UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

"Trabajo de Grado previo a la obtención del Título de Ingeniero e Ingeniera en Electrónica y Telecomunicaciones"

TRABAJO DE GRADUACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN FILTRO CON ALGORITMO ADAPTATIVO EN FPGA PARA LA CANCELACIÓN DE RUIDO

AUTORES:

MANUEL ALEJANDRO AYALA VELASCO

NADEZHDA ROCÍO CÓRDOVA ÁLVAREZ

DIRECTOR:

Ing. Fabián Gunsha

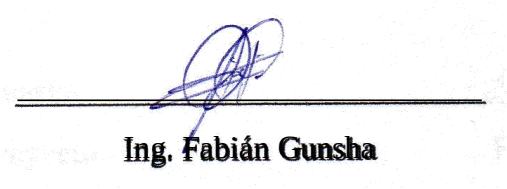
AÑO:

2016

CERTIFICACIÓN DEL DIRECTOR DEL PROYECTO

Certifico que el presente trabajo de investigación previo a la obtención del título de Ingeniero e Ingeniera en Electrónica y Telecomunicaciones. Con el tema: ***"Diseño e Implementación de un Filtro con Algoritmo Adaptativo en FPGA para la Cancelación de Ruido"*** ha sido elaborado por Alejandro Ayala y Nadezhda Córdova, el mismo que ha sido revisado y analizado en un cien por ciento por el asesoramiento permanente de mi persona en calidad de Director, por lo que se encuentran aptos para su presentación y defensa respectiva.

Es todo cuanto puedo informar en honor a la verdad.



Ing. Fabián Gunsha

REVISIÓN

Los miembros del Tribunal de Graduación del Proyecto de Investigación de título: ***"Diseño e Implementación de un Filtro con Algoritmo Adaptativo en FPGA para la Cancelación de Ruido"*,** presentado por: Manuel Alejandro Ayala Velasco y Nadezhda Rocío Córdova Álvarez, y dirigida por: Ing. Fabián Gunsha.

Una vez escuchada la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en el cual se ha constatado el cumplimiento de las observaciones realizadas, remite la presente para uso y custodia en la biblioteca de la Facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman:

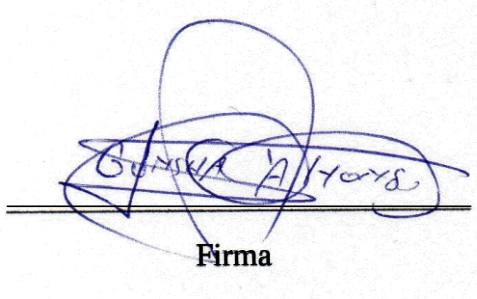
Ing. Paulina Vélez

Presidenta del Tribunal  **Firma**

Ing. Fabián Gunsha

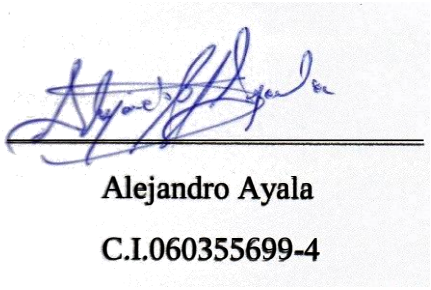
Director del Proyecto  **Firma**

Ing. Alfonso Gunsha

Miembro del Tribunal  **Firma**

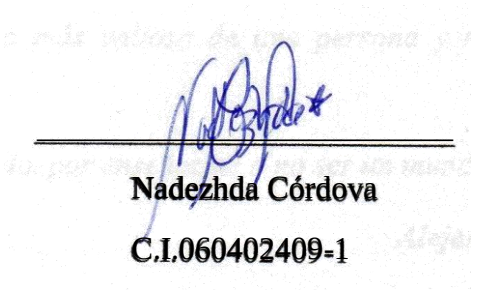
AUTORÍA DE LA INVESTIGACIÓN

La responsabilidad del contenido de este Proyecto de Graduación, nos corresponde exclusivamente a: Alejandro Ayala y Nadezhda Córdova como autores, e Ing. Fabián Gunsha en calidad de Director del Proyecto; y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.



Alejandro Ayala

C.I.060355699-4



Nadezhda Córdova

C.I.060402409-1

AGRADECIMIENTO

Cada vida es un libro distinto que muy pocos lo saben leer, pero muchos ayudan escribiéndolo, agradezco a todos quienes de una u otra manera en el trascurso del camino brindaron su ayuda para cumplir con la culminación de mi carrera universitaria.

Gracias a toda mi familia por ser un pilar muy importante en mi formación.

A mis padres Teófilo Y Alexandra, y a mi abuelita Lety, por los esfuerzos, las enseñanzas, y orientarme a ser mejor cada día.

A mis hermanas y mi hermano por su afecto incondicional.

A todos aquellos que considero amigos/as, quienes me han ayudado a perseverar en esta ardua labor.

A los docentes y al Ing. Fabián Gunsha que brindaron sus conocimientos para mi formación profesional.

A todos quienes dedicaron su tiempo lo más valioso de una persona y no lo recuperaran.

A Nade mi compañera del trabajo de grado, por enseñarme a no ser un mundano.

Alejandro.

AGRADECIMIENTO

Dicen que el lugar en el que nos encontramos es el resultado de nuestras decisiones, pero yo no creo que sea tan sencillo. Creo que en realidad nuestras decisiones por sí solas no podrían llevarnos a ningún lugar si no tenemos a alguien a nuestro lado que nos impulse a tomar esas decisiones, que nos incentive a tratar de volar para alcanzar nuestros sueños.

Por ello, al llegar a la culminación de mi carrera universitaria quiero agradecer a todos quienes han estado conmigo animándome a seguir luchando hasta el final.

A Rocío, mi mamá, por ser la imagen misma del esfuerzo y el trabajo; por enseñarme que, por amor a los hijos, se puede lograr lo que sea.

A Fabián, mi papá, por el Trébol de Cuatro Hojas de Pablo Neruda; por haberme mostrado con él los principios que guiarían mi vida.

A Samai, mi hermana, por su alegría y firmeza, por ser la hermana mayor de su hermana mayor.

A Ilich e Ytzel, mis pequeños, por sus graciosas ocurrencias, que dan luz a mis días más oscuros.

A mis abuelitos, mis tíos y mis primos, por su apoyo en todo momento.

A Pablo, por estar conmigo y ofrecerme la oportunidad de volver a empezar.

A Alejandro, mi mejor amigo y compañero en este proyecto, por su paciencia y comprensión tanto en nuestra amistad como en el desarrollo de nuestro trabajo de grado.

A Verónica y Belén, por ser unas muy valiosas amigas.

Al Ing. Fabián Gunsha, Director del Proyecto, por haber transmitido sus conocimientos hacia nosotros.

A todos ustedes y a quienes no he alcanzado a nombrar, gracias por enseñarme que el conocimiento no sirve de nada si no se aprende a Ser Persona y que "Aunque el bosque esté desierto, los colores llegarán…"

Nadezhda.

DEDICATORIA

Establecer metas es el primer paso para trasformar lo invisible en visible. Al hacerla visible esta meta la dedico

A mi abuelita, a mis padres, a mis hermanas y hermano que son mi motivo para seguir hacia adelante, no rendirme hasta lograr lo que me proponga.

A Chelita que siempre está en mi corazón y no me olvido de sus enseñanzas.

Alejandro.

Dedico este trabajo a mi hijo Ilich, esperando que, si un día llega a leerlo pueda inspirar su curiosidad e ingenio, así como él me inspira a seguir adelante para cumplir mis metas. A mi madre, Rocío, que es el pilar más importante de mi vida. Y a Rosanita, quien anhelaba más que nadie la llegada de este día, y aunque no lo haya logrado ver, siempre estuvo conmigo guiándome y lo seguirá haciendo.

Nadezhda.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo de grado muestra el diseño y la implementación de un filtro adaptativo LMS (least mean square) que tiene como objetivo principal la eliminación de señales no deseadas en frecuencias de audio. Este filtro adaptativo es programado en el tablero de desarrollo ZYBO de Xilinx, que se encuentra compuesto por una FPGA de la familia Artix – 7, mediante las plataformas Vivado y SDK; y, en conjunto con las herramientas que ofrece Matlab se completó la implementación. Las señales que se usaron como entradas al sistema de filtrado están dentro de la banda de frecuencias medias; mientras que las entradas de ruido fueron generadas internamente en la tarjeta y desde fuentes externas. Se describen también los tres escenarios de prueba a los que se sometió el filtro para determinar su correcto funcionamiento, con onda generada – ruido interno, audios pre-grabados y con audios en tiempo real.

UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CENTRO DE IDIOMAS INSTITUCIONAL

Ms. Geovanny Armas 02 de junio de 2016

SUMMARY

This graduation work shows the design and implementation of an LMS (last mean square) adaptive filter whose main objective is the elimination of unwanted signals in audio frequencies. This adaptive filter was programmed in the ZYBO (Xilinx) development board; which is composed by a FPGA that belongs to the family Artix – 7 by means of the Vivado and SDK platforms and together with the tools offered by Matlab, the implementation was completed. The signals that were used as inputs to the filtering system are within the medium frequency band; while noise inputs were generated internally on the board from external sources. The three tests scenarios to which the filter was subjected to determine its correct operation, with wave generated – internal noise, pre-recorded audio and audio in real-time are also described.

INTRODUCCIÓN

Los filtros digitales corresponden a una familia de sistemas lineales e invariantes en el tiempo, caracterizados por una función de transferencia que posee coeficientes constantes.

En algunos casos al utilizar filtros digitales en el tratamiento de señales, estos presentan requerimientos de un procesado de señales que tienden a variar en el tiempo, y la naturaleza de las variaciones no es determinable. En casos así, lo recomendable es la aplicación de filtros con la capacidad de aprender del proceso mismo, de manera que el sistema de filtrado pueda ajustarse a las particularidades de la señal a eliminar.

Para resolver los mencionados inconvenientes existe la propuesta de la utilización de filtros adaptativos, que son sistemas con la facultad de modificar sus parámetros durante el procesamiento de las señales, es decir son sistemas variantes en el tiempo. El filtro cambia sus características cumpliendo criterios predeterminados y altera dinámicamente su función de transferencia. Un sistema adaptativo puede ser implementado por software o por hardware, en las plataformas de lógica reconfigurable.

La implementación por software implica que los criterios de reconfiguración del sistema sean expresados en un algoritmo; existen programas especializados que permiten el diseño e implementación de filtros adaptativos, el más utilizado es Matlab.

Para la implementación por hardware entra a consideración el uso de las plataformas de lógica reconfigurable, el presente trabajo contempla la utilización de una FPGA que pertenece a la familia de Xilinx. El emplear sistemas reconfigurables, junto con técnicas y herramientas de diseño digital avanzado ofrece ventajas en cuanto a facilidad y flexibilidad de desarrollo, reducción de costos y proporciona una gran herramienta para el estudio y procesamiento de señales.

La FPGA permite alcanzar velocidades de hardware adecuadas para aplicaciones definidas con flexibilidad de software. La posibilidad de reutilización del hardware reconfigurable ofrece un gran beneficio debido a que puede utilizarse exactamente el mismo hardware para varias aplicaciones cambiando únicamente su programación interna.

CAPÍTULO I

1. FUNDAMENTACIÓN TEÓRICA

1.1. Antecedentes del Tema

El concepto de filtro ha sido parte integral del progreso de la ingeniería eléctrica. Varios logros tecnológicos no habrían sido posibles sin los filtros, y ya que han cumplido un papel importante, se han realizado muchos esfuerzos en relación con la teoría, el diseño y la construcción de los mismos; es por esto que muchos artículos y libros se han escrito acerca de ellos (Alexander & Sadiku, 2006).

Un filtro es un sistema que, dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. Los filtros digitales tienen como entrada una señal analógica o digital y su salida tiene otra señal analógica o digital, pudiendo haber cambiado en amplitud, frecuencia o fase, dependiendo de las características del filtro (Álvarez Cedillo, Lindig Bos, & Martínez Romero, 2008).

Los filtros de coeficientes constantes no suelen ser suficientes a la hora de tratar señales que son variantes en el tiempo o que son parte de entornos contaminados, debido a que trabajan en frecuencias fijas y por tal motivo se ven limitados en el procesamiento adecuado de la señal (Jones, 2005).

Combinar un filtro con un algoritmo adaptativo, genera posibilidades de que el sistema de filtrado se ajuste a cualquier señal de ingreso, esto permite recalcular los coeficientes del filtro de una forma dinámica y efectiva; convirtiéndolo de esta manera en un filtro adaptativo (Wantanabe, 2012).

Un filtro adaptativo varía su comportamiento de acuerdo al instante en que se aplique el estímulo. Sus características cambian en base a criterios predeterminados, es decir modifican de forma dinámica su función de transferencia; lo que hace aplicables a estos filtros en casos donde las características de la señal a eliminar cambian. (Ruiz, 2010).

Estos filtros no son invariantes en el tiempo y tampoco son lineales, lo que hace que su estudio sea más complejo que el de un filtro digital; por lo mismo, estos tipos de filtros se implementan en forma de algoritmos sobre microprocesadores, DSP o FPGA (Wantanabe, 2012).

Las FPGA permiten obtener velocidades de hardware adecuadas para determinadas aplicaciones con flexibilidad al software. La posibilidad de reutilización del hardware reconfigurable disminuye su costo, puede utilizarse exactamente el mismo hardware para varias aplicaciones cambiando únicamente su programación interna (Alba & Ruiz, 2009).

1.2. Enfoque Teórico

1.2.1. Filtro Electrónico

Un filtro es un dispositivo diseñado para dejar pasar señales con determinadas frecuencias y atenuar o rechazar todas las demás. La banda de paso de un filtro es el rango definido de frecuencias que deja pasar con una mínima atenuación, y que está delimitada por la frecuencia de corte (Floyd, 2008).

1.2.1.1. Filtro Analógico

Un filtro analógico es un sistema de tiempo continuo que obedece a una ecuación diferencial lineal con coeficientes constantes (González, 2012)

1.2.1.2. Filtro Digital

Un filtro digital es cualquier procedimiento que permite transformar los datos digitalizados en otros datos mediante un determinado algoritmo. Constituye una herramienta muy útil a la hora de tratar una señal analógica, porque se pueden variar sus parámetros (Bustamante, 1999).

Se caracterizan por ser sistemas predecibles, flexibles, simulables, consistentes y precisos. Por una parte, es posible combinar sus especificaciones mediante la reprogramación, sin la adición de componentes. De otro lado su carácter digital permite calcular y simular su respuesta usando procesadores de uso general, y

también implementar topologías no realizables mediante el uso de componente físicos convencionales (Gonzales & Parra, 2005).

1.2.1.2.1. Respuesta de Impulso Infinita (IIR)

Los filtros digitales de Respuesta Infinita al Impulso o filtros IIR (por sus siglas en inglés Infinite Impulse Response), es un filtro digital en el que, si la entrada es una señal impulso, a la salida se tendrá un número infinito de términos no nulos (Wantanabe, 2012).

1.2.1.2.2. Respuesta de Impulso Finita (FIR)

Es un acrónimo en inglés Finite Impulse Response o Respuesta Finita al Impulso. Es un filtro digital en el cual, si a la entrada se tiene una señal impulso, en la salida se tendrá un número finito de términos no nulos; la salida se basa en entradas actuales y anteriores (Álvarez Cedillo, Lindig Bos, & Martínez Romero, 2008).

1.2.2. Filtro Adaptativo

Los filtros adaptativos son sistemas que varían en el tiempo, de forma que se adaptan a cambios en su entorno, optimizando su funcionamiento de acuerdo a una serie de algoritmos conocidos como algoritmos adaptativos. La forma de determinar el comportamiento óptimo del filtro adaptativo es minimizando una función monótona creciente de la señal o secuencia de error. Esta secuencia se define como la diferencia entre una señal que se toma como referencia, o señal deseada, y la salida del filtro adaptativo (Soria Olivas, 2003).

En otras palabras, diseñar un filtro adaptativo consistirá en determinar la regla de variación de los coeficientes. El resto será automático (Albiol, Naranjo, & Prades, 2007).

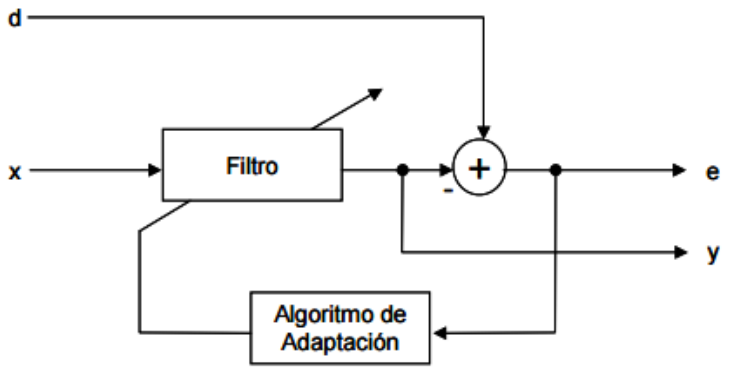


Figura 1. Estructura General de un Filtro Adaptativo

Fuente: Comparación e Implementación de los Algoritmos SLMS y OAECLMS en un DSP

Un filtro adaptativo es aquel cuyos coeficientes son actualizados mediante un algoritmo que cumple con un criterio predefinido, que puede ser minimizar el error cuadrático medio, como es el caso del LMS (Zelaya , 2004).

1.2.3. Algoritmo Adaptativo LMS

El algoritmo LMS es un algoritmo de gradiente estocástico en el que se hacen iteraciones de cada valor de los coeficientes de un filtro transversal en la dirección del gradiente con la magnitud al cuadrado de la señal de error con respecto al valor del coeficiente. La principal característica del algoritmo LMS es que utiliza un parámetro de tamaño de paso fijo para controlar la corrección aplicada a cada valor de los coeficientes de una iteración a la próxima (Zelaya , 2004).

El algoritmo LMS se compone de dos procesos básicos: Uno de filtraje y uno adaptativo. El primero involucra la salida procesada de un filtro lineal en respuesta a una señal de entrada y la generación de una estimación de error resultante de la comparación de la salida con la respuesta deseada. El segundo implica el ajuste automático de parámetros del filtro de acuerdo con la estimación del error (Solarte & Jojo, 2012).

El algoritmo LMS fue desarrollado por Widrow y Hoff en 1960, y su construcción se basa en el algoritmo de la máxima pendiente, el cual emplea la ecuación

$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \frac{1}{2}\mu\boldsymbol{\nabla J}(n)$, n = 1, 2, ..., para la actualización del vector de pesos $\boldsymbol{w}(n)$ (Haykin, 2014).

Donde μ es el parámetro de tamaño de paso y $\boldsymbol{\nabla J}(n)$ es el vector gradiente, el cual está dado por $\boldsymbol{\nabla J}(n) = -2\boldsymbol{p} + \boldsymbol{R}\boldsymbol{w}(n)$.

Donde $\boldsymbol{p}$ es el vector de correlación cruzada entre el vector de entradas $\boldsymbol{u}(n)$ y la respuesta deseada $d(n)$, y $\boldsymbol{R}$ es la matriz de correlación de las entradas. Además, se supone que $\boldsymbol{p}$ y $\boldsymbol{R}$ son conocidas. Una buena aproximación del vector gradiente $\boldsymbol{\nabla J}(n)$, se obtiene al realizar cálculos instantáneos de las matrices $\boldsymbol{p}$ y $\boldsymbol{R}$, que se basan en valores muestreados del valor de entrada y la respuesta deseada, así (Haykin, 2014):

$$\widehat{\boldsymbol{p}}(\boldsymbol{n}) = \boldsymbol{u}(n)d^*(n)$$

$$\widehat{\boldsymbol{R}}(\boldsymbol{n}) = \boldsymbol{u}(n)u^H(n)$$

En consecuencia, el valor estimado del vector gradiente sería[1]:

$$\widehat{\boldsymbol{\nabla J}}(\boldsymbol{n}) = -2\boldsymbol{u}(n)d^*(n) + 2\boldsymbol{u}(n)u^H(n)\widehat{\boldsymbol{w}}(n)$$

Por lo tanto, se obtiene una nueva relación recursiva para actualizar el vector de pesos:

$$\widehat{\boldsymbol{w}}(n+1) = \widehat{\boldsymbol{w}}(n) + \mu\boldsymbol{u}(n)[d^*(n) - \boldsymbol{u}^H(n)\widehat{\boldsymbol{w}}(n)]$$

La cual, constituye la ecuación de actualización del vector de pesos $\widehat{\boldsymbol{w}}(n)$, empleada por el algoritmo LMS (Haykin, 2014).

Por otra parte, el origen del algoritmo LMS se basa totalmente en señales estocásticas estacionarias; no obstante, este algoritmo es también aplicable en ambientes determinísticos, y estocásticos no estacionarios (Haykin, 2014).

[1] El superíndice H denota que se trata de un vector Hermitiano. Se dice que un vector o matrices de valores complejos es Hermitiano, si es igual a su traspuesta conjugada (Haykin, 2014).

Finalmente teniendo cuenta la figura … el filtro LMS funciona de la siguiente manera:

Dados, un vector de entrada de tamaño N en el tiempo n:

$$\boldsymbol{u}(n) = [u(n), u(n-1), \dots, u(n-M+1)]^T$$

Un vector de pesos, cuya condición inicial es $\widehat{\boldsymbol{w}}(0)$, y la respuesta deseada $d(n)$, se debe calcular el vector de pesos $\widehat{\boldsymbol{w}}(n+1)$ en el tiempo n+1.

$$\widehat{\boldsymbol{w}}(n+1) = \widehat{\boldsymbol{w}}(n) + \mu \boldsymbol{u}(n) e^*(n)$$

Donde el error de estimación está dado por:

$$e(n) = d(n) - y(n)$$

Y la salida del filtro es:

$$y(n) = \widehat{w}^H(n)\boldsymbol{u}(n)$$

1.2.4. Ruido

El ruido es un fenómeno generalmente imposible de controlar que se encuentra presente en cualquier sistema de comunicaciones, y que contribuye con el deterioro de la señal de salida; por lo tanto, el ruido, se puede considerar como una señal parásita, aunque el término señal no sea tan aplicable ya que el ruido no presenta información alguna, excepto en casos bastante excepcionales (Pérez, 2007).

El ruido puede presentarse en diferentes formas debido a su naturaleza, es así que Hernández (2003) muestra una clasificación del ruido en dos categorías: correlacionado y no correlacionado. El ruido correlacional es aquel existe solo cuando una señal está presente, y el ruido no correlacionado es el que está presente independientemente que exista o no señal.

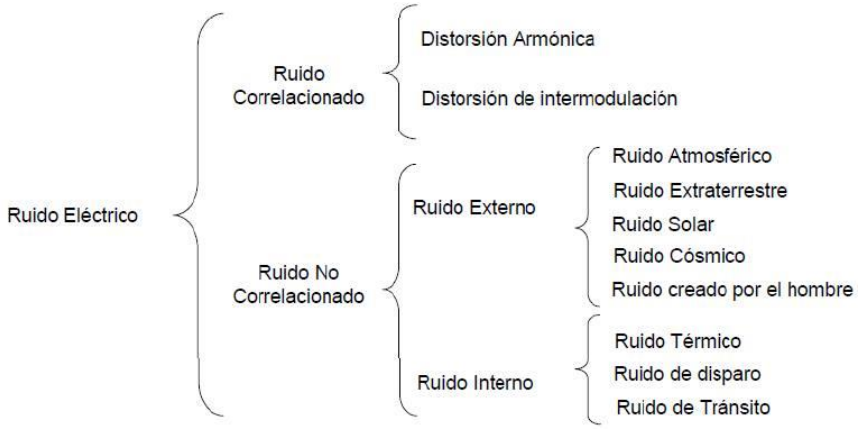


Figura 2. Principales formas de Ruido Eléctrico

Fuente: Análisis Comparativo de Algoritmos para Reducción de Ruido en Señales Utilizando Wavelets

1.2.5. FPGA

Una FPGA (Field Programmable Gate Array) es un dispositivo lógico que contiene una matriz bidimensional de celdas lógicas genéricas y conmutadores programables (Chu, 2011).

Cada chip de FPGA está hecho de un número limitado de recursos predefinidos como interconexiones programables para implementar un circuito digital reconfigurable y bloques de E/S para permitir que los circuitos tengan acceso al mundo exterior (National Instruments Corporation, 2012).

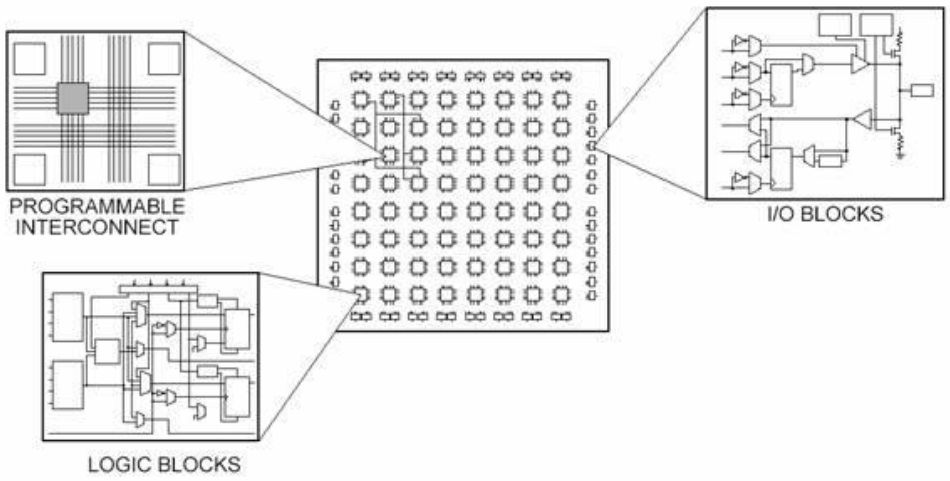


Figura 3. Partes de una FPGA

Fuente: National Instruments

Una célula lógica se puede configurar (es decir, programar) para realizar una función simple, y un interruptor programable se puede personalizar para proporcionar interconexiones entre las celdas lógicas. Un diseño personalizado se puede implementar mediante la especificación de la función de cada celda lógica y establecer selectivamente la conexión de cada conmutador programable. Una vez que el diseño y la síntesis se completa, se puede utilizar un cable adaptador simple para descargar la celda lógica deseada y cambiar la configuración al dispositivo FPGA y obtener el circuito de medida (Chu, 2011).

1.2.6. ZYBO

La ZYBO (ZYnq BOard) es un tablero de desarrollo con varias características; listo para usar, con un software de entrada de nivel embebido y un circuito digital construido alrededor del miembro más pequeño de la familia Xilinx Zynq 7000, Z-7010. El Z-7010 está basado en la arquitectura Xilinx ALL PROGRAMABLE SYMSTEM-ON-CHIP, que integra estrechamente un procesador ARM Cortex-A9 con una FPGA de la familia Artix-7 (DIGILENT, 2016).

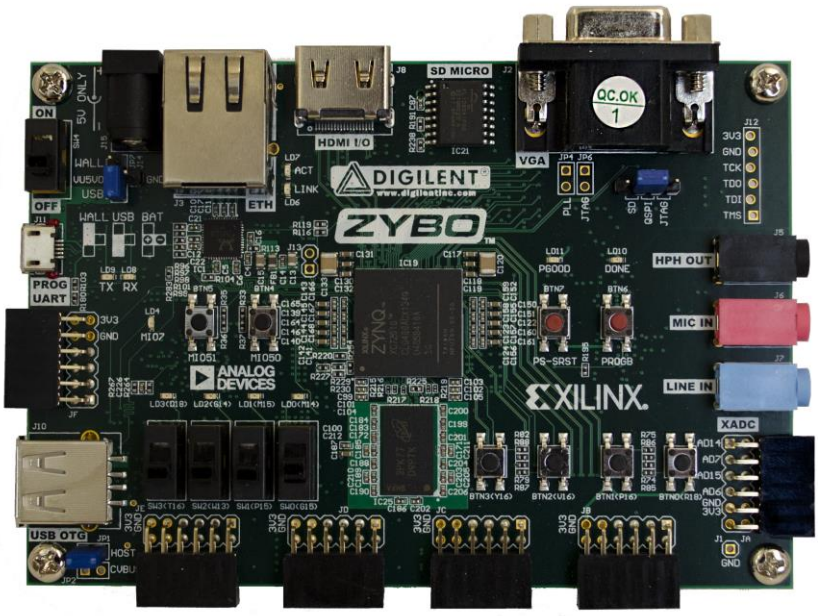


Figura 4. Tablero de Desarrollo ZYBO

Fuente: Digilent

El tablero ZYBO cuenta también con un amplio conjunto de periféricos multimedia y de conectividad, como son puertos I/O de audio y video, puertos USB de doble funcionalidad, puerto Ethernet y una ranura para memoria SD, por nombrar algunos de ellos; listos para ser usados sin necesidad de adicionar hardware. Además, seis puertos Pmod están disponibles para la integración de circuitería externa[2].

El ZYBO es compatible con el nuevo software de alto rendimiento de Xilinx, Vivado Suite Design, así también con el conjunto de herramientas ISE/EDK. Este conjunto de herramientas combina el diseño lógico de FPGA con el desarrollo de Software de ARM, en un formato fácil de usar e implementar (DIGILENT, 2016).

Las bases para la implementación de este proyecto se asientan en el chip ZYNQ XC7Z010-1CLG400C y el codificador SSM2603 del bloque de audio en el tablero.

1.2.6.1. ZYNQ XC7010-1CL400C

El Zynq es un nuevo tipo de dispositivo que combina la estructura FPGA con un poderoso procesador de aplicaciones, y por lo tanto sus características, capacidad y aplicaciones potenciales generan más ventajas que las de una FPGA o un procesador por individual (Crockett, Elliot, Enderwitz, & Stewart, 2014).

La composición general de la Zynq se fundamenta en dos secciones: el Sistema Lógico Programable (PL) y el Sistema de Procesamiento (PS). Estos pueden ser usados de forma independiente o conjunta, es decir si solo se desea usar una de las dos partes la otra se desactivará, debido a que tienen dominios de poder separados.

La PL contiene varios puertos y buses destinados al acoplamiento con la PS; no posee la misma configuración de hardware como una típica FPGA y debe estar configurado ya sea directamente por el procesador o mediante el puerto JTAG, y este último es el que ha sido aplicado para el presente proyecto (DIGILENT, 2016).

[2] Para ver la composición completa del tablero de desarrollo y sus especificaciones ver el Manual de Referencia en el enlace: https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf

La PS tiene varios componentes, incluyendo la Unidad de Aplicación de Procesamiento (APU), especificación AMBA (Bus de Microcontrolador de Arquitectura Avanzada) para interconexión, Controlador de memoria DDR3 y varios controladores de periféricos con sus entradas y salidas multiplexadas de 54 pines (llamados multiplexed I/O, pines MIO). Los controladores de periféricos que no tienen sus entradas y salidas conectadas a los pines MIO, pueden obtener su ruta de E/S a través de la PL, mediante la interfaz Extendida-MIO (EMIO); también, se encuentran conectados al procesador como esclavos vía interconexión AMBA y contienen registros de control de lectura/escritura que son direccionables en el espacio de memoria del procesador (DIGILENT, 2016).

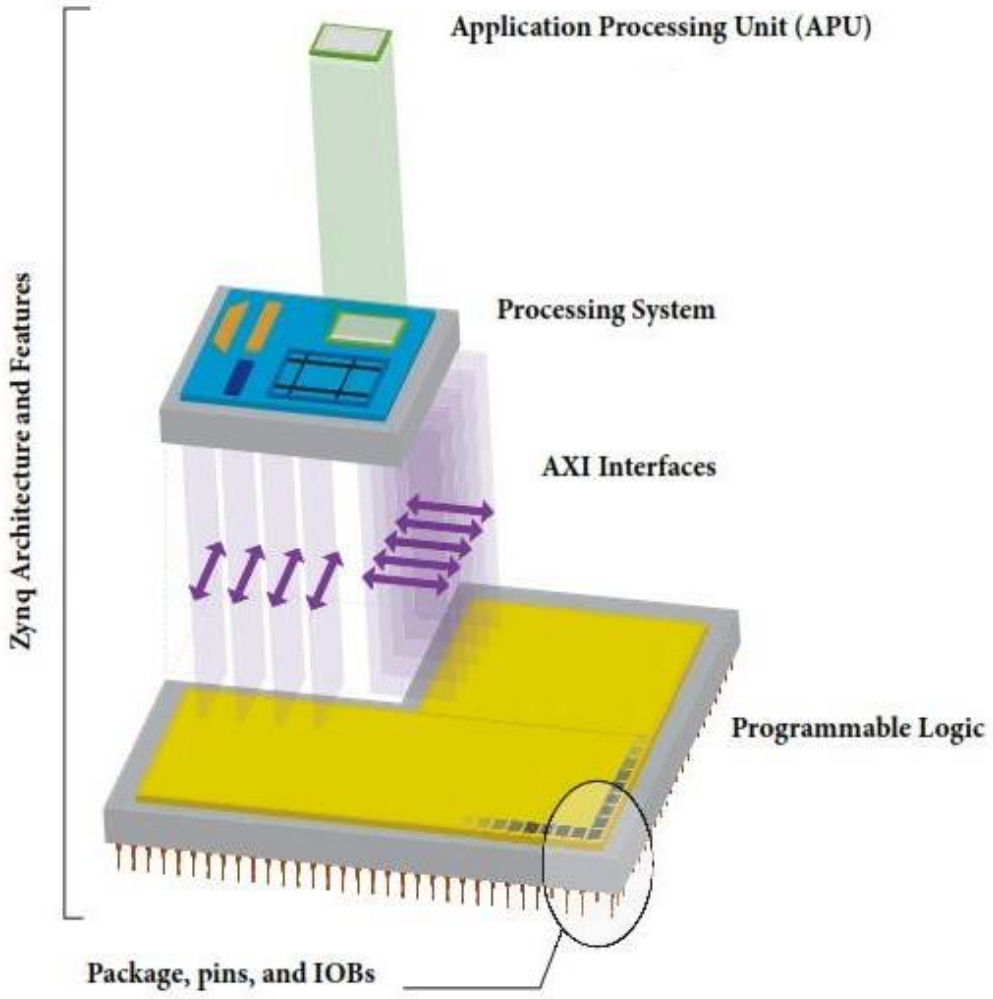


Figura 5. Arquitectura General del ZYNQ

Fuente: Libro ZYBO

La PL también está conectada como esclavo, y los diseños puede implementar múltiples núcleos en la estructura FPGA, que también contiene cada uno de los registros de control direccionables. (DIGILENT, 2016)

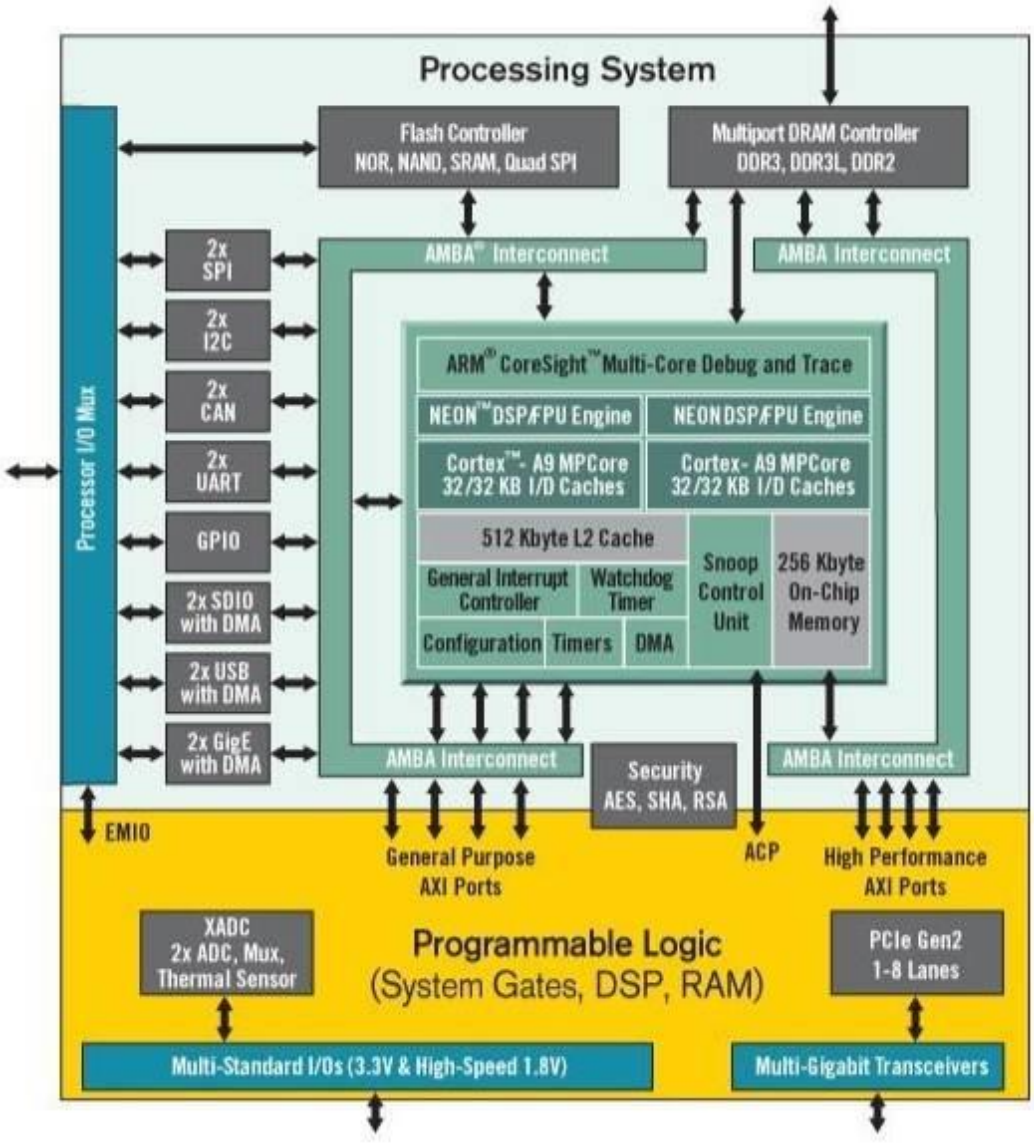


Figura 6. Arquitectura de la ZYNQ

Fuente: Manual de Referencia ZYBO

1.2.6.2. LOW POWER AUDIO CODEC SSM2603

El SSM2603 proporciona un procesamiento de audio digital integrado para el Zynq 7000. Permite el registro y la reproducción de audio con frecuencias de muestro de 8 a 96kHz. En el lado analógico, el codificador se conecta

a tres conectores audio estándar de 3.5 mm. Hay dos entradas: un micrófono mono estéreo y una línea de ingreso estéreo. Hay una salida estéreo, el conector de auriculares (DIGILENT, 2016).

Cuenta con dos canales de conversión analógico-digital (ADC) de 24 bits y dos canales de conversión digital analógico (DAC) de 24 bits (ANALOG DEVICES, 2013).

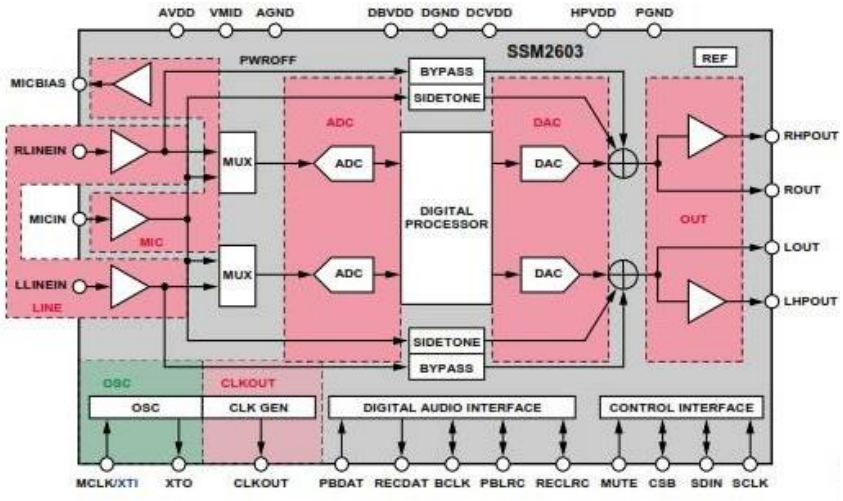


Figura 7. Estructura Funcional de del Codificador SSM2603

Fuente: Data Sheet SSM2603

La interfaz digital del SSM2603 está conectada hacia la PL del ZYNQ. Los datos de audio se transfieren mediante el protocolo I2S. La configuración se realiza a través de un bus I2C (DIGILENT, 2016).

