

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Optimización de protocolos de comunicación SDR ejecutados en FPGA.

para optar por el grado académico de
Maestría en Electrónica con Énfasis en Sistemas Empotrados.

Harold Daniel Moreno Urbina

Cartago, 10 de diciembre de 2015.

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

Harold Daniel Moreno Urbina

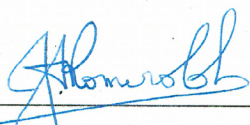
Cartago, 10 de diciembre de 2015

Céd: 1-0123-0456

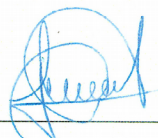
Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Tesis de Maestría
Tribunal Evaluador

Tesis de maestría defendida ante el presente Tribunal Evaluador como requisito para optar
por el grado académico de maestría, del Instituto Tecnológico de Costa Rica.

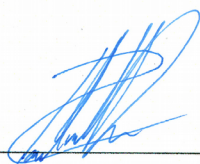
Miembros del Tribunal



Dr. Jorge Arturo Romero Chacón
Profesor Lector



M.Sc. Aníbal Coto Cortéz
Profesor Lector



M.Sc. Pablo Mendoza Ponce
Profesor Asesor

Los miembros de este Tribunal dan fe de que la presente tesis de maestría ha sido aprobada
y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 9 de diciembre de 2015

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica

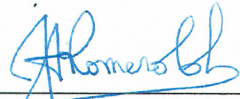
Tesis de Maestría
Tribunal Evaluador
Acta de Evaluación

Tesis de maestría defendida ante el presente Tribunal Evaluador como requisito para optar por el grado académico de maestría, del Instituto Tecnológico de Costa Rica.

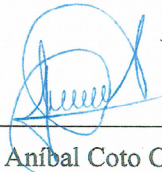
Estudiante: Harold Daniel Moreno Urbina

Nombre del Proyecto: Optimización de protocolos de comunicación SDR ejecutados en FPGA.

Miembros del Tribunal



Dr. Jorge Arturo Romero Chacón
Profesor Lector



M.Sc. Anibal Coto Cortéz
Profesor Lector



M.Sc. Pablo Mendoza Ponce
Profesor Asesor

Los miembros de este Tribunal dan fe de que la presente tesis de maestría ha sido aprobada y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Nota final de la Tesis de Maestría: -90-

Cartago, 9 de diciembre de 2015

Resumen

En este proyecto se muestra una metodología para la optimización de sistemas de comunicación implementados en una plataforma definida por software SDR reduciendo el uso de CPU y memoria. El proyecto se desarrolló con el kit de desarrollo Zedboard basado en el chip Zynq7000 de Xilinx. Este sistema se le denomina híbrido por poseer un sistema de procesamiento, PS, basado en un procesador ARM y lógica programable, PL, correspondiente a una FPGA. PS ejecuta el software GNU Radio bajo una sistema operativo Linux y para la optimización de bloques en hardware se usa el componente de lógica programable PL. Para una efectiva comunicación de datos entre los componente PS y PL se utiliza la interfaz de comunicación Xillybus.

Haciendo uso de las herramientas de Xilinx se implementan dos bloques comúnmente usados en transmisores y receptores de comunicaciones digitales: Filtro FIR y FFT. Estos bloques de hardware sustituyen bloques de software en GNU Radio que realizan la misma función, de este modo se tiene un sistema híbrido en el cual algunas operaciones se realizan en software y otras en hardware.

Se compara el uso de memoria y CPU de la Zedboard para dos bloques con la misma funcionalidad, el primero ejecutado únicamente en software y el segundo en el sistema híbrido. Se obtiene que para filtros FIR de 5, 13 y 25 “taps” el uso de CPU para el bloque de software fue de 84.52%, 77.73% y 66.92% respectivamente, con un uso de memoria de 25% y para el sistema híbrido el uso de CPU se redujo a los valores 15.27%, 8.68% y 4.87% respectivamente, manteniendo un uso de memoria de 24%. Para la FFT el uso de CPU se redujo de un 96.00% a un 32.25% con una reducción del uso de memoria desde 29% a un 24%.

Palabras clave: SDR, GNU Radio, FIR, FFT, Xillybus, Zedboard, Zynq7000, dispositivos híbridos.

Abstract

This project shows a methodology for the optimization digital communication systems implemented on software radios SDR reducing the CPU and memory usage. The project was developed using the kit Zedboard which is based on Xilinx Zynq 7000 series SoC. This SoC is known as hybrid because it counts with a processing systems, PS, based on a ARM processor and a programmable logic based on a FPGA. The PS executes GNU Radio software under a Linux operating System and for the optimizations the communication blocks are implemented on hardware. To communicate PS and PL the Xillybus interface is used.

Using Xilinx software tools two commonly used blocks for digital communication transmission and receptors are implemented: a FIR filter and a FFT. Those blocks replace GNU Radio software blocks that perform the same functionality, allowing to execute the same implementation with two different approaches, one completely on software and one on a hybrid system.

Memory and CPU usage of the Zedboard for two blocks with the same functionality are compared, the first case is a block running entirely in software and the second in the hybrid system. For FIR filters with 5, 13 and 25 taps CPU usage was 4.52%, 77.73% y 66.92% respectively, the memory usage measured was 25%; for the hybrid system CPU usage is reduced to the values between 15.27%, 8.68% y 4.87% respectively with a memory usage of 24%. For FFT CPU usage was reduced from 96.00% to 32.255% with reduced memory usage from 29% to 24%.

Keywords: SDR, GNU Radio, FIR, FFT, Xillybus, Zedboard, Zynq7000, hybrid systems.

A mis padres que ya no se encuentran con nosotros.

A mis hermanos.

Agradecimientos

A mis queridos hermanos Marcos, Jean Pierre y Aby Teresita que siempre han estado ahí apoyando, haciéndome la vida más plena y feliz. A mi grandes amigos José Manuel, Víctor, José Francisco, Cristopher y Michael por acompañarme por tantos y tantos años, se les agradece montones. A la familia Ramírez Zamora, gracias por haberme hecho parte de su familia, les recuerdo siempre con mucho amor. A Marco Villalta, gracias por toda esa ayuda. A mis compañeros en la EIE. Un gran agradecimiento al profesor guía de este proyecto Pablo Mendoza Ponce.

Harold Daniel Moreno Urbina

Cartago, 10 de diciembre de 2015

Índice General

Resumen.....	5
Abstract.....	6
Agradecimientos.....	8
Capítulo 1. Introducción.....	1
1.1 Radios definidas por software ó Software Defined Radio, SDR.....	1
1.2 GNU Radio.....	4
1.3 Transmisión inalámbrica.....	5
1.4 Sistemas híbridos.....	6
1.4.1 DE2i-150.....	7
1.4.2 Zedboard.....	8
1.5 Implementaciones GNU Radio en la literatura.....	10
Capítulo 2. Objetivos y Metodología.....	12
2.1 Objetivos.....	12
2.2 Metodología.....	13
Capítulo 3. Funcionamiento de la plataforma de trabajo.....	15
3.1 Zedboard.....	15
3.2 Xillybus.....	16
3.3 Xilinx en Zedboard.....	18
3.4 Comunicación “host” a FPGA.....	21
3.5 GNU Radio en ZedBoard.....	22
Capítulo 4. Creación de IP cores propios para ser ejecutados en la FPGA.....	25
4.1 IP core base de Xillybus.....	25
4.2 Hardware para FFT de 4 puntos.....	28
4.3 Hardware para filtro FIR.....	30
4.4 Pruebas de verificación para el hardware FFT y FIR.....	33
Capítulo 5. Pruebas de rendimiento.....	42
5.1 Herramienta de Linux “top”.....	42
5.2 Rendimiento del bloque filtro FIR.....	43
5.3 Rendimiento del bloque FFT.....	46
Capítulo 6. Análisis de resultados, conclusiones y recomendaciones.....	49
6.1 Análisis de resultados.....	49
6.2 Conclusiones.....	52
6.3 Recomendaciones.....	54
Bibliografía.....	55
Apéndice A. Códigos.....	60
Apéndice B. Fotografías.....	68

Índice de Figuras

Arquitectura ideal de SDR.....	3
Estructura de bloques de GNU Radio.....	4
Bloques de GNU Radio con GRC.....	5
Diagrama del SoC Zynq-7000.....	8
Vista genérica del remplazo de bloque en GNU Radio. A) Sistema Original. B) Remplazo de bloque para procesado en PL.....	9
Diagrama de bloques genérico de la comunicación entre Xillybus y el dominio PL.....	16
Comunicación entre Xillybus y el dominio PS.....	17
Bloque completo del diseño para la generación del bitstream. El componente principal es el PS de Zynq, se encuentra también memoras, acceso a periférico y los bloques de Xillybus que se detallan en las siguiente figuras.....	19
IP core de Xillybus dentro del bloque completo de diseño.....	20
Bloque Xillybus Lite dentro del bloque de diseño completo mostrado anteriormente.....	20
Sistema completo de ZedBoard ejecutando Xilinx con los respectivos periféricos.....	21
Fotografía de los drivers detectados por Xilinx, entre ellos los correspondientes a Xillybus.....	22
Bloque de GNURadio para la comprobación del funcionamiento de la comunicación entre PS y PL..	24
Generación del archivo de configuración con ISE 14.7.....	26
Implementación base de Xillybus.....	27
Implementación de xillydemo con el componente HLS_wrapper que describe la FFT de 4 puntos.....	30
Estructura del filtro FIR a ser implementado [33].....	31
Respuesta en frecuencia del filtro deseado.....	31
Detalles de diseño del filtro deseado.....	32
"taps" del filtro FIR a implementar.....	32
Implementación del filtro FIR en el proyecto xillydemo.v.....	33
Bloque GNU Radio del filtro FIR.....	35
Salida del filtro FIR en GNU Radio.....	36
Salida del FIR de la segunda señal.....	37
Bloque GNU Radio para ejecutar en la Zedboard.....	38
Bloque GNU Radio para leer datos de salida de la FPGA.....	39
Salida de FIR en FPGA de Zedboard. Entrada suma de senoidales de 200 y 600 Hz respectivamente..	39
Salida de FIR en FPGA de Zedboard. Entrada suma de senoidales de 200 y 250 Hz respectivamente..	40
Toma de pantalla del comando top de la Zedboard con el S.O. GNURadio-Zynq.....	43
Toma pantalla de top para filtro FIR de 13 "taps" ejecutado en PS únicamente.....	45
Toma pantalla de top para filtro FIR de 13 "taps" ejecutado en PS y PL.....	46
Toma pantalla de "top" para ejecución en PS de FFT.....	47
Toma pantalla de "top" para ejecución híbrida de FFT.....	48
USRP y SBX, vista interna del equipo.....	68
Sistema USRP con su respectiva antena, conectado a la computadora mediante Ethernet.....	69
DE2i-150 ejecutando Ubuntu 12.04.....	70
Zedboard ejecutando imagen de "booteo" de fábrica.....	71

Índice de Tablas

Tabla 1: Rendimiento %CPU de filtros FIR ejecutados únicamente en zona PS.....	44
Tabla 2: Rendimiento %CPU de filtros FIR ejecutados en ambas zonas PS y PL.....	44
Tabla 3: Rendimiento %CPU de transformada FFT de cuatro puntos en PS y en PS y PL.....	47

Lista de símbolos y abreviaciones

ADC	Analog to Digital Converter, Convertidor Analógico a Digital.
AM	Amplitud Modulada.
AMBA	Arquitectura Avanzada de Microcontrolador de Bus. Advanced Microcontroller Bus Architecture.
AXI	Interfaz Avanzada Extendible. Advanced eXtensible Interface.
BER	Razón de bit erróneos. Bit Error Rate.
BLER	Razón de Bloques Erróneos. Block Error Rate.
COV	Detección Basada en COVarianza. COVariance Based Detection.
CPU	Unidad Central de Procesamiento. Central Processing Unit.
DAC	Digital to Analog Converter. Convertidor Digital a Analógico.
DFT	Transformada Discreta de Fourier. Discrete Fourier Transform.
DQPSK	Differential Quadrature Phase-Shift Keying.
DSP	Digital Signal Processing. Procesamiento Digital de Señales.
ED	Detección de Energía. Energy Detection.
EVM	Magnitud del Vector de Error. Error Vector Magnitude.
FAT32	Tabla de posicionamiento de archivos de 32 bits. File Allocation Table of 32 bits.
FM	Frecuencia Modulada.
FFT	Fast Fourier Transform, Transformada Rápida de Fourier.
FIFO	First In, First Out. Primero en llegar, primero en salir.
FIR	Finite Impulse Response. Respuesta Finita al Impulso.
HDL	Lenguaje de Descripción de Hardware. Hardware Description Language.
HLS	High Level System.

IAEA	International Atomic Energy Agency, Agencia Internacional de Energía Atómica.
ICTP	International Centre for Theoretical Physics. Centro Internacional de Física Teórica
IP	Propriedad Intelectual. Intellectual Property.
LUT	Tabla de Consulta. Lookup Table.
MME	Autovalores Mínimos y Máximo. Maximum and Minimum Eigenvalue.
OFDM	Orthogonal Frequency Division Multiplexing.
PC	Computadora Personal. Personal Computer.
PL	Lógica Programable. Programmable Logic.
PRR	Razón de Paquetes Recibido. Packet Received Ratio
PS	Sistema de Procesamiento. Processing System.
RAM	Memoria de Acceso Aleatorio. Random Access Memory.
RF	Radiofrecuencia.
RTFS	Directorio de Archivos de Raíz. Root File System.
SD	Protección Digital. Secure Digital.
SDR	Radios SDefinidos por Software. Software Defined Radio.
SNR	Razón Señal a Ruido. Signal-to-Noise Ratio.
SO	Sistema Operativo.
SoC	Sistema en Chip. Sistem on Chip.
SWIG	Envoltura Simplificada y Generador de Interfaz, Simplified Wrapper and Interface Generator.
TETRA	Terrestrial Trunked Radio
USRP	Universal Serial Radio Peripheral.
VSZ	Tamaño de Memoria Virtual. Virtual memory SiZe.

