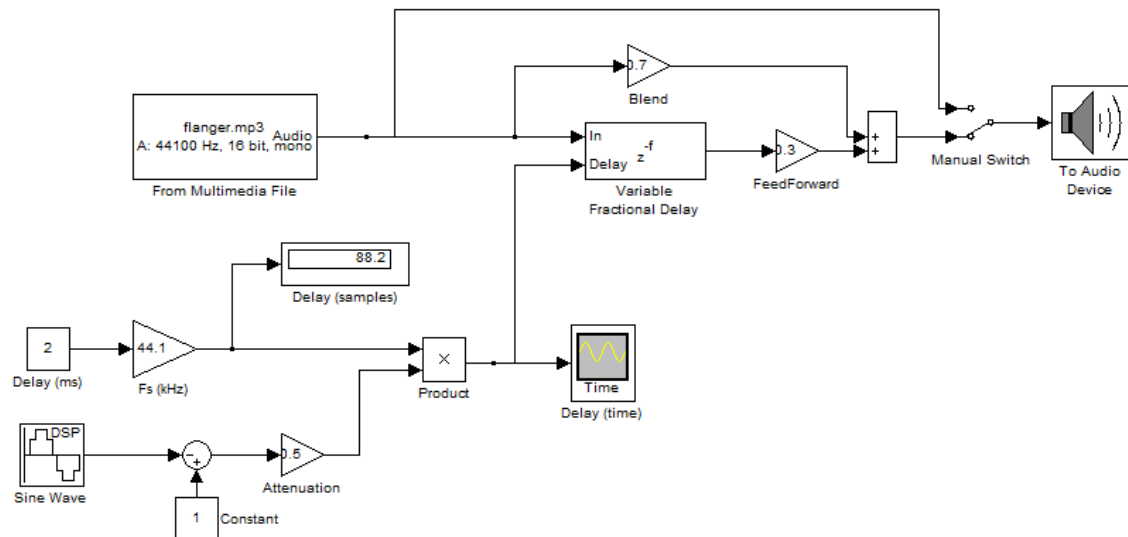


Figura 27: Modelo Simulink para el efecto flanger

En este filtro se percibe un realce significativo de las frecuencias altas, produciendo un sonido metálico oscilante molesto. Una forma de evitar esto, es añadir un filtro paso bajo de primer orden en un lazo de realimentación, para atenuar los realces de las frecuencias altas.

Una forma de implementar un filtro en frecuencia en Simulink es utilizando el bloque **Discrete Filter**, de la librería Simulink carpeta *Discrete* (Figura 28). En los parámetros de configuración del bloque se puede editar los ceros y polos del filtro.



Figura 28: Filtro paso bajo de primer orden

En Simulink se pueden crear subsistemas, seleccionando los bloques que pertenezcan a ese sistema y clicando con el botón derecho *create subsystem*.

De esta forma, se puede utilizar estos subsistemas en otros modelos y así tener un diseño más ordenado y simplificado.

Convirtiendo el filtro paso bajo en un subsistema y añadiéndolo en el modelo flanger, el diseño sería como el de la figura 29:

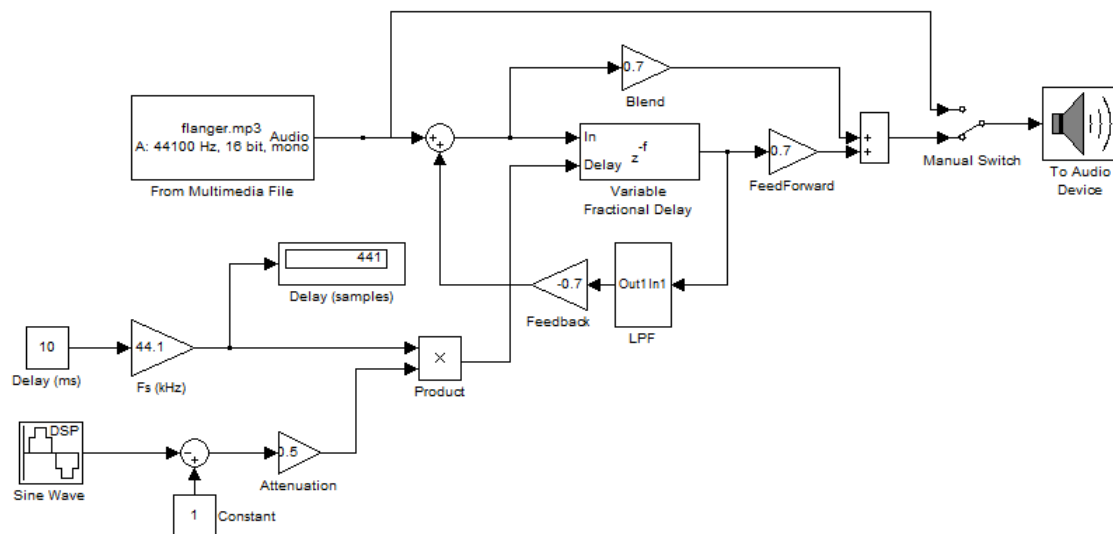


Figura 29: Diagrama Simulink efecto flanger con LPF de primer orden

En las pruebas realizadas se ha utilizado una señal moduladora de 1Hz y un tiempo delay de entre 1ms a 10ms, que se ha ido incrementado 2ms, en cada prueba.

Con un delay de 1ms el efecto flanger se aprecia más sutilmente, estando como principal la señal original. Conforme se va aumentando el retraso, el efecto coge más presencia, hasta 10ms. Si se aumenta más el delay, la señal original no se aprecia y produce sonidos indeseables, por ejemplo, en la prueba realizada con voz se aprecia cómo se cambia el timbre y además se añade un sonido extraño de fondo no deseado.

4.3.3.4. Modelo efecto chorus

Para el efecto chorus, se utiliza el mismo modelo que para el efecto flanger, pero con los parámetros de configuración diferentes. La principal diferencia es que para conseguir este efecto se tiene que modular el retraso con una señal de ruido aleatorio de baja frecuencia. Además, la frecuencia de esta señal tiene que ser lo suficientemente baja para que la velocidad de cambio del retardo no sea rápida, lo cual produciría efectos audibles indeseados.

La señal de ruido aleatorio para modular la señal, se implementa mediante el bloque **Random Source**. Dentro de la configuración del bloque se le asigna la unidad como amplitud máxima. A esta señal, para eliminar las frecuencias altas se le aplica un filtro paso bajo, con una frecuencia de paso, baja. Para implementar este filtro paso bajo, se ha utilizado el bloque **Lowpass Filter**. En

sus parámetros de configuración (Figura 30) te permite diseñar el filtro y visualizar su respuesta en frecuencia. Para el modelo, se ha utilizado un filtro IIR con banda de paso de 20Hz y banda de stop 31.5Hz.

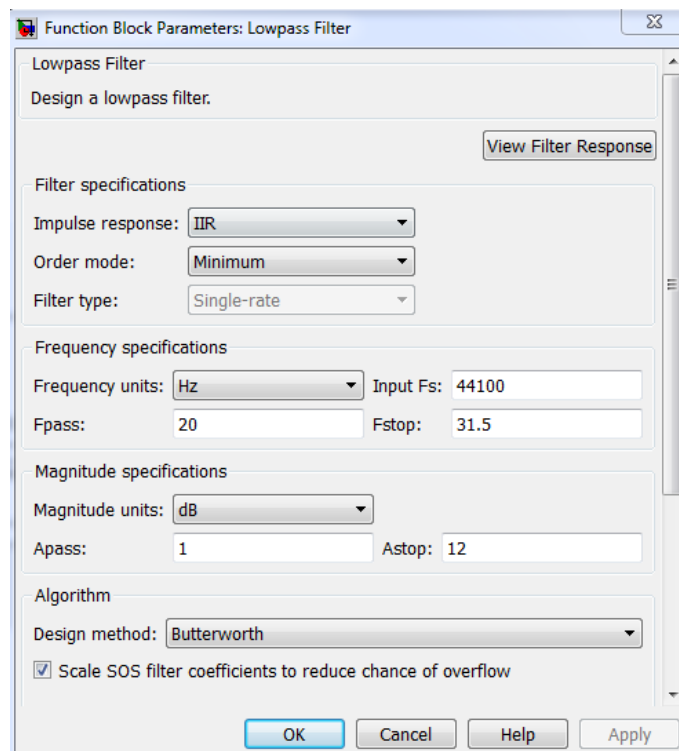




Figura 30: Configuración bloque Lowpass Filter

En este efecto el tiempo de retraso varía entre 10ms y 25ms. El modelo para el efecto chorus se muestra en la figura 31.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 48 de 120

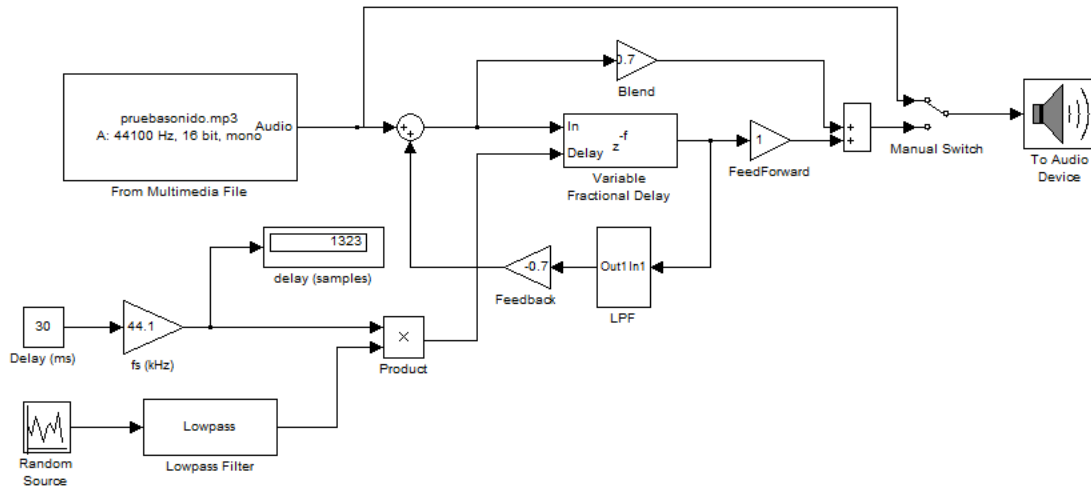


Figura 31: Diagrama Simulink para el efecto chorus

En las pruebas realizadas, se ha utilizado una onda aleatoria uniforme de máximo uno de amplitud. Los retrasos han ido de 5ms a 30ms, que se ha ido incrementado 5ms en cada prueba.



Con un delay de 5ms, se aprecia un chorus con menos densidad de ecos, conforme se aumenta el delay, esta densidad aumenta. En las pruebas realizadas con más de 30ms, el sonido tiene demasiada densidad de ecos y provoca unos rebotes indeseados en la señal. También se han hecho pruebas por debajo de los 5ms, pero en ese rango el efecto es casi inapreciable.

4.4. Flujo de diseño. Simulink-VHDL

Mediante el software Simulink se realiza diseños de bloques modelando un flujo de datos. Anteriormente se ha visto cómo implementar la teoría de los filtros en un modelo en Simulink, realizando pruebas en tiempo real y modificando los parámetros de configuración de cada filtro, comprobando así el comportamiento del efecto.

Una vez que se conoce la estructura y la configuración del filtro, este se puede implementar en un PLD, para ello, se tiene que programar el modelo creado en Simulink en lenguaje HDL.

Existen diferentes formas para programar el modelo en HDL. Una de ellas es utilizar una herramienta de Simulink, que genera un código HDL a partir de un modelo. Se encuentra en la pestaña *Tools / HDL Code Generation*. En la

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 49 de 120

ventana opciones, se permite elegir el tipo lenguaje, el directorio para guardar el código generado, además de una serie de parámetros. La ventana de configuración se muestra en la figura 32.

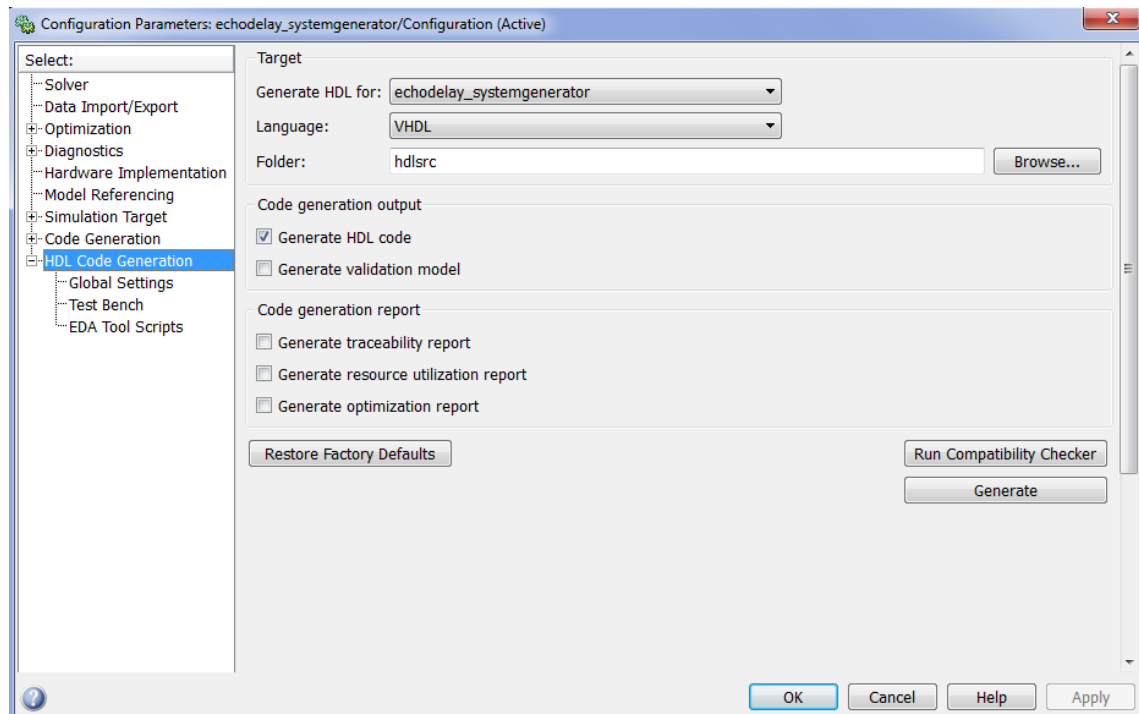




Figura 32: Configuración generación de código HDL

Alternativamente, en los últimos años el fabricante Xilinx ha creado conjunto a Mathworks, un plug-in para Simulink, para generar código HDL. El código generado es específico para las FPGAs del fabricante, logrando así diseños optimizados para cada placa. Este plug-in se llama **System Generator** y además incluye una librería de bloques específicos para su uso en Simulink.

Otra forma de crear el modelo en VHDL, es programar el comportamiento de cada bloque en ese lenguaje, utilizando un software de análisis y síntesis de diseños HDL. De esta forma, se realiza un código propio a partir de un diseño en Simulink.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 50 de 120



4.4.1. System Generator

System Generator es un plug-in del fabricante Xilinx, implementado en Simulink. Con este software se puede tener la funcionalidad de Simulink, para modelar diseños a nivel de sistemas y además proporcionar abstracciones de alto nivel del modelo para posteriormente integrarlo en una FPGA. System Generator tiene una librería propia de bloques que modelan sistemas hardware, que se pueden implementar en una FPGA, y estos se pueden relacionar con bloques Simulink.

La guía de usuario de System Generator indica que existen diferentes tipos de flujos de diseño, a los que está destinado. Los tipos de flujos de diseño son los siguientes:

- Exploración de algoritmos: este plug-in es particularmente útil para exploración de algoritmos, diseñados de prototipos y análisis de modelados. De esta forma, el diseño hardware implementado por bloques del System Generator se estimula con bloques de Simulink, para proporcionar simulaciones y análisis de resultados. Estos datos se pueden utilizar para estimar el coste del diseño hardware y comprobar si la funcionalidad es la deseada. Además, parte del sistema creado puede ser implantado en una FPGA, ahorrándonos tiempo en programación de código.
- Implementación de una parte de un diseño complejo: por ejemplo, podemos diseñar un componente mediante System Generator, para posteriormente incluirlo en el diseño del sistema VHDL.
- Implementación de un diseño completo: también se puede dar el caso de que System Generator proporcione todas las herramientas necesarias para crear el diseño completo. Para ello, el software traslada el diseño a código HDL y añade los archivos necesarios para poder procesar el diseño. Estos archivos se corresponden con el código VHDL del diseño, una señal de reloj para que habilita las señales del sistema, test bench (banco de pruebas) a un reloj y los resultados de Simulink para comprobar los resultados y archivos que permiten compilar el sistema System Generator en un software de desarrollo.

Para realizar un diseño Simulink con System Generator, como requerimiento se tiene que tener instalado el software Vivado, de Xilinx y posteriormente mediante una aplicación asignarle la versión Matlab que se tenga instalada.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 51 de 120

Para empezar un modelo Simulink, con bloques del System Generator, primero se tiene que añadir el bloque System Generator (figura 33).



Figura 33: Bloque System Generator.

