IMPLEMENTACIÓN DE FILTROS DIGITALES SOBRE FPGA USANDO VIVADO HLS

Juan Pablo Puerta Orrego

Ingeniería Electrónica

David Andrés Márquez Viloria

INSTITUTO TECNOLÓGICO METROPOLITANO

2017

RESUMEN

Este trabajo de grado tiene como objetivo la implementación de filtros digitales sobre un sistema de desarrollo ZedBoard, usando el entorno de desarrollo Vivado High Level Synthesis (HLS). El aporte principal de este trabajo es la realización de un tutorial sobre la implementación de algoritmos de procesamiento digital de señales sobre FPGA. Específicamente se realizó la implementación de filtros FIR para señales unidimensionales. Las implementaciones se realizaron sobre señales de frecuencia modulada lineal, también conocida como Chirp, que es una onda sinusoidal simple que aumenta su frecuencia con el tiempo, permitiendo hacer un análisis fácil del funcionamiento del filtro implementado debido a que podemos ver el resultado de la aplicación del filtro también en el dominio del tiempo.

Palabras clave: VIVADO HLS, FPGA, DSP, FILTROS, CHIRP

TABLA DE CONTENIDO

1. INTRODUCCIÓN

El flujo de diseño de sistemas digitales basados en lenguajes de descripción de hardware (HDL) implica un desarrollo extenso en cuanto a la construcción de los algoritmos para una aplicación determinada. Xilinx ha desarrollado una herramienta de síntesis de alto nivel (HLS) para producción de soluciones en FPGAs, llamada Vivado HLS. Esta herramienta evoluciona la programación HDL a lenguajes HLS, con lo que otorga el beneficio a los diseñadores de trabajar con alto nivel de abstracción y optimización de algoritmos. Esto supone una mayor rapidez de diseño evitando entrar en detalles de implementación, dado que su metodología de trabajo se basa en transformar descripciones C a implementaciones RTL para sintetizarlas en una FPGA.

Además, Vivado HLS ofrece un conjunto de directivas para la síntesis con el propósito de optimizar el código sin necesidad de modificarlo y encontrar así la implementación óptima del diseño. En cuanto a la comunicación con el procesador, Vivado HLS crea un conjunto de drivers en el momento de exportación en bloque IP de la implementación RTL realizada, que posteriormente son usados en la herramienta SDK para desarrollar el software para la comunicación.

El informe presenta el modo de uso de Vivado HLS mediante la implementación de filtros FIR óptimo (Equiripple) sobre una señal Chirp. El objetivo es mostrar cómo se realiza la implementación del algoritmo en Vivado HLS, adquirir el diseño RTL y llevarlo a Vivado HLx, donde se realiza el diseño de bloques para finalmente, en una aplicación en SDK (Software Development Kit), probar su funcionamiento comunicando el procesador con el periférico a través de la interfaz AXI4-Lite que los comunica mediante los drivers creados en el momento de exportación de Vivado HLS.

2. ESTADO DEL ARTE

Existe en la literatura muchos trabajos que realizan procesamiento digital de señales usando la metodología clásica en lenguaje HDL, estos trabajos buscan acelerar los algoritmos a través de la arquitectura reconfigurable de la FPGA. Un ejemplo de este tipo de trabajos es presentado en Carlo Safarian, B.K Mohanty (2015). FPGA Implementation of LMS-based FIR Adaptive Filter for Real Time Digital Signal Processing Applications. Este artículo estudia los diseños de filtros adaptativos LMS propuestos para la implementación en una FPGA y a partir de los diferentes diseños implementados, se propone un diseño de filtro adaptativo LMS utilizando el Xilinx DSP48.

Dentro del estudio se exploran técnicas de intercambio de recursos tales como multiplicar y acumular que se pueden utilizar para reducir el número total de uso de multiplicadores lo cual reduce espacio en la arquitectura del diseño. Los DSP48 tienen multiplicador acumulador (MAC), los cuales ofrecen ahorro de área y energía aplicando un recurso adicional en el desarrollo del filtro, a partir de esto se procede a la implementación de la propuesta.

La creación del filtro adaptativo se desarrolla con una sección de filtrado (Utiliza un filtro de respuesta de impulsa finito FIR) y otra sección de adaptación (Utiliza el algoritmo LMS). En cada iteración, la sección de filtrado calcula una salida de filtro a partir de la cual se calcula un valor de error. La sección adaptativa utiliza el valor de error para calcular el término de aumento de peso para actualizar el vector de peso para la siguiente iteración.
La arquitectura propuesta se implementa en Xilinx 7K325T FPGA. El componente DSP48 se utiliza para la implementación de la multiplicación y la adición. El diseño se simula usando la herramienta del generador del sistema Xilinx y el código HDL correspondiente sintetizado e implementado en FPGA usando el software Xilinx Vivado. La simulación se realizó utilizando el simulador Vivado.

Debido al a que el método de desarrollo tradicional en la programación de plataformas para el procesamiento de señales digitales es difícil y cuesta mucho trabajo y tiempo. En Lui Hanbo, Wang Shaojun, Zhang Yigang. 2015. Design of FIR filter with high level synthesis. Se diseña un filtro digital FIR utilizando VIVADO HLS, DSP Builder y LabVIEW FPGA. Allí se presentan las características, ventajas y desventajas de las herramientas de desarrollo HLS y las ventajas de análisis de alto nivel.

En el artículo se encuentra una breve introducción a cerca de las tres herramientas HLS (VIVADO HLS, DSP Builder y LabVIEW FPGA) que los principales fabricantes y desarrolladores de FPGA han lanzado. Partiendo de las características de cada una de las herramientas se implementa el diseño del filtro digital FIR, partiendo de los flujos de diseño de cada una de las herramientas.

Cuando se implementa el diseño del filtro digital FIR, HLS utiliza lenguajes de programación de alto nivel para el nivel del sistema y la descripción del nivel del algoritmo. Este método reduce el umbral de diseño, además, también obtiene resultados superiores.

La velocidad de HLS es más rápida. En otras palabras, el tiempo de síntesis e incluso el tiempo de desarrollo son más cortos. Como un proyecto real necesita muchas veces de depuración y compilación, y cada modificación de diseño necesita ser sintetizada de nuevo. Así que la velocidad de síntesis más rápida de HLS puede reducir en gran medida el tiempo de desarrollo.

Una variedad de opciones de optimización configuradas por las herramientas de desarrollo HLS es más fácil de mejorar la calidad del diseño y acortar el tiempo de desarrollo. Podemos encontrar que los métodos de HLS FIR filtro digital de diseño en este documento han sido alternativa a los métodos tradicionales de desarrollo FPGA.

3. METODOLOGÍA

Para mostrar el manejo de la herramienta de diseño digital de alto nivel VIVADO HLS, se diseñará un filtro digital tipo FIR, donde con la ayuda de Fdatool de MatLab se definen las constantes del filtro, se desarrollar el algoritmo en C con VIVADO HLS donde se aplican procesos de depuración, simulación, síntesis e implementación del diseño RTL para empaquetarlo en un bloque IP y posteriormente procesarlo en VIVADO DESIGN SUITE donde se hace el diseño VHDL adicionando el procesador y el periférico para finalmente crear una aplicación en SDK para probar su funcionamiento comunicando el procesador con el periférico a través de la interfaz AXI4-Lite que los comunica mediante los drivers creados en el momento de exportación de Vivado HLS.

En la figura 1 se muestra las etapas del diseño del filtro en lenguaje de alto nivel.

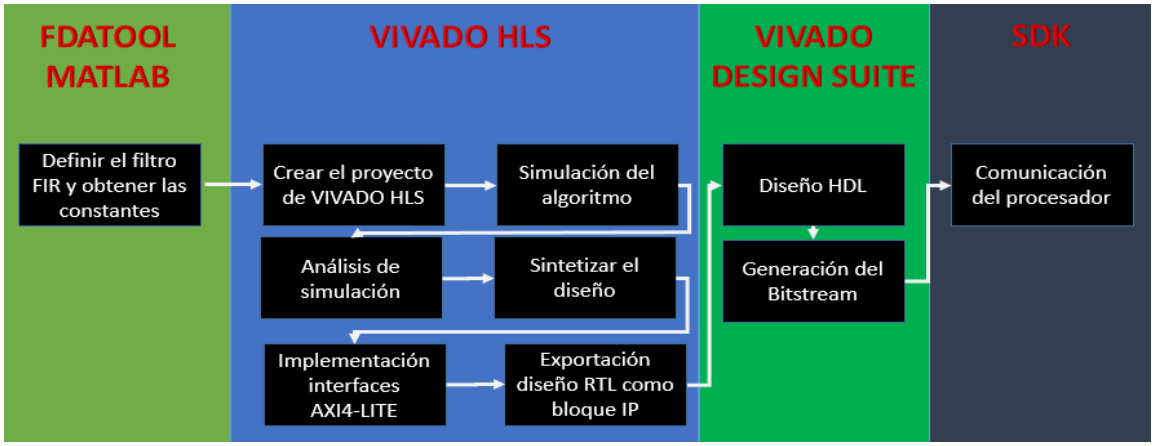

Figura 1 Fases del diseño del filtro FIR VIVADO HLS

3.1 Hardware y Software utilizados

3.1.1 Software

Para el desarrollo de trabajo, se utilizan los siguientes recursos tecnológicos:

- Software VIVADO HLS 2016
- Software VIVADO HLx 2016
- Matlab R2013B

- Software Development Kit (SDK)

3.1.2 Hardware

- PC
- Tarjeta de desarrollo ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1).

Por otra parte, se contó con el soporte técnico en infraestructura, desde los laboratorios con los recursos tecnológicos como PC, software necesario y el hardware donde se realizará la implementación; también se cuenta con el apoyo del docente que desde su conocimiento y experiencia aporta material relevante para el desarrollo del proyecto.

3.2 Especificaciones de diseño del filtro FIR

Un filtro FIR es un tipo de filtro digital que su respuesta ante una señal impulso de entrada es un número finitos de términos no nulos a su salida, siendo esta salida combinación lineal de las entradas.

La ecuación de salida de un filtro FIR es la siguiente

$$Y[n] = a_0.X[n] + a_1.X[n-1] + a_2.X[n-2] + \ldots + a_{n-1}.X\left[n-(n-1) + a_n.X[n-N]\right]$$

Ecuación 1. Ecuación salida del filtro FIR

Siendo $Y[n]$ la muestra de salida actual, $X[n]$ la muestra de entrada actual, $X[n-i]$ $i = 1 \ldots . N$ las muestras anteriores y los factores $a_1, a_2, \ldots . , a_N$ los coeficientes del filtro.

En la aplicación Fdatool de MatLab se establecen las especificaciones de diseño, donde se escoge realizar un filtro FIR pasa bajo de orden 30 utilizando el método equiriple. Este número de coeficientes se ha escogido bajo para no consumir desmasiados recursos de hardware, pues en las pruebas del diseño este ha sido un inconveniente debido a que en VIVADO HLS existe un límite de recursos disponibles como los DSP48, los cuales se emplean para operaciones como multiplicaciones o sumas. Aunque en VIVADO HLS existen métodos de optimización para reducir recursos, sería en otro trabajo que se enfatice sobre esto.

En la figura 2. Se muestra la selección de parámetros en *Fdatool*.

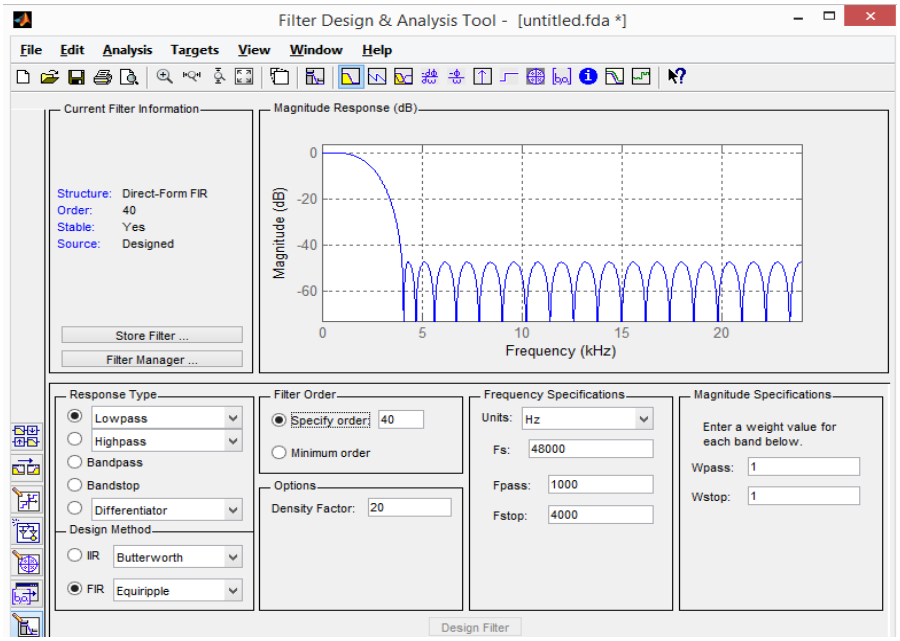

Figura 2 Parámetros del filtro en FdaTool de MatLab

Una vez diseñado el filtro, la herramienta de MatLab permite generar la cabecera en código C que servirá para la construcción del algoritmo en VIVADO HLS.

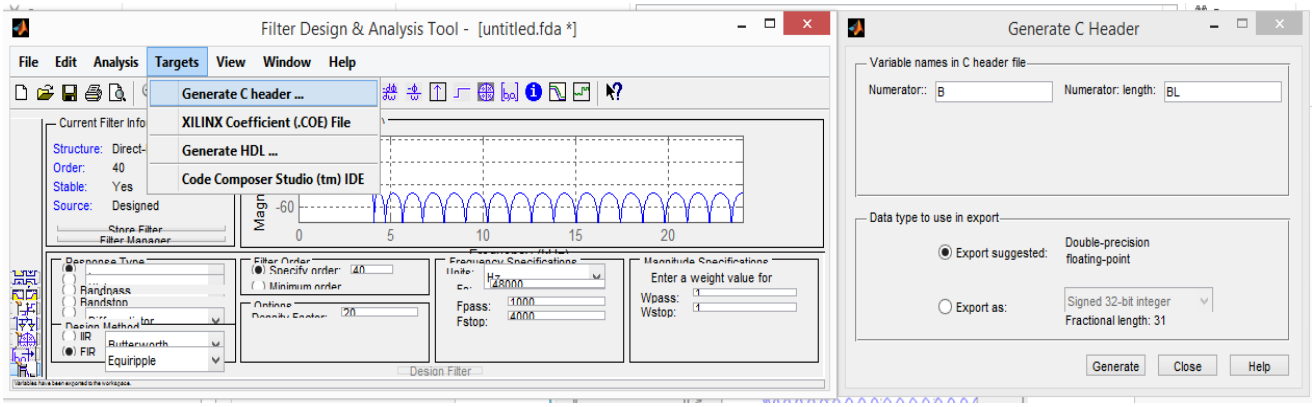

Figura 3 Procedimiento para generar constantes del filtro

Ya con las constantes obtenidas se puede diseñar el filtro FIR en VIVADO HLS.

3.3 Proyecto en VIVADO HLS

Una vez obtenidos las constantes del filtro como insumo principal para la implementación del diseño, se procede a crear el proyecto en VIVADO HLS e implementar el filtro digital

bajo programación en C. A continuación se presentan los pasos necesarios para lograr el desarrollo del diseño digital.

3.3.1 Asistente de creación nuevo proyecto VIVADO HLS

VIVADO HLS tiene como entrada principal una función (top-function) que permite escribirse en C, C++ o System C, adicionamente incluir el TestBench y ficheros.

El diseño del filtro se realizará en lenguaje C tanto para la función del filtro, el TestBench y el fichero de cabeceras. La herramienta tiene dos opciones de incluir cada entrada, la creación de un fichero en blanco para escribir el código correspondiente y la opción de disponer ficheros *.c y *.h ya creados para importarlos. Para esto se abre VIVADO HLS 2016.2 y en la pantalla inicial se escoge Create New Proyect, aparecerá un asistente que guiará al usuario durante la creación.

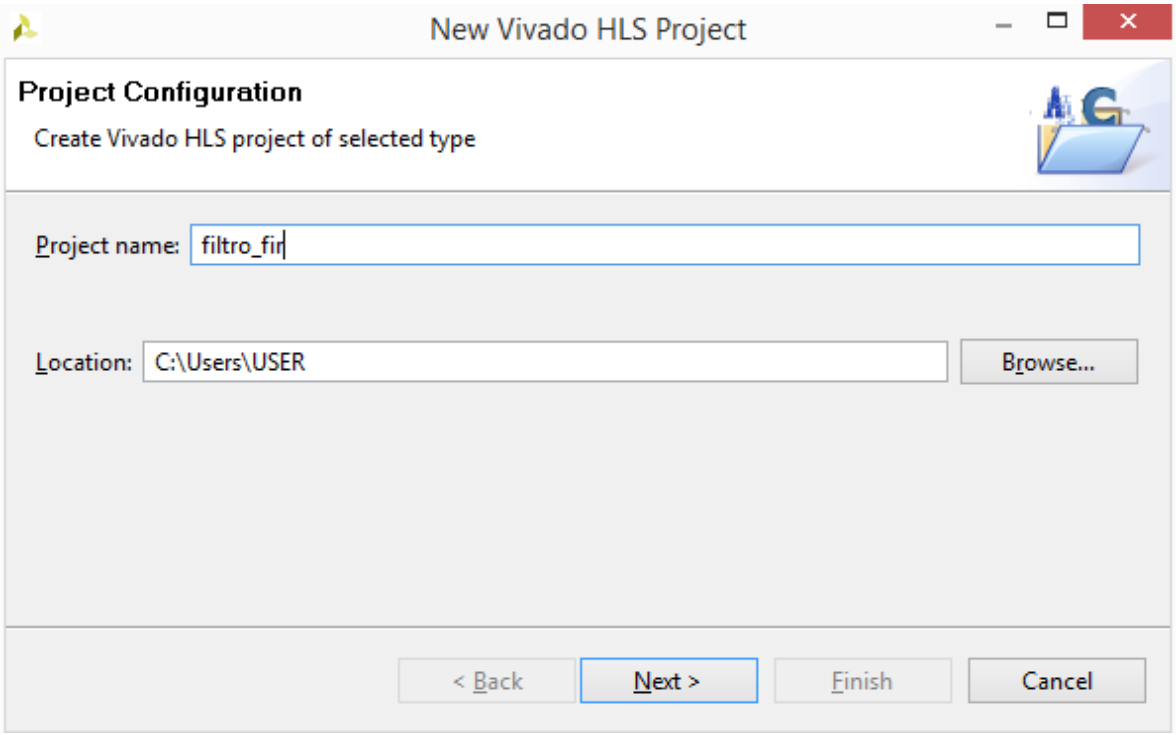

Figura 4 Nombre y directorio de nuevo proyecto

El siguiente recuadro pedirá inicialmente el nombre de la función principal, se tiene la opción de asociar un fichero ya construido o crear un nuevo archivo.

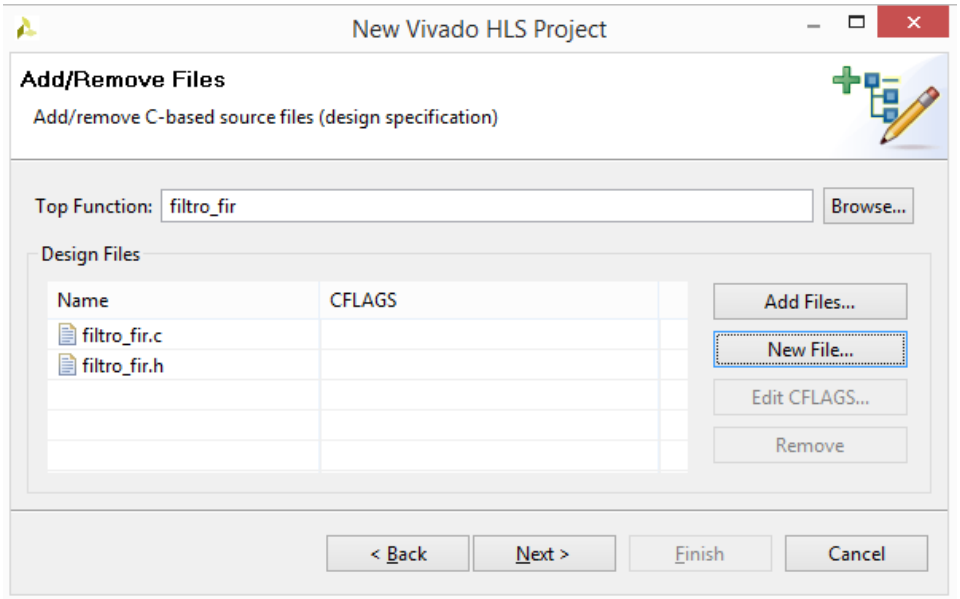

Figura 5 Crear o añadir fichero fuente de nuevo proyecto

Cuando se le da clic en *Next*, el asistente continúa con la incorporación de los ficheros correspondientes al *test bench.*

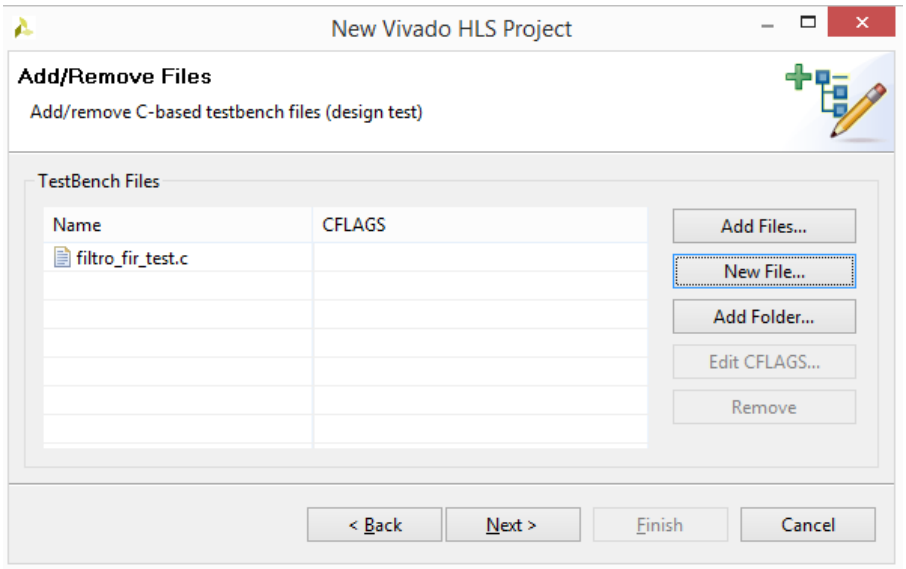

Figura 6 Crear o añadir fichero test de nuevo proyecto

Por último se debe seleccionar el nombre de la solución a analizar, el periodo, el margen de incertidumbre y la tarjeta donde se va a implementar. El periodo se deja por defecto 10nS, el margen de incertidumbre sin especificar es del 12,5% del periodo del reloj y para

este caso se elige la tarjeta *ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1).*

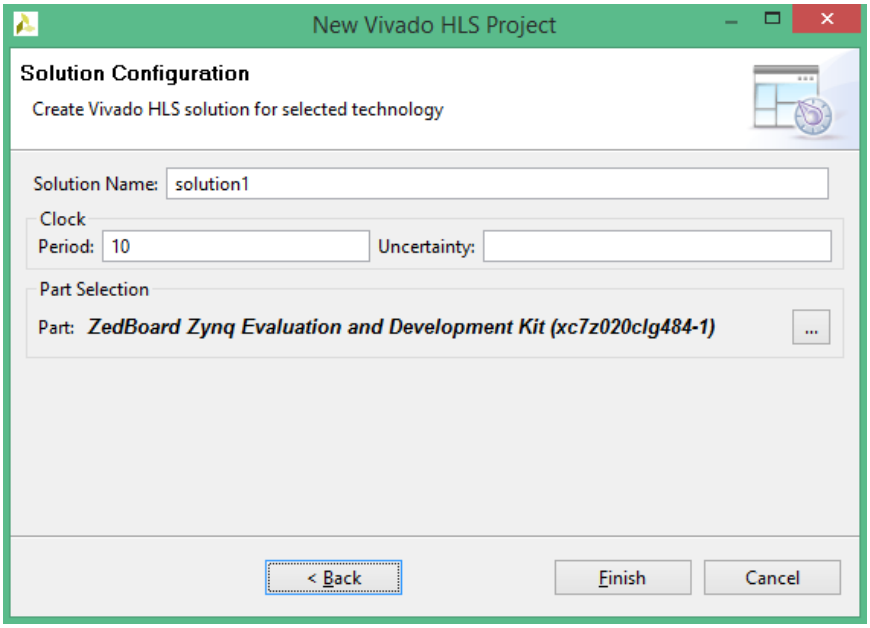

Figura 7 parámetros de nuevo proyecto

Finalizando el asistente, se visualiza la pantalla de trabajo bajo la interfaz de síntesis. Allí se puede visualizar en el explorador del proyecto, los ficheros que se crearon.