Capítulo 1. Introducción

En este informe se presenta la implementación y optimización de un sistema de comunicaciones basado en Software Defined Radio (SDR) ejecutados en plataformas híbridas entre FPGA (Field Programmable Gate Array) y un procesador de sistema general con arquitectura ARM. La optimización consiste en implementar el sistema SDR que se ejecutan en computadoras con procesadores de propósito general (x86 o ARM) únicamente, hacer un perfil de funcionamiento del mismo y detectar que bloques o sub-sistemas del mismo pueden ser mejorados según criterios de rendimiento para luego ser remplazados por un sistema híbrido en el cual partes del sistema serán ejecutados por una FPGA y otras en un procesador ARM. El rendimiento de ambas implementaciones es comparado según los criterios de rendimiento previamente establecidos para medir el impacto del uso de sistema híbrido.

En esta introducción se describen los conceptos básicos de SDR, GNU Radio, la implementación del sistema de comunicación y los kit de desarrollo híbridos DE2i-150 y Zedboard.

1.1 Radios definidas por software ó Software Defined Radio, SDR.

Tradicionalmente los sistemas de comunicaciones como radio AM y FM, televisión y las primeras generaciones de sistemas de telefonía móvil se han construido con componentes de hardware, a saber, filtros, moduladores / demoduladores, amplificadores, etc. Estos sistemas tienen como característica que el hardware cumple una única función y ésta no puede ser variada de forma sencilla lo que presente un problema si se desea cambiar de protocolo o bien hacer mejoras en el mismo.

Para solventar algunos de los problemas derivados de esto y con la ayuda de los avances de la

electrónica digital se ha recurrido a sustituir algunas o todas las funcionalidades de los componentes analógicos por componentes programables, una de las opciones más recientes es el concepto de Software Defined Radio que se define más formalmente como [1]: un radio transceptor en el cual los parámetros claves son definidos por software y en el cual los aspectos fundamentales de la operación de radio pueden ser reconfigurados mediante una actualización de dicho software.

El *Wireless Innovation Forum* ó *WI Forum* [2], anteriormente conocido como *SDR Forum* es una organización sin fines de lucro que tiene como objetivo la innovación en el uso del espectro radioeléctrico y el avance de las tecnologías de radio que apoyan el desarrollo de tecnologías mundiales críticas o esenciales. Según el WI Forum, la definición de SDR viene a ser [3]: Radio en el que alguno o todos los componentes funcionales de la capa física son definidas por software.

Entre los beneficios de la utilización de SDR para el desarrollo de sistemas de comunicaciones se puede enumerar los siguientes [1][2] :

- Economía de escala.
- Roaming global.
- Actualización y mejoramiento de servicios.
- Modulación y codificación adaptativas.
- Desarrollo de familias de productos usando una arquitectura común.
- Reutilización del software para nuevos productos, reduciendo el costo de desarrollo.

La Figura 1 muestra la arquitectura ideal de SDR [1][4]:

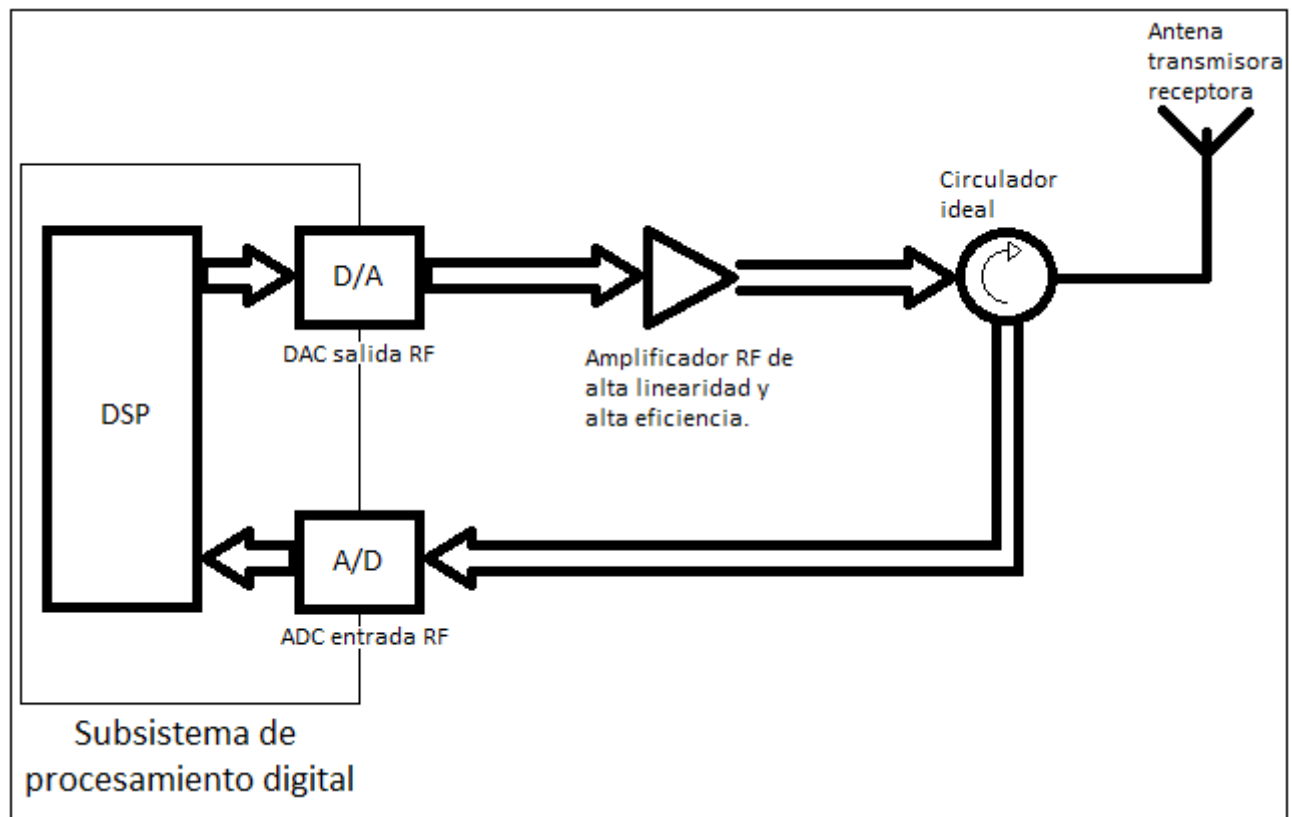


Figura 1: Arquitectura ideal de SDR

En esta arquitectura el bloque denominado como *DSP* (*Digital Signal Processing*, Procesamiento Digital de Señales) se encarga del protocolo, canalización, esquema de modulación, ecualización para la transmisión y recepción y cualquier otra solución de procesamiento. El amplificador se encarga de hacer la correcta transferencia de la señal del *DAC* (Convertidor Digital Analógico ó *Digital-to-Analog Converter*) a una señal de mayor potencia para ser transmitida correctamente por la antena. Los convertidores *DAC* y *ADC* (Convertidor Analógico a Digital ó *Analog-to-Digital Converter*) tienen la función de servir como frontera entre las señales analógicas transmitidas y recibidas por la antena y el bloque de procesamiento digital. El circulador lineal es utilizado para la separación de las señales de recepción y transmisión de la antena.

1.2 GNU Radio

GNU Radio [5] es un software libre y de código abierto que sirve como herramienta de desarrollo para la implementación de bloques de procesamiento SDR. Este software además de poder ser usado solamente como una herramienta de simulación tiene la capacidad de ser usado con hardware externo de radiofrecuencia (RF).

GNU Radio está basado en C++ y Python y consiste en básicamente dos componentes fundamentales que son los *blocks* (bloques) y *flow graph* (grafo de flujo) [6][7][8], de este modo los bloques que hacen el procesamiento de los datos están escritos en código C++ y la comunicación entre bloques se hace a través de código escrito en Python. La interpretación entre los dos lenguajes se hace a través de SWIG (*Simplified Wrapper and Interface Generator*, Envoltura Simplificada y Generador de Interfaz). La Figura 2 muestra la estructura básica de GNU Radio y la Figura 3 muestra un ejemplo sencillo de un bloque generado con la interfaz gráfica de GNU radio GRC (*GNU Radio Companion*).

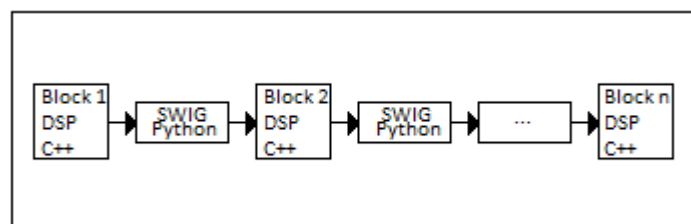


Figura 2: Estructura de bloques de GNU Radio.

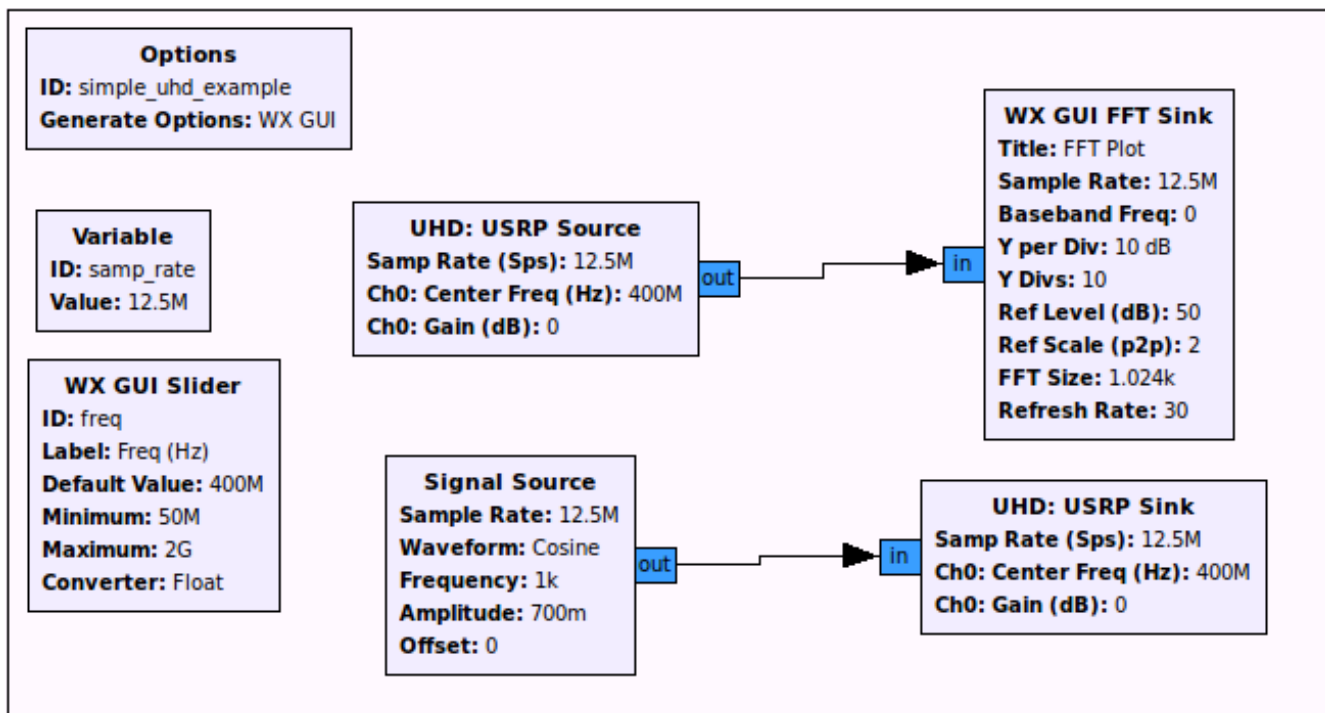


Figura 3: Bloques de GNU Radio con GRC.

Existe una amplia variedad de bloques disponibles en GNU Radio [9] como modulaciones de frecuencia, cálculos matemáticos, filtros, modelos de canales, codificadores, etc. Del mismo modo, estos bloques se pueden editar o el usuario puede crear sus propios bloques siguiendo los correspondientes lineamientos.

1.3 Transmisión inalámbrica.

Existe una diversidad de hardware compatible con GNU Radio [10] con rangos de precios que varían desde pocas decenas de dólares hasta varios miles. Para este proyecto se cuenta con equipo para SDR de la empresa Ettus Research [11] (propiedad de la Universidad de Costa Rica), específicamente se

cuenta con la plataforma USRP N210 [12] y las tarjetas de RF SBX [13]. El periférico USRP se conecta con una computadora que ejecute GNU Radio a través de Ethernet. El USRP se encarga junto con las tarjetas SBX, de transmitir en la frecuencia deseada la información en banda base que se genera con el GNU, proceso conocido como *upconversion*; del mismo modo se puede hacer la recepción de información, en este caso *downconversion*, para el correspondiente procesamiento en la computadora utilizando GNU Radio. Con esto, se puede realizar una transmisión y recepción completas desde una computadora, a través del USRP, recibiendo la información en un segundo USRP la información y pasándola a una segunda computadora en la que el GNU Radio hace el correspondiente procesamiento; teniendo de este modo un sistema de comunicación inalámbrico completo de punto a punto.

Para remplazar la computadora en donde se ejecuta el GNU Radio se ha trabajado con dos sistemas híbridos que se detallarán en la siguiente sección, estos son DE2i-150 de Terasic y ZedBoard, la finalidad del uso de estos sistemas híbridos es mejorar el rendimiento con la utilización de la FPGA en lugar de la utilización única de procesadores de propósito general, aprovechando al mismo tiempo las ventajas inherentes de estos sistemas.

1.4 Sistemas híbridos.

La parte clave de este proyecto es el remplazo de una computadora con un procesador de propósito general corriendo un sistema operativo completo, en donde se procesan los datos de transmisión/recepción en banda base utilizando el GNU Radio. Se trabajó inicialmente con el kit de Terasic Inc. DE2i-150 [12] y luego con la Zedboard [13].

El término “híbrido” se refiere a un sistema que combina al menos dos sistemas computacionales con distinta naturaleza. En el caso de este proyecto se combina un sistema computacional de propósito

general x86 y ARM, que ejecutan la lista de instrucciones de manera secuencial, al que se le denomina PS (Processing System) con hardware de lógica programable descrita con lenguaje HDL (Hardware Description Language) para FPGA, a la que se le denomina PL (Programmable Logic) que es capaz de procesar datos y señales de forma paralela. Los dos dominios de procesamiento PS y PL requieren ser comunicados mediante una interfaz para poder intercambiar datos, buscando así optimizar el sistema completo asignando operaciones diferentes a cada dominio. Las siguientes dos secciones describen los dos sistemas híbridos con los que se trabajó. En el capítulo 3 se detalla la interfaz utilizada para comunicar los sistemas PS y PL.