1.2.7. Matlab

Matlab es un entorno de trabajo para el cálculo científico; su nombre es la abreviatura de MATrix LABoratory dado que el tipo de dato básico que gestiona es una matriz. Puede ser utilizado en computación matemática, modelo y simulación, análisis y procesamiento de datos, visualización y representación de gráficos, así como el desarrollo de algoritmos.

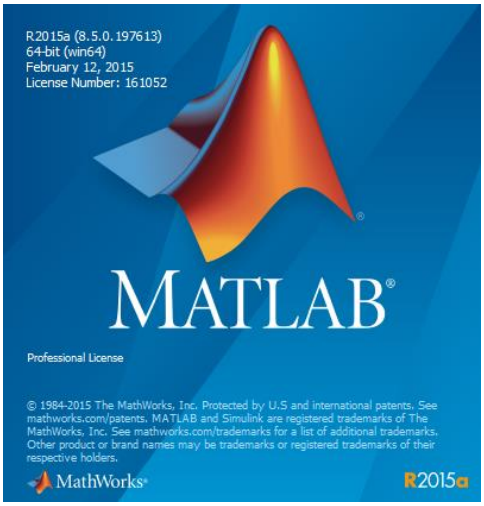


Figura 8. Logo de Matlab

Fuente: Mathworks

En el desarrollo de este proyecto ha sido utilizado para la creación del núcleo IP que contiene la función del Filtro Cancelador de Ruido LMS. Se utiliza la herramienta HDL Coder de MathWorks para trasformar un bloque de Simulink en un modelo basado en una descripción RTL (ver código VHDL en el Anexo 4), que será empaquetado para usarlo en el Catálogo de IP de Vivado.

1.2.8. Diseño de Software y Hardware

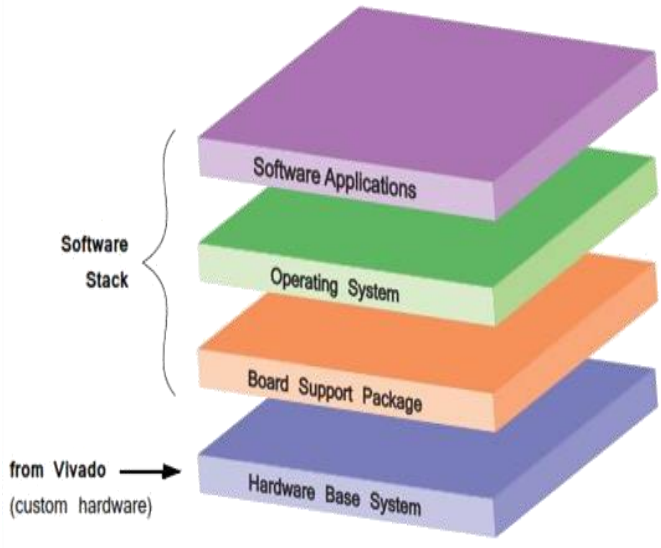


Figura 9. Capas de diseño Zynq de Software y Hardware

Fuente: Libro Zynq

1.2.8.1. Desarrollo de Hardware

El desarrollo del sistema de hardware consiste en el diseño de los bloques periféricos y otra lógica que se ejecutará en la PL, creando las conexiones adecuadas entre estos bloques y el PS, y la configuración adecuada del PS (Crockett, Elliot, Enderwitz, & Stewart, 2014).

Para el diseño en una FPGA se utiliza el Lenguaje de Descripción de Hardware, en este caso VHDL. Para el desarrollo de este proyecto se ha empleado la arquitectura de descripción por Flujo de Datos o RTL (Lógica de Transferencia de Registros).

1.2.8.2. Desarrollo de Software

El software (en el PS) se utilizará para implementar tareas de propósito general, de procesamiento secuencial, un sistema operativo, las aplicaciones y las interfaces gráficas de usuario (Crockett, Elliot, Enderwitz, & Stewart, 2014).

Las capas de software para el desarrollo de este proyecto están definidas como:

- Aplicación de Software
- Sistema Operativo Standalone: Es un entorno simplificado, semi-organizado, y un con un único subproceso, que ofrece funciones básicas tales como entrada/salida estándar y el acceso a los recursos de hardware del procesador.
- BSP: Permite la interacción entre la PS y la PL. Contiene todas las librerías estándar y drivers del diseño importado desde Vivado.

1.2.8.3. Herramientas de Diseño

Vivado e ISE Suite pertenecen a la Edición de Sistemas de Diseño de Vivado del que es propietario Xilinx. Con el entorno que provee este conjunto de herramientas el usuario es capaz de desarrollar todo tipo de aplicaciones para las FPGA. Integran todo el software necesario para el uso del hardware reconfigurable en sus productos.

Figura 10. Herramientas de Diseño de Software y Hardware

Fuente: Autores

1.2.8.4. Vivado Design Suite

En Vivado Design Suite se puede diseñar hardware mediante lenguaje VHDL escribiendo directamente la programación o a través de núcleos IP mediante diagramas de bloques; para que después sea el mismo Vivado el generador del archivo que contiene el código VHDL con la descripción de la arquitectura de hardware diseñada.

Otra funcionalidad de este software es que permite la síntesis de la arquitectura de hardware en un archivo bitstream, haciendo compatible la mencionada arquitectura con el tablero de desarrollo y permitiendo exportar la misma para conjugarla con el stack de software, que dará lugar a la implementación.

1.2.8.5. SDK

SDK es un conjunto de diseño de software basado en la popular plataforma eclipse, que incluye la compatibilidad de controladores para el todo el soporte de núcleos IP de Xilinx, biblioteca GCC para ARM y extensiones NEON utilizando los lenguajes C y C++.

A partir del flujograma de la figura 11, se diseña la aplicación en C++, para la función LMS.

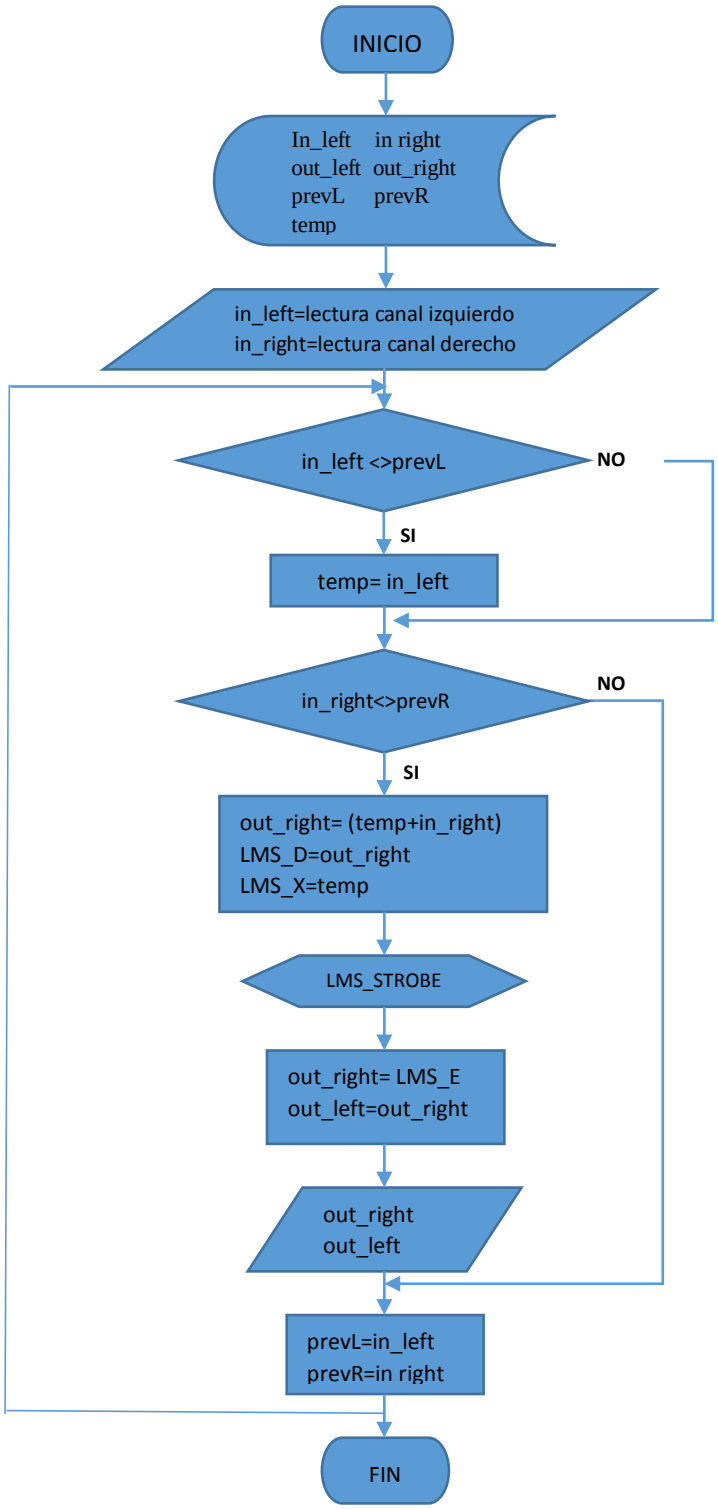


Figura 11. Flujograma de la asignación de entradas y salidas para el algoritmo LMS

Fuente: Autores

1.3. Glosario de Términos Básicos

ALGORITMO ADAPTATIVO

Proceso matemático capaz de adaptarse a los cambios que se produzcan a través del tiempo.

AMBA

Bus de Microcontrolador con Arquitectura Avanzada.

APU

Unidad de Aplicación de Procesamiento.

AXI

Interfaz Extensible Avanzada.

AXI4LITE

Enlace simplificado que soporta únicamente una transferencia de datos por conexión.

BISTREAM

Archivo creado como parte del flujo de diseño para la programación de la PL con la configuración deseada.

EMIO

Entradas y salidas multiplexadas extendidas.

FILTRO ADAPTATIVO

Sistema digital compuesto por un filtro lineal programable, cuyos coeficientes se pueden reprogramar en base a un algoritmo adaptativo.

FPGA

Matriz de compuertas programables por campo.

GPIO

Entradas y salidas de propósito general.

HDL

Lenguaje de Descripción de Hardware.

HLS

Alto Nivel de Síntesis. Software que permite la creación de componentes de sistemas de acuerdo a las especificaciones del usuario.

IDE

Entorno de desarrollo integrado.

IP

Bloque funcional de Propiedad Intelectual que corresponden a los componentes periféricos.

I2C

Bus de datos serial.

I2S

Estándar eléctrico de bus serial usado para interconectar circuitos de audio digital.

JTAG

Protocolo de prueba estandarizado.

LMS

Mínimos Cuadrados Medios.

MIO

Entradas y salidas multiplexadas.

NCO

Oscilador Controlado Numéricamente.

PERIFÉRICO

Bloque funcional dentro de un sistema integrado que está lógica y físicamente separado del ´procesador principal.

PL

Lógica Programable.

PS

Sistema de Procesamiento.

RTL

Lógica de Transferencia de Registros

SDK

Kit de Desarrollo de Software. Entorno de diseño para la creación de aplicaciones embebidas.

UART

Receptor/Transmisor Asincrónico Universal.

VHDL

Lenguaje de descripción de hardware VHSIC.

VHSIC

Circuito integrado de muy alta velocidad.

VIVADO

Paquete de software Xilinx utilizado para el desarrollo de sistemas de Zynq.

XDC

Extensión del archivo que contiene las limitaciones para el hardware.

XILINX

Compañía Americana que se dedica a la producción de Dispositivos Lógicos Programables.

ZYBO

Tablero de desarrollo que incorpora un chip Zynq y una gran variedad de periféricos.

ZYNQ-7000

Término general para la primera generación de dispositivos Zynq.

CAPÍTULO II

2. METODOLOGÍA

2.1. Tipo De Estudio

2.1.1. Nivel de Investigación

El nivel de la investigación para este proyecto es el Descriptivo, debido a que se caracteriza el diseño y la implementación de un filtro con algoritmo adaptativo en una FPGA para la cancelación de ruido, con la finalidad de conocer la estructuración que debe tener este, y su comportamiento en los escenarios de prueba a los que es sometido.

2.1.2. Diseño de la Investigación

El diseño de la investigación es Experimental, porque el filtro con algoritmo adaptativo LMS en una FPGA es sujeto de pruebas en tres escenarios, con onda generada – ruido interno, audios pre-grabados y audios en tiempo real; con el propósito de observar las respuestas que produce el mencionado filtro.

2.2. Técnicas e Instrumentación

2.2.1. Técnicas

Las técnicas usadas en este proyecto son la observación indirecta y la comprobación auditiva, es decir que a través de la vista (con instrumentos especializados) y el oído se captan las respuestas del filtro adaptativo LMS en FPGA, cuando es probado en los diferentes escenarios propuestos.

2.2.2. Instrumentación

Los instrumentos necesarios para esta investigación son libros, artículos de revistas, investigaciones previas, etc., para la parte documentada. En lo que se refiere a la parte de pruebas se utilizan el osciloscopio y el software WavePad.

2.3. Población y Muestra

2.3.1. Población

La población comprende todos los tipos de ruidos que contaminan la señal de audio a tratar, generados desde diferentes fuentes, como un bloque interno en la tarjeta que emite ruidos tonales, ruidos de diferentes entornos pre-grabados y en tiempo real.

2.3.2. Muestra

La muestra se determina mediante muestreo intencional, es decir, que se establece por el criterio de los autores.

2.4. Hipótesis

La implementación de un filtro con algoritmo adaptativo LMS utilizando una FPGA cancelará el ruido en señales de audio.

2.5. Operacionalización de las Variables

VARIABLES	DIMENSIONES	INDICADORES	INSTRUMENTOS
Independiente: Filtro Adaptativo LMS en FPGA	Eficiencia del Algoritmo Adaptativo	Velocidad de adaptación	Osciloscopio WavePad
		Respuesta en frecuencia	
Dependiente: Cancelación de Ruido	Existencia de la Señal Contaminante	Percepción del ruido	Audición

Tabla 1. Operacionalización de Variables

Fuente: Autores

2.6. Procedimientos

El presente proyecto se desarrolla en cuatro etapas, las mismas que se describen a continuación.

2.6.1. Filtro Adaptativo LMS

El algoritmo LMS se compone de dos procesos básicos: uno de filtrado y otro adaptativo. En el primero implica realizar, en cada iteración, una estimación del error cuadrático medio a través de la comparación de la señal de salida con la señal deseada. Para el segundo se realiza una actualización del vector de pesos que permita realizar el ajuste de los coeficientes del filtro.

El bloque de Simulink del filtro LMS permite implementar un filtro adaptativo FIR utilizando un algoritmo LMS mediante la creación de un subsistema regido por las siguientes ecuaciones:

$$y(n) = \boldsymbol{w}^T(n-1)\boldsymbol{u}(n)$$

$$e(n) = d(n) - y(n)$$

$$\boldsymbol{w}(n) = \propto \boldsymbol{w}(n-1) + \mu e(n)\boldsymbol{u}^*(n)$$

Variable	Descripción
n	Índice de tiempo presente.
$\boldsymbol{u}(n)$	Vector de muestras de entrada asignadas temporalmente en el paso n.
$\boldsymbol{u}^*(n)$	La conjugada compleja del vector de muestras de entrada asignadas temporalmente en el paso n.
$\boldsymbol{w}(n)$	El vector de peso del filtro estimado en el paso n.
$y(n)$	Salida filtrada en el paso n
$e(n)$	Error estimado en el paso n.
$d(n)$	Respuesta deseada en el paso n.
μ	Tamaño del paso de adaptación.

$\propto$	Factor de fuga $(0 < \propto \leq 1)$

Tabla 2. Descripción de Variables usadas en el Bloque LMS

Fuente: Mathworks

El subsistema estima los pesos o coeficientes del filtro, necesarios para minimizar el error, e(n), entre la señal de salida y(n) y la señal deseada, d(n). Este proceso se describe en la figura 11 que contiene el flujograma del proceso.

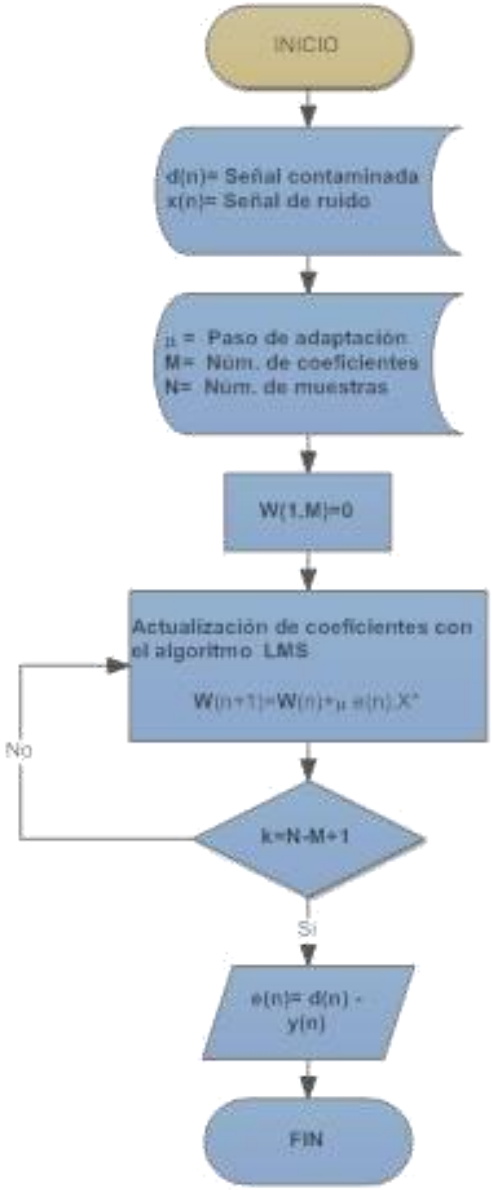


Figura 12. Diagrama de Fuljo del Algoritmo LMS en el subsistema de Simulink

Fuente: Autores

La señal que desea filtrar se introduce en el puerto de entrada y esta puede ser un escalar o un vector columna. La señal deseada se conecta al puerto de referencia, y debe tener el mismo tipo de datos, complejidad y dimensiones que la señal de entrada.

El puerto de salida emite la señal de entrada filtrada, que es la estimación de la señal deseada. El puerto de error muestra el resultado de restar la señal de salida de la señal deseada.

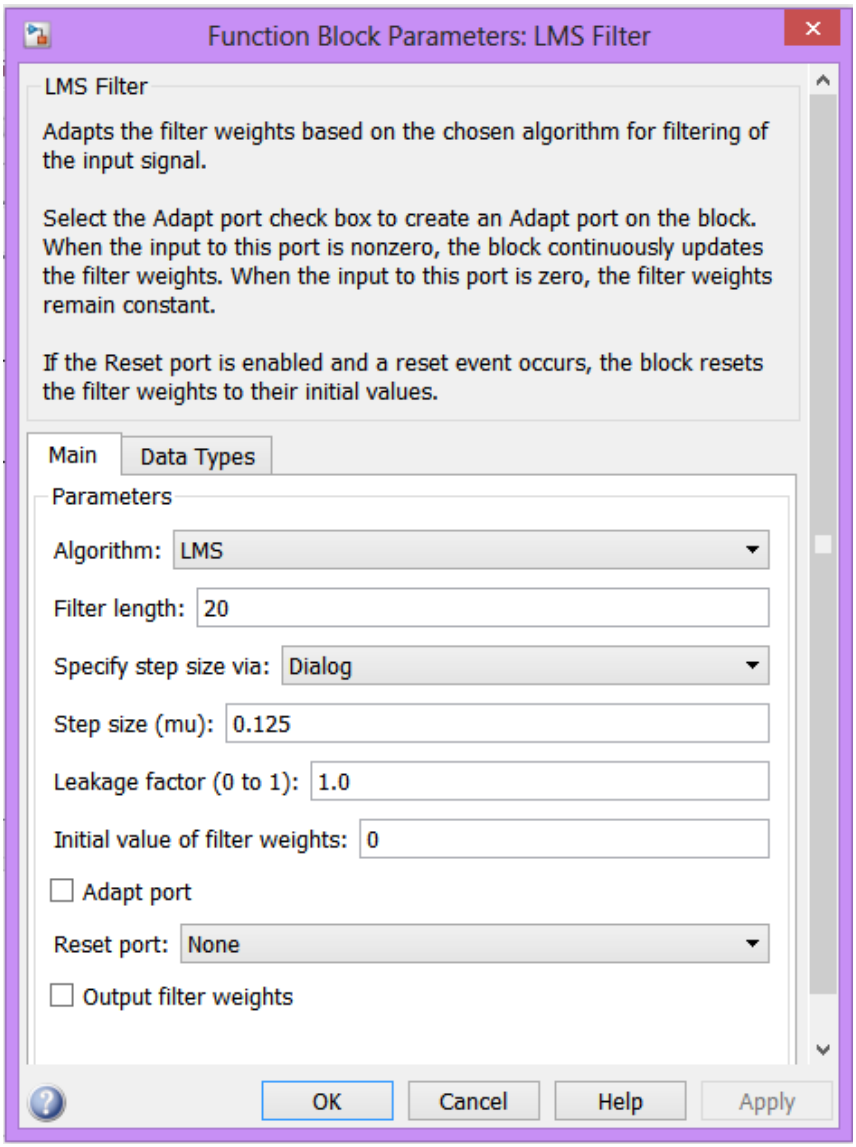


Figura 13. Parámetros del Bloque LMS

Fuente: Autores

2.6.2. Matlab

2.6.2.1. Simulink

Para empezar, se abre el espacio de trabajo de Simulink seleccionando el ícono Simulink Library, desde la barra de inicio y posteriormente haciendo clic en New Model.

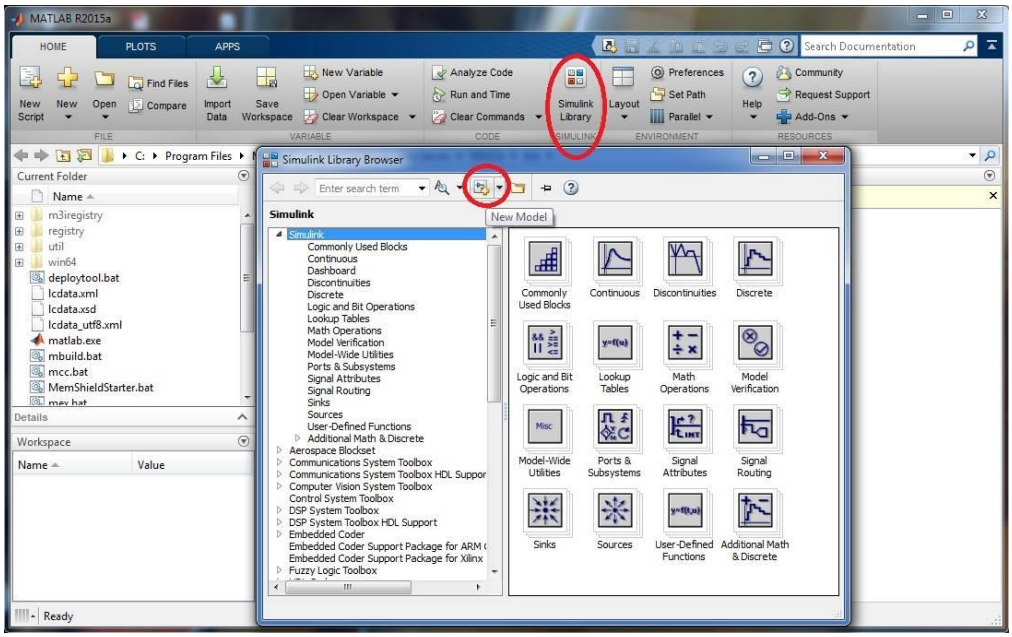


Figura 14. Nuevo Proyecto en Simulink

Fuente: Autores

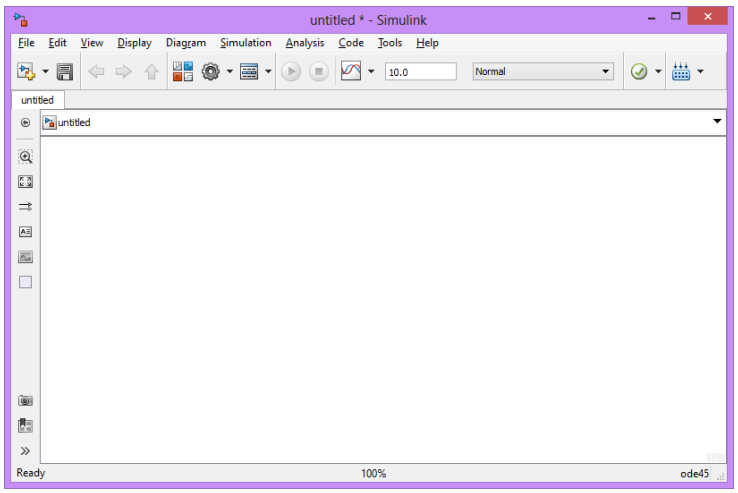


Figura 15. Entorno de Trabajo de Simulink

Fuente: Autores

La primera acción a realizar con esta herramienta es establecer el subsistema LMS con la configuración indicada en la primera etapa del desarrollo de este trabajo, para que posteriormente sea transformado en un bloque IP para utilizarlo en Vivado.

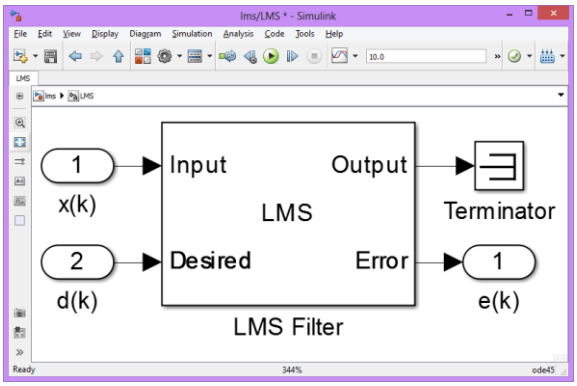


Figura 16. Estructura del Subsistema LMS

Fuente: Autores

Luego hay que diseñar el diagrama de bloques que representa la orientación general del proyecto y que permite realizar la simulación del sistema que será implementado.

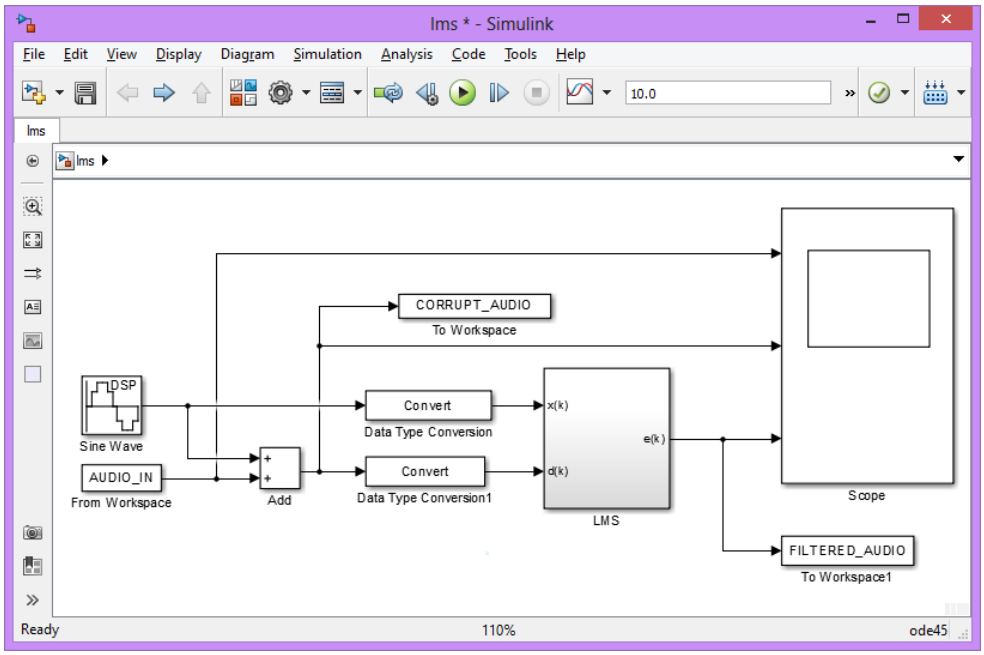


Figura 17. Diagrama en Simulink del Proyecto

Fuente: Autores

En el modelo se aprecia dos fuentes, un bloque de Onda Seno que hace las veces de la entrada de ruido y un bloque From Workspace que simula la entrada de la señal de audio, y su función es importar las muestras desde el espacio de trabajo de Matlab.

El ruido, en este caso tonal, se agrega a las muestras de audio para crear una señal de audio contaminada.

Para generar el código HDL del modelo LMS de Simulink usando HDL Coder, es necesario que las entradas al sistema sean fijadas en formato numérico. Dos bloques de conversión del tipo de datos, son usados para convertir la señal de audio contaminada y la señal de ruido tonal a formato de puntos fijos. Las señales de punto fijo son las entradas al subsistema LMS.

En el bloque LMS se tiene a la salida del subsistema, la señal de error, **e (k),** que es la entrada a un osciloscopio con la señal de audio contaminada y las entradas de ruido tonal, para una inspección visual de las señales. Dos bloques To Workspace son también presentados permitiendo a la señal LMS y a la señal de audio corrompida, salir al espacio de trabajo de Matlab para reproducir el audio.

Para profundizar en el bloque del subsistema LMS se hace doble clic en él. Se verá el sistema que muestra la Figura 16.

Esto caracteriza un solo Filtro LMS en un bloque.

Para el diseño en Vivado se requiere únicamente del Filtro por lo tanto se va a generar solo el código HDL del subsistema LMS.

2.6.2.2. HDL Workflow Advisor

Para la generación del Código HDL es necesario apoyarse en el Asistente de Flujo de trabajo. Para acceder a él se debe seleccionar el bloque del cual se pretende originar el código, en este caso el subsistema LMS, y hacer clic derecho sobre él. Luego se elige HDL Code > HDL Workflow Advisor.

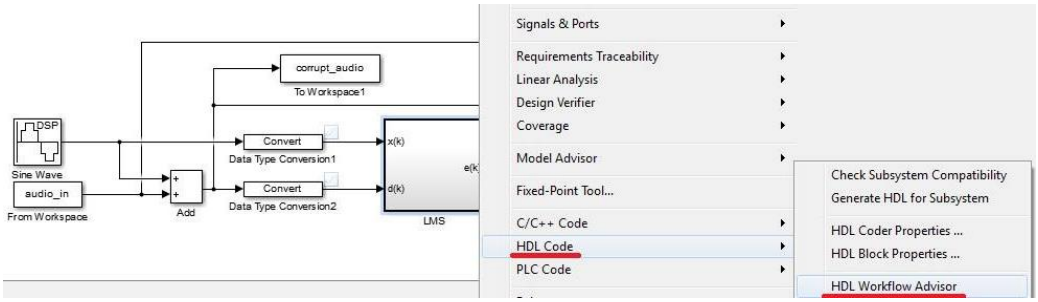


Figura 18. Acceso a HDL Workflow Advisor

Fuente: Autores

La ventana de HDL Workflow Advisor se abrirá, tal como se muestra en la Figura 19, y este servirá de guía a través de los pasos requeridos para generar el código.

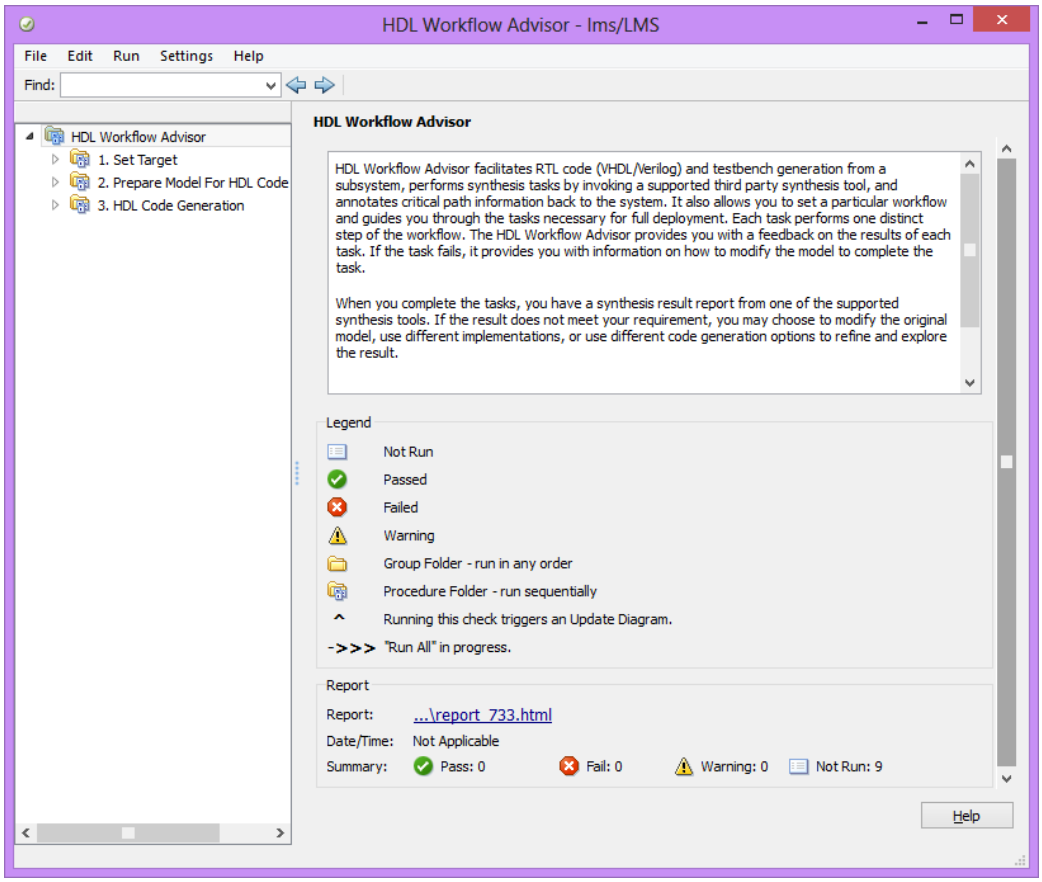


Figura 19. Ventana de HDL Workflow Advisor

Fuente: Autores

En el panel de la izquierda, se debe expandir Set Target y entonces seleccionar 1.1. Set Target Device and Synthesis Tool. Aquí es donde se especifica el formato de salida de RTL y la plataforma de destino.

En el panel de Parámetros de entrada, se elige IP Core Generation como el flujo de trabajo de destino, y Generic Xilinx Platform como la plataforma de destino, como lo indica la Figura 20.

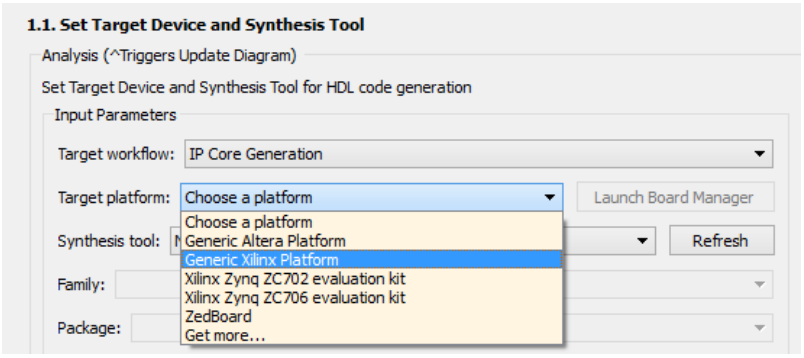


Figura 20. Selección de la Plataforma Xilinx

Fuente: Autores

En esta etapa, ahora estarán disponibles las opciones de especificación de partes adicionales. Direccionar la Zybo confirmando la parte querida mediante la inspección del chip Zynq de la tarjeta. Ingresar los detalles en el HDL Coder como se muestra en la Figura 21, y luego, presionar Run This Task para aplicar la configuración.

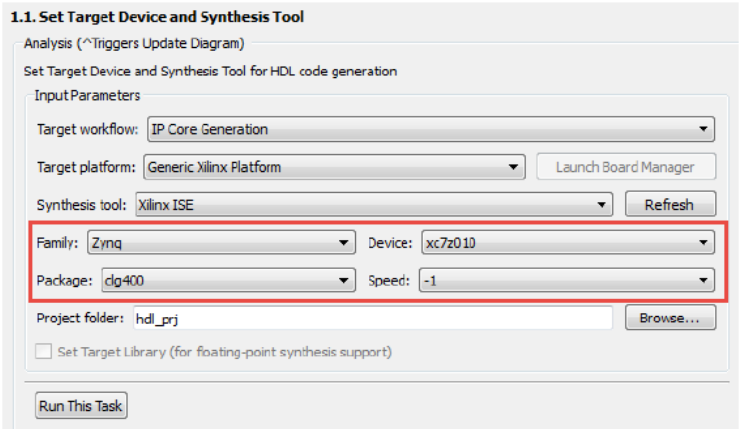


Figura 21. Ingreso de Parámetros ZYBO en HDL Workflow Advisor

Fuente: Autores

Ahora hay que dirigirse a la sección 1.2. Set Target Interface desde el panel de la izquierda. Aquí se define la interfaz de destino para la generación del código HDL.

En el panel de parámetros de ingreso, se escoge Compressing – blocking para Processor/FPGA synchronization. Esto va a inferir automáticamente una interfaz AXI4-Lite para todos los puertos en el diseño, y especifica una dirección de memoria para cada una, como se muestra en la Figura 22.

Para aplicar la configuración se oprime Run This Task.

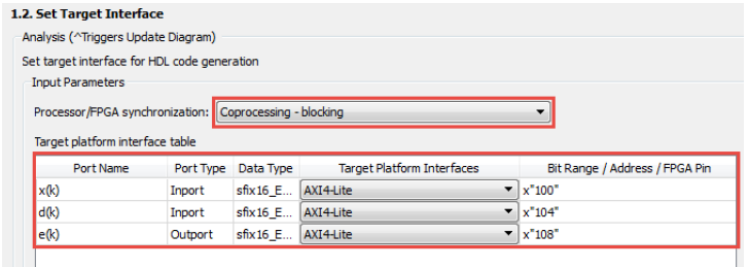


Figura 22. Conjunto de Interfaces de destino

Fuente: Autores

En el panel izquierdo después de expandir Prepare Model for HDL Code Generation, y se selecciona 2.1. Check Global Settings. Se debe hacer clic en Run This Task para comprobar las configuraciones a nivel de modelo, las mismas que se someten a verificación para determinar si el modelo está listo para la generación de código HDL.

Los siguientes pasos son de control y se pueden realizar en conjunto. Se hace clic derecho en Check Sample Times en el panel de la izquierda, y se escoge Run Selected Task como se muestra en la Figura 23, esto va a realizar las comprobaciones una tras otra para prevenir al programador de la ejecución individual de cada una.

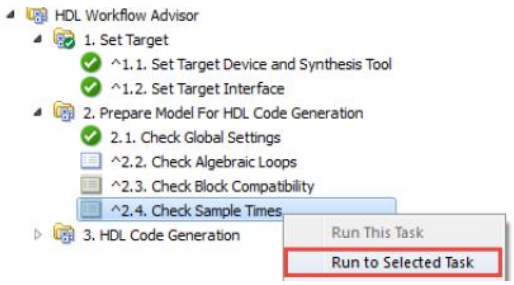


Figura 23. Ejecución de las Tareas de Verificación en HDL Workflow Advisor

Fuente: Autores

Los pasos finales involucran la especificación de las configuraciones básicas del código RTL, como por ejemplo qué lenguaje usará (VHDL/Verilog), y qué reportes de generación de código debe emitir.