Para añadir este bloque se abre la pestaña *Tools / Xilinx / BlockAdd*. En la configuración del bloque, te permite elegir la FPGA de Xilinx donde se va a implementar el sistema, el lenguaje hardware utilizado, la herramienta de síntesis utilizada, el directorio del proyecto y te da la opción de crear un testbench. En la figura 34, se puede visualizar la ventana de configuración:

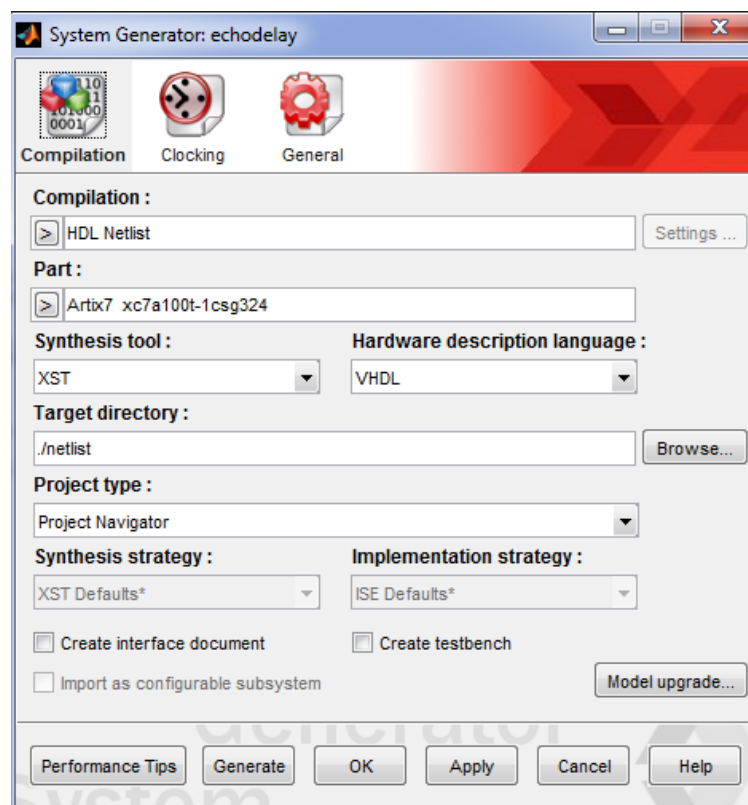




Figura 34: Ventana de configuración System Generator

En la pestaña clocking se puede seleccionar el reloj de la FPGA y el reloj de simulación de Simulink. Es recomendable que la configuración de Simulink tenga la asignación de periodo de muestreo automática, para que el sistema no tenga conflictos y evitar errores. En la figura 35 se observa, la ventana de configuración de los relojes.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 52 de 120

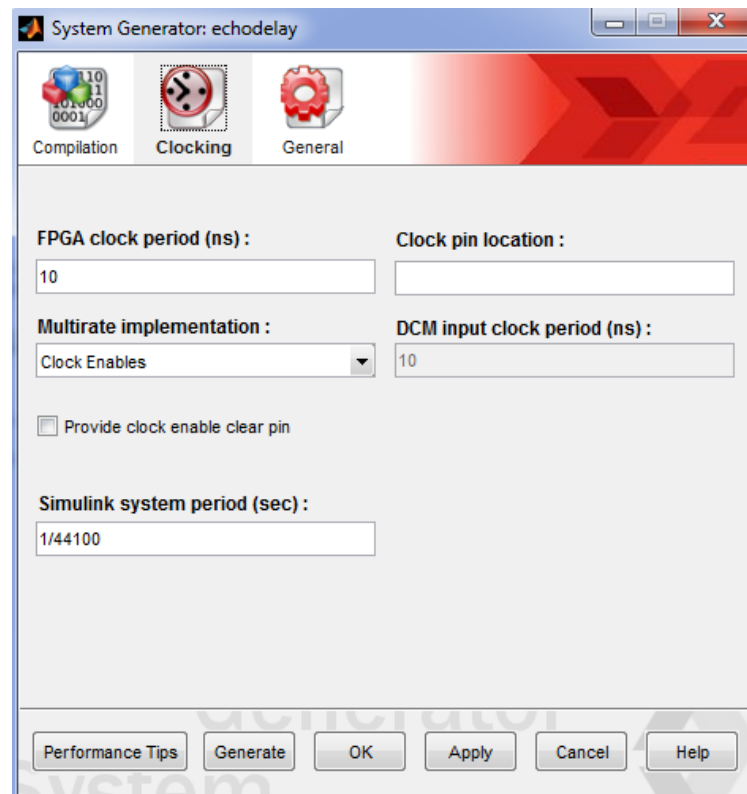


Figura 35: Configuración del reloj System Generator.

Una vez se tiene los parámetros de configuración definidos, se puede empezar a diseñar el sistema.



Para relacionar bloques Simulink con bloques System Generator, existen unos bloques especiales de entrada de datos y salida de datos llamados *gateways* (Figura 36).



Figura 36: Bloques gateways.

Estos bloques convierten el tipo de dato Simulink, a tipo de dato de bloques de Xilinx y posteriormente realiza la conversión contraria. Son imprescindibles para relacionar bloques Simulink, con bloques de Xilinx. Para señales de entrada continuas el bloque *Gateway In* muestrea la señal de entrada a una frecuencia especificada en la configuración del bloque.

En la figura 37, se muestra un ejemplo de diseño con un bloque de Memoria RAM doble, que realiza la misma función que el filtro eco. Para ello se escribe la muestra de entrada, en la dirección de memoria equivalente a la dirección

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 53 de 120

actual más delay muestras. Esta dirección se incrementa cada frecuencia de muestreo. La dirección de lectura se inicializa en 1 y se incrementa igualmente, accediendo a la muestra escrita, delays periodos de muestreo, equivalente a la dirección memoria de escritura. De esta forma se está accediendo a la muestra con un tiempo de retraso igual al delay por el periodo de muestreo.

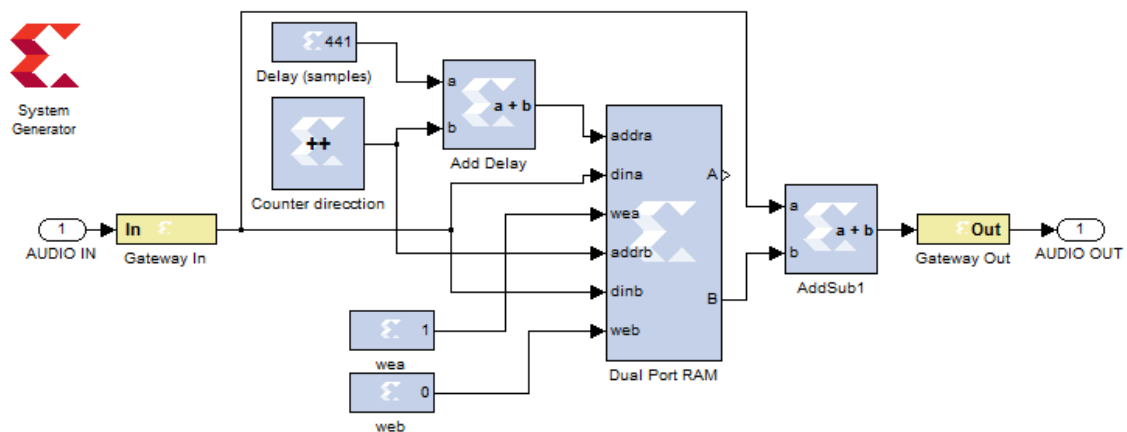




Figura 37: Modelo System Generator con RAM de doble puerto.



Como conclusión, se ha realizado un estudio de la utilidad de la herramienta, exponiendo los diferentes diseños de uso y explicando la configuración básica de un modelo.

Se puede decir que la herramienta es muy útil para ayudar en el diseño hardware, ya que permite la utilidad de todas las herramientas de Simulink y de Matlab, en cambio, solamente es aplicable para FPGAs de Xilinx, siendo el código generado específico para esas tecnologías.

El método de generación de código HDL, en cambio, es muy genérico y puede ser una tarea compleja realizar posteriormente la síntesis e implementación en la FPGA.

En este proyecto no se ha escogido ninguno de los dos métodos anteriores, para generar el código VHDL, ya que se ha preferido realizar una programación de cada bloque del modelo, para lograr así un código genérico válido para cualquier FPGA, que tenga las especificaciones físicas requeridas por el diseño.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 54 de 120

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 55 de 120

5.DISEÑO DE CIRCUITOS

En esta sección se desarrolla el diseño de los diferentes módulos VHDL creados para el sistema de procesamiento de señales de audio y las simulaciones y pruebas realizadas mediante Test Bench de cada uno de ellos.

También se exponen diferentes alternativas de diseño para el almacenamiento de muestras, comprobando el consumo de recursos y la utilidad de cada alternativa.

Por último, se exponen las diferentes formas elegidas de interacción con la placa de desarrollo y la construcción de cada filtro.

5.1. Interfaz del convertidor analógico digital PmodAD1

El convertidor PmodAD1 utiliza una interfaz parecida a la SPI (*Serial Peripheral Interface*) con la diferencia de que las líneas de transmisión de datos (MOSI y MISO) están diseñadas para operar exclusivamente como salidas, haciéndose ambas líneas de transmisión de datos Master-In-Slave-Out.

El dispositivo proporciona 12 bits de información durante los 16 ciclos de reloj con los 4 bits más significativos, obteniendo así una muestra de 16 bit pero con 12 bit de información.

El PmodAd1 tiene 6 pines de entrada, el primero de selección del chip, el segundo y el tercero de entrada de datos de los dos canales, el cuarto de frecuencia de reloj, el quinto de toma de tierra y el sexto de toma de alimentación de 3.3V o 5V (Figura 38 y 39).

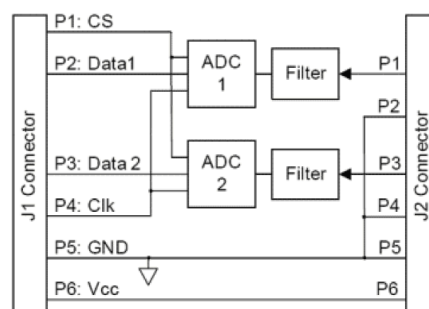


Figura 38: Diagrama del circuito PmodAD1

Header J1		
Pin	Signal	Description
1	CS	Chip Select
2	D0	Input Data 1
3	D1	Input Data 2
4	SCK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

Header J2		
Pin	Signal	Description
1	A0	Input Data 1
2	GND	Power Supply Ground
3	A1	Input Data 2
4	GND	Power Supply Ground
5	GND	Power Supply Ground
6	VCC	Positive Power Supply

Figura 39: Descripción de los pines del PmodAD1

Para modelar el comportamiento de esta interfaz, se ha creado la entidad **pmodad1**. Esta entidad está formada por cuatro puertos de entrada y cuatro puertos de salida:

```



entity pmodad1 is
    Port ( clk : in   STD_LOGIC;
           reset : in  STD_LOGIC;
           rd : in    STD_LOGIC;
           miso : in   STD_LOGIC;
           ad1_clk : out STD_LOGIC;
           cs : out    STD_LOGIC;
           x : out    STD_LOGIC_VECTOR (11 downto 0);
           done : out  STD_LOGIC);
end pmodad1;

```

Los puertos de entrada son los siguientes:

- **clk** (*clock*): señal de reloj a la que operan todos los procesos síncronos del sistema.
- **reset**: bit que cuando tiene valor 1, borra las variables del sistema y lo inicializa.
- **rd** (*read*): señal lógica que cuando tiene valor 1 indica, que se van a activar los convertidores en el siguiente ciclo, para empezar a muestrear.
- **miso**: bit de datos procedente del ADC.

Los puertos de salida son los siguientes:



			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 57 de 120

- **ad1_clk**: frecuencia de reloj para los convertidores.
- **cs** (*chip select*): señal lógica que cuando tiene valor 1 activa los dos ADC.
- **x**: señal de datos de 12 bits.
- **done**: bandera que cuando tiene valor 1, indica que se ha obtenido una muestra de 12 bits.

La arquitectura de esta entidad la forman tres procesos que implementan una máquina de cuatro estados. El proceso **adc_state_control** modela el comportamiento de la máquina de estados IDLE_ADC, SETUP_ADC, START_ADC, FINISH_READ, su funcionamiento es el siguiente:

- **IDLE_ADC**: estado que espera a que la señal **rd='1'**, para empezar a coger muestras.
- **SETUP_ADC**: activa la señal **cs** (**cs='1'**), indicando al convertidor que se va a empezar a muestrear.
- **START_ADC**: en este estado se permanece hasta que el contador **ADC_REGISTER_CNT=16**, es decir, durante 16 ciclos de reloj en los cuales se obtiene la muestra de 16 bits. Durante este estado, en el proceso **adc_register_ctrl** se guarda el bit de datos del puerto de entrada miso en el vector **ADC_REGISTER** y se desplaza un bit a la izquierda. Cuando el estado observa que se han llegado a los 16 ciclos, activa la bandera **X_FLAG** (**X_FLAG='1'**), para indicar al proceso **adc_register_ctrl** que guarde los primeros 12 bit del vector **ADC_REGISTER**, en el vector de salida **x**.
- **FINISH_READ**: Una vez se obtiene la muestra de 12 bits de datos, se pasa a este estado. En este estado se activa la **AD_SAMPLE** (**AD_SAMPLE='1'**) para indicar al proceso **adc_register_ctrl** que active la señal de salida **done** (**done='1'**), la cual indica que se ha obtenido una muestra de 12 bits.

Por último, se utiliza el proceso **adc_state_change** para implementar la señal de reloj **clk** a la máquina de estado, realizando así una máquina de estados síncrona (Figura 40).

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 58 de 120

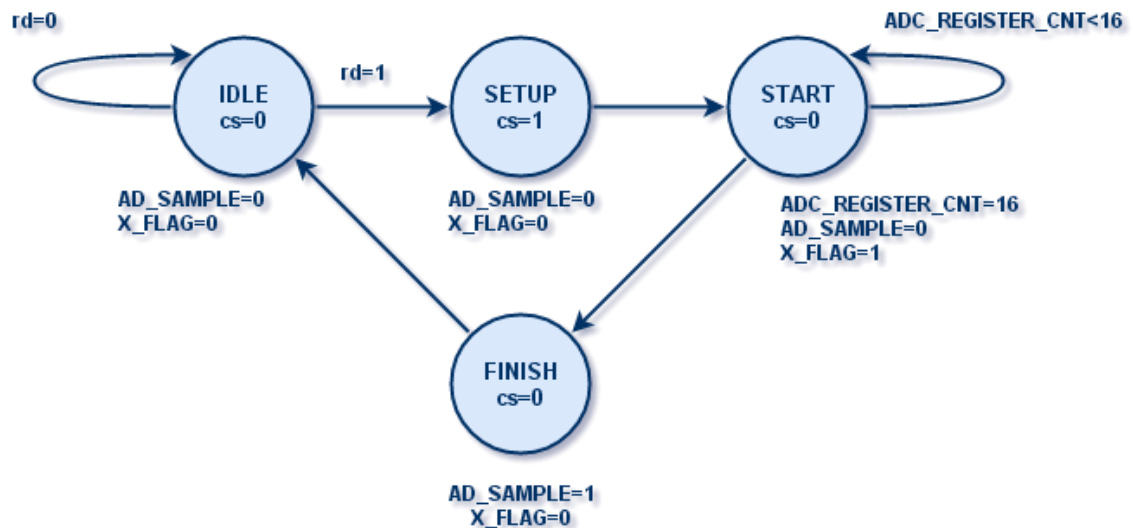


Figura 40: Máquina de estados del proceso `adc_state_control` (PmodAD1)

Para comprobar la funcionalidad del módulo, se ha creado el test bench **pmodad1_tb**. Este test simula las entradas del pmodad1 (clk, reset, rd, y miso). Primero se activa el reset, para inicializar el sistema durante 300ns. Posteriormente se activa la entrada rd, para que mantenga activos los convertidores.

La entrada miso se simula mediante un bucle while, dentro del bucle, cada vez que se activa la salida cs (*chip select*), se almacena en el registro `adc_register` una muestra de 16bit, correspondiente a 128 muestras de una señal senoidal almacenada previamente. Seguidamente, se le asigna a la entrada miso el bit más significativo de esa muestra, la cual se desplaza un bit cada ciclo de reloj del `ad1_clk` (reloj del bit) hasta que termina de asignarle todos los bits a la entrada miso.

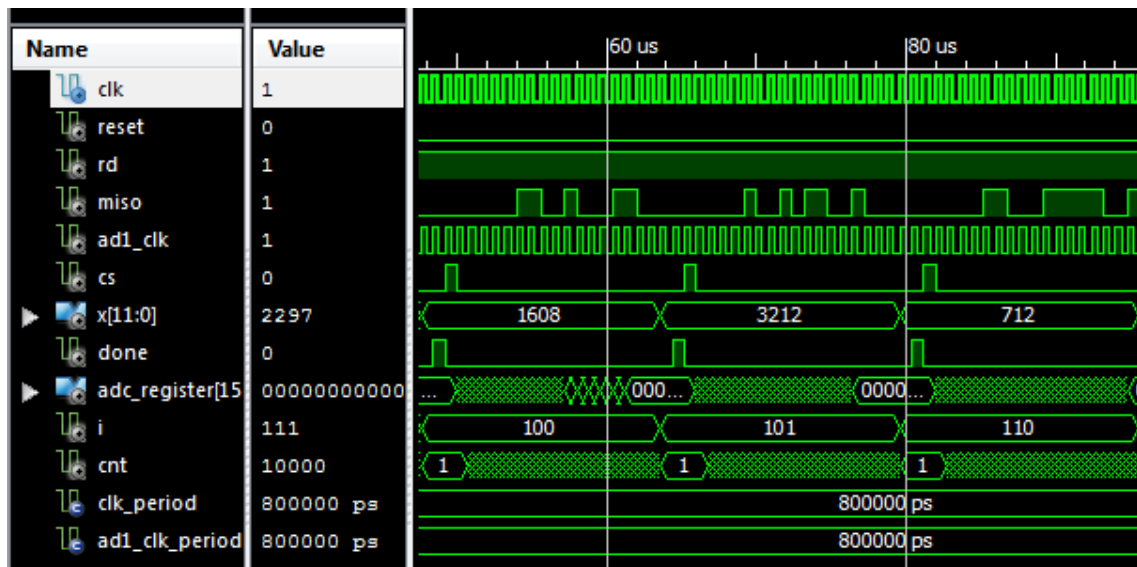


Figura 41: Simulación PmodAD1

En la figura 41 se puede observar el comportamiento del pmodAD1. Primero se activa la señal cs, indicando que se va a muestrear. Seguidamente se obtiene la muestra de 16bit por la entrada miso. Por ejemplo, en el primer periodo se obtiene la muestra “0000110010001100” (3212 en decimal), que posteriormente se lee y se asigna a la señal x. Finalmente se activa la salida done, indicando que se ha obtenido una muestra.