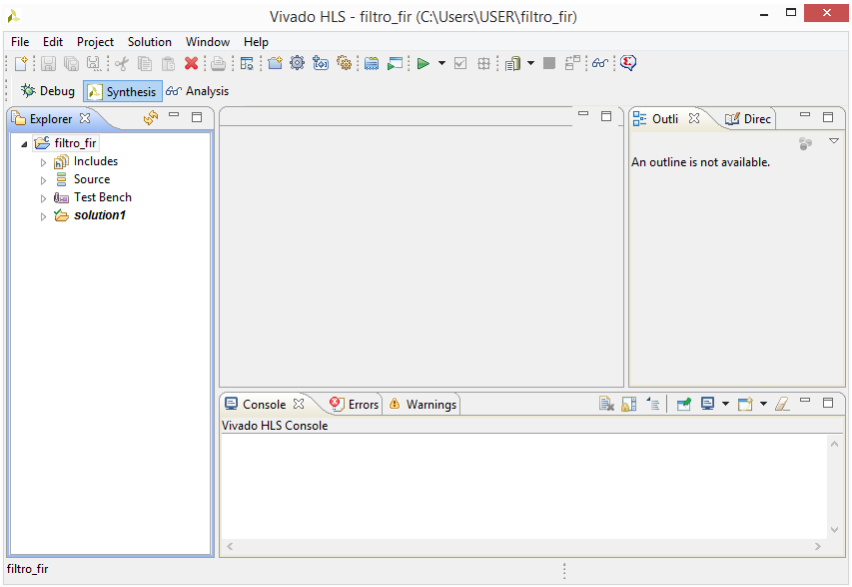

Figura 8 Interfaz de usuario para síntesis

La interfaz de usuario está dividida como muestra la figura 9

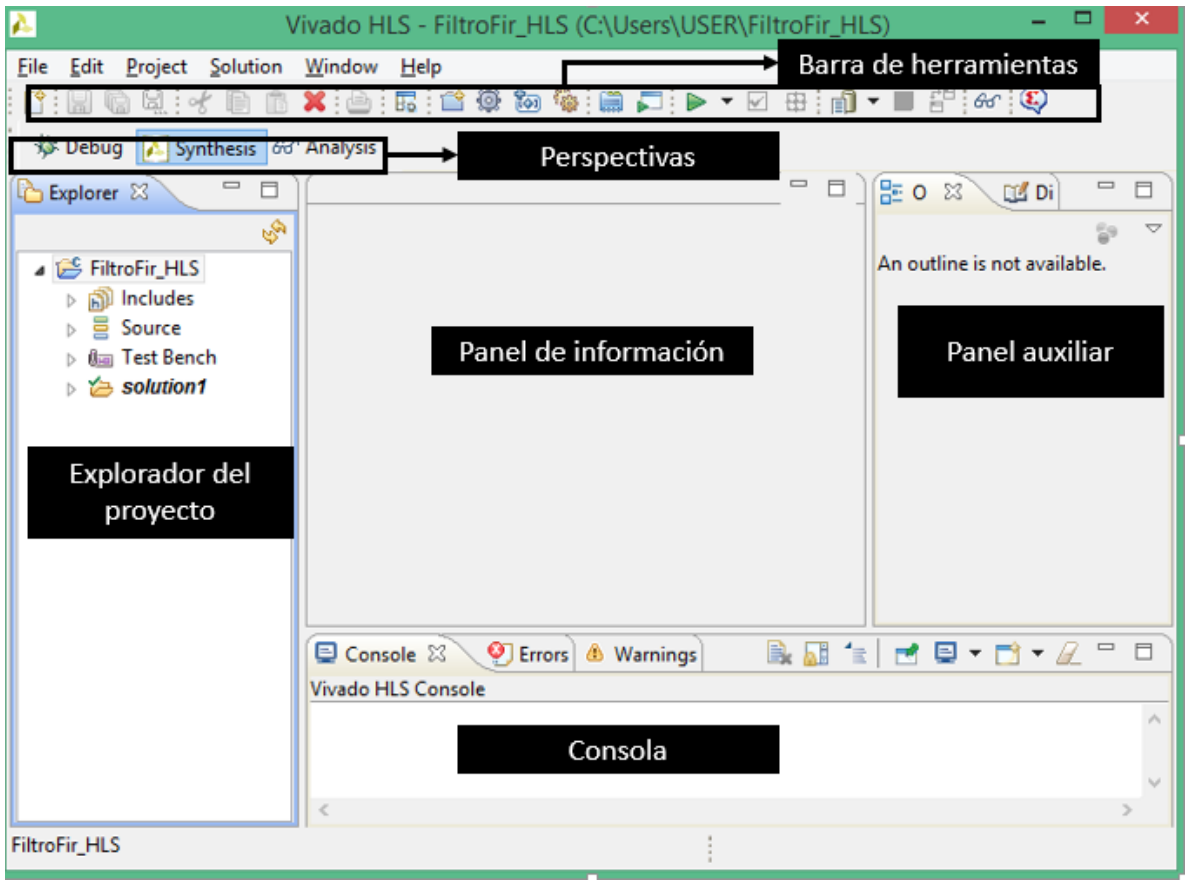

Figura 9 Interfaz de usuario divisiones

En esta instancia, se crea el algoritmo de la función principal, donde se procede a transformar la entrada y proporcionar la salida del filtro. Para ello, en *Source* del explorador del proyecto, se abre el archivo ".c", donde aparece la hoja en blanco donde se escribirá el código.

3.3.2 Algoritmo del filtro y test bench

Una vez desarrollado todo el asistente de creación de un nuevo proyecto, se inicial la programación del algoritmo de la función principal y el código de prueba en el test bench.

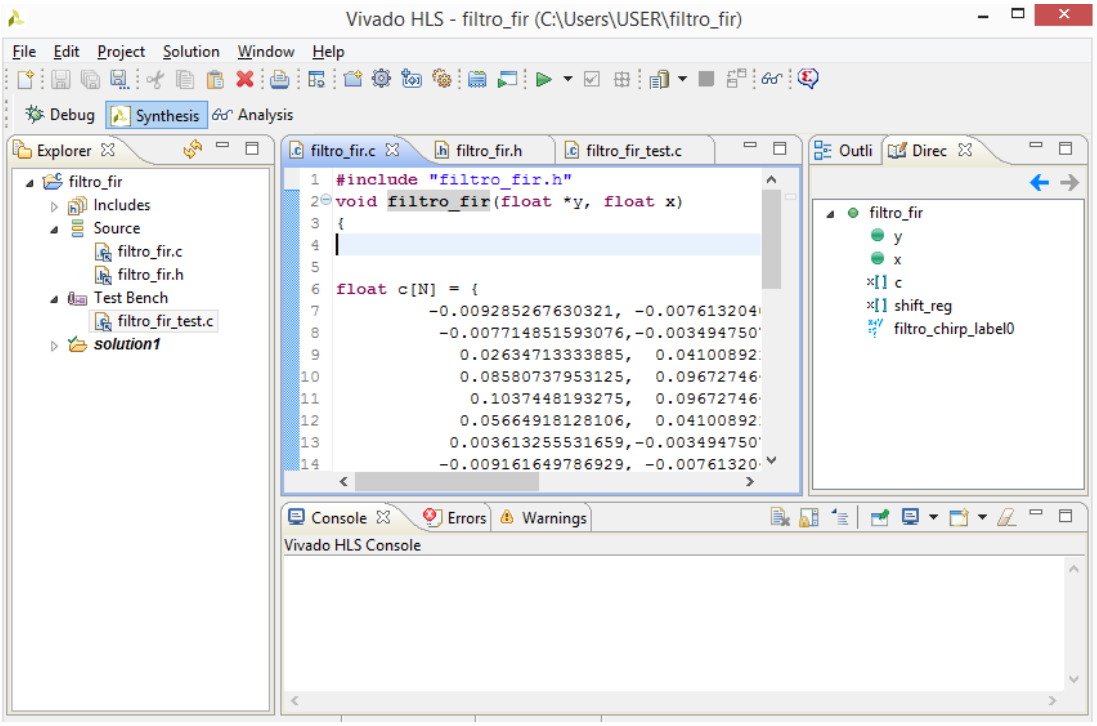

Figura 10 Contenido en panel de información, código filtro FIR

El código de la función principal se muestra a continuación, donde fue necesario crear un fichero de cabecera ".h", donde se declaran las entradas y salidas y el tamaño del orden del filtro. Para adicionar un fichero a la función se da clic derecho en *Source – New File* y se nombra el nuevo fichero como ".h".

Código de cabecera

```
#define N 41
void filtro_chirp (float *y, float x);
```

Código de función principal:

```
#include "filtro_chirp.h"
void filtro_chirp (float *y, float x)
{

float c[N] = {
0.001968686623771,-0.000767201033771,-0.001826826856408,
-0.003484029805512,-0.005544694520555,-0.007695410053516,
0.009498032244464,-0.01041392351443,-0.009858796819663,
-0.007280569104874,-0.002247859631425,0.00546484018849,
```

```
 0.01580923920211, 0.02842523411543, 0.04264241042767,
0.05752524322864,0.07195914816285,0.08476756281697,
0.09484492070986,0.101287477973,0.1035036098925,0.101287477973,
0.09484492070986,  0.08476756281697,0.07195914816285,0.05752524322864,
0.04264241042767,  0.02842523411543,0.01580923920211,0.00546484018849,
-0.002247859631425,-0.007280569104874,-0.009858796819663,
-0.01041392351443,-0.009498032244464,-0.007695410053516,
-0.005544694520555,-0.003484029805512,-0.001826826856408,
-0.000767201033771, 0.001968686623771};
static float shift_reg[N];
float acc;
float data;
int i;
acc=0;
//Shift_Accum_Loop:
filtro_chirp_label0:for (i=N-1;i>=0;i--)
{
if (i==0){
shift_reg[0]=x;
data = x;
}
else{
shift_reg[i]=shift_reg[i-1];
data = shift_reg[i];
}
acc+=data*c[i];
}
*y=acc;
}
```

El código anterior procesa la entrada “x”, haciendo la multiplicación ponderada de cada valor de la entrada con cada una de las constantes del filtro FIR halladas anteriormente con la herramienta FdaTool de MatLab, llevando el resultado a “*y”.

IMPORTANTE: Las salidas del algoritmo se declaran como apuntador (con “*” previo al nombre de la variable), ya que va a una dirección de memoria y para poder validar el código a partir del *test bench* debido a que este recoge el contenido del puntero de salida una vez haya finalizado la función.

Luego de la creación del código de la función del filtro, se construye el algoritmo de prueba en el *test bench* (fichero construido en el asistente de nuevo proyecto). A continuación se muestra el código correspondiente.

```
#include <stdio.h>
#include <math.h>
#include "filtro_fir.h"
int main ()
{
        float output;
        float signal=0;

float chirp[48001]={/*aquí las 48001 muestras de la señal Chirp*/};
        int i,j;
        j=0;
        for (i=0;i<48001;i++)
        {
        signal = chirp[j];
        j=j+1;
        fprintf(fp,"%f ",output);
        // Ejecuta la función con la última señal
        filtro_fir(&output,signal);
        // Imprime resultados.
        printf("%f\n",output);
        }
        return 0;
}
```

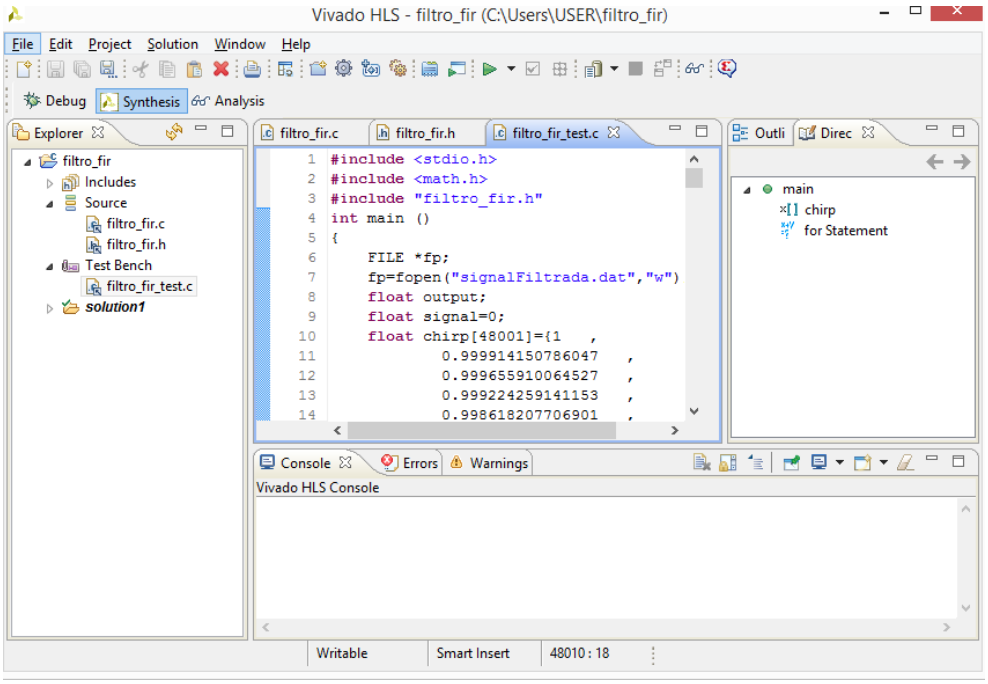

Figura 11 Contenido en panel de información, código test bench

El *test bench* adoptado bajo la finción main, llama a la función principal pasándole como argumento cada muestra de la señal de entrada, en este caso es la señal Chirp que se declara completa, y muestra el resultado obtenido luego del procesamiento de la señal en la función principal.

3.3.3 Simulación del algoritmo

El paso que sigue luego de construir tanto el algoritmo de la función del filtro como el *test bench* es realizar la simulación del diseño, esto con el fin de verificar si el código C realizado para el filtro responde de forma adecuada a las muestras de entrada que se van introduciendo.

La simulación se aplica desde el botón *Run C Simulation* situado en la barra de herramientas. Allí aparecerá la ventana que se muestra en la Figura 16, donde no se selecciona ninguna opción para que el resultado lo muestre en la consola. Las opciones que aparecen en la ventana son perspectivas de depuración, que en el caso de un resultado erróneo serviría de ayuda al usuario para corregir el problema.

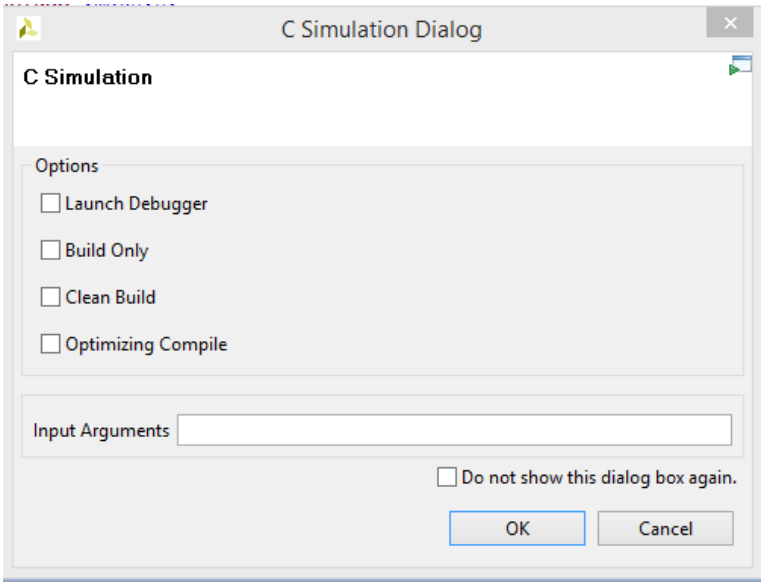

Figura 12 Simulación del diseño

Una vez ejecutada la simulación se genera en el explorador del proyecto la carpeta *csim* con la información de la simulación; también en la consola aparece el resultado de la simulación y el mensaje de la ejecución con o sin errores.

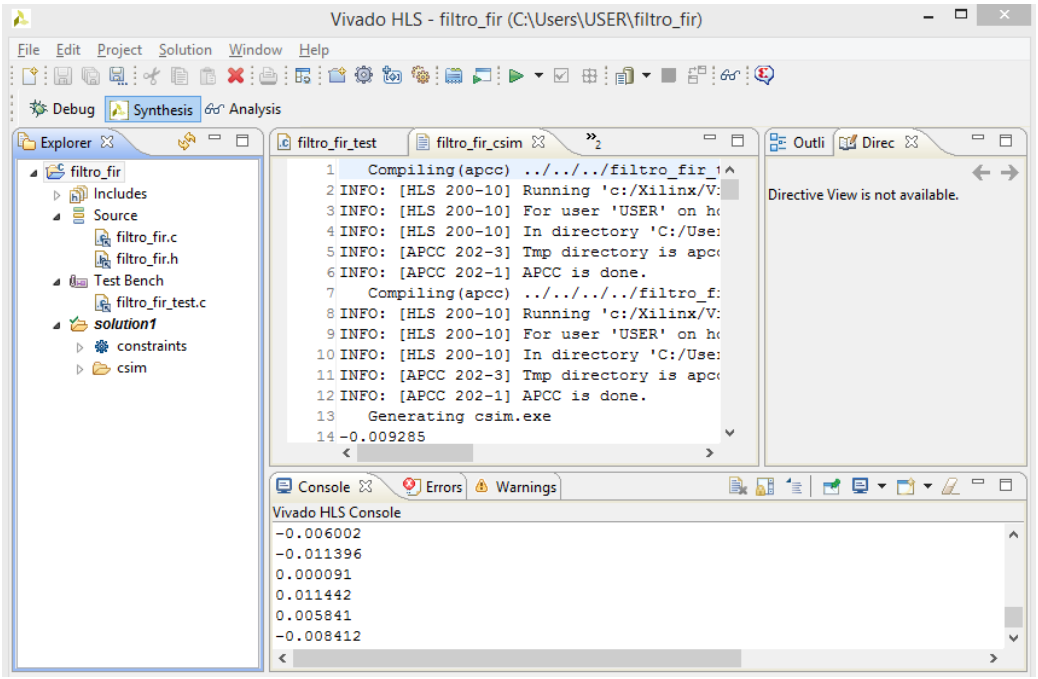

Figura 13 Resultado de simulación

En este punto se lleva el resultado a MatLab para validar si el filtro funciona correctamente y se compara con el generado en Fdatools.

En la figura 14 se muestra la respuesta del filtro aplicando la función *Filter* de Matlab a una señal *Chirp* con las constantes previamente generadas en Fdatools.

La figura 15 muestra la respuesta del filtro graficando los valores obtenidos de la simulación del diseño en VIVADO HLS.

La figura 16 y 17 muestra la comparación del filltro de Matlab y el filtro de VIVADO HLS donde se observa el resultado esperado del algoritmo.