Se expande HDL Code Generation en el panel de la izquierda, y además se amplía 3.1. Set Code Generation Options. Se hace clic en Set Basic Options y se selecciona VHDL como el lenguaje en el panel Target. También se puede elegir cualquier reporte de la generación del código que se desee.

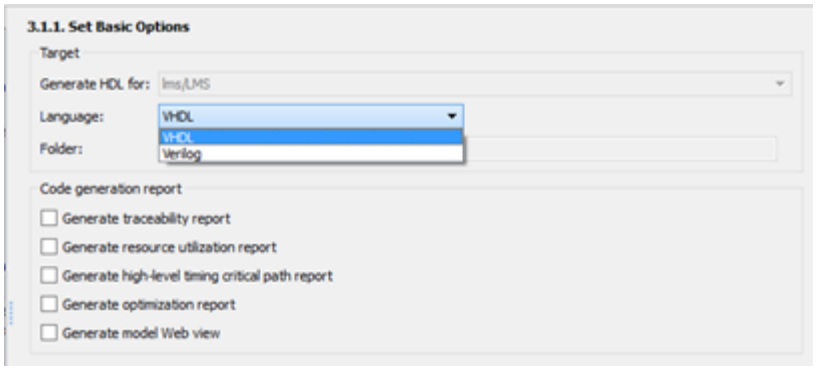


Figura 24. Selección del Lenguaje de trabajo

Fuente: Autores

En el panel izquierdo se selecciona Set Advanced Options, se da clic derecho y se elige Run to Selected Task para aplicar la configuración.

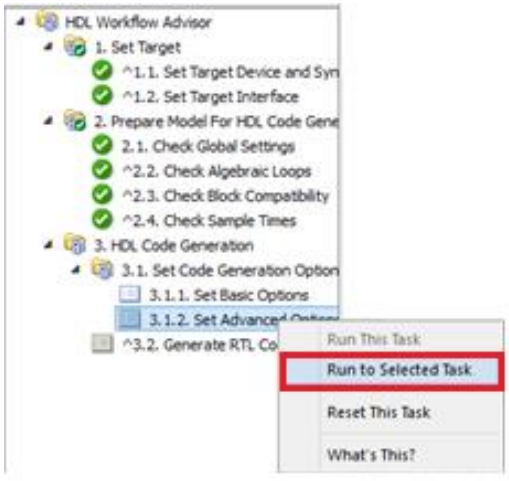


Figura 25. Ejecución de las Tareas de Configuración para la Generación de Código

Fuente: Autores

Por último, se escoge Generate RTL Code and IP Core del panel de la izquierda; este es el paso que finalmente generará el código HDL para el núcleo IP LMS. Se asigna el nombre del núcleo IP, en este caso se lo ha nombrado como lms_pcore, y se hace clic en Run This Task.

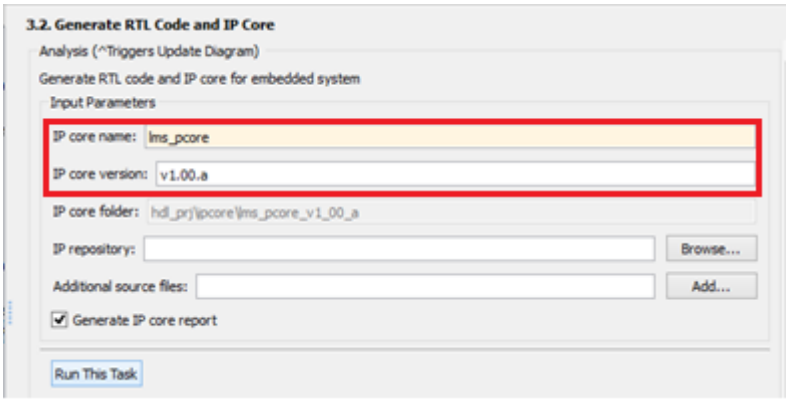


Figura 26. Ingreso del nombre y la Versión del bloque que se generará

Fuente: Autores

Una vez que el HDL Coder haya finalizado la generación del código HDL, la ventana de reporte de Generación de Código se abrirá. Este provee un resumen de los resultados del HDL Coder y también proporciona más información de la interfaz de destino y sincronización.

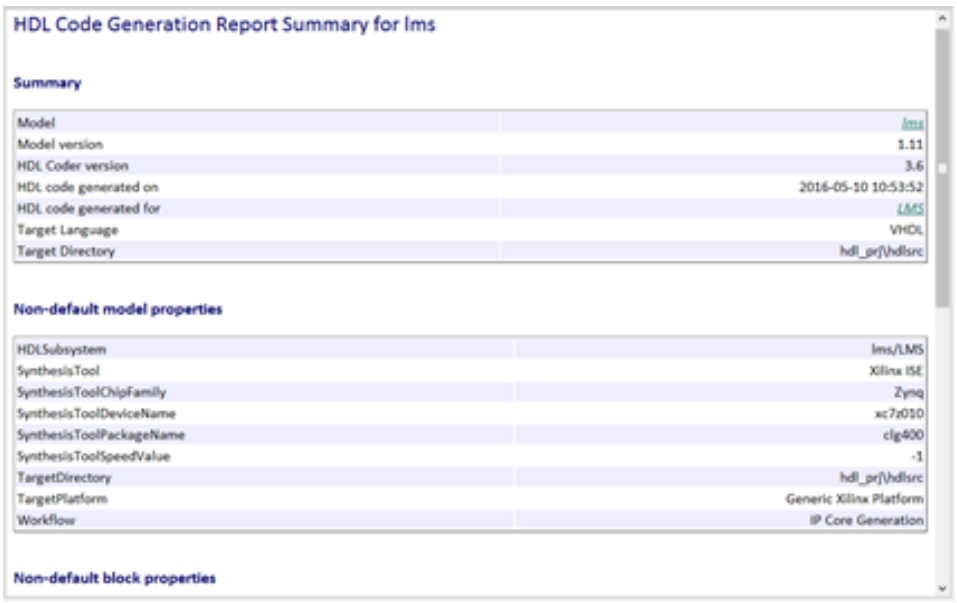
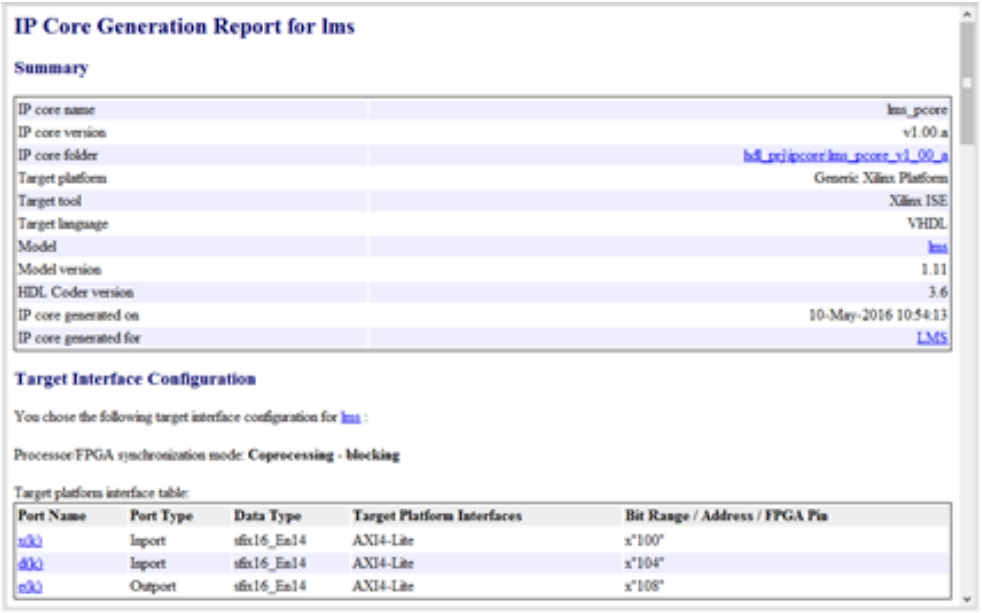


Figura 27. Resumen de la Generación del Código

Fuente: Autores



IP Core Generation Report for lms

Summary

IP core name	lms_pcore
IP core version	v1.00.a
IP core folder	hdl_prj\ipcore\lms_pcore_v1_00_a
Target platform	Generic Xilinx Platform
Target tool	Xilinx ISE
Target language	VHDL
Model	lms
Model version	1.11
HDL Coder version	3.6
IP core generated on	10-May-2016 10:54:13
IP core generated for	LMS

Target Interface Configuration

You chose the following target interface configuration for lms :

Processor/FPGA synchronization mode: **Coprocessing - blocking**

Target platform interface table:

Port Name	Port Type	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
clk()	Inport	sfix16_En14	AXI4-Lite	x"100"
dclk()	Inport	sfix16_En14	AXI4-Lite	x"104"
rclk()	Outport	sfix16_En14	AXI4-Lite	x"108"

Figura 28. Reporte del núcleo IP LMS

Fuente: Autores

Para tener un núcleo IP LMS funcional, el código generado en Matlab requiere ser empaquetado, y esto se lo hace con el IP Packager de Vivado Suite Design. Este proceso garantiza que se pueda usar el bloque LMS en los diseños del Integrador IP.

2.6.3. Vivado Suit Design

En esta parte se esboza la arquitectura con todos los periféricos que serán usados en el tablero Zybo, ya sean externos al Zynq (como el bloque de audio, botones, switches, etc.) o internos (el bloque LMS creado en Matlab o el bloque NCO[3] creado a través de Vivado HLS).

El primer paso es crear un nuevo proyecto en Vivado con la opción de Crear Subdirectorio activada y definiendo RTL Project como el tipo de proyecto; VHDL como lenguaje de destino y la Zybo como tablero de desarrollo (Para un proceso más detallado acerca de la creación de un nuevo proyecto en Vivado Suite Design dirigirse al Apéndice A).

[3] Bloque que produce una onda seno, utilizado en las pruebas con ruido generado internamente.

Luego de crear el proyecto se habilita el núcleo IP LMS (ver Apéndice B) y se lo agrega al catálogo IP (Ver Apéndice C) junto con los núcleos que no consten en la librería de Vivado para poder usarlos dentro del diseño del proyecto.

El diseño se lo realiza mediante el sistema de bloques, por lo tanto, desde la sección IP Integrator se toma la opción Create Block Design.

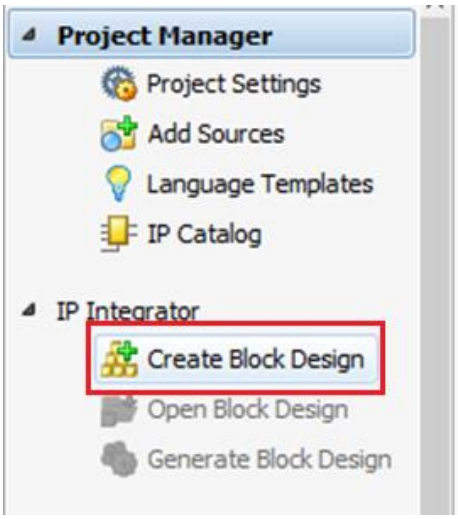


Figura 29. Botón para crear un nuevo diagrama de bloques

Fuente: Autores

El siguiente paso es indicar el nombre del proyecto en el espacio indicado y empezar a añadir los bloques al bosquejo.

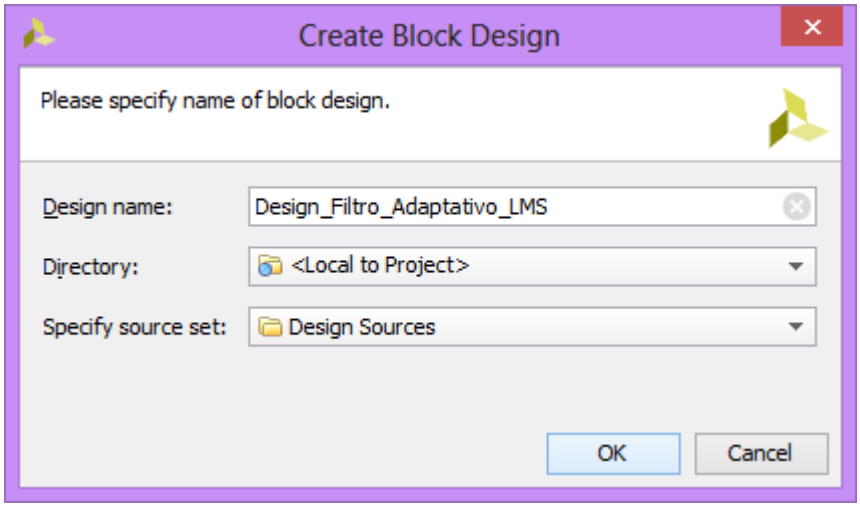


Figura 30. Asignación de nombre al diseño de bloques.

Fuente: Autores

El primer bloque que se agrega es el Sistema de Procesamiento Zynq. Se presiona el botón de agregar IP, entonces aparece el menú emergente del catálogo de IPs y se navega en él hasta encontrar el Zynq7 Processing System.

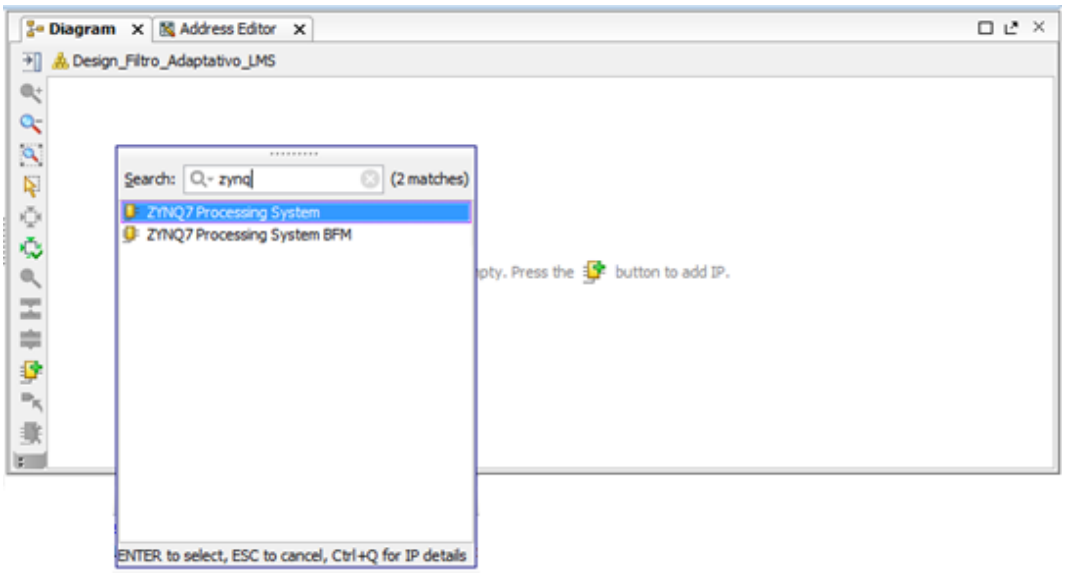


Figura 31. Búsqueda del Bloque Zynq en el catálogo IP

Fuente: Autores

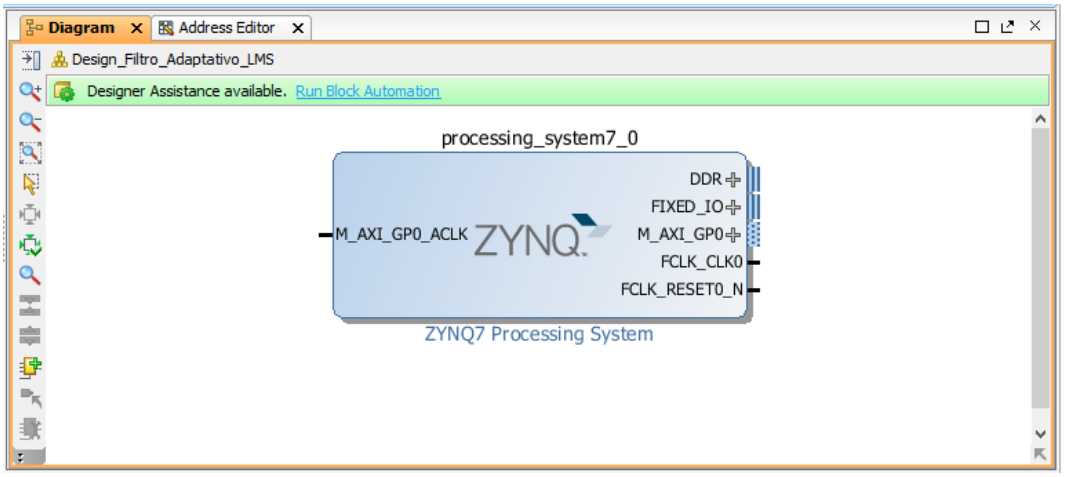


Figura 32. Adición del bloque Zynq

Fuente: Autores

En la parte superior de la ventana de trabajo se oprime Run Block Automation, lo que abrirá una ventana de diálogo en la que se debe activar la opción Apply Board

Preset y dar clic en aceptar, para de esta manera crear las conexiones externas para las interfaces DDR y FIXED_IO.

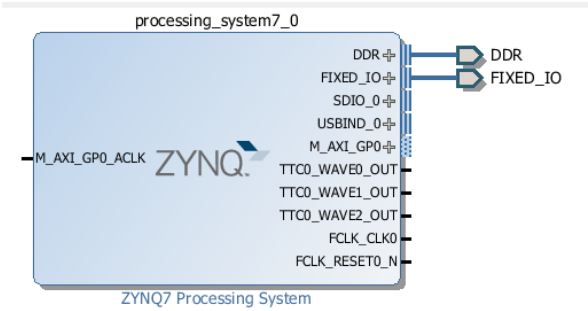


Figura 33. Bloque de procesamiento Zynq

Fuente: Autores

Ya que la plataforma Zybo fue especificada como el tablero de desarrollo al momento de crear el proyecto, Vivado configurará el bloque Zynq de acuerdo a los requerimientos de la tarjeta.

Todos los pasos realizados hasta aquí han servido para establecer la PS de la Zynq y también configurarla. En los siguientes pasos se suman los bloques que se ubicaran en la PL para agregar funcionalidad al sistema.

Los siguientes bloques que se agregan son el núcleo LMS, y para el caso de las pruebas con ruido interno, el bloque NCO que contiene la onda seno.

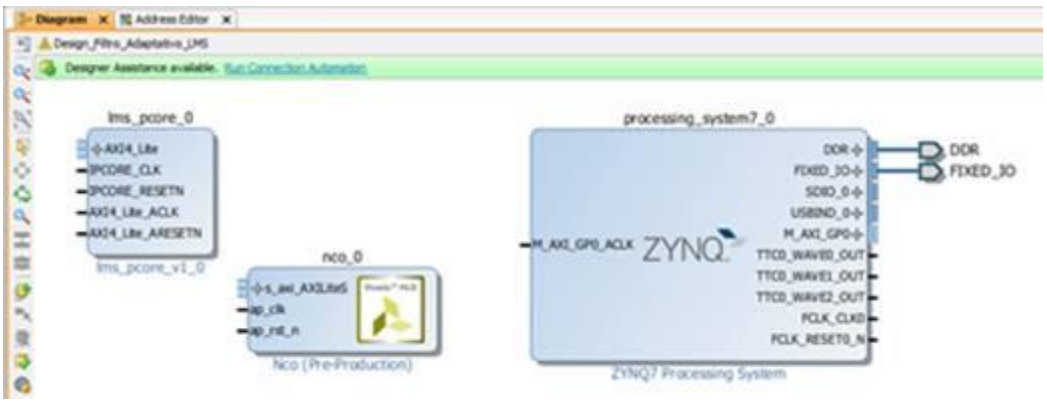


Figura 34. Adición de los núcleos LMS y NCO

Fuente: Autores

En la parte superior del espacio de trabajo aparecerá la opción Run Connection Automation, que debe ser seleccionada para realizar la conexión automática entre el Zynq y los dos bloques agregados, mediante un bloque de Interconexión AXI.

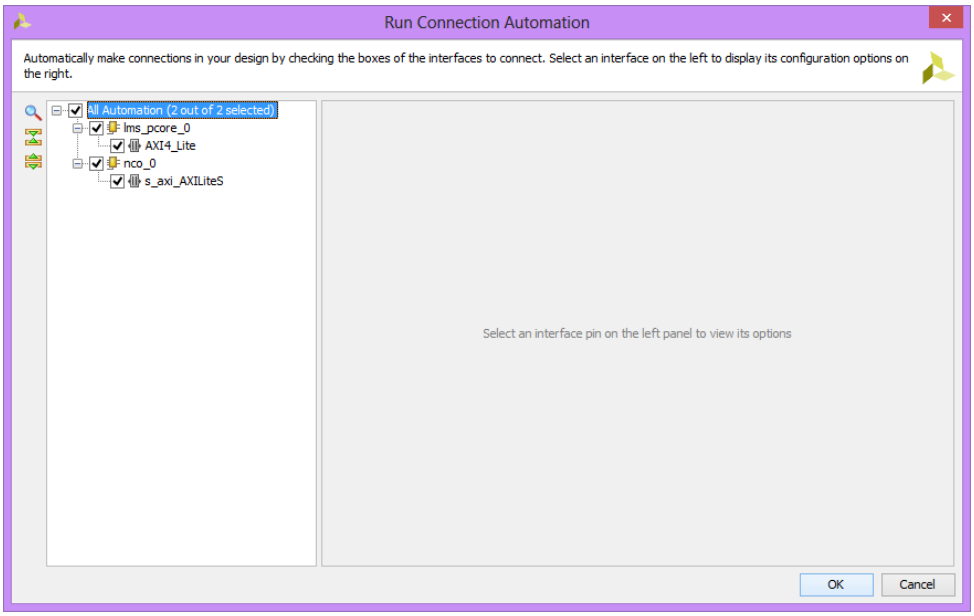


Figura 35. Conexión Automática entre el Zynq y los bloques LMS y NCO

Fuente: Autores

En la figura 36 se puede observar que el bloque LMS tiene dos puertos de ingreso sin conexión, que son CLK y RESETN, y deben ser conectados en orden para que el bloque sea funcional. La conexión para el CLK se realiza con el puerto Axi_Lite_ACLK y el RESET con la interfaz AXI_Lite_ARESETN.

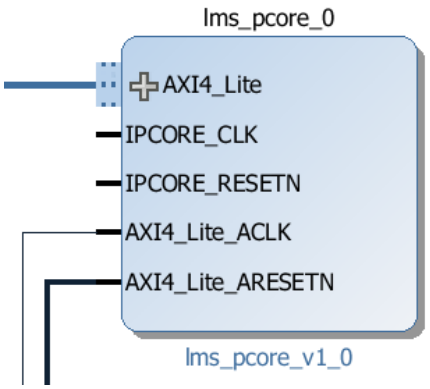


Figura 36. Bloque LMS con puertos CLK y RESETN desconectados

Fuente: Autores

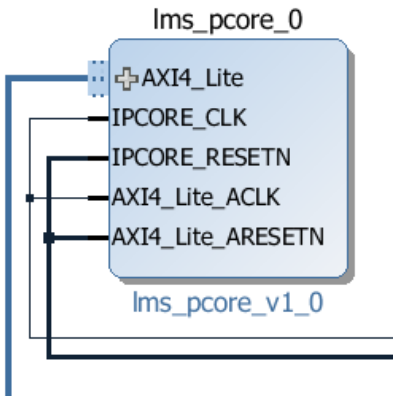


Figura 37. Conexión de los puertos CLK y RESETN en el bloque LMS

Fuente: Autores

Ahora se agrega una IP controladora de audio al diseño y se configura el bloque Zynq para usar el codificador de audio disponible en el tablero de desarrollo Zybo.

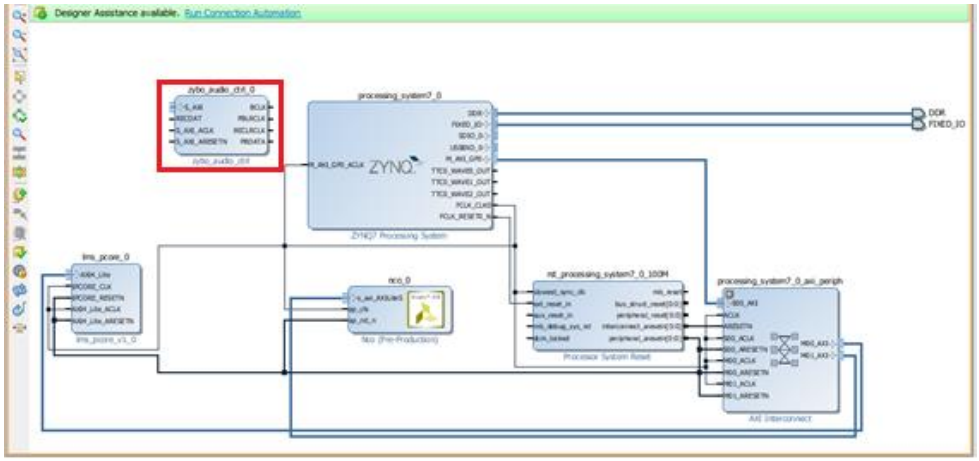


Figura 38. Adición del Bloque de Audio

Fuente: Autores

La configuración que se hace en el Zynq incluye la adición de un segundo reloj a la estructura PL que será conectado al pin MCLK en el codificador; y, la habilitación de la interfaz I2C para la comunicación de las señales de control entre la PS del Zynq y el codificador.

La primera conexión entre los bloques Zynq y el de audio se la establece haciendo clic en Run Connection Automation.

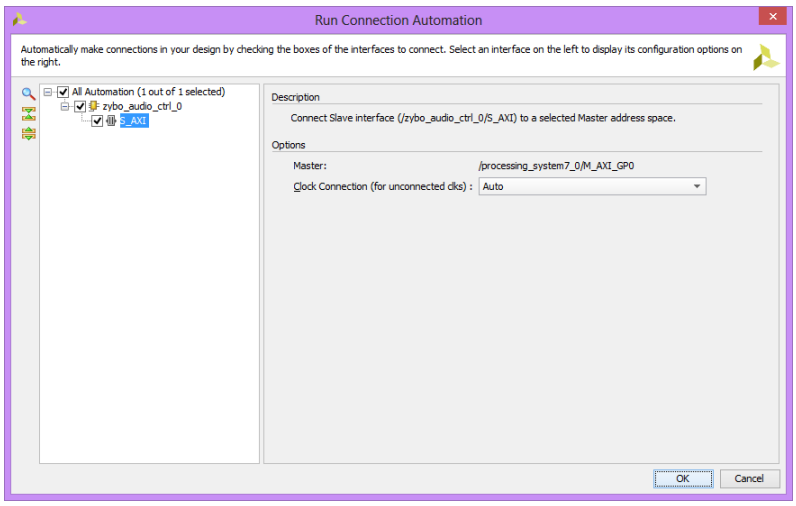


Figura 39. Conexión Automática entre el Zynq y el bloque de audio

Fuente: Autores

Cinco puertos del bloque de audio quedan sin conexión por lo que se les debe establecer una conexión externa hacia los pines físicos del controlador de audio de la tarjeta.

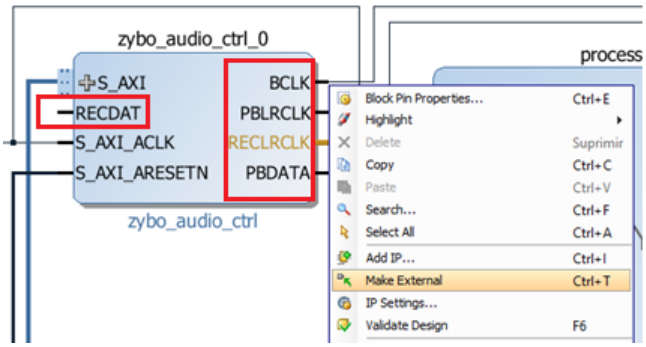


Figura 40. Conexión externa de los puertos libres en el bloque de audio

Fuente: Autores

El siguiente paso es hacer las modificaciones antes mencionadas en el Zynq. Se debe abrir la ventana Re-Customize para el Zynq, dando doble clic en el bloque; entonces se puede ver un esquema como el que muestra la figura 41.

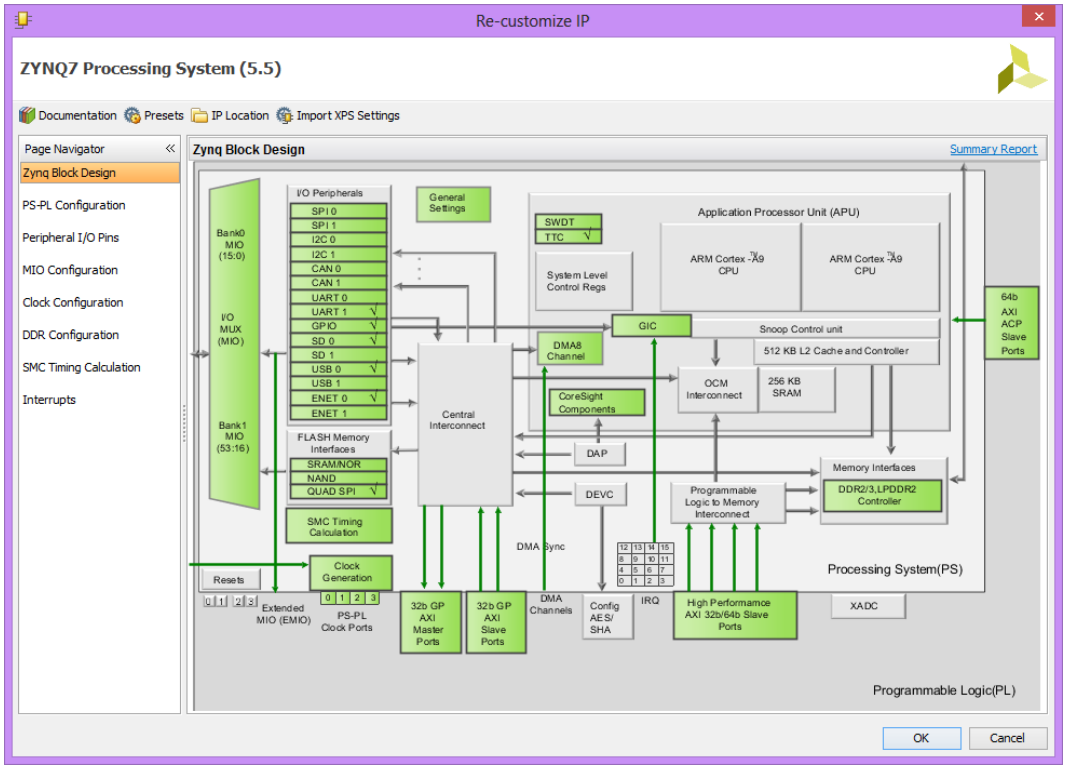


Figura 41. Ventana de Personalización de los requerimientos en la Zynq

Fuente: Autores

Se debe seleccionar la Configuración de Reloj en el Panel de Navegación. Se expande PL Fabric clocks y se habilita FCLK_CLK1. Es necesario cambiar la Frecuencia Requerida del FCLK_CLK1 a 12.288 MHz, como se muestra en la figura 42.

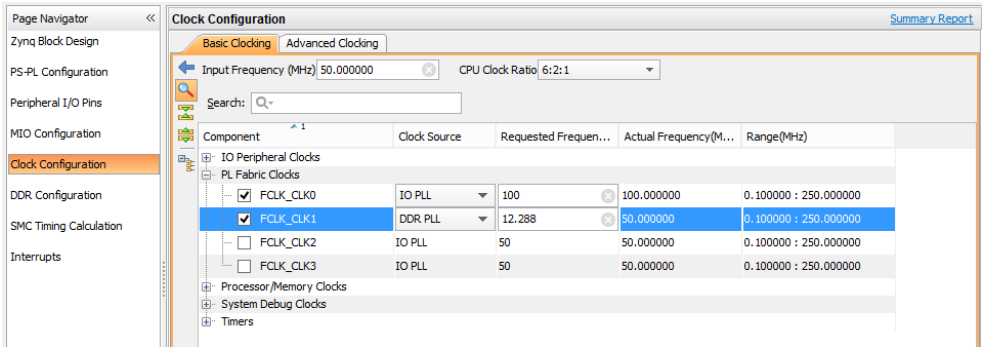


Figura 42. Habilitación del Puerto FCLK_CLK1

Fuente: Autores

Ahora se debe habilitar una de las interfaces de comunicación I2C de la PS, para permitir la comunicación entre el Zynq y el codificador de audio. Para esto se elige la Configuración MIO en el Panel de Navegación, se expande la opción de I/O Peripherals y se habilita el periférico I2C0, entonces EMIO será seleccionado automáticamente como IO.

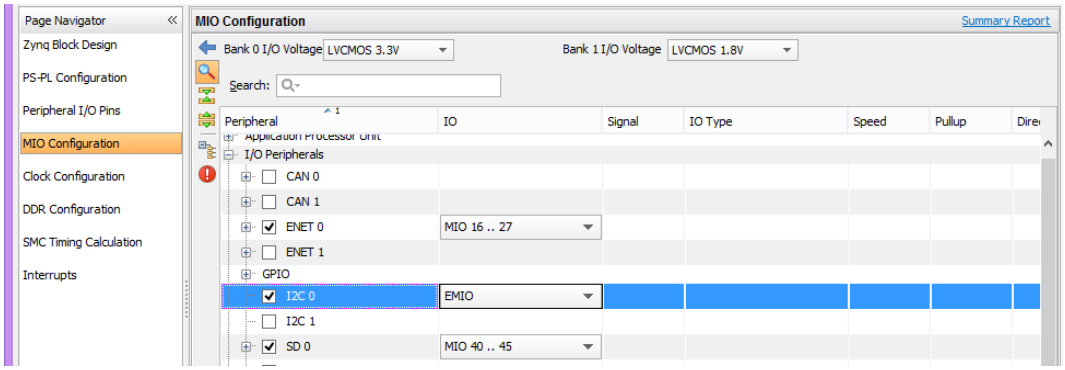


Figura 43. Habilitación del Periférico I2C0

Fuente: Autores

Se aplican los cambios al cerrar la venta, y se observa que dos interfaces han sido añadidas al bloque Zynq, IIC_0 y FCLK_CLK1. Estas interfaces son las que

conducirán las señales al codificador de audio, y ya que este se encuentra situado en el tablero, es decir es externo al Zynq, deben ser marcadas como externas.

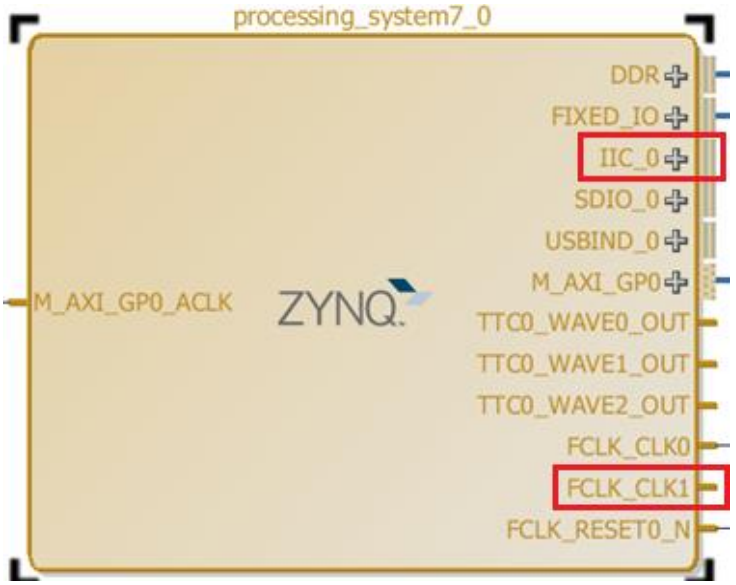


Figura 44. Nuevos puertos en el bloque Zynq

Fuente: Autores

Para finalizar el diseño, se han añadido dos bloques GPIO, uno de canal simple y otro de canal dual. El primero, tendrá un ancho de 1 bit para permitir la comunicación con el codificador de audio; y, el segundo, con un ancho de 32 bits para conectar los pulsadores y switches que serán utilizados como entradas.[4]

Primero se añade el GPIO de canal simple, y se da clic en Run Connection Automation para conectar la interfaz axi_gpio_0/S_AXI, de esta manera se conecta el controlador GPIO con la PS vía Interconexión AXI.

Se da doble clic en el bloque GPIO para personalizar la configuración del bloque, y en la ventana que se abre se selecciona la pestaña de configuración IP. Se especifica el ancho del GPIO y luego se establece como externa la interfaz de salida del GPIO.

[4] El GPIO de canal Dual es necesario para el diseño que se usará en las pruebas con generación de ruido interno.

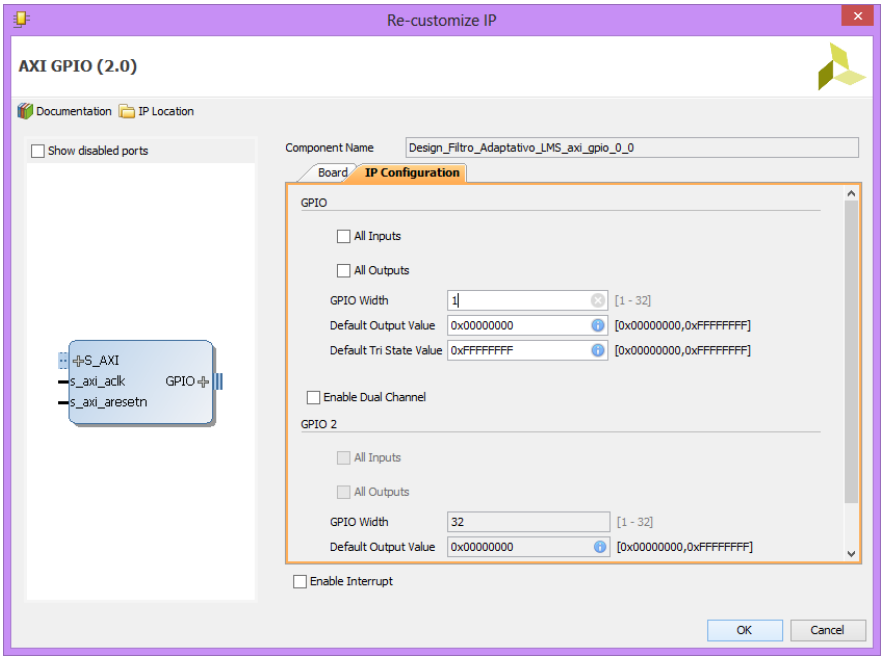


Figura 45. Personalización del bloque GPIO

Fuente: Autores

Para el segundo bloque GPIO se sigue el proceso del bloque anterior, pero en la configuración personalizada se activa el canal dual. Al guardar los cambios, se observa que el bloque ahora tiene dos interfaces de salida, las que se deben conectar a los pulsadores y a los switches respectivamente. Para ello se debe correr la opción Run Connection Automation; para la salida /axi_gpio_1/GPIO se especifica btns_4bits y para la salida /axi_gpio_1/GPIO2 se selecciona sws_4bits, como las interfaces en el tablero.

