Sistema basado en FPGA para el procesamiento de señales de audio digital

Daniel Felipe Gómez Díaz

Ingeniería Electrónica

David Andrés Márquez Viloria

INSTITUTO TECNOLÓGICO METROPOLITANO

19/03/2017

RESUMEN

En el siguiente informe se presenta la implementación de un DSP desarrollado en una tarjeta basada en FPGA (ZedBoard). Inicialmente se realiza un análisis del funcionamiento del convertidor análogo – digital, digital – análogo que se encuentra inmerso en el circuito integrado ADAU1761, este chip es el responsable de realizar una adquisición, conversión, procesamiento y salida del audio de la tarjeta.

Teniendo la adquisición de la señal digital, se procede con el procesamiento de la misma y posteriormente se reproduce en un auricular. Este proceso es realizado en cortos intervalos de tiempo para que se pueda tener la adquisición del audio, y con un leve retraso, poder escuchar la señal de audio procesada. De esta manera obtenemos un procesamiento en tiempo real.

El procesamiento en tiempo real es posible a través de la implementación de un buffer que permite almacenar un segmento de la señal adquirida, para que sea procesada y luego sea reproducida. Este buffer permite que se realice una adquisición continua de la señal sin perder datos mientras el procesamiento es aplicado.

Palabras clave: FPGA, Audio, Procesamiento, DSP, Buffer.

AGRADECIMIENTOS

Quiero agradecer inicialmente a mis padres que con una lucha incansable han logrado mantener una familia unida, con valores y virtudes, han apoyado cada uno de mis pasos dándome fuerzas y razones para superarme cada vez más, agradecer inmensamente a mi esposa que día a día me inyecta una dosis de alegría y entusiasmo, para luchar por los sueños y las metas planteadas en un proyecto de vida ambicioso.

Quiero agradecer a cada uno de los profesores que me han acompañado a lo largo de la vida, desde los profesores del colegio, hasta los profesores de la Universidad, cada una de estas personas han marcado mi vida de alguna manera, logrando rescatar lo mejor de cada uno de ellos.

Agradecer al ITM por lograr ser una institución reconocida, que impulsa cada vez a más a las personas que quieren superarse, que quieren estudiar y lograr tener un reconocimiento en el medio en el cual se desempeñan.

ACRÓNIMOS

ADC Analog to digital converters

DSP Digital signal processor

FPGA Field Programmable Gate Arrays

SDA Serial data

SCL Serial clock

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Los algoritmos de Procesamiento Digital de Señales DSP (Digital Signal Processing) tiene una amplia aplicación en muchos aspectos de la vida cotidiana, como las comunicaciones (celular, radio, televisión, e internet), el entretenimiento (video juegos, redes sociales), y la economía (bolsa de valores, transacciones bancarias), entre otros aspectos. Es por esto que con el pasar de los años se ha creado la necesidad de poder realizar operaciones cada vez más complejas y con menor requerimientos de recursos de la máquina. Uno de los primeros avances formales en DSP fue un artículo (NYQUIST, 1928), desde este momento se han venido estudiando y desarrollando nuevas técnicas que permitan una mayor eficiencia en la ejecución de algoritmos complejos y de alto nivel de procesamiento.

El procesamiento digital de señales requiere de la realización de una gran cantidad de operaciones matemáticas a una gran velocidad, el progreso obtenido en las grandes industrias computacionales, ha permitido que el DSP pueda ser probado en tareas mucho más exigentes como la grabación y reproducción de señales de audio y video, tales procesos deben de ser ejecutados en tiempo real y con requerimiento mínimo de recursos.

En las últimas décadas, se ha producido una migración significativa en el procesamiento de señales análogas a señales digitales, de la misma manera se han desarrollado nuevas técnicas y aplicaciones que nunca existieron en el mundo análogo, algunas de las aplicaciones actuales para el DSP son la verificación de la calidad del suministro eléctrico, los radares, la medicina, oceanografía, astronomía, telefonía, audio, video, industria automotriz, sismología, etc.

En el transcurso de este informe, se aplicará un buffer a una señal de audio, la cual debe ser adquirida, realizar la conversión de análogo a digital, procesarla, convertirla nuevamente a análoga y poder reproducirla. Para el proceso de conversión y tratamiento

de la señal es necesario la implementación de un buffer que permita almacenar segmentos de la señal para que pueda ser procesada y posteriormente enviada al dispositivo de reproducción como un audífono o un parlante, sin perder los datos de la señal que van ingresando al dispositivo de manera continua.

El procesamiento de señales digitales es distinguido de otras aéreas informáticas, debido a que el único tipo de datos que trabaja son las señales. Un gran porcentaje de estas señales son generadas por sensores, de esta manera el procesamiento de señales digitales se vuelve un punto necesario y vital para los equipos de una industria, para los celulares que día a día utilizamos, para nuestros carros, para elementos de uso militar y medicinal, etc.

El procesamiento de señales digitales ha realizado muchos cambios y ha logrado impactar de manera importante los dos principales sentidos humanos, los cuales son, la escucha y la visión, el procesamiento de señales ha logrado poder modificar una señal de audio con el fin de poder tener una mejor calidad de sonido, al igual que lograr modificar cualquier tipo de imagen para tener colores más claros y con una mejor definición.

En cuanto al procesamiento de señales digitales obtenidas de señales de audio, se debe decir que gracias al DSP se han podido crear efectos en las señales de audio que las personas amantes a la música valoran de una gran manera, en los estudios de grabación los sonidos son grabados en muchos de los casos instrumento por instrumento y por diferentes canales lo que facilita el trabajo de un ingeniero en sonido ya que de esta manera la canción queda un poco más flexible y se puede manejar fácilmente para tener un buen trabajo final. Con el ingreso del DSP es posible recrear espacios, tratar de que lo que escuche en su momento, la persona sea transportada a otro lado totalmente diferente solo con la percepción de los sentidos.

En este trabajo si se habla sobre DSP es indispensable hablar sobre lo que es un ADC (Convertidor Análogo - Digital), este elemento electrónico permite poder adquirir una señal de naturaleza analógica y convertirla a digital para poder ser modificada.

Un convertidor electrónico es un dispositivo capaz de adquirir cierto tipo de señal ya sea analógica o digital y convertirla. En este proyecto, desde una fuente de audio se adquiere una señal analógica la cual convertimos a digital por medio del ADC que tiene embebido la ZedBoard, la frecuencia de muestreo de este convertidor es de 48KHz, lo que quiere decir que tendremos 48000 muestras cada segundo transcurrido.

2. METODOLOGÍA

2.1. Ventajas de un sistema de procesamiento digital frente a un sistema analógico.

2.1.1. Flexibilidad

Al tratarse de sistemas programados se facilita el cambio de los algoritmos sin necesidad de modificar el circuito como ocurre con los sistemas analógicos. Dependiendo de la programación en el proceso de fabricación, adicional a esto los circuitos integrados disponen de diferentes tipos se memoria (ROM, EEPROM, RAM).

2.1.2. Costo

Un sistema programable puede modificar su funcionamiento sin modificar ningún circuito electrónico como ocurre con los sistemas analógicos, que deben de modificar el número de componentes.

2.1.3. Implementación de sistemas sin equivalente analógico

Digitalmente se pueden generar formas de onda arbitrarias. Se pueden almacenar las señales para un procesado posterior.

2.2. Descripción del hardware usado para el procesamiento de audio en FPGA

La implementación de este trabajo es realizada empleando sistemas de desarrollo ZedBoard el cual está basado en FPGA. Esta placa es utilizada normalmente con fines educativos por la cantidad de periféricos que maneja y debido a su diseño, pero también es usada en muchas aplicaciones industriales. Se programa por medio del Software Vivado, propiedad de la compañía Xilinx. Se utiliza la ZedBoard en el proyecto debido a que cuenta con un convertidor análogo – digital y digital – análogo (ADC). El circuito integrado de audio códec ADAU1761 inmerso en la zedboard es utilizado especialmente para el procesamiento de audio.

El códec ADAU1761 es configurado a través del bus I2C. En este proyecto se toma una aplicación para poder hacer uso de los dos ADC y realizar el muestreo del audio mono a 48KHz. Las muestras adquiridas son enviadas al chip Zynq a través del bus de audio estándar I2S para su procesamiento. Las muestras de audio procesadas son enviadas a través del bus I2S a los convertidores DAC del códec para su posterior reproducción en algún tipo de altavoz (Auriculares), conectados al puerto de salida del circuito integrado.

Los convertidores ADC, son utilizados para la digitalización de las señales análogas. Según el teorema de Nyquist, la frecuencia de muestreo de un ADC de audio debe de ser al menos el doble de la componente de frecuencia más alta del audio.

Luego de tener la señal digitalizada el procesamiento puede tener lugar en un sistema digital como lo es un DSP, microprocesador o microcontrolador, el procesamiento de la señal digital adquirida varia su complejidad dependiendo de la aplicación que se requiera, como por ejemplo algo tan complejo como lo es el reconocimiento de voz o algo un poco más sencillo como los el filtrado y la mezcla de audio. Si se desea realizar la reproducción del audio procesado, es necesario contar con un convertidor digital – análogo, convierte la corriente de valores digitales en una señal analógica y esta señal analógica podría ser enviada posteriormente a un altavoz.

El ADAU1761 dedicado al procesamiento de audio, cuenta con dos ADC de 24 bits y dos DAC de 24 bits con una amplia gama de frecuencia de muestreo de 8KHz a los 96KHz.

2.2.1. Codec ADAU1761

El ADAU1761 es un chip de audio que se puede utilizar para múltiples propósitos acústicos tales como: reproducción, grabación y procesamiento. Cuenta con dos ADC de audio de 24 bits y dos convertidores DAC de 24 bits, soporta una frecuencia de muestreo de 8Khz a 96Khz. El audio digital capturado y transmitido se lee / escribe a través del puerto de datos del chip, el puerto de datos es compatible con la interfaz de audio I2S.

Además del puerto de datos, el ADAU1761 contiene un puerto de control, que antes de empezar a utilizar el chip debe de ser configurado, la configuración de este chip se logra

enviando órdenes a través del puerto de control que es compatible con I2C. El puerto de control proporciona acceso a los registros internos, en el chip se encuentran 67 registros de control, las direcciones de estos registros van desde la dirección 0x4000 hasta a dirección 0x40F. La escritura en estos registros establece diferentes aspectos tales como: controlar amplificación, activar o desactivar mezcladores, configurar la frecuencia de muestreo, etc.

2.2.2. I2C Bus

El bit I2C, es utilizado para configurar el chip escribiendo en sus registros de control. La línea de datos en serie (SDA) en el bus I2C lleva los comandos de configuración entre el controlador maestro (Chip zynq) y el códec. El ADAU1761 es siempre un esclavo en el bus, lo que significa que este no puede iniciar una transmisión de datos. Inicialmente, el códec se encuentra inactivo y monitorea las líneas SDA y SCL para recibir una condición de inicio. El maestro I2C inicia una transferencia de datos estableciendo una condición de inicio, definido por un flanco de bajada en SDA, mientras que SCL permanece en alto. Esto indica que existe un flujo de datos. El Codec desplaza los siguientes ocho bits (la dirección de 7 bits más el bit R / W) MSB primero. El Codec reconoce la dirección transmitida y responde (reconoce) tirando de la línea de datos baja durante el noveno pulso de reloj. Este noveno bit se conoce como un bit de reconocimiento, mediante el cual el maestro sabe que el Codec está activo y dispuesto a cooperar. Después de eso, el maestro I2C envía el byte alto de subdirección del registro de control al que desea escribir, el códec reconoce la recepción del byte tirando de la línea de datos baja durante el noveno pulso de reloj. El siguiente byte a enviar es el byte de sub-dirección inferior del registro de control a escribir, de una manera similar el Codec también reconoce en el noveno ciclo de reloj. Finalmente, los datos a ser escritos en el registro de control son enviados y deben ser reconocidos en el noveno ciclo por el Codec. Esto se conoce como una operación de escritura de una sola palabra. El cual es terminado por el maestro con una condición de parada.

2.2.3. I2S Bus

El I2S es un estándar para la transferencia de datos de audio dentro de los sistemas digitales. Las señales en el protocolo I2S son dos relojes y líneas de datos conocidas como líneas de entrada y salida de datos.

2.2.4. Reloj izquierdo – derecho (LRCLK)

La función del reloj izquierdo - derecho (LRCLK) es identificar la frecuencia de muestreo del sistema de audio y enmarcar los dos canales de datos de audio que existen en la única línea de datos de audio. Como resultado de la primera función mencionada, la frecuencia requerida de la señal de reloj izquierda / derecha está siempre en la frecuencia de muestreo de audio del sistema, tal como 44,1 kHz, 48 kHz. Este reloj se utiliza para separar los datos del canal izquierdo y derecho.

2.2.5. Reloj de bits (BCLK)

El reloj de bits (BCLK) también es referido a menudo como el reloj en serie. El único propósito del reloj de bits es sincronizar el desplazamiento de los bits de datos de audio dentro o fuera del puerto de audio de datos. La frecuencia mínima requerida para el reloj de bits está directamente relacionada con la frecuencia de muestreo del audio del sistema y la longitud de la palabra de audio. Recuerde que hay dos canales de datos de audio presentados en cada período del reloj Izquierdo / Derecho, y la frecuencia del reloj Izquierdo / Derecho debe estar en la frecuencia de muestreo de audio. Por lo tanto, la frecuencia de reloj de bits mínima requerida es el doble de la frecuencia de muestreo de audio que el número de bits en cada palabra de audio.

Uno de los estándares de la industria para representar datos de audio es una palabra compuesta de 24 bits codificados en formato de punto fijo de dos complementos. La palabra de datos de audio siempre se transmite primero con el bit más significativo (MSB). Los números de punto fijo se especifican mediante una notación A.B, donde A es el número de bits a la izquierda del punto decimal y B es el número de bits a la derecha del

punto decimal. El ADAU1761 utiliza el formato de punto fijo de dos complementos de 1,23 para la salida ADC y la entrada DAC.