1.4.1 DE2i-150

El sistema DE2i-150 cuenta, en términos generales, con un procesador Intel Celeron Dual Core Processor N2600 y una FPGA Cyclone IV EP4CGX150DF31. Para propósitos de este proyecto se ha instalado una versión completa del sistema operativo Ubuntu 12.04 y sobre este se corre el GNU Radio. Se ha logrado hacer una transmisión exitosa entre el sistema DE2i-150 y una computadora portátil para la recepción mediante la utilización de los dos USRP disponibles.

A pesar de intentar por varios métodos no se ha podido transmitir datos desde un bloque del GNU Radio a la FPGA para intentar reemplazar bloques con dos objetivos claros, reducir la carga del CPU y ejecutar de manera más rápida algunos bloques del sistema de comunicación, como pueden ser filtros o FFT (*Fast Fourier Transform* o Transformada Rápida de Fourier).

1.4.2 Zedboard

Durante un curso conjunto del ICTP (International Centre for Theoretical Physics, Centro Internacional de Física Teórica) y la IAEA (International Atomic Energy Agency, Agencia Internacional de Energía Atómica) se cedió en forma de préstamo una tarjeta Zedboard para el desarrollo de este proyecto.

Este “kit” de desarrollo cuenta como componente principal el SoC (System on Chip) Zynq-7000 de la empresa Xilinx, específicamente el Zynq®-7000 All Programmable SoC XC7Z020-CLG484.

Dicho SoC cuenta con un procesador ARM de doble núcleo cortex A-9 (Dual ARM® Cortex™-A9 MPCore™ with CoreSight™) y una FPGA Artix-7 [14] . La Figura 4 muestra un diagrama de dicho SoC.

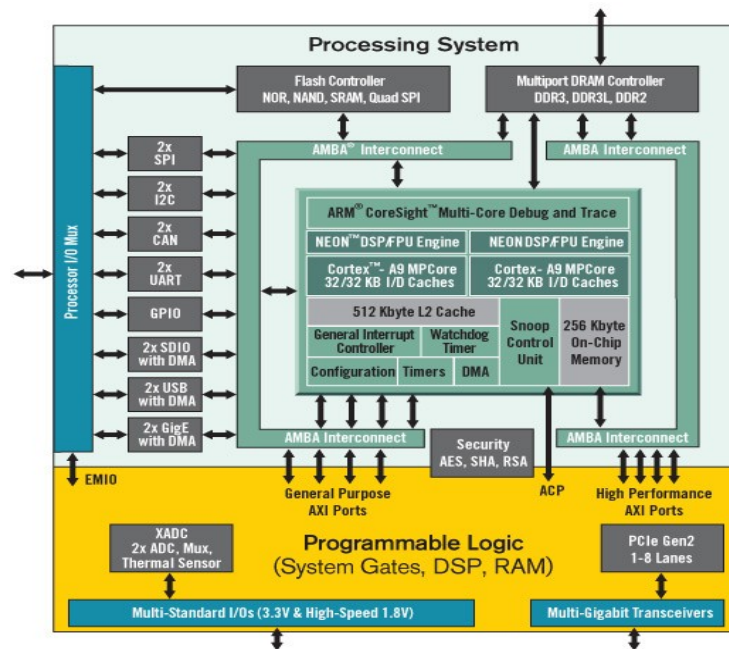


Figura 4: Diagrama del SoC Zynq-7000

Dichos sistema cuenta con dos partes claramente diferenciadas, estas son la parte de lógica programable (denominada como PL – Programmable Logic-, amarilla en la Figura 4) que contiene la FPGA y la

parte de sistema de procesamiento (denominado PS -Processing System-, en celeste en la Figura 4).

Una parte clave de este sistema es la interfaz de comunicación entre el PS y el PL, que se hace utilizando la interfaz abierta AMBA creada por ARM [15] haciendo uso de los puertos AXI (Advanced eXtensible Interface).

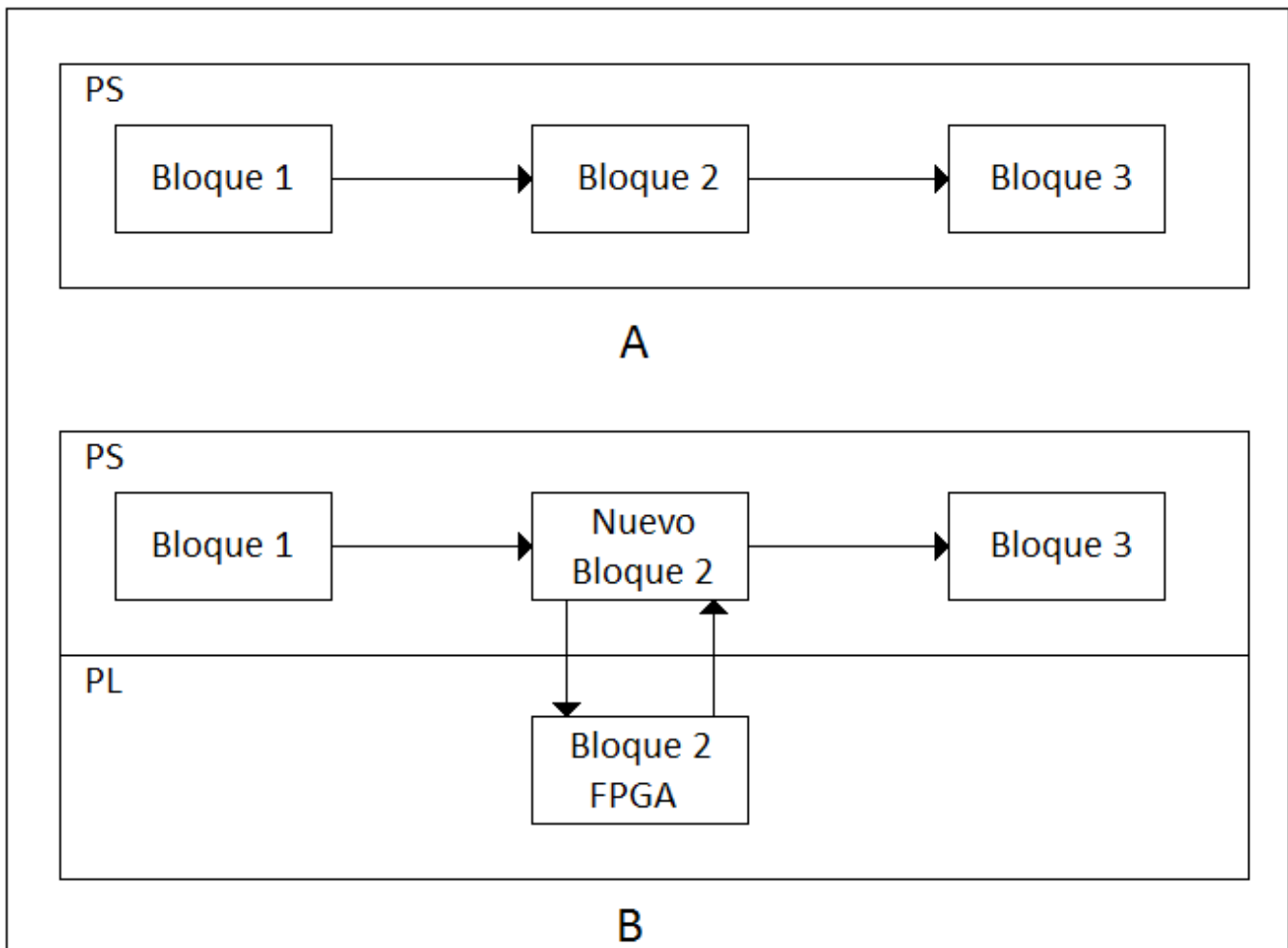


Figura 5: Vista genérica del reemplazo de bloque en GNU Radio. A) Sistema Original. B) Reemplazo de bloque para procesado en PL.

Para efectos de este proyecto se ejecuta una versión de Linux en el sistema PS, este sistema tiene instalado el software GNU Radio, este software ejecuta bloques típicos encontrados en sistemas de comunicación, primero se hace enteramente en la parte PS del sistema, esto es, los bloques son ejecutados enteramente en software. Como segunda etapa, se reemplaza uno o más bloques de software

por bloques de hardware en el dominio PL del SoC Zynq7000. La interfaz de comunicación entre ambos dominios, PS y PL, se detalla en el capítulo 3. La Figura 5 muestra globalmente como se realiza el co-procesamiento. La parte de PL del SoC se utilizará para remplazar bloques del GNU radio con la finalidad de mejorar el rendimiento general del sistema. Esto es, un bloque completo de GNU Radio se cambiará por otro bloque que envía los datos a procesar a la FPGA y luego recibirá los datos de la FPGA ya procesados que serán a su vez enviado en el formato correspondiente para que se siga operando con el GNU Radio.

1.5 Implementaciones GNU Radio en la literatura.

Varias implementaciones de sistemas de comunicaciones utilizando la combinación de GNU Radio y el hardware USRP han sido encontradas en la literatura técnica. Abirami [16], Tucker[17] y Serban [18] dan una guía genérica de como realizar proyectos de GNU Radio usando USRP cubriendo aspectos básicos del diseño.

Ren [19] hace una implementación de una parte del estándar TETRA (Terrestrial Trunked Radio) correspondientes a una modulación $\pi/4$ DQPSK, utilizando dos algoritmos diferentes para la elección de los símbolos de la señal en la recepción, no se da ninguna explicación de por qué se eligieron esos dos algoritmos específicos y únicamente se eligió como mejor el que mostró el BER (Bit Error Rate) mas bajo.

Marwanto [20] y Berardinelli [21] hacen implementaciones de sistemas OFDM (Orthogonal Frequency Division Multiplexing), que es un esquema con alta prevalencia en los sistemas de comunicaciones inalámbricos de última generación actuales. Marwanto hace una medición del rendimiento de su implementación haciendo el cálculo de la razón de paquetes erróneos con respecto al total de envíos,

denominándolo como PRR (Packet Received Ratio). Berardinelli hace un estudio de rendimiento más profundo, midiendo la razón de señal a ruido SNR (Signal-to-Noise Ratio) en función de la potencia de entrada, la magnitud del vector de error EVM (Error Vector Magnitude) y la razón de bloques erróneos BLER (Block Error Rate) muy similar a los BER y PRR de los autores anteriores.

Zitouni [22] , Mate [23] y Ferreira [24] hacen estudios relacionados a la respuesta en frecuencia de la combinación de GNU Radio con USRP. Zitouni se enfoca en dos mediciones específicas, a saber, ancho de banda RF y la potencia de salida media en función del valor del convertidor DAC. Mate hace su estudio buscando formas de detección de señales, utilizando varias métricas como detección de energía ED (Energy Detection), detección basada en covarianza COV (Covariance Based Detection) y una detección basada en autovalores máximos y mínimos MME (Maximum and Minimum Eigenvalue). Ferreira propone un método para la medición de la respuesta en frecuencia de las tarjetas de recepción y transmisión conectadas al USRP, estas mediciones incluyen la potencia y frecuencia de las señales detectadas.

Truong [25] hace un extenso estudio de las fuentes de la latencia que existe en la combinación de GNU Radio y USRP, investigando las posibles latencias generadas por el protocolo en estudio y del sistema en sí que puede ser debido interrupciones del sistema operativo, del tamaño de los buffers de los datos a enviar, etc. Da conclusiones en función de experimentos hechos con tamaños de buffers diferentes, encontrando una relación entre tiempos de latencia y el tamaño de estos.

No se encontró en la literatura técnica artículos relacionados a la optimización de parámetros específicos de rendimiento de las FPGA para algoritmos de comunicación basados en SDR o del rendimiento del USRP. Los parámetros de interés a ser optimizados en este proyecto son el uso de CPU y memoria.

Capítulo 2. Objetivos y Metodología.

2.1 Objetivos.

Objetivo Principal:

El objetivo principal se puede resumir como:

- Estudio comparativo del impacto sobre el consumo de recursos computacionales, en una arquitectura empotrada, por parte de un sistema de radio de software SDR implementado completamente en software y una implementación híbrida (software y hardware).

Se compara el consumo de CPU y memoria en una arquitectura empotrada cuando se ejecuta un sistema SDR implementado completamente en software y cuando se implementa el mismo sistema sustituyendo algunos bloques por aceleradores en hardware en el mismo chip. La comparación se realiza utilizando la misma tarjeta de desarrollo, la cual posee un FPGA con un Hard-Processor ARM.

Objetivos específicos:

- Ejecución de GNU Radio en un sistema híbrido. Se busca tener un sistema Linux en PS (Processing System) y que este sea capaz de ejecutar GNU Radio.
- Intercomunicar los sistemas PS y lógica programable ó PL (Programmable Logic). Se busca que el componente PS sea capaz de enviar y recibir datos de la FPGA del sistema híbrido.
- Descripción en *HDL* de los bloques FFT y FIR. Corroborar el correcto funcionamiento de ambos bloques.

- Creación del bloque en GNU Radio para la comunicación entre el procesador de propósito general y la FPGA. Se busca crear un bloque de GNU Radio que sea capaz de enviar datos a la FPGA y recibir de vuelta datos procesados. La FPGA realiza las funciones de FIR y FFT descritas previamente.
- Medición del rendimiento del sistema, comparativa con el sistema original. Se usa herramientas de Linux para medir el uso de memoria y CPU para comparar el rendimiento de ambos sistemas.

2.2 Metodología.

En esta sección se detalla la metodología propuesta para la realización de este proyecto. En líneas generales se va a seguir uno a uno los diferentes objetivos específicos que unidos llevan al cumplimiento del objetivo general del proyecto.

Como paso inicial se eligió dos operaciones matemáticas muy comunes en los sistemas de comunicación y que tienen una estructura algorítmica que permite una alta paralelización para el cálculo del resultado o resultados de salida, estos son una FFT (Fast Fourier Transform, Transformada Rápida de Fourier) y un filtro FIR (Finite Impulse Response Filter, Filtro de Respuesta Finita al Impulso).

Como se reemplazan bloques de software por bloques de hardware implementado en una FPGA, se requiere explotar una de las características inherentes de las FPGA que es la paralelización, esto es, la realización de múltiples operaciones, finales o intermedias, al mismo tiempo. Esto estructuralmente representa una ventaja con respecto a la forma en la cual un procesador de propósito general (como

sistemas basados en x86 o ARM) realiza cálculos, ya que estos se hacen de manera secuencial.

Se describe en HDL los bloques elegidos para ser remplazados, para esto se utiliza las herramientas de Xilinx incluidas con la licencia de la Zedboard.