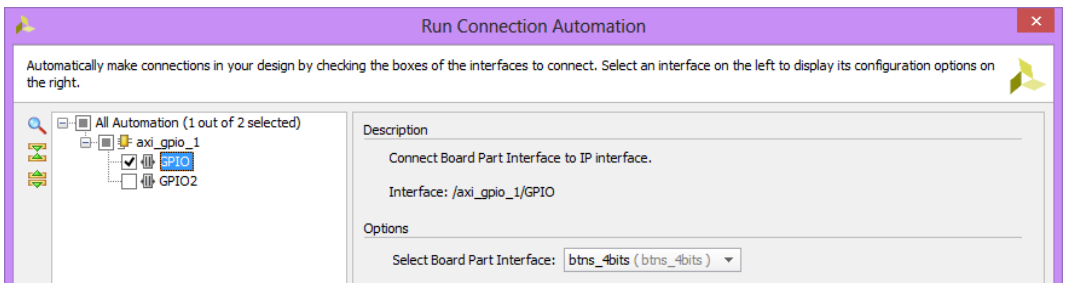


Figura 46. Asignación de Interfaz btns en el GPIO2

Fuente: Autores

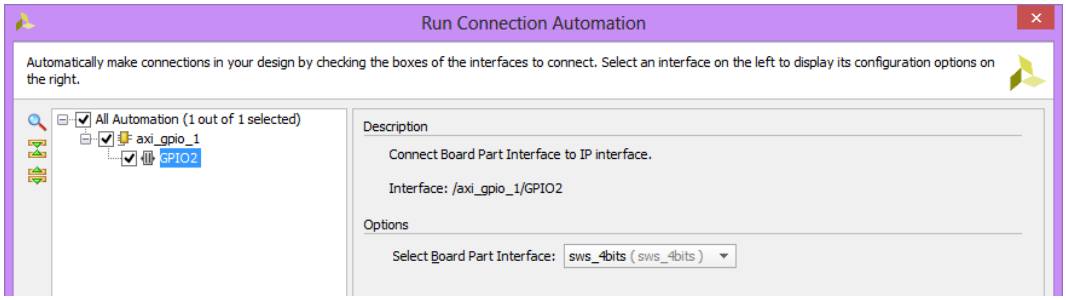


Figura 47. Asignación de Interfaz sws en el GPIO2

Fuente: Autores

Se guarda el diseño, y el resultado final se puede apreciar en las siguientes figuras:

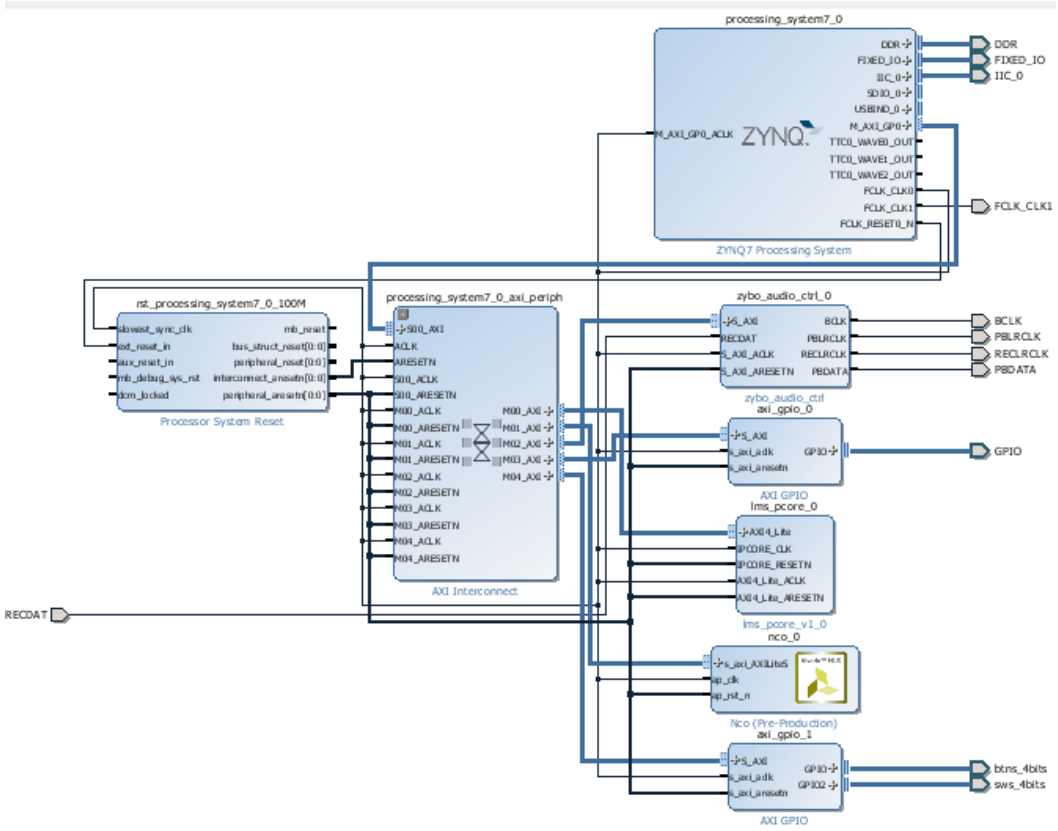


Figura 48. Arquitectura de hardware completa para pruebas con ruido interno

Fuente: Autores

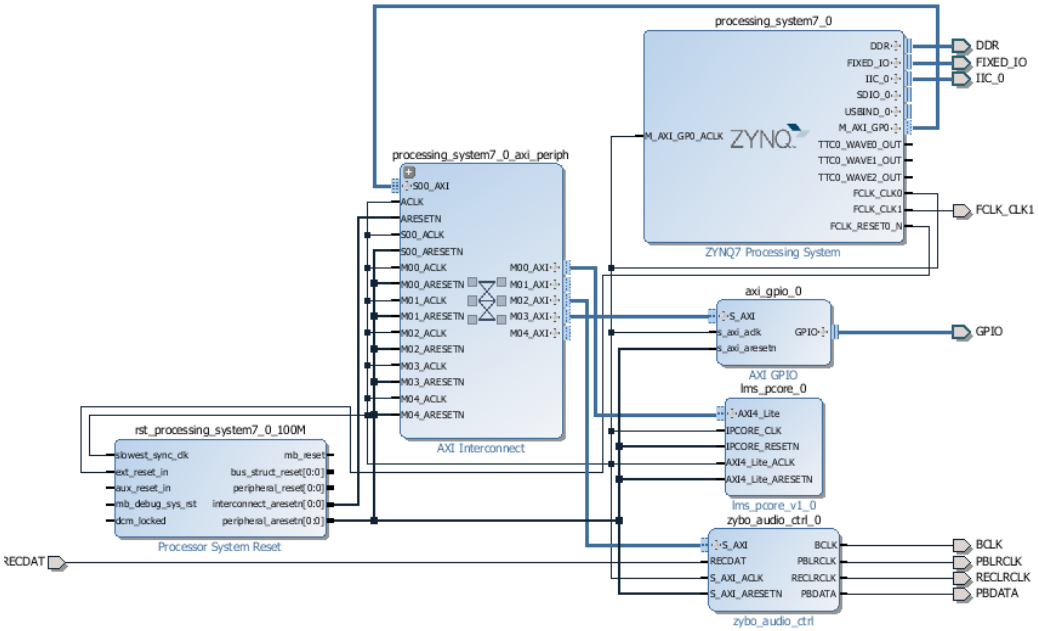


Figura 49. Arquitectura General del Proyecto

Fuente: Autores

Antes de correr la síntesis y la implementación del diseño, se deben generar los archivos RTL para el diseño de bloques. Para esto se elige la opción Create HDL Wrapper desde la pestaña Sources en el archivo principal.

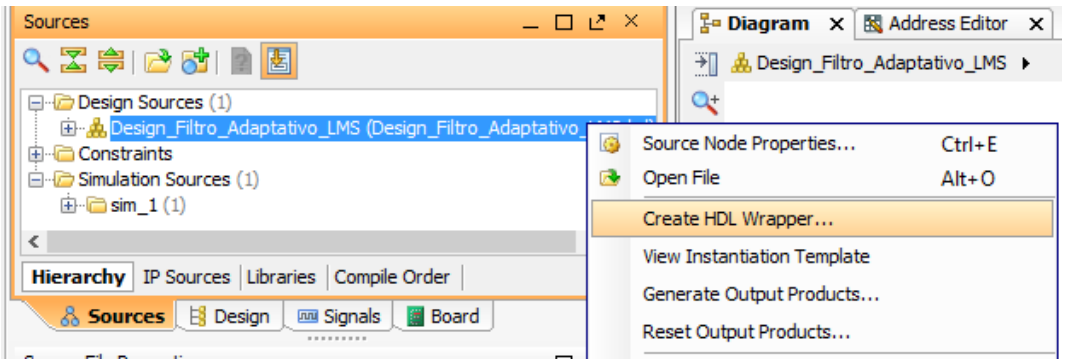


Figura 50. Generación del Archivo RTL

Fuente: Autores

También es necesario agregar el archivo que contiene las limitaciones de hardware del diseño, el mismo que tiene una extensión XDC y que ha sido programado en lenguaje VHDL con anterioridad.

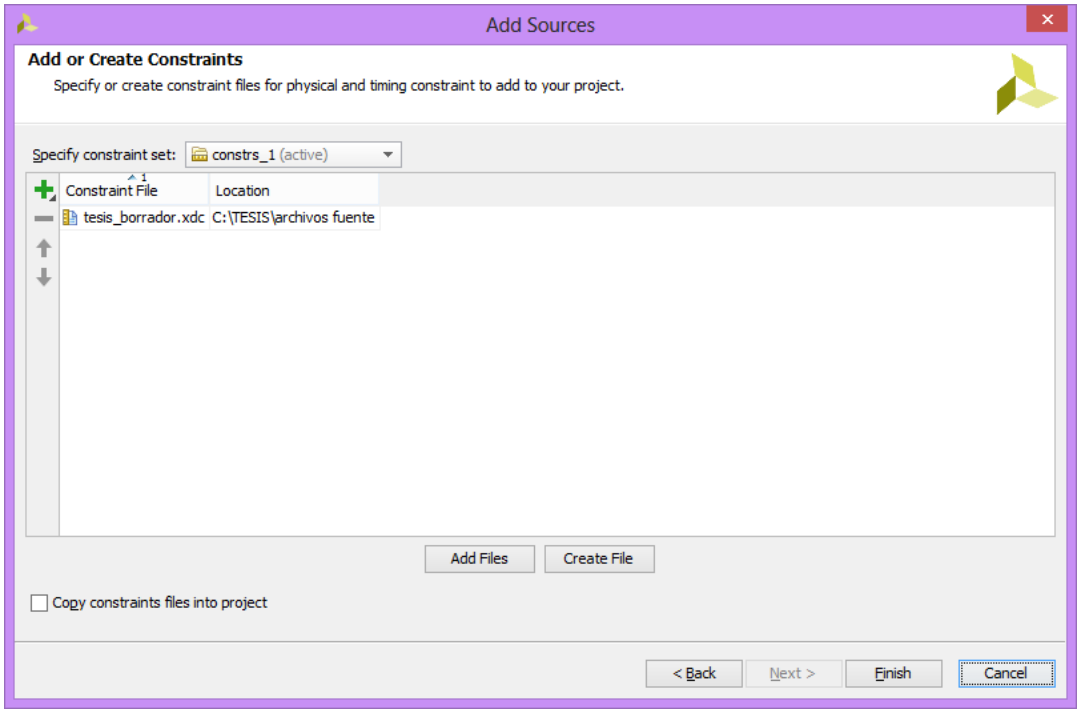


Figura 51. Adición de las Limitaciones de Hardware

Fuente: Autores

Previo a exportar el diseño, se genera el bitstream que programará la PL del Zynq. Esto se hace seleccionando la opción Generate Bitstream desde la sección Program and Debug.

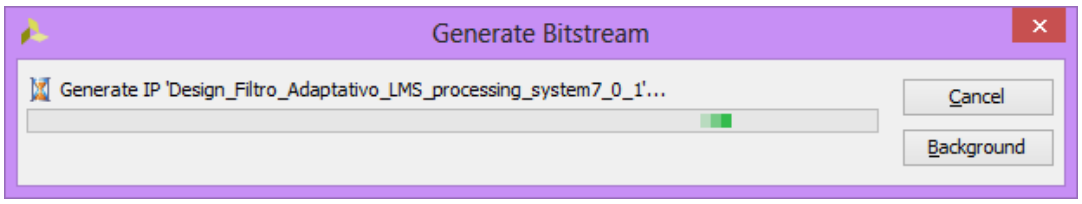


Figura 52. Generación del Bitstream

Fuente: Autores

El proceso puede tomar unos minutos dependiendo de la velocidad del computador en el que se esté procesando el diseño. Cuando la generación se haya completado, se debe elegir Open Implemented Design.

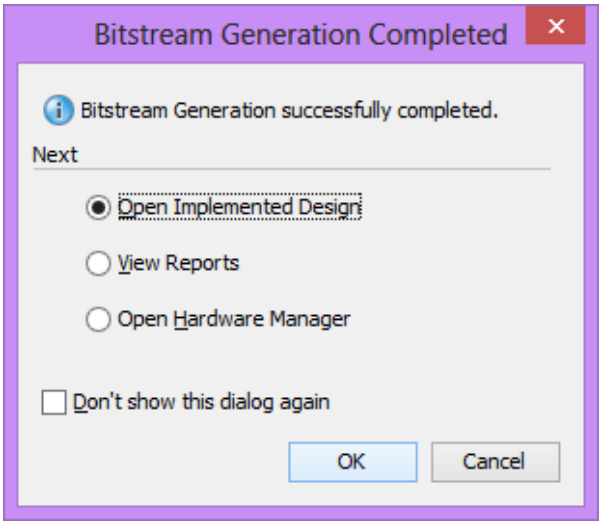


Figura 53. Mensaje de Reporte de Generación del Bitstream

Fuente: Autores

Finalmente, la exportación del hardware hacia el SDK se realiza escogiendo la opción de Exportar en la barra de menú. La opción Include Bitstream estará marcada.

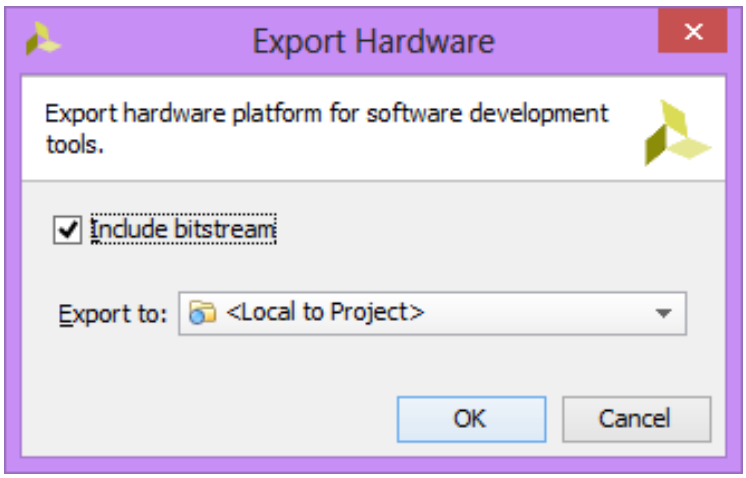


Figura 54. Ventana para exportar el proyecto

Fuente: Autores

Y ahora se enlaza al SDK en Vivado seleccionando Launch SDK desde la barra de menú.

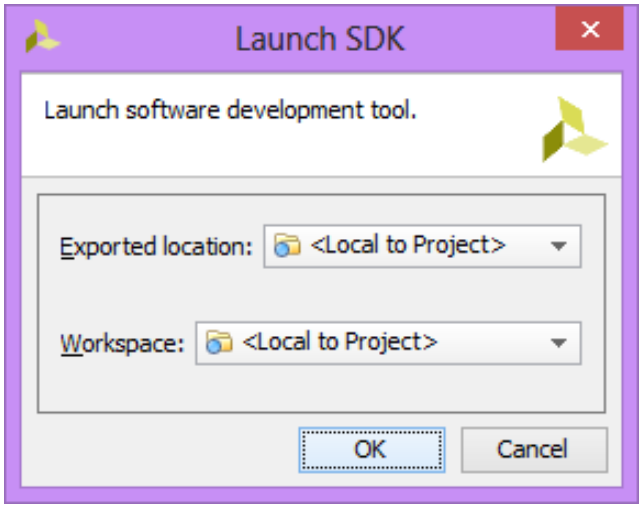


Figura 55. Ventana de Enlace con SDK

Fuente: Autores

2.6.4. SDK

Una vez que el SDK ha sido enlazado previamente desde Vivado, se puede iniciar con la creación de una nueva aplicación.

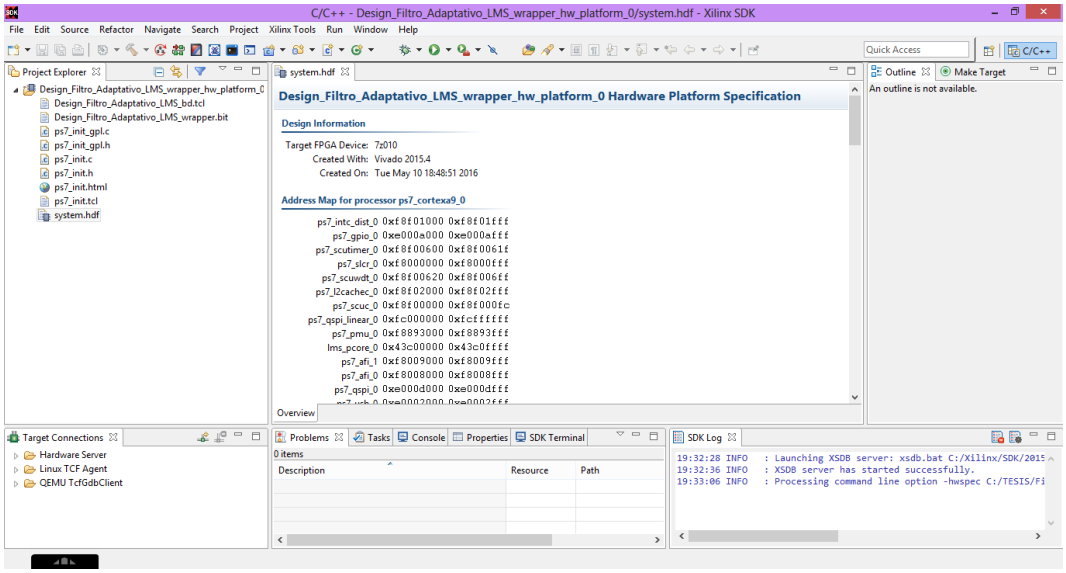


Figura 56. Entorno de trabajo en SDK

Fuente: Autores

Desde la barra de menú se selecciona la opción Application Project, y en la ventana que aparece se indican los parámetros del proyecto. Por defecto la opción Create a

New Board Support Package, estará seleccionada. En le ventana de plantillas se elige Empty Application.

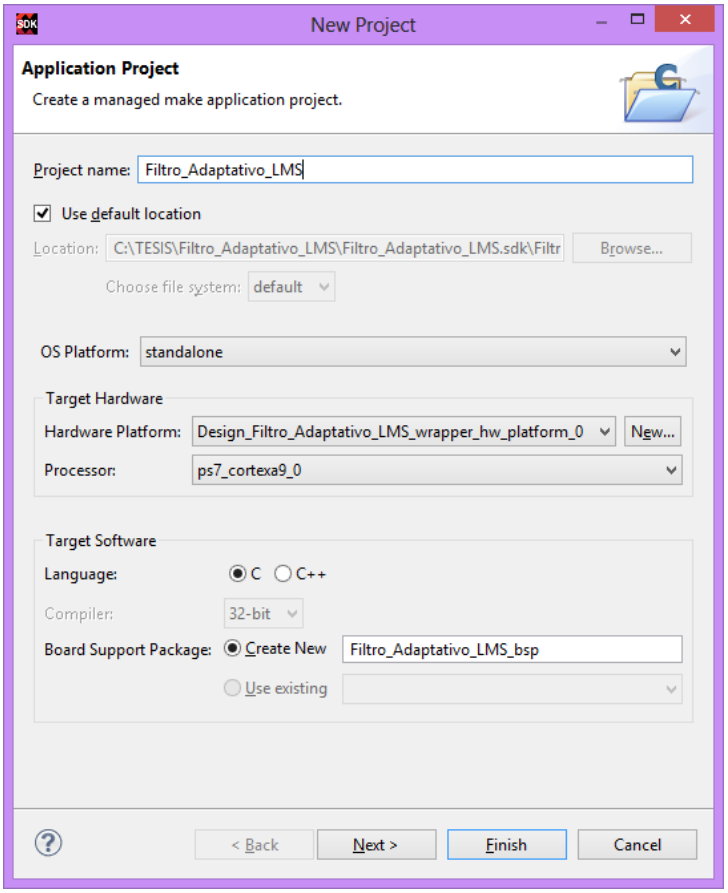


Figura 57. Ventana para asignación de parámetros del proyecto

Fuente: Autores

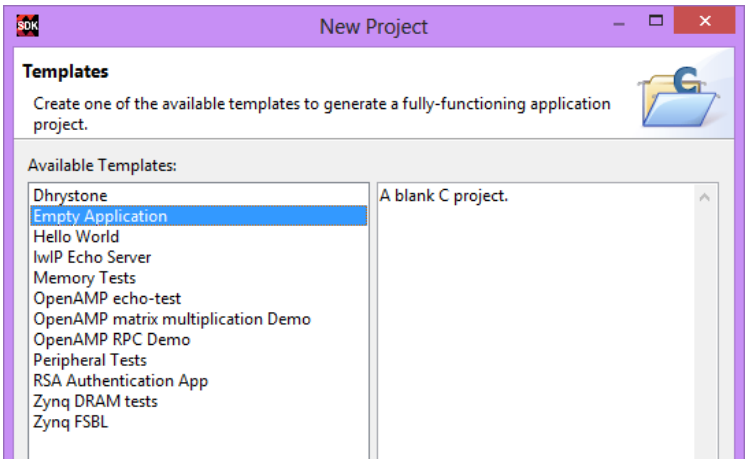


Figura 58. Ventana de plantillas

Fuente: Autores

Hay que recordar que cuando se crearon los periféricos IP personalizados, un conjunto de archivos controladores de software se generaron para cada uno. Ahora se debe llevar esos archivos a SDK, esto se hace mediante la adición de nuevos repositorios.

En la ventana de Repositories Preferences se selecciona New, tal como lo muestra la figura 59 y se añade el driver del bloque NCO, abriendo el directorio que lo contiene.

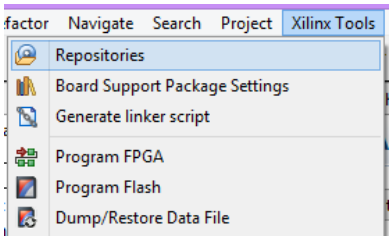


Figura 59. Acceso a los Repositorios

Fuente: Autores

SDK buscará automáticamente los repositorios y re-compilará el proyecto para incluir los archivos de controladores.

Los núcleos IP LMS y del codificador de audio, también poseen sus controladores de software, pero debido a su estructura de directorios, sus archivos son importados directamente hacia el espacio de trabajo, en lugar de usar un repositorio.

En el panel de Exploración del Proyecto, se expande el archivo con el nombre Filtro_Adaptativo_LMS, y se selecciona importar después de dar clic derecho en la carpeta con el nombre src.

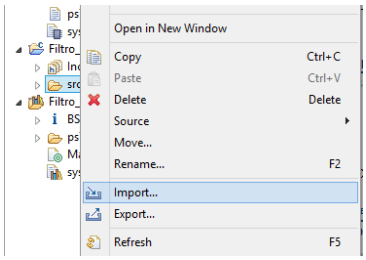


Figura 60. Acceso a la importación de Archivos

Fuente: Autores

En la ventana de importar, se expande General y se da doble clic en File System, se busca la carpeta que contiene los archivos relacionados a cada núcleo, y de ellos se eligen los que correspondan a la aplicación que se está creando y que tengan extensión c y h.

Estos archivos contienen las funciones que van a desempeñar los núcleos, esto significa muestreo, aplicación del algoritmo LMS en el filtrado además de la emisión del sonido filtrado, como también la interfaz que le permitirá elegir al usuario la emisión de audio que desea tener. (Programación descrita en los Anexos 4 y 5).

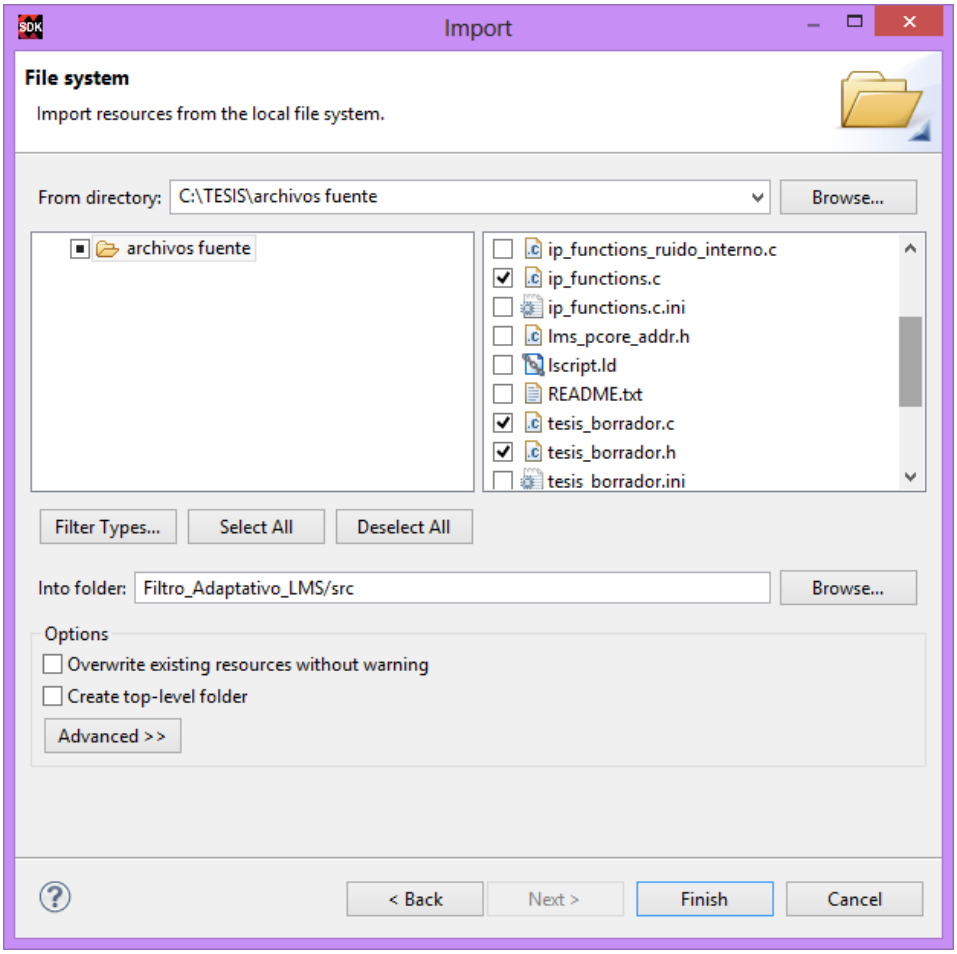


Figura 61. Ventana de selección de archivos fuente

Fuente: Autores

Ahora se programará la PL del Zynq con el bitstream generado en Vivado.[5] De la barra de menú se elige Program FPGA desde Xilinx Tools, entonces la ventana de programación se abrirá y se la debe configurar de acuerdo a la figura 63.

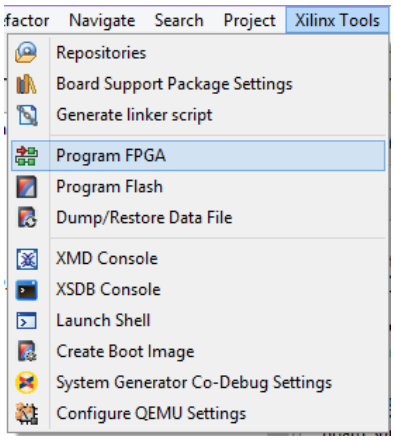


Figura 62. Acceso al programador de la FPGA

Fuente: Autores

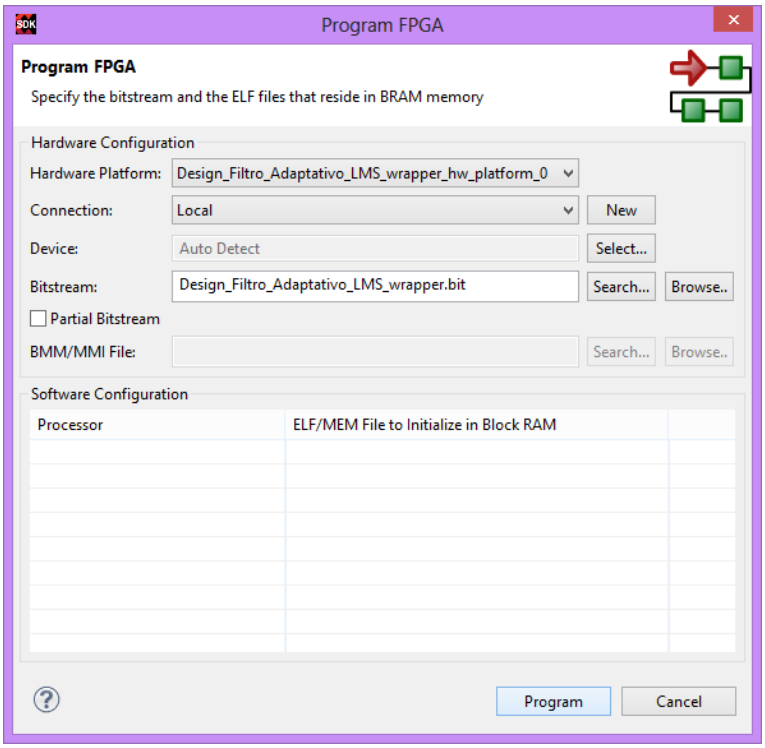


Figura 63. Ventana del Programador de la FPGA

Fuente: Autores

[5] Para este procedimiento, y los siguientes es necesario ya que se encuentre conectada y encendida la ZYBO. La tarjeta debe estar configurada para el uso de JTAG.

Al dar clic en el botón Programa, la PL del Zynq se configurará con el bitstream y el LED DONE se iluminará.

Figura 64. LED DONE Activado

Fuente: Autores

Para facilitar la interacción entre el usuario y la aplicación de este proyecto, se requiere el terminal PuTTY.

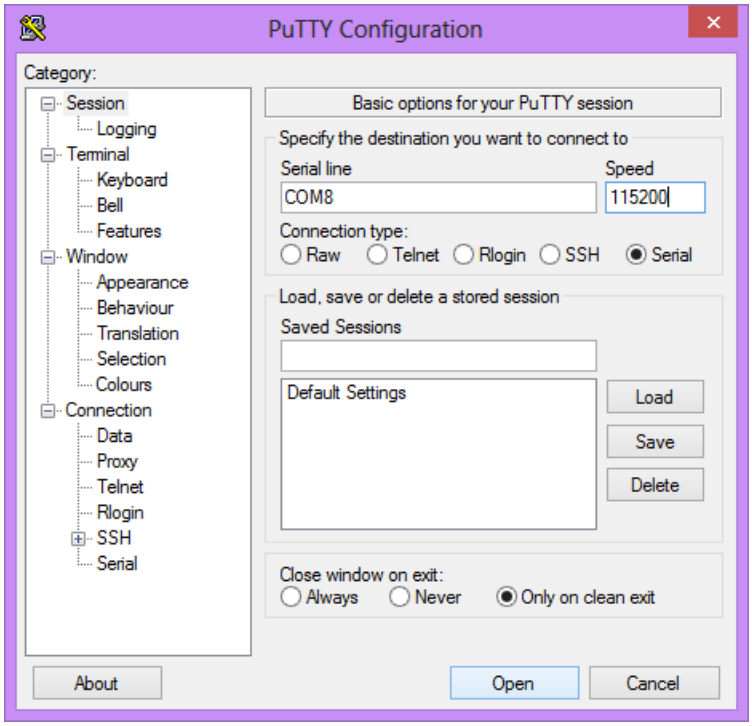


Figura 65. Configuración del Terminal PuTTY

Fuente: Autores

Finalmente, se debe correar la aplicación en la PS, esto se realiza dando clic derecho en el archivo con el nombre del proyecto y eligiendo Launch On Hardware (GDB) desde Run As.

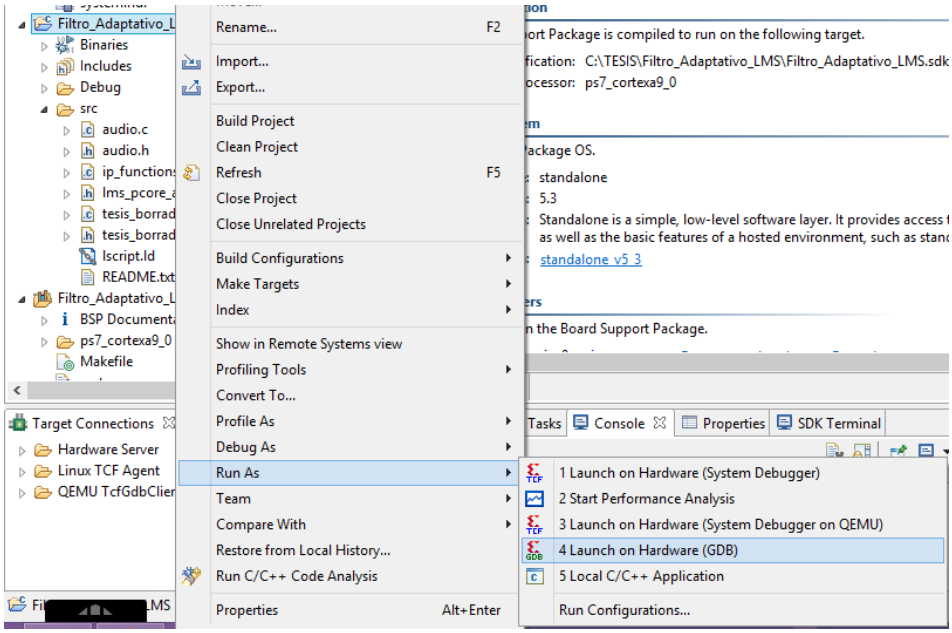


Figura 66. Activar la Ejecución del Proyecto

Fuente: Autores

Al iniciarse la ejecución del software, la pantalla del terminal PUTTY se verá como indica la figura 67.

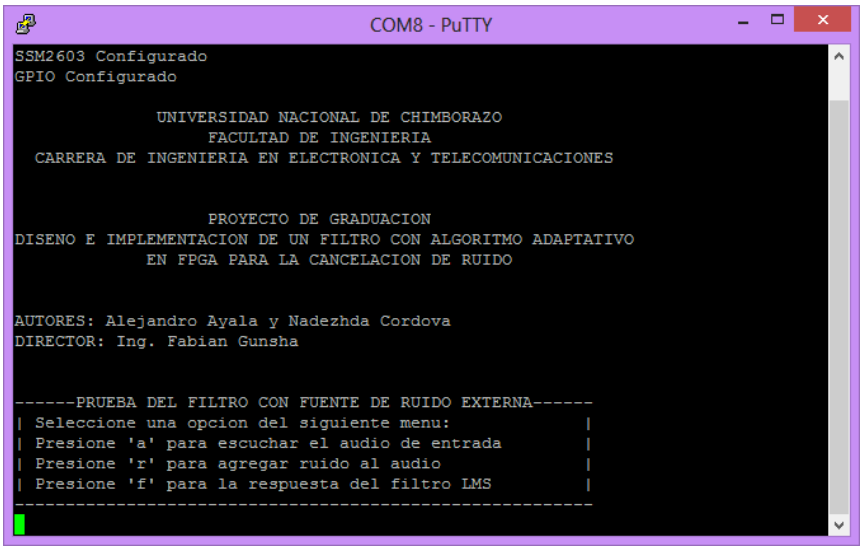


Figura 67. Terminal PuTTY

Fuente: Autores

2.7. Procesamiento y Análisis

Una vez que el filtro con algoritmo adaptativo LMS ha sido implementado, requiere ser sometido a prueba para evaluar su capacidad de aproximarse a la cancelación de ruido en señales de audio.

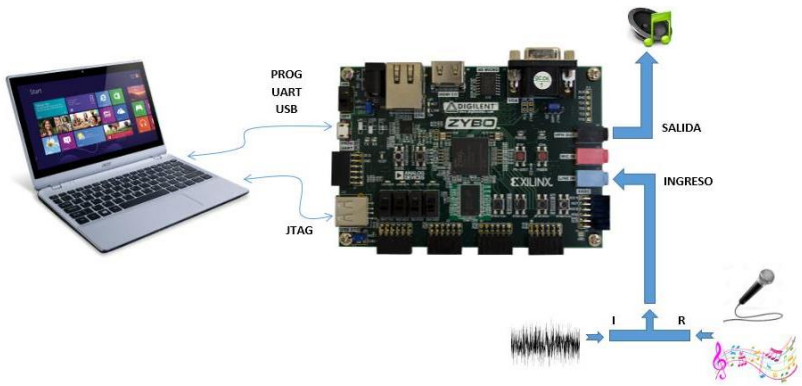


Figura 68. Esquema Final del Filtro Adaptativo LMS en la FPGA – ZYBO

Fuente: Los Autores

Las pruebas que se llevarán a cabo evidencia tres escenarios:

2.7.1. Escenario 1: Onda Generada - Ruido Interno

Para esta prueba se toma una señal de audio y se le agrega una señal de ruido generada en por un bloque interno en el tablero de desarrollo. Este bloque tiene la capacidad de emitir 15 tonos de ruido, los mismos que se emiten de acuerdo a la combinación que se establezca en los switches del tablero de desarrollo.

Figura 69. Implementación para el Escenario 1

Fuente: Autores

2.7.2. Escenario 2: Audios Pre-Grabados

Para esta prueba se trabaja con una señal de audio y una señal de ruido pre-grabadas.

Figura 70. Implementación para el Escenario 2

Fuente: Autores

2.7.3. Escenario 3: Audio En Tiempo Real

Para esta prueba se hace uso de dos micrófonos, los mismo que contendrán una señal de voz y una señal de ruido correspondiente al entorno, ambas obtenidas en tiempo real, es decir en el instante de la prueba.

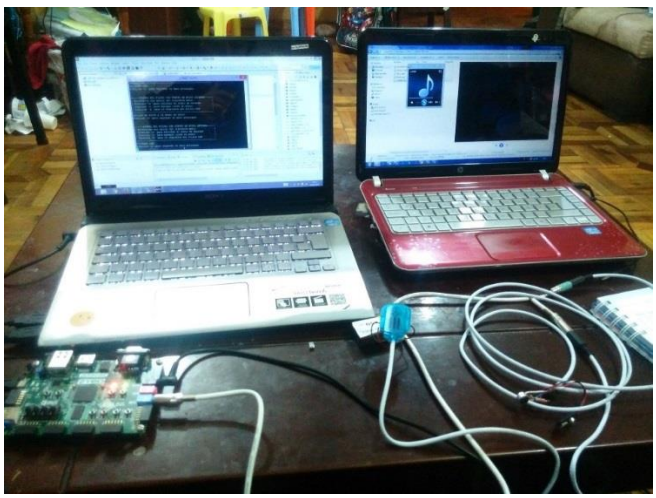


Figura 71. Implementación para el Escenario 3

Fuente: Autores

CAPÍTULO III

3. RESULTADOS

El filtro con algoritmo adaptativo LMS en FPGA se somete a tres escenarios de prueba con diferentes fuentes de generación de ruido. Los resultados obtenidos en cada una de las pruebas realizadas para cada caso, se pueden apreciar en las imágenes del Software Wave Pad que contiene los espectros de las señales.

En cada imagen se observan tres señales. La primera es la señal de audio pura. La segunda es la señal de audio contaminada con el ruido; lo que se nota en esta señal es una especie de relleno de la señal de audio con la señal ruido. Y, en la señal final, se tiene la salida filtrada, que indica la correcta funcionalidad del filtro porque es bastante aproximada a la señal original.