2.3. Creación del proyecto en VIVADO

Inicialmente ejecutamos vivado design suite, click en file → New Project, damos click en Next tal cual como lo vemos en la Figura 1.

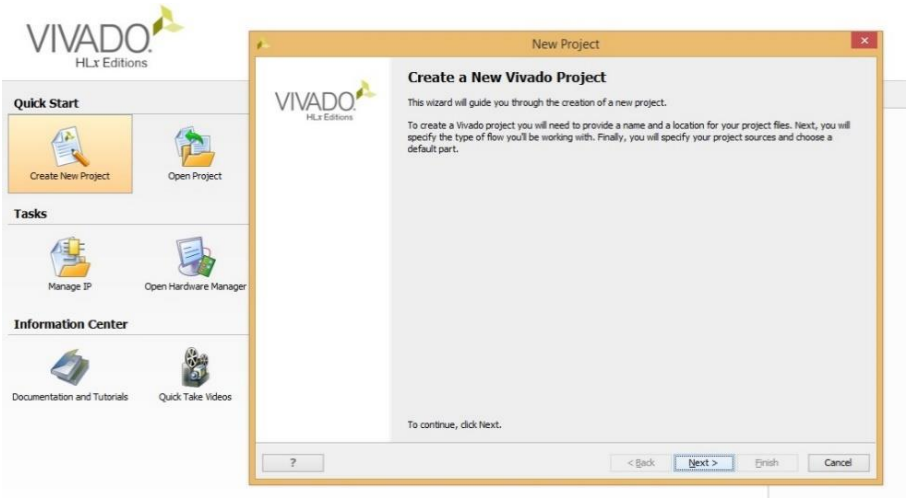

Figura 1. Ventana de creación del proyecto en vivado.

Ingresamos el nombre del proyecto y la ubicación, es recomendable crear una nueva carpeta directamente en el disco C: para grabar allí todos los archivos Figura 2 dejar seleccionada la opción Create Project subdirectory.

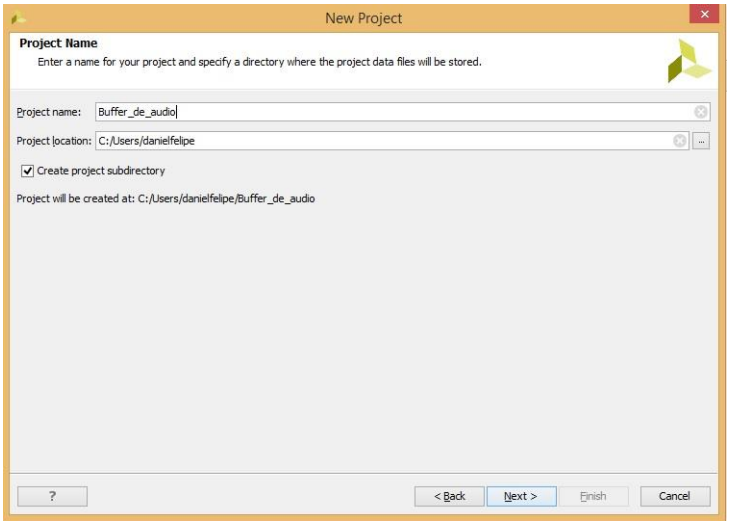

Figura 2. Nombre y ubicación del nuevo proyecto.

En la **¡Error! No se encuentra el origen de la referencia.** tendremos varias opciones entre las cuales seleccionaremos *RTL Project* y dejamos seleccionada la opción de *Do not specify sources at this time*, seleccionando la opción *RTL Project*, tendremos la facilidad de modificar los archivos

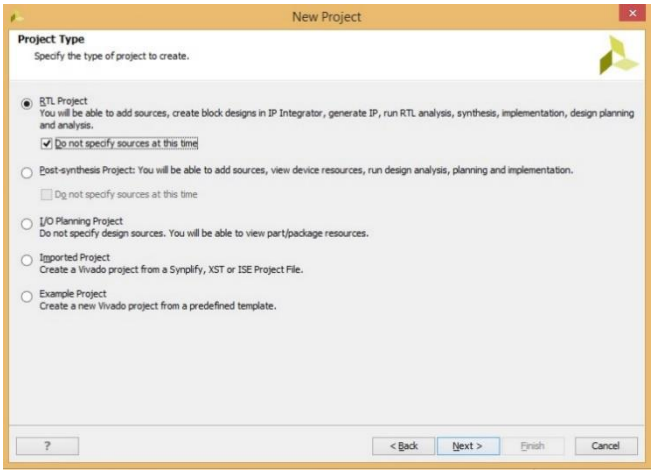

Figura 3. Tipo de proyecto

fuente según como consideremos necesario.

En la **¡Error! No se encuentra el origen de la referencia.**, debemos saber en qué tipo de tarjeta realizaremos la prueba del proyecto, para este caso usaremos la ZedBoard Zynq Evaluation and Development Kit, seleccionamos la tarjeta y luego click en siguiente y finalizar. Luego damos en siguiente y posteriormente en finalizar, el asistente para los nuevos proyectos se cierra y se abre el proyecto creado en Vivado.

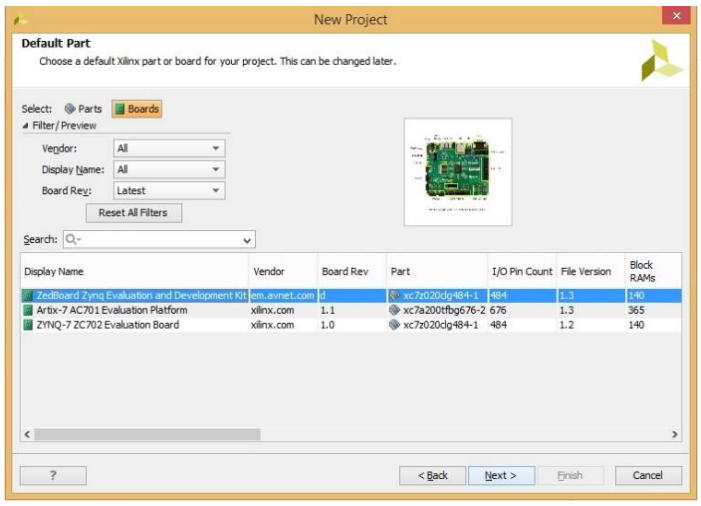

Figura 4. Selección de board para el proyecto

2.3.1. ¿Cómo crear un proyecto de procesador integrado utilizando el integrador de IP?

1. Click en Create Block Design ubicado en la parte superior izquierda (Ver Figura 5).
2. En el siguiente pantallazo escriba el nombre para el modulo y haga click en aceptar. Para el nombre del módulo utilice el nombre de System (Ver Figura 6).

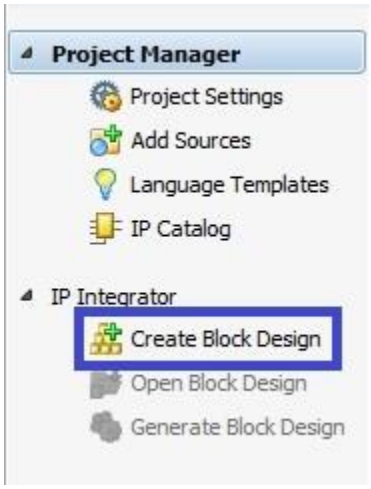

Figura 5. Ubicación para crear el diseño de bloques.

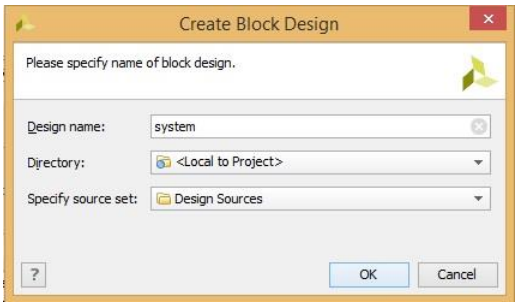

Figura 6. Ubicación y nombre del nuevo dieseño de bloques.

Luego de hacer click en OK aparecerá una ventana en blanco donde se realiza el diagrama de bloques en la interfaz gráfica de Vivado. En el diagrama en blanco se pueden agregar y conectar los diferentes bloques de hadware que necesita el diseño.

3. Ahora procedemos a agregar el bloque del sistema de procesamiento ZINQ7. Al agregar este bloque, se puede configurar uno de los núcleos del procesador ARM Cortex-A9 para su aplicación. Esto se logra agregando el modulo IP a través del wizardAdd (Ver Figura 7). Aparecerá una nueva ventana con la lista de los módulos IP que se pueden agregar al diseño.

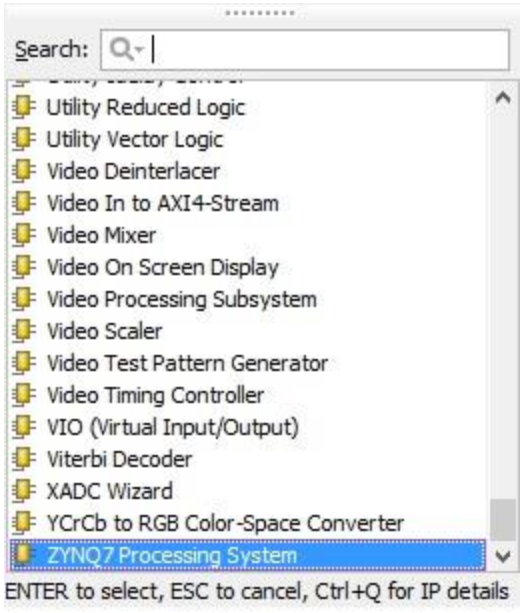

Figura 7. WizardAdd.

En el espacio para buscar podemos ingresar la palabra ZYNQ7, de esta manera muestra de manera inmediata el bloque que debemos agregar, dándole doble click el bloque se agregara automáticamente a la ventana donde se realizaran las conexiones de cada uno de estos bloques agregados al diseño.

2.4. Configuración del sistema de procesamiento ZYNQ7.

1. Click en Run Block Automation ubicada en la barra de información de color verde. En la ventana emergente se deben de dejar todas las configuraciones tal cual como están y se procede dándole click en aceptar.

Figura 8. Puertos por defecto del sistema de procesamiento.

El bloque agregado a la ventana de diseño debe parecerse al bloque en la Figura 9, se debe de aclarar que el bloque tiene unos terminales adicionales los cuales no necesariamente serán utilizados como por ejemplo en USB, Ethernet y los temporizadores.

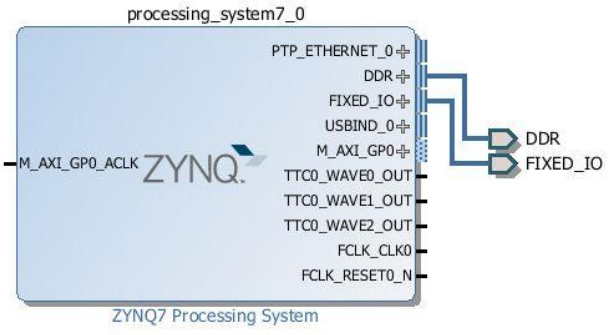

Figura 9. Sistema de procesamiento.

2. Dar doble click en el bloque processing_system7_0, de esta manera se puede abrir la personalización del módulo IP. En MIO Configuration, se debe habilitar el controlador I2C.

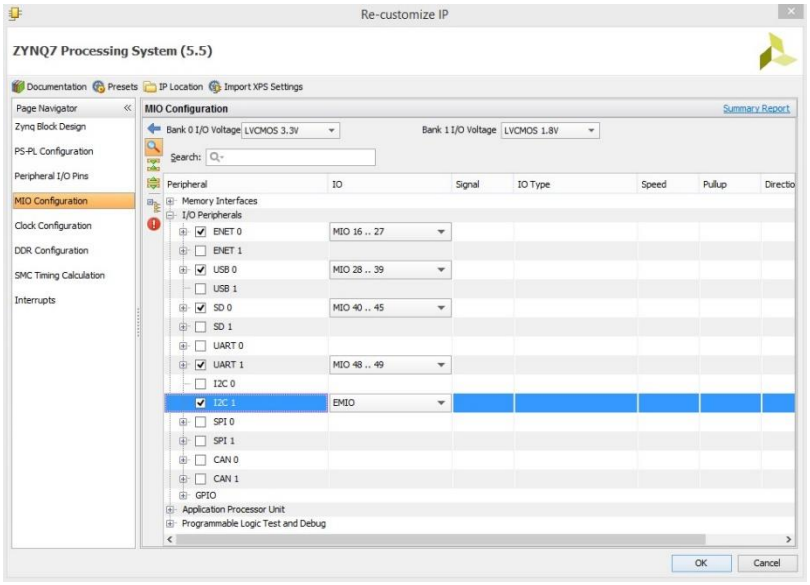

Figura 10. Modificación de puertos del Sistema de procesamiento.