Sustitución de los diferentes bloques en GNU Radio por bloques que se comunican con la FPGA, estos bloques tienen la tarea de recolectar los datos a ser enviados a la FPGA, enviarlos, recibir los datos procesador y el envío de ellos mismos al siguiente bloque según el sistema original, según se mostró en la Figura 5.

Una vez hecho esto se debe comprobar el correcto funcionamiento del sistema al completo, esto es, el sistema híbrido con los nuevos bloques de GNU Radio que utilizan la FPGA para remplazar los bloques originales.

Finalmente por medio de mediciones realizadas en los sistemas original y optimizado, se mide el rendimiento del sistema completo, entre ellos el uso de la CPU, manejo de memoria, verificando siempre que los cálculos se hacen de manera correcta.

Capítulo 3. Funcionamiento de la plataforma de trabajo.

En este capítulo se detalla todo el proceso de puesta a punto de la plataforma de trabajo. Esto incluye, funcionamiento básico de la tarjeta Zedboard según su distribución de fábrica, la instalación de una distribución de Linux en el procesador ARM del SoC Zynq-7020 de la Zedboard mediante la distribución Xillinux, que incluye la interfaz de transmisión de datos entre el procesador y la FPGA del SoC Zynq-7020. En una etapa posterior se comprueba el correcto funcionamiento de la comunicación de la transmisión de datos entre el procesador y la FPGA a través de la interfaz Xillybus. Finalmente se muestra código en C que se comunica con un bloque propio en la FPGA. Este último paso es una etapa preliminar para la consecución del objetivo principal de este proyecto ya que representa los cimientos en los que se desarrolla el co-procesamiento, esto es, descargar en la FPGA parte del trabajo computacional del sistema de comunicación con el de optimizar el rendimiento general del sistema.

3.1 Zedboard.

La tarjeta de desarrollo Zedboard viene de fábrica con una tarjeta micro SD para booteo de un sistema básico Linux con funcionalidades sencillas. Este sistema soporta el uso de monitor, UART y otros periféricos.

Entre las características de la arquitectura del área PL del SoC relevantes para este proyecto se puede enumerar:

- FPGA Artix-7.
- 85K celdas lógicas.

- 53 200 LUT (Lookup Table, Tabla de Consulta).
- 106 400 Flip-Flops.
- 560 KB de bloques RAM.
- 220 componentes DSP.

3.2 Xillybus

La creación de una interfaz de transmisión de datos entre los dominios PS y PL puede resultar en sí mismo un proyecto de gran envergadura. Por esta razón se eligió para este proyecto la utilización de Xillybus, que es una solución genérica que consiste en un “IP Core” que utiliza señales específicas y FIFOs para el trasiego de la información. Un diagrama genérico de esto se muestra en la Figura 6.

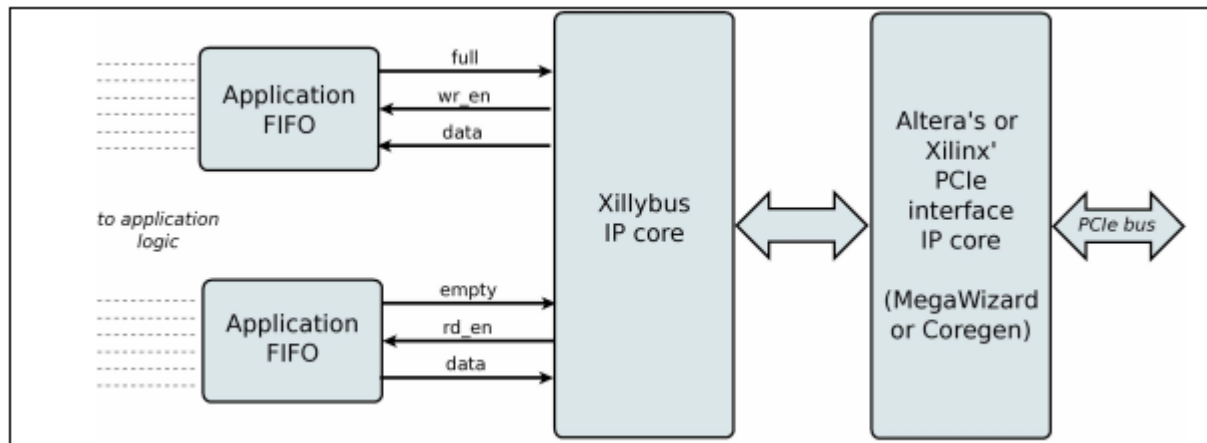


Figura 6: Diagrama de bloques genérico de la comunicación entre Xillybus y el dominio PL.

Es esta Figura se muestra en la parte izquierda lo que sería el dominio PL, esto es, la lógica

programable que resulta ser los bloques hechos para realizar este proyecto, se usan FIFO para comunicarse con el IP Core de Xillybus y de ahí al bus de datos para la comunicación con la PS, comunicación que se muestra en la Figura 7.

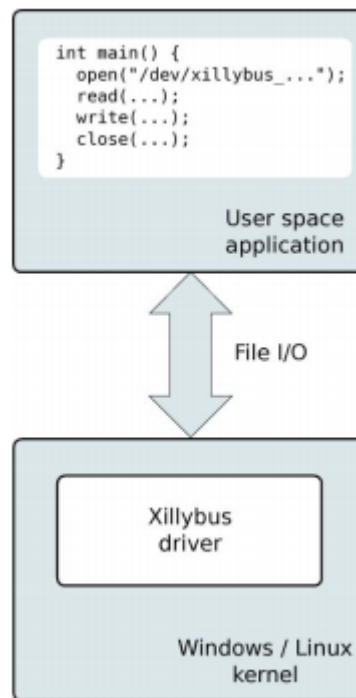


Figura 7: Comunicación entre Xillybus y el dominio PS.

En este caso el dominio PS mostrado como un pseudo-código. La línea “/dev/xillybus_something” representa el dispositivo de Xillybus que ha sido detectado por el kernel Linux del PS. Las líneas “read” y “write” representa la forma en que se lee o escribe datos a la FPGA desde un código ejecutado PS, ya que el dispositivo es visto como un archivo. El IP Core de Xillybus se encarga de esta manera de pasar los datos entre los componentes PS y PL, permitiendo de esta manera tener periféricos empotrados dentro de la FPGA que son capaces de comunicarse con el sistema de procesamiento PS.

3.3 Xillinux en Zedboard.

Xillinux [26] es una distribución de Linux basada en Ubuntu 12.04 que tiene como principal característica proveer un ambiente con los controladores necesarios para comunicarse con el bloque de hardware Xillybus. Para el caso específico de este proyecto se escogió una distribución para Zedboard, que se diferencia de las otras distribuciones en que tanto la distribución Xillinux como el dominio PL se ejecutan en el mismo SoC, a diferencia de otras FPGA en las que el sistema operativo se ejecuta en una computadora o “host” aparte.

La ejecución de Xillinux en la Zedboard se hace a través de una tarjeta microSD. A esta tarjeta se le debe crear dos particiones:

- Una partición ext4 montado por Linux, aquí se encuentra lo que viene a ser la distribución Xillinux.
- Una partición FAT32 de arranque. Esta partición contiene los archivos necesarios para el arranque, los binarios para la ejecución del kernel para Xillinux de la otra partición así como el Bitstream de configuración de la FPGA (esto es, el PL).

La partición ext4 se realiza de manera relativamente sencilla usando la distribución que se encuentra en la página Xillybus.com. Del mismo modo, varios de los archivos de la partición de arranque sencillamente se copian a esta partición según como se bajan de la página a excepción del archivo de configuración de la FPGA, ya que este se debe crear ya sea con Vivado o ISE. Las siguientes Figuras 8, 9 y 10 muestran los bloques de diseño generados con Vivado 2015.2.

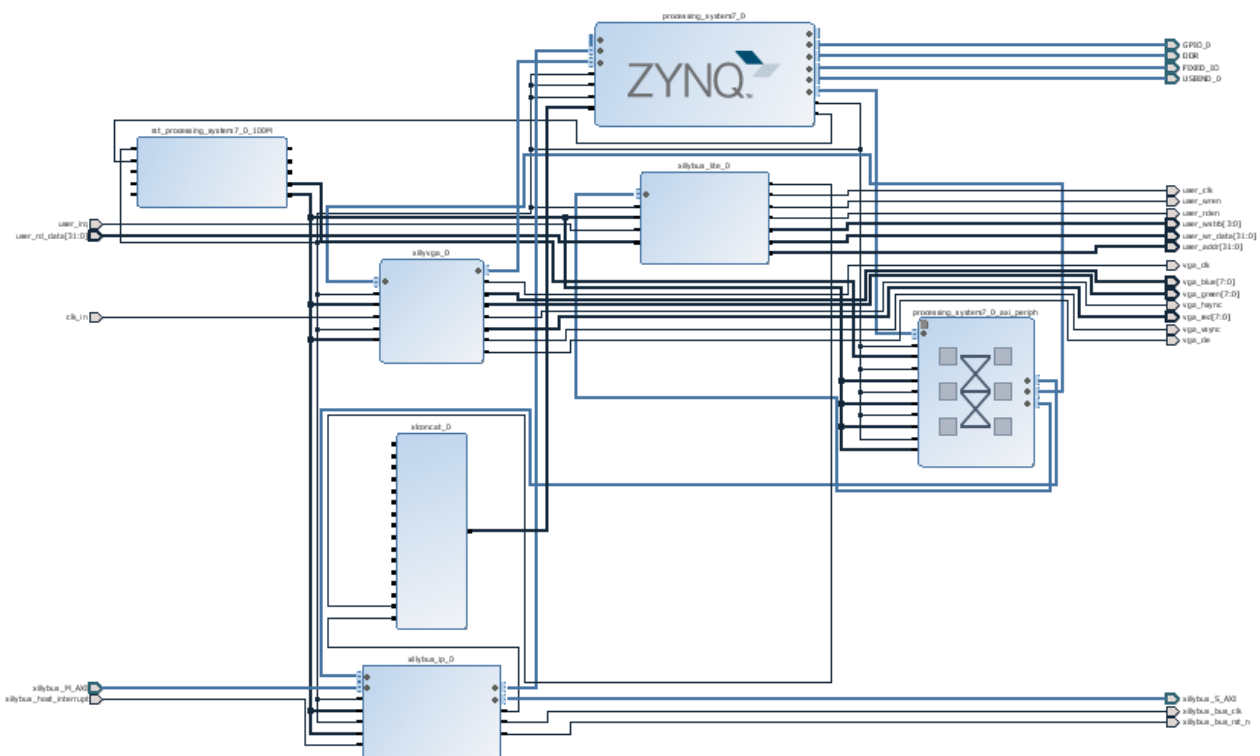


Figura 8: Bloque completo del diseño para la generación del bitstream. El componente principal es el PS de Zynq, se encuentra también memoras, acceso a periférico y los bloques de Xillybus que se detallan en las siguiente figuras.

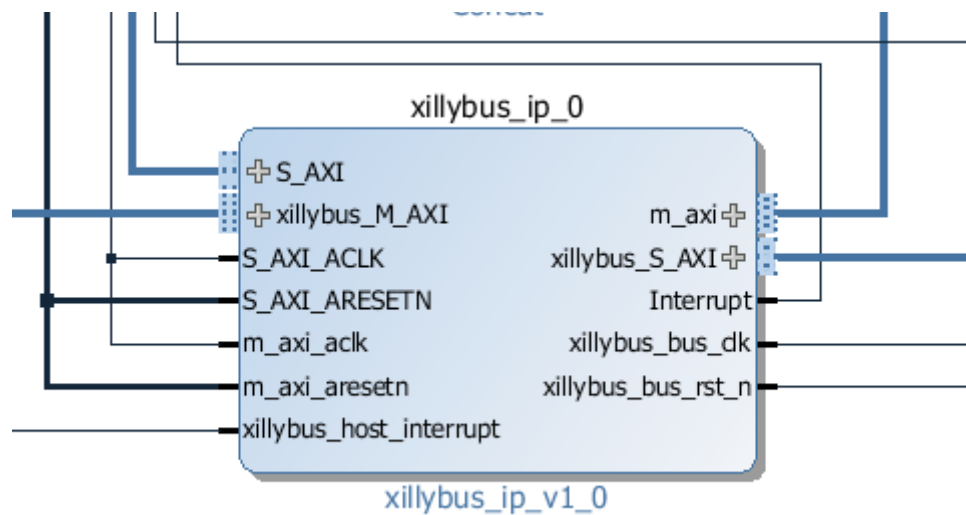


Figura 9: IP core de Xillybus dentro del bloque completo de diseño.

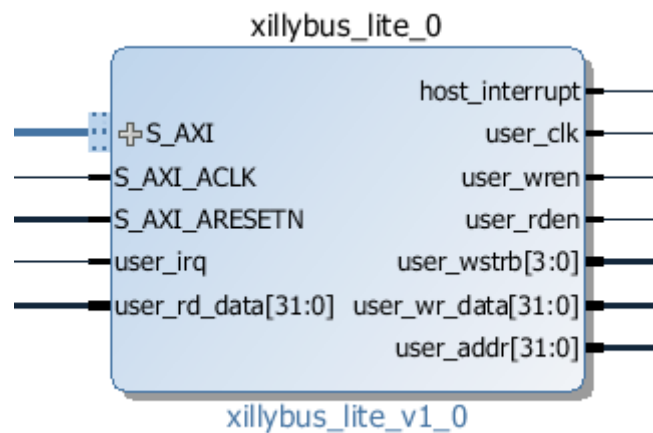


Figura 10: Bloque Xillybus Lite dentro del bloque de diseño completo mostrado anteriormente.

Luego de esto se genera el bitstream, que se coloca en la tarjeta microSD en la partición de arranque.

Una vez hecho esto, se puede hacer la prueba de funcionamiento dentro de la Zedboard.

La Figura 11 muestra la Zedboard conectada a monitor, teclado y mouse con el sistema Xillinux ejecutándose.



Figura 11: Sistema completo de ZedBoard ejecutando Xillinux con los respectivos periféricos.