5.2. Interfaz del convertidor analógico digital PmodI²S

Para la conversión digital analógica se utiliza el PmodI²S, el cual utiliza protocolo I²S (*Integrated Interchip Sound*). Este protocolo necesita varias frecuencias de reloj diferentes para su correcto funcionamiento.

El convertidor es capaz de tratar muestras de 16 bits hasta 24 bits y convertirlas en una señal analógica de audio estéreo, es decir, de dos canales.

El Pmod tiene 6 pines de entrada, los cuales corresponden a las siguientes señales:

- MCLK (*Master Clock*): es el reloj más rápido del sistema y es clave para sincronizar los demás relojes.
- LRCK (*Left-right Clock*): equivale a la frecuencia de muestreo donde se van a transmitir los datos de audio al canal izquierdo y al canal derecho.
- SCK (*Serial Clock*): es la frecuencia de reloj a la que se transmite cada bit.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 60 de 120

- SDIN (*Serial Data Input*): corresponde al dato de salida.
- GND: toma de tierra del Pmod.
- VCC: alimentación de 3.3V.

Dependiendo del tamaño del dato y de la frecuencia de muestreo del diseño, se elige un modo de funcionamiento y una frecuencia específica del *Master Clock*. El datasheet del fabricante proporciona esta relación (Figura 42 y 43):

Internal SCK Mode	External SCK Mode
16-bit data and SCK = 32*Fs if MCLK/LRCK = 1024, 512, 256, 128, or 64	Up to 24-bit data with data valid on the rising edge of SCK
Up to 24-bit data and SCK = 48*Fs if MCLK/LRCK = 768, 384, 192, or 96	
Up to 24-bit data and SCK = 72*Fs if MCLK/LRCK = 1152	



Figura 42: MCLK/LRCK ratio

LRCK (kHz)	MCLK (MHz)									
	64x	96x	128x	192x	256x	384x	512x	768x	1024x	1152x
32	-	-	-	-	8.1920	12.2880	-	-	32.7680	36.8640
44.1	-	-	-	-	11.2896	16.9344	22.5792	33.8680	45.1580	-
48	-	-	-	-	12.2880	18.4320	24.5760	36.8640	49.1520	-
64	-	-	8.9120	12.2880	-	-	32.7680	49.1520	-	-
88.2	-	-	11.2896	16.9344	22.5792	33.8680	-	-	-	-
96	-	-	12.2880	18.4320	24.5760	36.8640	-	-	-	-
128	8.1920	12.2880	-	-	32.7680	49.1520	-	-	-	-
176.4	11.2896	16.9344	22.5792	33.8680	-	-	-	-	-	-
192	12.2880	18.4320	24.5760	36.8640	-	-	-	-	-	-
Mode	QSM				DSM			SSM		

Figura 43: Frecuencia de muestreo y frecuencia MCLK

En el diseño se trabaja con datos de 16 bits debido a que es el mínimo posible en este dispositivo, ya que las muestras de datos son de 12 bits. Como los datos son de audio, se utiliza la frecuencia de muestreo 44.1kHz. Con esto, se puede ver como la relación entre la frecuencia MCLK es 256 veces la frecuencia LRCK.

En el siguiente diagrama temporal de la figura 44 se observa el comportamiento del circuito. Cuando LRCK='0', transmite 16 bits de datos al canal izquierdo, es decir, 16 periodos del reloj SCK y cuando LRCK='1', transmite 16 bits de datos al canal derecho.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 61 de 120

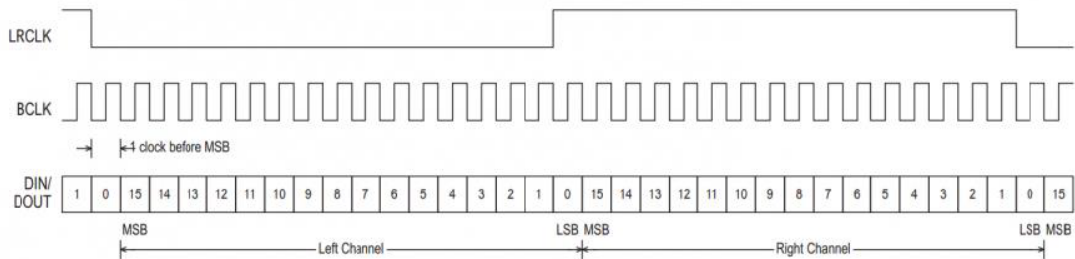


Figura 44: Diagrama temporal del PmodPS

Se observa como en un ciclo de reloj de LRCK hay 32 ciclos de SCK, por lo tanto, si se tiene una señal de muestreo LRCK de 44.1kHz de datos de 16 bits, las señales de reloj MCLK y SCK equivalen a:

$$MCLK = LRCK \cdot 256 = 44.1kHz \cdot 256 = 11.2896MHz$$

$$SCK = LRCK \cdot 32 = \frac{MCLK}{8} = 1.4112MHz$$

Este comportamiento se modela en la entidad i2s_mod, la cual tiene 3 puertos de entrada y 4 puertos de salida.

```

entity i2s_mod is
  Port ( clk : in   STD_LOGIC;
         reset : in   STD_LOGIC;
         data_in : in   STD_LOGIC_VECTOR (15 downto 0);
         mclk : out   STD_LOGIC;
         lrck : out   STD_LOGIC;
         sclk : out   STD_LOGIC;
         sdin : out   STD_LOGIC);
end i2s_mod;

```

La arquitectura de esta entidad utiliza dos procesos para implementar el funcionamiento. En el primer proceso resta uno, a un contador entero de 8 bits (255) cada flanco de subida del reloj MCLK (el cual es equivalente al CLK).

Para obtener el reloj SCK, este adquiere el tercer bit del contador, es decir, es como si dividiera entre 8 el reloj MCLK y para el reloj LRCK, adquiere el octavo bit del contador para obtener una señal 256 más lenta. De esta forma, se tienen todos los relojes del sistema con la relación especificada.

En otro proceso, una variable temporal adquiere el dato de entrada de 16 bits en cada periodo del LRCK (frecuencia de muestreo) y este dato lo

desplaza un bit a la izquierda cada flanco de bajada de la frecuencia de transmisión de bit SCK.

Por último, cada último bit (bit 16 del dato temporal) se asigna al bit de salida SDIN, cada flanco de bajada de la frecuencia SCK. De esta forma, se representan los 16 bits del dato en serie, dentro de la frecuencia de muestreo LRCK.

Para la comprobación de este módulo, el test bench creado simula la entrada de una muestra de 16 bit, cada periodo de reloj LRCK. Observando que los bits de la entrada data_in, corresponden con la misma posición de la salida sdin, según el diagrama temporal de la figura 44.

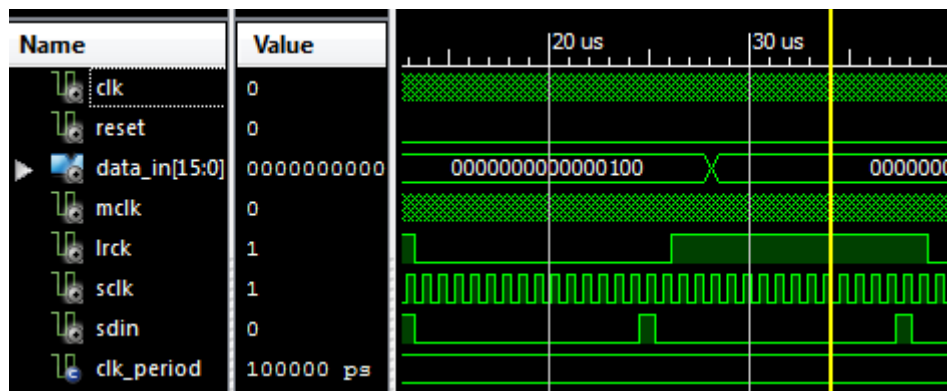




Figura 45: Simulación PmodPS

En la figura 45 se puede observar como el bit 2 de la entrada data_in, corresponde con el bit 2 de la salida sdin, tanto en el canal izquierdo como en el derecho.

5.3. Generación de relojes del sistema

Para el sistema se escoge una frecuencia de muestreo de 44.1kHz, que es la frecuencia de muestreo estándar de audio más utilizada y una muestra de 16 bit con 12 bit de información.

Ya se ha comprobado en el punto anterior la necesidad de la creación de varias frecuencias de reloj relacionadas entre sí, dependiendo de la frecuencia de muestreo y del tamaño del dato, es decir, se tiene una frecuencia de muestreo, una frecuencia de bit de datos y una frecuencia master. Esta relación es la misma que la demostrada en el convertidor DAC. La frecuencia de bits de datos es la misma en el ADC y en el DAC, ya que los dos tienen muestras del mismo tamaño, 16 bits.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 63 de 120

A partir de la frecuencia máster (MCLK), se obtiene la frecuencia de muestreo y la frecuencia de bits de datos, mediante un proceso que utiliza un contador. Por lo tanto, solamente hace falta obtener la frecuencia máster.

La frecuencia máster (MCLK), la se obtiene a partir de la frecuencia global del sistema, que es de 50MHz. Para llegar a una frecuencia de muestreo de 44.1kHz se sabe que la frecuencia máster tiene que ser 11.2896MHz (Figura 36). Una forma de acercarse a ese valor, es dividir entre 5 la frecuencia global, obteniendo así una frecuencia de 10MHz. Implementado la relación entre frecuencias, se obtiene una frecuencia de muestreo de:

$$mclk = \frac{clk}{5} = \frac{50MHz}{5} = 10MHz$$

$$ad1clk = sclk = \frac{mclk}{8} = 1.25MHz$$

$$sample_clk = lrck = \frac{mclk}{256} = 39.0635kHz$$

Se puede comprobar cómo se aproxima a la frecuencia de muestreo de 44.1kHz, pero sin ser muy precisos. Una forma de ampliar la precisión, es multiplicar por un número estero y dividir por otro número entero la frecuencia global del sistema. Realizando cálculos, si se multiplica por 7 y se divide por 31 la frecuencia global, se obtiene una precisión aceptable:

$$mclk = \frac{clk \cdot 7}{31} = 11.2903MHz$$

$$ad1clk = sclk = \frac{mclk}{8} = 1.4113MHz$$

$$sample_clk = lrck = \frac{mclk}{256} = 44.1028kHz$$

Se comprueba como de esta forma se ha obtenido una frecuencia de muestreo de 44.1kHz.

Esta generación de frecuencias de reloj la implementa la entidad **gen_reloj**, la cual ha sido generada por un *IP Core Generator*, una herramienta de ISE que genera código de entidades con una funcionalidad concreta con una determinada arquitectura, la cual se programa con unas ventanas de configuración que asisten al diseñador. Esta herramienta, proporciona facilidades al diseñador a la hora de implementar código. Este utilitario genera varios programas, uno de ellos con extensión .xco, el cual contiene la información necesaria para construir el componente. También genera un

archivo .vhd que solamente puede utilizarse en simulaciones, este archivo en proceso de síntesis se ignora y tampoco se puede instanciar un componente con ese archivo.

Para añadir un IP Core al diseño, se añade una fuente nueva dentro del módulo que se desee y se selecciona la opción de añadir el tipo IP (*CORE Generator & Architecture Wizard*).

En este caso, para generar frecuencias a partir de una frecuencia máster se ha escogido el *IP Core Clocking Wizard*, el cual se encuentra en el directorio *FPGA Features and Design / Clocking / Clocking Wizard*.

En la ventana de configuración del *Clocking Wizard*, se dan una serie de parámetros, los cuales se eligen en función del diseño. En la primera hoja se selecciona el reloj primario de 100MHz, que corresponde a la frecuencia del puerto de entrada del sistema. En la segunda hoja, se permiten generar diferentes frecuencias en función de la frecuencia primaria. En la pestaña *Output Freq. (MHz) / Requested*, se indica la frecuencia deseada por el usuario y luego en la pestaña Actual, se indica la permitida que se acerque más a la solicitada. Por ejemplo, si se solicita una frecuencia de 11.2896MHz para el reloj MCLK, el sistema los aproximará a 11.290MHz, que sería la frecuencia permitida (Figura 46).

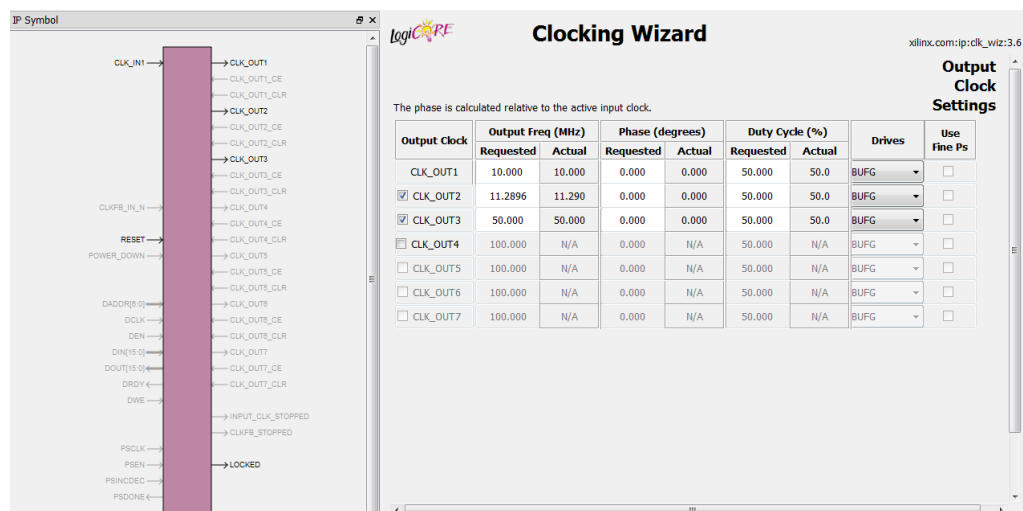




Figura 46: Ventana IP Core Clocking Wizard

Una vez configurados los parámetros de configuración del IP Core, se procede a generar el código, dándole en la pestaña *Generate*.

El código generado se trata de la entidad **gen_reloj** que tiene la siguiente estructura:

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 65 de 120

```

entity gen_reloj is
port
  (-- Clock in ports
  CLK_IN1          : in      std_logic;
  -- Clock out ports
  CLK_OUT1         : out     std_logic;
  CLK_OUT2         : out     std_logic;
  CLK_OUT3         : out     std_logic;
  -- Status and control signals
  RESET           : in      std_logic;
  LOCKED          : out     std_logic
  );
end gen_reloj;

```

El puerto de entrada corresponde con la señal de reloj de 100MHz, correspondiente a la del pin de entrada. Los puertos de salida CLK_OUT1 y CLK_OUT2 se les asignan a la señal MCLK, los cuales generan la señal de reloj de 10MHz y 11.290MHz para realizar pruebas con esos dos valores. El puerto de salida CLK_OUT3 genera la frecuencia de reloj de 50MHz la cual va a ser utilizada como señal de reloj del sistema.

5.4. Sistema DSP

Este módulo se encarga de proporcionar comunicación entre el convertidor digital, el procesador digital y el convertidor analógico. Por lo tanto, está formado por los componentes **pmodad1**, **i2s_mod**, correspondientes a los convertidores, el componente **gen_reloj**, que genera las frecuencias de reloj del sistema y por componentes adicionales requeridos para realizar los filtros, en el procesamiento digital, y requeridos para la interacción con la placa de desarrollo.



En definitiva, la estructura básica de este módulo define un sistema de procesamiento digital de señales, al cual se le puede añadir diferentes módulos para definir la funcionalidad del procesamiento digital y así diseñar un filtro de audio específico en función de las características deseadas.

La entidad que implementa este módulo se llama **sistema_dsp** y en su estructura básica tiene los siguientes puertos:

```

entity sistema_dsp is
  Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;

```


			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 66 de 120

```

    miso : in  STD_LOGIC;
    adl_clk : out STD_LOGIC;
    cs : out  STD_LOGIC;
    mclk : out STD_LOGIC;
    sclk : out STD_LOGIC;
    lrck : out STD_LOGIC;
    sdin : out STD_LOGIC);
end sistema_dsp;



```

Los puestos de entrada clk, reset y miso, corresponden con la frecuencia de reloj de 100MHz del pin de la placa, la señal reset del sistema y los datos de entrada del pmodad1. Los puertos de salida corresponden, con las frecuencias de reloj de los convertidores, la selección de chip del pmodad1 y la salida del pmodi2s.

La arquitectura básica de esta entidad utiliza dos procesos que realizan la función de comunicación entre los componentes del sistema. El primer proceso se trata de una máquina de dos estados. Estos dos estados son:

- **SAMPLE_IO:** En este estado se activa la bandera WT_FLAG, la cual indica escritura de muestras. También se activa la bandera RD_FLAG la cual indica que el pmodad1 va a leer muestras. Una vez que llega una muestra completa de 16bit, lo cual nos lo indica la señal done del pmodad1 (bandera SAMPLE_DONE_FLAG), se pasa al siguiente estado.
- **HOLD_IO:** En este estado se mantiene activada la bandera de escritura WT_FLAG y se desactiva la bandera RD_FLAG poniéndola a cero. No se pasa al estado SAMPLE_IO, hasta que la operación NEW_SAMPLE and GET_SAMPLE=1 se cumpla, definida en el proceso **sampler_proc**. De esta forma se mantiene la muestra hasta el siguiente periodo del reloj de muestreo, donde se obtiene una nueva muestra.