3. En la pestaña Clock configuration, expanda PL fabric clocks, habilite FCLK_CLK1 y ajústelo en 10MHz.

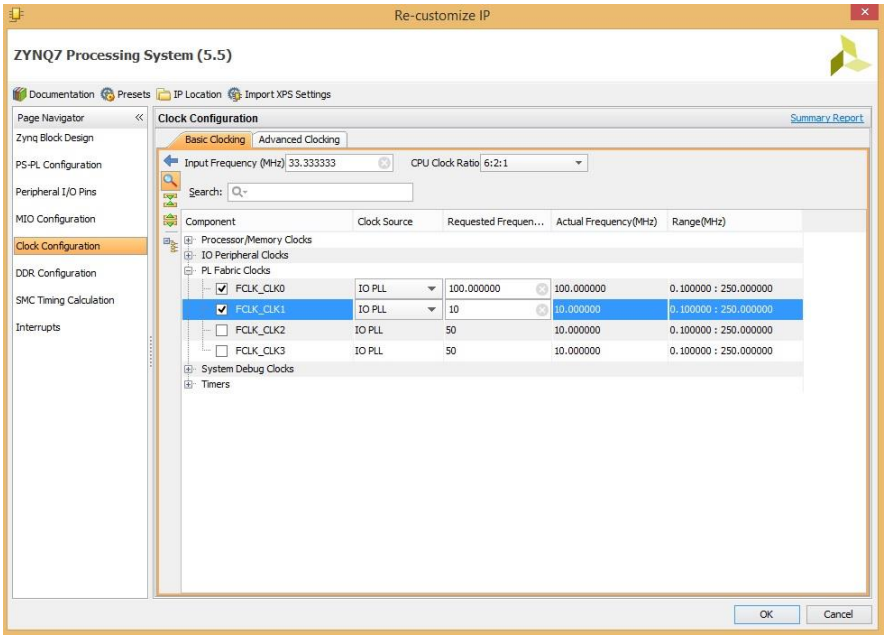

Figura 11. Configuración de reloj.

4. Este reloj es utilizado para el codec ADAU 1761. Luego dar click en aceptar para cerrar la ventana de personalización del módulo, el modulo debe de quedar parecido al siguiente (Ver Figura 12).

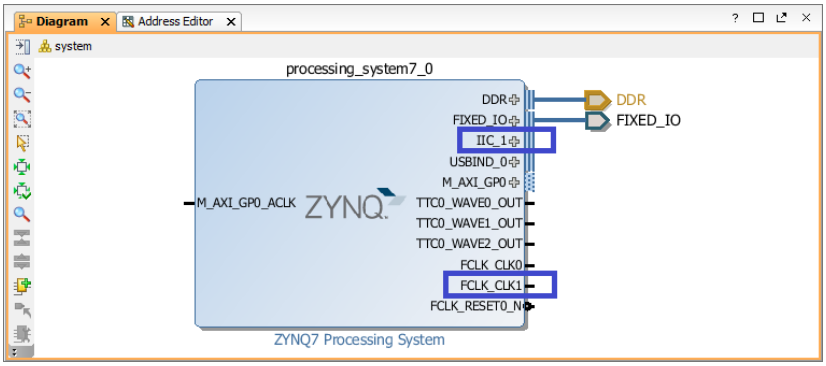

Figura 12. Puerto de control y señal de reloj.

5. Pase el mouse sobre la terminal IIC_1 hasta que el mouse cambie a un lápiz. Luego click derecho y seleccione la opción Make External.

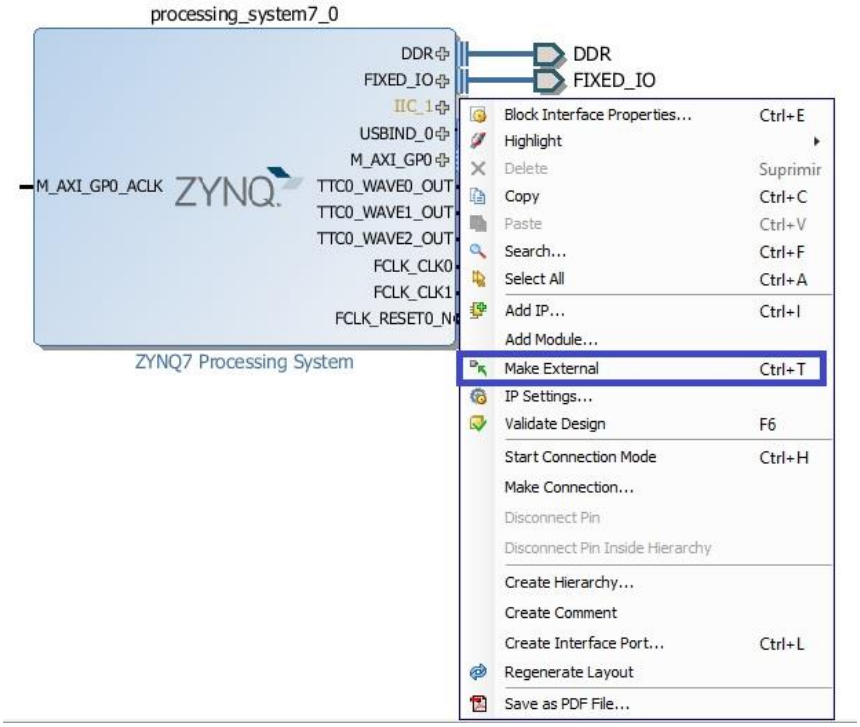

Figura 13. IIC_1 puerto externo.

6. Repita el paso 5 para el terminal FCLK_CLK1. El bloque debe quedar parecido al mostrado en la Figura 14.

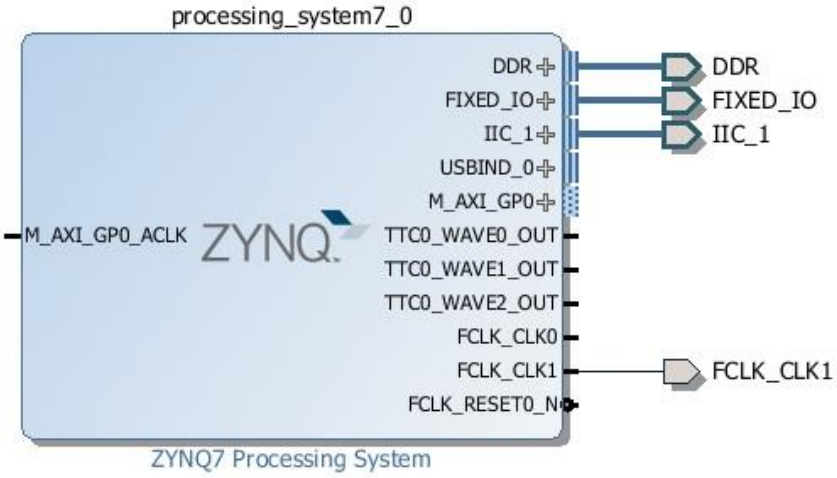

Figura 14. Sistema de procesamiento con puertos externos.

2.4.1. Agregar a la biblioteca de Vivado los módulos IP de zed_audio_ctrl y xilinx_com_hls_nco_1_0.

1. Click en Project Settings, esta opción se puede encontrar en la lista de navegación ubicada al lado izquierdo.

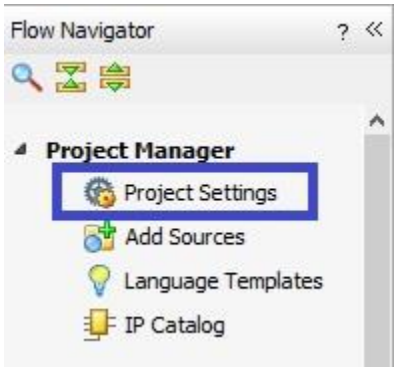

Figura 15. Ubicación configuración del proyecto.

2. En la ventana de Project settings seleccionamos la opción IP y luego en Repository Manager, en esta ventana seleccionamos el símbolo más de color verde que aparece al lado izquierdo superior (ver Figura 16).

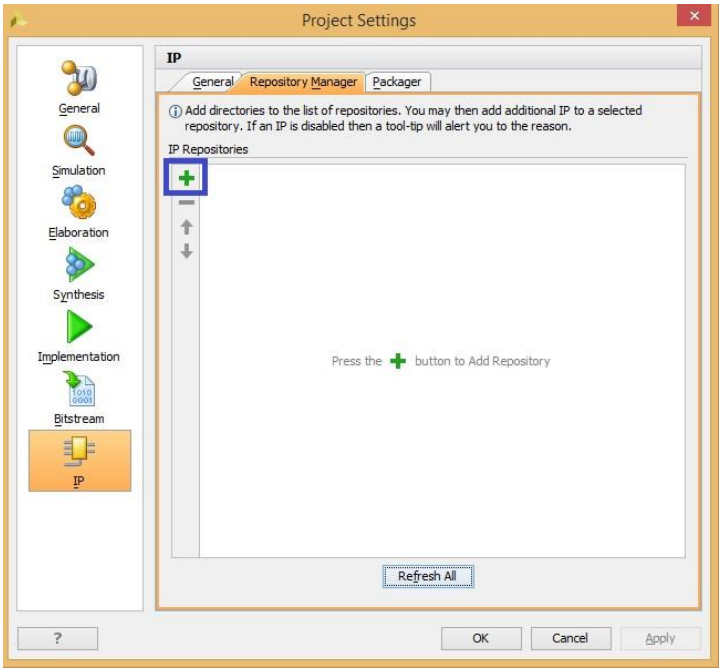

Figura 16. Ventana para agregar módulos IP.

3. Se abre una nueva ventana y en esta debemos de navegar hasta el punto donde se encuentren los archivos descargados inicialmente, posteriormente seleccionamos la carpeta.

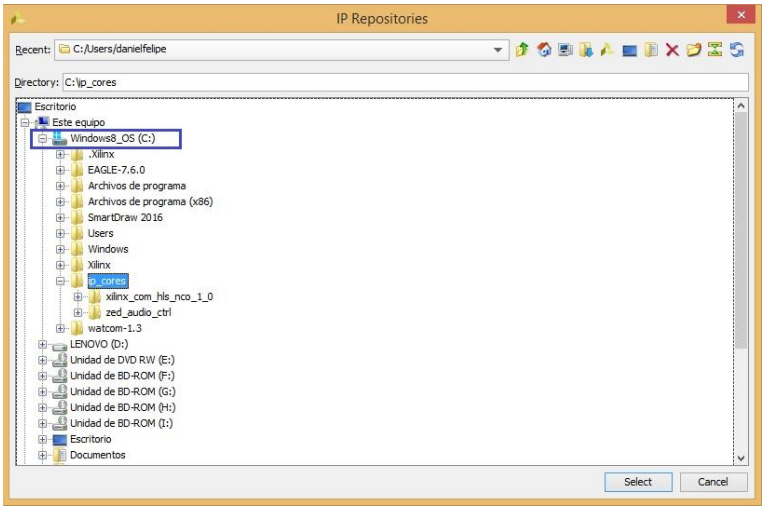

Figura 17. Módulos IP.

En la Figura 18 podemos notar que ya Vivado detecta las nuevas direcciones IP en el directorio, luego damos click en aplicar y posteriormente en aceptar.

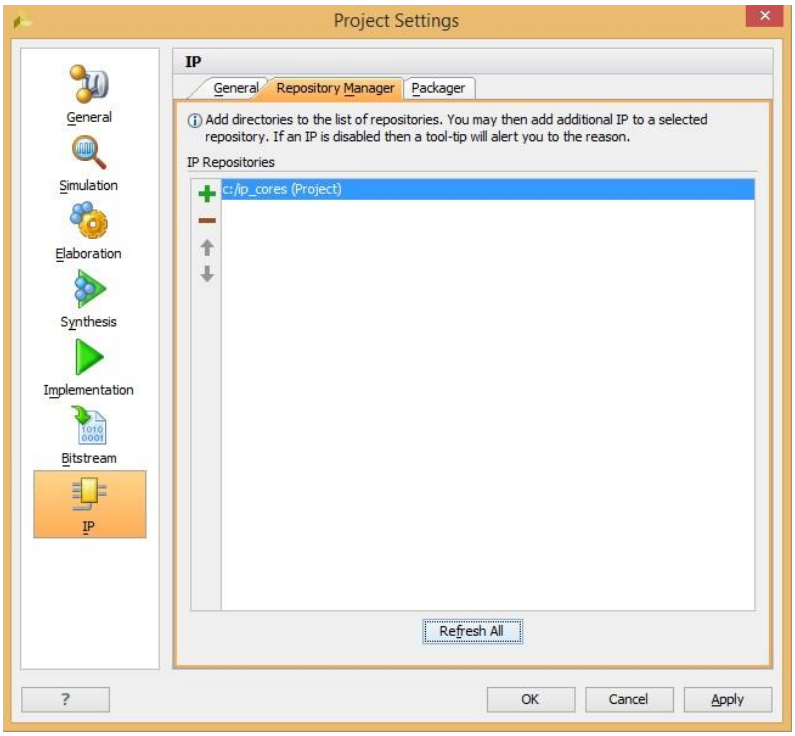

Figura 18. Administrador de módulos IP.

Con los nuevos módulos IP ya agregados a Vivado, solo queda agregarlos al diseño y realizar la conexión al sistema de procesamiento.

2.4.2. Agregar los módulos IP al diagrama de bloques y conectarlos al sistema de procesamiento.

1. Ubicado en la pestaña de diagrama en cualquier parte dar click derecho y seleccione Add IP, en el buscador introduzca “nco”, selecciónelo y haga doble click sobre el para agregarlo (Ver Figura 19).

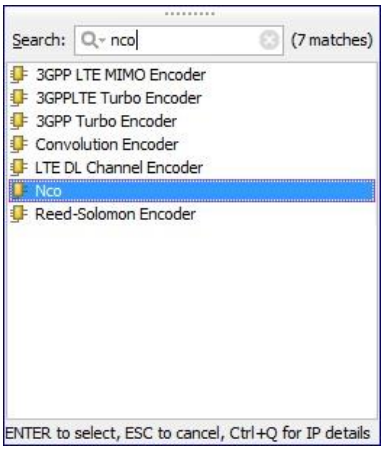

Figura 19. Módulo NCO.

2. Observe que en el asistente de configuración se encuentra disponible en la parte superior izquierda la opción de Run Connection Automation. Dar click y en la ventana emergente dejar las configuraciones por defecto y dar click en aceptar.
3. Repita los pasos 1 y 2 para zed_audio_ctrl.