3.4 Comunicación “host” a FPGA.

Una vez la plataforma está en funcionamiento fue necesario corroborar que se ha cargado correctamente los drivers para la comunicación entre PS y PL. La Figura 12 es una fotografía de los dispositivos detectados por el PS, en los que se destacan los correspondientes a Xillybus a saber:

- xillybus_mem_8
- xillybus_read_32
- xillybus_read_8
- xillybus_write_32
- xillybus_write_8

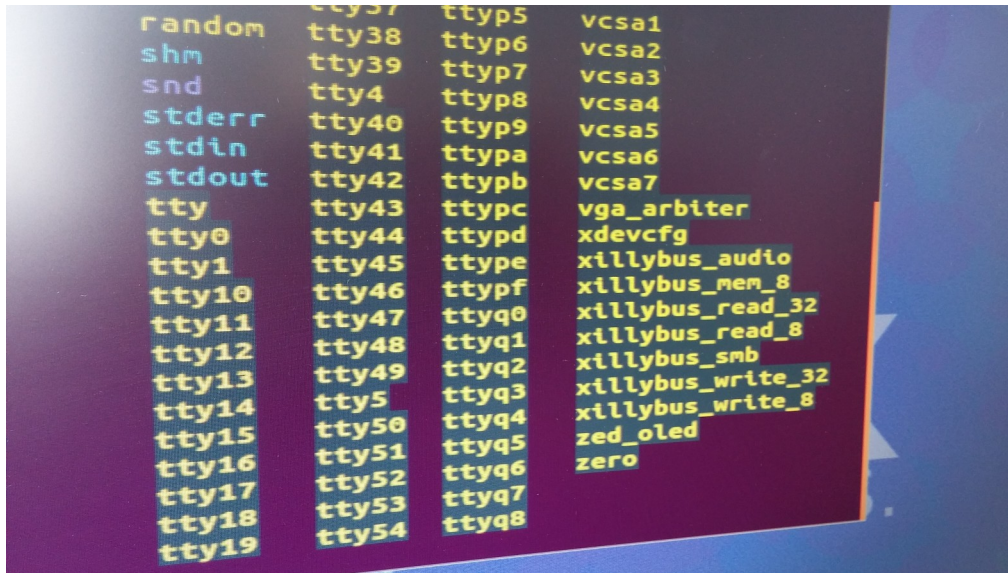


Figura 12: Fotografía de los drivers detectados por Xillinux, entre ellos los correspondientes a Xillybus.

Se corrobora el funcionamiento de “write” y “read” con los comandos en terminal “cat > /dev/xillybus_write_8” y “cat/dev/xillybus_read_8”. Del mismo modo se verifica el funcionamiento de escritura en memoria.

De una manera similar se puede crear código en C u otro lenguaje de programación capaz de abrir un archivo para escribir y leer desde y hacia la FPGA.

3.5 GNU Radio en ZedBoard.

GNU Radio es un componente fundamental para la realización de este proyecto. Una vez con el Xillinux instalado se procede a instalar el GNU Radio en dicha distribución. Se presenta la dificultad que GNU Radio no forma parte de los repositorios de Ubuntu por tanto se requiere instalar el GNU Radio mediante otro procedimiento.

Valiéndose de un sistema pre-compilado para la GNU Radio que cuenta con GNU Radio que se

encuentra la sección de “Pre-build System Files and SDK” de [29] y de la facilidad que da tener un sistema completo en una tarjeta SD, se instala una nueva distribución de Linux capaz de ejecutar correctamente GNU Radio sin la parte gráfica. Para efectos de referencia en este proyecto, a esta distribución se le nombra como “GNURadio-Zynq”

A pesar que se tiene dos distribuciones de Linux, ninguna de las dos cumple al mismo tiempo las condiciones de tener los bus de comunicación entre el PS y PL y poder ejecutar GNU Radio. Se opta en primera instancia por pasar los archivos ya compilados de GNU Radio de la distribución GNURadio-Zynq a la distribución Xillinux. Este método no resultó funcional porque GNU Radio tiene varias dependencias, sobre todo de componentes Python.

La segunda opción que se intentó fue instalar los módulos del kernel de Xillinux en la distribución GNURadio-Zynq. Con esto se logró que esta distribución tuviese el GNU Radio funcional y al mismo tiempo detectara el PL como dispositivos Xillybus, según lo explicado en la sección anterior.

Se realizó una prueba de comprobación del sistema completo en la tarjeta Zedboard con un bloque GNU Radio. La prueba consiste en enviar datos a la FPGA y que estos los devuelva de la misma manera al procesador, la finalidad de esta prueba es verificar únicamente el correcto funcionamiento de la escritura y lectura hacia y desde la FPGA, al mismo tiempo se prueba la ejecución de un bloque de GNU Radio, que es un archivo en lenguaje Python, en la distribución GNURadio-Zynq en la Zedboard.

El bloque de prueba se muestra en la Figura 13. La señal de entrada es la suma de dos señales senoidales de 200 Hz y 600 Hz respectivamente, representadas como “Signal Source”, la señal resultante se envía a la FPGA a través del driver “/dev/xillybus_write_32” y se lee al mismo tiempo la señal de salida de la FPGA mediante el driver “/dev/xillybus_read_32”. La lectura de la señal recibida se almacena en un archivo que luego se usa para comparar con la señal original en una PC.

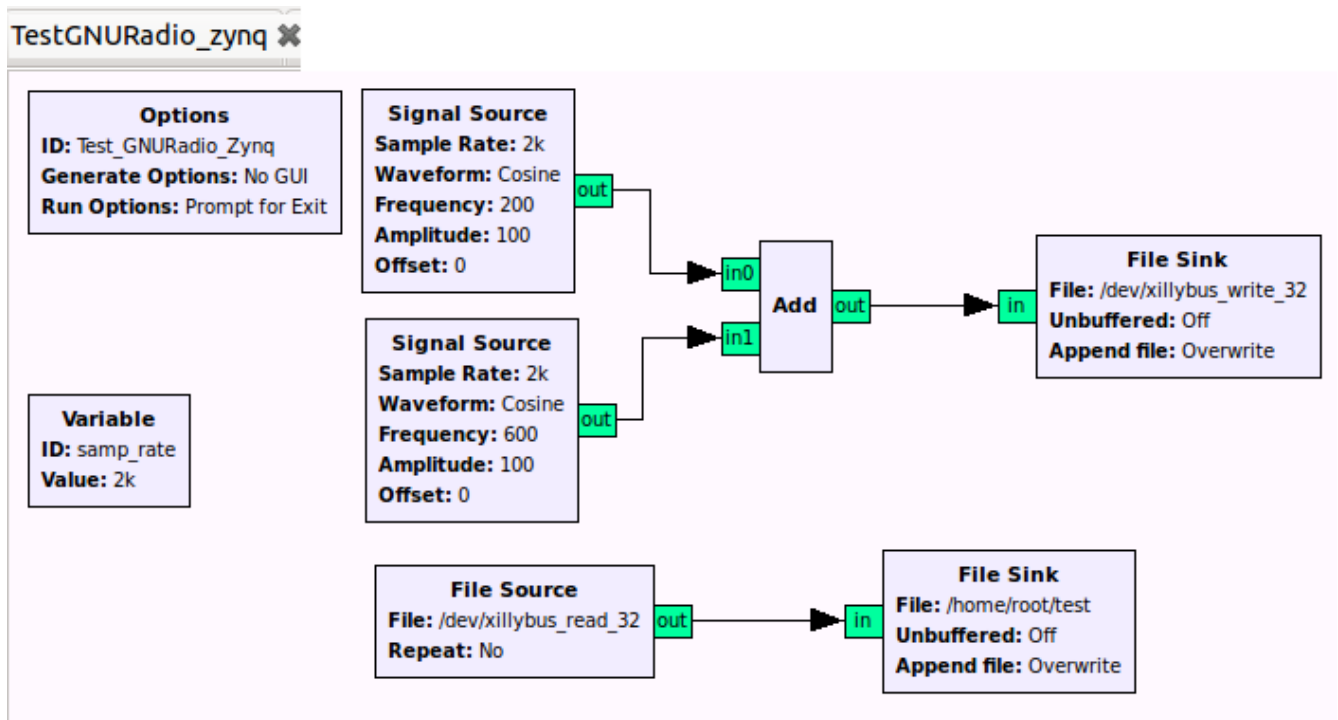


Figura 13: Bloque de GNURadio para la comprobación del funcionamiento de la comunicación entre PS y PL.

Luego de la comprobación del funcionamiento se pasa a la siguiente etapa del proyecto que es describir hardware para la FPGA con la finalidad de sustituir bloques de GNU Radio. Esto se detalla en el Capítulo 4.

Capítulo 4. Creación de IP cores propios para ser ejecutados en la FPGA.

Una vez se ha corroborado el correcto funcionamiento de la plataforma de trabajo, esto es, ejecutar un bloque de GNU Radio que sea capaz de escribir y leer datos hacia y desde el componente PS del SoC Zynq, mediante los puertos Xillybus hacia el componente PS; el siguiente paso es describir hardware que sustituya componentes de software.

Se describe en primera instancia el procedimiento genérico que se sigue para crear cualquier IP Core propio, luego se detalla el procedimiento para sintetizar un hardware que calcule una FFT (Transformada Rápida de Fourier, Fast Fourier Transform) de 4 puntos un filtro FIR (Filtro de Respuesta Finita al Impulso, Finite Impulse Response). Finalmente se corrobora el correcto funcionamiento de ambos sistemas haciendo uso programas descritos en C y Python.

4.1 IP core base de Xillybus.

En el capítulo anterior se describió el procedimiento y funcionamiento de Xilinx y los componentes Xillybus en la Zedboard. Este sistema completo se ejecuta desde una tarjeta SD, para este proyecto se utiliza una tarjeta micro SD Kingston de 8 GB clase 10 y un adaptador microSD a SD del mismo productor. A esta memoria se le crean dos particiones, una para el arranque o “boot” (estos dos términos se usan en este documento de forma indistinta) denominada como “boot” y otra partición con los archivos de sistema denominada como “rtfs” (Root File System, directorio Raíz en Linux) que contiene todos los directorios y archivos. Es de especial interés describir los archivos que componen la partición de boot, estos son:

- uImage: Binario del kernel de Linux.
- boot.bin: Archivo para el arranque inicial.
- devicetree.dtb: Archivo “Device Tree Blob”, contiene información del hardware para el kernel Linux.
- xillydemo.bit: Archivo de programación de la lógica PL o FPGA.

El último de los archivos “xillydemo.bit” es el que programa la FPGA en el arranque, por tanto, es el archivo que describe como se comporta la parte PL de la Zedboard y que puede ser remplazado para cambiar el funcionamiento del PL sin necesidad de cambiar el funcionamiento del PS.

Este archivo de configuración se genera con el software de Xilinx ISE 14.7 [28] y es uno de los resultados de un proceso en tres pasos que incluye la síntesis (Synthesize), implementación de diseño (Implement Design) y generación del archivo de configuración (Generate Programming File).

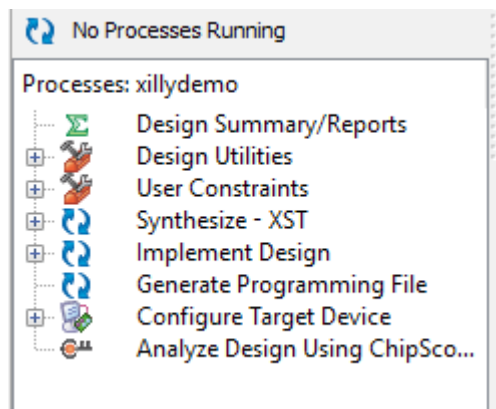


Figura 14: Generación del archivo de configuración con ISE 14.7

El sistema base a partir del cual se genera la configuración propia se basa en Xillybus y se obtiene siguiendo la guía para dispositivos con el SoC Zynq7000 de Xilinx, como la Zedboard [29]. Los archivos HDL que se utilizan en este proyecto son en el lenguaje Verilog. La Figura 15 muestra la implementación de diseño del sistema base, llamado Xillydemo.

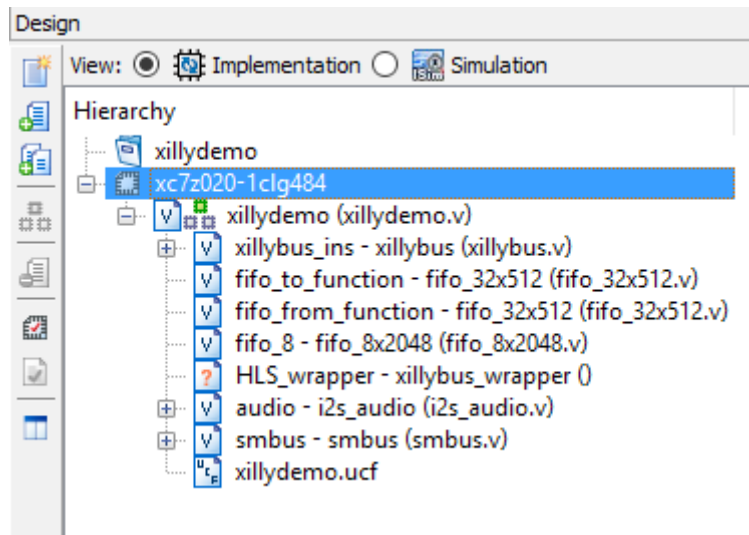


Figura 15: Implementación base de Xillybus.

Esta implementación base cuenta con el archivo “xillydemo.v” que describe el componente principal “xillydemo”, en este componente se instancia los componentes:

- xillybus
- fifo_to_function
- fifo_from_function
- fifo_8
- audio
- smbus
- HLS_wrapper

La mayoría de estos bloques se describen la sección 3.3 con la excepción de “HLS_wrapper”. Este aparece con un signo de pregunta (?) porque el sistema base no lo contiene y es este el archivo que se debe crear con la funcionalidad propia, para este proyecto ese archivo describirá en el primer caso un FFT de 4 puntos y en el segundo caso un Filtro FIR. Esto se describe en las siguientes secciones.

4.2 Hardware para FFT de 4 puntos.

La FFT es un algoritmo para el cálculo de la DFT (Transformada Discreta de Fourier, Discrete Fourier Transform) de una secuencia de datos. La FFT convierte la señal de entrada del dominio del tiempo (en este caso) al dominio de frecuencia, por tanto es un cálculo fundamental y de uso frecuente en sistemas de comunicación digital [30].