El proceso **sample_proc** activa la señal GET_SAMPLE, cada flanco de subida del reloj de muestreo y la señal NEW_SAMPLE (activa en bajo) la activa cuando se obtiene una muestra, es decir, cuando la bandera SAMPLE_DONE_FLAG está activa. De esta forma la operación NEW_SAMPLE and GET_SAMPLE siempre será 1 cada flanco de subida del reloj de muestreo hasta la activación de la bandera SAMPLE_DONE_FLAG. Este proceso también añade la frecuencia de reloj clk0 a la máquina de estados anterior, realizando así una máquina de estados síncrona (Figura 47).

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 67 de 120

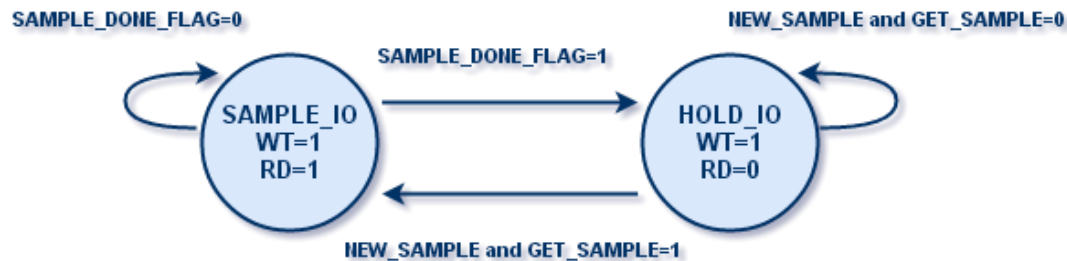


Figura 47: Máquina de estados del proceso *sampler_fsm* (*sistema_dsp*)

Para simular el comportamiento del módulo, se ha utilizado el mismo test bench que se utilizó para el *pmodad1*, observando que la interfaz entre los módulos es correcta, y que para una muestra de entrada del *pmodad1*, se obtiene una muestra de salida en el *pmodi2s*.

5.5. Bloques de almacenamiento de muestras

Todos los filtros diseñados en este proyecto están basados en retardadores de señales. Un retardador de señal se consigue almacenando muestras de datos de un tamaño de bits definido en algún dispositivo de almacenamiento, como por ejemplo una memoria.

Para saber el tamaño de la memoria que se necesita, se tiene que saber el tiempo de retraso máximo que se quiere conseguir, la frecuencia de muestreo, para así saber el número máximo de muestras que se va a almacenar, y el tamaño de cada muestra.

En el sistema se sabe que trabajamos con muestras de 12bits de datos, con una frecuencia de muestreo de 44.1kHz y el mayor retraso que se desea conseguir es de 200ms en el filtro eco. Por lo tanto, se puede obtener el tamaño de la memoria de la siguiente forma:

$$M = \tau \cdot fm = 200ms \cdot 44.1kHz = 8820 \text{ muestras de 12bits}$$

$$\text{Tamaño memoria} = 8820 \text{ muestras} \cdot 12bits = 105840bits \approx 103.34Kbits$$

Una vez comprobada la memoria que se necesita para el sistema, se observa que tipos de almacenamientos proporcionan la FPGA y la placa de desarrollo utilizada y si cumplen con los requisitos de nuestro diseño.

La FPGA Artix 7 proporciona tres opciones de almacenamiento, por registros, por RAM (*Random Access Memory*) distribuida y por bloque de RAM (BRAM). Según el datasheet del fabricante Xilinx, el tipo XCTA100T tiene una memoria RAM máxima distribuida de 1188Kb, un bloque de RAM de 4860Kb como máximo y 594Kb disponibles mediante registros. Por lo tanto, se observa que las tres formas de almacenamiento cumplen con los requisitos.

También la placa de desarrollo Nexys 4 DDR nos proporciona una memoria externa SRAM de 128MiB DDR (*Double Data Rate*), la cual la se podría utilizar igualmente. En la figura 48, se puede visualizar el resumen de las características de cada tipo de FPGA de la familia Artix 7 y en la figura 49 los recursos de los bloques lógicos configurables (CLBs)

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices ⁽²⁾	Block RAM Blocks ⁽³⁾			CMTs ⁽⁴⁾	PCIe ⁽⁵⁾	GTPs	XADC Blocks	Total I/O Banks ⁽⁶⁾	Max User I/O ⁽⁷⁾
		Slices ⁽¹⁾	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XC7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XC7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	250
XC7A75T	75,520	11,800	892	180	210	105	3,780	6	1	8	1	6	300
XC7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	8	1	6	300
XC7A200T	215,360	33,650	2,888	740	730	365	13,140	10	1	16	1	10	500

Notes:

- Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
- Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.
- Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
- Each CMT contains one MMCM and one PLL.
- Artix-7 FPGA Interface Blocks for PCI Express support up to x4 Gen 2.
- Does not include configuration Bank 0.
- This number does not include GTP transceivers.

Figura 48: Características por tipo de dispositivo (Artix-7 FPGA)

Device	Slices ⁽¹⁾	SLICEL	SLICEM	6-input LUTs	Distributed RAM (Kb)	Shift Register (Kb)	Flip-Flops
7A12T	2,000 ⁽²⁾	1,316	684	8,000	171	86	16,000
7A15T	2,600 ⁽²⁾	1,800	800	10,400	200	100	20,800
7A25T	3,650	2,400	1,250	14,600	313	156	29,200
7A35T	5,200 ⁽²⁾	3,600	1,600	20,800	400	200	41,600
7A50T	8,150	5,750	2,400	32,600	600	300	65,200
7A75T	11,800 ⁽²⁾	8,232	3,568	47,200	892	446	94,400
7A100T	15,850	11,100	4,750	63,400	1,188	594	126,800
7A200T	33,650	22,100	11,550	134,600	2,888	1,444	269,200

Notes:

- Each 7 series FPGA slice contains four LUTs and eight flip-flops; only SLICEMs can use their LUTs as distributed RAM or SRLs.
- Number of slices corresponding to the number of LUTs and flip-flops supported in the device.

Figura 49: Recursos del CLB (Artix-7 FPGA)

5.5.1. RAM distribuida

La RAM distribuida se construye con los LUTs, lo cual significa que compite con los recursos de la lógica, y su uso está restringido a aplicaciones que requieran poco almacenamiento. Cada CLB está formado por dos SLICES uno tipo M (SLICEM) y otro tipo L (SLICEL), los de tipo M pueden implementar lógica o memoria y los de tipo L sólo implementan lógica, por eso, con casi 101440 celdas lógicas sólo hay disponible 1118Kbits. En la figura 50, se puede observar la configuración de un CLB.

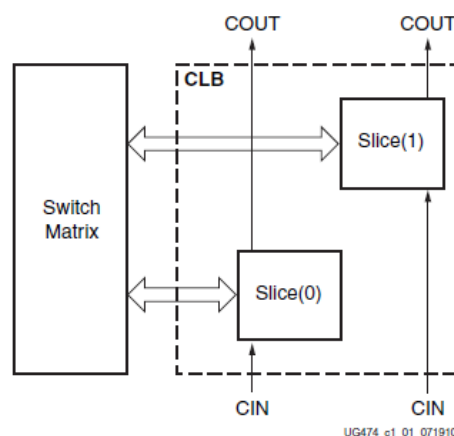


Figura 50: Slices de un CLB

Cada SLICE contiene cuatro LUTs y ocho flip-flops, lo que permite tener por cada CLBs una memoria distribuida de 256bits (figura 51).

Slices	LUTs	Flip-Flops	Arithmetic and Carry Chains	Distributed RAM ⁽¹⁾	Shift Registers ⁽¹⁾
2	8	16	2	256 bits	128 bits

Notes:

1. SLICEM only, SLICEL does not have distributed RAM or shift registers.

Figura 51: Logic Resources In One CLB

La memoria RAM distribuida puede ser de tipo single o dual port. Las de tipo single port, tienen la misma dirección para la escritura síncrona y la lectura asíncrona, es decir, escribe y lee en memoria no simultáneamente. Las de tipo dual port, tienen una dirección para la escritura síncrona y lectura asíncrona y otra dirección para solamente lectura asíncrona, es decir, se puede escribir y leer en memoria simultáneamente. Para implementar el tipo dual port, se usan dos LUTs del SLICEM. En la figura 52 y 53 se muestra un ejemplo de RAM distribuida de uno y dos puertos.

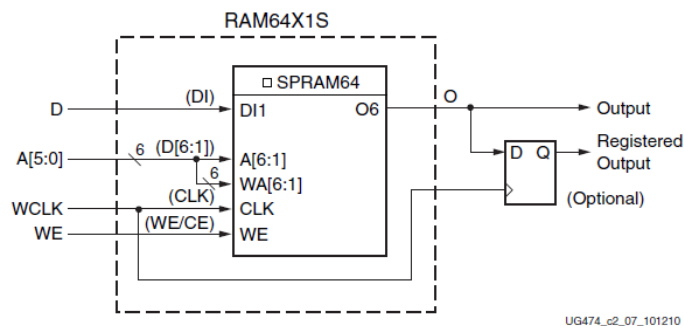


Figura 52: Ejemplo de RAM distribuida de un solo puerto (RAM64X1S)

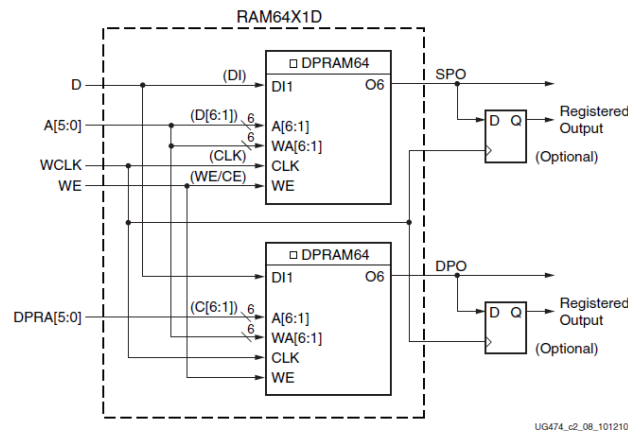


Figura 53: Ejemplo de RAM distribuida de doble puerto (RAM64X1D)



5.5.2. Bloque de RAM (BRAM)

Es un bloque especial separado de las celdas lógicas. En las series 7 de FPGAs de Xilinx, cada bloque puede almacenar de 36Kbits de datos y puede ser configurado como dos bloques independientes de 18Kbits o uno de 36Kbits. Cada bloque RAM puede tener diferentes configuraciones, desde 64K x 1 hasta 1K x 36 o en el caso de bloques de 18Kbits, configuraciones desde 16K x 1, hasta 1K x 18. Los bloques de RAM están organizados por columnas, la FPGA utilizada tiene 4 columnas con 40 bloques de 36Kb. También, como se ve en la figura 40, tiene 270 bloques de 18Kb o 135 bloques de 36Kb, lo cual puede llegar a almacenar un máximo de 4860Kb.

Cada bloque de RAM de 36Kb, consiste en un área de 36Kb de almacenamiento y dos puertas de acceso totalmente independientes (puerto A y B), e igual ocurre con bloque RAM de 18Kb. La estructura es totalmente simétrica y ambos puertos son intercambiables.

Ambos puertos soportan la escritura y la lectura. Cada puerto tiene sus propias señales: address, data in, data out, clock, clock enable, y wire enable. Las operaciones de lectura y escritura son síncronas y requieren de un flanco de reloj.

Cada bloque RAM puede unirse en cascada para crear memorias más grandes. En la figura 54 se visualiza un ejemplo de bloque RAM de 36Kbit.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 72 de 120

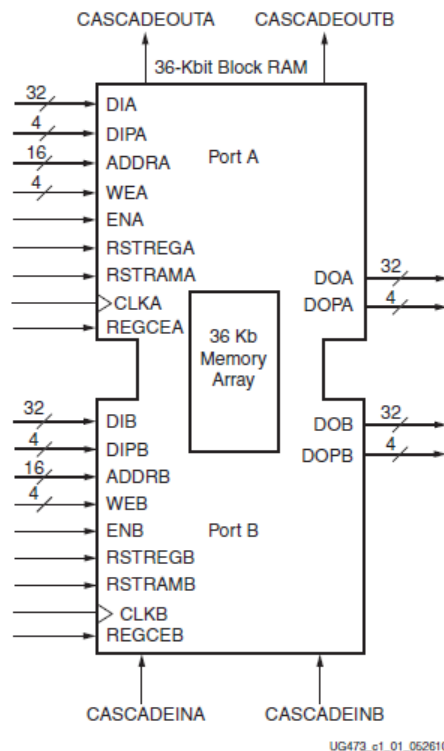


Figura 54: Bloque de RAM de 36Kb (RAMB36)



5.5.3. Métodos para incorporar módulos de memoria al diseño

En ISE existen tres métodos diferentes para incorporar módulos de memoria en un diseño: mediante instanciación HDL, utilizando un *IP Core Generator* o mediante inferencia utilizando *templates* HDL.

5.5.3.1. Mediante instanciación HDL

Este método consiste en instanciar un componente de tipo memoria ya creado por Xilinx. El *template* para cada tipo de memoria, se puede ver en la pestaña *Edit / Languages templates* (Figura 55). Ahí se elige el tipo lenguaje, la placa familia de la placa y el tipo de memoria.

En este caso se pueden elegir las siguientes memorias:

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 73 de 120

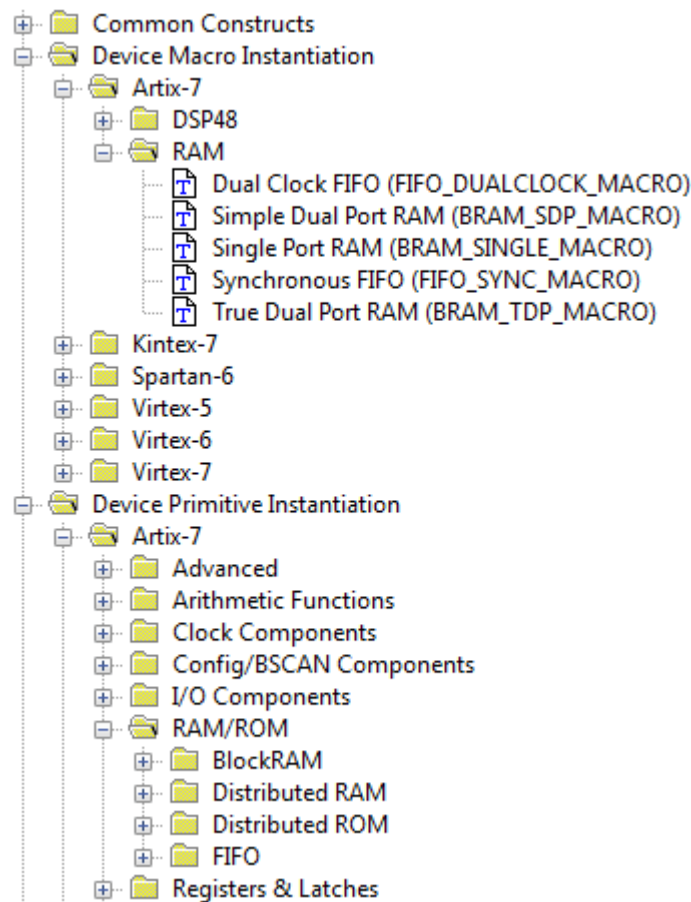




Figura 55: Templates para Memorias RAM

La ventaja de este método es que la instanciación define el comportamiento y las conexiones de la RAM escogida, teniendo únicamente el diseñador que realizar asignaciones a cada entrada y salida del componente instanciado. En cambio, la instanciación es específica para cada dispositivo, lo cual puede crear incompatibilidades si cambiamos de FPGA.

5.5.3.2. Mediante un IP Core Generator

Al igual que se utiliza el *IP Core Generator* para crear un generador de frecuencias, se puede usar para crear los tipos de memorias vistas anteriormente.

Para crear una RAM tanto de tipo distribuida, como de bloques, mediante un *IP Core Generator*, primero se añade dentro del módulo deseado uno nuevo, y se selecciona el tipo IP (*Core Generator & Architecture Wizard*). Luego en la ruta *Memories & Storages Elements / RAM & ROMs* se escoge el IP (Figura 56).

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 74 de 120

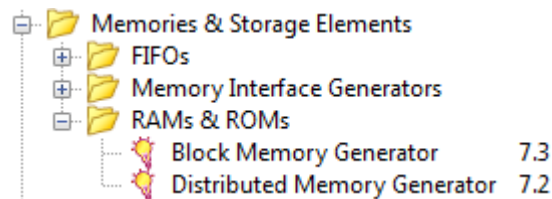


Figura 56: Directorio Block and Distributes Memory Generator

En función del tipo memoria se elige un generador u otro. Para la memoria RAM de tipo distribuida le corresponde el IP *Distributed Memory Generator* (Figura 57).