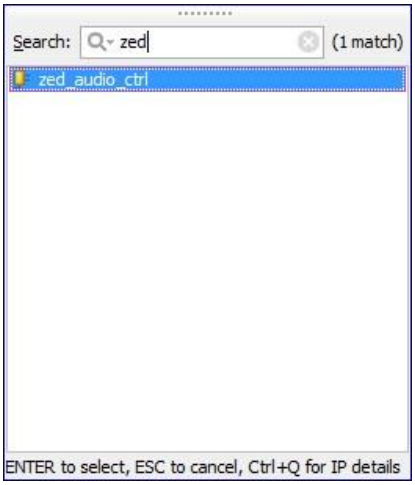

Figura 20. Módulo zed_audio_ctrl.

4. En la vista de diagrama, dar click derecho en cualquier parte libre y seleccionar la opción Regenerate Layout. El diagrama del diseño deberá ser parecido al mostrado en Figura 21.

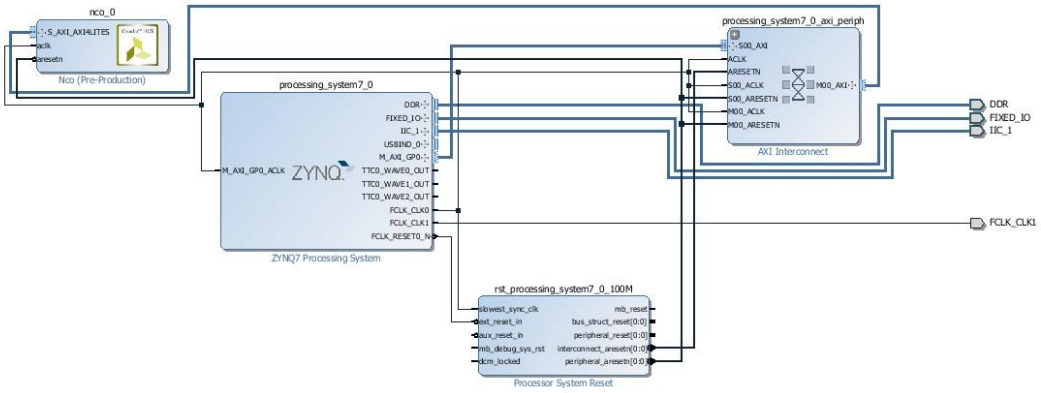

Figura 21. Diagrama de bloques.

5. Mover el mouse sobre el terminal BCLK en el bloque de zed_audio_ctrl hasta que cambie a la forma de un lápiz. Dar click derecho y seleccione Make External.
6. Repita el paso número 5 para LRCLK, SDATA_O y SDATA_I, el diagrama debe de tener la forma del diagrama mostrado en la

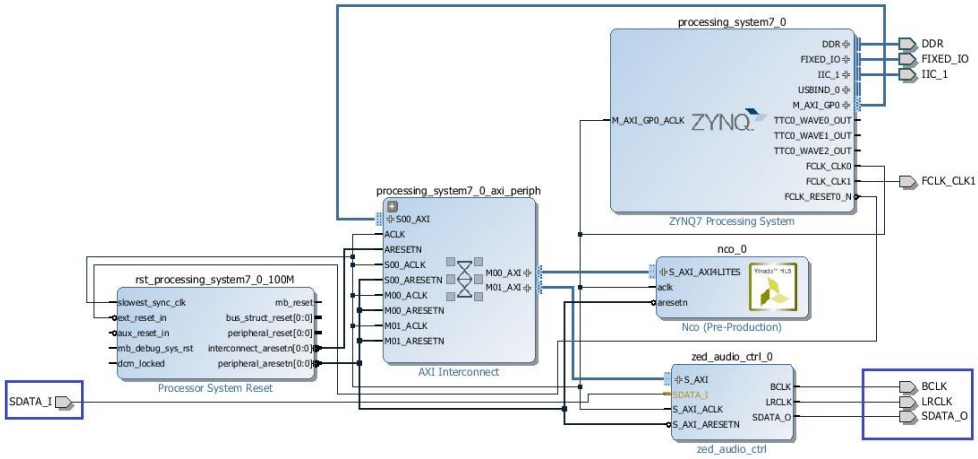

Figura 22. Diagrama de bloque.

Los puertos encerrados por un cuadro azul en la Figura 22 serán asociados con los pines conectados al codificador de audio ADAU1761, el reloj (LRCLK = BCLK), las líneas de entrada y salida (SDATA_I = SDATA_ADC), (SDATA_DAC = SDATA_O).

2.4.3. Asignar valores a los bits inferiores [1:0] del ADAU1761 dirección I2C

1. Click derecho en cualquier espacio libre de la ventana de diseño, seleccionar Create Port.

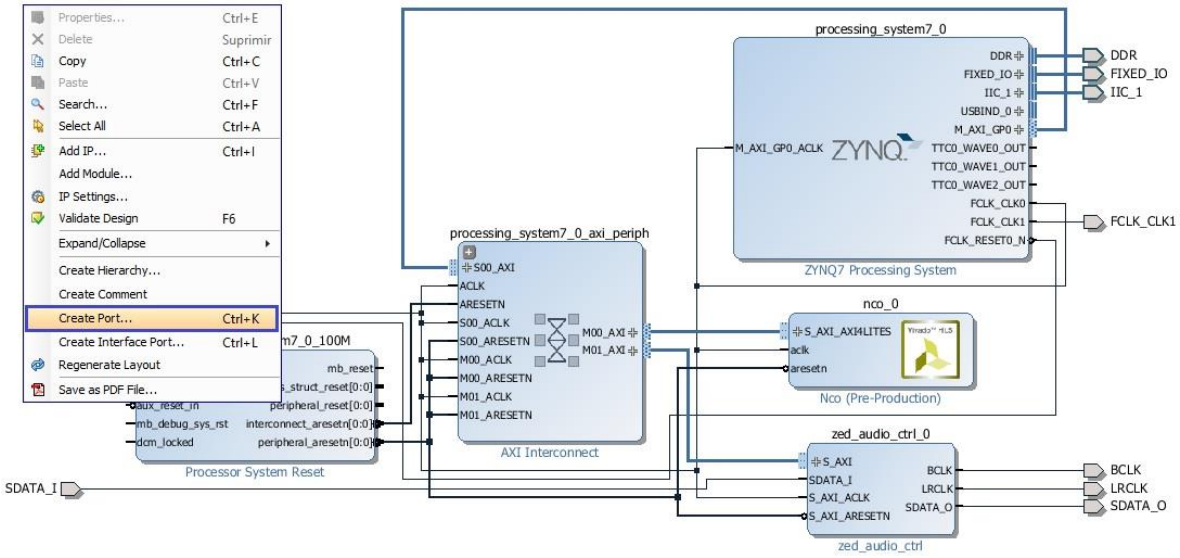

Figura 23. Asignación de valores.

2. En la ventana de crear puerto ingresar los valores que se ven en la Figura 24.

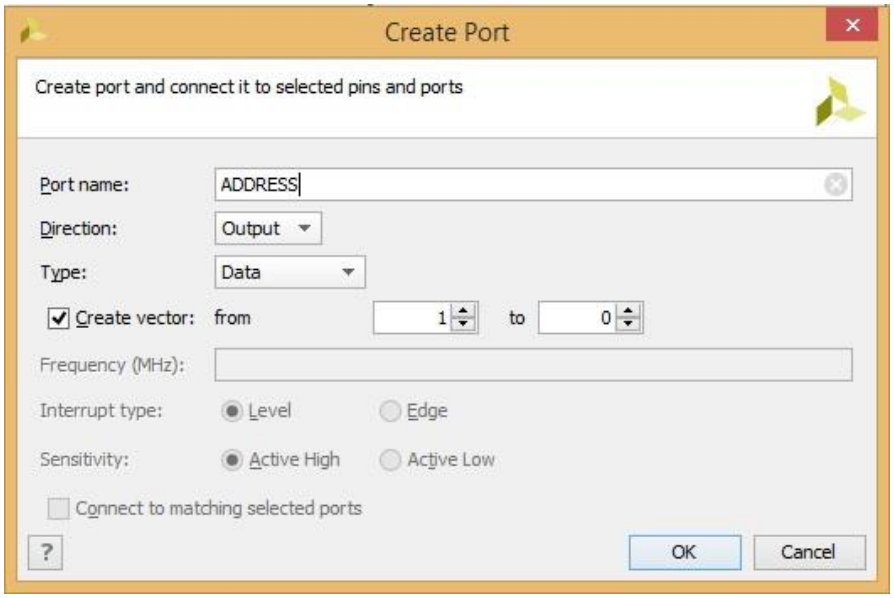

Figura 24. Creación de puertos y asignación de valores.

3. En la ventana de diseño, dar click derecho en cualquier espacio libre y seleccionar Add IP. Introduzca en el buscador “cons”. Seleccionar y dar doble click para agregarlo al diseño. El modulo IP agregado se utiliza para poder asociar el puerto “ADDRESS [1:0]” a un valor lógico fijo de cero.

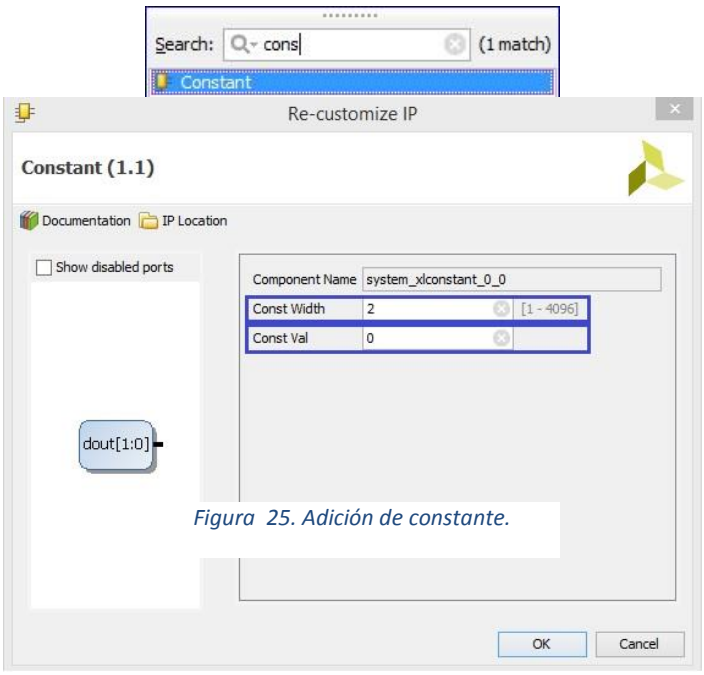

Figura 25. Adición de constante.

Figura 26. Modificación módulo constant.

4. Dar doble click sobre el modulo IP Constante para abrir la personalización del módulo, luego ingresar los valores ingresados en la Figura 26.
5. Conectar utilizando el mouse el módulo constant, con el puerto ADDRESS [1:0].
6. Click derecho en cualquier espacio libre de la ventana de diseño y seleccionar la opción Regenerate Layout. El diagrama debe de quedar similar al mostrado en la Imagen 27.

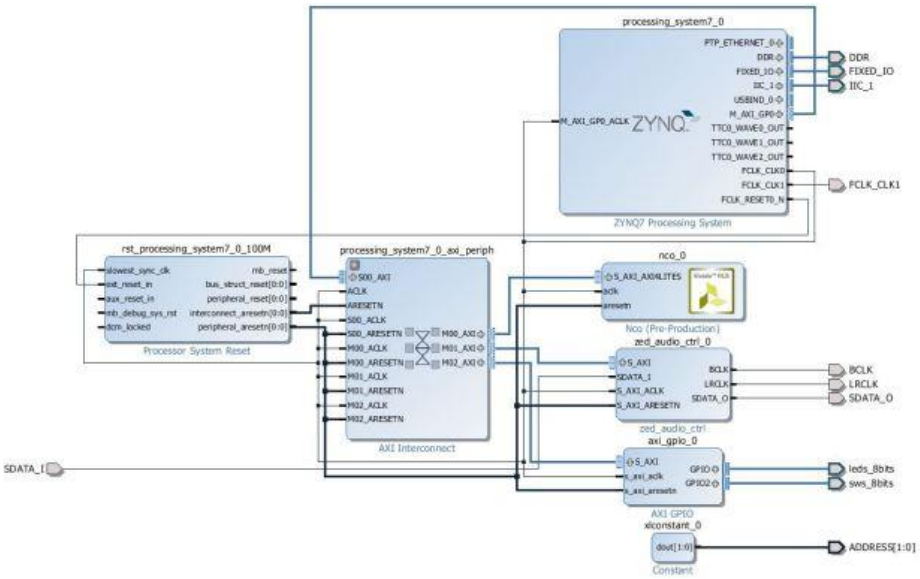

Figura 27. Diagrama de bloque final.

2.4.4. Agregar las constantes físicas

1. En la ventana Flow Navigator, seleccionar agregar fuentes desde la sección Project Manager.

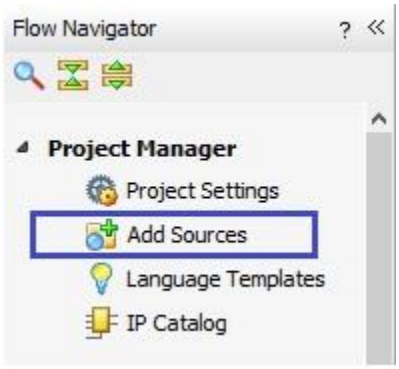

Figura 28. Adicción de fuentes.