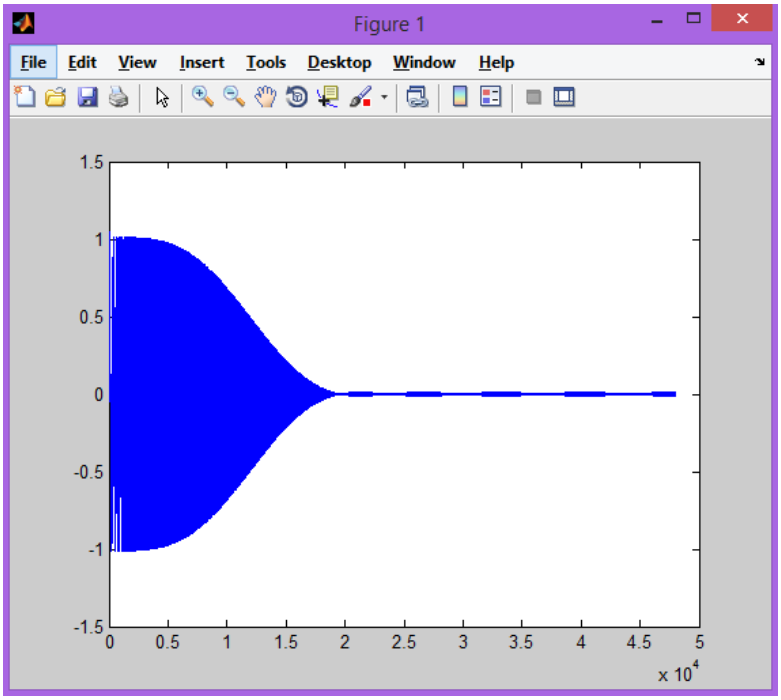

Figura 14 Función Filter de Matlab en señal Chirp

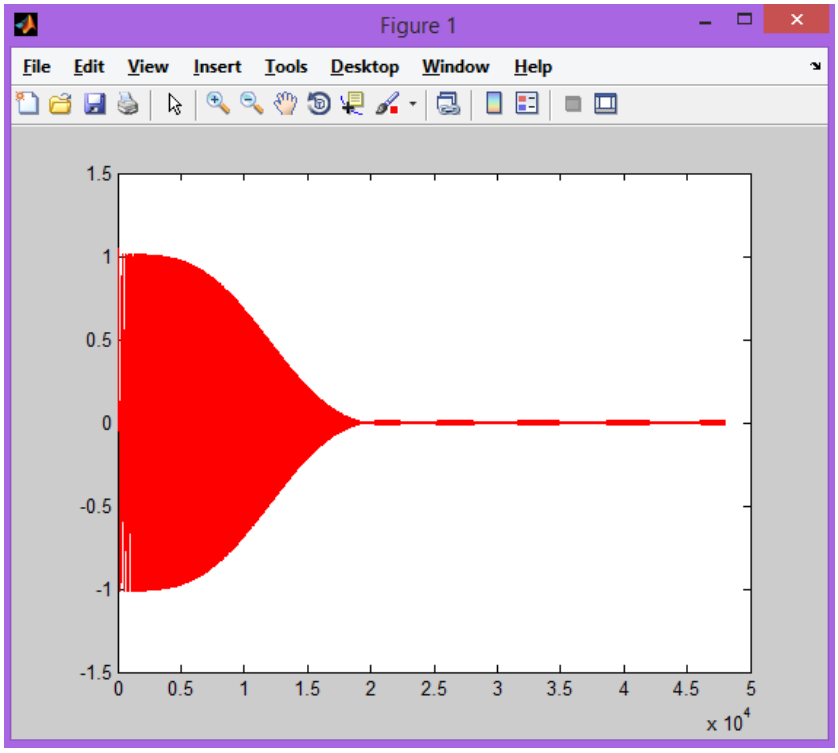

Figura 15 Resultado del test bench filtro FIR VIVADO HLS

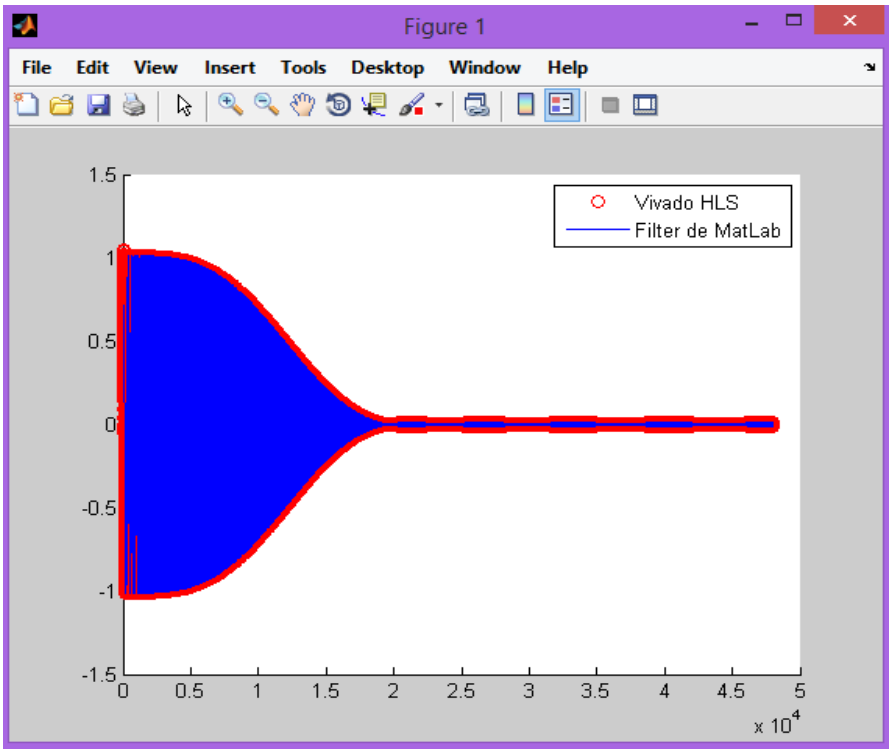

Figura 16 Comparación función Filter con resultado del test bench

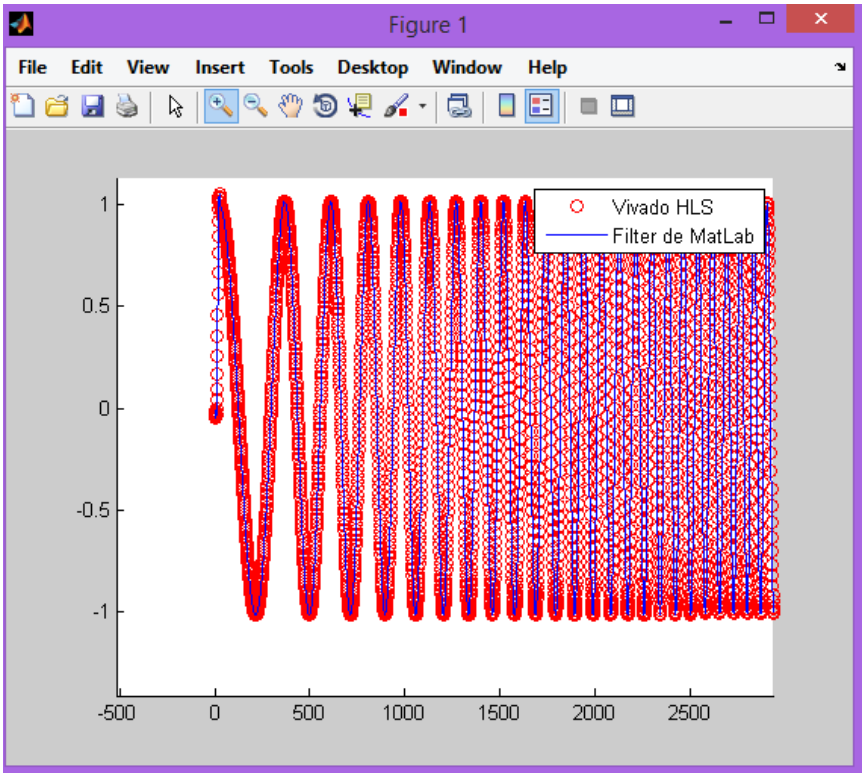

Figura 17 Comparación función Filter con resultado del test bench

3.3.4 Síntesis del diseño

Una vez verificado el código, se continúa la síntesis del diseño en C a diseño RTL. Para esto se da clic en el botón *Run C Synthesis.* Esta síntesis genera reportes de tiempo, ciclos de reloj, recursos de hardware e interfaz. La síntesis también crea su propia carpeta de ficheros en el explorador del proyecto con el nombre *syn.*

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.09	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
342	342	343	343	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	9
FIFO	-	-	-	-
Instance	0	5	460	879
Memory	0	-	96	32
Multiplexer	-	-	-	119
Register	-	-	185	-
Total	0	5	741	1039
Available	280	220	106400	53200
Utilization (%)	0	2	~0	1

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_FILTRO_FIR_PERIFERICO_AWVALID	in	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_AWREADY	out	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_AWADDR	in	5	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_WVALID	in	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_WREADY	out	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_WDATA	in	32	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_WSTRB	in	4	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_ARVALID	in	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_ARREADY	out	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_ARADDR	in	5	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_RVALID	out	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_RREADY	in	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_RDATA	out	32	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_RRESP	out	2	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_BVALID	out	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_BREADY	in	1	s_axi	FILTRO_FIR_PERIFERICO	scalar
s_axi_FILTRO_FIR_PERIFERICO_BRESP	out	2	s_axi	FILTRO_FIR_PERIFERICO	scalar
ap_clk	in	1	ap_ctrl_hs	filtro_fir	return value
ap_rst_n	in	1	ap_ctrl_hs	filtro_fir	return value
interrupt	out	1	ap_ctrl_hs	filtro_fir	return value

Figura 18 Reporte de síntesis del proyecto VIVADO HLS

3.3.5 Implementación interfaces AXI4-LITE

Para que el periférico pueda ser controlado por el procesador, es necesario agrupar los puertos de entrada y de salida en una única interfaz AXI4 Lite, con esto se crea un conjunto de drives en el momento de la exportación del diseño RTL para la comunicación.

En la barra de perspectivas se selecciona Directive (Figura 23), allí se visualiza la función principal con las entradas y salidas, donde a cada una de ellas se les inserta la directiva INTERFACE, seleccionando el destino de archivo de fuente (Source File), se escoge el modo s_axilite y se coloca una etiqueta en la opción bundle (Figura 23), esta etiqueta será el nombre que tenga la interfaz, para este diseño se nombró cada puerto como *FILTRO_FIR_PERIFERICO* para poderlos agrupar en el diseño RTL.

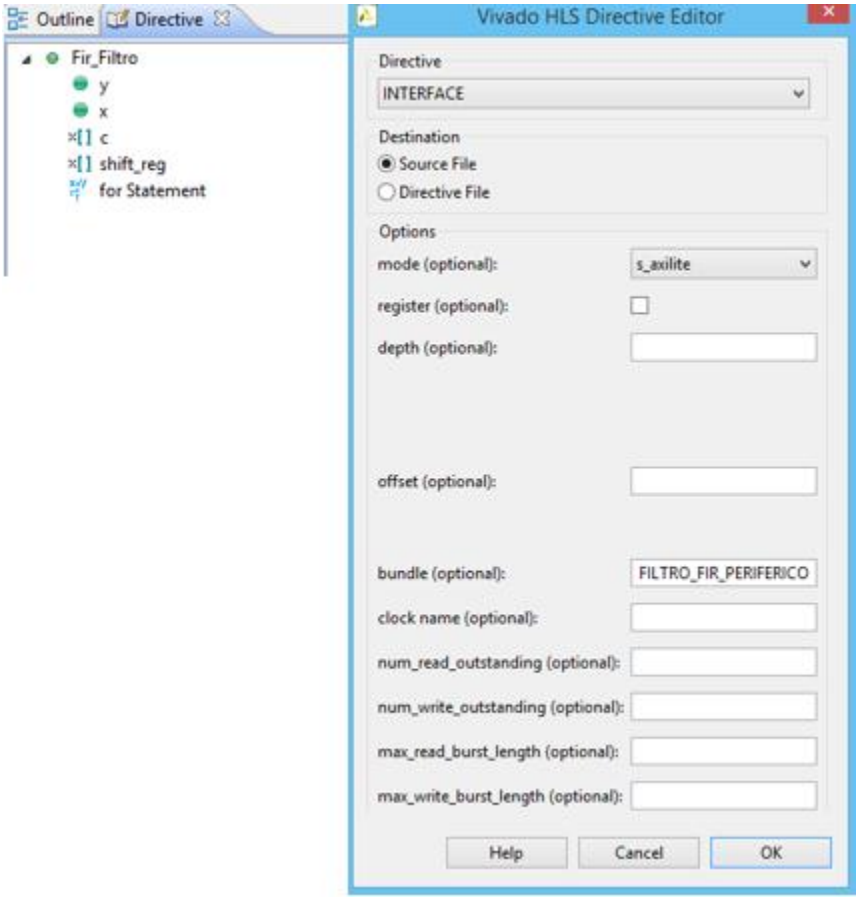

Figura 19 Implementación interfaces- creación interfaz

Una vez insertadas las directivas, quedarán visibles en el diseño las interfaces implementadas de cada puerto (Figura 20) como pragmas.

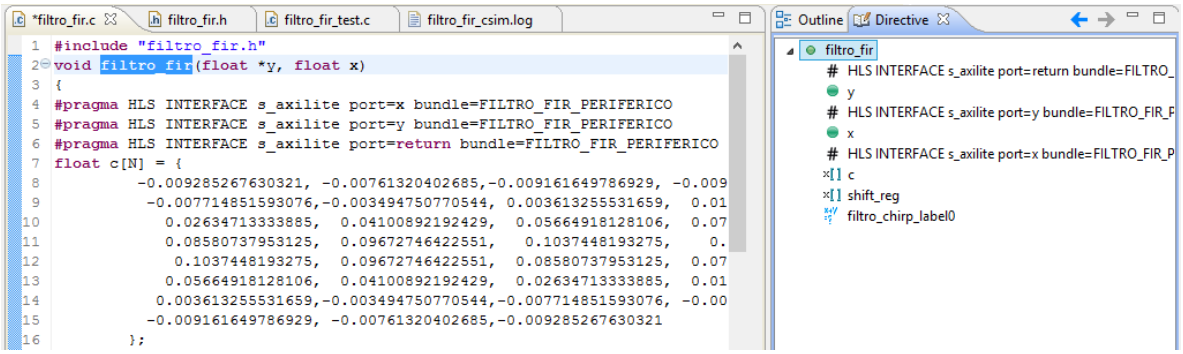

Figura 20 Implementación interfaces- interfaces creadas

3.3.6 Exportación del diseño RTL como bloque IP

De igual forma que la síntesis, es necesario hacer una verificación del diseño RTL. Esto se hace en el botón *Run C/RTL* de la barra de herramientas, el cual devuelve un informe informando si pasa o no la verificación.

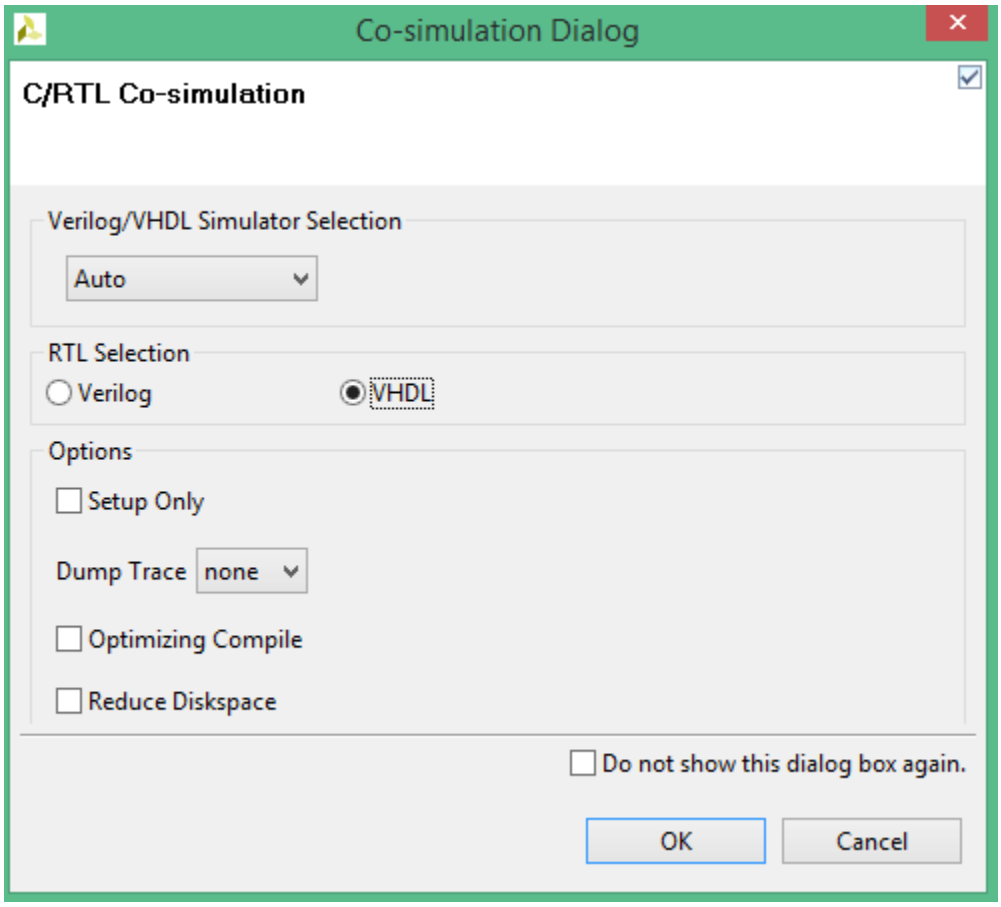

Figura 21 Verificación del diseño RTL

Si la verificación resulta correcta, se llega al último paso del diseño en VIVADO HLS exportando el diseño RTL como un bloque IP (*Intellectual Property*) a VIVADO Design Suite 2016, donde se sintetiza a un fichero *bitstream* para programar la FPGA.

Para la exportación del diseño RTL, hay u botón en la barra de herramientas llamado *Export RTL*, el cual permite escoger entre varios formatos de salida (*IP Catalog, System Generator for DSP y Syntethesized Checkpoint*), para este caso se escogerá *IP Catalog.*

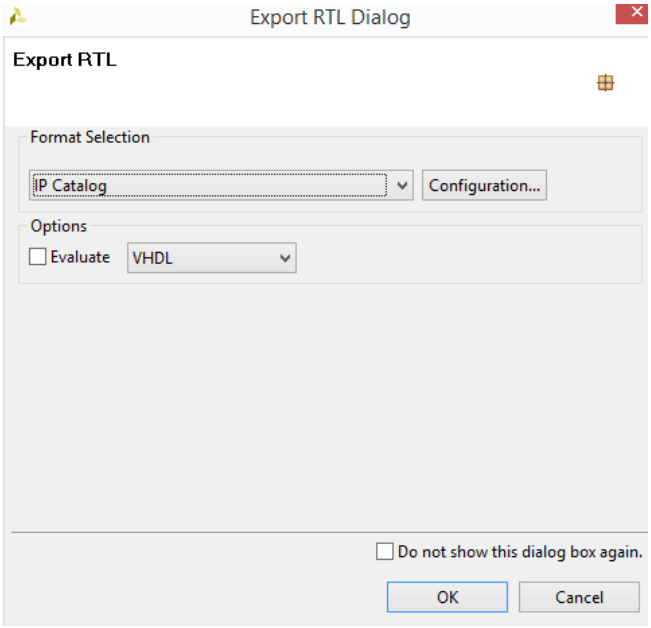

Figura 22 Exportación del diseño RTL como bloque IP

Cuando finaliza el proceso de exportación, se genera una carpeta llamada *impl* donde en ella se crea un subdirectorio *ip,* allí se crea información del periférico en un fichero *.zip,* que se importará desde Vivado Design Suit. El fichero se puede ver en la figura 23.

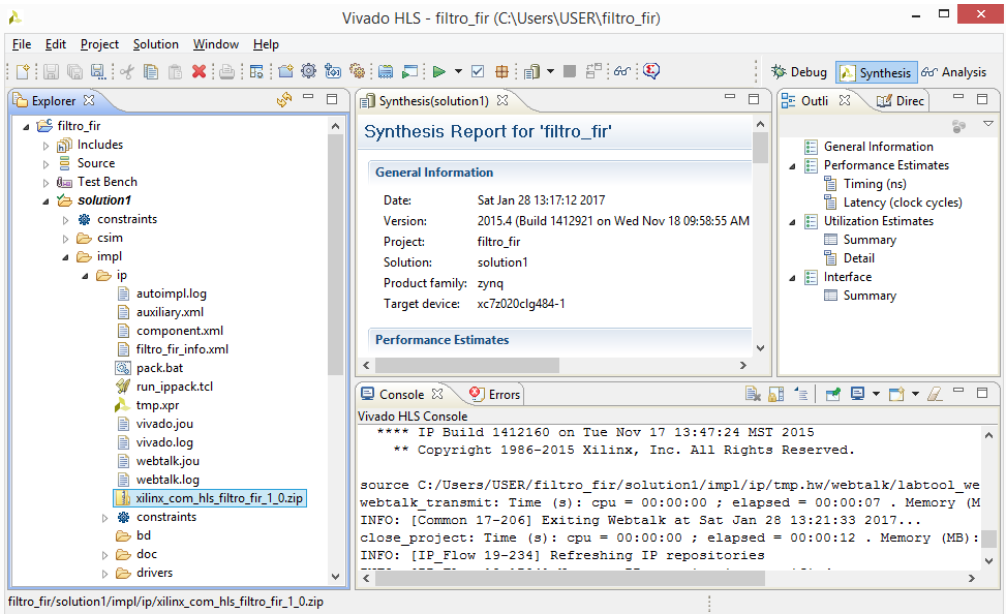

Figura 23 Fichero con información del periférico RTL

A partir de este momento, ya el diseño en HLS finaliza y se continúa con la implementación en *Vivado Design Suit* para diseñar el diseño HDL con diagrama de bloques, donde se complementará el bloque IP del diseño de HLS con el procesador de la ZedBoard y se generará el *Bitstream*.

3.4 Generación del Bitstream en Vivado HLx editions

Ya con el bloque IP generado en VIVADO HLS, el paso que sigue para poder implementarlo en hardware es configurar el procesador con el diseño en HLS. Para ello, se continúa en *VIVADO HLx editions,* a continuación se presenta el procedimiento en esta herramienta.

3.4.1 Creación de nuevo proyecto VIVADO HLx editions

En este punto se muestra la creación de un nuevo proyecto con el asistente.

Figura 24 Creación de un nuevo proyecto VIVADO HLx Editions

Una vez iniciado, se asigna el nombre y se deja seleccionado la creación de subdirectorio de proyecto y siguiente.