3.1. Pruebas con Onda Generada - Ruido Interno

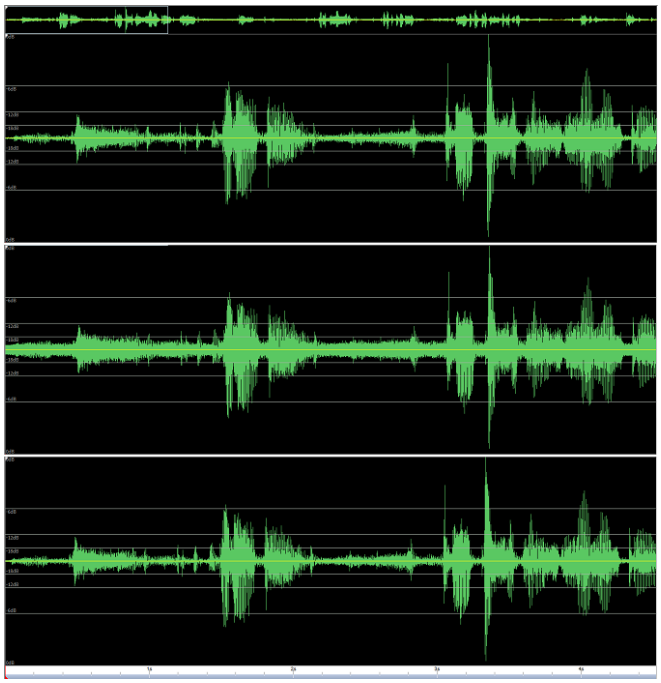


Figura 72. Resultados obtenidos a través de Wave Pad para la prueba con Onda Generada - Ruido Interno

Fuente: Autores

3.2. Pruebas con Audios Pre-Grabados

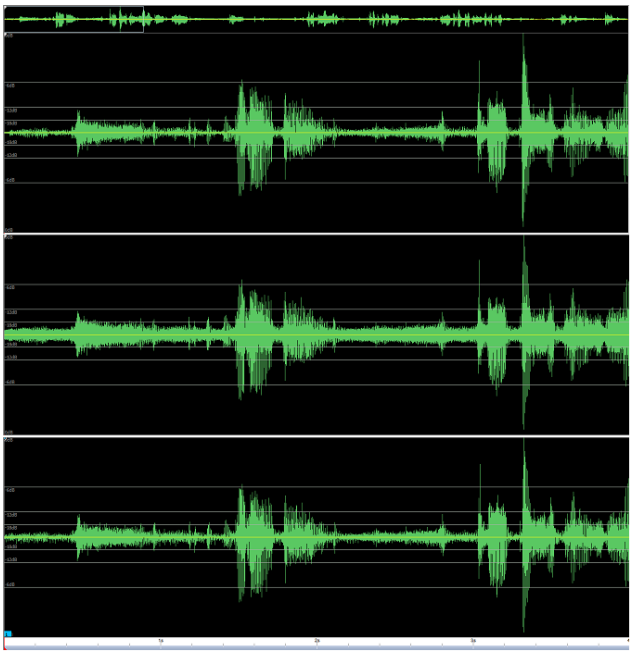


Figura 73. Resultados obtenidos a través de Wave Pad para la prueba con Audios Pre-Grabados

Fuente: Autores

3.3. Pruebas con Audios En Tiempo Real

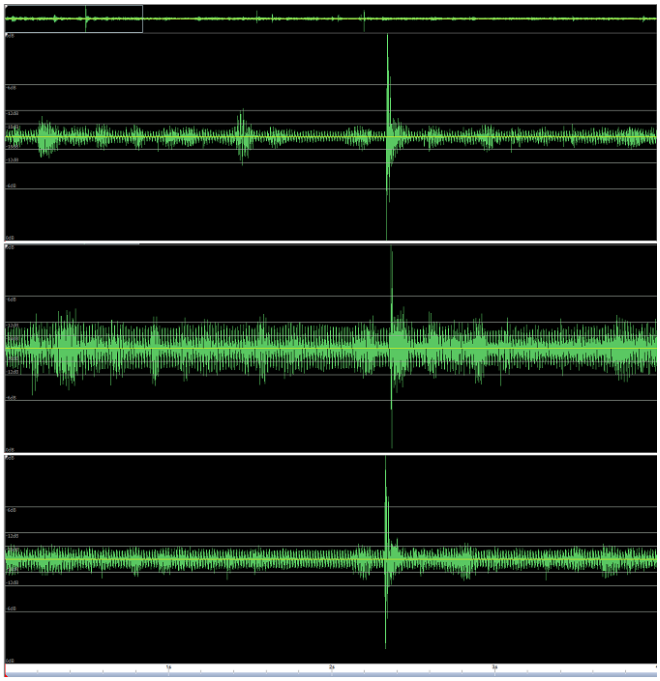


Figura 74. Resultados obtenidos a través de Wave Pad para la prueba con Audios en Tiempo Real

Fuente: Autores

3.4. Comprobación de la Hipótesis

El procedimiento mostrado a continuación sigue un conjunto de pasos para afirmar o negar la hipótesis.

Las pruebas realizadas siguen un esquema de aseveración aproximadas de una media poblacional desconocida.

3.4.1. Planteamiento de la Hipótesis

3.4.1.1. Hipótesis Nula (H0)

La implementación de un filtro con algoritmo adaptativo LMS utilizando una FPGA no cancelará el ruido en señales de audio.

3.4.1.2. Hipótesis Alternativa (H1)

La implementación de un filtro con algoritmo adaptativo LMS utilizando una FPGA cancelará el ruido en señales de audio.

3.4.2. Establecimiento del Nivel de Significancia

Para obtener un resultado confiable para la investigación, se eligió un nivel de significancia de 0.05, lo que representa un nivel de 0.95 de confianza.

Si el nivel de confianza es demasiado próximo a 1 su probabilidad de buen funcionamiento sería altísima, pero a costa de una longitud del intervalo demasiado grande, convirtiéndolo así en algo inútil. Por este motivo, suele tomarse 0.95, que representa un valor de compromiso.

3.4.3. Determinación del Valor Estadístico de Prueba

Para la comprobación de la hipótesis se utiliza el método estadístico de Chi-Cuadrado mediante la Prueba de Bondad de Ajuste, en la que toman en cuenta el número de éxitos en las pruebas realizadas, en este caso el número de veces que el filtrado ha cumplido con la cancelación de ruido, determinado mediante la audición.

Para la aceptación de la hipótesis nula o de la hipótesis alternativa, se toma en cuenta la relación del resultado con los siguientes parámetros:

$$\text{H0 se acepta si: } {X_E}^2 \leq {X_C}^2$$

$$\text{H1 se acepta si: } {X_E}^2 > {X_C}^2$$

Después de las pruebas realizadas se obtiene la siguiente tabla:

	C	AV	AVTR	TOTAL
Escenario 1	13	3	2	18
Escenario 2	9	2	2	13
Escenario 3	12	2	1	15

Tabla 3. Datos obtenidos en las Pruebas

Fuente: Autores

Tomando en cuenta los datos obtenidos en la tabla y los principios que rigen el método estadístico se tiene que:

Números de pruebas n $\quad n = O_1 + O_2 + O_3 + \cdots + O_k = 46$

probabilidad de ocurrencia p_i $\quad p_i = 1/3$

numero de resultados posibles k $\quad k = 3$

conjunto de frecuencias observadas O

Grados de libertad $G_l = k - 1$ $\quad G_l = 2$

Frecuencia esperada (teórica) $E_i = n * p_i$ $\quad E_i = 15.33$

Chi-cuadrado estadístico ${x^2}_E = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$

$$x^2 = \frac{(18 - 15.33)^2}{15.33} + \frac{(13 - 15.33)^2}{15.33} + \frac{(15 - 15.33)^2}{15.33}$$

$$x^2 = 0.4650 + 0.3541 + 0.0071$$

$$\boxed{x^2{}_E = 0.8262}$$

Chi-cuadrado critico $x^2{}_C$ se obtiene de la tabla siguiente:

Grados de Libertad	Probabilidad					
	0.80	0.85	0.90	0.95	0.975	0.99
1	0.0642	0.0358	0.0158	0.0039	0.0010	0.0002
2	0.4463	0.3250	0.2107	0.1026	0.0506	0.0201
3	1.0052	0.7978	0.5844	0.3518	0.2158	0.1148
4	1.6488	1.3666	1.0636	0.7107	0.4844	0.2971
5	2.3425	1.9938	1.6103	1.1455	0.8312	0.5543
6	3.0701	2.6613	2.2041	1.6354	1.2373	0.8721
7	3.8223	3.3583	2.8331	2.1673	1.6899	1.2390
8	4.5936	4.0782	3.4895	2.7326	1.1797	1.6465
9	5.3801	4.8165	4.1682	3.3251	2.7004	2.0879
10	6.1791	5.5701	4.8652	3.9403	3.2470	2.5582

Tabla 4. Tabla de Valores Críticos de Chi-Cuadrado

Fuente: Monografías.com

Ya que $\alpha = 0.05$ entonces la probabilidad se toma de 0.95 por lo tanto $x^2{}_C$ =0.1026

Se compara $x^2{}_E$ con $x^2{}_C$ y se determina que es mayor, por lo tanto, se acepta la Hipótesis Alternativa H1, que versa: La implementación de un filtro con algoritmo adaptativo LMS utilizando una FPGA dará una respuesta que se aproxime a la cancelación del ruido en señales de audio.

CAPÍTULO IV

4. DISCUSIÓN

El presente trabajo de investigación tiene por objetivo el diseño e implementación de un filtro con algoritmo adaptativo en FPGA para la cancelación de ruido en señales en frecuencias audibles.

Profundizando en el análisis de los filtros digitales, algoritmos adaptativos y filtros adaptativos propiamente, el estudio evidencia las ventajas que estos últimos sistemas ofrecen para filtrar señales variantes en el tiempo.

Esta experiencia también hace visibles los beneficios proporcionados por un dispositivo de lógica programable, en cuanto a hardware, como es la velocidad de procesamiento y la disminución de costos debido a la reutilización de hardware.

El sistema implementado no genera inconvenientes de retardo, ni alteración considerable de las señales procesadas, inclusive a la hora de tratar señales en tiempo real.

Existen varios estudios e implementaciones en dispositivos programables como DSPs y PSoCs, con resultados exitosos pero la diferencia y mejoría en este proyecto es la utilización del tablero de desarrollo ZYBO, que brinda la posibilidad de programar una FPGA en lenguaje de alto nivel y diseñar la arquitectura por diagramas de bloques.

Cabe recalcar que el filtro con algoritmo adaptativo LMS puede cubrir necesidades de tratamiento de señales, no solo en tratamiento de frecuencias de audio, sino también para el área de comunicaciones.

CAPÍTULO V

5. CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- El tablero de desarrollo FPGA- ZYBO, posee un procesador ARM CORTEX 9 de 650MHz y una memoria DDR3 de 512MB, lo que permite un mejor rendimiento a la hora del tratamiento de señales digitales, porque no genera retrasos o cambios considerables en las señales tratadas.
- El algoritmo LMS se caracteriza por tener una velocidad de convergencia muy lenta debido a que necesita mayor número de iteraciones. En el bloque de Simulink, la longitud del vector de pesos se encuentra configurada por defecto con un valor de 16. después de realizar el proceso experimental este valor se incrementa y se fija en 20, pues a partir de ese valor no existen mejoras considerables en el proceso de filtrado.
- La implementación del filtro adaptativo LMS en la FPGA – ZYBO, no requiere circuitos externos para los procesos de conversión analógica – digital y viceversa, debido a que el SSM2603 cuenta con ADC y DAC internos.
- En el filtro adaptativo LMS, a diferencia de los filtros digitales, no es necesario tener en cuenta el orden del filtro para optimizar el proceso de filtrado, porque si se desea eliminar una señal solo basta con introducirla en la entrada de la señal de ruido.
- Cuando en el filtro adaptativo LMS implementado en una FPGA, se ingresan señales de frecuencias muy próximas entre sí, la señal propuesta como ruido es eliminada casi en su totalidad, pero deja como rastro un perceptible zumbido.

5.2. RECOMENDACIONES

- El tablero de desarrollo ZYBO posee dos líneas de entrada, una de micrófono y una de audio, pero no es posible usarlas simultáneamente;

esto se debe a que el codificador de audio SSM2603 en la configuración del registro de la ruta de audio analógico, específicamente en el bit INSEL, solo permite la conexión con el ADC de una de las entradas. Por lo tanto, para la implementación de este proyecto la entrada de audio precisa una división en sus dos canales respectivos; el derecho para la señal de entrada y el izquierdo para el ruido. En caso de llegar a futuras reproducciones de este proyecto se recomienda seguir el mismo esquema para un correcto funcionamiento.

- Se recomienda revisar los voltajes tolerables de las entradas de audio (0.54 a 1.09 V) en el tablero de desarrollo, para evitar daños del codificador SSM2603 y el deterioro total o parcial de la ZYBO.
- Para contrarrestar la atenuación que provoca el filtro en la señal de salida, se recomienda la aplicación de un amplificador.

CAPÍTULO VI

6. PROPUESTA

6.1. Título de la Propuesta

"Diseño e Implementación de un Filtro con Algoritmo Adaptativo en FPGA para la Cancelación de Ruido"

6.2. Introducción

En el tratamiento de señales digitales, existen casos en donde las señales a analizar son variantes en el tiempo, por lo que un filtro de coeficientes constantes no es suficiente para la eliminación de ruido en el caso de señales de audio.

Es así que, para los mencionados casos, lo recomendable es la implementación de un filtro adaptativo, que tiene la capacidad de alterar dinámicamente su función de transferencia, en base a un criterio preestablecido como es el que determina el algoritmo adaptativo.

Las características que posee un filtro adaptativo hace que para su implementación sea necesario un hardware que provea una velocidad adecuada a los procesos de cambio, es así que una FPGA se muestra como un dispositivo con ventajas para cumplir con los requerimientos mencionados.

6.3. Objetivos

6.3.1. General

Diseñar e Implementar un Filtro con Algoritmo Adaptativo en FPGA para la Cancelación de Ruido.

6.3.2. Específicos

Identificar el Algoritmo Adaptativo LMS.

Estudiar la FPGA - ZYBO

Diseñar un filtro con algoritmo adaptativo LMS.

Implementar el filtro adaptativo en la FPGA - ZYBO.

Realizar pruebas para comprobar el funcionamiento del filtro adaptativo.

6.4. Fundamentación Científico – Teórica

Los filtros adaptativos, son sistemas con la facultad de modificar sus parámetros durante el procesamiento de una señal, es decir que cambian sus características obedeciendo a criterios predeterminados establecidos por un algoritmo adaptativo, en el caso del presente trabajo los criterios los establece el algoritmo LMS. Lo que permite facilitar el procesamiento de señales en los casos donde las señales varían en el tiempo.

La implementación de este proyecto, mediante una plataforma lógica programable como es la FPGA de Xilinx, permite alcanzar velocidades de hardware adecuadas para la aplicación definida para la cancelación de ruido.

6.5. Descripción de la Propuesta

El presente proyecto consiste en la cancelación de ruido mediante un filtro con algoritmo adaptativo LMS. Para esto se han cumplido cuatro etapas con softwares especializados para cada una. Es así que se tiene:

Etapa 1: Caracterización del bloque del filtro con algoritmo adaptativo

Etapa 2: Diseño general del proyecto y creación del IP LMS mediante Matlab.

Etapa 3: Bosquejo del hardware mediante diseño de bloques a través de Vivado.

Etapa 4: Programación de Software, carga y ejecución de la aplicación mediante SDK.

6.6. Diseño Electrónico de la Propuesta

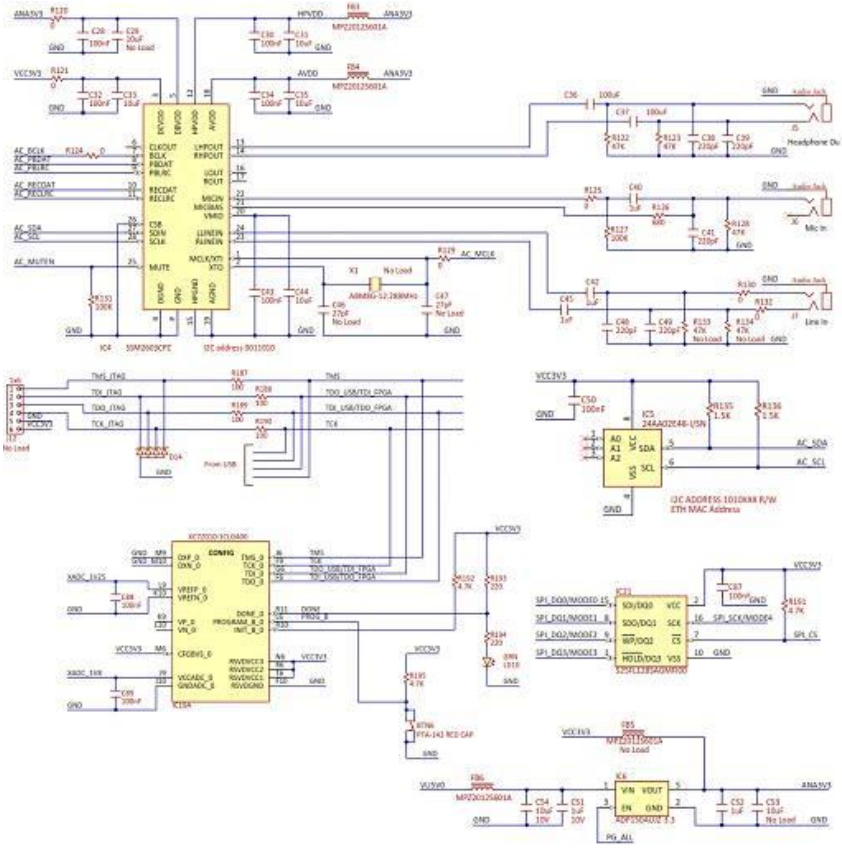


Figura 75. Diseño Electrónico de la Propuesta

Fuente: Autores

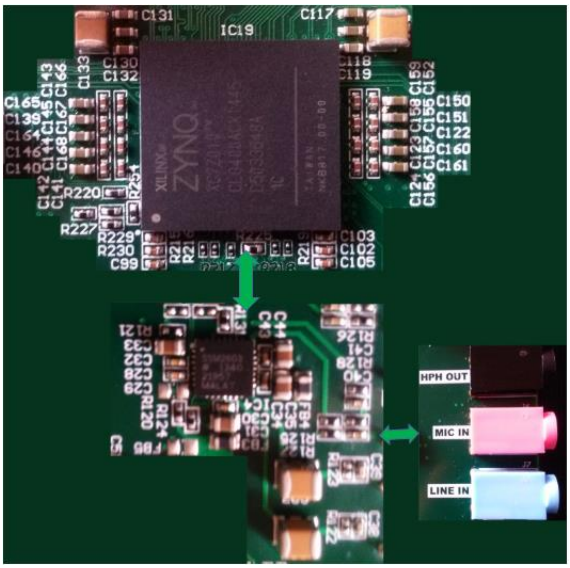


Figura 76. Propuesta Electrónica

Fuente: Autores

6.7. Diseño Organizacional

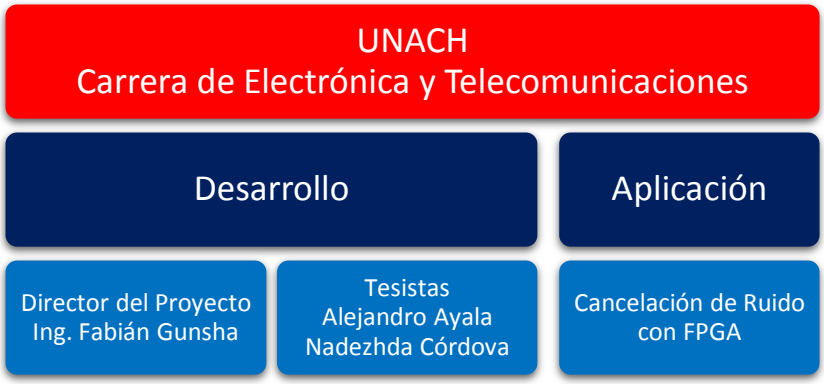


Figura 77. Diseño Organizacional

Fuente: Autores

6.8. Monitoreo y Evaluación de la Propuesta

Para el monitoreo y evaluación de la propuesta, se realizará pruebas en tres diferentes ambientes, con ruido generado internamente, audios pre-grabados y audios en tiempo real. Para evaluar el equipo correctamente se documentará cada prueba realizada, de forma que, al revisar el control y la confiabilidad de funcionamiento del filtro con algoritmo adaptativo en una FPGA, se pueda establecer las limitaciones que presenta y poder corregirlas.

.

7. BIBLIOGRAFÍA

Alba, E. F., & Ruiz, J. (2009). Implementación de Filtros Digitales en FPGA. *Bioingeniería y Física Médica Cubana*, 9-13.

Albiol, A., Naranjo, V., & Prades, J. (2007). *Tratamiento Digital de la Señal.* Valencia: Universidad Politécnica de Valencia Servicio de Publicación.

Alexander, C. K., & Sadiku, M. N. (2006). *Fundamentos de Circuitos Eléctricos.* México: McGraw - Hill Interamericana.

Álvarez Cedillo, J. A., Lindig Bos, K. M., & Martínez Romero, G. (2008). Implementación de Filtros Digitales tipo FIR en FPGA. *Polibits*, 83-87. Obtenido de Polibits.

ANALOG DEVICES. (21 de Junio de 2013). Datasheet SSM2603.

Bustamante, R. (Febrero de 1999). *Universidad Nacional Mayor de San Marcos.* Obtenido de http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/electronica/Febrero_1999/Pdf/04_filtros.pdf

Chu, P. P. (2011). *FPGA PROTOTYPING BY VHDL EXAMPLES.* Hoboken: John Wiley & Sons.

Crockett, L. H., Elliot, R. A., Enderwitz, M. A., & Stewart, R. W. (2014). *The Zynq Book.* Glasgow: Strathclyde Academic Media.

DIGILENT. (11 de Abril de 2016). ZYBO FPGA BOARD REFERENCE MANUAL. Pullman, Washington, Estados Unidos.

Floyd, T. L. (2008). *Dispositivos Electrónicos.* México: PEARSON EDUCACIÓN.

Gonzales, R., & Parra, C. (30 de Diciembre de 2005). *Sitio Web de la Universidad Distrital Francisco José de Caldas.* Obtenido de Universidad Distrital Francisco José de Caldas: ftp://ftp.udistrital.edu.co/Documentacion/Electronica/Dsp/capitulo5.PDF

González, F. J. (18 de Abril de 2012). Tema 4. Filtros Analógicos. Madrid, España.

Haykin, S. (2014). *Adaptative Filter Theory.* Hamilton: Pearson.

Jones, D. L. (12 de Mayo de 2005). *OpenStax CNX.* Obtenido de OpenStax CNX: http://cnx.org/contents/c2d33b02-f6b3-4e1b-8eaa-44946614adcc@1.1/Adaptive-Filters

National Instruments Corporation. (20 de Junio de 2012). *National Instruments Corporation*. Obtenido de National Instruments Corporation Web Site: http://www.ni.com/white-paper/6983/es/

Pérez, C. (2007). *Universidad de Cantabria.* Obtenido de Universidad de Cantabria Web Site: http://personales.unican.es/perezvr/

Ruiz, J. O. (2010). Implementación de Filtros Adaptativos en Tecnologías de Lógica Reconfigurable. *Revista Colombiana de Tecnologías de Avanzada*, 92-98.

Solarte, V., & Jojo, P. E. (2012). El Algoritmo Acelerador Regresivo versión r(ARr) y los efectos de cuantificación. *Revista Universitaria en Telecomunicaciones, Informática y Control*, 29-37.

Soria Olivas, E. (2003). *Tratamiento Digital de Señales: problemas y ejercicios reueltos.* Valencia: Pearson Educación.

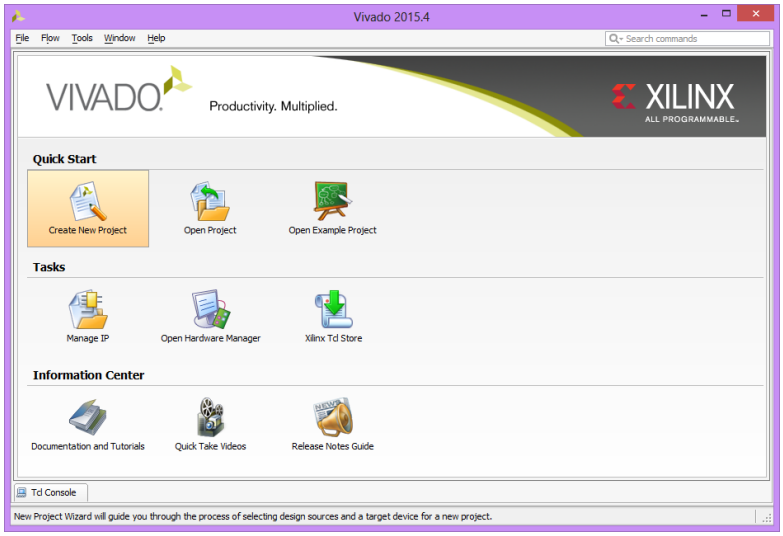
Wantanabe, C. E. (11 de Septiembre de 2012). *Universidad Politécnica de Madrid.* Obtenido de Archivo Digital UPM: http://oa.upm.es/13819/

Zelaya , W. L. (Enero de 2004). Diseño de un filtro digital adaptativo como cancelador de ruido basado en el algoritmo LMS. San Salvador, El Salvador.

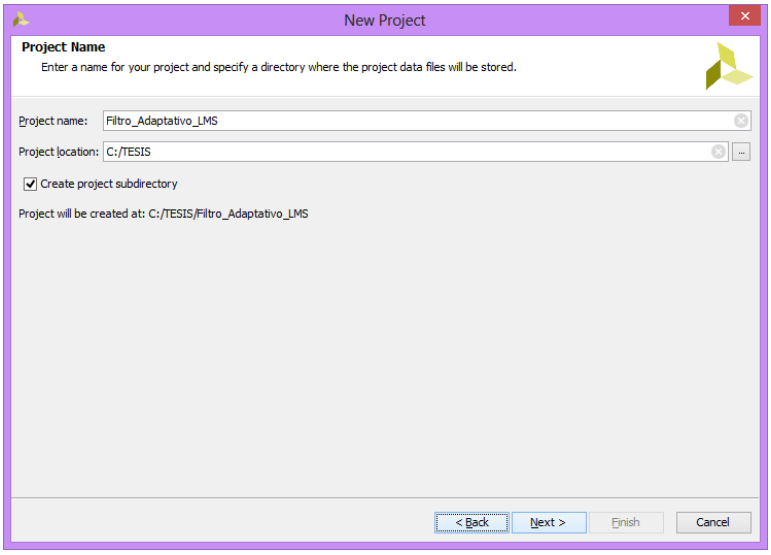
8. APÉNDICES

APÉNDICE A PASOS PARA LA CREACIÓN DE UN NUEVO PROYECTO EN VIVADO SUITE DESIGN.

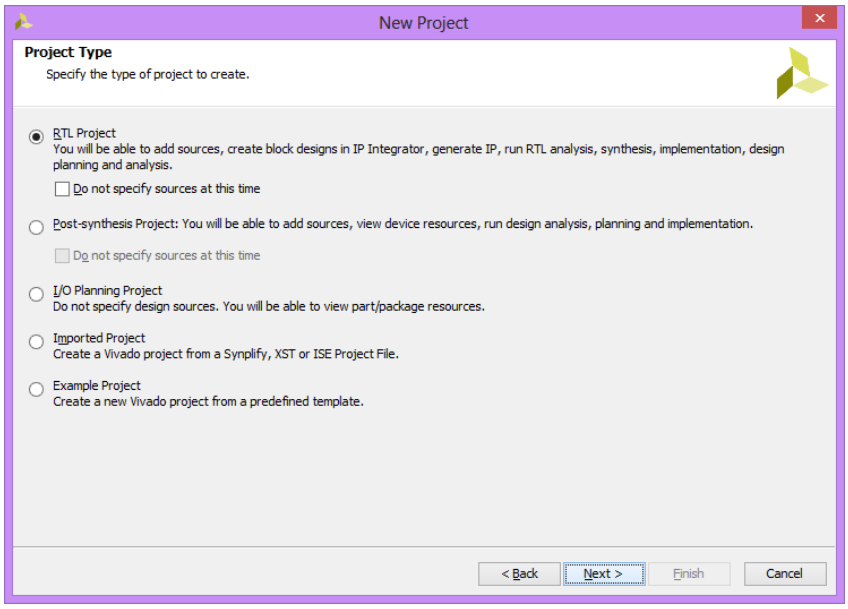
1. Iniciar Vivado ya sea por el icono de acceso directo en el escritorio o en su carpeta de ubicación.
2. En la pantalla de inicio de Vivado se selecciona el icono crear nuevo proyecto.



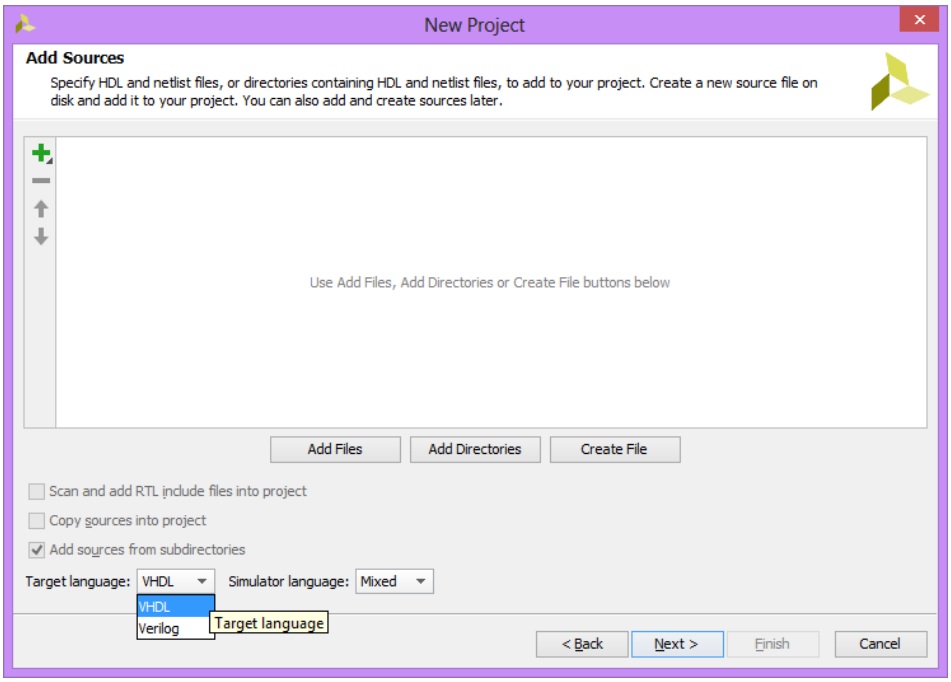
3. La ventana de creación de nuevo proyecto se abrirá y se oprime siguiente.
4. Aparecerá una ventana donde se deberá ingresar el nombre del proyecto y su ubicación. Hay que asegurarse de que la opción Crear subdirectorio del proyecto esté seleccionada, esto creará una carpeta que contenga al proyecto. Dar clic en siguiente.



5. En la ventana que aparecerá se elige proyecto RTL y se da clic en siguiente. (la pestaña de no especificar opciones no debe estar marcada.)

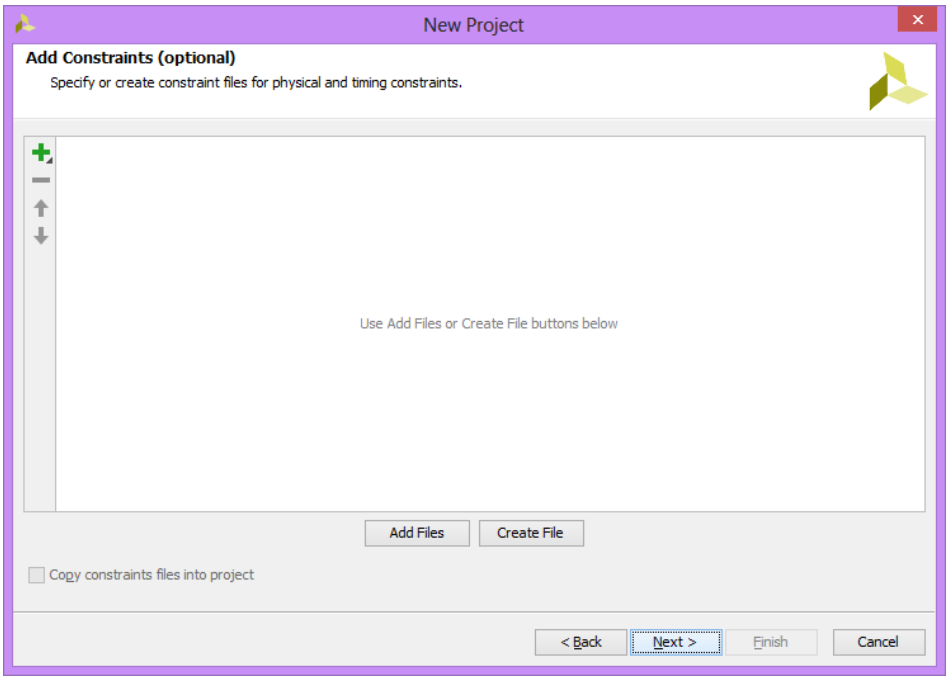


6. En la ventana de agregar fuentes, que aparecerá a continuación, seleccionar VHDL como el lenguaje de destino y MIXED como el lenguaje de simulación. En caso de que existan archivos fuentes, se los puede agregar en este momento al proyecto, en caso de no tener archivos de este tipo solo se oprime siguiente.

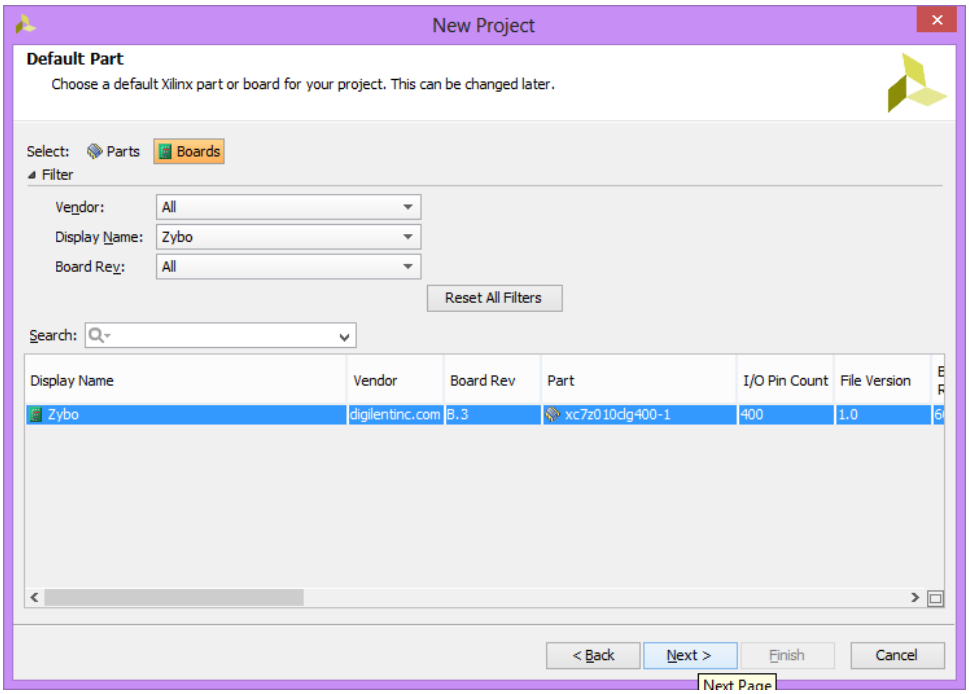


7. Las dos ventanas siguientes permiten agregar IPs existentes y archivos de limitaciones según corresponda, de existir los archivos antes mencionados

deberán agregarse en cada una de estas etapas, caso contrario solamente se debe dar clic en siguiente.



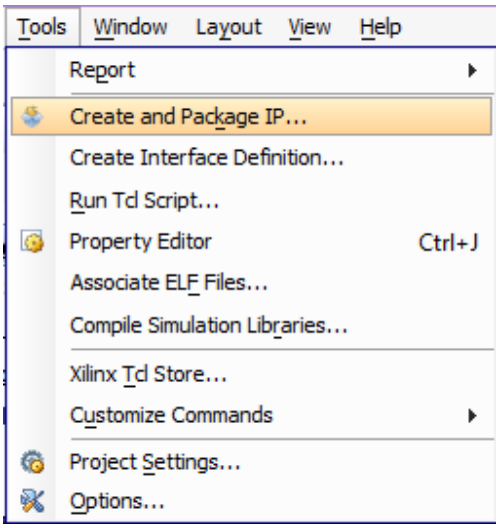
8. En la ventana partes por defecto se accede a la pestaña BOARDS y se elige ZYBO de la lista desplegable en Display Name; se selecciona All en la lista de Board Rev y se toma la tarjeta. Dar clic en siguiente.

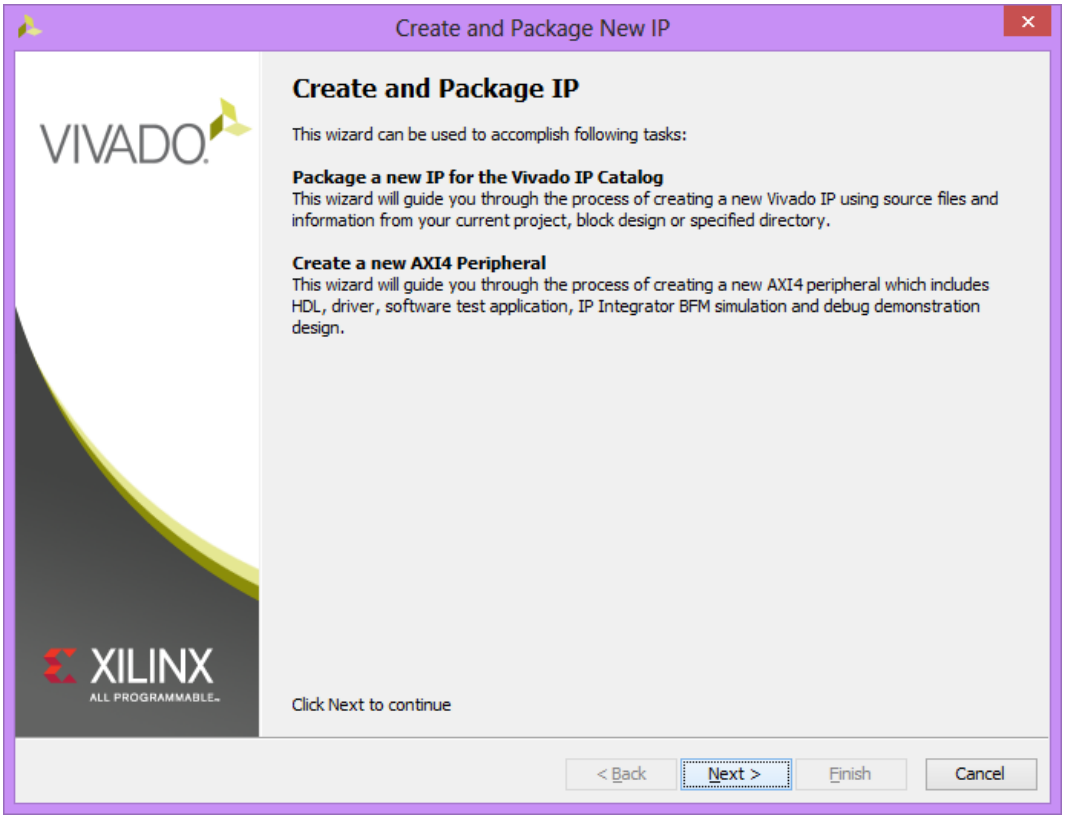


9. La ventana que aparecerá contendrá el resumen del nuevo proyecto con las especificaciones que se han indicado en los pasos anteriores. Se da clic en Finalizar y se creará el proyecto.