2. En la ventana de agregar fuentes, seleccionar la opción Add or créate constraints, como muestra la Figura 29.

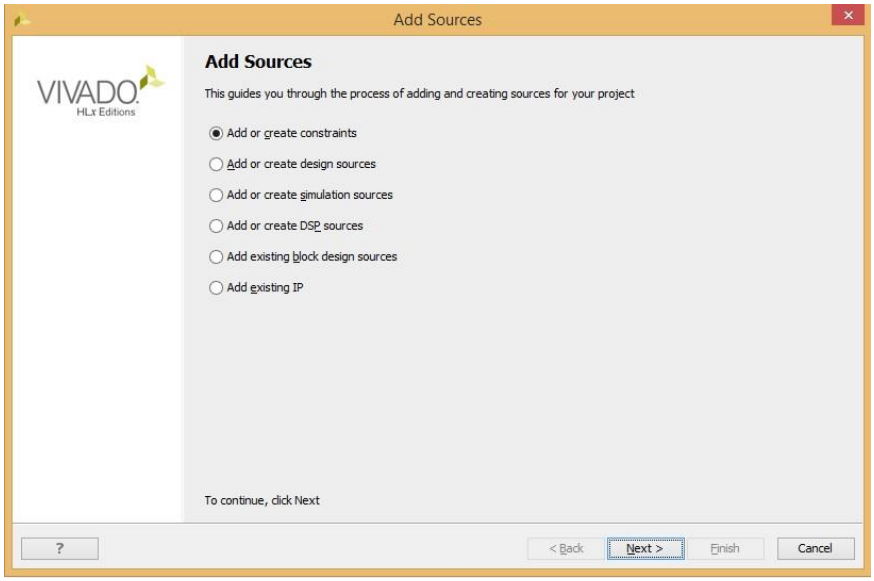

Figura 29. Wizard para agregar fuentes.

3. Click en siguiente, luego haga click en el símbolo verde de más y seleccionar crear archivo.

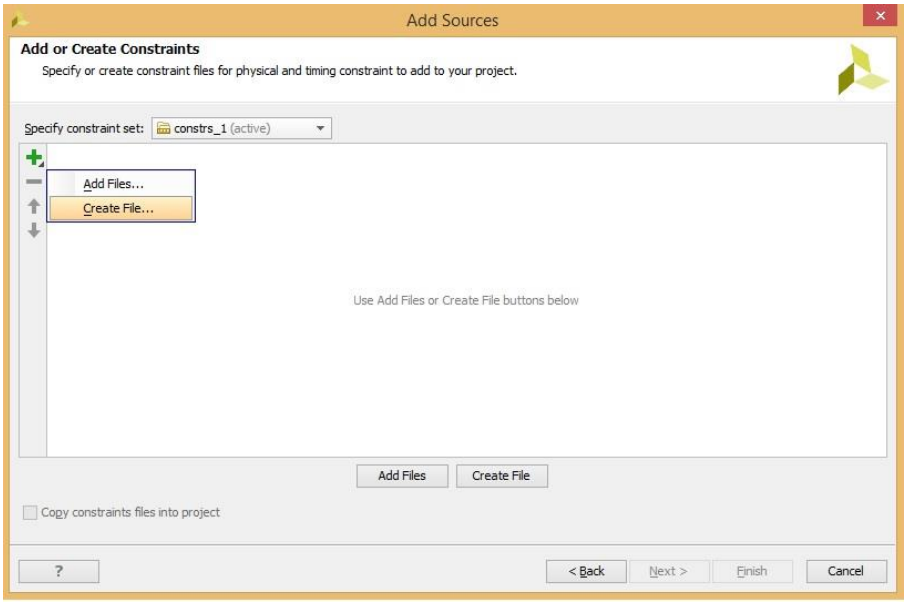

Figura 30. Crear nuevo archivo para el convertidor.

En la siguiente ventana, seleccionar XDC como tipo de archivo e ingrese el nombre de adc_dac_audio como el nombre del archivo.

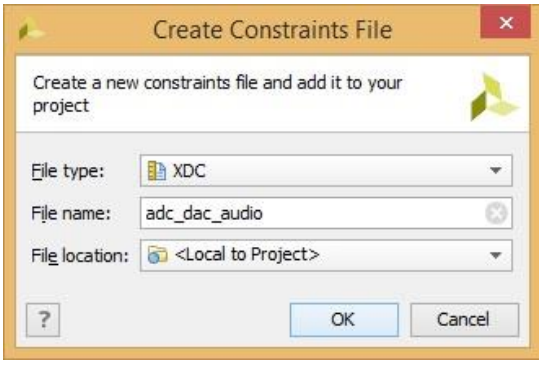

Figura 31. Nombre y tipo del nuevo archivo a crear.

Dar click en OK, y a continuación dar click en aceptar.

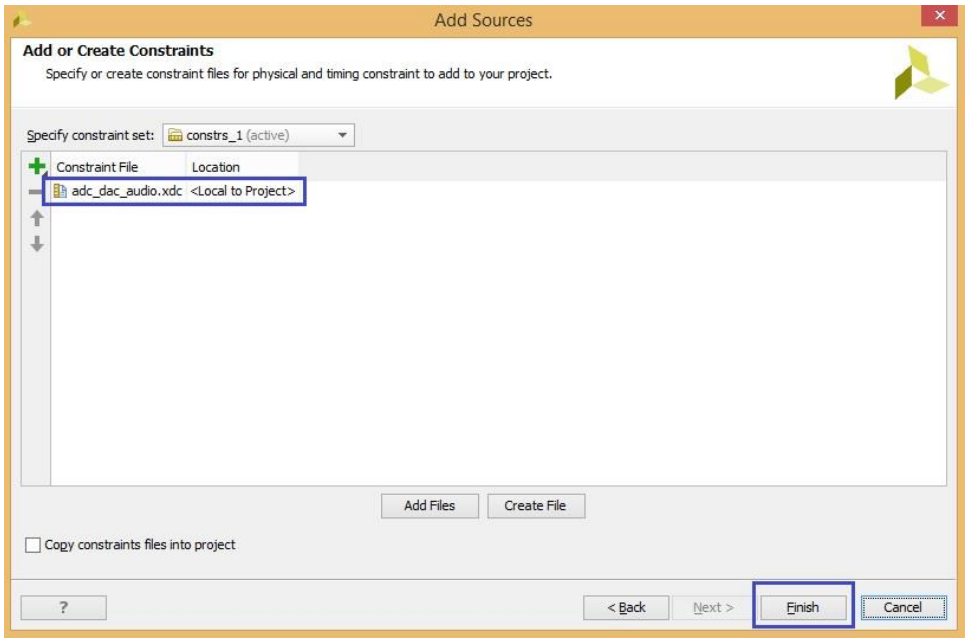

Figura 32. Finalizar proceso de creación de archivo adc_dac_audio.xdc

En la ventana del lado izquierdo donde se encuentran las fuentes se debe desplegar la carpeta de "Constraints", luego desplegar "constrs_1" y dar doble cick en adc_dac_audio.xdc.

En la ventana de instrucciones del adc_dac_audio.xdc, ingresar las siguientes líneas de código:

```
# ZedBoard Audio Codec Constraints
set_property PACKAGE_PIN AA6 [get_ports BCLK]
set_property IOSTANDARD LVCMOS33 [get_ports BCLK]

set_property PACKAGE_PIN Y6 [get_ports LRCLK]
set_property IOSTANDARD LVCMOS33 [get_ports LRCLK]

set_property PACKAGE_PIN AA7 [get_ports SDATA_I]
set_property IOSTANDARD LVCMOS33 [get_ports SDATA_I]

set_property PACKAGE_PIN Y8 [get_ports SDATA_O]
set_property IOSTANDARD LVCMOS33 [get_ports SDATA_O]

#MCLK
set_property PACKAGE_PIN AB2 [get_ports FCLK_CLK1]
set_property IOSTANDARD LVCMOS33 [get_ports FCLK_CLK1]

set_property PACKAGE_PIN AB4 [get_ports iic_1_scl_io]
set_property IOSTANDARD LVCMOS33 [get_ports iic_1_scl_io]

set_property PACKAGE_PIN AB5 [get_ports iic_1_sda_io]
set_property IOSTANDARD LVCMOS33 [get_ports iic_1_sda_io]

set_property PACKAGE_PIN AB1 [get_ports {ADDRESS[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ADDRESS[0]}]

set_property PACKAGE_PIN Y5 [get_ports {ADDRESS[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ADDRESS[1]}]
```

Click en guardar, el código anteriormente copiado tiene como función realizar la conexión de BCLK, LRCLK, SDATA_O, SDATA_I, FCLK_CLK1, iic_1_scl_io, icc_1_sda_io y ADDRESS[1], a los puertos físicos del ADAU1761 (codificador de audio).

2.5. Generar Bitstream

1. En el panel de fuentes, dar click derecho sobre *system.bd* y seleccionar *Create HDL Wrapper (Ver* Figura 33*)* para crear el archivo verilog de nivel superior del diagrama de bloques.

   Seleccionar *Let Vivado manage wrapper and auto-update (*Figura 34*)*, cuando aparezca el siguiente mensaje, note que *System_wrapper.v*, fue creado y se colocó en la parte superior de la jerarquía de las fuentes de diseño *(Imagen 35)*.

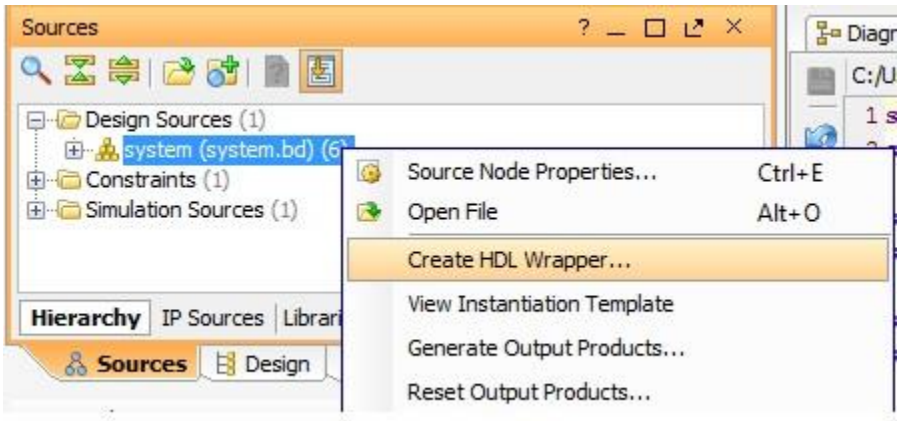

Figura 33. HDL Wrapper.

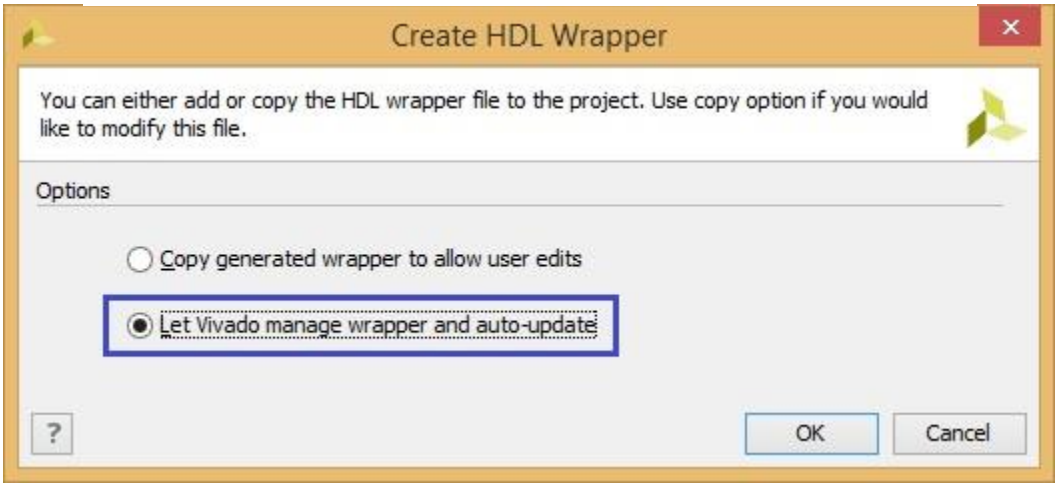

Figura 34. Crear HDL Wrapper.

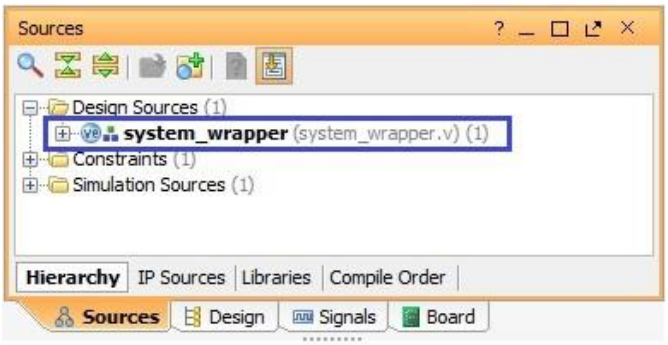

Figura 35. System Wrapper.

2. En el navegador de flujo, en la pestaña de *Program and debug*, dar click en *Generate bitstream (Ver* Figura 36*)*, posteriormente aparecerá un cuadro de dialogo en el cual se pide que guarde la modificación realizada, dar click en guardar.

Figura 36. Bitstream.

El proceso de *Generate bitstream* puede tardar algun tiempo, todo depende del rendimiento de la maquina, luego de que termine el proceso aparecera un cuadro de dialogo en el se debe de seleccionar *View reports*, y luego dar click en OK.

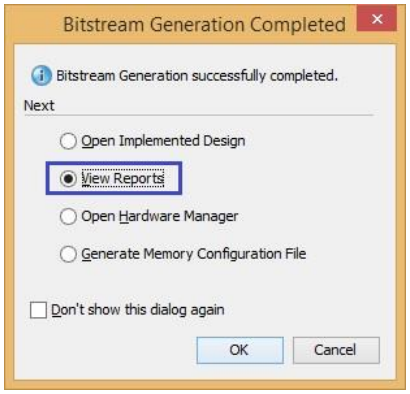

Figura 37. Generar Bitstream.