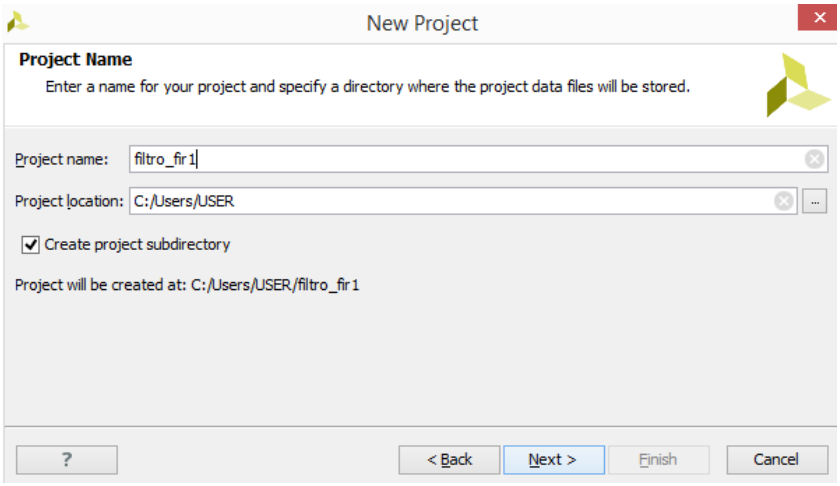

Figura 25 Nombre de nuevo proyecto

Luego se especifica qué tipo de proyecto se quiere crear, en este caso un *RTL proyect.*

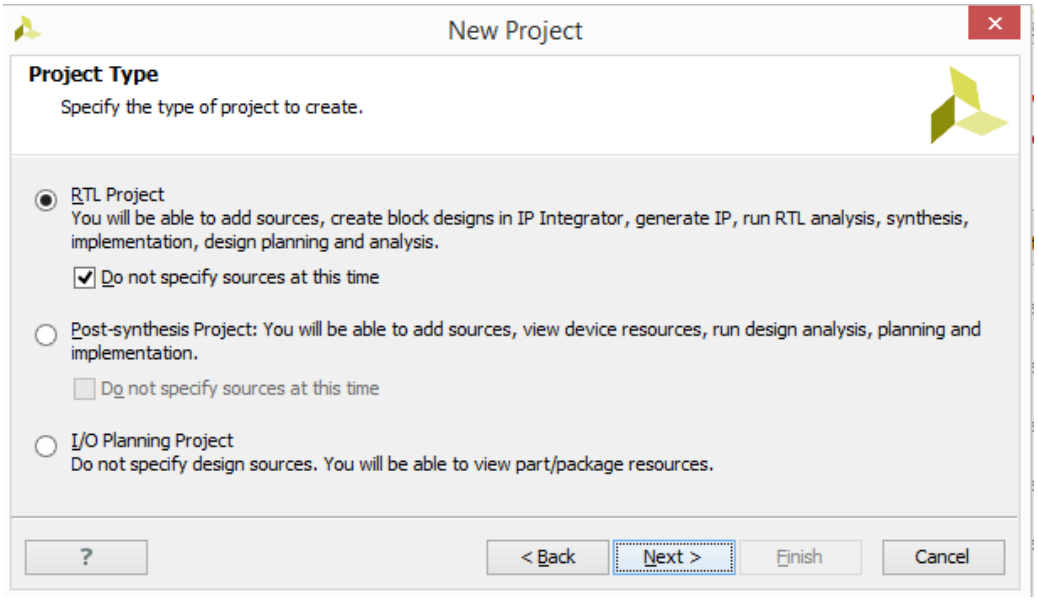

Figura 26 Selección tipo de diseño

Luego se selecciona la tarjeta para la solución del proyecto, en este caso se escoge la *Zed Board Zynq Evaluation and development Kit*. Luego se finaliza el asistente de nuevo proyecto.

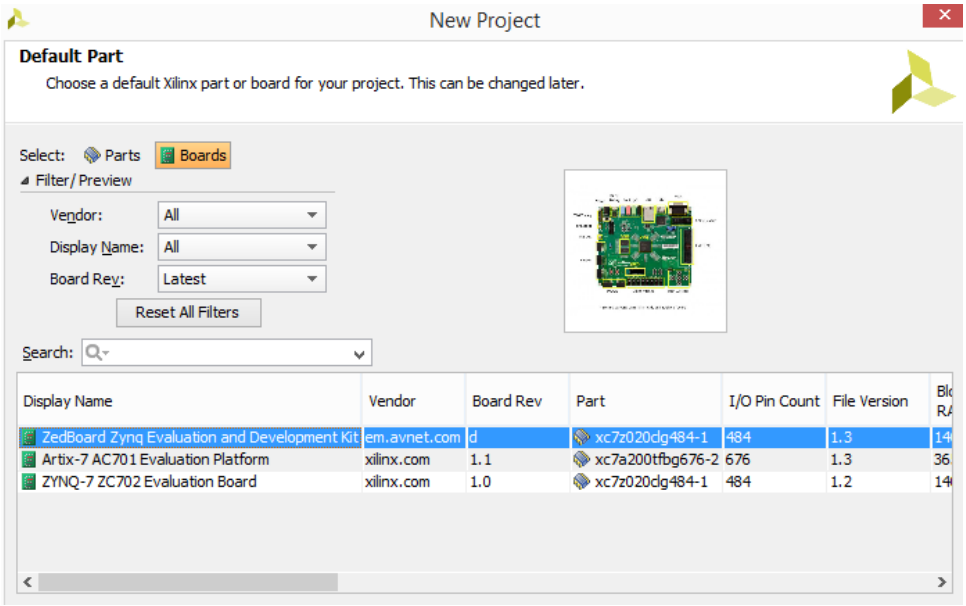

Figura 27 Selección de tarjeta de diseño

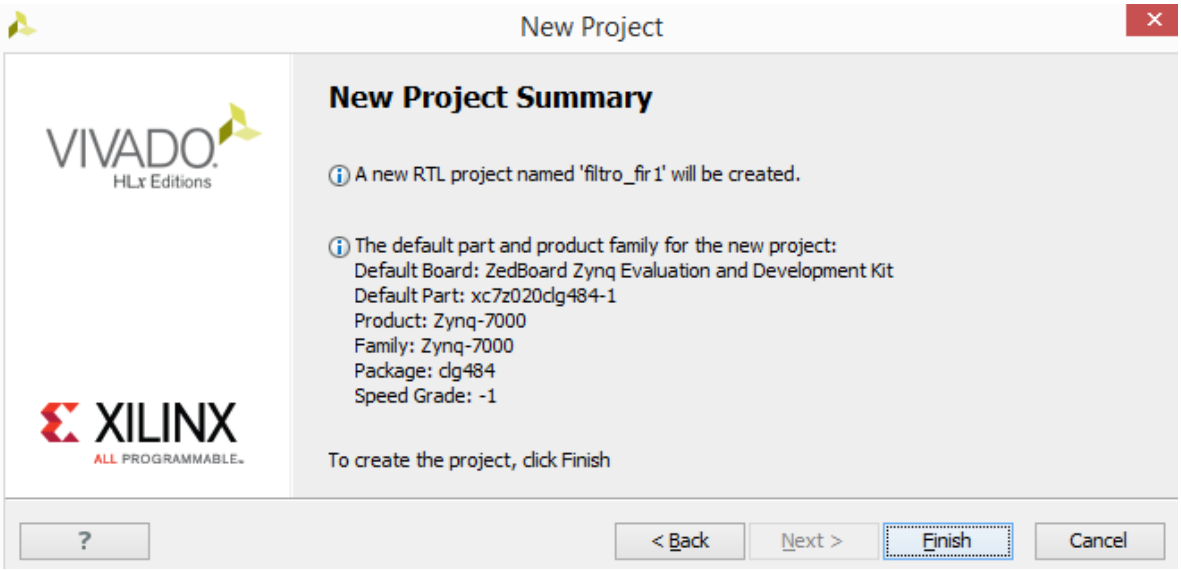

Figura 28 Resumen de nuevo proyecto a crear

3.4.2 Bloque IP VIVADO HLS desde el explorador VIVADO HLx

En los campos de *Flow Navigator,* se abre *IP Catalog,* se selecciona *IP Settings* para incorporar el bloque IP generado en VIVADO HLS.

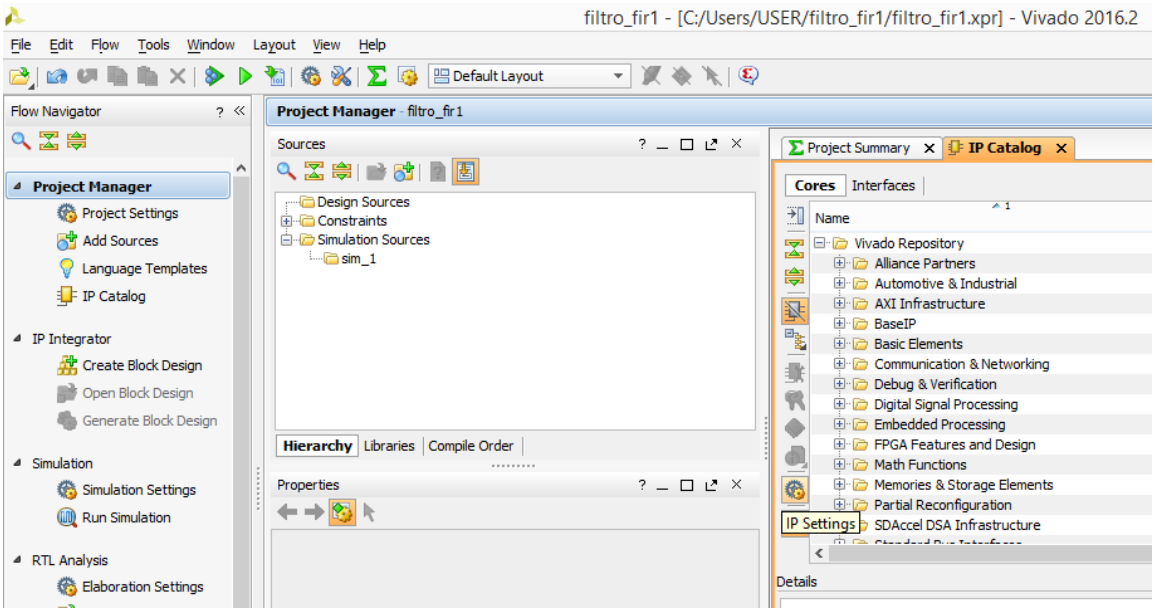

Figura 29 Incorporar bloque IP desde VIVADO HLS (1)

Cuando se selecciona *IP Settings* se genera la ventana que se muestra en la *Figura 30*, allí se selecciona *Repository Manager*

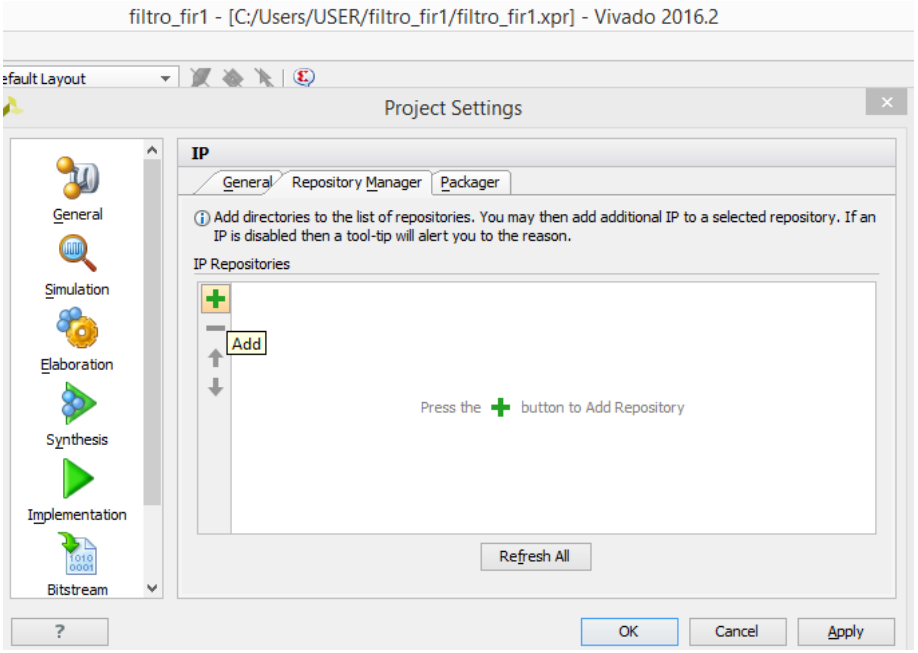

Figura 30 Incorporar bloque IP desde VIVADO HLS (2)

Para adicionar el bloque IP se debe buscar la carpeta *ip* que se encuetra en la subcarpeta *impl* de la solución del proyecto de VIVADO HLS.

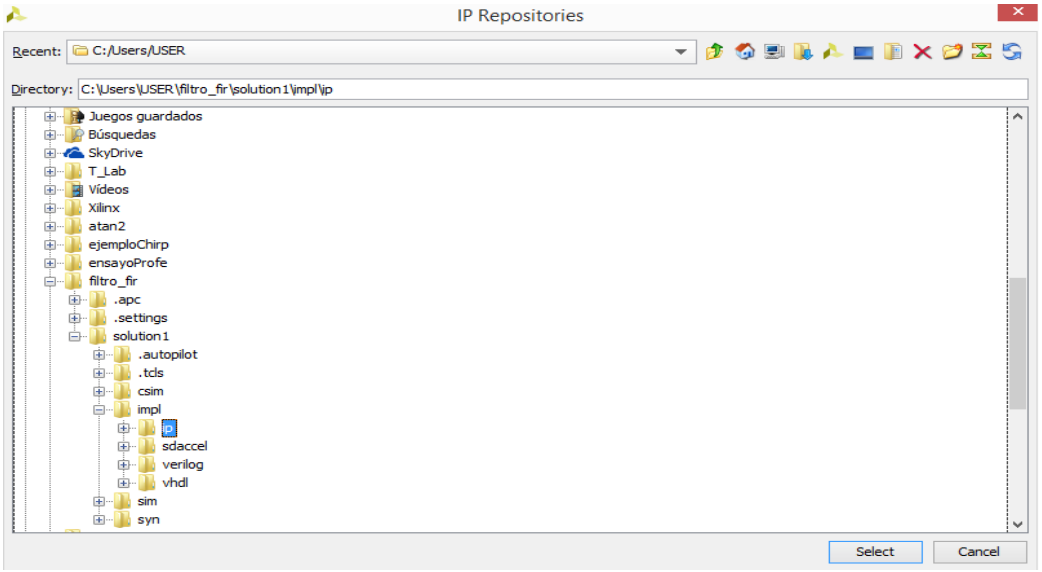

Figura 31 Incorporar bloque IP desde VIVADO HLS (3)

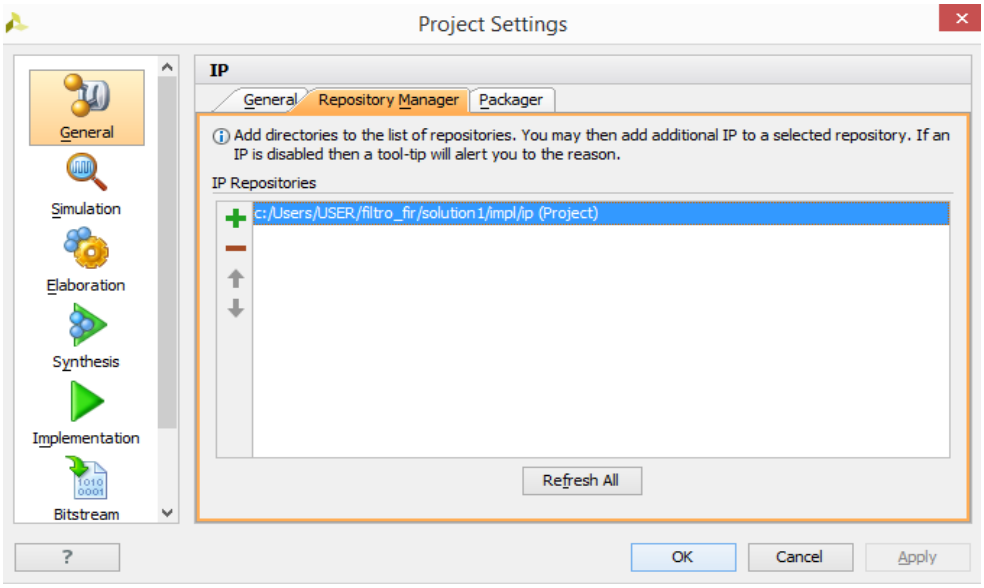

Figura 32 Incorporar bloque IP desde VIVADO HLS (4)

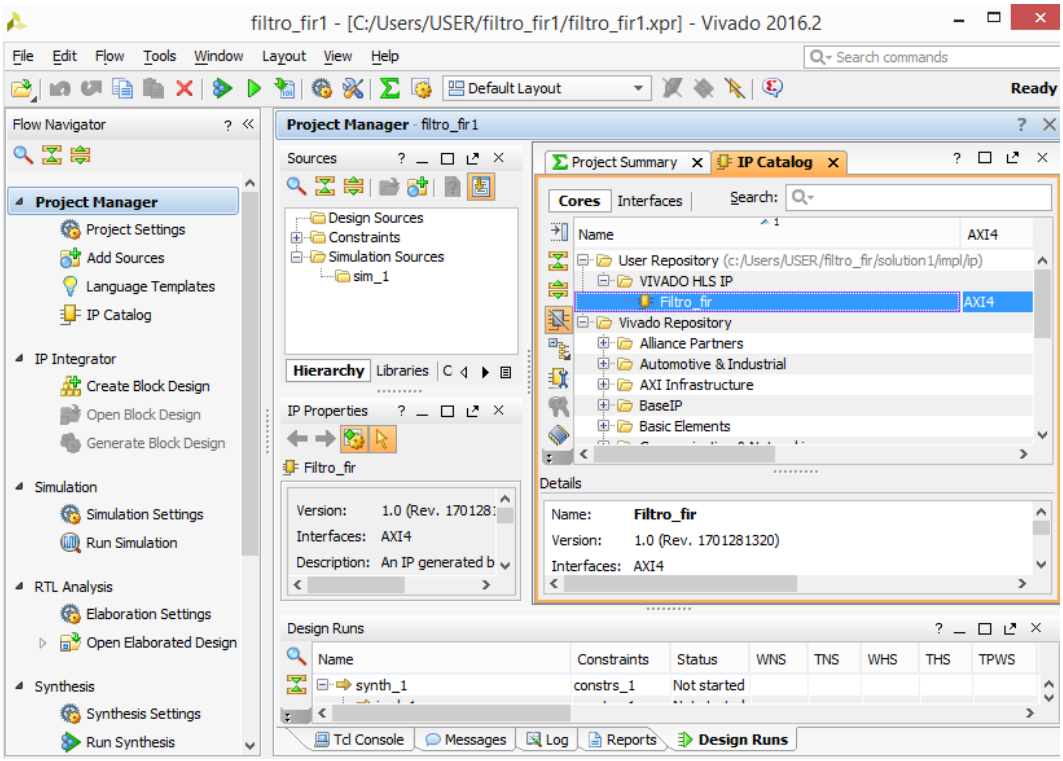

Figura 33 Bloque IP añadido a VIVADO HLx

En la figura 33 se observa cómo queda añadido el bloque IP generado en VIVADO HLS.

3.4.3 Configuración del periférico HLS con el procesador de la Zed Board

En este aparte se mostrará los pasos para configurar el bloque IP con el procesador de la Zed Boar. Para esto se debe crear un nuevo bloque de diseño.

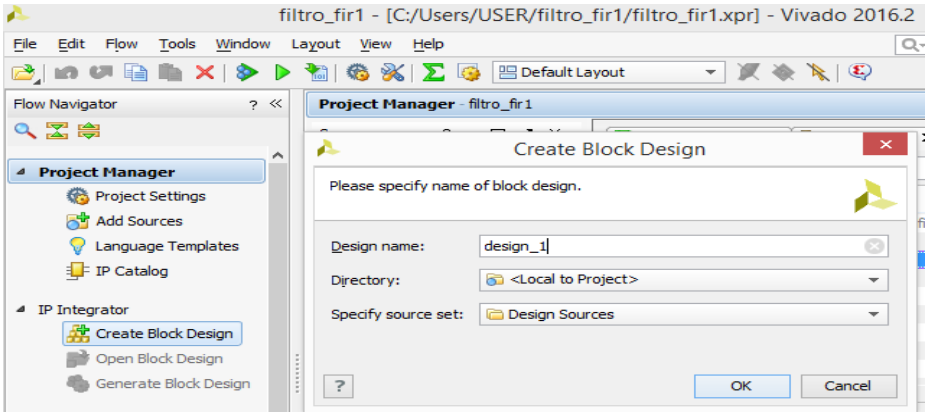

Figura 34 Creación nuevo bloque de diseño

Una vez asignado el nombre del diseño del bloque, se da *OK* para que se abra el espacio en blanco para la configuración de los bloques.

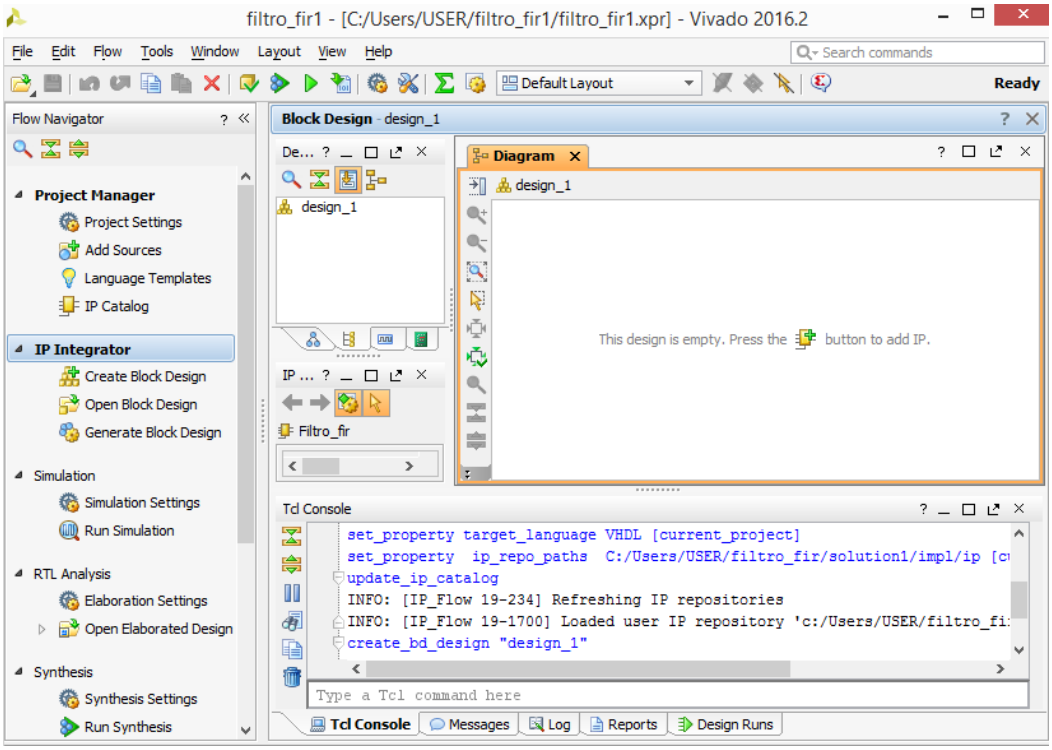

Figura 35 Espacio para diseño por bloques

Se agrega el procesador de la *ZedBoard* y luego se hace el proceso de configuración de este.