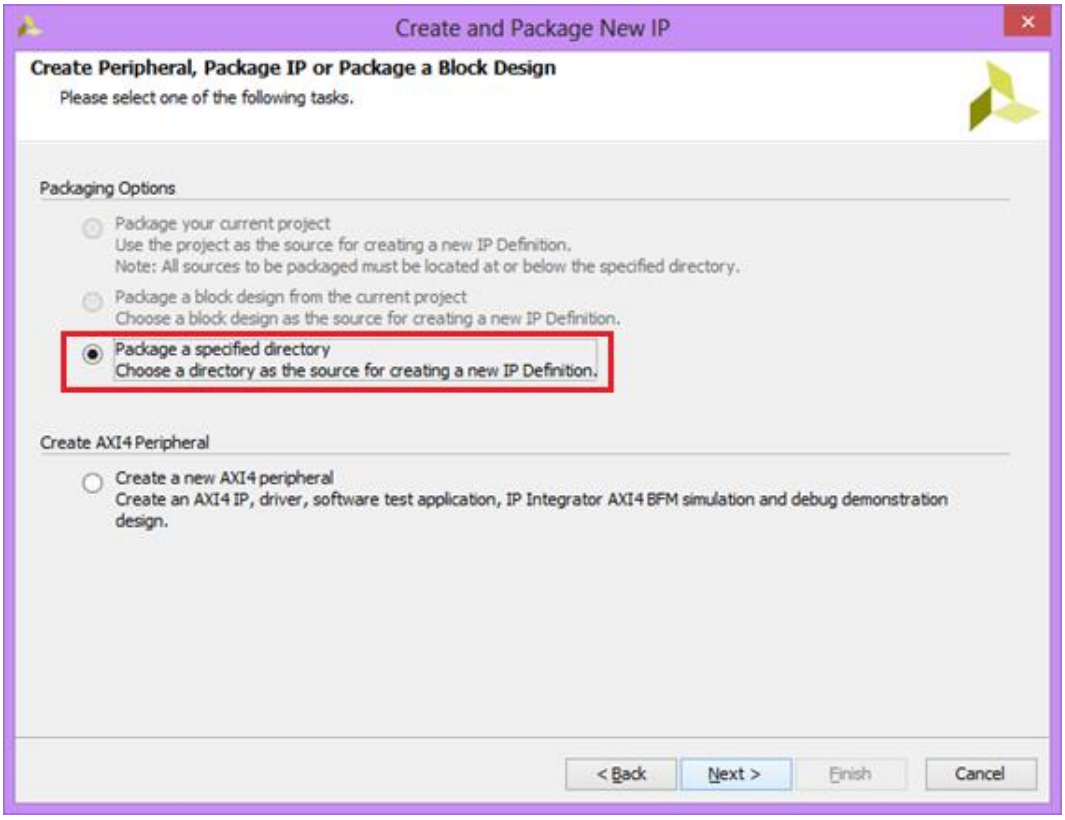
APÉNDICE B PASOS PARA LA HABILITACIÓN DEL NÚCLEO IP LMS

1. Para iniciar, se elige Tools > Create and Package IP de la barra de menú, y se presiona Siguiente.

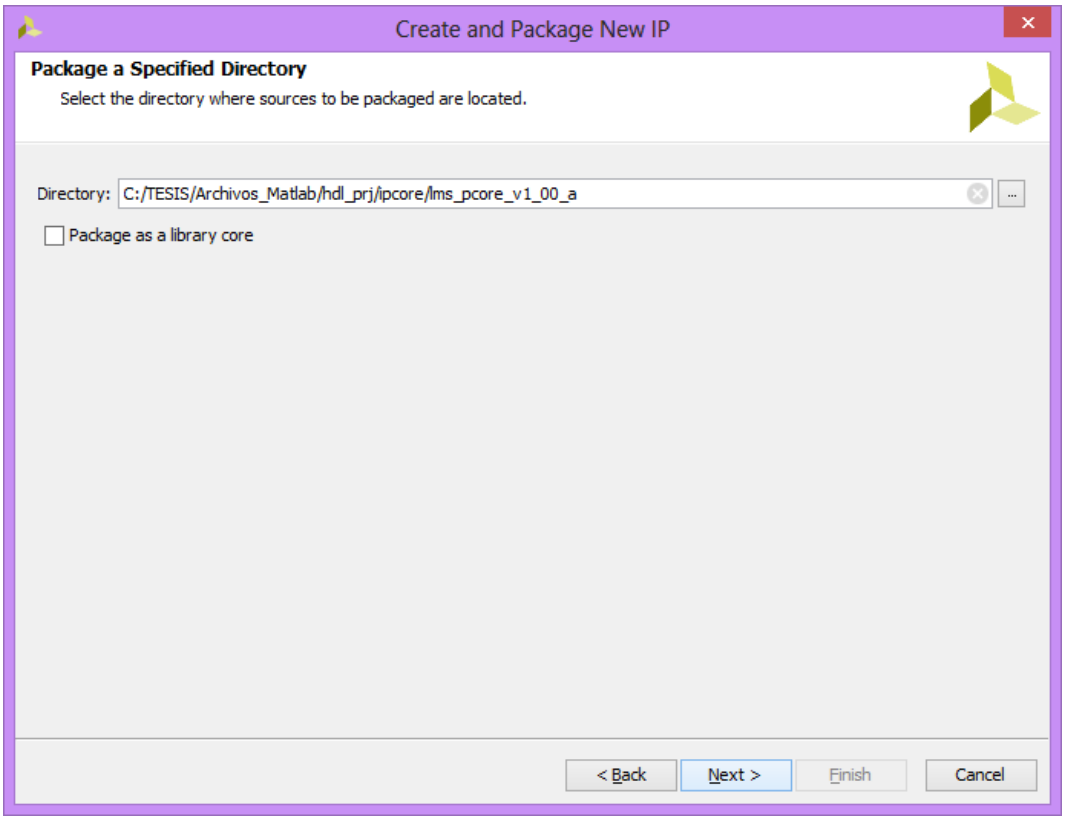




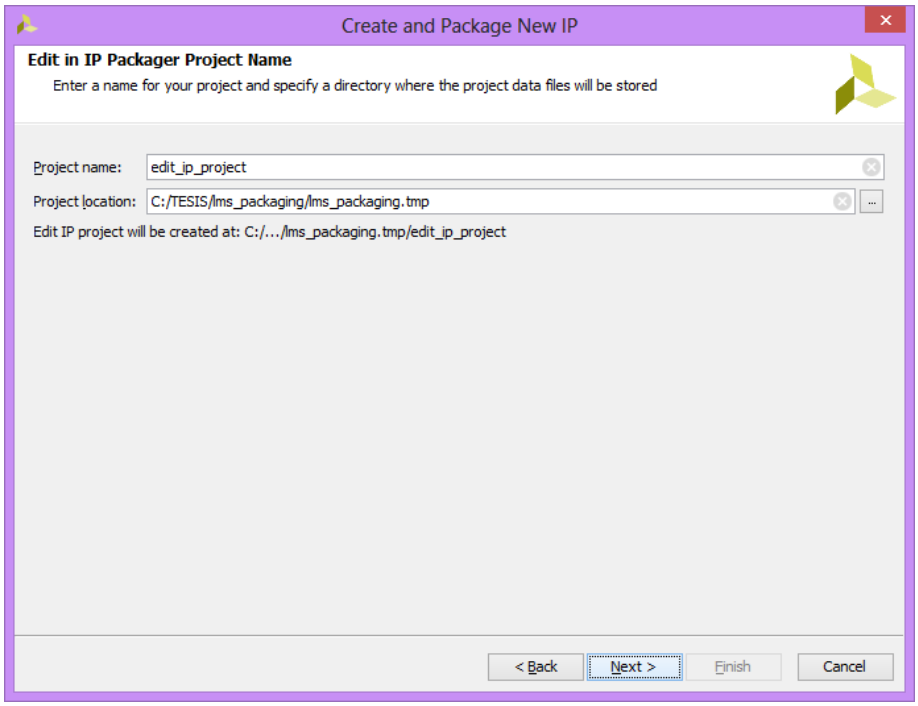
2. Se selecciona la opción Package a specified directory, y se da clic en Siguiente.



3. Se requiere ingresar la ubicación que corresponda del archivo como IP location, y se oprime Siguiente.



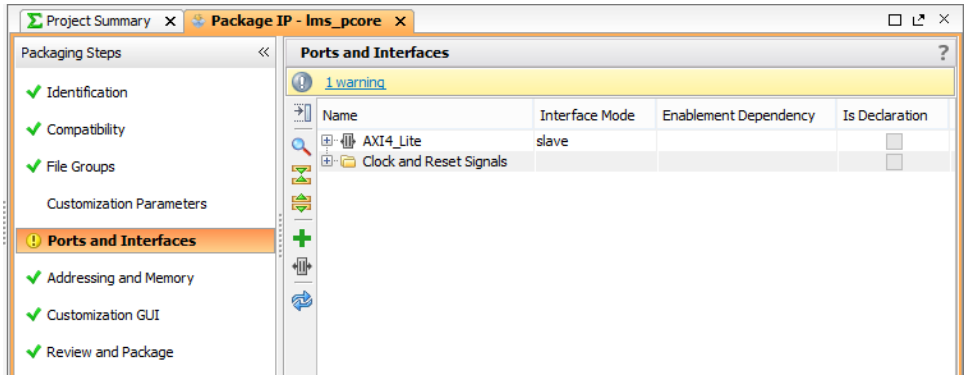
4. En la ventana Edit In IP Package se acepta el nombre y la ubicación del proyecto haciendo clic en siguiente.



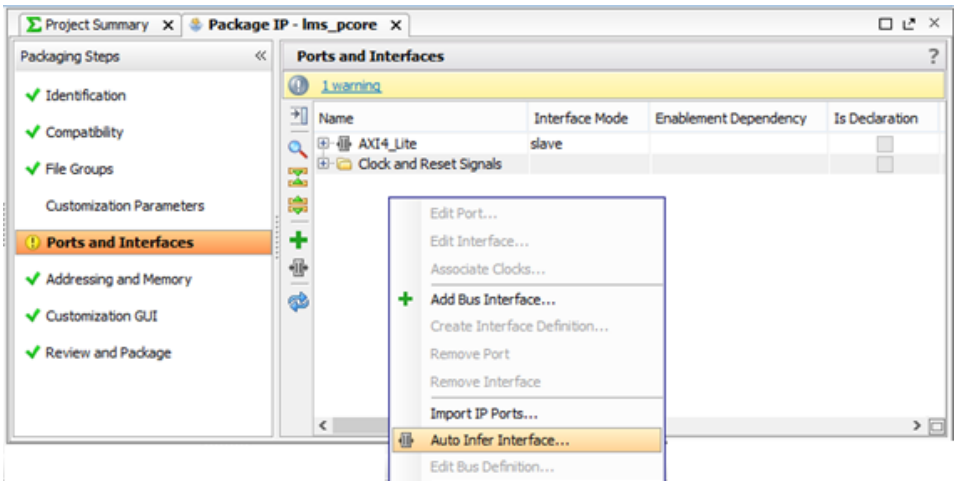
5. En la ventana de resumen se oprime Finalizar para enlazar al IP PACKAGER.

6. En el panel izquierdo de la ventana del IP Packager se ubica PORTS and INTERFACES. El panel de interfaces IP se abrirá y se puede ver que el IP

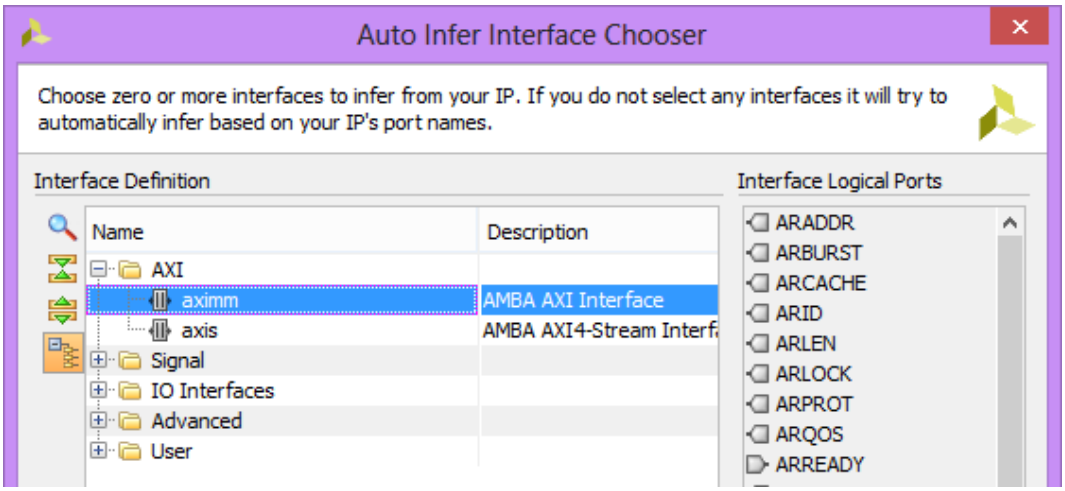
Packager ha identificado los puertos AXI individuales, pero no ha inferido una interfaz AXI.



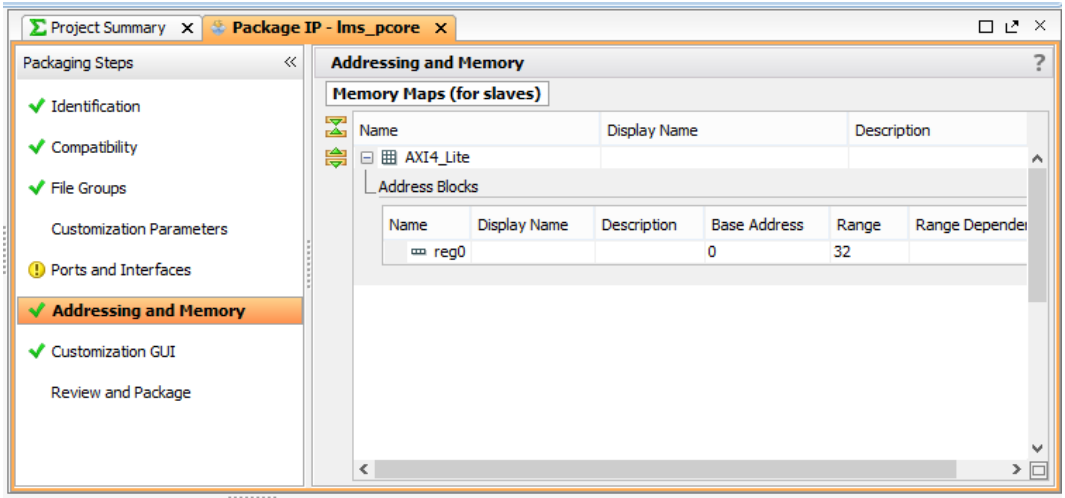
7. Para inferir en una interfaz AXI, primero se hace clic derecho en cualquier sección en blanco del panel de puertos IP, y se escoge Auto Infer Interface…



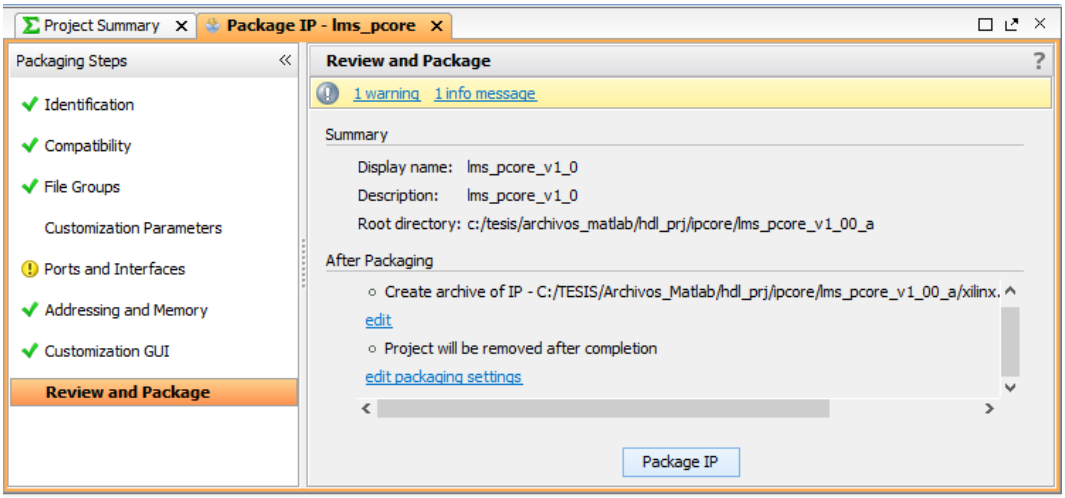
8. La ventana del selector de Auto Inter Interface se abrirá, se toma AXIMM de la lista y los puertos AXI individuales en el diseño se mapearán en una interfaz AXILite.



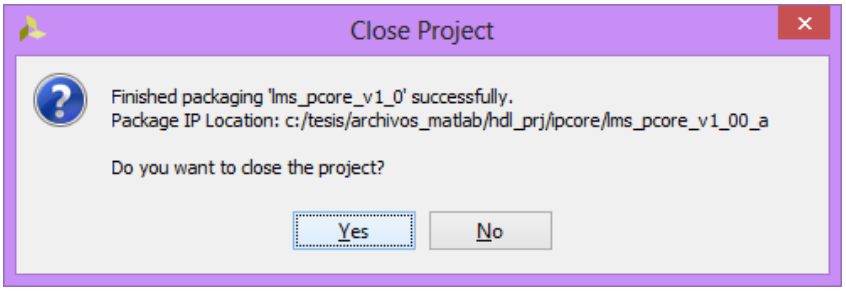
9. Se accede a Addressing and Memory del panel de la izquierda. Y aquí el IP Packager ha especificado una dirección en el rango de 65536, pero es necesario cambiarla para poder trabajar, por ello se cambia el valor en el rango a 32.



10. Finalmente hay que dirigirse a Review and Package del menú de la izquierda, se revisa la información que provee el software y se hace clic en Package IP.

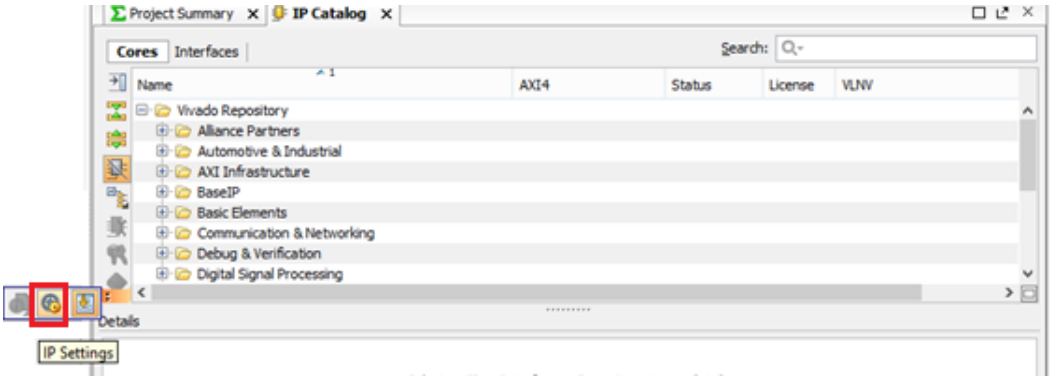


11. Con esto se tiene un núcleo IP habilitado para el uso en Vivado.

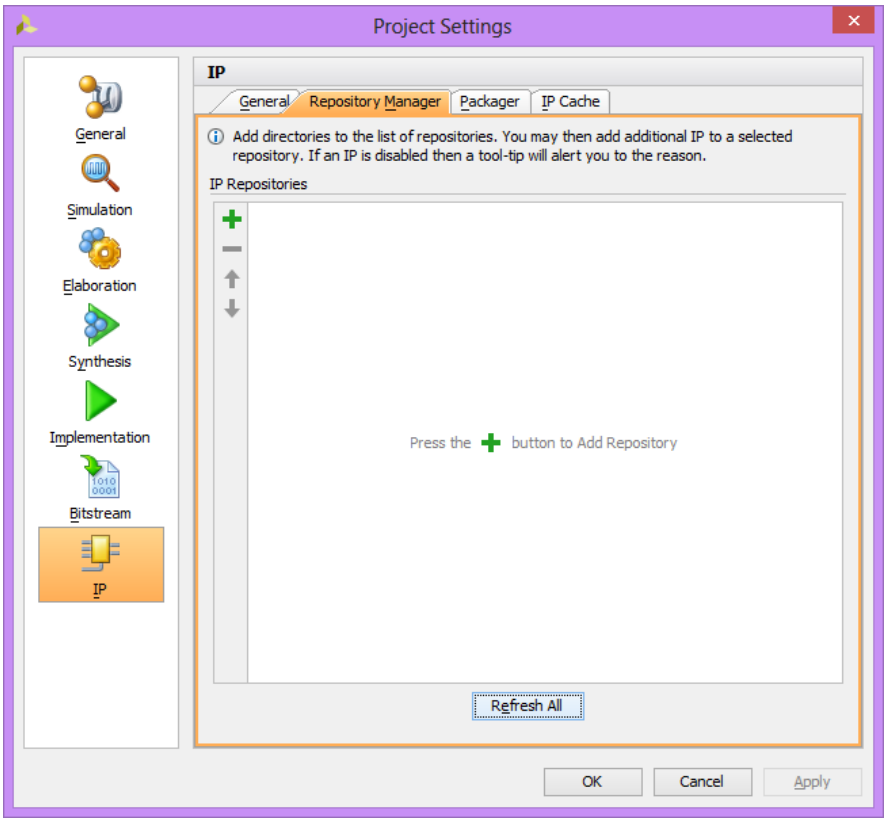


APÉNDICE C PASOS PARA LA ADICIÓN DE LOS NÚCLEOS AL CATÁLOGO IP DE VIVADO

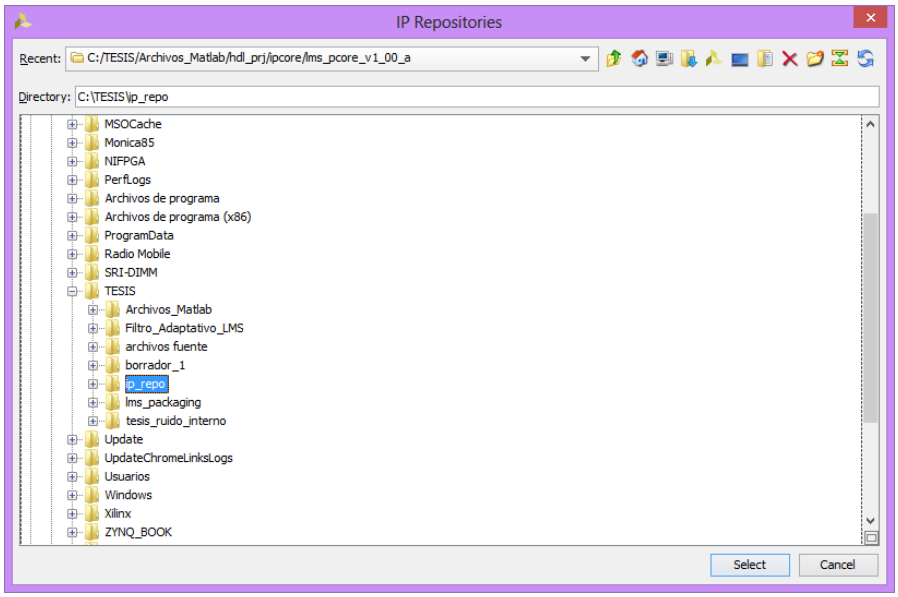
1. En el flujo de navegador se ubica el catálogo IP desde la sección del administrador del proyecto. El catalogo IP se abrirá en el espacio de trabajo.
2. Se verifica que todos los archivos comprimidos relacionados con los núcleos IP del usuario estén almacenados en una misma carpeta, en este caso la carpeta tiene por nombre IP REPO.
3. En el catálogo IP de Vivado dar clic en el botón configuraciones IP.



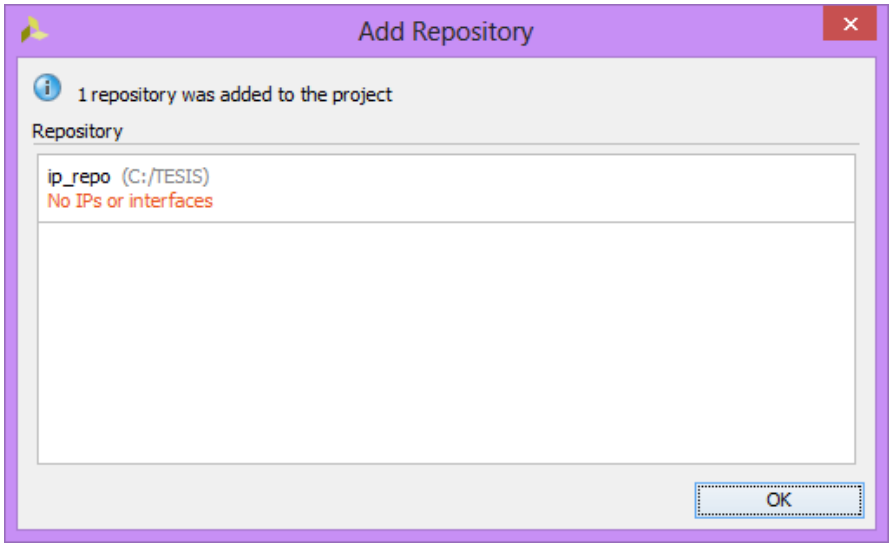
4. En la ventana que se abrirá se da clic en el símbolo + del panel de repositorios IP y se busca la carpeta IP REPO.



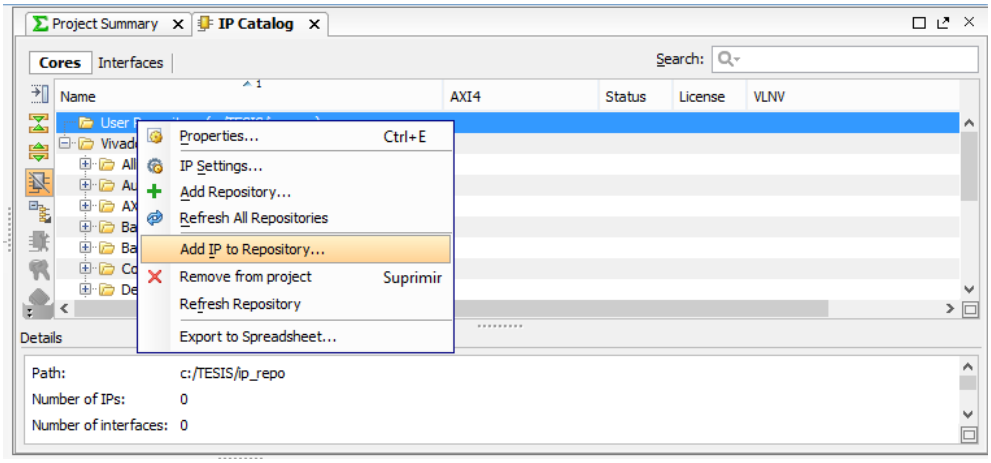
5. Se da clic en seleccionar para agregar esta carpeta como repositorio al catálogo IP.



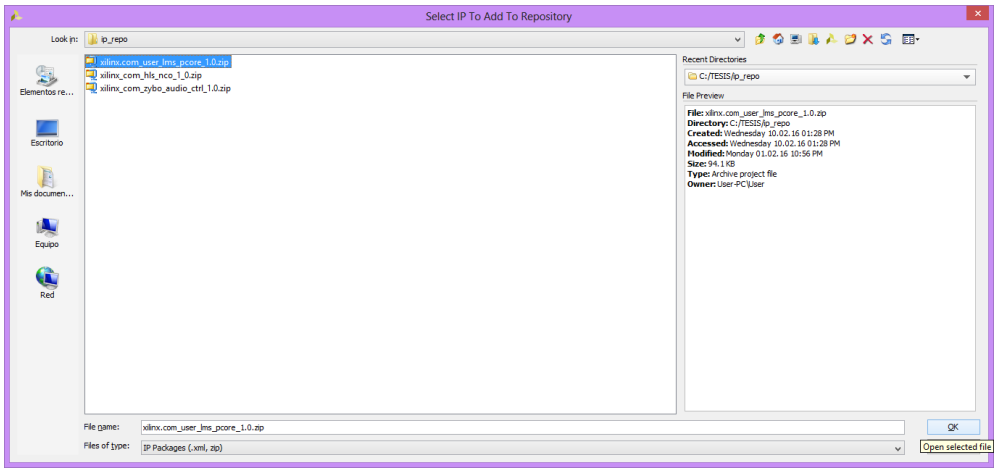
6. Dar clic en OK.



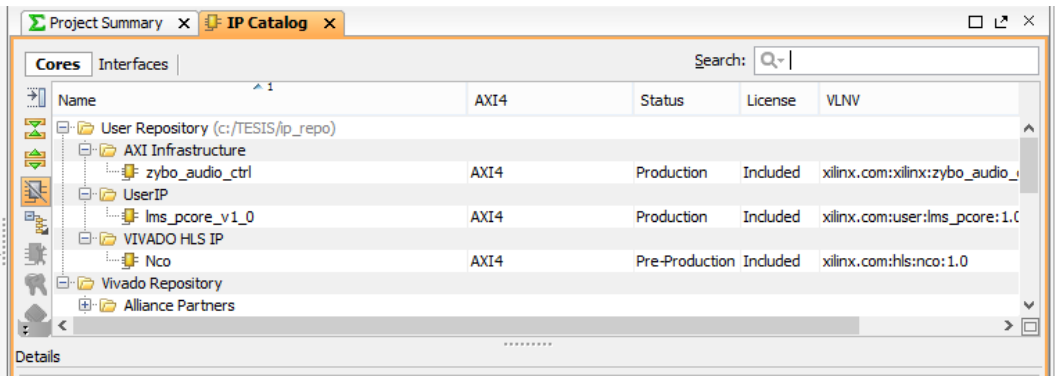
7. Ahora se puede ver que en el catálogo IP aparece un nuevo repositorio denominado USER REPOSITORY, en el que se debe agregar todas las IPs contenidas en los archivos comprimidos.
8. Dar clic derecho en este repositorio y seleccionar Agregar IP al repositorio.



9. En la ventana que se abrirá se escoge individualmente los comprimidos de cada núcleo IP, y luego se da clic en abrir.



10. Repetir los mismos pasos con todos los núcleos IP que se desea añadir y se puede apreciar en el catálogo IP que se van creado carpetas que contienen esas IPs.



Ahora que ya han sido agregadas al catálogo IP pueden ser utilizadas en los diseños.

9. ANEXOS

ANEXO 1 CONFIGURACIÓN ADICIONAL EN MATLAB

1. Descargar e instalar Xilinx ISE 14.7 desde el sitio web o a través del siguiente link: http://www.xilinx.com/products/desing-tools/ise-desing-suite.html
2. Usando el explorador de Windows se localiza el directorio de instalación llamado ise.exe de la aplicación ISE de Xilinx dentro Xilinx ISE 14.7, generalmente la aplicación puede ser encontrada en la siguiente dirección si está instalada en el disco C: C:\Xilinx\14.7\ISE_DS\ISE\bin\nt\ise.exe
3. Se copia la dirección en el portapapeles y se abre Matlab.
4. En el entorno de trabajo se debe ingresar el siguiente comando: hdlsetuptoolpath ('ToolName', 'Xilinx ISE', 'ToolPath', 'C: \Xilinx\14.7\ISE_DS\ISE\bin\nt\ise.exe')

   Donde el cuarto parámetro (la dirección de la aplicación) es el directorio de instalación previamente copiado en el portapapeles.
5. Si la configuración del HDL ToolPath es exitosa la siguiente información es mostrada como resultado:

```
>> hdlsetuptoolpath('ToolName', 'Xilinx ISE', 'ToolPath', 'C:\Xilinx\14.7\ISE_DS\ISE\bin\nt\ise.exe')
Setting XILINX environment variable to:
C:\Xilinx\14.7\ISE_DS\ISE
Setting XILINX_EDK environment variable to:
C:\Xilinx\14.7\ISE_DS\EDK
Setting XILINX_PLANAHEAD environment variable to:
C:\Xilinx\14.7\ISE_DS\PlanAhead
```

HDL Coder ahora puede ser usado para sintetizar código HDL para las Plataformas de Hardware Xilinx.

ANEXO 2 TUTORIAL PARA LA CREACIÓN DE UN PROGRAMA BÁSICO EN VIVADO SUITE DESIGN

1. En la ventana de flujo de navegación se selecciona Crear diseño de boques desde la sección integrador IP. La ventana para crear un diseño de bloques se abrirá.
2. Primero se debe indicar el nombre del proyecto en el espacio indicado y luego dar clic en ok. Entonces el espacio de trabajo se abrirá.
3. El primer bloque que se añade al diseño es el Sistema De Procesamiento Zynq, lo que se puede hacer de dos formas:
    a. Dando clic derecho en cualquier parte del espacio de trabajo y seleccionando agregar IP.
    b. Presionando el botón de agregar IP en la barra de herramientas, en el lado izquierdo del espacio de trabajo.
4. El menú emergente del catálogo de IPs se abrirá, entonces se busca y selecciona el Zynq7 Processing System.
5. Ahora se presiona Run Block Automation en la parte superior del espacio de trabajo, lo que abrirá una ventana de diálogo en la que se debe verificar que la opción Apply Board Preset esté seleccionada. Clic en OK. De esta manera se crean las conexiones externas para las interfaces DDR y FIXED_IO.

   Ya que la plataforma ZYBO fue especificada como el tablero de desarrollo destinado, en la creación del proyecto, Vivado configurará el bloque del procesador Zynq de acuerdo a los requerimientos de la tarjeta.

Todos los pasos realizados hasta aquí permiten agregar lo que vendría a ser la PS de la Zynq y también configurarla. En los siguientes pasos se agregan los bloques que se ubican en la PL para agregar funcionalidad al sistema; en este caso solo se agrega un bloque que permita la utilización de los leds del tablero de desarrollo, este bloque es conocido como AXI GPIO.

6. Se añade el bloque AXI GPIO y se usa la herramienta del asistente de diseño del Integrador IP, para realizar la conexión automática entre el bloque AXI GPIO y el bloque del sistema de procesamiento Zynq 7.
7. Al dar clic en Run Connection Automation en la parte superior del espacio de trabajo, se abrirá una ventana en la cual se debe seleccionar S_AXI en el menú

izquierdo y en la parte derecha se verifica que la opción auto esté seleccionada. Clic en OK y todas las conexiones entre los bloques se harán automáticamente.

8. La ultima conexión requerida es la del bloque GPIO a los leds del tablero de desarrollo; nuevamente se da clic en Run Connection Automation y esta vez se selecciona GPIO del menú izquierdo, pero en la parte derecha se escoge leds_4bits. Dar clic en OK.

Ahora el bloque AXI GPIO se encuentra conectado a los leds del tablero de desarrollo y con esto el diseño está completo.

El integrador IP asigna el mapa de memoria automáticamente para todas las IP presentes en el diseño, y se puede observar esto en el editor de direcciones. En este caso los registros del mapa de memoria para los bloques IP en la PL están en el rango de los 64K.

9. Se guarda el diseño seleccionando File>Save Block Design en la barra de menú.
10. Se valida el diseño seleccionando Tools>Validate Design en la barra de menú, y se ejecutará una comprobación de reglas de diseño(DRC).

    Este proceso se puede realizar también seleccionando el botón de validar diseño desde la barra de herramientas principal o dando clic derecho en cualquier parte del espacio de trabajo y seleccionando validar diseño.
11. La ventana de validación del diseño aparecerá para confirmar que la validación ha sido exitosa. Aquí solo se da clic en OK.

Con el diseño validado se puede generar los archivos HDL para el sistema.

12. Seleccionar la pestaña Fuentes en la barra de menú, dar clic derecho en el primer archivo de las fuentes de diseño y se escoge Create HDL Wrapper. Una ventana de dialogo aparecerá y se debe elegir Let Vivado Manage Wrapper And Auto-Update. Dar clic en OK.

Con los archivos de diseño HDL generados el siguiente paso en Vivado es implementar el diseño y generar el archivo de flujo de bits (bitstream).

13. En el navegador de flujo se da clic en Generar Bitstream de la sección Program and Debug. En caso de que apareciera una ventana preguntando si se desea guardar el diseño, se cliquea en Guardar.

14. Aparecerá una ventana de diálogo antes de comenzar el proceso de generar bitstream, se da clic en SI para continuar. Este proceso puede tomar unos minutos dependiendo de la capacidad de la computadora.
15. Una vez finalizada la generación del bitstream se abrirá una ventana indicando que el proceso ha sido completado con éxito. En esta ventana se debe seleccionar abrir diseño implementado y dar clic en OK.

Con el bitstream generado, la construcción de hardware está completa. Esto puede ser exportado hacia el programa que permite diseñar la aplicación de software para controlar e interactuar con el hardware diseñado.

16. El paso final en Vivado es exportar el diseño a SDK, donde se creará la aplicación de software que permitirá a la PS de la Zynq controlar los leds en el tablero de desarrollo.
17. Seleccionar File > Export > Export Hardware… desde la barra de menú.
18. En la ventana de exportar hardware se debe seleccionar la opción incluir bitstream y dar clic en OK.
19. Se enlaza el SDK en Vivado seleccionando File>Launch SDK desde la barra de menú. Clic en OK.

ANEXO 3 TUTORIAL PARA LA CREACIÓN DE UN SOFTWARE DE APLICACIÓN BÁSICO EN SDK

1. Una vez abierto SDK seleccionar File > New > Application Project de la barra de menú.
2. Se abrirá la ventana de nuevo proyecto y en ella se ingresa únicamente el nombre del proyecto. Clic en Finalizar.
3. En la pantalla de plantillas de nuevo proyecto elegir Empty Application, y clic en Finalizar.

El nuevo proyecto debe abrirse automáticamente. Si esto no sucede, se requiere cerrar la ventana de bienvenida.

Con el nuevo proyecto de aplicación creado, ahora se puede importar algunos códigos fuente preparados con anterioridad para la aplicación.

4. En el panel de explorador de proyecto se expande la carpeta con el nombre del proyecto y se hace clic derecho en el directorio SRC, de las opciones que se despliegan se selecciona importar.
5. La ventana para importar archivos se abrirá. Se expande la opción General, y se escoge File System. Clic en Siguiente.
6. En la ventana que se abrirá se oprime BROWSE y se busca el directorio en el que están contenidos los archivos fuente. Clic en OK.
7. Seleccionar el archivo con el código fuente y dar clic en Finalizar.

El siguiente paso es programar la PL de la Zynq con el archivo Bitstream generado y exportado desde Vivado.

8. Verificar que el tablero de desarrollo esté encendido y conectado en el puerto JTAG de la PC.
9. Cargar el Bitstream a la PL de la Zynq seleccionando Xilinx Tool> Program FPGA desde la barra de menú. La ventana Program FPGA aparecerá con el archivo Bitstream listo por lo que solo se debe dar clic en Program.

Se puede verificar que la tarjeta ha sido programada cuando el led DONE se haya puesto verde.

Con la PL de la Zynq configurada exitosamente ahora se puede enlazar el software de aplicación en la PS de la Zynq.

10. Clic derecho en la carpeta con el nombre del proyecto y desde el menú seleccionar Run As > Launch On Hardware (GDB).

Se puede visualizar el funcionamiento de la tarjeta con la programación realizada de acuerdo a las especificaciones del usuario.