2.6. Exportar el diseño al SDK

1. Click en *file,* luego en *Export, Export hardware*, y en el cuadro de dialogo seleccione *Include bitstream*, y para finalizar click en OK.

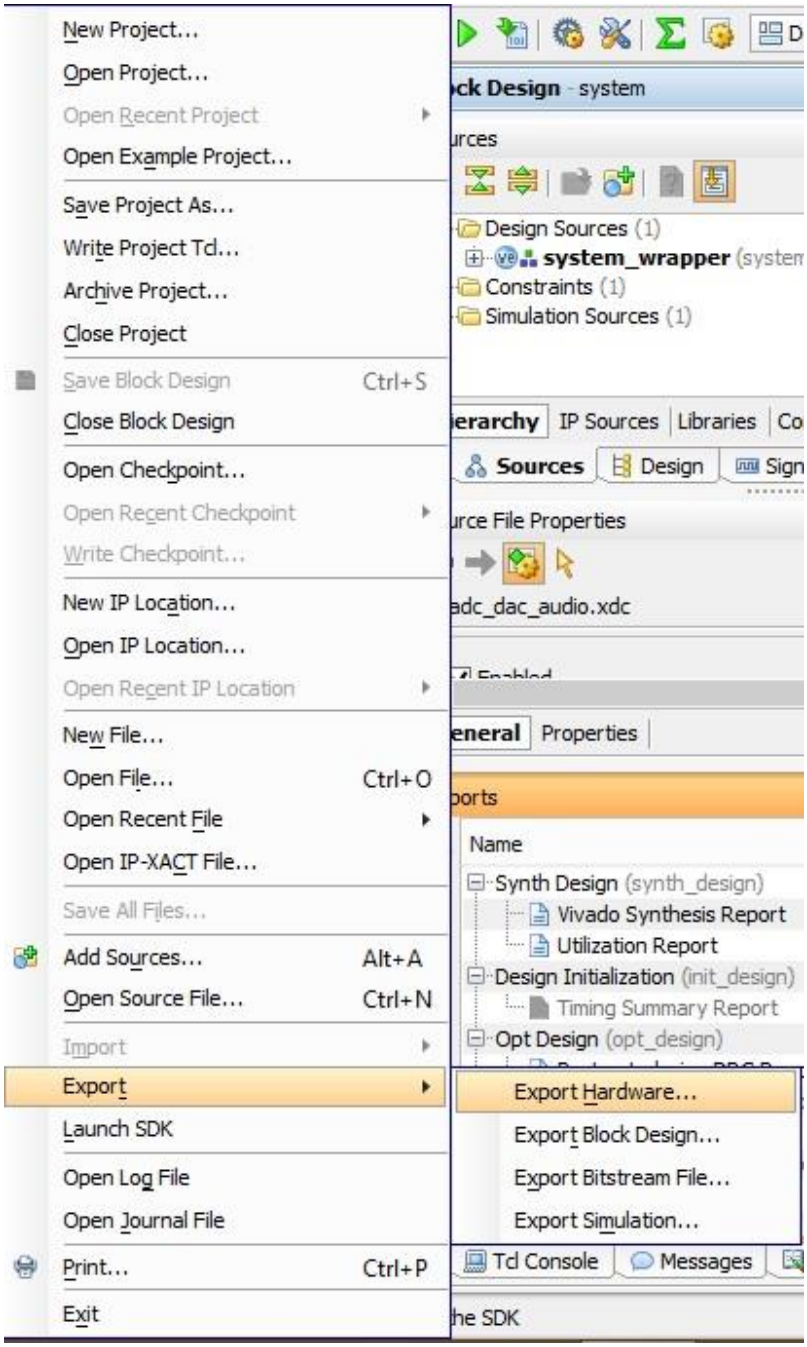

Figura 38. Proceso para exportar al SDK.

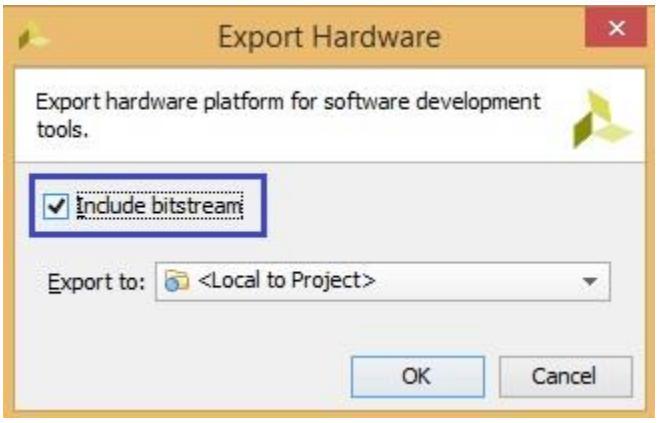

Figura 39. Incluir el bitstream al SDK.

2. Seleccione *file*, y *Launch SDK*, de esta manera se abrirá el SDK y podrá observar que todos los archivos relacionados con el diseño incluyendo los módulos IP se han exportado al SDK.

2.7. SDK

1. En el SDK, seleccionar *File, New, Application Project.*
2. En la siguiente ventana se deben de ingresar los parámetros tal cual como se muestra en la Figura 40.

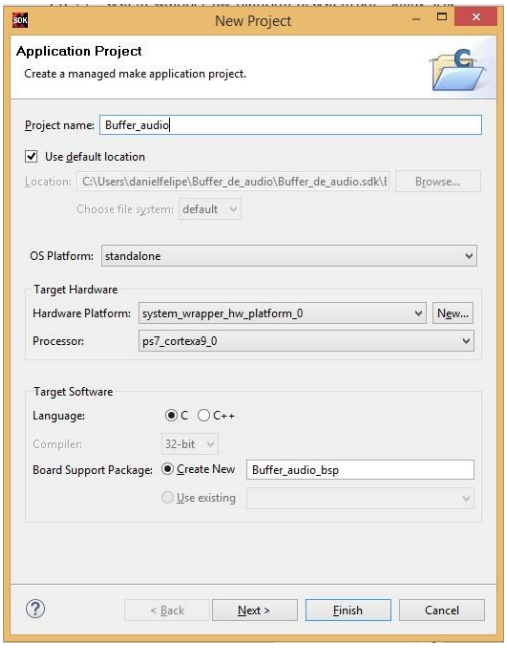

Figura 40. Parámetros del SDK.

En la siguiente ventana seleccionar *Empty application* y luego click en finalizar. Esto realizara el compilado del BSP.

3. Expandir *Buffer_audio*, click derecho sobre *src*, *New, Source file.*

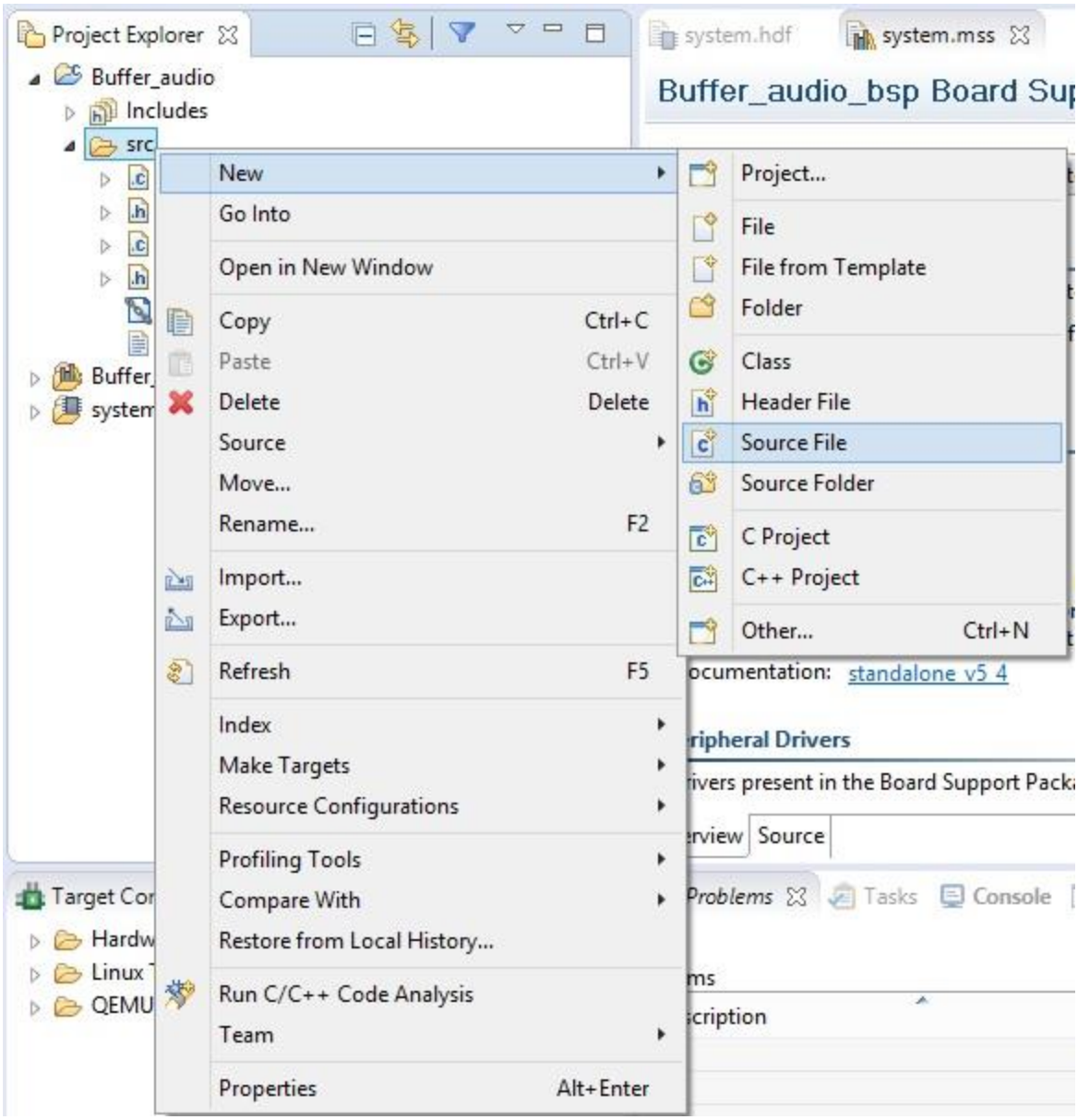

Figura 41. Creación de archivo fuente.

4. En la siguiente ventana, escriba *Audio_buffer.c* como *Source file,* y click en finalizar, de esta manera se creará un archivo en C vacío.

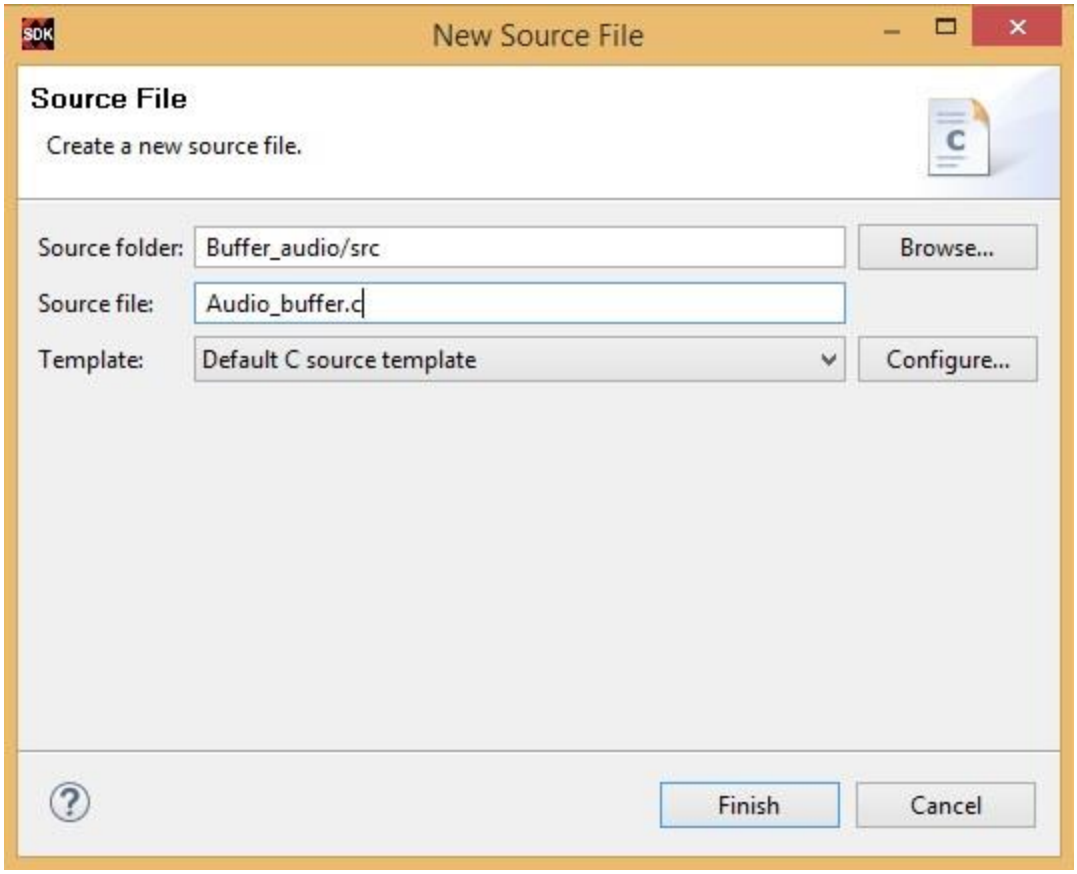

Figura 42. Crear Audio_buffer.c.

5. Expanda *Buffer_audio*, haga click derecho sobre *src*, *new y Header file.*

6. En la siguiente ventana, escriba *audio.h*, y luego click en finalizar, esto creara un archivo de cabecera vacío en el directorio de src.