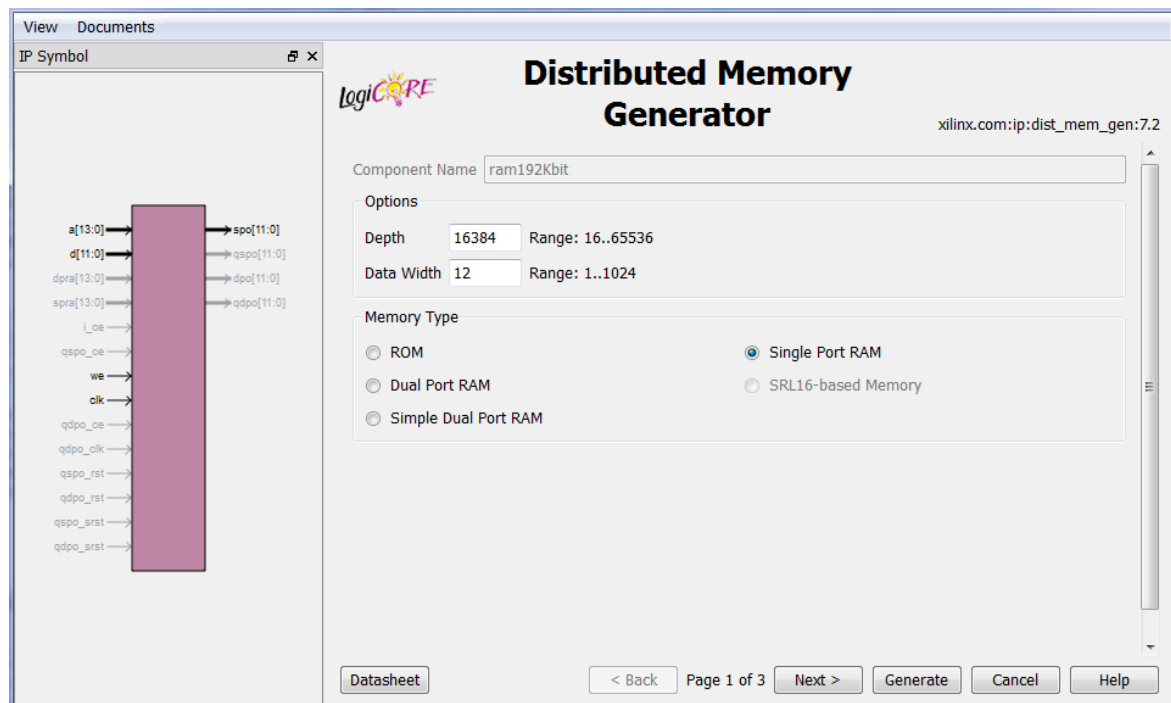




Figura 57: Distributed Memory Generator

En el apartado opciones, se puede modificar la profundidad de memoria (*Depth*) y el ancho del dato que se quiere almacenar (*Data Width*).

En este sistema, las muestras que queremos almacenar son de 12bits de datos (*Data Width*), y como máximo se va a almacenar 8820 muestras (teniendo en cuenta la $f_m=44.1\text{kHz}$ y el máximo tiempo de retardo de 200ms). Sabiendo el número máximo de muestras para almacenar, se calcula la profundidad de la memoria o el número máximo de direccionamiento en memoria:

$$2^x \geq 8820$$

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 75 de 120

$$x \geq \frac{\log_{10} 8820}{\log_{10} 2} = 13.1066$$

Como solamente se pueden direccionar a números enteros se selecciona un índice de 14, de esta forma se tiene una dirección de memoria de 14 bits que puede direccionar hasta 2^{14} posiciones en memoria, es decir, la memoria tendrá un ancho de 16384.

En la pestaña *IP Symbol*, se observan como en función de tamaño del dato y del ancho de la memoria, la entrada a de dirección y la entrada d de datos modifica su número de bits.

Posteriormente se selecciona el tipo de memoria para el diseño. Para este sistema se puede elegir la de un solo puerto o la de doble puerto. Como se comenta anteriormente, la memoria de un solo puerto escribe y lee en memoria, pero no simultáneamente. En cambio, la de doble puerto puede escribir y leer en memoria simultáneamente.

Una memoria distribuida RAM de uno y dos puertos tiene las siguientes señales de entrada y salida (Figura 58).

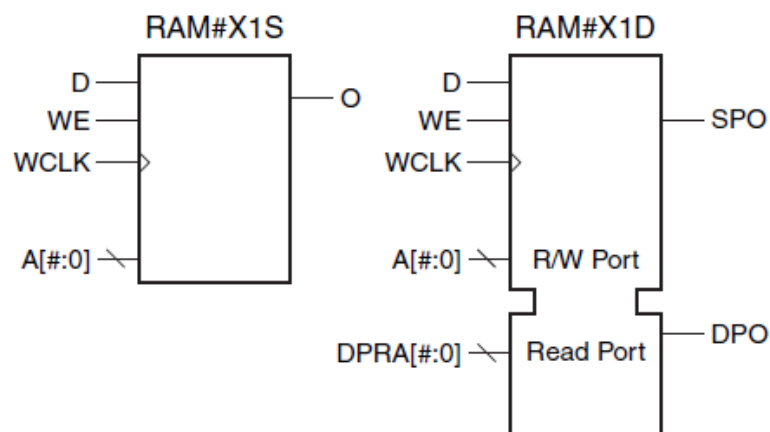


Figura 58: Single-Port, Dual-Port Distributed RAM Primitives

- Puertos de entrada
 - Address – A[#:0], DPRA[#:0]: entrada de dirección A[#:0] (para memoria de uno y dos puertos) y DPRA[#:0] (para memoria de dos puertos) selecciona la celda de memoria para escribir o leer un dato. El ancho del puerto determina el número de direcciones de la memoria. En puerto uno, se puede escribir o

leer en memoria, en el segundo puerto solamente se puede leer en memoria.

- Data In – D: es el dato de entrada que se quiere escribir en memoria.
- Enable - WE (write enable): cuando está a uno indica que se escribe el dato d en la dirección de memoria a. Cuando está a cero indica que no puede haber escritura en memoria.
- Clock - WCLK: el reloj se utiliza para la operación de escritura, que es síncrona. La señal puede estar activa por flanco de bajada o de subida sin requerimiento de otros dispositivos lógicos.
- Puertos de salida
 - Data Out – O, SPO, DPO: la salida de datos O (de un puerto o SPO), DPO (doble puerto) refleja el contenido de la celda de memoria referenciada por la entrada de dirección. Cuando $we=1$, SPO es igual al dato de entrada D y cuando $we=0$, SPO es igual al dato almacenado en la dirección de memoria A. La salida DPO siempre corresponde al dato almacenado en la celda de memoria con dirección DPRA. Los puertos de salida de datos, son asíncronos, no dependen del flanco de reloj.

En la figura 59 se ven las características temporales de los puertos de entrada y salida en la operación de escritura y lectura.

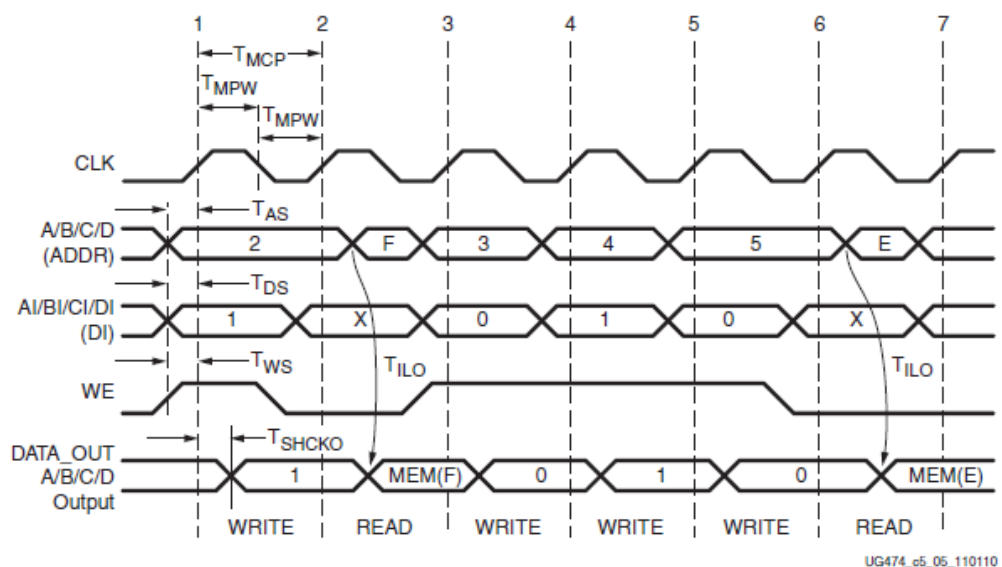




Figura 59: Características temporales memoria RAM Distribuida

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 77 de 120

Con este diagrama se ve la operación de escritura y lectura en memoria.

- Operación de escritura:
 - En el tiempo T_{WS} , antes del primer flanco de reloj, se activa la habilitación de escritura WE, habilitando la operación de lectura en memoria.
 - En el tiempo T_{AS} , se asigna la dirección 2 de memoria.
 - En el tiempo T_{DS} , el dato de entrada es el valor 1 y apunta a la dirección de memoria 2 y en el tiempo T_{SCHKO} se escribe el dato en la dirección de memoria.
- Operación de lectura:
 - La operación de lectura es asíncrona en la memoria RAM distribuida. Siempre que la habilitación de escritura WE este desactivada, se podrá acceder a la dirección de memoria para leer el dato (excepto en la memoria de doble puerto, que siempre se podrá acceder en el segundo puerto, ya que solo es de lectura). El contenido de la memoria de la dirección especificada se muestra en la salida después de un tiempo de retraso T_{ILO} . En el diagrama se observa cómo se lee en memoria el dato almacenado en la dirección F después del tiempo de retraso.

En el caso de implementar un bloque de RAM se usa el IP *Core Block Memory Generator* (Figura 60).

Al igual que en la memoria distribuida, en la configuración permite elegir varios tipos de bloque de memoria RAM, entre ellas las de un puerto y de dos puertos. También permite elegir el tamaño de cada puerto y la profundidad de memoria. Además puedes elegir el modo de operación de la memoria (*Write First, Read First, No Change*), en el caso de las memorias de doble puerto solamente está disponible el modo de operación por defecto (*Write First*).

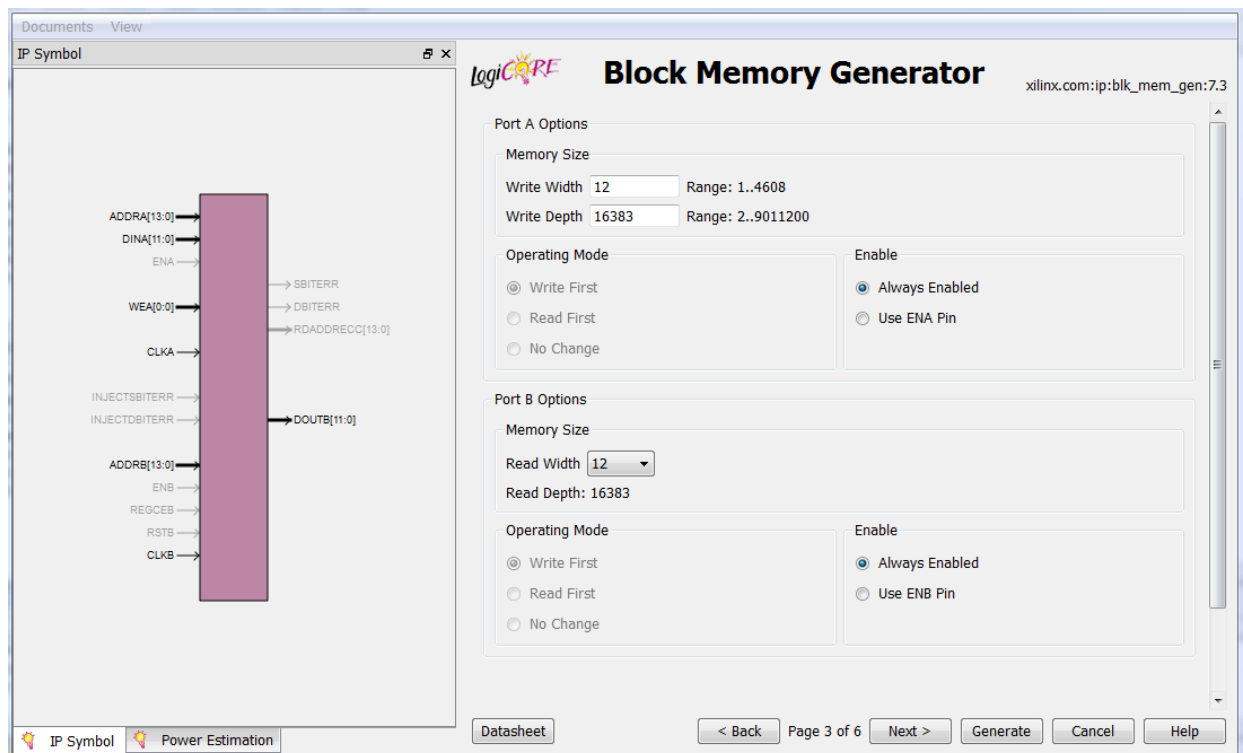


Figura 60: Block Memory Generator

Los bloques de memoria RAM de uno o dos puertos tienen las siguientes señales de entrada y salida.

- Puertos de entrada
 - Address – ADDRA[#:0], ADDR#B[#:0]: es la dirección de memoria del puerto A y del puerto B.
 - Data In – DINA: es el dato de entrada que se quiere escribir en memoria.
 - Enable - WE (*write enable*): cuando está a uno indica que se escribe el dato d en la dirección de memoria específica. Cuando está a cero indica que no puede haber escritura en memoria.
 - Clock - CLKA: entrada de reloj del bloque de memoria.

La operación de lectura en la memoria BRAM es síncrona. El dato almacenado en la dirección de memoria leída es cargado a la salida de los latches después del tiempo de acceso a memoria.

La operación de escritura también es síncrona. La dirección de escritura en memoria es registrada en el puerto de escritura y el dato de entrada es almacenado en esa dirección de memoria.

Existen tres modos de escritura determinados por el comportamiento del dato disponible después de un flanco de subida de reloj de escritura: WRITE_FIRST, READ_FIRST, y NO_CHANGE. El modo por defecto es WRITE_FIRST. En el primer modo, los datos recién escritos van a la salida del bus. En el modo READ_FIRST muestra los datos almacenados previamente mientras se escriben nuevos datos. En el modo NO_CHANGE se mantiene la salida previamente generada por una operación de lectura.

En las figuras 61,62 y 63 se pueden ver el comportamiento temporal de cada modo de escritura.

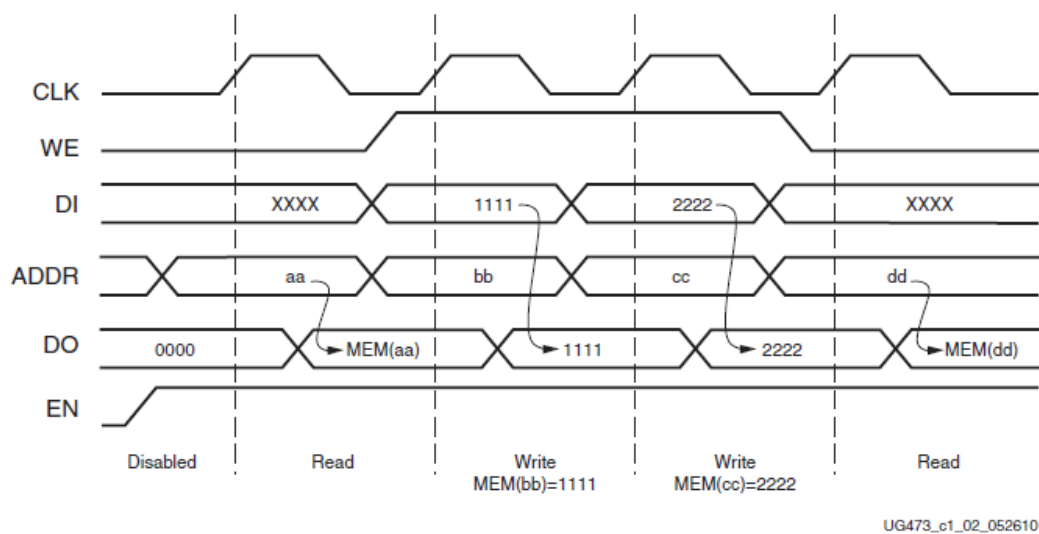


Figura 61: Modo WRITE_FIRST

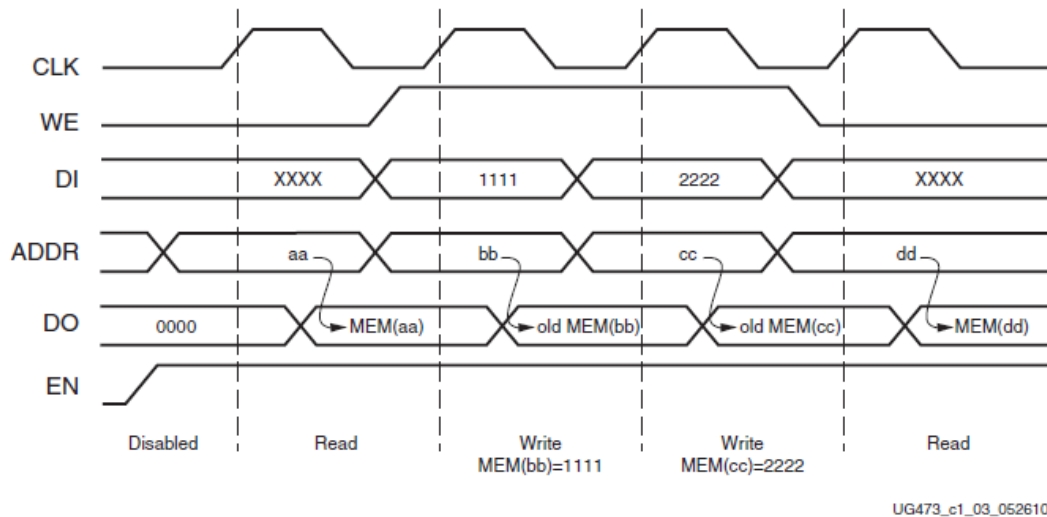


Figura 62: Modo READ_FIRST

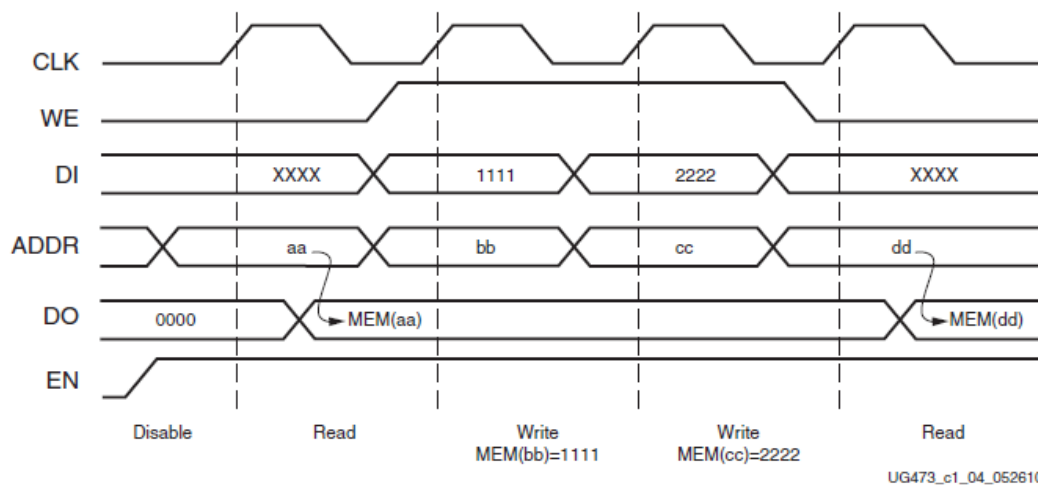


Figura 63: Modo NO_CHANGE

5.5.4. Proceso de control de lectura y escritura en memoria

Para el control de lectura y escritura en memoria se usa un proceso síncrono dentro del módulo **sistema_dsp**. El diseño de este proceso dependerá del tipo de memoria escogido, ya que el funcionamiento y las entradas y salidas pueden variar. Pero como regla general, será un proceso que primero estará inicializado en modo escritura y apuntando a la dirección más baja de memoria. El proceso síncrono estará controlado por la frecuencia de muestreo del sistema y dependerá de si la memoria es de un puerto o de doble puerto.