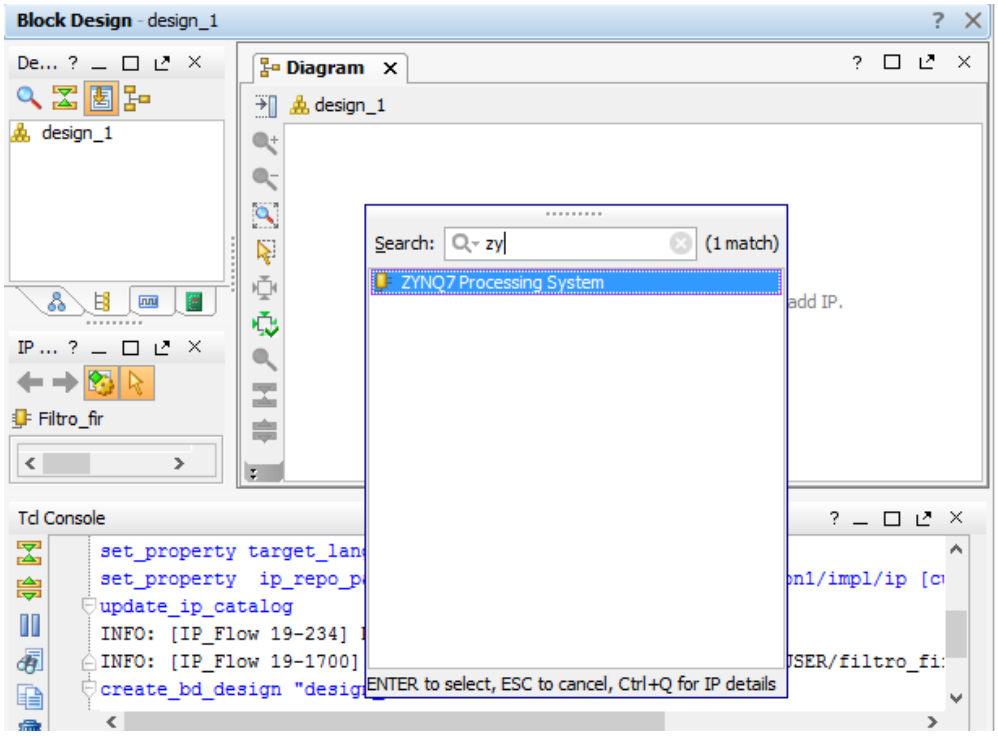

Figura 36 Selección del procesador ZYNQ7 (1)

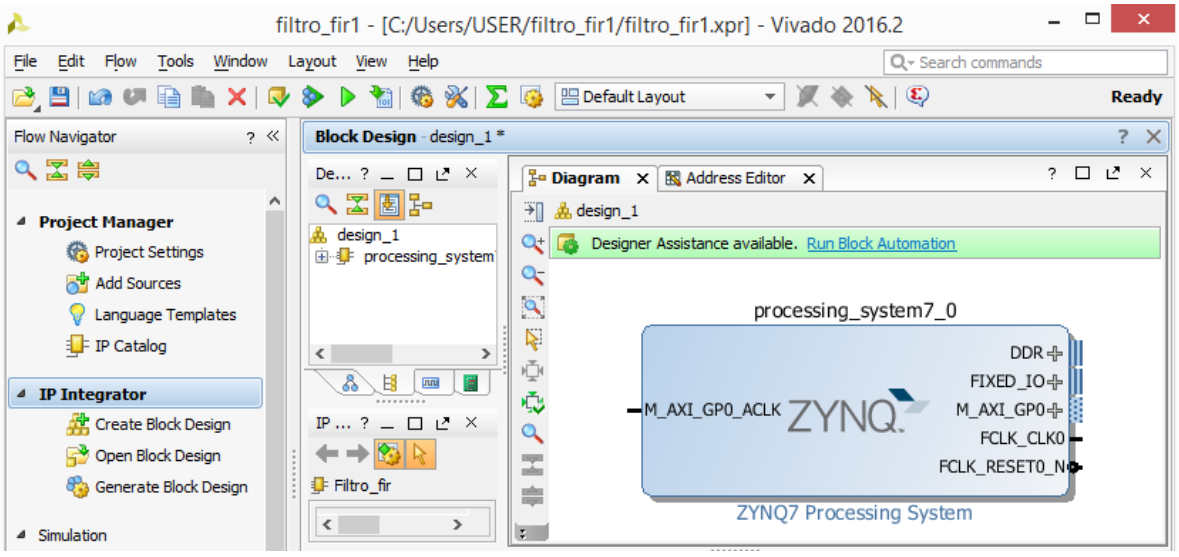

Figura 37 Selección del procesador ZYNQ7 (2)

Una vez seleccionado se da doble clic sobre el bloque para configurarlo.

En la pestaña *Presets* se escoge la tarjeta *ZedBoard,* con esto se preestablecen opciones generales del procesador de la tarjeta, con esto solamente es necesario unos pocos pasos de ajustes.

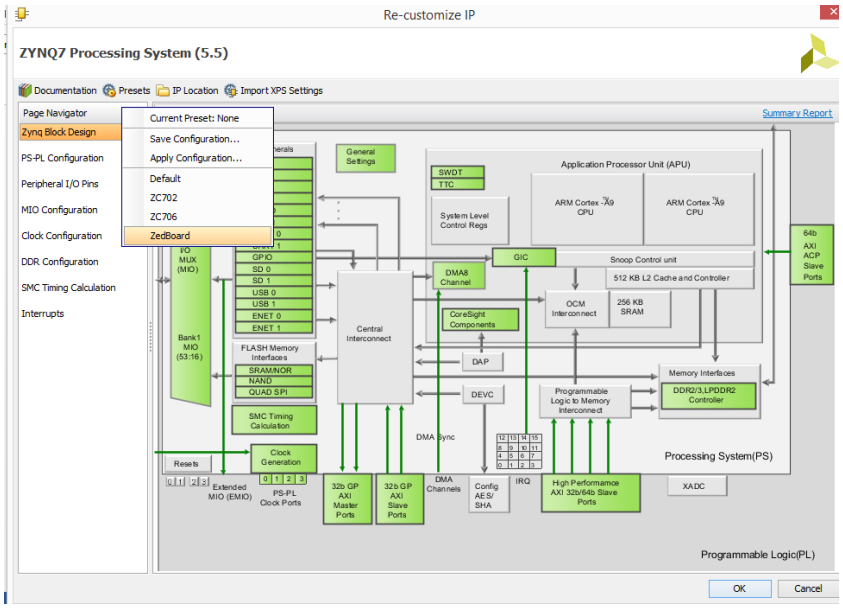

Figura 38 Configuración procesador- Preset ZedBoard

Lo siguiente es desactivar los *timers,* para ello se va al botón *MIO Configuration* y en *Application Processor Unit* se desactivan los timers que se encuentren activos.

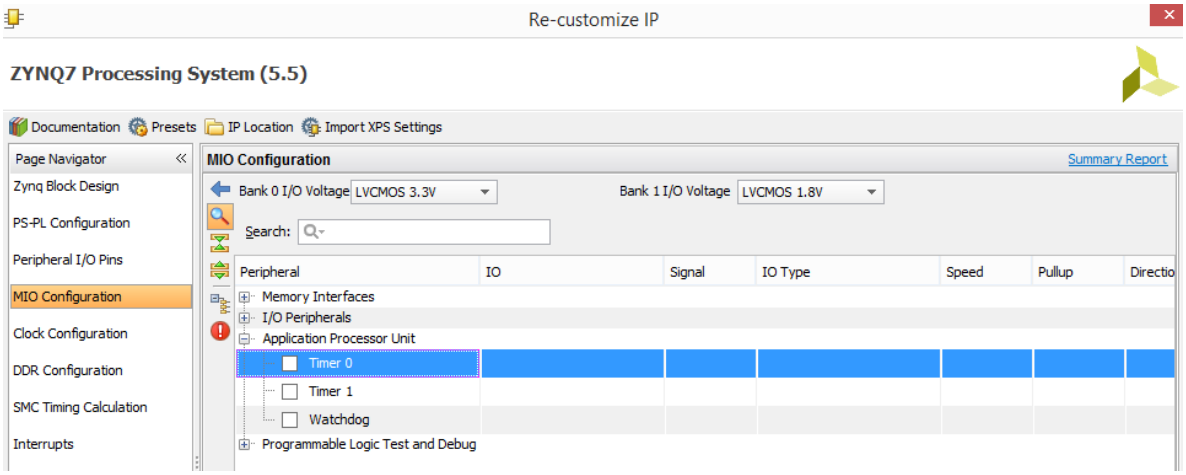

Figura 39 Configuración procesador- Desactivar Timer

Luego en el botón *Interrupts* se activa las opción *IRQ_F2P* para habilitar las interrupciones del periférico al precesador y con esto finaliza la configuración de ajustes del procesador.

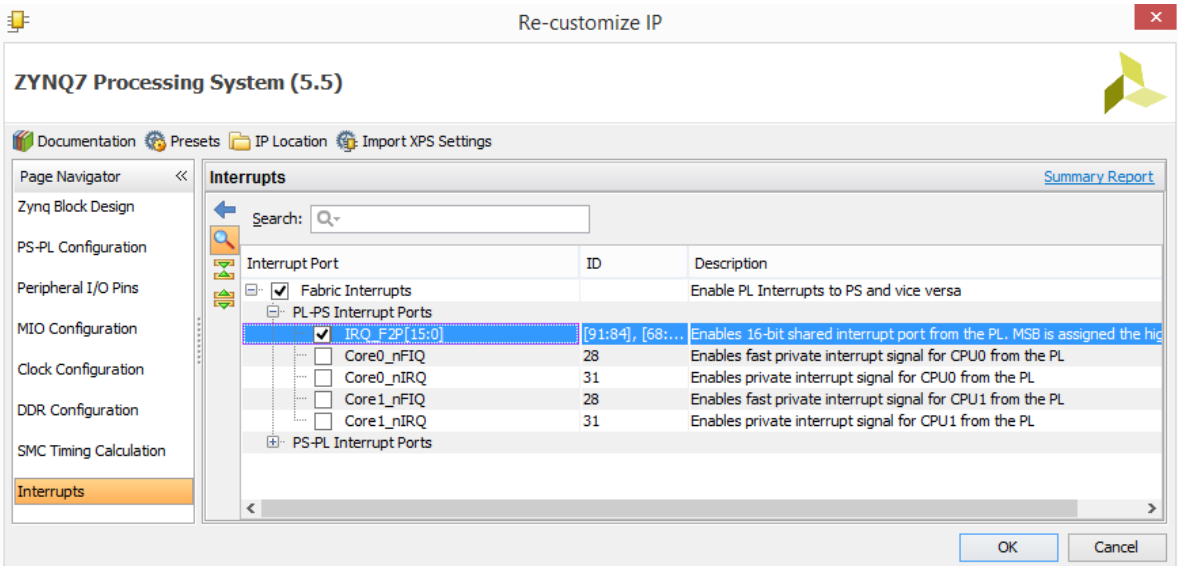

Figura 40 Configuración procesador- Interrupciones

Cuando se finaliza la configuración, se habilita un asistente de diseño con la opción *Run Block Automation,* se da clic sobre este para generar conexiones externas. Cuando se abra el asistente se debe desmarcar la opción *Apply Board Presets* para no omitir la configuración que se había realizado previamente.

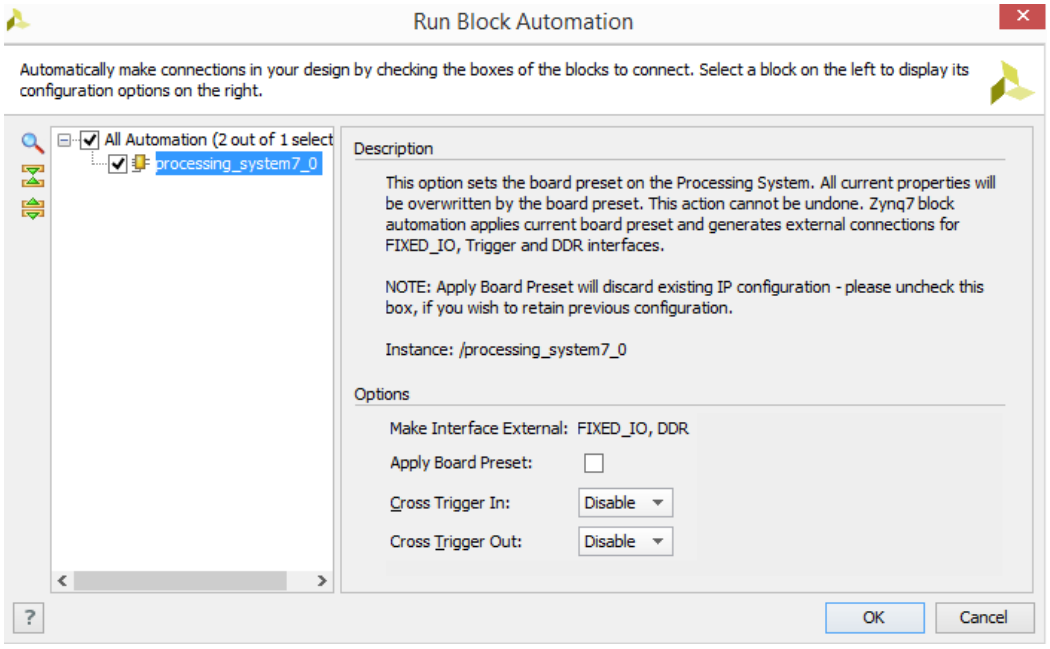

Figura 41 Configuración procesador- Generar conexiones externas

Con lo anterior se finaliza la configuración del procesador. Ahora el paso siguiente es llamar el bloque IP generado en VIVADO HLS y conectarlo al procesador.

En el campo de diseño de bloques se da clic derecho y se añade un bloque IP en *Add IP*

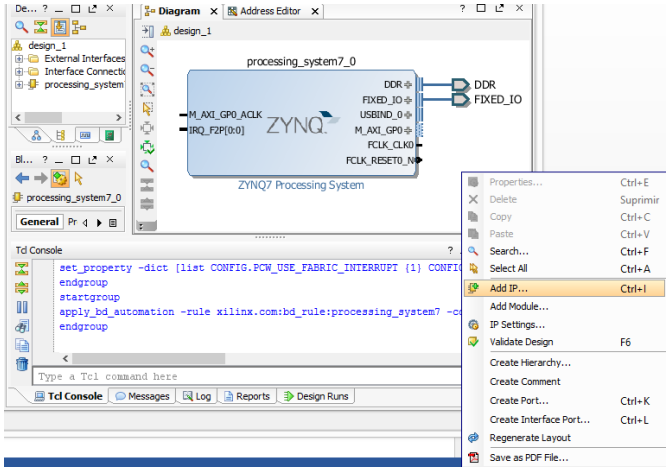

Figura 42 Adicionar bloque IP(1)

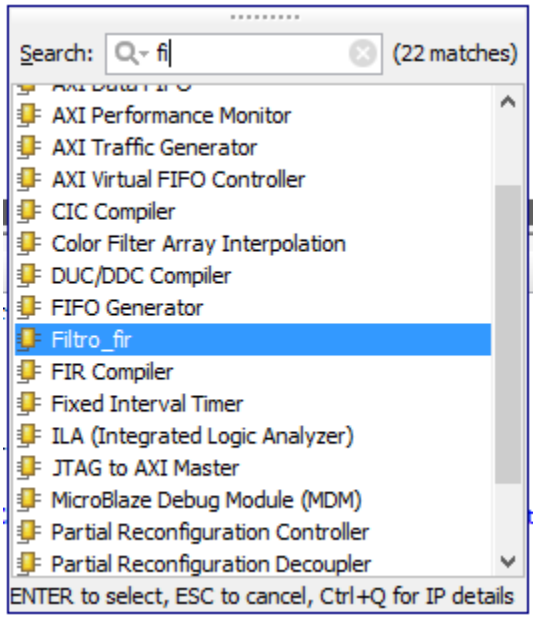

Figura 43 Adicionar bloque IP (2)

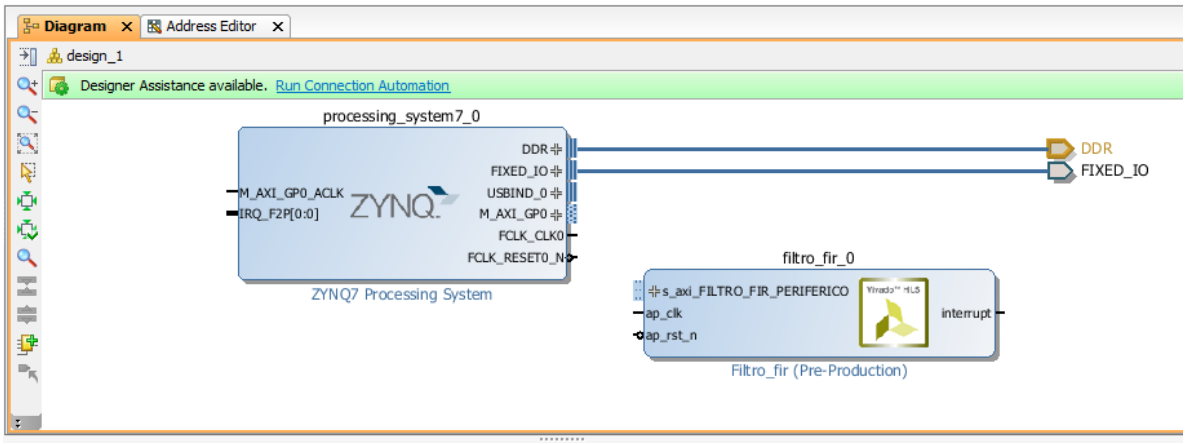

Figura 44 Adicionar bloque IP (3)

En la figura 44 se observa el bloque IP y como los puertos aparecen con el nombre que se les dio en el momento de insertar las directivas en el diseño en VIVADO HLS.

Cuando se incorpora el bloque IP aparece *Run Connection Automation* en el asistente de diseño debido a la interfaz de entradas y salidas del filtro que hace posible hacer la conexión del dispositivo esclavo *a_axi_FILTRO_FIR_PRERIFERICO* con el dispositivo maestro *M_AXI_GPIO.*

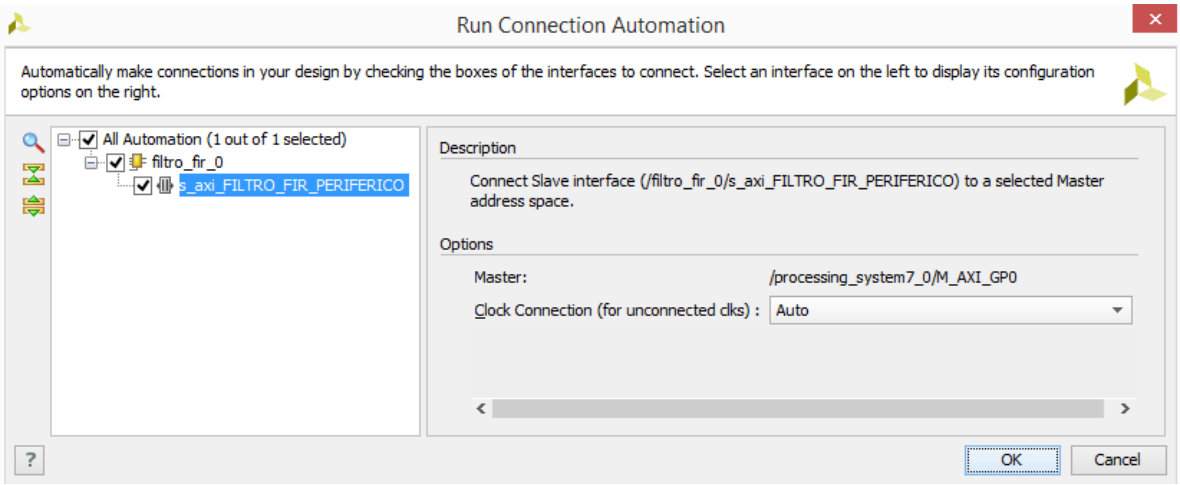

Figura 45 Conexión automática de periféricos (1)

Al aplicar la conexión automática por *Run Connection Automation* aparece el mensaje que se muestra en la figura 45, se da OK y el diseño se configura con un conector que integra los dos periféricos (figura 46).