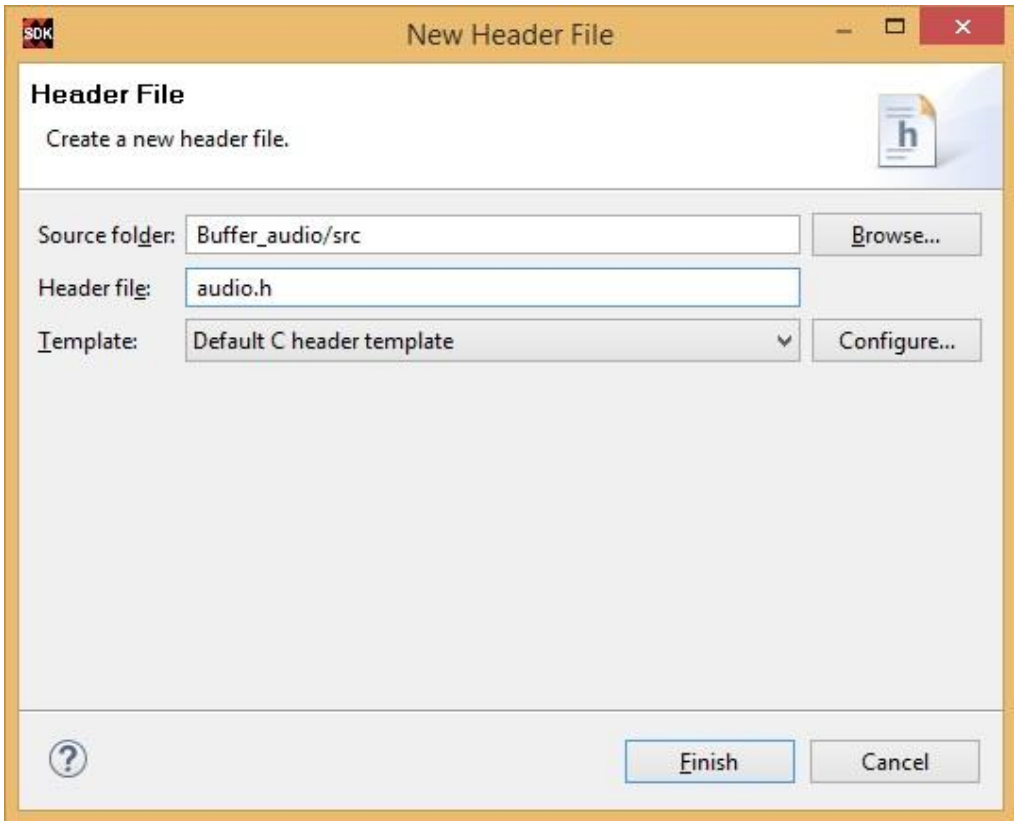

Figura 43. Archivo de cabecera.

7. Para finalizar debemos realizar doble click sobre audio.h y sobre Audio_Buffer.c, para luego realizar el copiado de los siguientes códigos.

Para audio.h:

```
#ifndef SRC_AUDIO_H_
#define SRC_AUDIO_H_
#endif /* SRC_AUDIO_H_ */
#ifndef __AUDIO_H_
#define __AUDIO_H_
#include "xparameters.h"
/* Redefine la dirección base del controlador de audio de xparameters.h */
#define AUDIO_BASE          XPAR_ZED_AUDIO_CTRL_0_BASEADDR
/*Dirección de esclavo para el controlador de audio ADAU*/
#define IIC_SLAVE_ADDR              0x70
/*I2C frecuencia de reloj serie en Hz*/
#define IIC_SCLK_RATE               400000
/*Registros internos del ADAU*/
enum audio_regs {
        R0_CLOCK_CONTROL                                =       0x00,
        R1_PLL_CONTROL                                  =       0x02,
        R2_DIGITAL_MIC_JACK_DETECTION_CONTROL           =       0x08,
        R3_RECORD_POWER_MANAGEMENT                      =       0x09,
        R4_RECORD_MIXER_LEFT_CONTROL_0                  =       0x0A,
        R5_RECORD_MIXER_LEFT_CONTROL_1                  =       0x0B,
        R6_RECORD_MIXER_RIGHT_CONTROL_0                 =       0x0C,
        R7_RECORD_MIXER_RIGHT_CONTROL_1                 =       0x0D,
        R8_LEFT_DIFFERENTIAL_INPUT_VOLUME_CONTROL       =       0x0E,
```

```
R9_RIGHT_DIFFERENTIAL_INPUT_VOLUME_CONTROL       =      0x0F,
R10_RECORD_MICROPHONE_BIAS_CONTROL               =      0x10,
R11_ALC_CONTROL_0                                =      0x11,
R12_ALC_CONTROL_1                                =      0x12,
R13_ALC_CONTROL_2                                =      0x13,
R14_ALC_CONTROL_3                                =      0x14,
R15_SERIAL_PORT_CONTROL_0                        =      0x15,
R16_SERIAL_PORT_CONTROL_1                        =      0x16,
R17_CONVERTER_CONTROL_0                          =      0x17,
R18_CONVERTER_CONTROL_1                          =      0x18,
R19_ADC_CONTROL                                  =      0x19,
R20_LEFT_INPUT_DIGITAL_VOLUME                    =      0x1A,
R21_RIGHT_INPUT_DIGITAL_VOLUME                   =      0x1B,
R22_PLAYBACK_MIXER_LEFT_CONTROL_0                =      0x1C,
R23_PLAYBACK_MIXER_LEFT_CONTROL_1                =      0x1D,
R24_PLAYBACK_MIXER_RIGHT_CONTROL_0               =      0x1E,
R25_PLAYBACK_MIXER_RIGHT_CONTROL_1               =      0x1F,
R26_PLAYBACK_LR_MIXER_LEFT_LINE_OUTPUT_CONTROL=      0x20,
R27_PLAYBACK_LR_MIXER_RIGHT_LINE_OUTPUT_CONTROL=     0x21,
R28_PLAYBACK_LR_MIXER_MONO_OUTPUT_CONTROL        =      0x22,
R29_PLAYBACK_HEADPHONE_LEFT_VOLUME_CONTROL       =      0x23,
R30_PLAYBACK_HEADPHONE_RIGHT_VOLUME_CONTROL =      0x24,
R31_PLAYBACK_LINE_OUTPUT_LEFT_VOLUME_CONTROL =     0x25,
R32_PLAYBACK_LINE_OUTPUT_RIGHT_VOLUME_CONTROL =    0x26,
R33_PLAYBACK_MONO_OUTPUT_CONTROL                 =      0x27,
R34_PLAYBACK_POP_CLICK_SUPPRESSION               =      0x28,
R35_PLAYBACK_POWER_MANAGEMENT                    =      0x29,
```

```
        R36_DAC_CONTROL_0                   =       0x2A,
        R37_DAC_CONTROL_1                   =       0x2B,
        R38_DAC_CONTROL_2                   =       0x2C,
        R39_SERIAL_PORT_PAD_CONTROL         =       0x2D,
        R40_CONTROL_PORT_PAD_CONTROL_0      =       0x2F,
        R41_CONTROL_PORT_PAD_CONTROL_1      =       0x30,
        R42_JACK_DETECT_PIN_CONTROL         =       0x31,
        R67_DEJITTER_CONTROL                =       0x36,
        R58_SERIAL_INPUT_ROUTE_CONTROL      =       0xF2,
        R59_SERIAL_OUTPUT_ROUTE_CONTROL     =       0xF3,
        R61_DSP_ENABLE                      =       0xF5,
        R62_DSP_RUN                         =       0xF6,
        R63_DSP_SLEW_MODES                  =       0xF7,
        R64_SERIAL_PORT_SAMPLING_RATE       =       0xF8,
        R65_CLOCK_ENABLE_0                  =       0xF9,
        R66_CLOCK_ENABLE_1                  =       0xFA
};
/*Registros de controladores de audio*/
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,
};
#endif
```