ANEXO 4 CÓDIGO VHDL GENERADO POR HDL CODER PARA EL NÚCLEO IP LMS.

```
-- -------------------------------------------------------------
--
-- File Name: hdl_prj(1)\hdlsrc\lms\LMS.vhd
-- Created: 2016-05-13 21:24:55
--
-- Generated by MATLAB 8.5 and HDL Coder 3.6
--
--
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Model base rate: 2.26757e-05
-- Target subsystem base rate: 2.26757e-05
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        2.26757e-05
-- -------------------------------------------------------------
--
--
-- Output Signal                 Clock Enable  Sample Time
-- -------------------------------------------------------------
-- e_k                           ce_out        2.26757e-05
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------
```

```
-- -------------------------------------------------------------
--
-- Module: LMS
-- Source Path: lms/LMS
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.LMS_pkg.ALL;

ENTITY LMS IS
 PORT( clk                       :   IN    std_logic;
       reset                     :   IN    std_logic;
       clk_enable                :   IN    std_logic;
       x_k                       :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
       d_k                       :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
       ce_out                    :   OUT   std_logic;
       e_k                       :   OUT   std_logic_vector(15 DOWNTO 0)  -- sfix16_En14
       );
END LMS;
ARCHITECTURE rtl OF LMS IS
 -- Constants
 CONSTANT C_LMS_FILTER_STEP_SIZE        : signed(15 DOWNTO 0) :=
"0100000000000000";  -- sfix16_En17
 -- Signals
 SIGNAL enb                        : std_logic;
```

```
SIGNAL x_k_signed                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL d_k_signed                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL LMS_Filter_out1           : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL LMS_Filter_out2           : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL weight                    : vector_of_signed16(0 TO 19);  -- sfix16_En14 [20]
SIGNAL filter_sum                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL data_pipeline             : vector_of_signed16(0 TO 19);  -- sfix16_En14 [20]
SIGNAL data_pipeline_tmp         : vector_of_signed16(0 TO 18);  -- sfix16_En14 [19]
SIGNAL filter_products           : vector_of_signed16(0 TO 19);  -- sfix16_En14 [20]
SIGNAL mul_temp                  : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_1                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_2                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_3                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_4                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_5                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_6                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_7                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_8                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_9                : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_10               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_11               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_12               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_13               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_14               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_15               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_16               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_17               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_18               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_19               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL sum_1                     : signed(15 DOWNTO 0);  -- sfix16_En14
```

```
SIGNAL add_cast          : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_1        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp          : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_2             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_2        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_3        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_1        : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_3             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_4        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_5        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_2        : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_4             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_6        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_7        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_3        : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_5             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_8        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_9        : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_4        : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_6             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_10       : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_11       : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_5        : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_7             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_12       : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_13       : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_6        : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_8             : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_14       : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_15       : signed(15 DOWNTO 0);  -- sfix16_En14
```

```
SIGNAL add_temp_7                 : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_9                      : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_16                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_17                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_8                 : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_10                     : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_18                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_19                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_9                 : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_11                     : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_20                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_21                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_10                : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_12                     : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_22                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_23                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_11                : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_13                     : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_24                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_25                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_12                : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_14                     : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_26                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_27                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_13                : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_15                     : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_28                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_29                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_14                : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_16                     : signed(15 DOWNTO 0);  -- sfix16_En14
```

```
SIGNAL add_cast_30               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_31               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_15               : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_17                    : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_32               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_33               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_16               : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sum_18                    : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_34               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_35               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_17               : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_36               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_37               : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_18               : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL sub_cast                  : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL sub_cast_1                : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL sub_temp                  : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL mu_err                    : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL mul_temp_20               : signed(31 DOWNTO 0);  -- sfix32_En31
SIGNAL mu_err_data_prod          : vector_of_signed16(0 TO 19);  -- sfix16_En14 [20]
SIGNAL mul_temp_21               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_22               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_23               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_24               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_25               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_26               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_27               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_28               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_29               : signed(31 DOWNTO 0);  -- sfix32_En28
SIGNAL mul_temp_30               : signed(31 DOWNTO 0);  -- sfix32_En28
```

```
  SIGNAL mul_temp_31                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_32                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_33                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_34                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_35                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_36                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_37                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_38                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_39                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL mul_temp_40                : signed(31 DOWNTO 0);  -- sfix32_En28
  SIGNAL weight_adder_output          : vector_of_signed16(0 TO 19);  -- sfix16_En14
[20]
  SIGNAL add_cast_38              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_39              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_19                : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_40              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_41              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_20                : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_42              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_43              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_21                : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_44              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_45              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_22                : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_46              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_47              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_23                : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_48              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_49              : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_24                : signed(16 DOWNTO 0);  -- sfix17_En14
```

```
SIGNAL add_cast_50              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_51              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_25              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_52              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_53              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_26              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_54              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_55              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_27              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_56              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_57              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_28              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_58              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_59              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_29              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_60              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_61              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_30              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_62              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_63              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_31              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_64              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_65              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_32              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_66              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_67              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_33              : signed(16 DOWNTO 0);  -- sfix17_En14
SIGNAL add_cast_68              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_cast_69              : signed(15 DOWNTO 0);  -- sfix16_En14
SIGNAL add_temp_34              : signed(16 DOWNTO 0);  -- sfix17_En14
```

```
  SIGNAL add_cast_70               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_71               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_35               : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_72               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_73               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_36               : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_74               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_75               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_37               : signed(16 DOWNTO 0);  -- sfix17_En14
  SIGNAL add_cast_76               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_cast_77               : signed(15 DOWNTO 0);  -- sfix16_En14
  SIGNAL add_temp_38               : signed(16 DOWNTO 0);  -- sfix17_En14


BEGIN
  x_k_signed <= signed(x_k);
  d_k_signed <= signed(d_k);
  enb <= clk_enable;

-- ************************
-- ********* LMS *********
-- ************************


-- * LMS: FIR section


  LMS_Filter_del_temp_process1 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      data_pipeline_tmp <= (OTHERS => (OTHERS => '0'));
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
```

```
      data_pipeline_tmp(0 TO 17) <= data_pipeline_tmp(1 TO 18);
      data_pipeline_tmp(18) <= x_k_signed;
    END IF;
  END IF;
 END PROCESS LMS_Filter_del_temp_process1;

 data_pipeline(0 TO 18) <= data_pipeline_tmp(0 TO 18);
 data_pipeline(19) <= x_k_signed;

 mul_temp <= data_pipeline(0) * weight(0);
 filter_products(0) <= resize(shift_right(mul_temp(29 DOWNTO 0) + ( "0" &
(mul_temp(14) & NOT mul_temp(14) & NOT mul_temp(14) & NOT mul_temp(14) & NOT
mul_temp(14) & NOT mul_temp(14) & NOT mul_temp(14) & NOT mul_temp(14) & NOT
mul_temp(14) & NOT mul_temp(14) & NOT mul_temp(14) & NOT mul_temp(14) & NOT
mul_temp(14) & NOT mul_temp(14))), 14), 16);

 mul_temp_1 <= data_pipeline(1) * weight(1);
 filter_products(1) <= resize(shift_right(mul_temp_1(29 DOWNTO 0) + ( "0" &
(mul_temp_1(14) & NOT mul_temp_1(14) & NOT mul_temp_1(14) & NOT
mul_temp_1(14) & NOT mul_temp_1(14) & NOT mul_temp_1(14) & NOT
mul_temp_1(14) & NOT mul_temp_1(14) & NOT mul_temp_1(14) & NOT
mul_temp_1(14) & NOT mul_temp_1(14) & NOT mul_temp_1(14) & NOT
mul_temp_1(14) & NOT mul_temp_1(14))), 14), 16);

 mul_temp_2 <= data_pipeline(2) * weight(2);
 filter_products(2) <= resize(shift_right(mul_temp_2(29 DOWNTO 0) + ( "0" &
(mul_temp_2(14) & NOT mul_temp_2(14) & NOT mul_temp_2(14) & NOT
mul_temp_2(14) & NOT mul_temp_2(14) & NOT mul_temp_2(14) & NOT
mul_temp_2(14) & NOT mul_temp_2(14) & NOT mul_temp_2(14) & NOT
mul_temp_2(14) & NOT mul_temp_2(14) & NOT mul_temp_2(14) & NOT
mul_temp_2(14) & NOT mul_temp_2(14))), 14), 16);

 mul_temp_3 <= data_pipeline(3) * weight(3);
 filter_products(3) <= resize(shift_right(mul_temp_3(29 DOWNTO 0) + ( "0" &
(mul_temp_3(14) & NOT mul_temp_3(14) & NOT mul_temp_3(14) & NOT
```

```
mul_temp_3(14) & NOT mul_temp_3(14) & NOT mul_temp_3(14) & NOT
mul_temp_3(14) & NOT mul_temp_3(14) & NOT mul_temp_3(14) & NOT
mul_temp_3(14) & NOT mul_temp_3(14) & NOT mul_temp_3(14) & NOT
mul_temp_3(14) & NOT mul_temp_3(14))), 14), 16);


 mul_temp_4 <= data_pipeline(4) * weight(4);

 filter_products(4) <= resize(shift_right(mul_temp_4(29 DOWNTO 0) + ( "0" &
(mul_temp_4(14) & NOT mul_temp_4(14) & NOT mul_temp_4(14) & NOT
mul_temp_4(14) & NOT mul_temp_4(14) & NOT mul_temp_4(14) & NOT
mul_temp_4(14) & NOT mul_temp_4(14) & NOT mul_temp_4(14) & NOT
mul_temp_4(14) & NOT mul_temp_4(14) & NOT mul_temp_4(14) & NOT
mul_temp_4(14) & NOT mul_temp_4(14))), 14), 16);


 mul_temp_5 <= data_pipeline(5) * weight(5);

 filter_products(5) <= resize(shift_right(mul_temp_5(29 DOWNTO 0) + ( "0" &
(mul_temp_5(14) & NOT mul_temp_5(14) & NOT mul_temp_5(14) & NOT
mul_temp_5(14) & NOT mul_temp_5(14) & NOT mul_temp_5(14) & NOT
mul_temp_5(14) & NOT mul_temp_5(14) & NOT mul_temp_5(14) & NOT
mul_temp_5(14) & NOT mul_temp_5(14) & NOT mul_temp_5(14) & NOT
mul_temp_5(14) & NOT mul_temp_5(14))), 14), 16);


 mul_temp_6 <= data_pipeline(6) * weight(6);

 filter_products(6) <= resize(shift_right(mul_temp_6(29 DOWNTO 0) + ( "0" &
(mul_temp_6(14) & NOT mul_temp_6(14) & NOT mul_temp_6(14) & NOT
mul_temp_6(14) & NOT mul_temp_6(14) & NOT mul_temp_6(14) & NOT
mul_temp_6(14) & NOT mul_temp_6(14) & NOT mul_temp_6(14) & NOT
mul_temp_6(14) & NOT mul_temp_6(14) & NOT mul_temp_6(14) & NOT
mul_temp_6(14) & NOT mul_temp_6(14))), 14), 16);


 mul_temp_7 <= data_pipeline(7) * weight(7);

 filter_products(7) <= resize(shift_right(mul_temp_7(29 DOWNTO 0) + ( "0" &
(mul_temp_7(14) & NOT mul_temp_7(14) & NOT mul_temp_7(14) & NOT
mul_temp_7(14) & NOT mul_temp_7(14) & NOT mul_temp_7(14) & NOT
mul_temp_7(14) & NOT mul_temp_7(14) & NOT mul_temp_7(14) & NOT
mul_temp_7(14) & NOT mul_temp_7(14) & NOT mul_temp_7(14) & NOT
mul_temp_7(14) & NOT mul_temp_7(14))), 14), 16);


 mul_temp_8 <= data_pipeline(8) * weight(8);
```

```
 filter_products(8) <= resize(shift_right(mul_temp_8(29 DOWNTO 0) + ( "0" &
(mul_temp_8(14) & NOT mul_temp_8(14) & NOT mul_temp_8(14) & NOT
mul_temp_8(14) & NOT mul_temp_8(14) & NOT mul_temp_8(14) & NOT
mul_temp_8(14) & NOT mul_temp_8(14) & NOT mul_temp_8(14) & NOT
mul_temp_8(14) & NOT mul_temp_8(14) & NOT mul_temp_8(14) & NOT
mul_temp_8(14) & NOT mul_temp_8(14))), 14), 16);


 mul_temp_9 <= data_pipeline(9) * weight(9);

 filter_products(9) <= resize(shift_right(mul_temp_9(29 DOWNTO 0) + ( "0" &
(mul_temp_9(14) & NOT mul_temp_9(14) & NOT mul_temp_9(14) & NOT
mul_temp_9(14) & NOT mul_temp_9(14) & NOT mul_temp_9(14) & NOT
mul_temp_9(14) & NOT mul_temp_9(14) & NOT mul_temp_9(14) & NOT
mul_temp_9(14) & NOT mul_temp_9(14) & NOT mul_temp_9(14) & NOT
mul_temp_9(14) & NOT mul_temp_9(14))), 14), 16);


 mul_temp_10 <= data_pipeline(10) * weight(10);

 filter_products(10) <= resize(shift_right(mul_temp_10(29 DOWNTO 0) + ( "0" &
(mul_temp_10(14) & NOT mul_temp_10(14) & NOT mul_temp_10(14) & NOT
mul_temp_10(14) & NOT mul_temp_10(14) & NOT mul_temp_10(14) & NOT
mul_temp_10(14) & NOT mul_temp_10(14) & NOT mul_temp_10(14) & NOT
mul_temp_10(14) & NOT mul_temp_10(14) & NOT mul_temp_10(14) & NOT
mul_temp_10(14) & NOT mul_temp_10(14))), 14), 16);


 mul_temp_11 <= data_pipeline(11) * weight(11);

 filter_products(11) <= resize(shift_right(mul_temp_11(29 DOWNTO 0) + ( "0" &
(mul_temp_11(14) & NOT mul_temp_11(14) & NOT mul_temp_11(14) & NOT
mul_temp_11(14) & NOT mul_temp_11(14) & NOT mul_temp_11(14) & NOT
mul_temp_11(14) & NOT mul_temp_11(14) & NOT mul_temp_11(14) & NOT
mul_temp_11(14) & NOT mul_temp_11(14) & NOT mul_temp_11(14) & NOT
mul_temp_11(14) & NOT mul_temp_11(14))), 14), 16);


 mul_temp_12 <= data_pipeline(12) * weight(12);

 filter_products(12) <= resize(shift_right(mul_temp_12(29 DOWNTO 0) + ( "0" &
(mul_temp_12(14) & NOT mul_temp_12(14) & NOT mul_temp_12(14) & NOT
mul_temp_12(14) & NOT mul_temp_12(14) & NOT mul_temp_12(14) & NOT
mul_temp_12(14) & NOT mul_temp_12(14) & NOT mul_temp_12(14) & NOT
mul_temp_12(14) & NOT mul_temp_12(14) & NOT mul_temp_12(14) & NOT
mul_temp_12(14) & NOT mul_temp_12(14))), 14), 16);
```

```
 mul_temp_13 <= data_pipeline(13) * weight(13);

 filter_products(13) <= resize(shift_right(mul_temp_13(29 DOWNTO 0) + ( "0" &
(mul_temp_13(14) & NOT mul_temp_13(14) & NOT mul_temp_13(14) & NOT
mul_temp_13(14) & NOT mul_temp_13(14) & NOT mul_temp_13(14) & NOT
mul_temp_13(14) & NOT mul_temp_13(14) & NOT mul_temp_13(14) & NOT
mul_temp_13(14) & NOT mul_temp_13(14) & NOT mul_temp_13(14) & NOT
mul_temp_13(14) & NOT mul_temp_13(14) & NOT mul_temp_13(14) & NOT
mul_temp_13(14) & NOT mul_temp_13(14))), 14), 16);


 mul_temp_14 <= data_pipeline(14) * weight(14);

 filter_products(14) <= resize(shift_right(mul_temp_14(29 DOWNTO 0) + ( "0" &
(mul_temp_14(14) & NOT mul_temp_14(14) & NOT mul_temp_14(14) & NOT
mul_temp_14(14) & NOT mul_temp_14(14) & NOT mul_temp_14(14) & NOT
mul_temp_14(14) & NOT mul_temp_14(14) & NOT mul_temp_14(14) & NOT
mul_temp_14(14) & NOT mul_temp_14(14) & NOT mul_temp_14(14) & NOT
mul_temp_14(14) & NOT mul_temp_14(14) & NOT mul_temp_14(14) & NOT
mul_temp_14(14) & NOT mul_temp_14(14))), 14), 16);


 mul_temp_15 <= data_pipeline(15) * weight(15);

 filter_products(15) <= resize(shift_right(mul_temp_15(29 DOWNTO 0) + ( "0" &
(mul_temp_15(14) & NOT mul_temp_15(14) & NOT mul_temp_15(14) & NOT
mul_temp_15(14) & NOT mul_temp_15(14) & NOT mul_temp_15(14) & NOT
mul_temp_15(14) & NOT mul_temp_15(14) & NOT mul_temp_15(14) & NOT
mul_temp_15(14) & NOT mul_temp_15(14) & NOT mul_temp_15(14) & NOT
mul_temp_15(14) & NOT mul_temp_15(14) & NOT mul_temp_15(14) & NOT
mul_temp_15(14) & NOT mul_temp_15(14))), 14), 16);


 mul_temp_16 <= data_pipeline(16) * weight(16);

 filter_products(16) <= resize(shift_right(mul_temp_16(29 DOWNTO 0) + ( "0" &
(mul_temp_16(14) & NOT mul_temp_16(14) & NOT mul_temp_16(14) & NOT
mul_temp_16(14) & NOT mul_temp_16(14) & NOT mul_temp_16(14) & NOT
mul_temp_16(14) & NOT mul_temp_16(14) & NOT mul_temp_16(14) & NOT
mul_temp_16(14) & NOT mul_temp_16(14) & NOT mul_temp_16(14) & NOT
mul_temp_16(14) & NOT mul_temp_16(14) & NOT mul_temp_16(14) & NOT
mul_temp_16(14) & NOT mul_temp_16(14))), 14), 16);


 mul_temp_17 <= data_pipeline(17) * weight(17);

 filter_products(17) <= resize(shift_right(mul_temp_17(29 DOWNTO 0) + ( "0" &
(mul_temp_17(14) & NOT mul_temp_17(14) & NOT mul_temp_17(14) & NOT
mul_temp_17(14) & NOT mul_temp_17(14) & NOT mul_temp_17(14) & NOT
mul_temp_17(14) & NOT mul_temp_17(14) & NOT mul_temp_17(14) & NOT
mul_temp_17(14) & NOT mul_temp_17(14) & NOT mul_temp_17(14) & NOT
mul_temp_17(14) & NOT mul_temp_17(14) & NOT mul_temp_17(14) & NOT
mul_temp_17(14) & NOT mul_temp_17(14))), 14), 16);
```

```
  mul_temp_18 <= data_pipeline(18) * weight(18);

  filter_products(18) <= resize(shift_right(mul_temp_18(29 DOWNTO 0) + ( "0" &
(mul_temp_18(14) & NOT mul_temp_18(14) & NOT mul_temp_18(14) & NOT
mul_temp_18(14) & NOT mul_temp_18(14) & NOT mul_temp_18(14) & NOT
mul_temp_18(14) & NOT mul_temp_18(14) & NOT mul_temp_18(14) & NOT
mul_temp_18(14) & NOT mul_temp_18(14) & NOT mul_temp_18(14) & NOT
mul_temp_18(14) & NOT mul_temp_18(14))), 14), 16);


  mul_temp_19 <= data_pipeline(19) * weight(19);

  filter_products(19) <= resize(shift_right(mul_temp_19(29 DOWNTO 0) + ( "0" &
(mul_temp_19(14) & NOT mul_temp_19(14) & NOT mul_temp_19(14) & NOT
mul_temp_19(14) & NOT mul_temp_19(14) & NOT mul_temp_19(14) & NOT
mul_temp_19(14) & NOT mul_temp_19(14) & NOT mul_temp_19(14) & NOT
mul_temp_19(14) & NOT mul_temp_19(14) & NOT mul_temp_19(14) & NOT
mul_temp_19(14) & NOT mul_temp_19(14))), 14), 16);


-- linear sum of filter products


  add_cast <= filter_products(0);

  add_cast_1 <= filter_products(1);

  add_temp <= resize(add_cast, 17) + resize(add_cast_1, 17);

  sum_1 <= add_temp(15 DOWNTO 0);


  add_cast_2 <= sum_1;

  add_cast_3 <= filter_products(2);

  add_temp_1 <= resize(add_cast_2, 17) + resize(add_cast_3, 17);

  sum_2 <= add_temp_1(15 DOWNTO 0);


  add_cast_4 <= sum_2;

  add_cast_5 <= filter_products(3);

  add_temp_2 <= resize(add_cast_4, 17) + resize(add_cast_5, 17);

  sum_3 <= add_temp_2(15 DOWNTO 0);
```

```
add_cast_6 <= sum_3;
add_cast_7 <= filter_products(4);
add_temp_3 <= resize(add_cast_6, 17) + resize(add_cast_7, 17);
sum_4 <= add_temp_3(15 DOWNTO 0);

add_cast_8 <= sum_4;
add_cast_9 <= filter_products(5);
add_temp_4 <= resize(add_cast_8, 17) + resize(add_cast_9, 17);
sum_5 <= add_temp_4(15 DOWNTO 0);

add_cast_10 <= sum_5;
add_cast_11 <= filter_products(6);
add_temp_5 <= resize(add_cast_10, 17) + resize(add_cast_11, 17);
sum_6 <= add_temp_5(15 DOWNTO 0);

add_cast_12 <= sum_6;
add_cast_13 <= filter_products(7);
add_temp_6 <= resize(add_cast_12, 17) + resize(add_cast_13, 17);
sum_7 <= add_temp_6(15 DOWNTO 0);

add_cast_14 <= sum_7;
add_cast_15 <= filter_products(8);
add_temp_7 <= resize(add_cast_14, 17) + resize(add_cast_15, 17);
sum_8 <= add_temp_7(15 DOWNTO 0);

add_cast_16 <= sum_8;
add_cast_17 <= filter_products(9);
add_temp_8 <= resize(add_cast_16, 17) + resize(add_cast_17, 17);
sum_9 <= add_temp_8(15 DOWNTO 0);
```

```
add_cast_18 <= sum_9;
add_cast_19 <= filter_products(10);
add_temp_9 <= resize(add_cast_18, 17) + resize(add_cast_19, 17);
sum_10 <= add_temp_9(15 DOWNTO 0);

add_cast_20 <= sum_10;
add_cast_21 <= filter_products(11);
add_temp_10 <= resize(add_cast_20, 17) + resize(add_cast_21, 17);
sum_11 <= add_temp_10(15 DOWNTO 0);

add_cast_22 <= sum_11;
add_cast_23 <= filter_products(12);
add_temp_11 <= resize(add_cast_22, 17) + resize(add_cast_23, 17);
sum_12 <= add_temp_11(15 DOWNTO 0);

add_cast_24 <= sum_12;
add_cast_25 <= filter_products(13);
add_temp_12 <= resize(add_cast_24, 17) + resize(add_cast_25, 17);
sum_13 <= add_temp_12(15 DOWNTO 0);

add_cast_26 <= sum_13;
add_cast_27 <= filter_products(14);
add_temp_13 <= resize(add_cast_26, 17) + resize(add_cast_27, 17);
sum_14 <= add_temp_13(15 DOWNTO 0);

add_cast_28 <= sum_14;
add_cast_29 <= filter_products(15);
add_temp_14 <= resize(add_cast_28, 17) + resize(add_cast_29, 17);
sum_15 <= add_temp_14(15 DOWNTO 0);
```

```
  add_cast_30 <= sum_15;
  add_cast_31 <= filter_products(16);
  add_temp_15 <= resize(add_cast_30, 17) + resize(add_cast_31, 17);
  sum_16 <= add_temp_15(15 DOWNTO 0);

  add_cast_32 <= sum_16;
  add_cast_33 <= filter_products(17);
  add_temp_16 <= resize(add_cast_32, 17) + resize(add_cast_33, 17);
  sum_17 <= add_temp_16(15 DOWNTO 0);

  add_cast_34 <= sum_17;
  add_cast_35 <= filter_products(18);
  add_temp_17 <= resize(add_cast_34, 17) + resize(add_cast_35, 17);
  sum_18 <= add_temp_17(15 DOWNTO 0);

  add_cast_36 <= sum_18;
  add_cast_37 <= filter_products(19);
  add_temp_18 <= resize(add_cast_36, 17) + resize(add_cast_37, 17);
  filter_sum <= add_temp_18(15 DOWNTO 0);

  LMS_Filter_out1 <= filter_sum;

-- * Calculate Filter Error

  sub_cast <= d_k_signed;
  sub_cast_1 <= filter_sum;
  sub_temp <= resize(sub_cast, 17) - resize(sub_cast_1, 17);
  LMS_Filter_out2 <= sub_temp(15 DOWNTO 0);
```

```
-- ***** LMS Weight Update Function *****

  mul_temp_20 <= C_LMS_FILTER_STEP_SIZE * LMS_Filter_out2;

  mu_err <= resize(shift_right(mul_temp_20(31) & mul_temp_20(31 DOWNTO 0) + ( "0"
& (mul_temp_20(17) & NOT mul_temp_20(17) & NOT mul_temp_20(17) & NOT
mul_temp_20(17) & NOT mul_temp_20(17) & NOT mul_temp_20(17) & NOT
mul_temp_20(17) & NOT mul_temp_20(17) & NOT mul_temp_20(17) & NOT
mul_temp_20(17) & NOT mul_temp_20(17) & NOT mul_temp_20(17) & NOT
mul_temp_20(17) & NOT mul_temp_20(17) & NOT mul_temp_20(17) & NOT
mul_temp_20(17) & NOT mul_temp_20(17))), 17), 16);


  mul_temp_21 <= data_pipeline(0) * mu_err;

  mu_err_data_prod(0) <= resize(shift_right(mul_temp_21(29 DOWNTO 0) + ( "0" &
(mul_temp_21(14) & NOT mul_temp_21(14) & NOT mul_temp_21(14) & NOT
mul_temp_21(14) & NOT mul_temp_21(14) & NOT mul_temp_21(14) & NOT
mul_temp_21(14) & NOT mul_temp_21(14) & NOT mul_temp_21(14) & NOT
mul_temp_21(14) & NOT mul_temp_21(14) & NOT mul_temp_21(14) & NOT
mul_temp_21(14) & NOT mul_temp_21(14))), 14), 16);


  mul_temp_22 <= data_pipeline(1) * mu_err;

  mu_err_data_prod(1) <= resize(shift_right(mul_temp_22(29 DOWNTO 0) + ( "0" &
(mul_temp_22(14) & NOT mul_temp_22(14) & NOT mul_temp_22(14) & NOT
mul_temp_22(14) & NOT mul_temp_22(14) & NOT mul_temp_22(14) & NOT
mul_temp_22(14) & NOT mul_temp_22(14) & NOT mul_temp_22(14) & NOT
mul_temp_22(14) & NOT mul_temp_22(14) & NOT mul_temp_22(14) & NOT
mul_temp_22(14) & NOT mul_temp_22(14))), 14), 16);


  mul_temp_23 <= data_pipeline(2) * mu_err;

  mu_err_data_prod(2) <= resize(shift_right(mul_temp_23(29 DOWNTO 0) + ( "0" &
(mul_temp_23(14) & NOT mul_temp_23(14) & NOT mul_temp_23(14) & NOT
mul_temp_23(14) & NOT mul_temp_23(14) & NOT mul_temp_23(14) & NOT
mul_temp_23(14) & NOT mul_temp_23(14) & NOT mul_temp_23(14) & NOT
mul_temp_23(14) & NOT mul_temp_23(14) & NOT mul_temp_23(14) & NOT
mul_temp_23(14) & NOT mul_temp_23(14))), 14), 16);


  mul_temp_24 <= data_pipeline(3) * mu_err;

  mu_err_data_prod(3) <= resize(shift_right(mul_temp_24(29 DOWNTO 0) + ( "0" &
(mul_temp_24(14) & NOT mul_temp_24(14) & NOT mul_temp_24(14) & NOT
```

```
mul_temp_24(14) & NOT mul_temp_24(14) & NOT mul_temp_24(14) & NOT
mul_temp_24(14) & NOT mul_temp_24(14) & NOT mul_temp_24(14) & NOT
mul_temp_24(14) & NOT mul_temp_24(14) & NOT mul_temp_24(14) & NOT
mul_temp_24(14) & NOT mul_temp_24(14))), 14), 16);


  mul_temp_25 <= data_pipeline(4) * mu_err;

  mu_err_data_prod(4) <= resize(shift_right(mul_temp_25(29 DOWNTO 0) + ( "0" &
(mul_temp_25(14) & NOT mul_temp_25(14) & NOT mul_temp_25(14) & NOT
mul_temp_25(14) & NOT mul_temp_25(14) & NOT mul_temp_25(14) & NOT
mul_temp_25(14) & NOT mul_temp_25(14) & NOT mul_temp_25(14) & NOT
mul_temp_25(14) & NOT mul_temp_25(14) & NOT mul_temp_25(14) & NOT
mul_temp_25(14) & NOT mul_temp_25(14))), 14), 16);


  mul_temp_26 <= data_pipeline(5) * mu_err;

  mu_err_data_prod(5) <= resize(shift_right(mul_temp_26(29 DOWNTO 0) + ( "0" &
(mul_temp_26(14) & NOT mul_temp_26(14) & NOT mul_temp_26(14) & NOT
mul_temp_26(14) & NOT mul_temp_26(14) & NOT mul_temp_26(14) & NOT
mul_temp_26(14) & NOT mul_temp_26(14) & NOT mul_temp_26(14) & NOT
mul_temp_26(14) & NOT mul_temp_26(14) & NOT mul_temp_26(14) & NOT
mul_temp_26(14) & NOT mul_temp_26(14))), 14), 16);


  mul_temp_27 <= data_pipeline(6) * mu_err;

  mu_err_data_prod(6) <= resize(shift_right(mul_temp_27(29 DOWNTO 0) + ( "0" &
(mul_temp_27(14) & NOT mul_temp_27(14) & NOT mul_temp_27(14) & NOT
mul_temp_27(14) & NOT mul_temp_27(14) & NOT mul_temp_27(14) & NOT
mul_temp_27(14) & NOT mul_temp_27(14) & NOT mul_temp_27(14) & NOT
mul_temp_27(14) & NOT mul_temp_27(14) & NOT mul_temp_27(14) & NOT
mul_temp_27(14) & NOT mul_temp_27(14))), 14), 16);


  mul_temp_28 <= data_pipeline(7) * mu_err;

  mu_err_data_prod(7) <= resize(shift_right(mul_temp_28(29 DOWNTO 0) + ( "0" &
(mul_temp_28(14) & NOT mul_temp_28(14) & NOT mul_temp_28(14) & NOT
mul_temp_28(14) & NOT mul_temp_28(14) & NOT mul_temp_28(14) & NOT
mul_temp_28(14) & NOT mul_temp_28(14) & NOT mul_temp_28(14) & NOT
mul_temp_28(14) & NOT mul_temp_28(14) & NOT mul_temp_28(14) & NOT
mul_temp_28(14) & NOT mul_temp_28(14))), 14), 16);


  mul_temp_29 <= data_pipeline(8) * mu_err;
```

```
 mu_err_data_prod(8) <= resize(shift_right(mul_temp_29(29 DOWNTO 0) + ( "0" &
(mul_temp_29(14) & NOT mul_temp_29(14) & NOT mul_temp_29(14) & NOT
mul_temp_29(14) & NOT mul_temp_29(14) & NOT mul_temp_29(14) & NOT
mul_temp_29(14) & NOT mul_temp_29(14) & NOT mul_temp_29(14) & NOT
mul_temp_29(14) & NOT mul_temp_29(14) & NOT mul_temp_29(14) & NOT
mul_temp_29(14) & NOT mul_temp_29(14))), 14), 16);


 mul_temp_30 <= data_pipeline(9) * mu_err;

 mu_err_data_prod(9) <= resize(shift_right(mul_temp_30(29 DOWNTO 0) + ( "0" &
(mul_temp_30(14) & NOT mul_temp_30(14) & NOT mul_temp_30(14) & NOT
mul_temp_30(14) & NOT mul_temp_30(14) & NOT mul_temp_30(14) & NOT
mul_temp_30(14) & NOT mul_temp_30(14) & NOT mul_temp_30(14) & NOT
mul_temp_30(14) & NOT mul_temp_30(14) & NOT mul_temp_30(14) & NOT
mul_temp_30(14) & NOT mul_temp_30(14))), 14), 16);


 mul_temp_31 <= data_pipeline(10) * mu_err;

 mu_err_data_prod(10) <= resize(shift_right(mul_temp_31(29 DOWNTO 0) + ( "0" &
(mul_temp_31(14) & NOT mul_temp_31(14) & NOT mul_temp_31(14) & NOT
mul_temp_31(14) & NOT mul_temp_31(14) & NOT mul_temp_31(14) & NOT
mul_temp_31(14) & NOT mul_temp_31(14) & NOT mul_temp_31(14) & NOT
mul_temp_31(14) & NOT mul_temp_31(14) & NOT mul_temp_31(14) & NOT
mul_temp_31(14) & NOT mul_temp_31(14))), 14), 16);


 mul_temp_32 <= data_pipeline(11) * mu_err;

 mu_err_data_prod(11) <= resize(shift_right(mul_temp_32(29 DOWNTO 0) + ( "0" &
(mul_temp_32(14) & NOT mul_temp_32(14) & NOT mul_temp_32(14) & NOT
mul_temp_32(14) & NOT mul_temp_32(14) & NOT mul_temp_32(14) & NOT
mul_temp_32(14) & NOT mul_temp_32(14) & NOT mul_temp_32(14) & NOT
mul_temp_32(14) & NOT mul_temp_32(14) & NOT mul_temp_32(14) & NOT
mul_temp_32(14) & NOT mul_temp_32(14))), 14), 16);


 mul_temp_33 <= data_pipeline(12) * mu_err;

 mu_err_data_prod(12) <= resize(shift_right(mul_temp_33(29 DOWNTO 0) + ( "0" &
(mul_temp_33(14) & NOT mul_temp_33(14) & NOT mul_temp_33(14) & NOT
mul_temp_33(14) & NOT mul_temp_33(14) & NOT mul_temp_33(14) & NOT
mul_temp_33(14) & NOT mul_temp_33(14) & NOT mul_temp_33(14) & NOT
mul_temp_33(14) & NOT mul_temp_33(14) & NOT mul_temp_33(14) & NOT
mul_temp_33(14) & NOT mul_temp_33(14))), 14), 16);
```

```
  mul_temp_34 <= data_pipeline(13) * mu_err;

  mu_err_data_prod(13) <= resize(shift_right(mul_temp_34(29 DOWNTO 0) + ( "0" &
(mul_temp_34(14) & NOT mul_temp_34(14) & NOT mul_temp_34(14) & NOT
mul_temp_34(14) & NOT mul_temp_34(14) & NOT mul_temp_34(14) & NOT
mul_temp_34(14) & NOT mul_temp_34(14) & NOT mul_temp_34(14) & NOT
mul_temp_34(14) & NOT mul_temp_34(14) & NOT mul_temp_34(14) & NOT
mul_temp_34(14) & NOT mul_temp_34(14) & NOT mul_temp_34(14) & NOT
mul_temp_34(14) & NOT mul_temp_34(14))), 14), 16);


  mul_temp_35 <= data_pipeline(14) * mu_err;

  mu_err_data_prod(14) <= resize(shift_right(mul_temp_35(29 DOWNTO 0) + ( "0" &
(mul_temp_35(14) & NOT mul_temp_35(14) & NOT mul_temp_35(14) & NOT
mul_temp_35(14) & NOT mul_temp_35(14) & NOT mul_temp_35(14) & NOT
mul_temp_35(14) & NOT mul_temp_35(14) & NOT mul_temp_35(14) & NOT
mul_temp_35(14) & NOT mul_temp_35(14) & NOT mul_temp_35(14) & NOT
mul_temp_35(14) & NOT mul_temp_35(14) & NOT mul_temp_35(14) & NOT
mul_temp_35(14) & NOT mul_temp_35(14))), 14), 16);


  mul_temp_36 <= data_pipeline(15) * mu_err;

  mu_err_data_prod(15) <= resize(shift_right(mul_temp_36(29 DOWNTO 0) + ( "0" &
(mul_temp_36(14) & NOT mul_temp_36(14) & NOT mul_temp_36(14) & NOT
mul_temp_36(14) & NOT mul_temp_36(14) & NOT mul_temp_36(14) & NOT
mul_temp_36(14) & NOT mul_temp_36(14) & NOT mul_temp_36(14) & NOT
mul_temp_36(14) & NOT mul_temp_36(14) & NOT mul_temp_36(14) & NOT
mul_temp_36(14) & NOT mul_temp_36(14) & NOT mul_temp_36(14) & NOT
mul_temp_36(14) & NOT mul_temp_36(14))), 14), 16);


  mul_temp_37 <= data_pipeline(16) * mu_err;

  mu_err_data_prod(16) <= resize(shift_right(mul_temp_37(29 DOWNTO 0) + ( "0" &
(mul_temp_37(14) & NOT mul_temp_37(14) & NOT mul_temp_37(14) & NOT
mul_temp_37(14) & NOT mul_temp_37(14) & NOT mul_temp_37(14) & NOT
mul_temp_37(14) & NOT mul_temp_37(14) & NOT mul_temp_37(14) & NOT
mul_temp_37(14) & NOT mul_temp_37(14) & NOT mul_temp_37(14) & NOT
mul_temp_37(14) & NOT mul_temp_37(14) & NOT mul_temp_37(14) & NOT
mul_temp_37(14) & NOT mul_temp_37(14))), 14), 16);


  mul_temp_38 <= data_pipeline(17) * mu_err;

  mu_err_data_prod(17) <= resize(shift_right(mul_temp_38(29 DOWNTO 0) + ( "0" &
(mul_temp_38(14) & NOT mul_temp_38(14) & NOT mul_temp_38(14) & NOT
mul_temp_38(14) & NOT mul_temp_38(14) & NOT mul_temp_38(14) & NOT
mul_temp_38(14) & NOT mul_temp_38(14) & NOT mul_temp_38(14) & NOT
mul_temp_38(14) & NOT mul_temp_38(14) & NOT mul_temp_38(14) & NOT
mul_temp_38(14) & NOT mul_temp_38(14) & NOT mul_temp_38(14) & NOT
mul_temp_38(14) & NOT mul_temp_38(14))), 14), 16);
```

```
  mul_temp_39 <= data_pipeline(18) * mu_err;

  mu_err_data_prod(18) <= resize(shift_right(mul_temp_39(29 DOWNTO 0) + ( "0" &
(mul_temp_39(14) & NOT mul_temp_39(14) & NOT mul_temp_39(14) & NOT
mul_temp_39(14) & NOT mul_temp_39(14) & NOT mul_temp_39(14) & NOT
mul_temp_39(14) & NOT mul_temp_39(14) & NOT mul_temp_39(14) & NOT
mul_temp_39(14) & NOT mul_temp_39(14) & NOT mul_temp_39(14) & NOT
mul_temp_39(14) & NOT mul_temp_39(14))), 14), 16);


  mul_temp_40 <= data_pipeline(19) * mu_err;

  mu_err_data_prod(19) <= resize(shift_right(mul_temp_40(29 DOWNTO 0) + ( "0" &
(mul_temp_40(14) & NOT mul_temp_40(14) & NOT mul_temp_40(14) & NOT
mul_temp_40(14) & NOT mul_temp_40(14) & NOT mul_temp_40(14) & NOT
mul_temp_40(14) & NOT mul_temp_40(14) & NOT mul_temp_40(14) & NOT
mul_temp_40(14) & NOT mul_temp_40(14) & NOT mul_temp_40(14) & NOT
mul_temp_40(14) & NOT mul_temp_40(14))), 14), 16);


-- * LMS_Filter Weight Accumulator


  add_cast_38 <= weight(0);

  add_cast_39 <= mu_err_data_prod(0);

  add_temp_19 <= resize(add_cast_38, 17) + resize(add_cast_39, 17);

  weight_adder_output(0) <= add_temp_19(15 DOWNTO 0);


  add_cast_40 <= weight(1);

  add_cast_41 <= mu_err_data_prod(1);

  add_temp_20 <= resize(add_cast_40, 17) + resize(add_cast_41, 17);

  weight_adder_output(1) <= add_temp_20(15 DOWNTO 0);


  add_cast_42 <= weight(2);

  add_cast_43 <= mu_err_data_prod(2);

  add_temp_21 <= resize(add_cast_42, 17) + resize(add_cast_43, 17);

  weight_adder_output(2) <= add_temp_21(15 DOWNTO 0);
```

```
add_cast_44 <= weight(3);
add_cast_45 <= mu_err_data_prod(3);
add_temp_22 <= resize(add_cast_44, 17) + resize(add_cast_45, 17);
weight_adder_output(3) <= add_temp_22(15 DOWNTO 0);

add_cast_46 <= weight(4);
add_cast_47 <= mu_err_data_prod(4);
add_temp_23 <= resize(add_cast_46, 17) + resize(add_cast_47, 17);
weight_adder_output(4) <= add_temp_23(15 DOWNTO 0);

add_cast_48 <= weight(5);
add_cast_49 <= mu_err_data_prod(5);
add_temp_24 <= resize(add_cast_48, 17) + resize(add_cast_49, 17);
weight_adder_output(5) <= add_temp_24(15 DOWNTO 0);

add_cast_50 <= weight(6);
add_cast_51 <= mu_err_data_prod(6);
add_temp_25 <= resize(add_cast_50, 17) + resize(add_cast_51, 17);
weight_adder_output(6) <= add_temp_25(15 DOWNTO 0);

add_cast_52 <= weight(7);
add_cast_53 <= mu_err_data_prod(7);
add_temp_26 <= resize(add_cast_52, 17) + resize(add_cast_53, 17);
weight_adder_output(7) <= add_temp_26(15 DOWNTO 0);

add_cast_54 <= weight(8);
add_cast_55 <= mu_err_data_prod(8);
add_temp_27 <= resize(add_cast_54, 17) + resize(add_cast_55, 17);
weight_adder_output(8) <= add_temp_27(15 DOWNTO 0);
```

```
add_cast_56 <= weight(9);
add_cast_57 <= mu_err_data_prod(9);
add_temp_28 <= resize(add_cast_56, 17) + resize(add_cast_57, 17);
weight_adder_output(9) <= add_temp_28(15 DOWNTO 0);

add_cast_58 <= weight(10);
add_cast_59 <= mu_err_data_prod(10);
add_temp_29 <= resize(add_cast_58, 17) + resize(add_cast_59, 17);
weight_adder_output(10) <= add_temp_29(15 DOWNTO 0);

add_cast_60 <= weight(11);
add_cast_61 <= mu_err_data_prod(11);
add_temp_30 <= resize(add_cast_60, 17) + resize(add_cast_61, 17);
weight_adder_output(11) <= add_temp_30(15 DOWNTO 0);

add_cast_62 <= weight(12);
add_cast_63 <= mu_err_data_prod(12);
add_temp_31 <= resize(add_cast_62, 17) + resize(add_cast_63, 17);
weight_adder_output(12) <= add_temp_31(15 DOWNTO 0);

add_cast_64 <= weight(13);
add_cast_65 <= mu_err_data_prod(13);
add_temp_32 <= resize(add_cast_64, 17) + resize(add_cast_65, 17);
weight_adder_output(13) <= add_temp_32(15 DOWNTO 0);

add_cast_66 <= weight(14);
add_cast_67 <= mu_err_data_prod(14);
add_temp_33 <= resize(add_cast_66, 17) + resize(add_cast_67, 17);
weight_adder_output(14) <= add_temp_33(15 DOWNTO 0);
```

```
add_cast_68 <= weight(15);
add_cast_69 <= mu_err_data_prod(15);
add_temp_34 <= resize(add_cast_68, 17) + resize(add_cast_69, 17);
weight_adder_output(15) <= add_temp_34(15 DOWNTO 0);

add_cast_70 <= weight(16);
add_cast_71 <= mu_err_data_prod(16);
add_temp_35 <= resize(add_cast_70, 17) + resize(add_cast_71, 17);
weight_adder_output(16) <= add_temp_35(15 DOWNTO 0);

add_cast_72 <= weight(17);
add_cast_73 <= mu_err_data_prod(17);
add_temp_36 <= resize(add_cast_72, 17) + resize(add_cast_73, 17);
weight_adder_output(17) <= add_temp_36(15 DOWNTO 0);

add_cast_74 <= weight(18);
add_cast_75 <= mu_err_data_prod(18);
add_temp_37 <= resize(add_cast_74, 17) + resize(add_cast_75, 17);
weight_adder_output(18) <= add_temp_37(15 DOWNTO 0);

add_cast_76 <= weight(19);
add_cast_77 <= mu_err_data_prod(19);
add_temp_38 <= resize(add_cast_76, 17) + resize(add_cast_77, 17);
weight_adder_output(19) <= add_temp_38(15 DOWNTO 0);

LMS_Filter_acc_temp_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    weight <= (OTHERS => (OTHERS => '0'));
```

```
  ELSIF clk'event AND clk = '1' THEN
    IF enb = '1' THEN
      weight(0 TO 19) <= weight_adder_output(0 TO 19);
    END IF;
  END IF;
 END PROCESS LMS_Filter_acc_temp_process2;

 e_k <= std_logic_vector(LMS_Filter_out2);
 ce_out <= clk_enable;


END rtl;
```

ANEXO 5 PROGRAMACIÓN DE LOS ARCHIVOS FUENTE DE EXTENSIÓN C Y H PARA PRUEBAS CON RUIDO INTERNO.