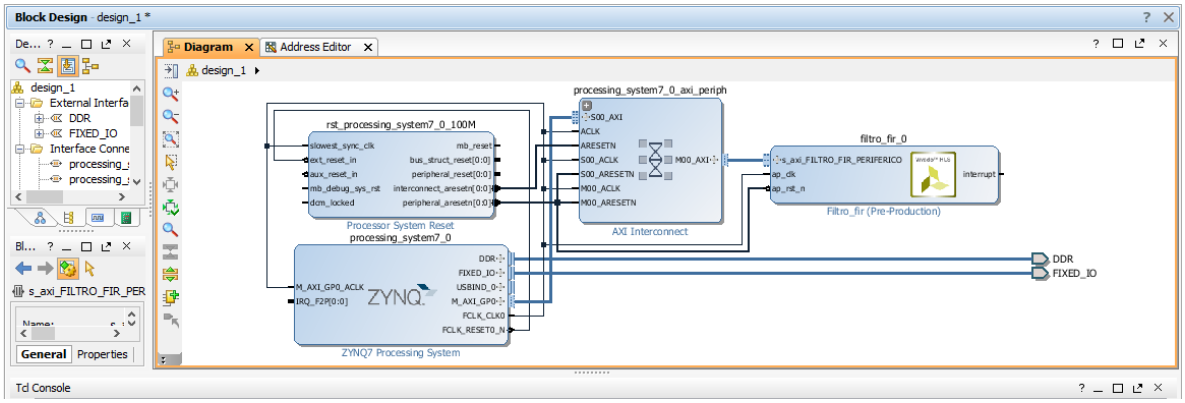

Figura 46 Conexión automática de periféricos (2)

La única conexión que no se hace automáticamente son las interrupciones, estas se hacen de manera manual, conectando *Interrup* del bloque IP con *IRQ_F2P.*

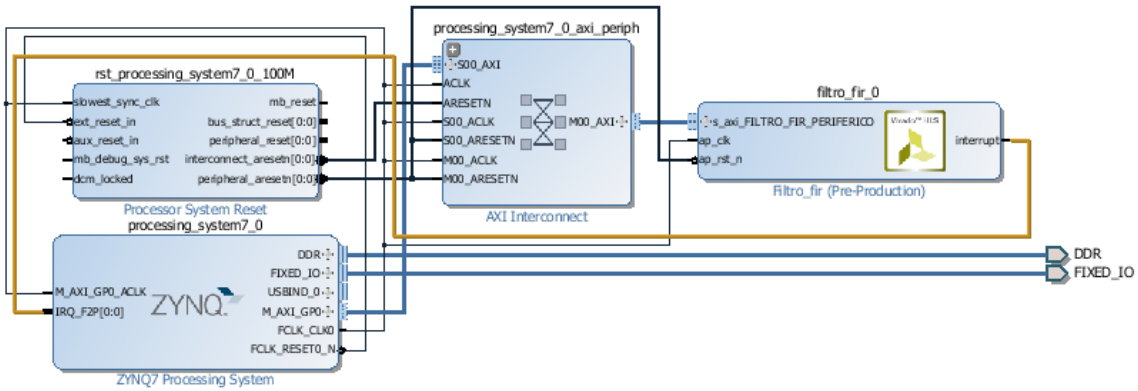

Figura 47 Conexión manual interrupciones

Una vez conectada el bloque IP con el procesador, se comprueba que si se haya asignado el rango de direcciones en la pestaña *Adress Editor.*

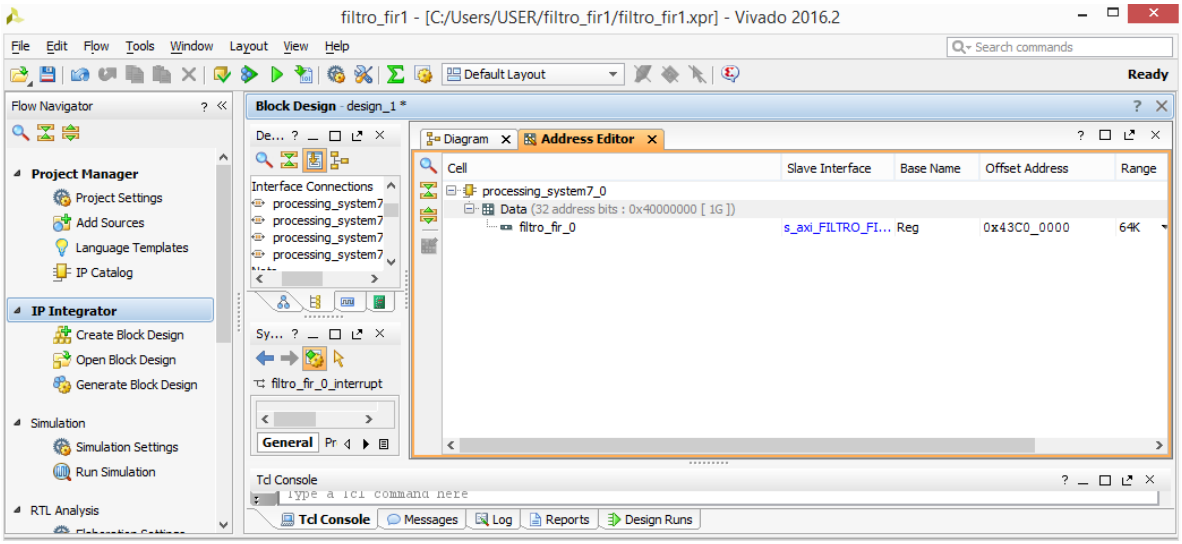

Figura 48 Validación asignación de direcciones

Antes de ir al proceso de generar el *Bitstream* es necesario validar el diseño con el botón *Validate Design* ubicado en la barra de herramientas.

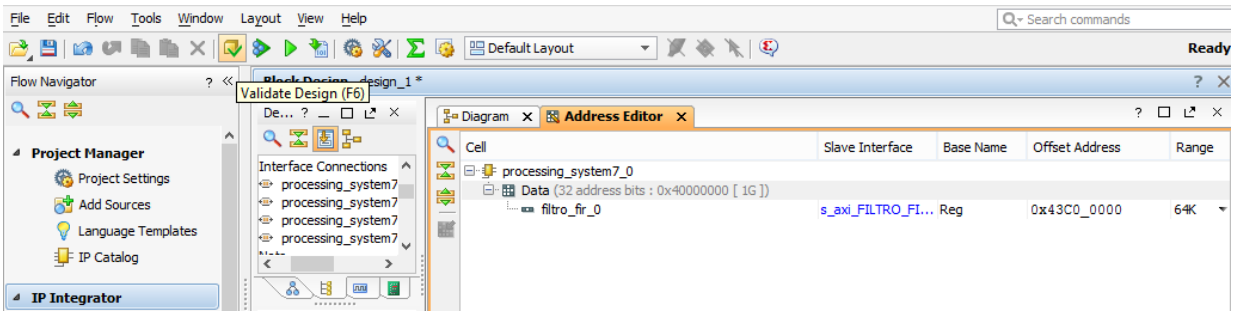

Figura 49 Validación de diseño bloques

3.4.4 Configuración de diseño para generar el Bitstream

Luego de la validación del diseño HDL, este debe pasar por una configuración general dentro del vivado previo a la generación del *Bitstream.*

Se deben generar los ficheros de implementación del diseño, dando clic derecho sobre el archivo fuente en el panel de fuentes del proyecto y seleccionando *Generate Output Products. (Figura 50).* A continuación emerge una ventana y se da clic en *Generate* con la opción de síntesis global señalada (Figura 51).

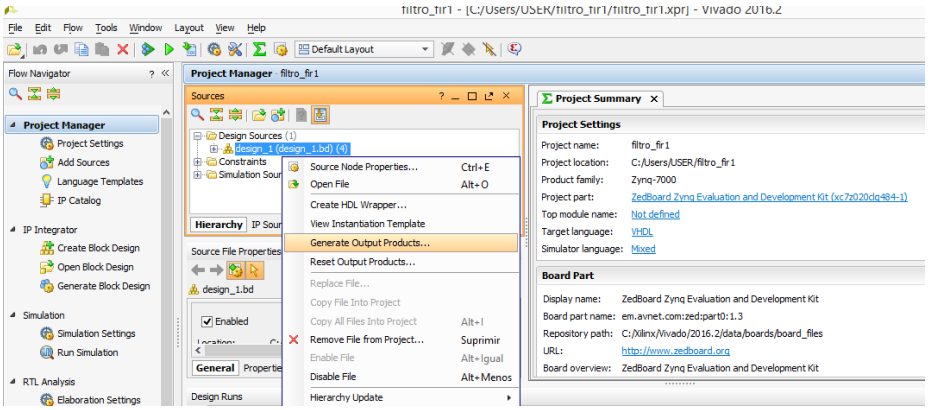

Figura 50 Generación ficheros de implementación (1)

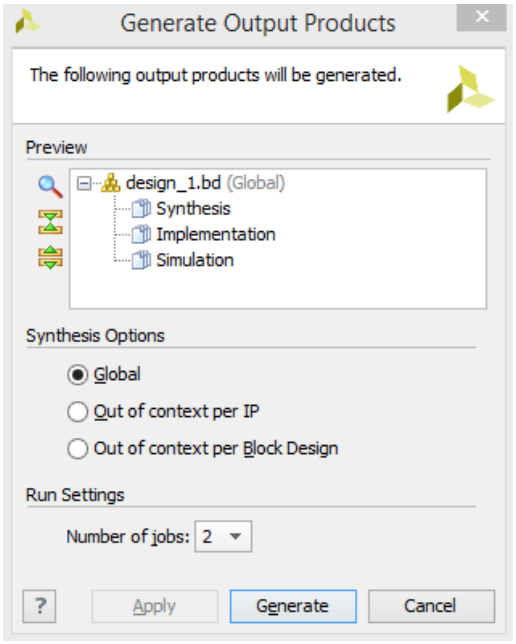

Figura 51 Generación ficheros de implementación (2)

Luego se debe generar una envolvente HDL para la síntesis e implementación. De nuevo en el panel de fuente se da clic derecho en el diseño y se selecciona *Create HDL Wrapper,* luego se escoge la opción *Let Vivado manage wrapper and auto-update*.

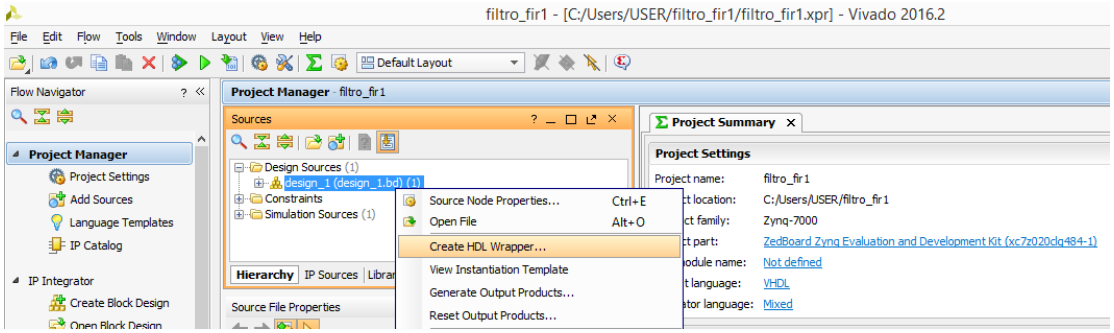

Figura 52 Generación de envolvente HDL (1)

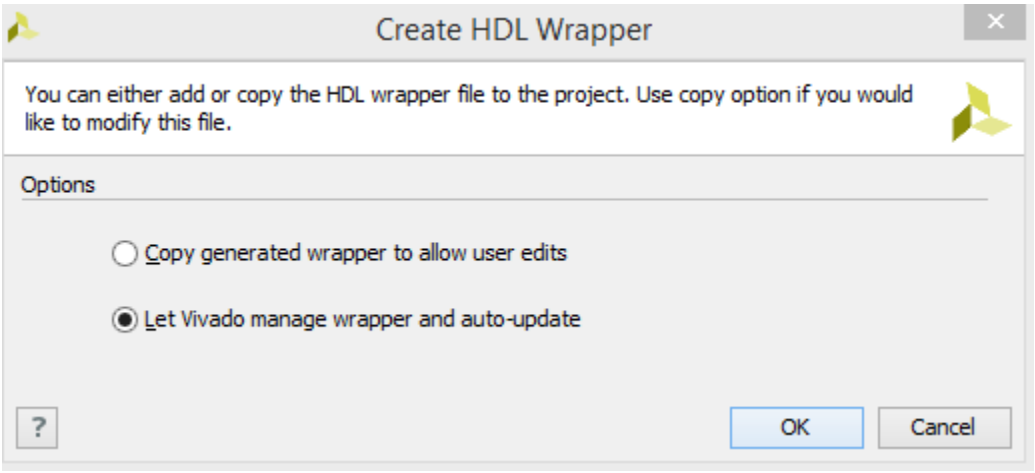

Figura 53 Generación de envolvente HDL (2)

Como último paso en VIVADO HLx se genera el *Bitstream,* para esto en el explorador del proyecto en las opciones de *Program and Debug,* se selecciona *Generate Bitstream.* Cuando termine el proceso de generación, emerge una ventana de proceso terminado, allí se selecciona *Open Implemented design.*

La exportación del diseño a la herramienta SDK se hace por *File-Export-Export Hardware* incluyendo en bitstream y luego se lanza el software SDK por *File-Launch SDK.*

3.5 Diseño en SDK

El *Software Development Kit (SDK)* es la herramienta que añade el software a la tarjeta y en este punto se explicará los pasos que se deben seguir para comunicar el periférico con el procesador.

3.5.1 Añadir aplicación sobre plataforma hardware

Cuando se abre el SDK se encuentra *hw_plataform_0,* que es el diseño donde se encuentran ficheros y drivers de periférico implementado en VIVADO HLS, los cuales se usarán en la estructura del diseño (Figura 54).

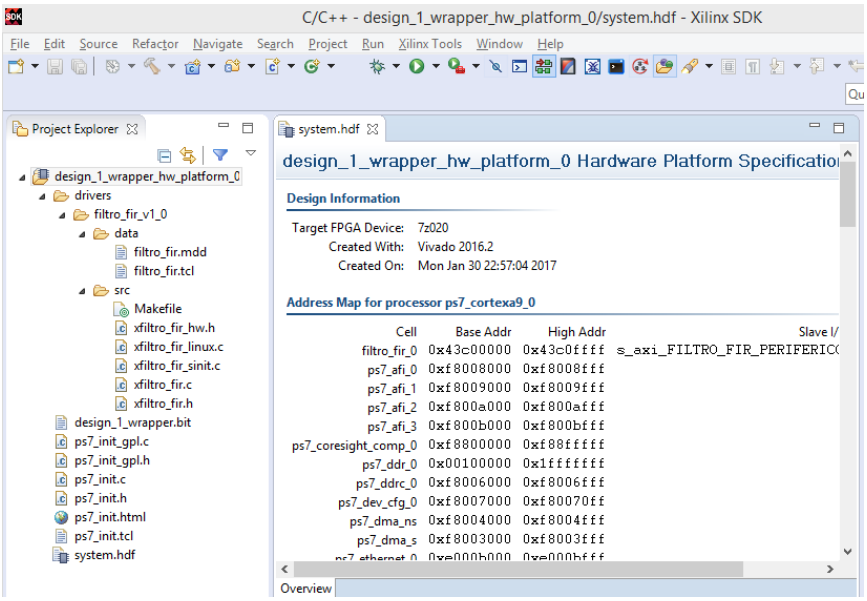

Figura 54 Módulo de diseño con su contenido

Se debe abrir una apliación desde *File-New-Application Proyect,* al seleccionarse emerge una ventana para nombrar la nueva aplicación y el lenguaje que se empleará para el desarrollo (Figura 55).

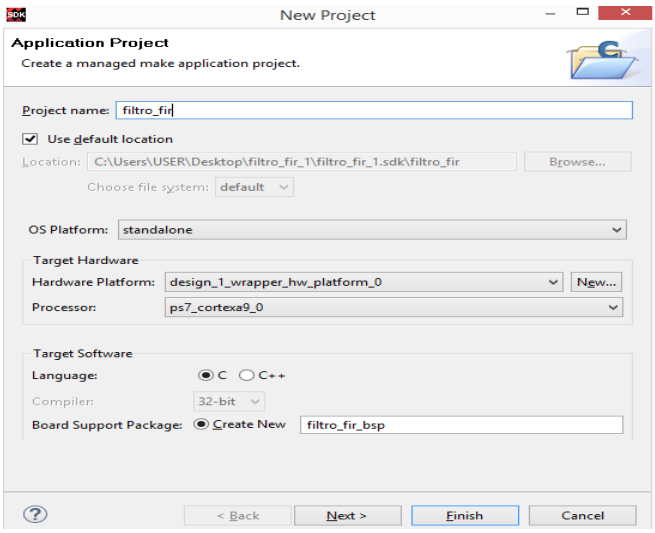

Figura 55 Nombre de nuevo proyecto y selección de lenguaje de programación

Luego de nombrar la nueva aplicación se da en *Next* y en la siguiente ventana del asistente se escoge la plantilla disponible de *Hello World* (Figura 56)para verificar el funcionamiento de la tarjeta FPGA.

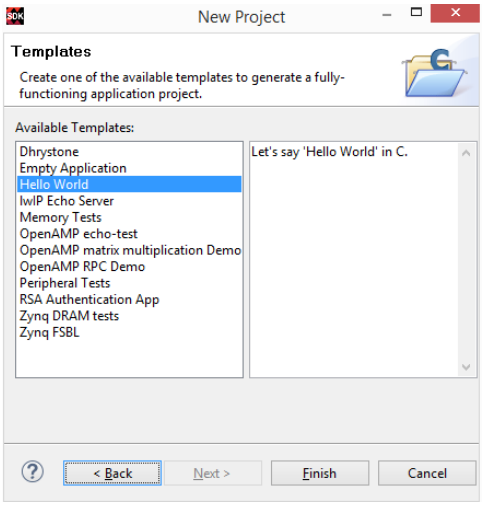

Figura 56 Selección de plantilla disponible para verificación de FPGA

Al finalizar el asistente se genera la aplicación del software en el explorador del proyecto donde se encuentra la función principal *Hello World,* con la cual se probará la tarjeta y luego sobre esta función se ingresa el algoritmo para la implementación del filtro fir que se construyó en VIVADO HLS.

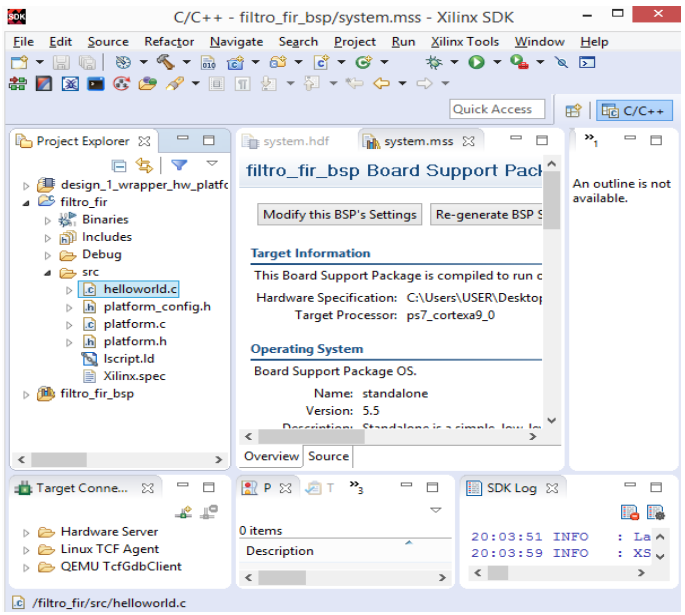

Figura 57 Función principal en explorador de proyectos

3.5.2 Drivers y cabeceras empleados en SDK

Una vez verificado el funcionamiento de la tarjeta con el *Hello world* se procede a incluir los drivers que se generó en VIVADO HLS donde para el diseño *filtro_fir* se emplean los driver *xfiltro_fir.h, xfiltro_fir.c y xfiltro_fir_hw.h.* En la figura 58 se muestra la ubicación de estos drivers en el explorador de proyecto.

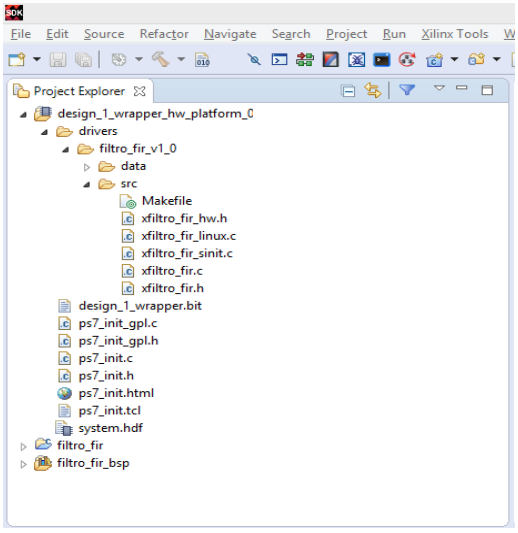

Figura 58 Drivers del diseño HLS en herramienta SDK

xfiltro_fir.h tiene la definición de las funciones para comunicar con el periférico (figura 59), *xfiltro_fir.c* contiene el cuerpo de las funciones (figura 60) y *xfiltro_fir_hw.h* sirve de consulta para saber que direcciones de memoria utilizar para leer o escribir una señal de control, de datos o interrupciones (figura 61).

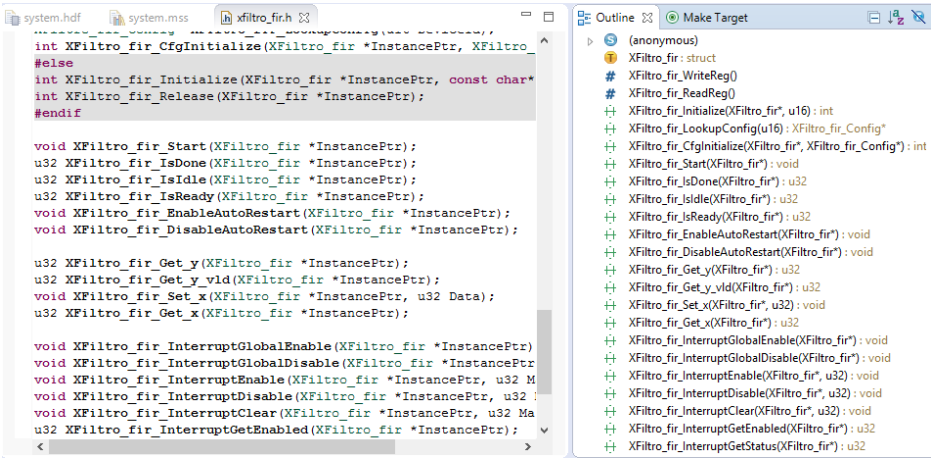

Figura 59 xfiltro.h – Definición de funciones para comunicar el periférico

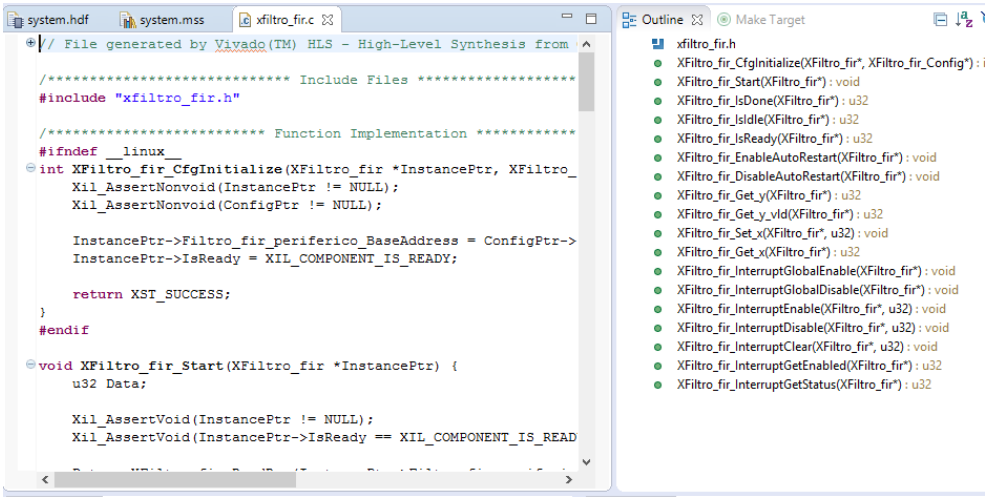

Figura 60 Driver xfiltro.c – Cuerpo del diseño del filtro

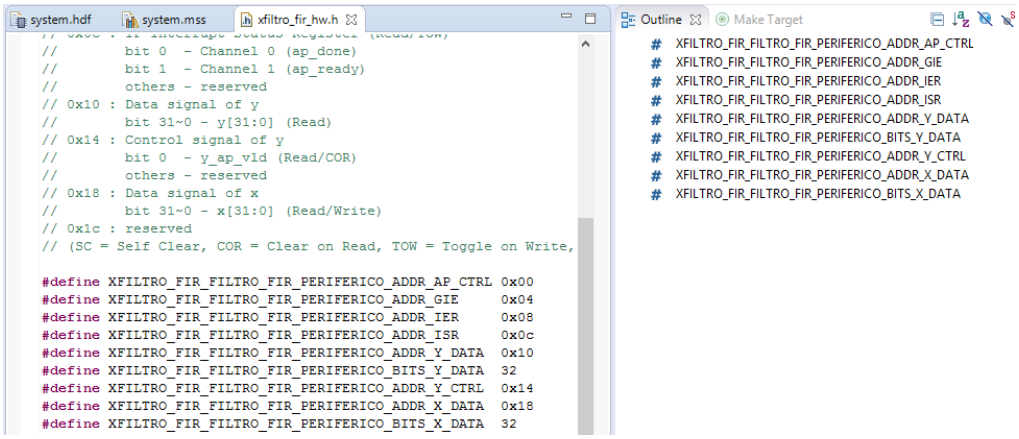

Figura 61 Driver xfiltro_fir_hw.h – Direcciones de memoria

Adicional a estos drivers, es necesario incluir el fichero *xparameters.h*, el cual tiene definiciones de parámetros, identificadores y direcciones base de memoria de los periféricos del procesador.