Para: Audio_Buffer_code:

```
#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"
#include "audio.h"
#include "xnco.h"

// Definición de parámetros.
#define UART_BASEADDR XPAR_PS7_UART_1_BASEADDR
#define GPIO_BASE XPAR_GPIO_0_BASEADDR
#define GPIO_ID XPAR_GPIO_0_DEVICE_ID
#define NCO_ID XPAR_NCO_0_DEVICE_ID

//------------------------------------------------
// Funciones
//------------------------------------------------
void read_superpose_play();;
unsigned char gpio_init();
unsigned char IicConfig(unsigned int DeviceIdPS);
void AudioPllConfig();
void AudioWriteToReg(unsigned char u8RegAddr, unsigned char u8Data);
```

```
void AudioConfigureJacks();
void LineinLineoutConfig();

//Variables globales
XIicPs Iic;
XNco Nco;

int main(void)
{
  xil_printf("ADC/DAC y procesamiento de audio digital\r\n");
 //Configurar la estructura de datos de la IIC
IicConfig(XPAR_XIICPS_0_DEVICE_ID);
//Configurar el PLL del codificador de audio
AudioPllConfig();
//Configuracion de los puertos de entrada y de salida
//Llamado de la funcion LineInLineOutConfig() para la configuración que activa
//los jack de conexión
AudioConfigureJacks();
xil_printf("ADAU1761 configured\n\r");

/* Loop infinito para leer el audio del ADC y enviarlo nuevamente al DAC */
read_superpose_play();

return 1;
}
```

```
/* ------------------------------------------------------------------------ *
 *       read_superpose_play()                                              *
 * ------------------------------------------------------------------------ *
 * Esta función adquiere la señal monofónica L o R desde el ADC, luego es
 *almacenada en un buffer y posteriormente es enviada al DAC para su
reproducción
 * ------------------------------------------------------------------------ */

void read_superpose_play(void)
{
        unsigned int BUFSIZE = 4000000, i=0, n=0;
        u32 in_left, /*in_right, out_left, out_right,*/ inbuff32[i], outbuff32[n];

//      while (!XUartPs_IsReceiveData(UART_BASEADDR)){


                for(i=0;i<BUFSIZE;i++)
                        {
                                in_left = Xil_In32(I2S_DATA_RX_L_REG);
                //              printf("%d\n",in_left);
                                inbuff32[i]=in_left;
                //              printf("%d\n",inbuff32[i]);
                //              printf("%d\n",i);
                                if(i>=BUFSIZE)
                                {
                                        i=0;
                                }
```

```
                }

        for(n=0;n<BUFSIZE;n++)
                {
                        outbuff32[n]=inbuff32[n];
                        Xil_Out32(I2S_DATA_TX_L_REG, outbuff32[n]);
                        if(n>=BUFSIZE)
                        {
                                n=0;
                        }
//                      printf("%d\n",outbuff32[n]);
                }
        }

/* ---------------------------------------------------------------------- *
 *                      IicConfig()                                       *
 * ---------------------------------------------------------------------- *
 * Esta función inicializa el controlador IIC buscando la configuración en la tabla
 * de configuración. También establece la frecuencia de reloj en la serie del IIC
 * ---------------------------------------------------------------------- */
unsigned char IicConfig(unsigned int DeviceIdPS)
{
        XIicPs_Config *Config;
        int Status;

        /* Inicialización del controlador IIC para que pueda usarcé */
```

```
	// Búsqueda de la configuración en la tabla
	Config = XIicPs_LookupConfig(DeviceIdPS);
	if(NULL == Config) {
		return XST_FAILURE;
	}

	// Inicialización de la configuración del controlador IIC
	Status = XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress);
	if(Status != XST_SUCCESS) {
	return XST_FAILURE;
	}

	//Establezca la frecuencia de reloj en serie del IIC
	XIicPs_SetSClk(&Iic, IIC_SCLK_RATE);

	return XST_SUCCESS;
}

/* ---------------------------------------------------------------------- *
 *                          AudioPllConfig()                              *
 * ---------------------------------------------------------------------- *
 * Configura el PLL interno de los codificadores de audio. Con MCLK = 10 MHz
 * configura el PLL para una frecuencia VCO = 49,152 MHz, y una frecuencia de
 * muestreo de audio de 48Khz
 * ---------------------------------------------------------------------- */
```

```
void AudioPllConfig() {

    unsigned char u8TxData[8], u8RxData[6];
    int Status;

    Status = IicConfig(XPAR_XIICPS_0_DEVICE_ID);
    if(Status != XST_SUCCESS) {
        xil_printf("\nError initializing IIC");

    }

    // Desactivar el reloj central
    AudioWriteToReg(R0_CLOCK_CONTROL, 0x0E);

    /*  MCLK = 10 MHz
        R = 0100 = 4, N = 0x02 0x3C = 572, M = 0x02 0x71 = 625
        PLL salida requerida   = 1024x48 KHz
        (PLLout)                        = 49.152 MHz
        PLLout/MCLK                     = 49.152 MHz/10 MHz
                                        = 4.9152 MHz
                                        = R + (N/M)
                                        = 4 + (572/625) */

    // Escribe 6 bytes en la dirección del registro R1 0x4002
    u8TxData[0] = 0x40; // Registrar dirección de escritura [15:8]
    u8TxData[1] = 0x02; // Registrar dirección de escritura [7:0]
    u8TxData[2] = 0x02; // byte 6 - M[15:8]
```

```
u8TxData[3] = 0x71; // byte 5 - M[7:0]
u8TxData[4] = 0x02; // byte 4 - N[15:8]
u8TxData[5] = 0x3C; // byte 3 - N[7:0]
u8TxData[6] = 0x21; // byte 2 - 7 = reservado, bits 6:3 = R[3:0], 2:1 = X[1:0], 0 = PLL modo de operación
u8TxData[7] = 0x01; // byte 1 - 7:2 = reservado, 1 = PLL Lock, 0 = Activación del reloj central

// Escribir bytes en PLL registro de control R1 @ 0x4002
XIicPs_MasterSendPolled(&Iic, u8TxData, 8, (IIC_SLAVE_ADDR >> 1));
while(XIicPs_BusIsBusy(&Iic));

// Registro de dirección 0x4002
u8TxData[0] = 0x40;
u8TxData[1] = 0x02;

do {
        XIicPs_MasterSendPolled(&Iic, u8TxData, 2, (IIC_SLAVE_ADDR >> 1));
        while(XIicPs_BusIsBusy(&Iic));
        XIicPs_MasterRecvPolled(&Iic, u8RxData, 6, (IIC_SLAVE_ADDR >> 1));
        while(XIicPs_BusIsBusy(&Iic));
}
while((u8RxData[5] & 0x02) == 0); // Mientras no está bloqueado

AudioWriteToReg(R0_CLOCK_CONTROL, 0x0F);      // 1111
```

```
        // bit 3:       CLKSRC = PLL Entrada de reloj
        // bits 2:1:    INFREQ = 1024 x fs
        // bit 0:               COREN = Core Clock habilitado
}

/* ------------------------------------------------------------------------ *
 *                          AudioWriteToReg                                 *
 * ------------------------------------------------------------------------ *
 * Función para escribir un byte en uno de los registros del controlador de
 * audio.
 * ------------------------------------------------------------------------ */
void AudioWriteToReg(unsigned char u8RegAddr, unsigned char u8Data) {

unsigned char u8TxData[3];
u8TxData[0] = 0x40;
u8TxData[1] = u8RegAddr;
u8TxData[2] = u8Data;

XIicPs_MasterSendPolled(&Iic, u8TxData, 3, (IIC_SLAVE_ADDR >> 1));
while(XIicPs_BusIsBusy(&Iic));
}
```

```
/* ------------------------------------------------------------------------ *
 *                          AudioConfigureJacks()                           *
 * ------------------------------------------------------------------------ *
 * Configura los diversos mezcladores del codec de audio, ADC, DAC y amplificadores
 * para aceptar la entrada mono y la salida al jack de audio.
 * ------------------------------------------------------------------------ */
void AudioConfigureJacks()
{
AudioWriteToReg(R4_RECORD_MIXER_LEFT_CONTROL_0, 0x01); // habilitar el mixer 1
AudioWriteToReg(R5_RECORD_MIXER_LEFT_CONTROL_1, 0x07); // activar el canal izquierdo de la línea y ajustar la ganancia a 6db
AudioWriteToReg(R6_RECORD_MIXER_RIGHT_CONTROL_0, 0x01);// habilitar el mixer 2
AudioWriteToReg(R7_RECORD_MIXER_RIGHT_CONTROL_1, 0x07);// activar el canal derecho de la línea y ajustar la ganancia a 6db
AudioWriteToReg(R19_ADC_CONTROL, 0x13); //habilitación de los codec.
AudioWriteToReg(R22_PLAYBACK_MIXER_LEFT_CONTROL_0, 0x21); //Activar el code izquierdo y habilitar el mixer 3
AudioWriteToReg(R24_PLAYBACK_MIXER_RIGHT_CONTROL_0, 0x41); //Activar el code derecho y habilitar el mixer 4
AudioWriteToReg(R26_PLAYBACK_LR_MIXER_LEFT_LINE_OUTPUT_CONTROL,
0x05); //Silenciar Mxr 3 en Mxr 5 y ajustar la ganancia a 6db; Habilitar mxr 5
AudioWriteToReg(R27_PLAYBACK_LR_MIXER_RIGHT_LINE_OUTPUT_CONTROL,
0x11); //Activar Mxr 4 en Mxr 6 y ajustar la ganancia a 6db; Habilitar mxr 6
```

```
AudioWriteToReg(R29_PLAYBACK_HEADPHONE_LEFT_VOLUME_CONTROL,
0xFF);//Mute Canal izquierdo del puerto HP (LHP)
AudioWriteToReg(R30_PLAYBACK_HEADPHONE_RIGHT_VOLUME_CONTROL,
0xFF); //Mute Canal derecho del puerto HP (LHP)
AudioWriteToReg(R31_PLAYBACK_LINE_OUTPUT_LEFT_VOLUME_CONTROL,
0xFE); //Establecer el volumen LOUT (0db); Activar el canal izquierdo del puerto de
salida de línea; Establecer el puerto de salida de línea a modo de salida de línea
AudioWriteToReg(R32_PLAYBACK_LINE_OUTPUT_RIGHT_VOLUME_CONTROL,
0xFE); // Establecer ROUT volumen (0db); Activar el canal derecho del puerto de
salida de línea; Establecer el puerto de salida de línea a modo de salida de línea
AudioWriteToReg(R35_PLAYBACK_POWER_MANAGEMENT, 0x03); // Habilitar la
reproducción de canales izquierdo y derecho.
AudioWriteToReg(R36_DAC_CONTROL_0, 0x03); //Habilitar ambos DAC
AudioWriteToReg(R58_SERIAL_INPUT_ROUTE_CONTROL, 0x01); //Conecte la
salida del puerto serie I2S (SDATA_O) a los DAC
AudioWriteToReg(R59_SERIAL_OUTPUT_ROUTE_CONTROL, 0x01); //Conecte la
entrada del puerto serie I2S (SDATA_I) a los ADC
AudioWriteToReg(R65_CLOCK_ENABLE_0, 0x7F); //Habilitar los reloj.
AudioWriteToReg(R66_CLOCK_ENABLE_1, 0x03); //Activar el resto de relojes
}
/* ------------------------------------------------------------------------ *
 *                          LineinLineoutConfig()                           *
 * ------------------------------------------------------------------------ *
 * Configurar la entrada ADC, DAC, Line_out y HP_Out.
 * ------------------------------------------------------------------------ */
void LineinLineoutConfig() {
AudioWriteToReg(R17_CONVERTER_CONTROL_0, 0x05);//48 KHz
```

```
AudioWriteToReg(R64_SERIAL_PORT_SAMPLING_RATE, 0x05);//48 KHz
AudioWriteToReg(R19_ADC_CONTROL, 0x13);
AudioWriteToReg(R36_DAC_CONTROL_0, 0x03);
AudioWriteToReg(R35_PLAYBACK_POWER_MANAGEMENT, 0x03);
AudioWriteToReg(R58_SERIAL_INPUT_ROUTE_CONTROL, 0x01);
AudioWriteToReg(R59_SERIAL_OUTPUT_ROUTE_CONTROL, 0x01);
AudioWriteToReg(R65_CLOCK_ENABLE_0, 0x7F);
AudioWriteToReg(R66_CLOCK_ENABLE_1, 0x03);
AudioWriteToReg(R4_RECORD_MIXER_LEFT_CONTROL_0, 0x01);
AudioWriteToReg(R5_RECORD_MIXER_LEFT_CONTROL_1, 0x05);//0 dB ganancia
AudioWriteToReg(R6_RECORD_MIXER_RIGHT_CONTROL_0, 0x01);
AudioWriteToReg(R7_RECORD_MIXER_RIGHT_CONTROL_1, 0x05);//0 dB ganancia
AudioWriteToReg(R22_PLAYBACK_MIXER_LEFT_CONTROL_0, 0x21);
AudioWriteToReg(R24_PLAYBACK_MIXER_RIGHT_CONTROL_0, 0x41);
AudioWriteToReg(R26_PLAYBACK_LR_MIXER_LEFT_LINE_OUTPUT_CONTROL,
0x03);//0 dB
AudioWriteToReg(R27_PLAYBACK_LR_MIXER_RIGHT_LINE_OUTPUT_CONTROL,
0x09);//0 dB
AudioWriteToReg(R29_PLAYBACK_HEADPHONE_LEFT_VOLUME_CONTROL,
0xE7);//0 dB
AudioWriteToReg(R30_PLAYBACK_HEADPHONE_RIGHT_VOLUME_CONTROL,
0xE7);//0 dB
AudioWriteToReg(R31_PLAYBACK_LINE_OUTPUT_LEFT_VOLUME_CONTROL,
0xE6);//0 dB
AudioWriteToReg(R32_PLAYBACK_LINE_OUTPUT_RIGHT_VOLUME_CONTROL,
0xE6);//0 dB
}
```

3. RESULTADOS Y DISCUSIÓN

Figura 44. Aplicación general.

Un DSP de audio se caracteriza por tener la capacidad de realizar la conversión de señales análogas a digitales para su procesamiento, adicional a esto también puede procesar estas señales, convertirlas nuevamente a análogas y sacarlas para su posterior reproducción, en los DSP se vuelve indispensable el uso de un elemento que permita realizar esta cantidad de operaciones sin llegar a saturar la capacidad de procesamiento, por esta razón es necesario el buffer de audio, para evitar que el sistema se sature y no se pueda obtener a la salida del DSP ningún resultado.

Cuando se trata de adquirir, convertir, procesar y reproducir alguna señal de audio, la Zed Board presenta perdidas al reproducir este audio, ocasionando así, la reproducción intermitente. Con la aplicación del buffer en el DSP se da solución a la perdida de los datos, logrando de esta manera conservar una fidelidad al audio adquirido inicialmente, por ejemplo, si se tiene una adquisición de 48000 muestras, las 48000 muestras procesadas podrán ser reproducidas correctamente.

El tamaño del buffer debe de ser calculado de manera tal que al reproducir el audio no se presenten fenómenos como el glitch o la latencia, en caso de presentarse alguno de estos fenómenos se debe aumentar o disminuir el tamaño del buffer.

4. CONCLUSIONES Y RECOMENDACIONES

Cuando el tamaño del buffer es muy grande el procesador de audio almacena una gran cantidad de muestras para el procesamiento, y se toma un tiempo mayor para poder realizar las próximas instrucciones, lo cual provoca un fenómeno llamado latencia.

Disminuir el tamaño del buffer, resuelve los inconvenientes de latencia, pero, cuando el buffer es tan pequeño y el procesador no tiene una gran capacidad entonces los datos se agotan rápidamente y el procesador no alcanza a buscar los datos adquiridos o no alcanza a vaciar el buffer para los datos que se van adquiriendo, esto provoca ruido que se escuchan como clicks, estos ruidos son denominados glitch.

Para lograr establecer un tamaño correcto para el buffer es necesario tener en cuenta algunas cosas:

- La realización de procesos complejos implica mayor tiempo de procesamiento. Si el procesador no tiene los recursos necesarios, se van a presentar interrupciones, intermitencias, en el flujo de datos. Para lograr un correcto funcionamiento entonces es necesario aumentar el tamaño del buffer.
- El aumento del tamaño del buffer provoca un aumento también en la latencia. El procesador va a procesar mayor cantidad de datos antes de poder pasar a la siguiente instrucción, lo cual conlleva a que aumente el tiempo del proceso. La solución a este inconveniente es disminuir el tamaño del buffer.

Todos los procesadores requieren de un buffer para poder almacenar temporalmente algunos datos mientras el procesador se encarga de realizar otras funciones como lo es la adquisición de los datos, configuración de los puertos, procesamiento de los datos adquiridos y por ultimo las conversiones ADC y DAC, debido a que el procesador no es capaz de realizar esta cantidad de tareas simultáneamente, el sistema necesita almacenar algunos datos temporales para posteriormente ser tratadas.

Los buffers de menor tamaño siempre serán los más preferidos, esto debido a la relación directa que existe entre el tamaño del buffer con la latencia. A mayor medida del buffer, la latencia será también mayor y consigo entonces el retraso en la reproducción del audio será mayor.

Al inicio del proyecto se presentaron inconvenientes al tratar de adquirir la señal, tratarla y posteriormente, sacarla hacia los altavoces, el inconveniente que se presento fue la intermitencia

en el resultado final, este inconveniente pudo ser mejorado con la aplicación de un buffer el cual optimizó los recursos del procesador para que estuviera en capacidad de adquirir la señal en un momento, posteriormente tratarla y finalmente enviar el resultado hacia la salida.

Con la aplicación del buffer se debe de tener muy en cuenta la cantidad de muestras que se requieren almacenar momentáneamente para darle tiempo al procesador de hacer las tareas de configuración y procesamiento, la cantidad de muestras deben de suficientes para los diferentes procesos del procesador, pero a su vez también no pueden ser tan grandes por la latencia que se le introduciría al sistema.

La fidelidad en términos de audio es muy importante y al tener un DSP es necesario pensar en la cantidad de datos que se pueden perder con la constante conversión de análogo a digital y digital análogo, por esto es necesario la aplicación de buffer de audio, de esta manera la perdida de datos no será considerable a la hora de su reproducción.

REFERENCIAS

Embedded Centric. (2017). *ADC/DAC and Digital Audio Processing*. [online] Available at: https://embeddedcentric.com/adc-dac-and-digital-audio-processing/ [Accessed 9 Mar. 2017].

Alexandridis, N. (1995). *Design of microprocessor-based systems*. 1st ed. Singapore: Simon & Schuster.

Crockett, L. (2014). *The zynq book : embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. 1st ed. Glasgow, Scotland: Strathclyde Academic Media.

Inacio, C. and Ombres, D. (1996). *The DSP decision*. 1st ed. Estados Unidos: The Institute of Electrical and Electronics Engineers, Inc-IEEE.

Lapsley, P., Bier, J., Shoham, A. and Lee, E. (2013). *DSP Processor Fundamentals*. 1st ed. Hoboken: Wiley-IEEE Press [Imprint].

Latencia., B. (2017). *Buffer y Latencia.*. [online] Audio-cfp.blogspot.com.co. Available at: http://audio-cfp.blogspot.com.co/2013/03/buffer-y-latencia.html [Accessed 9 Mar. 2017].

Sima, D., Fountain, T. and Kacsuk, P. (2004). *Advanced computer architectures*. 1st ed. Delhi.

Smith, S. (2011). *The scientist and engineer's guide to digital signal processing*. 1st ed. [S.l.: s.n.].

FIRMA ESTUDIANTES_____

_____

_____

FIRMA ASESOR _____

FECHA ENTREGA: 21 de Marzo de 2017

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____