Para las memorias de un solo puerto el código es más complejo, ya que no se pueden realizar escrituras y lecturas simultáneamente. Para este caso, cada vez que llega una muestra (`sample_done_flag` activo) se incrementa la dirección de memoria y se escribe el dato en esa dirección. Permanece en modo escritura hasta que se haya escrito en memoria las muestras deseadas para obtener el retraso específico. Posteriormente se reinicia la dirección de memoria al inicio, se lee el dato y luego se sobrescribe el dato en esa dirección cada vez que llegue una muestra. De esta forma se consigue un retraso de la señal igual al número de muestras almacenadas por la frecuencia de llegada de muestras (frecuencia de muestreo).

Para el caso de las memorias de doble puerto el código es más sencillo, ya que se escribe y lee simultáneamente en memoria. En este caso, cada vez que llegue una muestra, se escribe en la dirección de memoria equivalente a la suma de la dirección de lectura actual y el número de muestras que se quiera almacenar para conseguir el retraso específico. Al mismo tiempo se estarán incrementando la dirección de lectura cada vez que nos llegue una muestra. El dato de salida será al correspondiente que se sitúa en la dirección de lectura de memoria.

Por ejemplo, en la simulación de la figura de la figura 64, se observa como la dirección de escritura `addra`, corresponde con el valor del retraso (`delay`) incrementado en uno, por ser la primera dirección de escritura, el cual se incrementa cada vez que llegue una muestra. La dirección de lectura `addrb` empieza en el inicio de memoria y se va incrementando igualmente. De esta forma cuando `addrb` sea igual a `delay`, se leerá el dato que se escribió hace `delays` periodos de muestreo, es decir, se obtendrá un retardo equivalente al valor del `delay` por el periodo de muestreo.

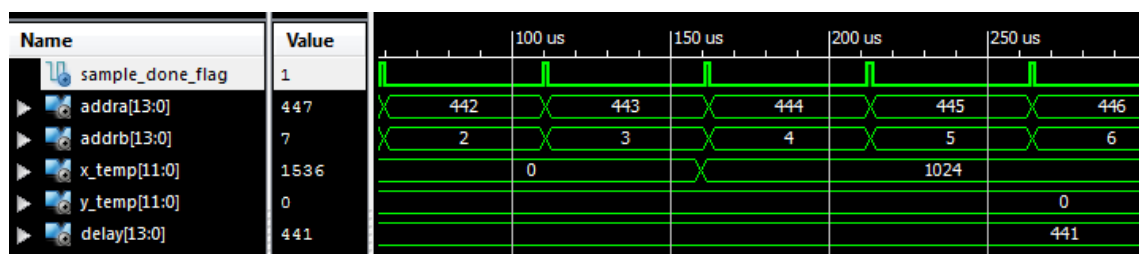


Figura 64: Ejemplo de escritura y lectura en memoria

5.5.5. Consumo de recursos en función del tipo de RAM escogida

Se han comparado el consumo de recursos utilizando una memoria RAM distribuida y una memoria RAM de bloques. En las figuras 65 y 66 podemos ver el resultado para cada tipo de memoria.

Device Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	157	126,800	1%
Number used as Flip Flops	153		
Number used as Latches	3		
Number used as Latch-thrus	0		
Number used as AND/OR logics	1		
Number of Slice LUTs	3,566	63,400	5%
Number used as logic	491	63,400	1%
Number using O6 output only	379		
Number using O5 output only	24		
Number using O5 and O6	88		
Number used as ROM	0		
Number used as Memory	3,072	19,000	16%
Number used as Dual Port RAM	0		
Number used as Single Port RAM	3,072		
Number using O6 output only	3,072		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used as Shift Register	0		



Figura 65: Consumo de recursos de una RAM distribuida de 192Kb

Device Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	115	126,800	1%
Number used as Flip Flops	111		
Number used as Latches	3		
Number used as Latch-thrus	0		
Number used as AND/OR logics	1		
Number of Slice LUTs	108	63,400	1%
Number used as logic	102	63,400	1%
Number using O6 output only	62		
Number using O5 output only	18		
Number using O5 and O6	22		
Number used as ROM	0		
Number used as Memory	0	19,000	0%
Number used exclusively as route-thrus	6		
Number with same-slice register load	4		
Number with same-slice carry load	2		
Number with other load	0		
Number of occupied Slices	48	15,850	1%
Number of LUT Flip Flop pairs used	142		

Figura 66: Consumo de recursos de una memoria BRAM de 192Kb

Se comprueba que al utilizar la memoria distribuida se amplía el uso de Slice LUTs hasta un 5% ya que los está utilizando como memoria. Sin embargo, si se utiliza el bloque de memoria, solamente se utiliza un 1% de Slice del LUTs ya que solamente los utiliza para lógica y no para memoria.

Como conclusión, se puede decir que para este sistema nos es indiferente utilizar un tipo u otro de memoria ya que la utilización de los recursos de la FPGA es mínima, sin embargo, es recomendable utilizar un bloque de memoria

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 83 de 120

ya que así no se utiliza Slices LUTs para almacenamiento en memoria y lo se reservan para operaciones lógicas.

5.6. Módulos de interacción con la placa de desarrollo

La placa de desarrollo Nexys4 DDR presenta una serie de periféricos que se pueden utilizar para una mejora en la interacción del usuario con el diseño.

En el diseño se utiliza los periféricos del display, los switches y los pulsadores para permitir al usuario realizar modificaciones en los parámetros de control del filtro y además visualizar en el display que tipo del filtro y efecto se está utilizando.

La interfaz con el display se realiza con el módulo **visu7seg**, el cual tiene la siguiente entidad:

```
entity visu7seg is
    Port ( clk      : in  STD_LOGIC;
          reset    : in  STD_LOGIC;
          display_0 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_1 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_2 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_3 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_4 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_5 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_6 : in  STD_LOGIC_VECTOR (4 downto 0);
          display_7 : in  STD_LOGIC_VECTOR (4 downto 0);
          an       : out STD_LOGIC_VECTOR (7 downto 0);
          a        : out STD_LOGIC;
          b        : out STD_LOGIC;
          c        : out STD_LOGIC;
          d        : out STD_LOGIC;
          e        : out STD_LOGIC;
          f        : out STD_LOGIC;
          g        : out STD_LOGIC);
end visu7seg;
```

Las entradas son el reloj del módulo, el reset, y lo que se quiere pintar en cada display de la placa, codificado en un vector de 5 bits. Cada valor del display se decodifica y corresponde a un valor concreto en formato 7 segmentos.

Las salidas son los diferentes segmentos del display (excluyendo el punto) y un vector de 8bits, en el que el bit igual a 0 indica el display activo y se va desplazando una frecuencia de reloj específica (Figura 67).

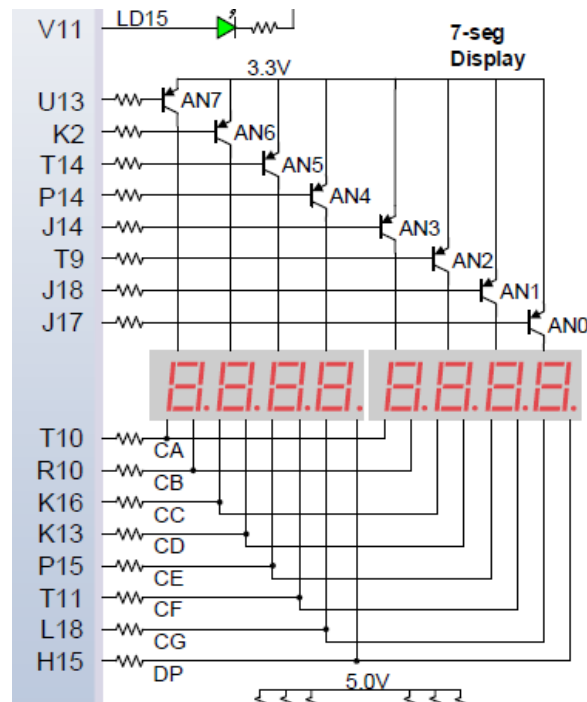


Figura 67: Esquema de pines del display

Según el datasheet de la placa de desarrollo, el periodo de refresco para pintar en 4 displays es de 1ms a 16ms (Figura 68). Por lo tanto, en función del reloj de entrada, se crea un reloj para el periodo de refresco del display.

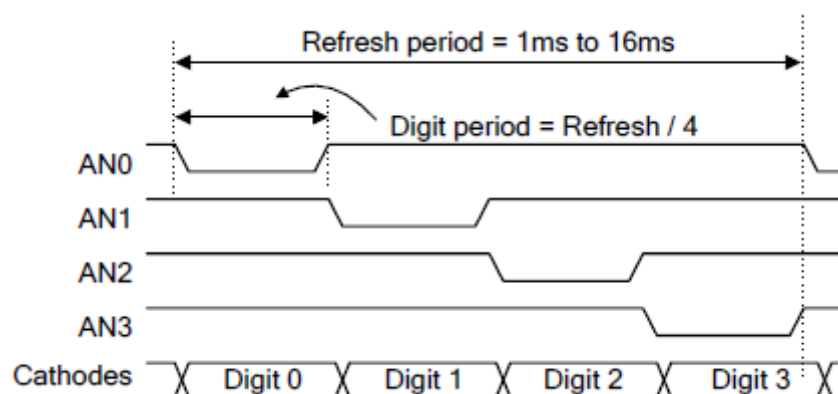




Figura 68: Diagrama temporal del periodo de refresco del display

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 85 de 120

Para obtener la frecuencia del display, en el proceso se utiliza un contador de 19 bits, que incrementa uno cada flanco de subido del reloj de entrada. Los bits 17 a 19 se utilizan para seleccionar el display en concreto, por lo tanto, la frecuencia del display será igual a:

$$f_{display} = \frac{clk}{(2^{17})}$$

Si se utiliza una frecuencia de reloj de 100MHz o de 50MHz, la frecuencia del display será igual a:

$$f_{display} = \frac{100000000Hz}{(2^{17})} = 762.94Hz \rightarrow T_{display} = 1.31ms$$

$$f_{display} = \frac{50000000Hz}{(2^{17})} = 381.47Hz \rightarrow T_{display} = 2.62ms$$



Para 4 displays se está dentro del rango requerido por el datasheet, por lo tanto, se pueden utilizar las dos frecuencias de reloj.

La interacción con los interruptores y pulsadores de la placa de desarrollo se realizan mediante asignaciones directas dentro del módulo **sistema_dsp**, el cual tendrá como entrada estos actuadores.

5.7. Construcción de filtros

El diseño del sistema se basa en un sistema genérico, el cual con ciertas modificaciones, activadas por actuadores, cambia la funcionalidad del sistema, modificando el flujo de datos y los tiempos de retardo.

El sistema genérico se compone por un módulo central, llamado **sistema_dsp** que controla todos los módulos del sistema. Primero controla la interfaz del PmodAD1, mediante el módulo **pmodAD1**, de aquí se obtiene una muestra, la cual es procesada de diferente forma, en función del filtro utilizado. Luego se tiene el módulo **ram192Kbit** que implementa una memoria BRAM de 192Kb, en el cual las operaciones de lectura y escritura están controladas mediante un proceso dentro del módulo **sistema_dsp**. También se tiene el módulo **gen_reloj** que genera los relojes máster de todo el sistema. Por último,



			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 86 de 120

se tiene el módulo **visu7seg** el cual controla los displays de la placa de desarrollo y el módulo **pmodl2S** que controla la interfaz de salida con el Pmodl²S.

Al sistema genérico se le añade el módulo **select_filter**, el cuál selecciona un tipo de filtro u otro, dependiendo del interruptor activado. En función del filtro seleccionado, el tiempo de retardo de las muestras y el flujo de datos dentro del módulo **sistema_dsp** cambiará. Además, se podrá modificar el parámetro de control del filtro mediante un pulsador, cambiando así el efecto cada vez que se active.

A la entidad **sistema_dsp**, definida anteriormente, se le añaden los puertos correspondientes con los módulos que hacen de interfaz con la placa de desarrollo, es decir, los puertos de entrada referidos a los actuadores y los puertos de salida para los displays. La definición de la entidad **sistema_dsp** es la siguiente:

```
entity sistema_dsp is
  Port ( clk : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        miso : in  STD_LOGIC;
        echo : in  STD_LOGIC;
        vibrato: in  STD_LOGIC;
        flanger: in  STD_LOGIC;
        chorus: in  STD_LOGIC;
        pulsador : in  STD_LOGIC;
        adl_clk : out  STD_LOGIC;
        cs : out  STD_LOGIC;
        mclk : out  STD_LOGIC;
        sclk : out  STD_LOGIC;
        lrck : out  STD_LOGIC;
        sdin : out  STD_LOGIC;
        an : out  STD_LOGIC_VECTOR (7 downto 0);
        a : out  STD_LOGIC;
        b : out  STD_LOGIC;
        c : out  STD_LOGIC;
        d : out  STD_LOGIC;
        e : out  STD_LOGIC;
        f : out  STD_LOGIC;
        g : out  STD_LOGIC);
end sistema_dsp;
```

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 87 de 120

Los tiempos de retardo y el flujo de datos seleccionados de cada filtro, se han elegido mediante las simulaciones de los modelos en Simulink.

Además se ha tenido que crear un módulo adicional, para los casos de los filtros que tienen un retraso modulado. Para ello se ha creado el módulo **sine_wave**, el cual genera una onda senoidal discreta con frecuencias y amplitudes variables, en función del modo de efecto, lo cual permite modular el retraso de la señal.

5.7.1. Selección de filtro mediante el módulo **select_filter**

El módulo **select_filter** selecciona, en función de sus entradas, el tipo de filtro y el efecto. La entidad de este módulo es la siguiente:

```
entity select_filter is
  Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
         echo : in  STD_LOGIC;
         vibrato: in  STD_LOGIC;
         flanger: in  STD_LOGIC;
         chorus: in  STD_LOGIC;
         pulsador : in  STD_LOGIC;
         echo_mode :out STD_LOGIC_VECTOR(3 downto 0);
         vibrato_mode :out STD_LOGIC_VECTOR(5downto 0);
         flanger_mode :out STD_LOGIC_VECTOR(5downto 0);
         chorus_mode:out STD_LOGIC_VECTOR(5downto 0));
end select_filter;
```

Las entradas echo, vibrato, flanger y chorus se corresponden con cada switch de la placa de desarrollo que activa cada filtro. Cada filtro tiene varios modos diferentes, correspondientes a cada efecto, que se pueden seleccionar a través del pulsador. No podrán ser seleccionados varios filtros a la vez.

Cuando se selecciona un filtro, este tendrá asignado un modo por defecto, que posteriormente se podrá cambiar mediante el pulsador.

Las salidas echo_mode, vibrato_mode,y flanger_mode se envían al módulo **sistema_dsp**. En este módulo se asigna un delay y un pintado de display mediante un proceso, en función del filtro y del modo seleccionado.

5.7.2. Tiempos de retardo y flujo de datos del filtro eco

Para el filtro eco, se utilizan 4 modos diferentes. El modo 1 realiza un retraso de 10ms, para realizar el efecto doubling, el modo 2 realiza un retraso de 50ms, para realizar el efecto Slapback, y el modo 3 y 4 realiza un retraso de 100ms y 200ms, para realizar dos ecos diferentes. Las asignaciones al delay, teniendo en cuenta el retraso de cada modo y una frecuencia de muestreo de 44.1kHz, son los detallados en la tabla 2:

Tabla 2: Delays en función del efecto del filtro eco

Efecto	Echo_mode	Tiempo de retardo	Delay (samples)
Doubling	1	10ms	441
Slapback	2	50ms	2205
Eco	3	100ms	4410
Eco	4	200ms	8820

Para el eco, el modo por defecto es de 100ms. El pintado en el display de cada modo se muestra en la tabla 3:



Tabla 3: Pintado en función del efecto del filtro eco

Efecto	Tiempo de retardo	Display
Doubling	10ms	"echo 10"
Slapback	50ms	"echo 50"
Eco	100ms	"echo 100"
Eco	200ms	"echo 200"

El flujo para este filtro es la suma de la señal de entrada con la señal retrasada delays muestras.