Los drivers mencionados, se incluyen en el fichero como cabecera de la aplicación *Hello World,* adicional a las cabeceras de funciones estándar de C (*stdio.h, stdlib.h y stdbool para operaciones booleanas y plataform.h para habilitar el cache e inicializar el UART).*

Se debe definir una variable que se refiere al periférico para utilizarla en la llamada a las funciones de los drivers. Esta variable es de tipo estructura *xfiltro_fir* (**Xfiltro_fir** filtroFir), y contiene la dirección base del periférico y la bandera *IsReady*. En la función principal se utiliza como apuntador con un *&filtroFir* tanto para la inicialización del periférico como para llevar un dato de entrada y traer el dato del resultado del filtro.

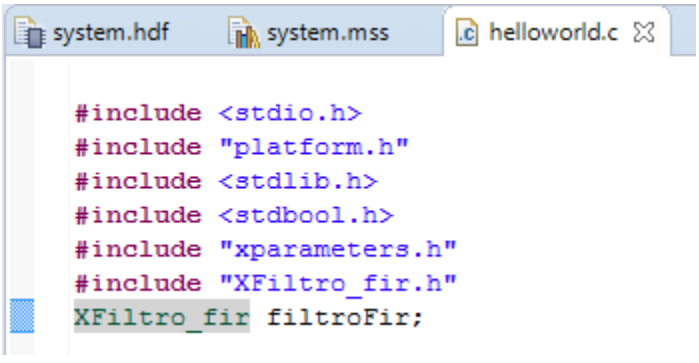

Figura 62 Cabeceras de la implementación del diseño en SDK

3.5.3 Función principal y función de inicialización del periférico

La implementación del diseño se divide en dos funciones, la función principal *main()* que es donde se tiene la señal *CHIRP* y el código de implementación para llevar las señales a los drivers de HLS y devolver la salida del filtro fir; y por otro lado una función de inicialización del periférico de HLS.

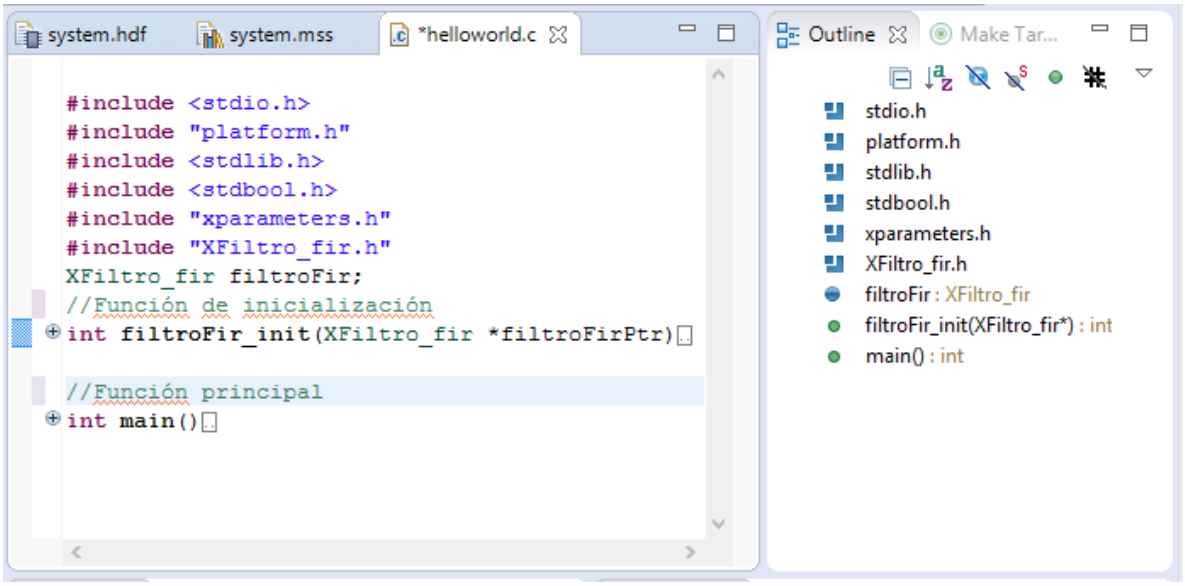

Figura 63 Estructura del diseño en SDK – Cabeceras, función de inicialización y función principal.

A continuación se muestra el código de la función principal, cabe anotar que dentro de ese código se encuentra la señal *CHIRP* donde se seleccionan los primeros mil datos de la señal y los últimos mil, esto debido a que en *SDK* el espacio de memoria no es suficiente para probar las 48000 muestras de la señal. Con lo anterior se busca mostrar la señal en la etapa donde el filtro no se ha aplicado y la etapa donde se ha aplicado el filtro totalmente.

```
int main()
{

	int i=0;
	int j=0;
	int status;
	unsigned int result_u;
	float result=0.0;
	float signal=0.0;
	unsigned int signal_u;
	init_platform();
```

```
	//Initialización, verificación de inicio del periférico HLS
	status = filtroFir_init(&filtroFir);
if(status != XST_SUCCESS)
	{
exit(-1);
	}
	else
		j=3;
	if (XFiltro_fir_jp2_IsReady(&filtroFir))
		j=1;
	//print("El periférico HLS está listo. Comenzando... \n\r");
	else
	{
	j=2;
	exit(-1);
	}

	float CHIRP[2000]={/*Mil muestras del inicio de la señal chirp y
mil muestras del final de la señal chirp*/};

	for (i=0;i<2000;i++)
		{
		signal = CHIRP[i];
		signal_u = *((unsigned int*)&signal);
		XFiltro_fir_jp2_Set_x(&filtroFir,signal_u);
		// Iniciar el dispositivo y leer los resultados
		XFiltro_fir_jp2_Start(&filtroFir);

		do
		{
		result_u = XFiltro_fir_jp2_Get_y(&filtroFir);

		result = *((float*)&result_u);
		}while(XFiltro_fir_jp2_IsDone(&filtroFir)==0);
		printf("%f\n",result);

		}

		cleanup_platform();
	    return 0;
	}
```

En el algoritmo se declara la señal *CHIRP* como un vector, donde en cada iteración a cada posición de ese vector la recorre una variable llamada *signal* que es llevada a la variable de entrada con la instrucción *XFiltro_fir_Set_x(&filtroFir,signal)* usando el apuntador *&filtroFir* para llevar al driver de HLS el valor, se procesa con la función del filtro diseñado en VIVADO HLS y devuelve con el mismo apuntador el valor filtrado con la instrucción *result = XFiltro_fir_Get_y(&filtroFir).*

Para poder mostrar los resultados del filtro es necesario realizar una conversión de tipos de datos a las entradas y salidas de las variables de la función del filtro creadas en *VIVADO HLS,* debido a que por los drivers de HLS son de tipo de dato entero. Entonces, un método para solucionar esta limitación del software es convertir el tipo de dato de entrada a entero y el dato de salida a punto flotante a partir de apuntadores como se muestra en las siguientes sentencias de la función principal:

```
signal_u = *((unsigned int*)&signal);

XFiltro_fir_jp2_Set_x(&filtroFir,signal_u);


result_u = XFiltro_fir_jp2_Get_y(&filtroFir);

result = *((float*)&result_u);
```

En el algoritmo se muestra una verificación de estatus del periférico con la función *filtroFir_init,* la cual se declara externa a la función *main().* Esta función utiliza las funciones *XFiltro_fir_ LookupConfig* con la cual se obtiene el identificador definido en *xparameters.,* configura el periférico y lo almacena en *XFiltro_fir_Config* el cual contiene la dirección base física del dispositivo, luego con *XFiltro_fir_ CfgInitialize* configura el dispositivo. Después de la configuración comprueba si el dispositivo está listo para aceptar una entrada nueva con la función *XFiltro_fir_ IsReady.*

La función se muestra a continuación:

```
//Función de inicialización
int filtroFir_init(XFiltro_fir_jp2 *filtroFirPtr)
{
      XFiltro_fir_jp2_Config *cfgPtr;
int status;
cfgPtr = XFiltro_fir_jp2_LookupConfig(XPAR_XFILTRO_FIR_JP2_0_DEVICE_ID);
if (!cfgPtr)
{
return XST_FAILURE;
}
else
status = XFiltro_fir_jp2_CfgInitialize(filtroFirPtr, cfgPtr);
if (status != XST_SUCCESS) {
return XST_FAILURE;
}

else
return status;
}
```

Así el algoritmo para la implementación en hardware se completa y el paso siguiente es la configuración de SDK para ejecutarlo.

3.5.4 Programación de la FPGA con el bitstream

Cuando ya se tiene el algoritmo completo en el SDK, es momento de programar la tarjeta FPGA con el fichero bitstream generado en VIVADO HLx. Para esto se debe ubicar en la barra de herramientas el botón *Program FPGA,* se le da clic y luego en la ventana emergente se selecciona *Program.*

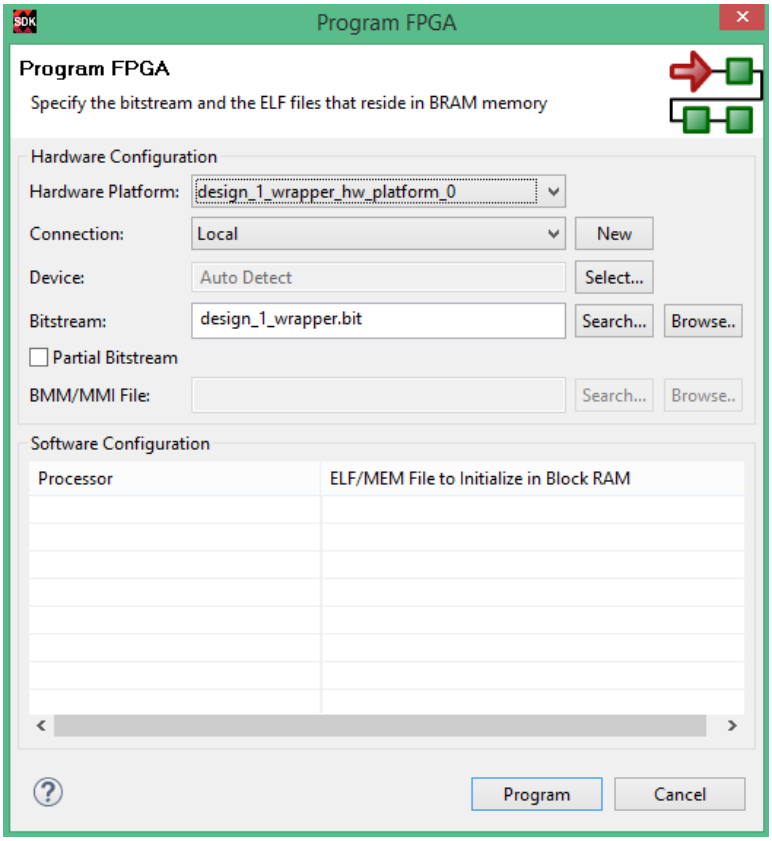

Figura 64 Programación de la tarjeta FPGA con el bistream creado en VIVADO HLx.

En la tarjeta se debe encender el LED LD12 de color azul, indicando que se programó correctamente.

3.5.5 Verificación de funcionamiento del diseño del filtro en la FPGA

Usualmente para verificar el diseño y observar los resultados luego de la implementación en la FPGA, se hace uso del Terminal conectándolo al puerto serie de la tarjeta y se imprimían en pantalla los resultados haciendo uso del *printf.* Como los drivers para

controlar el bloque HLS están fijados para trabajar con datos enteros, no es posible hacer la verificación de la salida para los datos tipo *float* con los que se está trabajando en este proyecto.

Como alternativa de verificación del diseño en la tarjeta, se recogerá la salida del filtro desde MatLab en un array, donde se abre un puerto serie desde Matlab con la configuración que se hace normalmente para el Terminal del SDK. Para esto es necesario que en el algoritmo del *SDK* únicamente se imprima el resultado del filtro.

Para abrir el puerto serie se sigue con lo siguiente en MatLab:

```
>> s1=serial('COM8','Baudrate',115200);

>> fopen(s1);
```

Luego en *SDK* se lanza la aplicación desde el explorador del proyecto, dando clic derecho al diseño en C y se selecciona *Run As* y luego seleccionar *Launch on Hardware.* En este momento se espera a que cargue el fichero.

Ahora se recoge la salida del filtro desde MatLab con *fscan* en un bucle *for,* donde escanea el puerto serial que se abrió y guarda el valor en un array, el cual posteriormente se grafica y así poder observar el resultado del filtro en la FPGA.

```
>> for i=1:2000
salida=fscanf(s1,'%f');
SalidaFiltroSDK(i)=salida;
end
```

Ahora en *SalidaFiltroSDK* se encuentra la señal *CHIRP* filtrada.

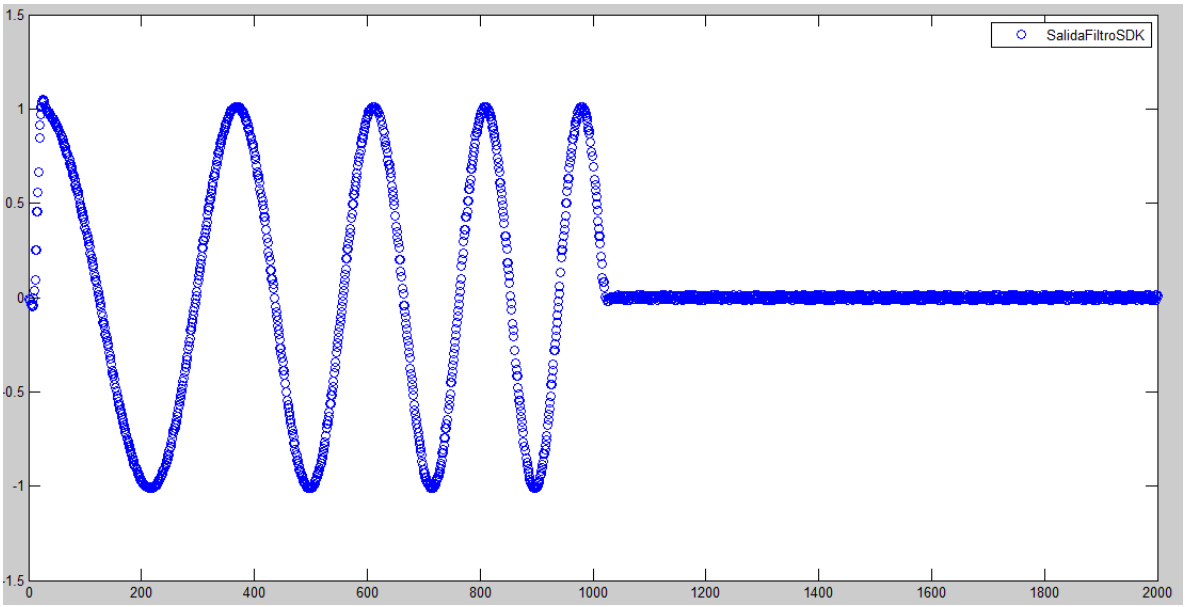

Figura 65 Resultado de aplicación del filtro en la ZedBoard

En la figura 65 se muestra el resultado del filtro desde la *ZedBoard.* Para comparar este resultado se hará uso del *TestBench* implementado en *VIVADO HLS* con las mismas muestras (se modifica el vector correspondiente a la señal chirp con las 2000 muestras) usadas en la *ZedBoard* y el resultado se graficará en *MatLab* de igual forma.

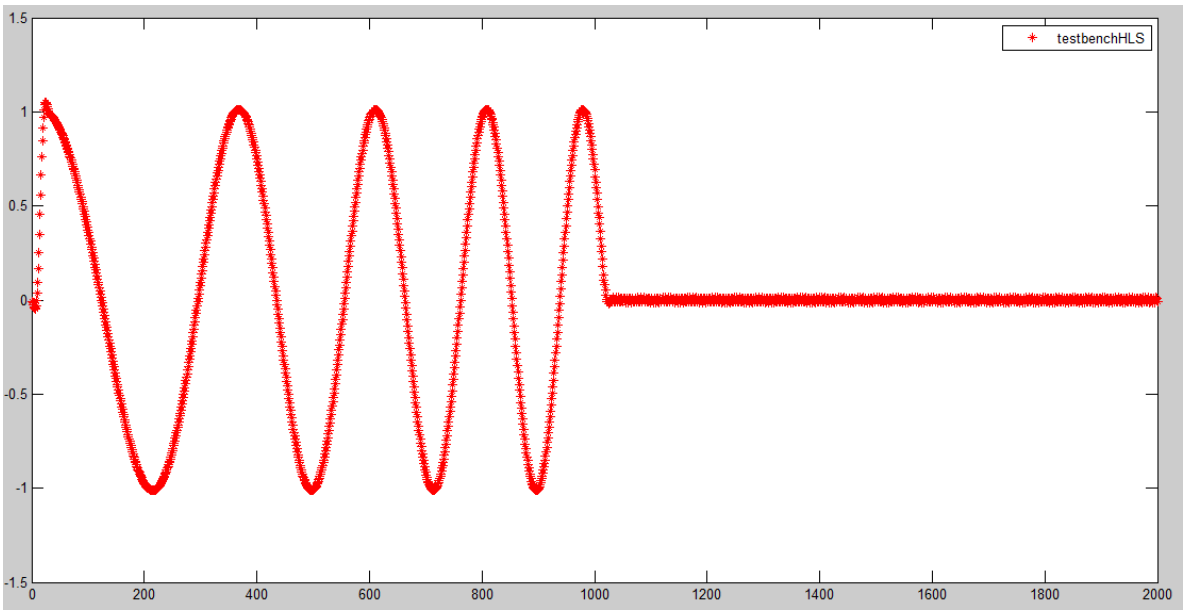

Figura 66 TestBench de Vivado HLS con las mismas muestras usadas en la ZedBoard

Ahora se superponen las señales para validar gráficamente la desviación (Figura 67).

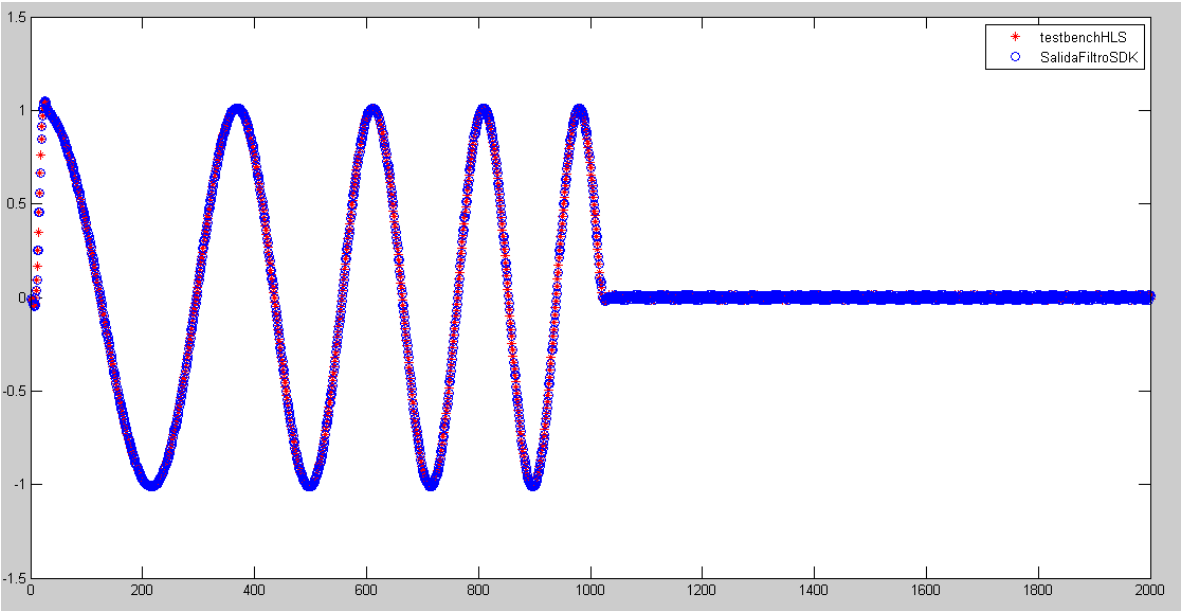

Figura 68 Comparación filtro en ZedBoard y TestBench de Vivado HLS

Como se muestra en la Figura 68, la respuesta de la aplicación en la ZedBoard tiene un comportamiento esperado referente a las pruebas simuladas.