El cálculo de esta transformada se hace mediante la ecuación 1:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j \frac{2\pi k}{N} n}, k=0, 1, 2, \dots, N-1 \quad (1)$$

Para el caso de cuatro puntos, esto es, cuatro datos de entrada, la ecuación (1) se representa en la ecuación (2)

$$X(k) = \sum_{n=0}^3 x[n] e^{-j \frac{\pi k}{2} n}, k=0, 1, 2, 3 \quad (2)$$

Es de especial interés el término exponencial, que se expresa también como:

$$e^{-j\frac{\pi k}{2}n} = \cos\left(\frac{\pi k}{2}n\right) - j\sin\left(\frac{\pi k}{2}n\right) \quad (3)$$

Por lo que se determina los cuatro valores de salida de la FFT como:

$$\begin{aligned} X[0] &= x[0] + x[1] + x[2] + x[3] \\ X[1] &= x[0] - jx[1] - x[2] + jx[3] \\ X[2] &= x[0] - x[1] + x[2] - x[3] \\ X[3] &= x[0] + jx[1] - x[2] - jx[3] \end{aligned} \quad (4)$$

Con el software Vivado High Level System de Xilinx 2015.2 [31] conocido como HLS y usando como referencia la guía de co-procesamiento de Xilinx [32] se describe en lenguaje C el funcionamiento de la FFT de cuatro puntos, el código completo se detalla en el Apéndice A de este documento.

El HLS da como resultado de salida el componente “HLS_wrapper” introducido en la sección anterior.

Este componente se agrega al proyecto de xillydemo.v

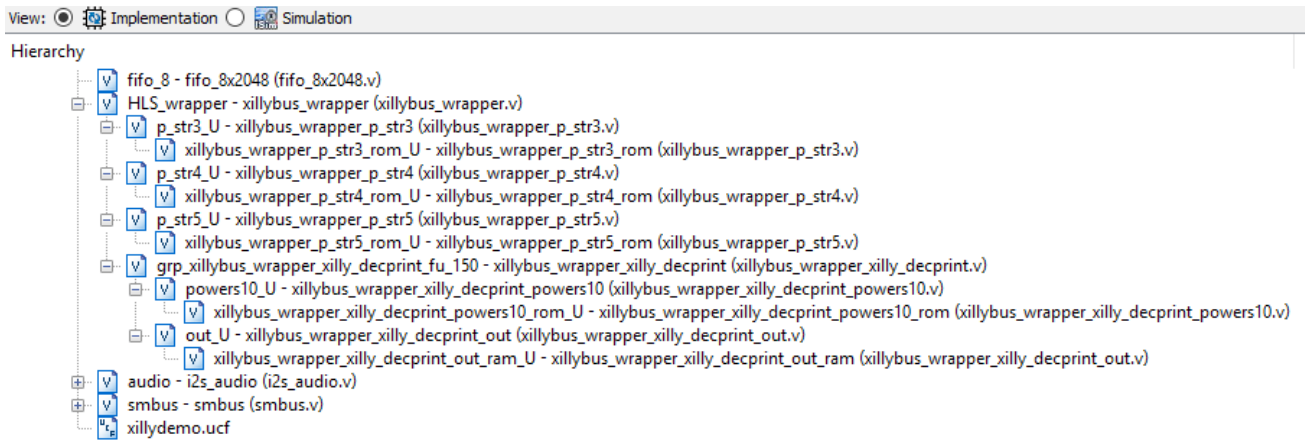


Figura 16: Implementación de xillydemo con el componente HLS_wrapper que describe la FFT de 4 puntos.

Como paso final de este proceso se genera el bitstream que da como resultado el archivo xillydemo.bit que se copia a la partición “boot” de la tarjeta SD para la configuración del componente PL de la Zedboard.

4.3 Hardware para filtro FIR.

Un filtro FIR es siempre un candidato idóneo para ser implementado en una FPGA dado que por su estructura se puede describir un hardware altamente paralelizable que lo implemente correctamente [33].

La Figura 17 muestra la estructura básica, en este, la salida actual depende de la entrada actual y de hasta N entradas anteriores respondiendo a la siguiente ecuación:

$$y[n] = a_0 * x[n] + a_1 * x[n-1] + a_2 * x[n-2] + \dots + a_N * x[n-N] \quad (6)$$

Donde $y[n]$ representa la salida actual y los términos x representan la entrada actual y hasta N entradas anteriores, N representa el orden del filtro y los términos de ganancia a se les denomina como “taps”.

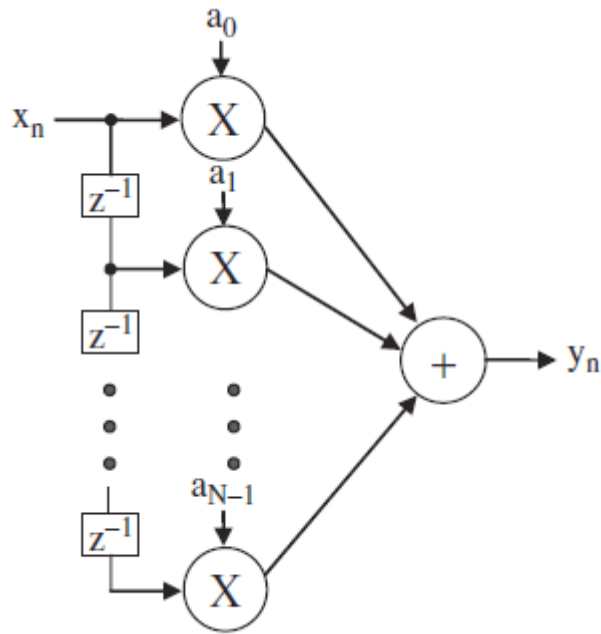


Figura 17: Estructura del filtro FIR a ser implementado [33].

Mediante la herramienta web [34] para el diseño de filtros, se obtiene los “taps” para un filtro paso-bajo con frecuencia de corte en 400 Hz y banda de rechazo para frecuencias mayores a 600 Hz. La respuesta en frecuencia del filtro deseado es:

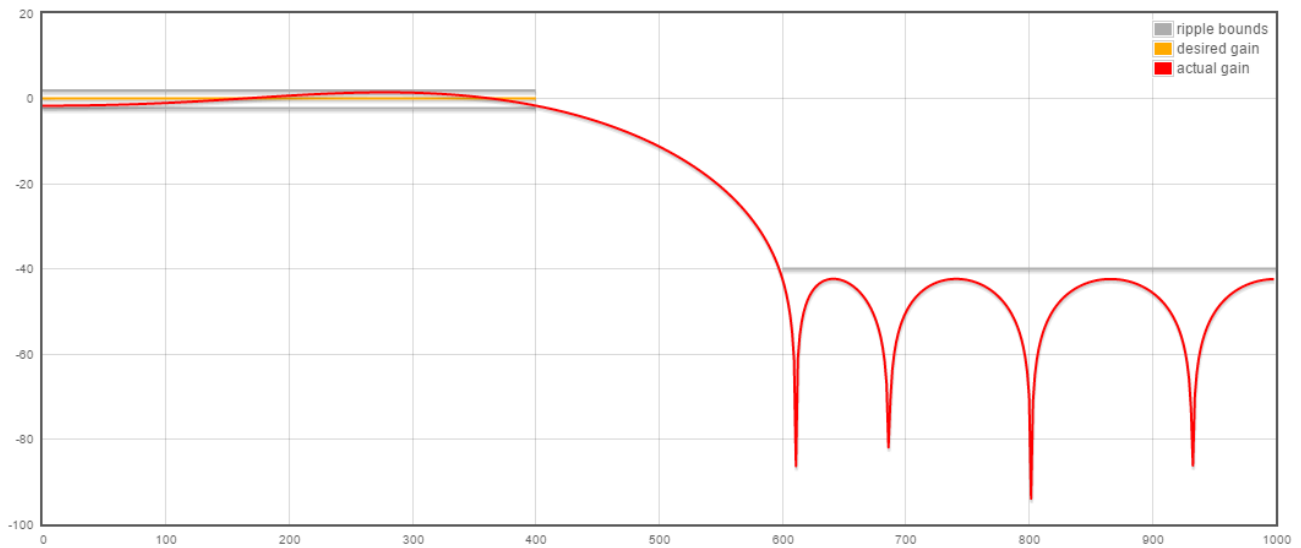


Figura 18: Respuesta en frecuencia del filtro deseado

Los detalles de diseño son:

from	to	gain	ripple/att.	act. rpl
0 Hz	400 Hz	1	5 dB	3.14 dB
600 Hz	1000 Hz	0	-40 dB	-42.42 dB

add passband

add stopband

!predefined

sampling freq.

2000 Hz

desired #taps

minimum

actual #taps

13

DESIGN FILTER

Figura 19: Detalles de diseño del filtro deseado.

Y los “taps” que se utiliza para la implementación en hardware son:

plain text int

fixed point precision (bits): 8

5
4
-15
-30
4
78
119
78
4
-30
-15
4
5

Figura 20: "taps" del filtro FIR a implementar

Del mismo modo que para la FFT, se utiliza la herramienta Vivado HLS 2015.2 para describir en lenguaje C el comportamiento del filtro FIR, dicho código se adjunta en el Apéndice A de este documento.

El resultado del paso anterior da la descripción del hardware denominado como “HLS_Wrapper” y este a su vez instancia otros componentes. Se agrega este hardware al proyecto de ISE de xillydemo.v:

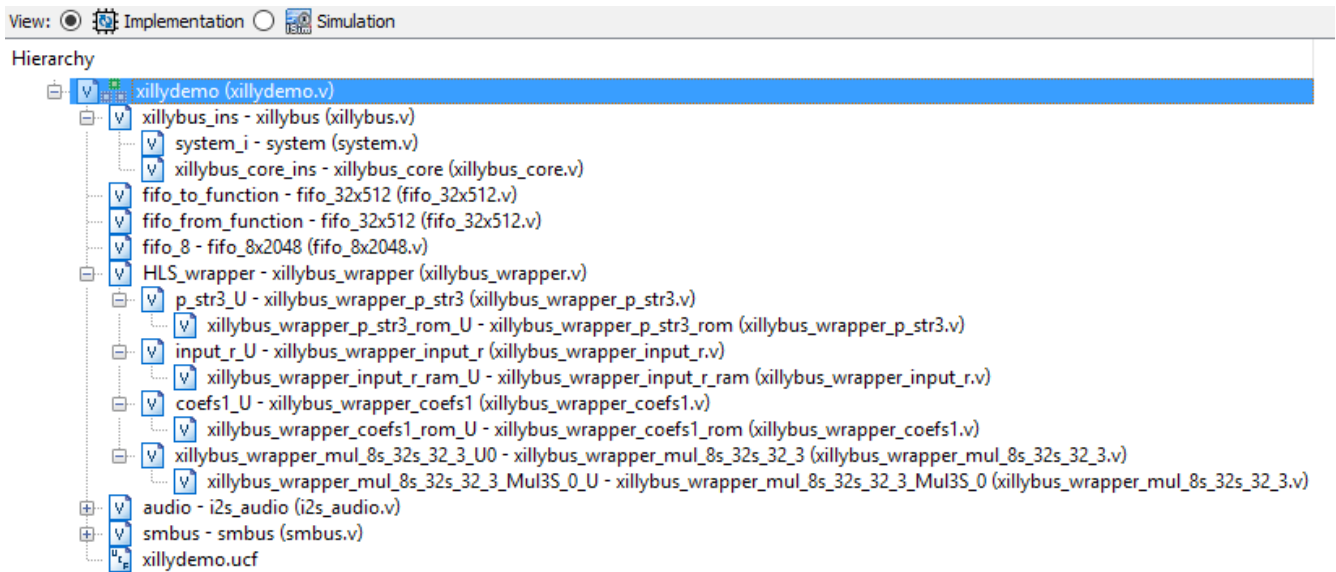


Figura 21: Implementación del filtro FIR en el proyecto xillydemo.v

Del mismo modo que en la FFT, se genera el “bitstream” que da como resultado el archivo xillydemo.bit que se copia a la partición “boot” de la tarjeta SD que configura la FPGA de la Zedboard.

Las pruebas de funcionamiento de ambos hardware, FFT y FIR, se muestran en la siguiente sección.

4.4 Pruebas de verificación para el hardware FFT y FIR.

En esta etapa del proyecto se verifica la respuesta de los bloques de hardware para la FFT y el FIR. En una primera etapa se realiza con código en C y luego con la ejecución de un bloque de GNU Radio, siendo este último un ejecutable de Python.

En el Apéndice A de este proyecto se detalla el código en lenguaje C que se ejecuta en la parte PS de la Zedboard para ambos casos, la FFT y el FIR. El código se basa en el tutorial de co-procesamiento de Xillybus detallado en [32]. Entre los detalles que son importantes de remarcar se encuentra la definición de los datos de entrada y salida de la FPGA, estos se hace mediante estructuras de C “struct”

hacia la FPGA “tologic” y desde la FPGA “fromlogic” para el caso de la FFT, ya que se le envía cuatro datos enteros “int” que son datos reales y se obtiene ocho datos de salida que corresponden a las partes real e imaginaria de cuatro datos de salida. Para el caso del filtro FIR los datos “int” se pueden enviar uno a uno, de la misma forma que se hace la recepción.

Ambos códigos cuentan con abrir los archivos correspondientes drivers de Xillybus, enviar y recibir los datos a través de ellos y luego cerrarlos, por ejemplo:

```
...  
fdr = open("/dev/xillybus_read_32", O_RDONLY);  
fdw = open("/dev/xillybus_write_32", O_WRONLY);  
...  
write(fdw, (void *) &tologic, sizeof(tologic));  
read(fdr, (void *) &fromlogic, sizeof(fromlogic));  
...  
close(fdr);  
close(fdw);  
...
```

Se hace varias pruebas básicas de funcionamiento y se corrobora los resultados esperados, por ejemplo

FFT:

entrada (real): 178, 104, 31, 4

Salida (real, imaginario): (317, 0), (147, -100), (101, 0), (147,
100) .

FIR:

```
input[32]={0, 1, 2, 3, 3, 3, 3, 2, 1, 0, 0, 0, 1, 2, 3, 3, 3, 2, 1, 0,  
0};
```

```
output[32]={0, 5, 14, 8, -33, -74, -22, 174, 440, 647, 740, 740, 652,
454, 182, -55, -48, -60, 173, 460, 693};
```

Se hizo pruebas como la anterior de manera extensa, se utilizó un archivo de entrada con una señal tipo “diente de sierra”, los resultados de ambos bloque se escribieron en un archivo de texto de salida y se compararon con los resultados esperados, al tener los mismos resultados se concluye que los bloques funcionaron de manera correcta para las pruebas hechas.