5.7.3. Tiempos de retardo y flujo de datos del filtro vibrato

Como se ha definido anteriormente, para el efecto vibrato el tiempo de retraso es modulado por una señal senoidal de 5Hz a 14Hz. En las pruebas realizadas mediante Simulink, los vibratos de mejores características se consiguieron con un retaso de 1ms, modulados por una onda senoidal de frecuencias de entre 1Hz hasta 24Hz. Para implementarlo en la placa se ha escogido el mismo criterio.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 89 de 120

Para crear la onda senoidal con diferentes frecuencias, se ha creado el componente **sine_wave** dentro del **sistema_dsp**. La entidad de este módulo, tiene como entradas, el reloj, el reset, el modo del vibrato y como salidas el delay.

```
entity sine_wave is
    Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          vibrato_mode : in STD_LOGIC_VECTOR(5 downto 0);
          delay: out  unsigned (13 downto 0));
end sine_wave;
```

En la arquitectura de este módulo, se almacena en memoria ROM (*Read Only Memory*) los valores de una señal senoidal discreta. Posteriormente, se leen los valores a una frecuencia de muestreo determinada. Dependiendo de la frecuencia de muestreo elegida cambiará la frecuencia de la señal senoidal discreta.

Para ello, primero se obtiene el número de valores necesarios para representar la señal senoidal, para ello se aplica el teorema de Nyquist.

La frecuencia máxima para el vibrato es de 24Hz, por lo tanto teóricamente, con una frecuencia de muestreo mayor del doble de la máxima se puede representar la onda. Por la relación, con otros relojes del sistema, elegimos 50 muestras para representar la onda. De esta forma, si muestreamos a 1.2kHz (50 veces f_{\max}) obtenemos la señal senoidal discreta de 24Hz.

Para crear los valores de la señal senoidal discreta, se ha utilizado el siguiente código en Matlab:

```
>> np=50;
>> A=20;
>> t=linspace (0,1-1/np, np);
>> sin_table= round(sin(2*pi*t)*A);
>> sin_positive=20+sin_table;
>> seno=int32(sin_positive);
```

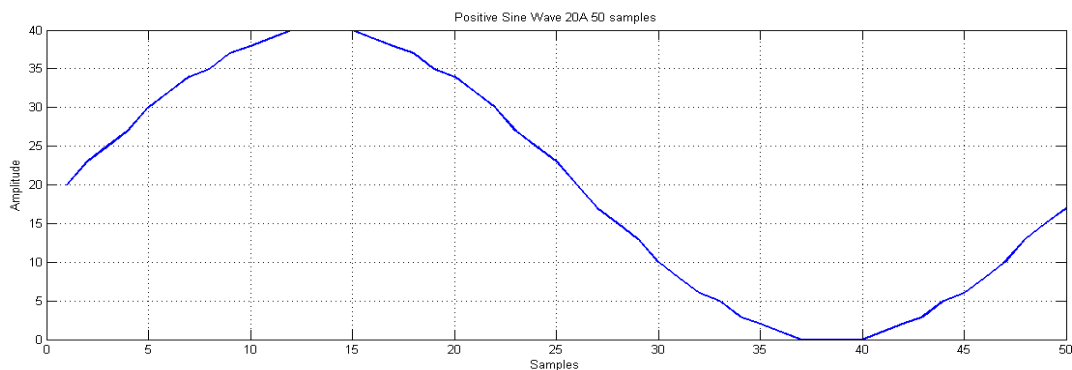



Figura 69: Onda senoidal positiva de amplitud 20 de 50 muestras.

Se le ha asignado una amplitud de 20, para llegar a un máximo de 40, es decir, con esta señal el delay tendrá valores de 0 a 40 muestras, lo que se traduce a un retraso máximo de aproximadamente 1ms (Figura 69).

Para conseguir la frecuencia de entre 1Hz a 24Hz, se crea un proceso de contador, que crea las diferentes frecuencias de muestreo, a las que se accederá a la ROM que contiene los valores de la onda senoidal

El contador es un vector de 13 bits, que se incrementa con la frecuencia de reloj de bit del sistema (ad1_clk o sclk) igual a 1.25MHz. Los bits 9, 10, 11 y 12, se asignan a la señal de muestreo de la onda senoidal mediante un select controlado por el modo del vibrato, de esta forma obtenemos las frecuencias a las que se va a muestrear los valores de la senoide, obteniendo así senoides de diferentes frecuencias. En tabla 4, se observa la relación:

Tabla 4: Delays en función del tipo de efecto vibrato.

Efecto	Vibrato_mode	Tiempo de retardo	Muestreo	Frecuencia Onda senoidal	Display
Vibrato	1	1ms	sclk/1024=1.22kHz	24.4Hz	"24 Hz"
Vibrato	2	1ms	sclk/2048=610.35Hz	12.2Hz	"12 Hz"
Vibrato	3	1ms	sclk/4096=305.18Hz	6.1Hz	"6 Hz"
Vibrato	4	1ms	sclk/8192=152.59Hz	3.05Hz	"3 Hz"
Vibrato	5	1ms	sclk/16384=76.29Hz	1.525Hz	"1 Hz"
Vibrato	6	1ms	sclk/32768=38.15Hz	0.762Hz	"05 Hz"

El flujo de este filtro no tiene el lazo de realimentación positiva, por lo tanto, solamente se asigna a la salida del sistema la señal retrasada.

5.7.4. Tiempos de retardo y flujo de datos del filtro flanger

En el filtro flanger también se utiliza una onda senoidal para modular el retraso. Según las simulaciones en Simulink, los efectos con mejores características tienen 1Hz de frecuencia en la señal moduladora y un retardo de 1ms a 32ms.

Para ello, se utiliza la onda senoidal del modo 5 del filtro vibrato, que tiene 1ms de retardo y 1Hz aproximadamente de frecuencia. Para aumentar el retraso, solamente se tiene que aumentar la amplitud de esta frecuencia. En la tabla 5 se muestra los diferentes modos del filtro flanger en función del tiempo de retardo.

Tabla 5: Delays en función del tipo de efecto flanger.

Efecto	Vibrato_mode	Frecuencia Onda senoidal	Tiempo de retardo	Muestreo	Display
Flanger	1	1.525Hz	1ms	sclk/16384=76.29Hz	"flange 1"
Flanger	2	1.525Hz	2ms	sclk/16384=76.29Hz	"flange 2"
Flanger	3	1.525Hz	4ms	sclk/16384=76.29Hz	"flange 3"
Flanger	4	1.525Hz	8ms	sclk/16384=76.29Hz	"flange 4"



Para este filtro, se añade un filtro de paso bajo de primer orden, en el lazo de realimentación, para ello se crea el módulo LPF. La entidad de este módulo es la siguiente:



```

entity LPF is
  Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
         datain_LPF : in  STD_LOGIC_VECTOR(11 downto 0);
         dataout_LPF : out STD_LOGIC_VECTOR(11 downto 0));
end LPF;

```

El flujo del filtro flanger está formado por la suma de la señal retrasada, con la suma, de la señal de entrada con la señal realimentada del LPF. Para ello se crea la variable `x_sum` que se encarga de sumar la entrada con la retroalimentada para posteriormente sumársela a la señal retrasada, logrando así la estructura del filtro paso todo.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 92 de 120

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 93 de 120

6.SISTEMA FÍSICO

6.1. Conversión A/D

Tanto para la conversión A/D como para la D/A se utilizan unos módulos periféricos llamados Pmod (*Peripheral Modules*) del fabricante Digilent. Estos dispositivos, permiten ampliar las capacidades de la FPGA y la placa de desarrollo. Además, estos periféricos pueden colocarse donde el comportamiento del circuito sea más efectivo, por ejemplo, en circuitos de acondicionamiento de señal y circuitos de control de alta potencia se podrá colocar cerca de los sensores y de los actuadores.

Para la conversión A/D se ha utilizado el PmodAD1. Este Pmod integra dos convertidores analógicos digital AD7476A, del fabricante Analog Devices, los cuales proporcionan dos canales de datos de 12 bit, con una velocidad de hasta 1 millón de muestras por segundo. El periférico también integra dos filtros Sallen-Key anti-aliasing y tiene 6 pines de conexión con interfaz GPIO (General Purpose Input/Output). En la imagen de la figura 70 se puede visualizar el PmodAD1.

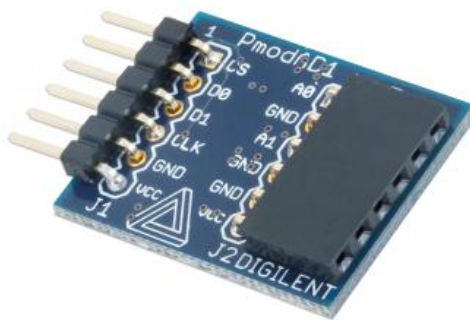


Figura 70: PmodAD1

6.2. Conversión D/A

En la conversión digital analógica, se ha utilizado el PmodI²S del fabricante Digilent. Este periférico utiliza el convertidor digital analógico CS4344 del fabricante Cirrus Logic, el cual convierte señales digitales de audio estéreo de hasta 24 bits. El Pmod tiene 6 pines de conexión con interfaz GPIO y para las señales de salida de audio un conector Jack estándar hembra estéreo de 3,5mm. En la figura 71 y 72 se puede visualizar el PmodI²S y el DAC CS4344.



			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		
			Página 94 de 120



Figura 71: DAC CS4344

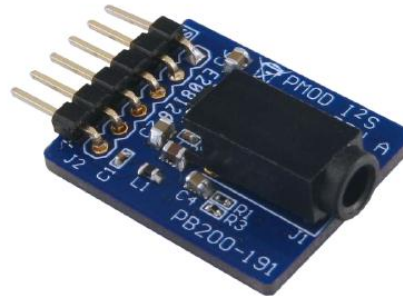


Figura 72: PmodI2S.

6.3. Construcción del sistema

El sistema está formado por la placa Nexys4 DDR a la que se conecta el PmodAD1 en el puerto Pmod JB, y por el PmodI²S el cual se conecta en el puerto JA. Adicionalmente se utilizan los 5 primeros switch (J15, L16, M13, R15, R17), el segundo pulsador (M17) y el display.

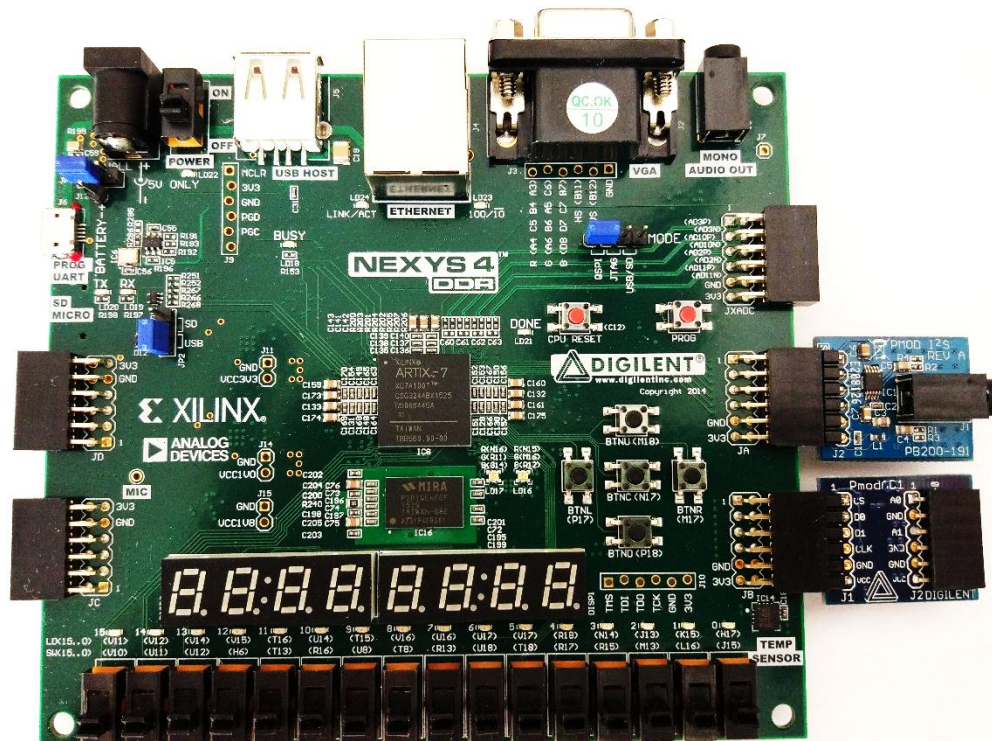


Figura 73: Construcción del sistema

A esta configuración, se le añade un circuito de acondicionamiento de señal en la entrada del PmodAD1, el cual se define en el siguiente punto.

6.4. Acondicionamiento de la señal de entrada

Las pruebas que se han llevado a cabo en la placa de desarrollo se han realizado con una salida de audio de PC. Estas salidas son señales que se mueven dentro de un rango de $\pm 500\text{mV}$, dependiendo de la amplificación o volumen de la señal.

Según el datasheet del convertidor analógico digital AD7476A, la entrada analógica del convertidor está comprendida entre 0 y la tensión de alimentación, en este caso de 3.3V, como se muestra en la tabla 4.

Tabla 6: Rango de entrada analógica AD7476A.

ANALOG INPUT				
Input Voltage Range	0 to V_{DD}	0 to V_{DD}	0 to V_{DD}	V
DC Leakage Current	± 0.5	± 0.5	± 0.5	$\mu\text{A max}$
Input Capacitance	20	20	20	pF typ

Esto quiere decir, que si se quiere utilizar la señal de audio del PC, se tiene que acondicionar para la entrada de este convertidor, para que esté dentro del rango especificado en el datasheet.

Para acondicionar la señal se ha diseñado un circuito de acondicionamiento de señal, basado en un sumador de tensión, que suma la mitad de la tensión de referencia a la señal de entrada, para conseguir una señal dentro de los rangos especificados. La figura 74 representa el esquemático:

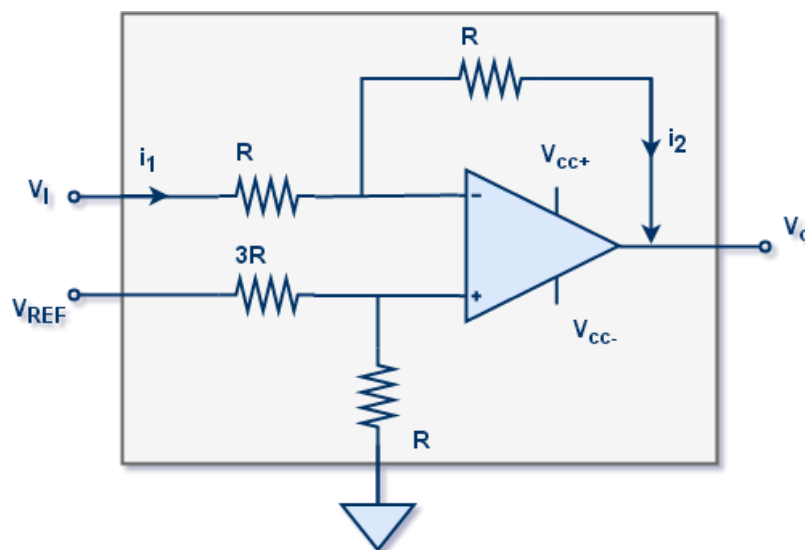


Figura 74: Esquemático del circuito de acondicionamiento de señal.

Considerando el amplificador operacional ideal, es decir, ganancia en tensión en lazo abierto infinita y corriente de los terminales de entrada nula, obtenemos la resolución del circuito:

$$V_+ = \frac{R}{4R} V_{REF}$$

$$i_1 = i_2$$

$$\frac{V_I - V_-}{R} = \frac{V_- - V_o}{R}$$

$$V_o = \frac{1}{2} V_{REF} - V_I$$

Por lo tanto, se amplifica la señal de entrada un medio la tensión de referencia. En la figura 75 se muestra la respuesta temporal para una entrada senoidal de 0.5 de amplitud.

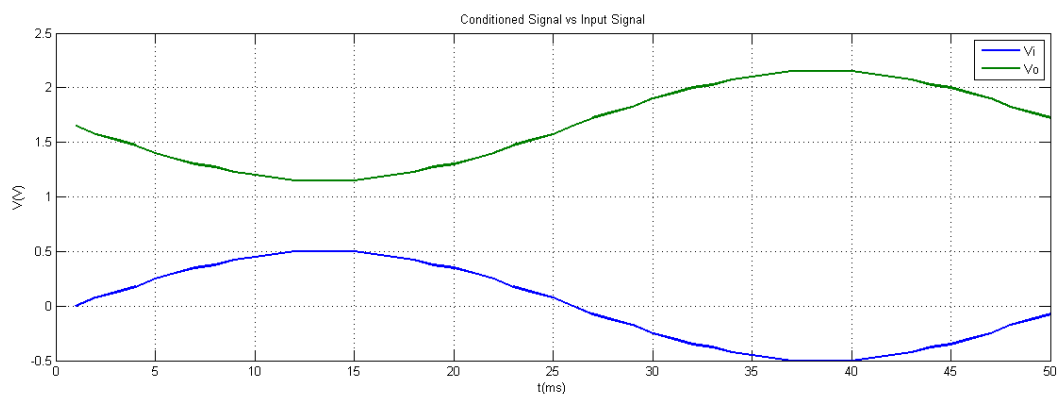


Figura 75: Respuesta del circuito de acondicionamiento de señal.