Con el resultado y la comprobación del diseño se finaliza la implementación de un diseño digital a partir del Software de alto nivel *VIVADO HLS.*

4. RESULTADOS Y DISCUSIÓN

4.1 Test bench VIVADO HLS

Vivado HLS ofrece la opción de *test bench* donde se puede observar el resultado en consola de la función del filtro al procesar una señal. Como herramienta importante para realizar un test previo a la implementación en hardware, se muestra los resultados de la aplicación y se realiza una comparación con otro software para validar su funcionamiento.

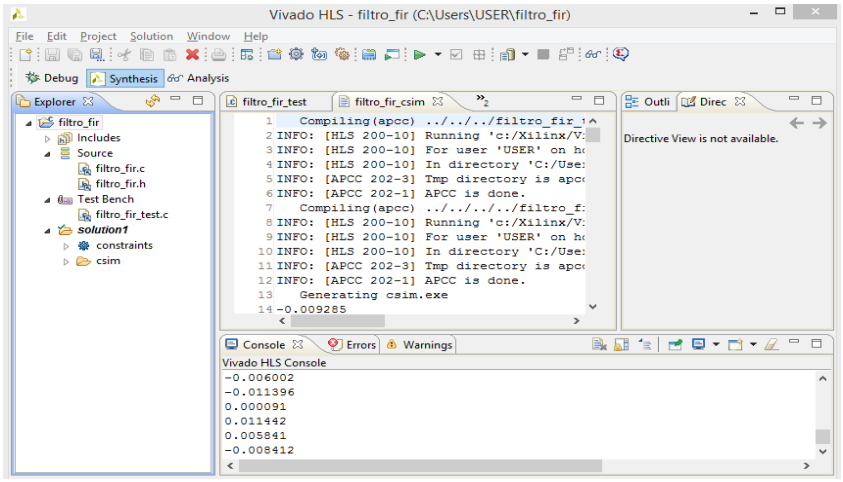

Figura 70 Resultados en Consola VIVADO HLS Test Bench

Con Matlab se grafica el resultado que se imprimió en la consola a partir del *test bench.*

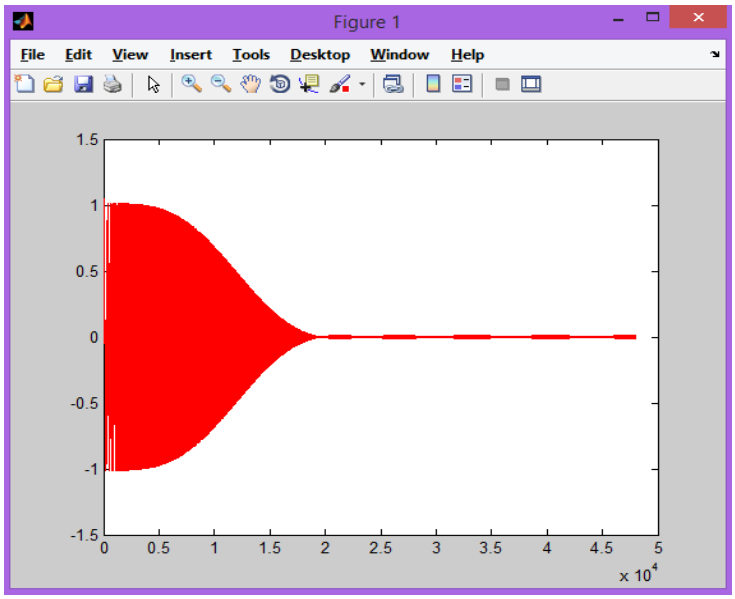

Figura 71 Resultado del test bench filtro FIR VIVADO HLS

En la figura anterior se muestra como la señal se va a cero una vez alcanza la frecuencia de corte del filtro.

Ahora, para comparar la respuesta de la simulación en VIVADO HLS a partir del *test bench*, se grafica la respuesta del filtro de MatLab usando la finción *Filter* aplicando en las mismas muestras de la señal *Chirp* con las mismas constantes del filtro digital implementadas en la función principal.

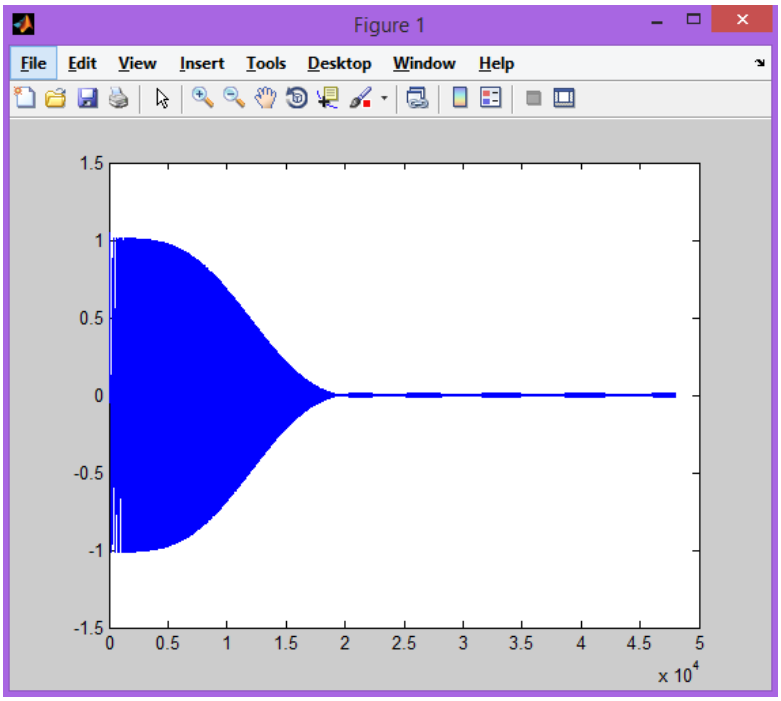

Figura 72 Función Filter de Matlab en señal Chirp

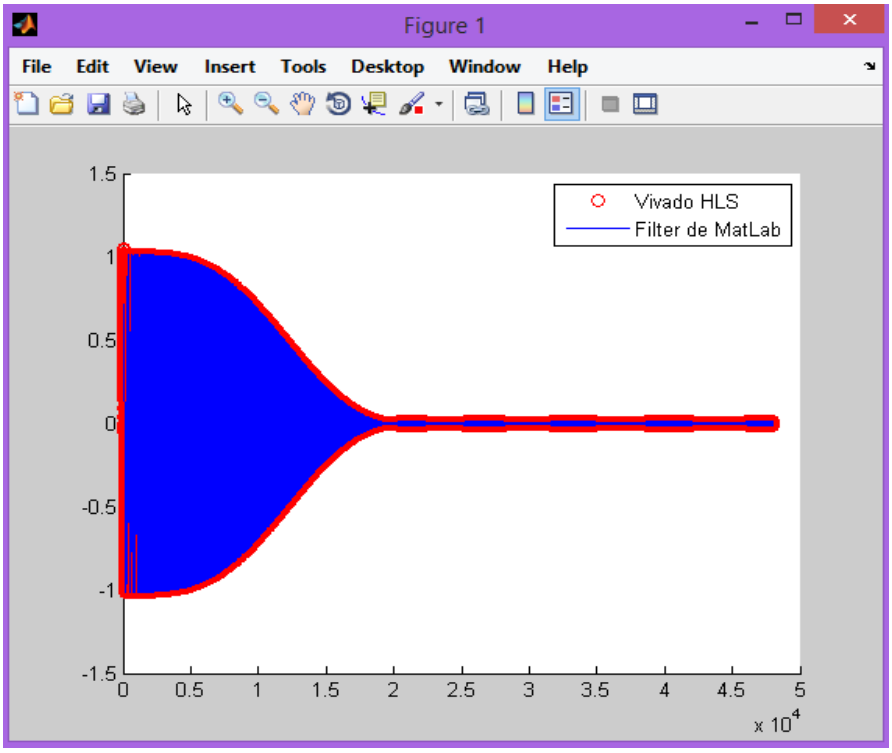

Figura 73 Comparación función Filter con resultado del test bench

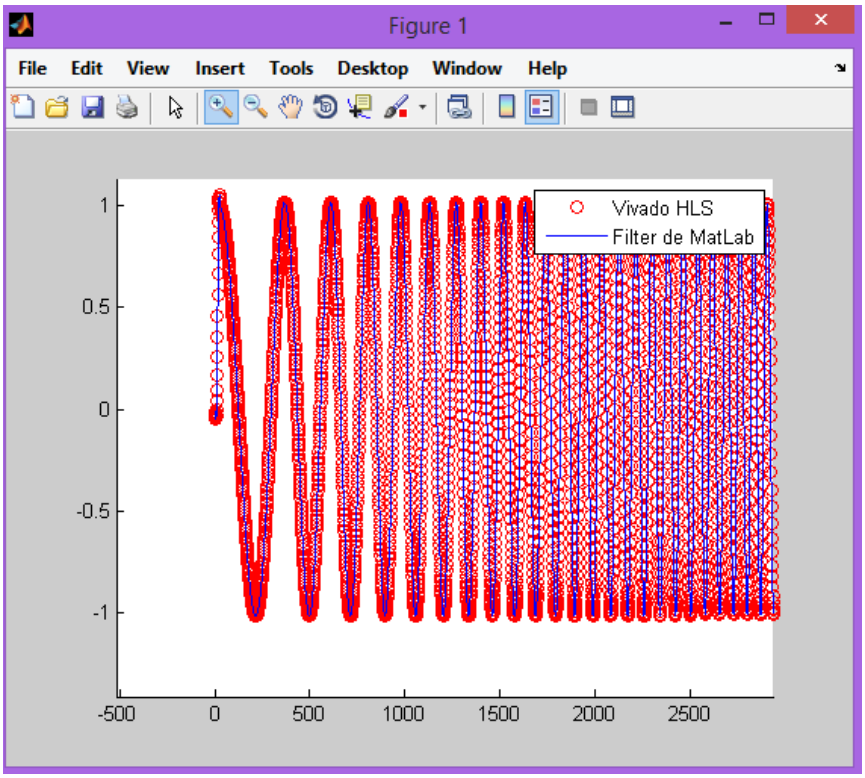

Figura 74 Comparación función Filter con resultado del test bench

Se puede observar como es correspondiente punto a punto el *test bench* de VIVADO HLS con el resultado de la función *Filter* de MatLab, mostrando la precisión del algoritmo y la respuesta del software.

Con la opción que ofrece *VIVADO HLS* de simular el algoritmo en alto nivel, se puede asegurar la funcionalidad del mismo antes de su implementación en Hardware de manera fácil y rápida tanto por la visualización de los resultados por consola, como por el método de simulación a través del *test bench.*

4.2 Resultado sobre Hardware

Una vez lanzado el algoritmo a la ZedBoard desde el software SDK, se imprimen los datos a traves del puerto serial en MatLab para graficar la respuesta del filtro como se muestra en el apartado 3.5.5.

Con el mismo método utilizado en la validación de la respuesta de la simulación, se hace la validación de la respuesta desde hardware. Por lo tanto los datos del filtro se imprimen en MatLab por medio de comunicación serian y se grafica.

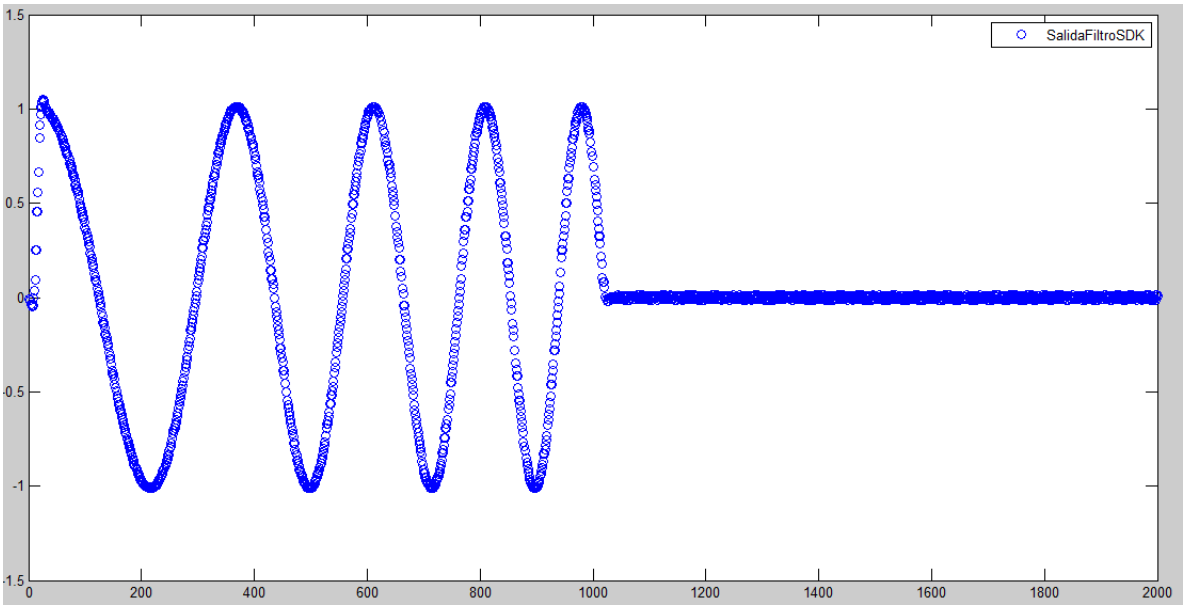

Figura 75 Resultado de aplicación del filtro en la ZedBoard

En la figura 75 se muestra la respuesta del filtro desde la zedboard, donde se procesan 4000 muestras de una señal Chirp, donde las primeras 2000 muestras corresponden a

frecuencias inferiores a la frecuencia de corte del filtro y las otras 2000 muestras son superiores a la frecuencia de corte.

El resultado del filtro aplicado desde la Zedboard, se compara con *TestBench* implementado en *VIVADO HLS* con las mismas muestras (se modifica el vector correspondiente a la señal chirp con las 4000 muestras) usadas en la *ZedBoard* y el resultado se graficará en *MatLab* de igual forma.

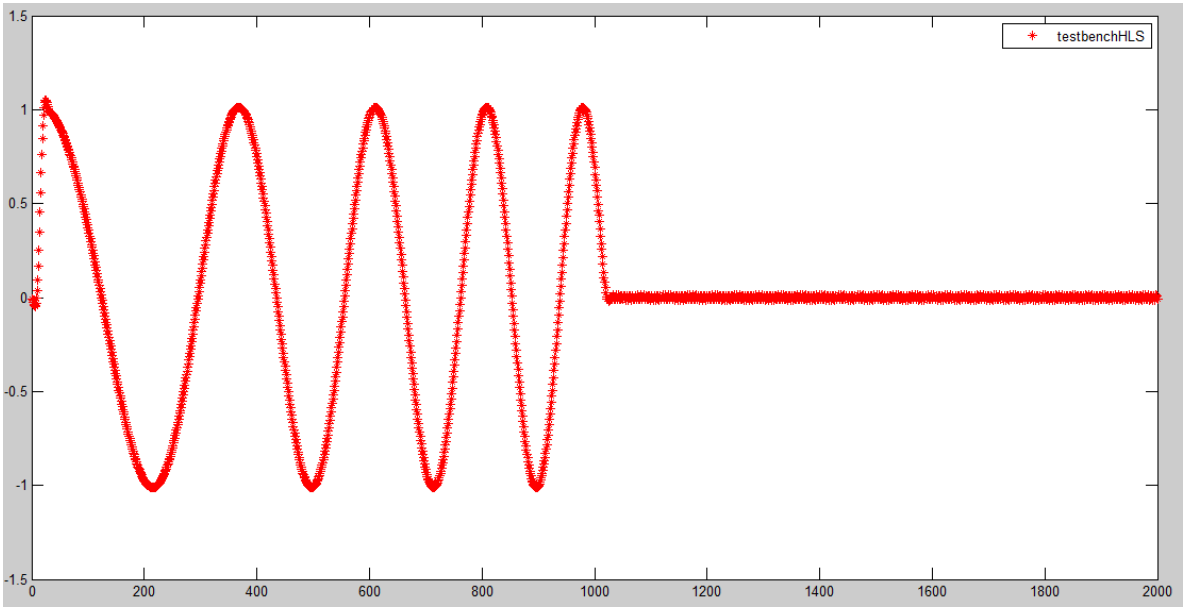

Figura 76 TestBench de Vivado HLS con las mismas muestras usadas en la ZedBoard

Ahora se superponen las señales para validar gráficamente la desviación (Figura 77).

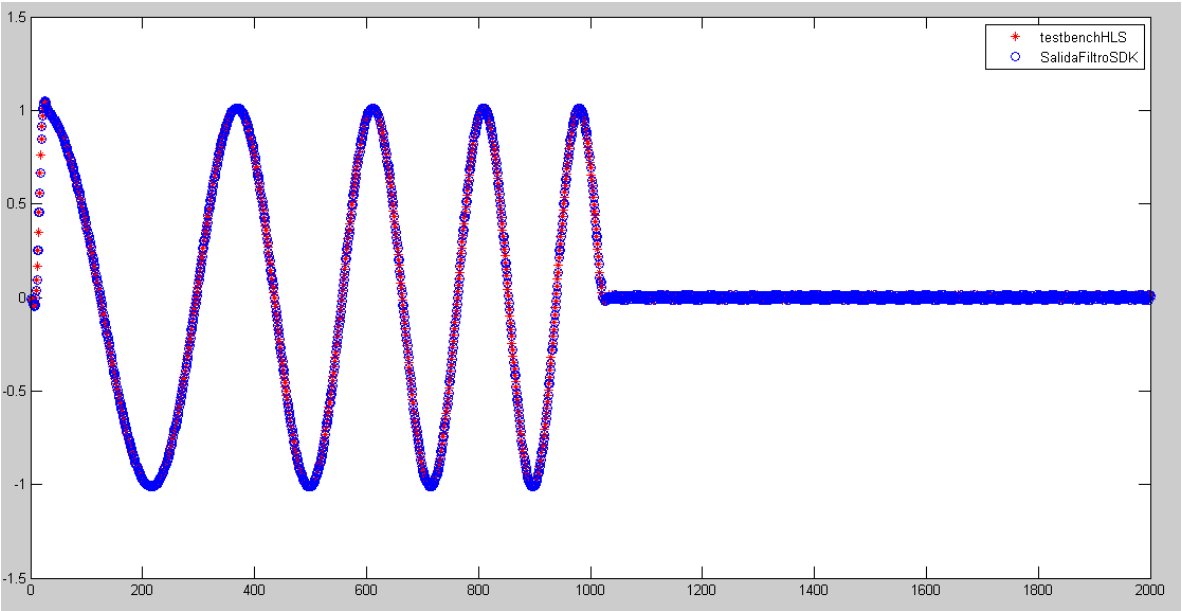

Figura 77 Comparación filtro en ZedBoard y TestBench de Vivado HLS

Como se muestra en la Figura 77, la respuesta de la aplicación en la ZedBoard tiene un comportamiento esperado referente a las pruebas simuladas.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

El propósito de este trabajo consistía en utilizar VIVADO HLS a partir de la construcción de un filtro digital, con la finalidad de entregar al usuario una herramienta de diseño digital con diversas ventajas frente a lo utilizado actualmente.

El tiempo de programación es un factor clave en la entrega de productos de diseño digital tanto a nivel académico como comercial. VIVADO HLS ofrece la opción de realizar la programación del diseño en lenguajes de alto nivel como *C Y C++,* con lo cual el algoritmo se resume a comparación de la programación en *VHDL.*

Con el *Test Bench* y la *Co-Simulation* de la herramienta *HLS* se tiene la posibilidad de verificación de los resultados del diseño previo a su implementación de manera rápida y con reportes de resultados inmediatos que permiten analizar las posibles fallas que pueda tener el diseño. Aunque en esta entrega no se abordó a profundidad todas las opciones que ofrece la herramienta *HLS,* en cuanto a la verificación del algoritmo existe la opción de visualizar el comportamiento de este mediante el *Debug* siendo esta la opción de verificar línea a línea la ejecución del algoritmo.

Los reportes entregados por la herramienta en cuanto a síntesis permiten visualizar los recursos consumidos en el diseño en cuanto a tiempo y hardware.

La herramienta *HLS* permite sintetizar el *RTL* del diseño en un bloque *IP* y establecer directivas de comunicación con lo cual permite de forma fácil y rápida la conexión del procesador con el periférico en VIVADO HLx.

En cuanto a la implementación del diseño sobre la tarjeta, hay limitaciones en cuanto a la visualización del resultado por el *terminal* debido a que los drivers de VIVADO HLS están diseñados para mostrar datos enteros, por lo tanto fue necesario apoyarse sobre otras herramientas y visualizar el resultado en punto flotante usando el puerto serial.

REFERENCIAS

[1] Ortega Lázaro, S. (2014). Implementación de periféricos en vivado para dispositivos ZYNQ.

[2] Safarian, C., Ogunfunmi, T., Kozacky, W. J., & Mohanty, B. K. (2015, July). FPGA implementation of LMS-based FIR adaptive filter for real time digital signal processing applications. In Digital Signal Processing (DSP), 2015 IEEE International Conference on (pp. 1251-1255). IEEE.

[3] Hanbo, L., Shaojun, W., & Yigang, Z. (2015, July). Design of FIR filter with high level synthesis. In Electronic Measurement & Instruments (ICEMI), 2015 12th IEEE International Conference on (Vol. 2, pp. 1067-1071). IEEE.

[4] UltraFast High-Level Productivity Design Methodology Guide, UG1197. 2015.

[5] https://www.youtube.com/channel/UC1ptV25-NEHRIEnM1kXMCrQ

[6] https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/41807/1/04_Lab4.pdf

[7] http://www.ingelec.uns.edu.ar/pds2803/materiales/cap10/12-cap12.pdf

[8] An Evaluation of Vivado HLS for Efficient System Design, Konstantinos Georgopoulos, Grigorios Chrysos, Pavlos Malakonakis, Antonis Nikitakis, Nikos Tampouratzis, Apostolos Dollas, Dionisios Pnevmatikatos, Yannis Papaefstathiou. 2016.

FIRMA ESTUDIANTES Juan Pablo Puerta O.

FIRMA ASESOR

FECHA ENTREGA: __17/03/2017___

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO__ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____