Para el filtro FIR también se crea el siguiente bloque en GNU Radio:

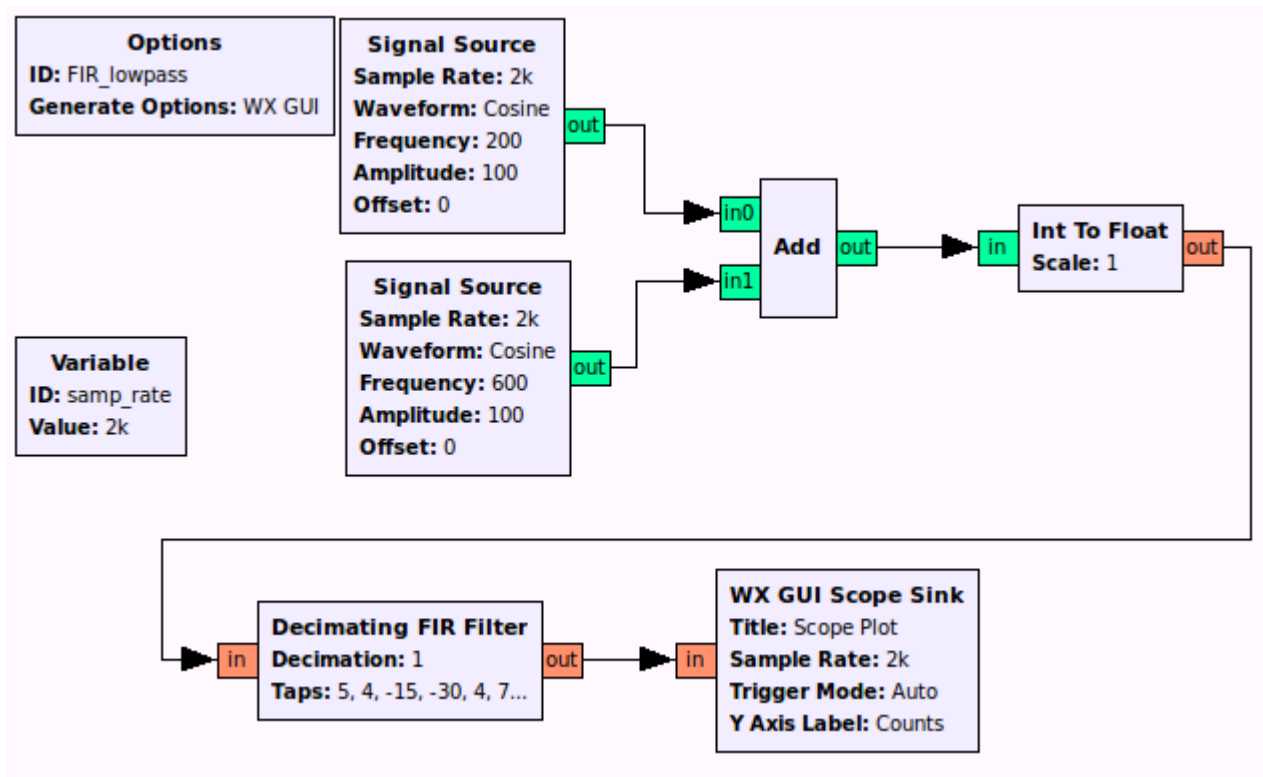


Figura 22: Bloque GNU Radio del filtro FIR

En este, se tiene como señal de entrada la suma de dos funciones senoidales de amplitud 100 y frecuencia de 200 y 600 Hz respectivamente, luego estos datos se pasan a un filtro FIR proveído por el

GNU Radio, como este filtro solo puede tener valores de entrada reales se hace una conversión previa de entero a flotantes con el bloque “Int To Float” finalmente se pasan a un visualizador llamado “WX GUI Scope Sink”. El resultado se muestra en la Figura 23:

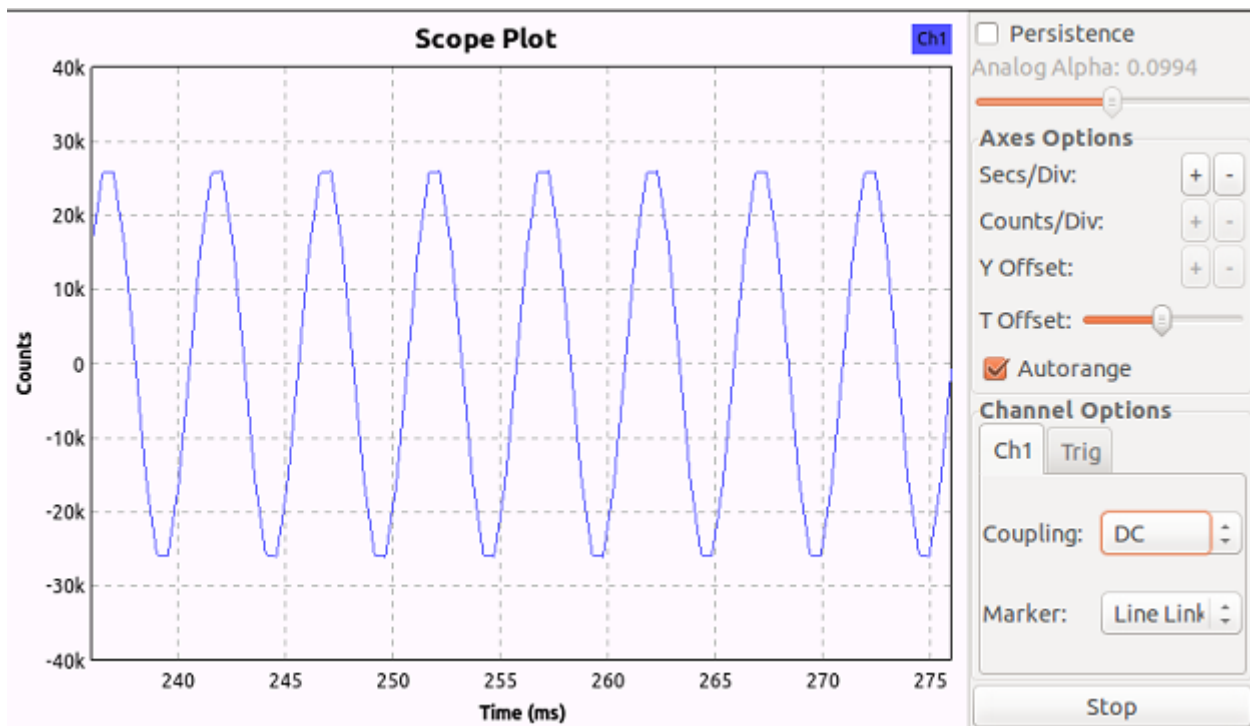


Figura 23: Salida del filtro FIR en GNU Radio.

Como era de esperar, la señal de salida es una señal senoidal con una frecuencia de 200 Hz.

Del mismo modo se crea un bloque para filtrar con el FIR una señal de entrada que sea la suma de dos senoidales, pero esta vez con frecuencias de 200 y 250 Hz respectivamente, dando como resultado la señal de la Figura 24:

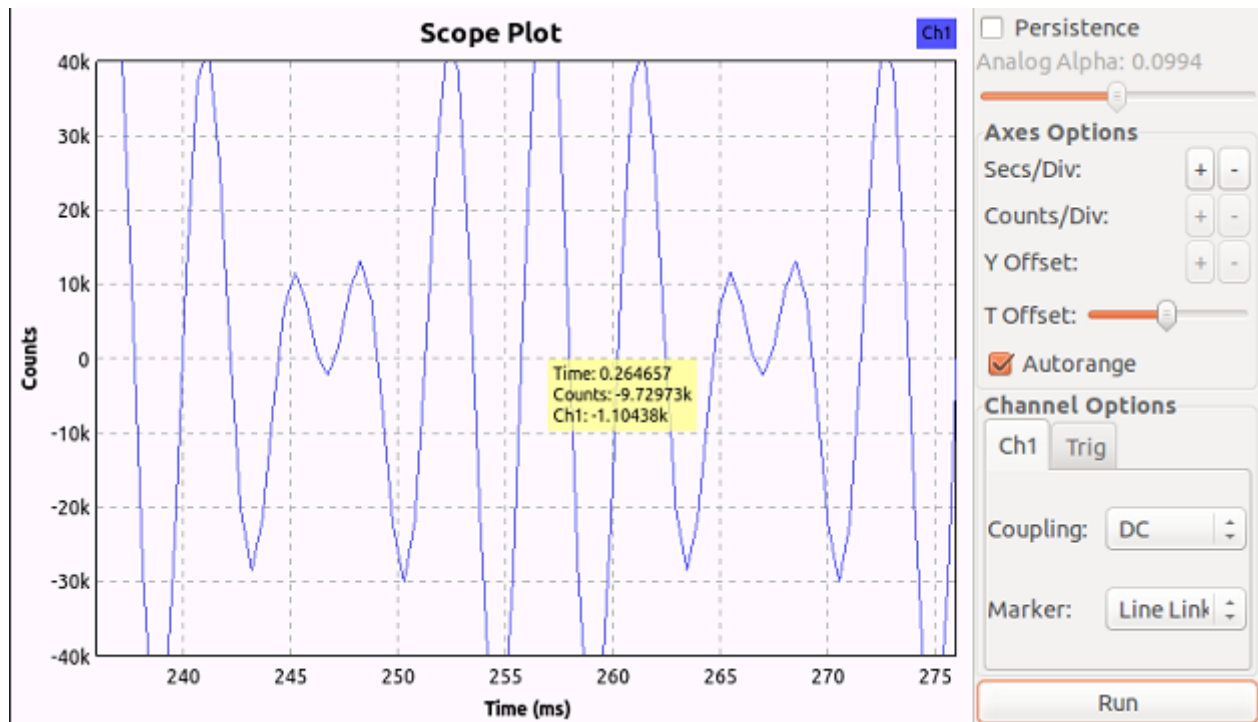


Figura 24: Salida del FIR de la segunda señal.

Se realiza las mismas pruebas anteriores pero en la Zedboard, esto es, ejecutar un bloque de GNU Radio que genere las señales senoidales en el componente PS y pase los datos al componente PL para que este haga el filtrado de la señal y se lea de vuelta los mismos. Como no se tiene parte gráfica en la distribución Linux que se ejecuta en la Zedboard, la salida de la FPGA se guarda en un archivo para luego graficarlo en una PC. El bloque GNU Radio que se ejecuta en la Zedboard es el siguiente:

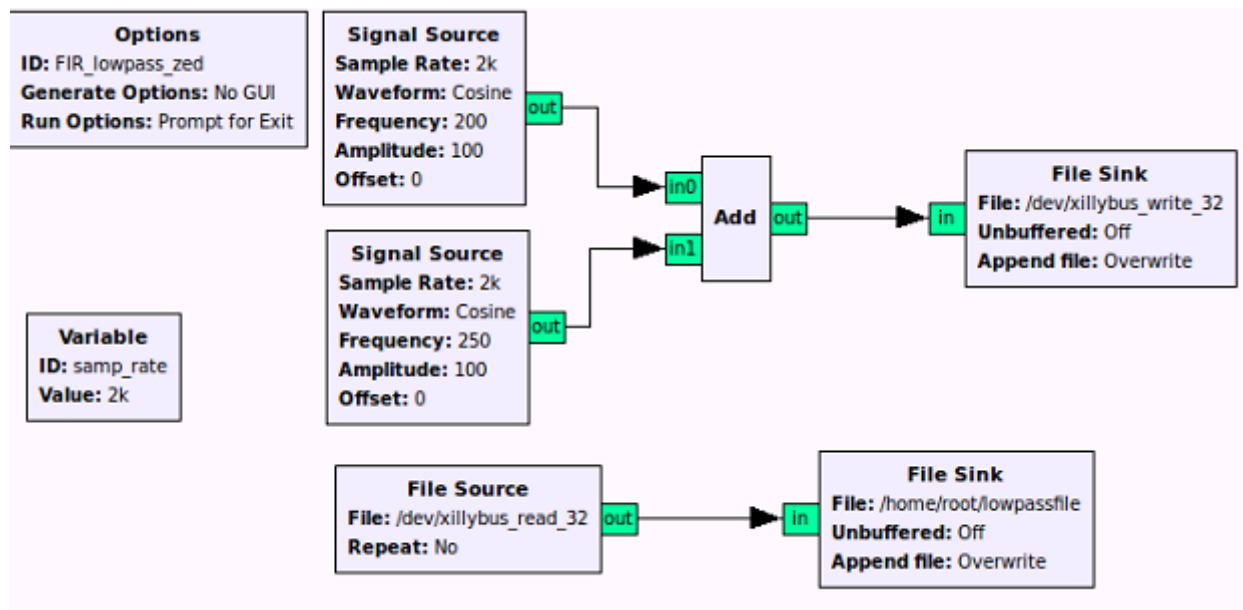


Figura 25: Bloque GNU Radio para ejecutar en la Zedboard.

Como se detalla en el bloque anterior, los datos se envían a “/dev/xillybus_write_32” y se leen de “/dev/xillybus_read_32”, el resultado se guarda en un archivo. El bloque GNU Radio al ser compilado genera un archivo de Python llamado “FIR_lowpass_zed.py”, cuyo código se muestra en el Apéndice A.

Luego de ejecutar el archivo que envía la señal a la FPGA y que recibe de vuelta el resultado de la misma, la información se guarda en un archivo con los resultados. Luego con este archivo, se crea un bloque de GNU Radio para graficar la información, para verificar el funcionamiento del FIR descrito para la FPGA en la Zedboard. El bloque usado para graficar el archivo de datos de salida de la FPGA es:

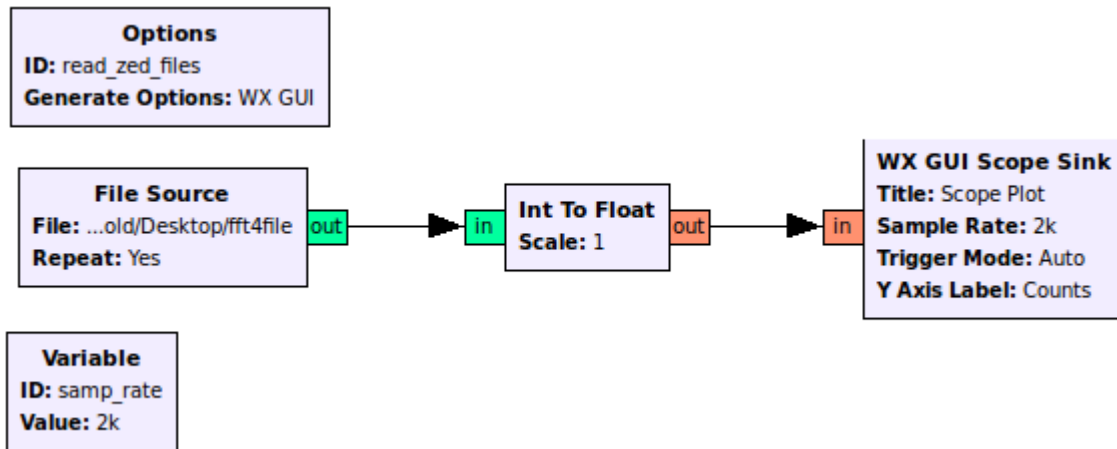


Figura 26: Bloque GNU Radio para leer datos de salida de la FPGA.