Se puede observar como la señal de entrada ya está acondicionada para la entrada del convertidor.

Para realizar este circuito, se ha utilizado el operacional TL072CP del fabricante Texas Instruments, el cual se puede visualizar en la figura 76.

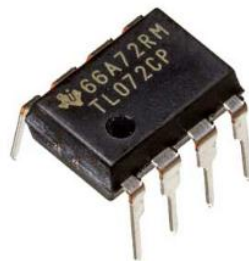


Figura 76: Operacional TL072CP.

Este operacional tiene un encapsulado DIP (Dual In-line Package) con la configuración de pines mostrada en la figura 77.

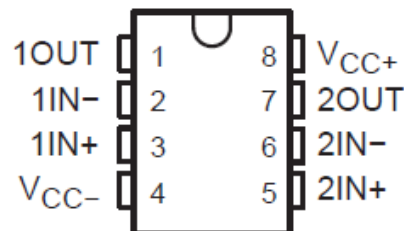


Figura 77: Configuración de pines TL072CP.

El circuito de acondicionamiento de señales, se ha diseñado con el software DesigSpark, para incluirlo en una PCB (*Printed Circuit Board*). El esquemático implementado es el de la figura 78.

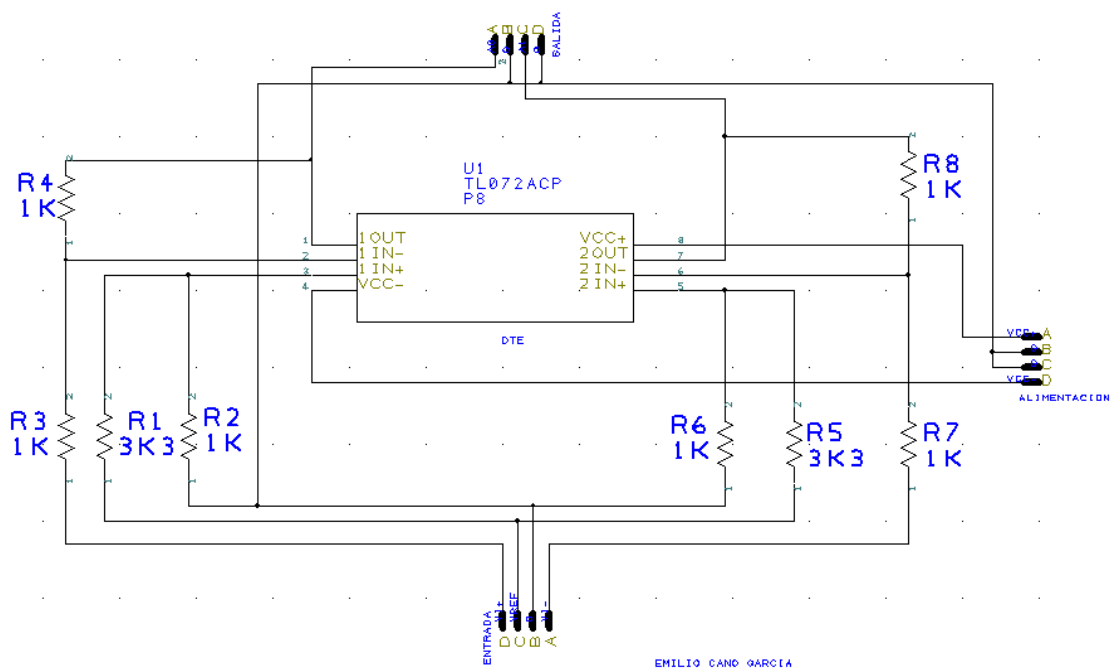




Figura 78: Esquemático circuito acondicionamiento de señal

La PCB se ha realizado en el laboratorio con la asistencia del profesor de taller. El resultado se muestra en la figura 79.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 98 de 120

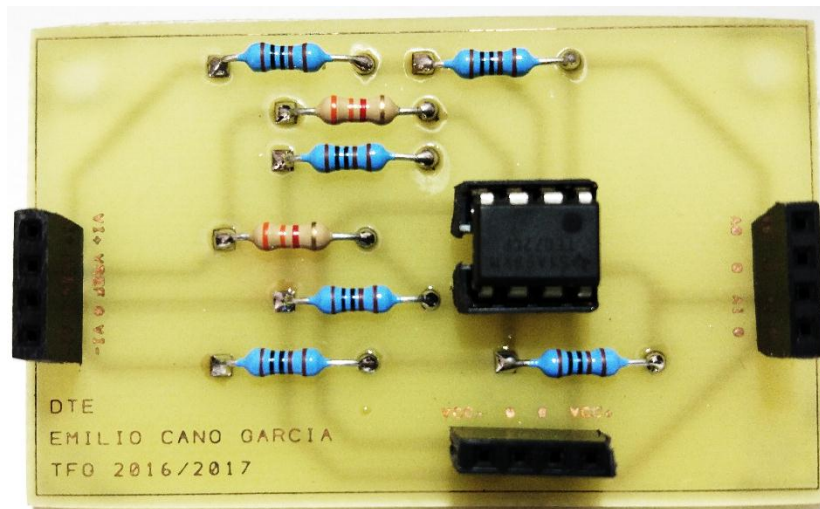


Figura 79: PCB Circuito de acondicionamiento de señal.



Este circuito tiene una tensión de referencia de 3.3V proporcionada por el pin de la placa de desarrollo, una tensión de alimentación de operacional de $\pm 15V$, dos señales de entrada de audio VI+ y VI-, dos señales de salida A0 y A1 para los dos canales del pmodAD1 y una toma de tierra para todo el circuito. La conexión con el pmodAD1 se realiza mediante el conector indicado en la figura 80.



Figura 80: Cable conexión PmodAD1.

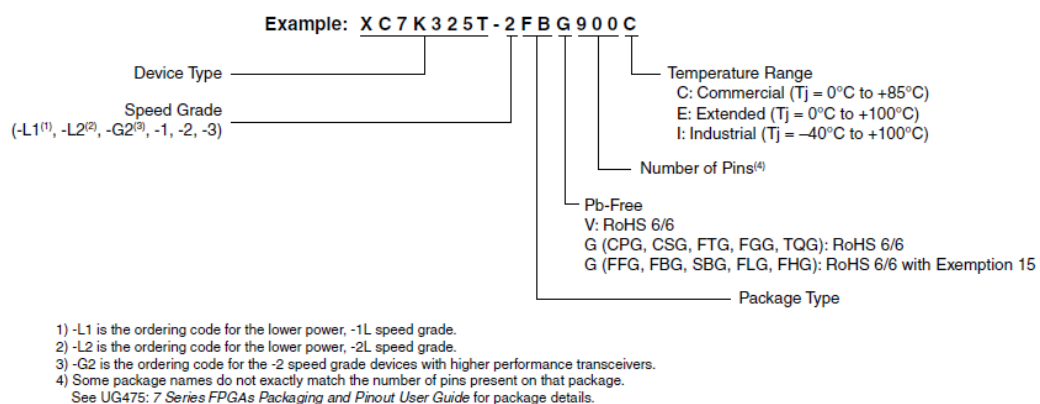
6.5. FPGA y placa de desarrollo

En este proyecto se utiliza una FPGA del fabricante Xilinx, en concreto una perteneciente a la serie 7 de la familia Artix. Las FPGAs de esta familia están optimizadas para aplicaciones de baja potencia que requieren transceptores serie, DSP de alto velocidad de procesamiento y rendimiento lógico. Según el

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 99 de 120

fabricante, los dispositivos de esta familia tienen un coste bajo, teniendo en cuenta el alto rendimiento y la lista de materiales que contienen.

Dentro de esta familia, la FPGA escogida se trata de la XC7A100T comercial con el encapsulado CSG324 (324 Ball Chip-Scale BGA). La referencia del dispositivo o partnumber es el XC7A100T-1CGS324C, el cual se puede interpretar observando la información del pedido que proporciona el datasheet del fabricante, en la figura 81. En la figura 82 se muestra el chip.



DS180_01_042916



Figura 81: Ordering Information

Las características más importantes de esta FPGA son las siguientes:

- 101440 celdas lógicas.
- 15850 slices. Cada uno de ellos tiene 4 LUTs y 8 flip-flops, lo cual, proporciona un máximo de 1.188Kb de memoria RAM distribuida.
- 4860 kbits en bloque de memoria RAM.
- 240 DSP slices
- Velocidad de reloj interna superior a 450MHz
- Un convertidor analógico digital (XADC)



Figura 82: FPGA ARTIX-7 XC7A100T-1CGS324C

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 100 de 120

La placa de desarrollo que utiliza esta FPGA se trata de la Nexys4 DDR del fabricante Digilent. Esta placa tiene incorporada una serie de puertos, periféricos y memoria externa, con la cual se pueden realizar tanto diseños de circuitos combinacionales simples, hasta diseños de procesadores complejos. Varios periféricos incorporados, incluidos acelerómetro, sensor de temperatura, micrófono, amplificador de altavoz y varios dispositivos de entrada salida, hacen que esta placa permita realizar una amplia gama de diseños sin la necesidad de utilizar otros componentes.

Los puertos y periféricos que proporciona la placa, son los siguientes.

- 16 user switches
- USB-UART Bridge
- 12-bit VGA output
- 3-axis accelerometer
- 128MiB DDR2
- Pmod for XADC signals
- 16 user LEDs
- Two tri-color LEDs
- PWM audio output
- Temperature sensor
- Serial Flash
- Digilent USB-JTAG por fot FPGA programming and communication
- Two 4-digit 7-segment displays
- Micro SD card connector
- PDM microphone
- 10/100 Ethernet PHY
- Four Pmod ports
- USB HID Host for mice, keyboards and memory sticks.

En la figura 83 se visualiza la placa de desarrollo indicando cada uno de los periféricos detallados en al table 7.

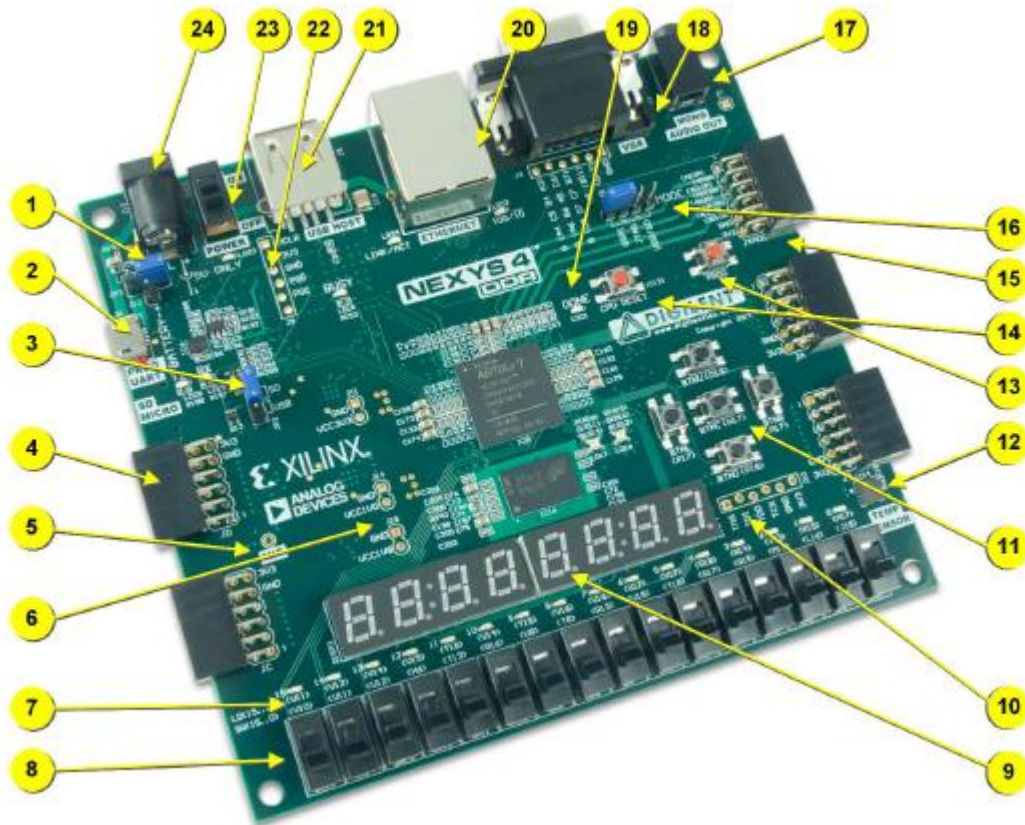






Figura 83: Placa de desarrollo Nexys4 DDR

Tabla 7: Periféricos de la placa de desarrollo Nexys4 DDR

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod port (XADC)
4	Pmod port(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 102 de 120

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 103 de 120

7.CONCLUSIONES

En este proyecto se ha realizado un procesamiento digital de audio implementado en una FPGA.



Al comienzo del trabajo, primero se realizó la comprobación del sistema conformado por las interfaces de los convertidores y la FPGA, sin realizarle ninguna modificación a la señal. En este paso se tuvieron que generar diferentes frecuencias de reloj, requeridas por el PmodI²S y se tuvo que crear el módulo central **sistema_dsp** para controlar todo el sistema. Posteriormente, en la comprobación del funcionamiento del sistema, se identificó que se requería de un circuito de acondicionamiento de señal, el cual se diseñó y se implementó en el laboratorio. Ya con el circuito de acondicionamiento diseñado, se comprobó el correcto funcionamiento del sistema, verificando con diferentes tipos de señales de entrada, que la señal de salida era la misma que la entrada. Estas comprobaciones se hicieron mediante el osciloscopio y percepción auditiva.

Una vez se acabó este punto, se realizó un estudio sobre las diferentes estructuras para diseñar filtros y los distintos efectos auditivos. Se escogieron los filtros basados en retardadores.

Estos filtros se basan en acumuladores, lo que conlleva almacenar muestras. Para ello, se estudió los diferentes métodos de almacenamiento de una FPGA y se implementaron cada uno de ellos, teniendo que diseñar una interfaz de control de escritura y lectura en memoria. De esta forma se comprobó las diferentes utilidades de cada tipo de almacenamiento y cuál de ellos era el más eficiente en nuestro diseño.

Para diseñar los filtros, se optó por utilizar la herramienta Simulink. Para ello, conforme se iban realizando las estructuras, se estudiaba las diferentes características de los bloques para saber configurarlos y así poder simular el sistema. En este paso, se dio a conocer la herramienta de generación de HDL y la herramienta System Generator de Xilinx. Como resultó bastante interesante, se dedicó un breve estudio del plug-in de Xilinx. Para ello, se tuvo que instalar el software Vivado y seguir varios tutoriales para aprender como vincular el plug-in con Matlab.

Una vez diseñados los filtros en Simulink y conociendo varias formas de transformar el código en VHDL, se optó por realizar la transformación del código



			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 104 de 120

a mano. Para ello, se modeló el funcionamiento de los bloques usados en Simulink, en código VHDL.

En el diseño del sistema, se creó un módulo central que hace de interfaz con todos los componentes y con las entradas y salidas del sistema. El componente más importante de este módulo es la interfaz con la memoria RAM, ya que todas las estructuras implementadas se basan en acumuladores.

Por último, se sintetizaron e implementaron los filtros en la FPGA para comprobar su funcionamiento. Para ello se programó las interfaces con los displays y actuadores, para de esta forma poder cambiar de efecto y visualizar de cual se trata. Los filtros más complejos han sido los caracterizados por tener un retraso modulado, los cuales suelen provocar ruidos indeseados.

Como resultado de este trabajo, se ha obtenido un sistema de procesamiento digital, formado por varias estructuras que implementan diferentes efectos.

			
	Memoria	Documento 1	
	Procesado de señales de audio mediante FPGAs.		Página 105 de 120

8.REFERENCIAS

Cirrus Logic., 2013, “10-Pin, 24-Bit, 192 kHz Stereo D/A Converter”. Datasheet of CS4344.

Digilent., 2016, “Pmod^{PS} Reference Manual”.

Digilent., 2016, “Nexys4 DDR™ FPGA Board Reference Manual”.

Analog Devices., 2011, “2.35 V to 5.25 V, 1 MSPS, 12-/10-/8-Bit ADCs in 6-Lead SC70”. Datasheet of AD7476A/AD7477A/AD7478A.

Xilinx., 2016, “7 Series FPGAs Data Sheet: Overview (DS180)”.

Xilinx., 2016, “7 Series FPGAs Memory Resources (UG473)”.

Xilinx., 2016, “7 Series FPGAs Configurable Logic Block (UG474)”.

Xilinx., 2018, “Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS181)”.

Xilinx., 2009, “System Generator for DSP User Guide (UG640)”.

Texas Instruments., 2005, “Low-Noise JFET-Input Operational Amplifiers”. Datasheet of TL07x series and TL08x series.

Chamberlin, Hal., 1987, “Musical Applications of Microprocessors”. Ed. Hayden Books, 2^o edición.

Kiran Kintali y Yongfeng Gu., 2015, “Model-Based Design with Simulink, HDL Coder, and Xilinx System Generator for DSP”. White Paper of Mathworks.

Datorro, Jon., 1997, “Effect:Design: Reverberator and Other Filters”. CCRMA, Stanford University. Stanford, USA.

Datorro, Jon., 1997, “Effect:Design: Delay-Line Modulation and Chorus”. CCRMA, Stanford University. Stanford, USA.

Ruiz, Costa-jussà, Marta y Duxans, Barrobés, Helenca., 2012, “Diseño y análisis de filtros en procesamiento de audio”. Universitat Oberta de Catalunya.

Ruiz, Costa-jussà, Marta y Duxans, Barrobés, Helenca., 2012, “Efectos digitales de la señal de audio”. Universitat Oberta de Catalunya.