1. TESIS_RUIDO_INTERNO.H

```
#ifndef TESIS_RUIDO_INTERNO_H_
#define TESIS_RUIDO_INTERNO_H_

/* ------------------------------------------------------------ *
 *                         Header Files                         *
 * ------------------------------------------------------------ */
#include <stdio.h>
#include <xil_io.h>
#include <sleep.h>
#include "xiicps.h"
#include <xil_printf.h>
#include <xparameters.h>
#include "xgpio.h"
#include "xuartps.h"
#include "stdlib.h"

/* ------------------------------------------------------------ *
 *                  Custom IP Header Files                      *
 * ------------------------------------------------------------ */
#include "audio_ruido.h"
#include "lms_pcore_addr.h"
#include "xnco.h"

/* ------------------------------------------------------------ *
 *                  Prototype Functions                         *
 * ------------------------------------------------------------ */
void menu ();
void tonal_noise();
void audio_stream();
void lms_filter();
unsigned char gpio_init();
void nco_init(void *InstancePtr);

/* ------------------------------------------------------------ *
 *                  Redefinitions from xparameters.h            *
 * ------------------------------------------------------------ */
#define NCO_ID XPAR_NCO_0_DEVICE_ID

#define LMS_LOC XPAR_LMS_PCORE_0_BASEADDR
#define LMS_X LMS_LOC + x_k__Data_lms_pcore
#define LMS_D LMS_LOC + d_k__Data_lms_pcore
#define LMS_E LMS_LOC + e_k__Data_lms_pcore
#define LMS_STROBE LMS_LOC + IPCore_Strobe_lms_pcore

#define UART_BASEADDR XPAR_PS7_UART_1_BASEADDR

#define BUTTON_SWITCH_BASE XPAR_GPIO_1_BASEADDR
#define BUTTON_SWITCH_ID XPAR_GPIO_1_DEVICE_ID
#define AUDIO_ENABLE_ID XPAR_AXI_GPIO_0_DEVICE_ID
```

```
/* ------------------------------------------------------------ *
 *                  Define GPIO Channels                        *
 * ------------------------------------------------------------ */
#define BUTTON_CHANNEL 1
#define SWITCH_CHANNEL 2

/* ------------------------------------------------------------ *
 *                  Audio Scaling Factor                        *
 * ------------------------------------------------------------ */
#define SCALE 7

/* ------------------------------------------------------------ *
 *                        Global Variables                      *
 * ------------------------------------------------------------ */
XIicPs Iic;
XGpio Gpio;
XGpio Gpio_audio_enable;
XNco Nco;


#endif
```

2. TESIS_RUIDO_INTERNO.C

```
#include "tesis_ruido_interno.h"

/* ------------------------------------------------------------ *
 *                        main()                                *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
int main(void)
{
IicConfig(XPAR_XIICPS_0_DEVICE_ID);
xil_printf("Entrada Principal\r\n");

AudioPllConfig();
xil_printf("SSM2603 Configurado\r\n");

gpio_init();
nco_init(&Nco);
xil_printf("Perifericos GPIO y NCO Configurados\r\n");

xil_printf("\r\n");
xil_printf("      UNIVERSIDAD NACIONAL DE CHIMBORAZO   \r\n");
xil_printf("           FACULTAD DE INGENIERIA          \r\n");
xil_printf("CARRERA DE INGENIERIA EN ELECTRONICA Y
TELECOMUNICACIONES   \r\n");
xil_printf("\r\n");
xil_printf("\r\n");
xil_printf("       PROYECTO DE GRADUACION             \r\n");
xil_printf("DISENO E IMPLEMENTACION DE UN FILTRO CON ALGORITMO
ADAPTATIVO \r\n");
xil_printf("   EN FPGA PARA LA CANCELACION DE RUIDO    \r\n");
xil_printf("\r\n");
```

```
xil_printf("\r\n");
xil_printf("AUTORES: Alejandro Ayala y Nadezhda Cordova\r\n");
xil_printf("DIRECTOR: Ing. Fabian Gunsha\r\n");


menu();
    return 1;
}

/* ------------------------------------------------------------ *
 *                          menu()                              *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
void menu(){
u8 inp = 0x00;
u32 CntrlRegister;

XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 0);

CntrlRegister = XUartPs_ReadReg(UART_BASEADDR,
XUARTPS_CR_OFFSET);

XUartPs_WriteReg(UART_BASEADDR, XUARTPS_CR_OFFSET,
                    ((CntrlRegister & ~XUARTPS_CR_EN_DIS_MASK) |
                     XUARTPS_CR_TX_EN | XUARTPS_CR_RX_EN));

xil_printf("\r\n\r\n");
xil_printf("-PRUEBA DEL FILTRO CON RUIDO GENERADO INTERNAMENTE-
\r\n");
xil_printf("| Seleccione una opcion del siguiente menu:
|\r\n");
xil_printf("| Presione 'a' para escuchar el audio de entrada
|\r\n");
xil_printf("| Presione 'r' para agregar ruido al audio
|\r\n");
xil_printf("| Presione 'f' para la respuesta del filtro LMS
|\r\n");
xil_printf("-----------------------------------------------\r\n");


while (!XUartPs_IsReceiveData(UART_BASEADDR));
                    inp = XUartPs_ReadReg(UART_BASEADDR,
XUARTPS_FIFO_OFFSET);

switch(inp){
case 'a':
             xil_printf("SENAL DE AUDIO\r\n");
             xil_printf("Presione 'm' para regresar al menu
principal\r\n");
             XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 1);
             audio_stream();
             break;
      case 'r':
             xil_printf("ADICION DE LA COMPONENTE DE RUIDO A LA
SENAL DE AUDIO\r\n");
             xil_printf("Presione 'm' para regresar al menu
principal\r\n");
```

```
            XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 1);
            tonal_noise();
            break;
      case 'f':
            xil_printf("FILTRADO LMS\r\n");
            xil_printf("Presione 'm' para regresar al menu
principal\r\n");
            XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 1);
            lms_filter();
            break;
default:
      menu();
      break;
} // switch
} // menu()
```

3. LMS_PCORE_ADDR.H

```
#ifndef LMS_PCORE_H_
#define LMS_PCORE_H_

#define  IPCore_Reset_lms_pcore    0x0  //write 0x1 to bit 0 to
reset IP core
#define  IPCore_Enable_lms_pcore   0x4  //enabled (by default)
when bit 0 is 0x1
#define  IPCore_Strobe_lms_pcore   0x8  //write 1 to bit 0 after
write all input data
#define  IPCore_Ready_lms_pcore    0xC  //wait until bit 0 is 1
before read output data
#define  x_k__Data_lms_pcore       0x100  //data register for
port x(k)
#define  d_k__Data_lms_pcore       0x104  //data register for
port d(k)
#define  e_k__Data_lms_pcore       0x108  //data register for
port e(k)

#endif /* LMS_PCORE_H_ */
```

4. AUDIO.H

```
#ifndef __AUDIO_H_
#define __AUDIO_H_

#include "xparameters.h"

/* Redefine audio controller base address from xparameters.h */
#define AUDIO_BASE
XPAR_ZYBO_AUDIO_CTRL_0_BASEADDR

/* Slave address for the SSM audio controller */
#define IIC_SLAVE_ADDR                0b0011010
```

```
/* I2C Serial Clock frequency in Hertz */
#define IIC_SCLK_RATE                   100000

/* SSM internal registers */
enum audio_regs {

R0_LEFT_CHANNEL_ADC_INPUT_VOLUME                                = 0x00,
R1_RIGHT_CHANNEL_ADC_INPUT_VOLUME                               = 0x01,
R2_LEFT_CHANNEL_DAC_VOLUME                                      = 0x02,
R3_RIGHT_CHANNEL_DAC_VOLUME                                     = 0x03,
R4_ANALOG_AUDIO_PATH                                            = 0x04,
R5_DIGITAL_AUDIO_PATH                                           = 0x05,
R6_POWER_MANAGEMENT                                             = 0x06,
R7_DIGITAL_AUDIO_I_F                                            = 0x07,
R8_SAMPLING_RATE                                                = 0x08,
R9_ACTIVE                                                       = 0x09,
R15_SOFTWARE_RESET                                              = 0x0F,
R16_ALC_CONTROL_1                                               = 0x10,
R17_ALC_CONTROL_2                                               = 0x11,
R18_NOISE_GATE                                                  = 0x12,

};

/* Audio controller registers */
enum i2s_regs {
I2S_DATA_RX_L_REG         = 0x00 + AUDIO_BASE,
I2S_DATA_RX_R_REG         = 0x04 + AUDIO_BASE,
I2S_DATA_TX_L_REG         = 0x08 + AUDIO_BASE,
I2S_DATA_TX_R_REG         = 0x0C + AUDIO_BASE,
I2S_STATUS_REG                  = 0x10 + AUDIO_BASE,
};

/* Prototype Functions */
unsigned char IicConfig(unsigned int DeviceIdPS);
void AudioPllConfig();
void AudioWriteToReg(u8 u8RegAddr, u16 u16Data);

#endif
```

5. AUDIO.C

```
#include "tesis_ruido_interno.h"
#include "audio_ruido.h"
#include "sleep.h"

/* ------------------------------------------------------------ *
 *                          IicConfig()                         *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
unsigned char IicConfig(unsigned int DeviceIdPS)
{
XIicPs_Config *Config;
int Status;

/* Initialise the IIC driver so that it's ready to use */
```

```
// Look up the configuration in the config table
Config = XIicPs_LookupConfig(DeviceIdPS);
if(NULL == Config) {
	return XST_FAILURE;
}

// Initialise the IIC driver configuration
Status = XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress);
if(Status != XST_SUCCESS) {
	return XST_FAILURE;
}

/*
 * Perform a self-test to ensure that the hardware was built
correctly.
 */
Status = XIicPs_SelfTest(&Iic);
if (Status != XST_SUCCESS) {
	xil_printf("IIC FAILED \r\n");
	return XST_FAILURE;

}
xil_printf("IIC Passed\r\n");


//Set the IIC serial clock rate.
Status = XIicPs_SetSClk(&Iic, IIC_SCLK_RATE);
if (Status != XST_SUCCESS) {
	return XST_FAILURE;
}

return XST_SUCCESS;
}

/* ------------------------------------------------------------ *
 *                  AudioPllConfig()                            *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
void AudioPllConfig() {

AudioWriteToReg(R15_SOFTWARE_RESET,               0b000000000);
//Perform Reset
usleep(75000);
AudioWriteToReg(R6_POWER_MANAGEMENT,
0b000110000); //Power Up
AudioWriteToReg(R0_LEFT_CHANNEL_ADC_INPUT_VOLUME,  0b000010111);
//Default Volume
AudioWriteToReg(R1_RIGHT_CHANNEL_ADC_INPUT_VOLUME, 0b000010111);
//Default Volume
AudioWriteToReg(R2_LEFT_CHANNEL_DAC_VOLUME,        0b001111001);
AudioWriteToReg(R3_RIGHT_CHANNEL_DAC_VOLUME,       0b001111001);
AudioWriteToReg(R4_ANALOG_AUDIO_PATH,              0b000010010);
//Allow Mixed DAC, Mute MIC
AudioWriteToReg(R5_DIGITAL_AUDIO_PATH,             0b000000111);
//48 kHz Sampling Rate emphasis, no high pass
AudioWriteToReg(R7_DIGITAL_AUDIO_I_F,              0b000001110);
//I2S Mode, set-up 32 bits
```

```
AudioWriteToReg(R8_SAMPLING_RATE,                    0b000000000);
usleep(75000);
1.      AudioWriteToReg(R9_ACTIVE,
        0b000000001); //Activate digital core
AudioWriteToReg(R6_POWER_MANAGEMENT,                 0b000100010);
//Output Power Up

}

/* ------------------------------------------------------------ *
 *                  AudioWriteToReg                             *
 * ------------------------------------------------------------ *
 * Function to write to one of the registers from the audio
 * controller.
 * ------------------------------------------------------------ */
void AudioWriteToReg(u8 u8RegAddr, u16 u16Data) {

unsigned char u8TxData[2];

u8TxData[0] = u8RegAddr << 1;
u8TxData[0] = u8TxData[0] | ((u16Data >> 8) & 0b1);

u8TxData[1] = u16Data & 0xFF;

XIicPs_MasterSendPolled(&Iic, u8TxData, 2, IIC_SLAVE_ADDR);
while(XIicPs_BusIsBusy(&Iic));
}
```

6. IP_FUNCTIONS_RUIDO_INTERNO.C

```
#include "tesis_ruido_interno.h"
#include "audio_ruido.h"

/* ------------------------------------------------------------ *
 *                        Filtro LMS                            *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */

void lms_filter()
{
u32 nco_in, nco_out, in_left, in_right, out_left, out_right,
step, \
        prevL, prevR, prevTone, temp;

/* Lectura del valor de los switchs para la generación del tono
*/
step = XGpio_DiscreteRead(&Gpio, SWITCH_CHANNEL);

nco_in = step;
xil_printf("Step = %d, nco_in = %d\r\n",step, nco_in);

while (!XUartPs_IsReceiveData(UART_BASEADDR)){
```

```
	/* Ingreso del valor de paso al núcleo NCO */
	XNco_Set_step_size_V(&Nco, nco_in);

	/* Recepción de la señal muestreada desde el núcleo NCO*/
	nco_out = XNco_Get_sine_sample_V(&Nco);

	if(nco_out!=prevTone) { /* Nueva muestra de rudio */
		temp = nco_out;
	}

	/* Muestreo del audio L+R */

	in_left = Xil_In32(I2S_DATA_RX_L_REG);
	in_right = Xil_In32(I2S_DATA_RX_R_REG);

	/* ------------------------------------------------------ *
	 * ---------------- CANAL IZQUIERDO -------------------- *
	 * ------------------------------------------------------ */
	if(in_left != prevL) /* Nueva muestra del canal izquierdo
*/
	{
		/* Agregar la componente de ruido a las muestras de
audio L+R */
		out_left = (temp + in_left);

		Xil_Out32(LMS_D, out_left >> SCALE);   // Audio+Ruido
como la señal d
		Xil_Out32(LMS_X, temp >> SCALE);       // Ruido como
la señal x
		Xil_Out32(LMS_STROBE, 0x01);

		/* Condicionamiento para los botones */
		if(XGpio_DiscreteRead(&Gpio, BUTTON_CHANNEL)>0){

			/* Espera hasta que el dato de salida esté
listo */
			out_left = (Xil_In32(LMS_E) << (SCALE-1)); //
salida del audio filtrado
		}

		/* salida del audio al codec */
		Xil_Out32(I2S_DATA_TX_L_REG, out_left);

	}

	/* ------------------------------------------------------ *
	 * ------------------ CANAL DERECHO -------------------- *
	 * ------------------------------------------------------ */
	if(in_right != prevR) /* Nueva muestra del canal derecho */
	{
		/* Agregar la componente de ruido a las muestras de
audio L+R */
		out_right = (temp + in_right);

		Xil_Out32(LMS_D, out_right >> SCALE);  // Audio+Ruido
como la señal d
```

```
            Xil_Out32(LMS_X, temp >> SCALE);             //
Ruido como la señal x
            Xil_Out32(LMS_STROBE, 0x01);

            /* Condicionamiento para los botones */
            if(XGpio_DiscreteRead(&Gpio, BUTTON_CHANNEL)>0){
                  out_right = (Xil_In32(LMS_E) << (SCALE-1)); //
output filtered audio
            }

            /* salida del audio al codec */
            Xil_Out32(I2S_DATA_TX_R_REG, out_right);
      }

      /* Actualización de los valores de entrada */
      prevL = in_left;
      prevR = in_right;
      prevTone = nco_out;


      if(step != XGpio_DiscreteRead(&Gpio, SWITCH_CHANNEL))
break;
} // while

if(XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET)=='m')
menu();
else lms_filter();

} // LMS filtering

/* ------------------------------------------------------------ *
 *                  tonal_noise()                               *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
void tonal_noise(void)
{
u32 nco_in, nco_out, in_left, in_right, out_left, out_right,
step, temp;

/* Lee el valor de los switches */
step = XGpio_DiscreteRead(&Gpio, SWITCH_CHANNEL);

nco_in = step;
xil_printf("Step = %d, nco_in = %d\r\n",step, nco_in);

while (!XUartPs_IsReceiveData(UART_BASEADDR)){

      /* Entrada del valor de paso al núcleo NCO */
      XNco_Set_step_size_V(&Nco, nco_in);

      /* Recepción de la muestra de la señal de ruido en el nco
*/
      nco_out = XNco_Get_sine_sample_V(&Nco);

      temp = nco_out;

      /* Muestra del audio L+R audio desde el codec */
```

```
	in_left = Xil_In32(I2S_DATA_RX_L_REG);
	in_right = Xil_In32(I2S_DATA_RX_R_REG);


	/* Adicion de la componente de ruido a las muetras de audio
L+R */
	out_left =  temp + in_left;
	out_right = temp + in_right;

	/* Salida del audio con el rudio a el codec */
	Xil_Out32(I2S_DATA_TX_L_REG, out_left);
	Xil_Out32(I2S_DATA_TX_R_REG, out_right);

	if(step != XGpio_DiscreteRead(&Gpio, SWITCH_CHANNEL))
break;
} // while

if(XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET) == 'm')
menu();
else tonal_noise();

} // tonal_noise()

/* ------------------------------------------------------------ *
 *                        audio_stream()                        *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */

void audio_stream(){

//int index = 0;
u32  in_left, in_right;

while (!XUartPs_IsReceiveData(UART_BASEADDR)){

	// lectura del ingreso del audio del codec
	in_left = Xil_In32(I2S_DATA_RX_L_REG);
	in_right = Xil_In32(I2S_DATA_RX_R_REG);

	// escritura del ingreso del audio del codec
	Xil_Out32(I2S_DATA_TX_L_REG, in_left);
	Xil_Out32(I2S_DATA_TX_R_REG, in_right);

}

if(XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET) == 'm')
menu();
else audio_stream();
} // audio_stream()


/* ------------------------------------------------------------ *
 *                 Inicializacion del GPIO                      *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
unsigned char gpio_init()
{
```

```
int Status;

Status = XGpio_Initialize(&Gpio, BUTTON_SWITCH_ID);
if(Status != XST_SUCCESS) return XST_FAILURE;
Status = XGpio_Initialize(&Gpio_audio_enable, AUDIO_ENABLE_ID);
if(Status != XST_SUCCESS) return XST_FAILURE;

XGpio_SetDataDirection(&Gpio_audio_enable, 1, 0x00);
XGpio_SetDataDirection(&Gpio, SWITCH_CHANNEL, 0xFF);
XGpio_SetDataDirection(&Gpio, BUTTON_CHANNEL, 0xFF);

return XST_SUCCESS;
}

/* ------------------------------------------------------------ *
 *                  Inicialización del NCO                      *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
void nco_init(void *InstancePtr){
XNco_Config *cfgPtr;
int status;

// Busca la configuracion del NCO
cfgPtr = XNco_LookupConfig(NCO_ID);
if (!cfgPtr) {
      print("ERROR: Busqueda de la configuracion del NCO
fallida.\n\r");
}

// Inicializa el controlador del NCO
status = XNco_CfgInitialize(InstancePtr, cfgPtr);
if (status != XST_SUCCESS) {
      print("ERROR: No se puede inicializar el NCO.\n\r");
}
}
```

ANEXO 6 PROGRAMACIÓN DE LOS ARCHIVOS FUENTE DE EXTENSIÓN C Y H PARA EL PROYECTO.

1. TESIS_BORRADOR.H

```
#ifndef TESIS_BORRADOR_H_
#define TESIS_BORRADOR_H_

/* ------------------------------------------------------------ *
 *                         Header Files                         *
 * ------------------------------------------------------------ */
#include <stdio.h>
#include <xil_io.h>
#include <sleep.h>
#include "xiicps.h"
#include <xil_printf.h>
#include <xparameters.h>
#include "xgpio.h"
#include "xuartps.h"
#include "stdlib.h"

/* ------------------------------------------------------------ *
 *                  Custom IP Header Files                      *
 * ------------------------------------------------------------ */
#include "audio.h"
#include "lms_pcore_addr.h"

/* ------------------------------------------------------------ *
 *                  Prototype Functions                         *
 * ------------------------------------------------------------ */
void menu();
void noise();
void audio_stream();
void lms_filter();
unsigned char gpio_init();
void nco_init(void *InstancePtr);

/* ------------------------------------------------------------ *
 *           Redefinitions from xparameters.h                   *
 * ------------------------------------------------------------ */

#define LMS_LOC XPAR_LMS_PCORE_OP_0_BASEADDR
#define LMS_X LMS_LOC + x_k__Data_lms_pcore
#define LMS_D LMS_LOC + d_k__Data_lms_pcore
#define LMS_E LMS_LOC + e_k__Data_lms_pcore
#define LMS_STROBE LMS_LOC + IPCore_Strobe_lms_pcore

#define UART_BASEADDR XPAR_PS7_UART_1_BASEADDR
#define AUDIO_ENABLE_ID XPAR_AXI_GPIO_0_DEVICE_ID

/* ------------------------------------------------------------ *
 *                  Audio Scaling Factor                        *
 * ------------------------------------------------------------ */
#define SCALE 7
```

```
/* ------------------------------------------------------------ *
 *                  Global Variables                            *
 * ------------------------------------------------------------ */
XIicPs Iic;
XGpio Gpio_audio_enable; // GPIO instance for digital mute


#endif /* TESIS_BORRADOR_H_ */
```

2. TESIS_BORRADOR.C

```
#include "tesis_borrador.h"

/* ------------------------------------------------------------ *
 *                         main()                               *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
int main(void)
{

IicConfig(XPAR_XIICPS_0_DEVICE_ID);
xil_printf("Entrada Principal\r\n");

AudioPllConfig();
xil_printf("SSM2603 Configurado\r\n");

gpio_init();
xil_printf("GPIO Configurado\r\n");

xil_printf("\r\n");
xil_printf("     UNIVERSIDAD NACIONAL DE CHIMBORAZO      \r\n");
xil_printf("              FACULTAD DE INGENIERIA         \r\n");
xil_printf("  CARRERA DE INGENIERIA EN ELECTRONICA Y
TELECOMUNICACIONES    \r\n");
xil_printf("\r\n");
xil_printf("\r\n");
xil_printf("              PROYECTO DE GRADUACION         \r\n");
xil_printf("DISENO E IMPLEMENTACION DE UN FILTRO CON ALGORITMO
ADAPTATIVO \r\n");
xil_printf("   EN FPGA PARA LA CANCELACION DE RUIDO     \r\n");
xil_printf("\r\n");
xil_printf("\r\n");
xil_printf("AUTORES: Alejandro Ayala y Nadezhda Cordova\r\n");
xil_printf("DIRECTOR: Ing. Fabian Gunsha\r\n");

menu();
    return 1;
}

/* ------------------------------------------------------------ *
 *                         menu()                               *
 * ------------------------------------------------------------ *
*/
```

```
void menu(){
u8 inp = 0x00;
u32 CntrlRegister;

XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 0);

CntrlRegister = XUartPs_ReadReg(UART_BASEADDR,
XUARTPS_CR_OFFSET);

XUartPs_WriteReg(UART_BASEADDR, XUARTPS_CR_OFFSET,
                     ((CntrlRegister & ~XUARTPS_CR_EN_DIS_MASK) |
                      XUARTPS_CR_TX_EN | XUARTPS_CR_RX_EN));

xil_printf("\r\n\r\n");
xil_printf("------PRUEBA DEL FILTRO CON FUENTE DE RUIDO EXTERNA--
----\r\n");
xil_printf("| Seleccione una opcion del siguiente menu:
|\r\n");
xil_printf("| Presione 'a' para escuchar el audio de entrada
|\r\n");
xil_printf("| Presione 'r' para agregar ruido al audio
|\r\n");
xil_printf("| Presione 'f' para la respuesta del filtro LMS
|\r\n");
xil_printf("-----------------------------------------------------
----\r\n");


while (!XUartPs_IsReceiveData(UART_BASEADDR));
                    inp = XUartPs_ReadReg(UART_BASEADDR,
XUARTPS_FIFO_OFFSET);

switch(inp){
case 'a':
       xil_printf("SENAL DE AUDIO\r\n");
       xil_printf("Presione 'm' para regresar al menu
principal\r\n");
       XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 1);
       audio_stream();
       break;
case 'r':
       xil_printf("ADICION DE RUIDO A LA SENAL DE AUDIO\r\n");
       xil_printf("Presione 'm' para regresar al menu
principal\r\n");
       XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 1);
       noise();
       break;
case 'f':
       xil_printf("FILTRADO LMS\r\n");
       xil_printf("Presione 'm' para regresar al menu
principal\r\n");
       XGpio_DiscreteWrite(&Gpio_audio_enable, 1, 1);
       lms_filter();
       break;
default:
       menu();
       break;
```

```
} // switch
} // menu()
```

3. LMS_PCORE_ADDR.H

```
#ifndef LMS_PCORE_H_
#define LMS_PCORE_H_

#define  IPCore_Reset_lms_pcore    0x0  //write 0x1 to bit 0 to
reset IP core
#define  IPCore_Enable_lms_pcore   0x4  //enabled (by default)
when bit 0 is 0x1
#define  IPCore_Strobe_lms_pcore   0x8  //write 1 to bit 0 after
write all input data
#define  IPCore_Ready_lms_pcore    0xC  //wait until bit 0 is 1
before read output data
#define  x_k__Data_lms_pcore       0x100  //data register for
port x(k)
#define  d_k__Data_lms_pcore       0x104  //data register for
port d(k)
#define  e_k__Data_lms_pcore       0x108  //data register for
port e(k)

#endif /* LMS_PCORE_H_ */
```

4. AUDIO.H

```
#ifndef __AUDIO_H_
#define __AUDIO_H_

#include "xparameters.h"

/* Redefine audio controller base address from xparameters.h */
#define AUDIO_BASE
XPAR_ZYBO_AUDIO_CTRL_0_BASEADDR

/* Slave address for the SSM audio controller */
#define IIC_SLAVE_ADDR                0b0011010

/* I2C Serial Clock frequency in Hertz */
#define IIC_SCLK_RATE                 100000

/* SSM internal registers */
enum audio_regs {
R0_LEFT_ADC_INPUT                          = 0x00,
R1_RIGHT_ADC_INPUT                         = 0x01,
R2_LEFT_DAC                                = 0x02,
R3_RIGHT_DAC                               = 0x03,
R4_ANALOG_AUDIO_PATH                       = 0x04,
R5_DIGITAL_AUDIO_PATH                      = 0x05,
R6_POWER_MANAGEMENT                        = 0x06,
R7_DIGITAL_AUDIO_I_F                       = 0x07,
```

```
R8_SAMPLING_RATE                                = 0x08,
R9_ACTIVE                                       = 0x09,
R15_SOFTWARE_RESET                              = 0x0F,
R16_ALC_CONTROL_1                               = 0x10,
R17_ALC_CONTROL_2                               = 0x11,
R18_NOISE_GATE                                  = 0x12,
};

/* Audio controller registers */
enum i2s_regs {
I2S_DATA_RX_L_REG          = 0x00 + AUDIO_BASE,
I2S_DATA_RX_R_REG          = 0x04 + AUDIO_BASE,
I2S_DATA_TX_L_REG          = 0x08 + AUDIO_BASE,
I2S_DATA_TX_R_REG          = 0x0C + AUDIO_BASE,
I2S_STATUS_REG             = 0x10 + AUDIO_BASE,
};

/* Prototype Functions */
unsigned char IicConfig(unsigned int DeviceIdPS);
void AudioPllConfig();
void AudioWriteToReg(u8 u8RegAddr, u16 u16Data);


#endif
```

5. AUDIO.C

```
#include "tesis_borrador.h"
#include "audio.h"
#include "sleep.h"

/* ------------------------------------------------------------ *
 *                          IicConfig()                         *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
unsigned char IicConfig(unsigned int DeviceIdPS)
{
XIicPs_Config *Config;
int Status;

/* Initialise the IIC driver so that it's ready to use */

// Look up the configuration in the config table
Config = XIicPs_LookupConfig(DeviceIdPS);
if(NULL == Config) {
      return XST_FAILURE;
}

// Initialise the IIC driver configuration
Status = XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress);
if(Status != XST_SUCCESS) {
      return XST_FAILURE;
}
```

```
/*
 * Perform a self-test to ensure that the hardware was built
correctly.
 */
Status = XIicPs_SelfTest(&Iic);
if (Status != XST_SUCCESS) {
	xil_printf("IIC FAILED \r\n");
	return XST_FAILURE;

}
xil_printf("IIC Passed\r\n");


//Set the IIC serial clock rate.
Status = XIicPs_SetSClk(&Iic, IIC_SCLK_RATE);
if (Status != XST_SUCCESS) {
	return XST_FAILURE;
}

return XST_SUCCESS;
}

/* ------------------------------------------------------------ *
 *                  AudioPllConfig()                            *
 * ------------------------------------------------------------ *
 * ------------------------------------------------------------ */
void AudioPllConfig() {


AudioWriteToReg(R15_SOFTWARE_RESET,               0b000000000);
//Perform Reset
usleep(75000);
AudioWriteToReg(R6_POWER_MANAGEMENT,              0b000110000);
//Power Up
AudioWriteToReg(R0_LEFT_ADC_INPUT,                0b000010111);
//Default Volume
AudioWriteToReg(R1_RIGHT_ADC_INPUT,               0b000010111);
//Default Volume
AudioWriteToReg(R2_LEFT_DAC,                      0b001111001);
AudioWriteToReg(R3_RIGHT_DAC,                     0b001111001);
AudioWriteToReg(R4_ANALOG_AUDIO_PATH,             0b000010010);
//Allow Mixed DAC, Mute MIC
AudioWriteToReg(R5_DIGITAL_AUDIO_PATH,            0b000000111);
//48 kHz Sampling Rate emphasis, no high pass
AudioWriteToReg(R7_DIGITAL_AUDIO_I_F,             0b000001110);
//I2S Mode, set-up 32 bits
AudioWriteToReg(R8_SAMPLING_RATE,                 0b000000000);
usleep(75000);
AudioWriteToReg(R9_ACTIVE,                        0b000000001);
//Activate digital core
AudioWriteToReg(R6_POWER_MANAGEMENT,              0b000100010);
//Output Power Up

}
/* ------------------------------------------------------------ *
 *                  AudioWriteToReg                             *
 * ------------------------------------------------------------ *
```

```
 * ---------------------------------------------------------- */
void AudioWriteToReg(u8 u8RegAddr, u16 u16Data) {

unsigned char u8TxData[2];

u8TxData[0] = u8RegAddr << 1;
u8TxData[0] = u8TxData[0] | ((u16Data >> 8) & 0b1);

u8TxData[1] = u16Data & 0xFF;

XIicPs_MasterSendPolled(&Iic, u8TxData, 2, IIC_SLAVE_ADDR);
while(XIicPs_BusIsBusy(&Iic));
}
```

6. IP_FUNCTIONS.C

```
#include "tesis_borrador.h"
#include "audio.h"

/* ------------------------------------------------------------ *
 *                          FILTRO LMS                          *
 * ------------------------------------------------------------ *
*/
void lms_filter()
{
u32 in_left, in_right, out_left, out_right, \
      prevL, prevR, temp;

while (!XUartPs_IsReceiveData(UART_BASEADDR)){


      /* Muestreo de la señal de audio y ruido */
      in_left = Xil_In32(I2S_DATA_RX_L_REG);
      in_right = Xil_In32(I2S_DATA_RX_R_REG);

/* ---------------------------------------------------------- *
 * ---------- LECTURA RUIDO - CANAL IZQUIERDO ------------ *
 * ---------------------------------------------------------- */
      if(in_left != prevL) /* Nueva muestra de ruido */
      {
            temp = in_left;
            }

/* ---------------------------------------------------------- *
 * ---------- LECTURA AUDIO - CANAL DERECHO -------------- *
 * ---------------------------------------------------------- */
      if(in_right != prevR) /* Nueva muestra de audio*/
      {
            /* Agrega la componente de ruido a la muestra de
audio*/
            out_right = (temp + in_right);

            Xil_Out32(LMS_D, out_right >> SCALE);  // Ingreso de
audio+ruido como la señal d
            Xil_Out32(LMS_X, temp >> SCALE);             //
Ingreso del ruido como la señal x
```

```
            Xil_Out32(LMS_STROBE, 0x01);

            out_right = (Xil_In32(LMS_E) << (SCALE-1)); // audio
filtrado
            out_left= out_right;

            /* salida del audio */
            Xil_Out32(I2S_DATA_TX_R_REG, out_right);
            Xil_Out32(I2S_DATA_TX_L_REG, out_left);
      }

      /* Actualizacion de valores de ingreso  */
      prevL = in_left;
      prevR = in_right;

} // while

If (UartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET)=='m')
menu();
else lms_filter();

} // lms_filter

/* ------------------------------------------------------------ *
 *                  Reproducir audio+ruido                      *
 * ------------------------------------------------------------ *
*/

void noise(void)
{
u32 in_left, in_right, out_left, out_right, temp;

while (!XUartPs_IsReceiveData(UART_BASEADDR)){

      /* Muestreo de la señal de audio y ruido*/
      in_left = Xil_In32(I2S_DATA_RX_L_REG);
      in_right = Xil_In32(I2S_DATA_RX_R_REG);

      temp= in_left;

      /* Agrega la componente de ruido a la muestra de audio */
      out_left =  temp + in_right;
      out_right = temp + in_right;

      /* salida del audio */
      Xil_Out32(I2S_DATA_TX_L_REG, out_left);
      Xil_Out32(I2S_DATA_TX_R_REG, out_right);


} // while

if(XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET) == 'm')
menu();
else noise();

} // noise()
```

```
/* ------------------------------------------------------------ *
 *                          audio_stream()                      *
 * ------------------------------------------------------------ *
*/
void audio_stream(){

//int index = 0;
u32  in_left, in_right;

while (!XUartPs_IsReceiveData(UART_BASEADDR)){

      // Lectura de la entrada de audio

      in_right = Xil_In32(I2S_DATA_RX_R_REG);
      in_left = in_right;

      // Escritura de la entrada de audio
      Xil_Out32(I2S_DATA_TX_L_REG, in_left);
      Xil_Out32(I2S_DATA_TX_R_REG, in_right);

}

if(XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET) == 'm')
menu();
else audio_stream();
} // audio_stream()


/* ------------------------------------------------------------ *
 *                  gpio_initi()                                *
 * ------------------------------------------------------------ *
*/
unsigned char gpio_init()
{
int Status;

Status = XGpio_Initialize(&Gpio_audio_enable, AUDIO_ENABLE_ID);
if(Status != XST_SUCCESS) return XST_FAILURE;

XGpio_SetDataDirection(&Gpio_audio_enable, 1, 0x00);


return XST_SUCCESS;
}
```