Los resultados para las dos señales mostradas previamente (primera suma de senoidales de 200 y 600 Hz y suma de senoidales de 200 y 250 Hz) se muestran en las Figuras 27 y 28.

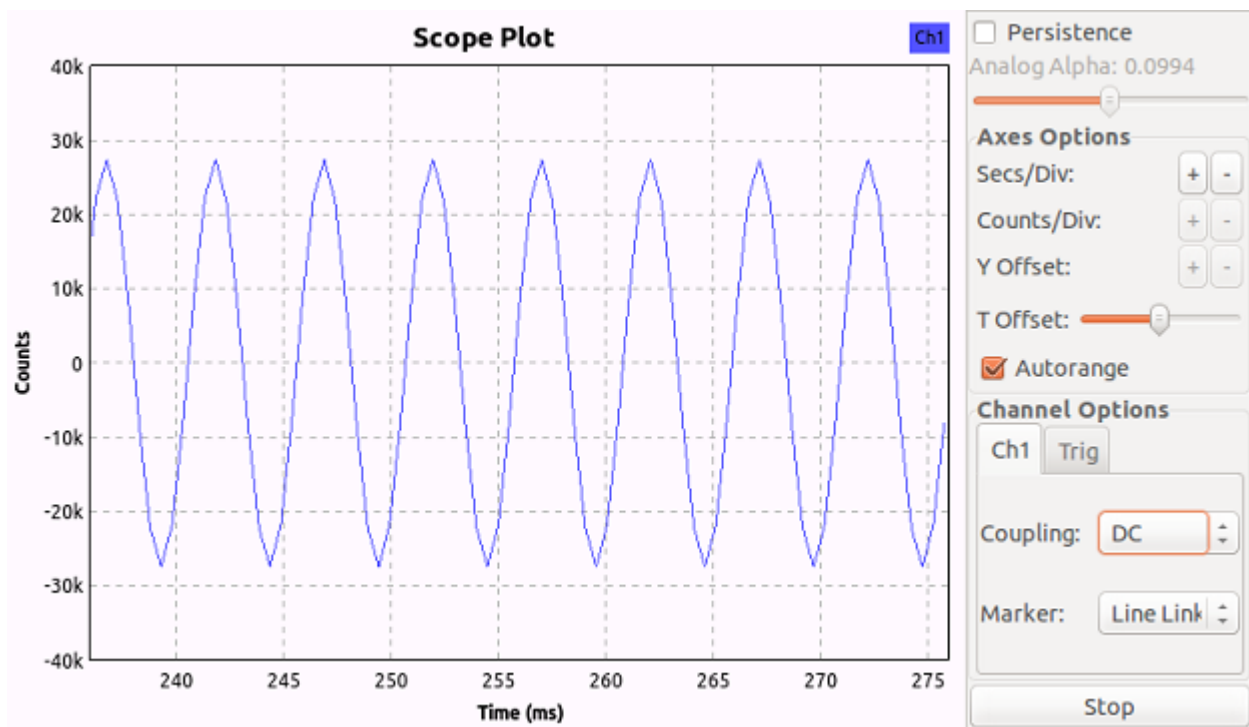


Figura 27: Salida de FIR en FPGA de Zedboard. Entrada suma de senoidales de 200 y 600 Hz respectivamente.



Figura 28: Salida de FIR en FPGA de Zedboard. Entrada suma de senoidales de 200 y 250 Hz respectivamente.

Comparando las Figuras 23 y 24 con las Figuras 27 y 28 se puede ver como el comportamiento de ambas es muy similar, lo que da pistas del correcto funcionamiento del filtro FIR en el componente PL de la Zedboard cuando se ejecuta un código de GNU Radio en el PS.

Como prueba de datos uno a uno, similar a lo hecho con el código C, se usó el software Octave [35] y los “scripts” de GNU Radio para leer los archivos de salida con dicho software [36]. Se hace una verificación de los datos uno a uno con los valores esperados y se obtiene que son los mismos.

Con lo anterior se da por satisfecho que ambos bloques FFT de cuatro puntos y filtro FIR fueron descritos satisfactoriamente para la FPGA de la Zedboard y se corroboró el correcto funcionamiento de la misma junto con una perfecta integración con GNU Radio ejecutado en el componente PS de dicha

tarjeta. El siguiente capítulo muestra pruebas de rendimiento del sistema híbrido (PS y PL).

Capítulo 5. Pruebas de rendimiento.

Se presenta en esta sección los resultados de las pruebas de rendimiento que se realizó sobre los bloques FIR y FFT en hardware y se compara con el rendimiento de estos mismos bloques cuando son ejecutados únicamente con software. Las pruebas consistieron en medir el rendimiento usando herramienta de Linux “top”[37], esta herramienta resume, entre otras cosas, el uso de memoria y CPU.

5.1 Herramienta de Linux “top”.

Esta herramienta lista los 30 procesos del sistema operativo de mayor uso y la actualiza periódicamente. Para cada uno de los procesos mostrados se indica el identificador asignado por el S.O., usuario al que pertenece, el estado (“Stat” de state), la memoria usada (“VSZ” de “Virtual memory SiZe” o tamaño de la memoria virtual), porcentaje de la memoria total utilizada (“%VSZ”), porcentaje de uso de CPU (“%CPU”) y el comando que ejecutad el proceso (“Command”).

```

Mem: 101376K used, 410356K free, 0K shrd, 2768K buff, 13044K cached
CPU:  0% usr  0% sys  0% nic 99% idle  0% io  0% irq  0% sirq
Load average: 0.01 0.02 0.01 2/63 1028

```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
1028	1026	root	R	2532	0%	0%	top
397	2	root	SW	0	0%	0%	[kworker/0:1]
1024	965	root	S	4872	1%	0%	sshd: root@pts/0
965	1	root	S	4588	1%	0%	/usr/sbin/sshd
993	1	avahi	S	3412	1%	0%	avahi-daemon: running [zedboard-zynq7.
980	1	ntp	S	3308	1%	0%	/usr/sbin/ntpd -u ntp:ntp -p /var/run/
994	993	avahi	S	3300	1%	0%	avahi-daemon: chroot helper
974	1	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
975	974	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1010	974	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1011	974	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1012	974	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1009	1	root	S	3132	1%	0%	-sh
1026	1024	root	S	3108	1%	0%	-sh
747	1	root	S	2940	1%	0%	/lib/udev/udev -d
940	747	root	S	2936	1%	0%	/lib/udev/udev -d
1013	747	root	S	2936	1%	0%	/lib/udev/udev -d
956	1	messageb	S	2808	1%	0%	/usr/bin/dbus-daemon --system
969	1	rpc	S	2404	0%	0%	/usr/sbin/rpcbind
1008	1	root	S	2208	0%	0%	/sbin/getty 115200 ttyPS0

Figura 29: Toma de pantalla del comando top de la Zedboard con el S.O. GNURadio-Zynq.

Para efectos de este proyecto, según los objetivos marcados, se puso especial énfasis en el porcentaje de la memoria usada “%VSZ” y uso de CPU “%CPU”. Para la tarjeta Zedboard la %VSZ se basa en los 512 MB de memoria disponible y %CPU corresponde al porcentaje de tiempo del total que el S.O. asigna al proceso en cuestión.

5.2 Rendimiento del bloque filtro FIR.

Para medir el rendimiento de un filtro FIR se utilizó además del filtro paso bajo de 13 “taps” mostrado en el capítulo anterior otros dos filtros con 5 y 25 “taps”, el primero de ellos es un filtro paso bajo con frecuencia de corte en 400 Hz y banda de rechazo para frecuencias superiores a los 1500 Hz; el segundo de ellos es un filtro paso bajo de 400 Hz y banda de rechazo para frecuencias superiores a 600

Hz. En ambos casos se aumentó la frecuencia de muestreo a los 4000 Hz en el bloque de GNU Radio. Para referencia los filtros se van a nombrar como “FIR5taps”, “FIR13taps” y “FIR25taps”.

Los tres filtros FIR fueron ejecutados puramente por software con el código Python resultante de GNU Radio como el mostrado en la Figura 22 del capítulo anterior, al tiempo que se ejecutaba “top” en una consola Linux. Para cada filtro se hizo 120 mediciones del resultado de “top”. Los resultados obtenidos se muestran en la siguiente tabla.

# “taps”	Media (%)	σ (%)
5	84.52	6.06
13	77.73	9.50
25	66.92	4.85

Tabla 1: Rendimiento %CPU de filtros FIR ejecutados únicamente en zona PS.

En todos los casos de la tabla anterior se obtuvo un %VSZ de 25%.

Luego se realizó las mismas pruebas pero usando los el bloque GNU Radio que hace uso de la FPGA para sustituir el bloque correspondiente al filtro FIR como se detalló en la Figura 25.

# “taps”	Media (%)	σ (%)
5	15.27	0.58
13	8.68	0.58
25	4.87	0.55

Tabla 2: Rendimiento %CPU de filtros FIR ejecutados en ambas zonas PS y PL.

En todos los casos de la ejecución híbrida PS y PL se obtuvo un uso de memoria %VSZ de 24%

Las Figuras 30 y 31 muestran una toma de pantalla de top para el filtro FIR de 13 “taps”, el primer caso para la ejecución en PS y el segundo para el sistema híbrido.

```

Mem: 172208K used, 339524K free, 0K shrd, 4632K buff, 53740K cached
CPU:  83% usr   2% sys   0% nic   5% idle   8% io   0% irq   0% sirq
Load average: 1.09 0.35 0.13 5/74 1066

```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
1059	1041	root	S	126m	25%	85%	python ./FIR13taps.py
1049	1048	root	R	2532	0%	0%	top
674	2	root	SW	0	0%	0%	[mmcqd/0]
397	2	root	SW	0	0%	0%	[kworker/0:1]
696	2	root	SW<	0	0%	0%	[kworker/0:1H]
750	2	root	SW	0	0%	0%	[usb-storage]
1046	967	root	S	4872	1%	0%	sshd: root@pts/1
1039	967	root	S	4872	1%	0%	sshd: root@pts/0
967	1	root	S	4588	1%	0%	/usr/sbin/sshd
995	1	avahi	S	3412	1%	0%	avahi-daemon: running [zedboard-zynq7.
982	1	ntp	S	3308	1%	0%	/usr/sbin/ntpd -u ntp:ntp -p /var/run/
996	995	avahi	S	3300	1%	0%	avahi-daemon: chroot helper
976	1	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
977	976	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1012	976	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1013	976	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1014	976	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1011	1	root	S	3132	1%	0%	-sh
1041	1039	root	S	3108	1%	0%	-sh
1048	1046	root	S	3108	1%	0%	-sh

Figura 30: Toma pantalla de top para filtro FIR de 13 “taps” ejecutado en PS únicamente.

```
Mem: 171288K used, 340444K free, 0K shrd, 4544K buff, 52548K cached
CPU:  7% usr  2% sys  0% nic 90% idle  0% io  0% irq  0% sirq
Load average: 0.16 0.09 0.04 2/76 1039
```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
1032	1019	root	S	118m	24%	10%	python ./FIR13taps_zed.py
1023	968	root	S	4872	1%	0%	sshd: root@pts/1
674	2	root	SW	0	0%	0%	[mmcqd/0]
397	2	root	SW	0	0%	0%	[kworker/0:1]
1017	968	root	S	4872	1%	0%	sshd: root@pts/0
968	1	root	S	4588	1%	0%	/usr/sbin/sshd
996	1	avahi	S	3408	1%	0%	avahi-daemon: registering [zedboard-zy
983	1	ntp	S	3308	1%	0%	/usr/sbin/ntpd -u ntp:ntp -p /var/run/
997	996	avahi	S	3300	1%	0%	avahi-daemon: chroot helper
977	1	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
978	977	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1013	977	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1014	977	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1015	977	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1012	1	root	S	3132	1%	0%	-sh
1019	1017	root	S	3112	1%	0%	-sh
1025	1023	root	S	3112	1%	0%	-sh
747	1	root	S	2940	1%	0%	/lib/udev/udev -d
841	747	root	S	2936	1%	0%	/lib/udev/udev -d
842	747	root	S	2900	1%	0%	/lib/udev/udev -d

Figura 31: Toma pantalla de top para filtro FIR de 13 “taps” ejecutado en PS y PL.

5.3 Rendimiento del bloque FFT.

Del mismo modo que el caso del FIR se ejecuta los programa Python generado por GNU Radio para una transformada FFT de cuatro puntos, en un caso se usa el cálculo únicamente por software y el segundo caso haciendo uso del bloque FFT en la FPGA. Se usa “top” para llevar registro del uso de memoria y CPU y se ejecuta cada uno de los código durante un tiempo mientras se hacen 5 tomas de pantalla espaciadas aproximadamente por 5 segundos.

Los resultados se muestran en la siguiente tabla:

Componente	Media (%)	σ (%)
PS	96.00	0.82
PS y PL	32.25	2.45

Tabla 3: Rendimiento %CPU de transformada FFT de cuatro puntos en PS y en PS y PL.

Para el caso de la FFT calculada en PS se tuvo en todo momento un uso de memoria %VSZ de 29% y para el caso híbrido PS y PL ese uso fue siempre de 24%.

```
Mem: 173572K used, 338160K free, 0K shrd, 4760K buff, 54528K cached
CPU: 83% usr 14% sys 0% nic 2% idle 0% io 0% irq 0% irq
Load average: 1.42 0.38 0.14 3/76 1080
```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
1071	1029	root	S	142m	29%	97%	python ./FFT4.py
1030	974	root	S	4872	1%	0%	sshd: root@pts/1
989	1	ntp	S	3308	1%	0%	/usr/sbin/ntpd -u ntp:ntp -p /var/run/
1064	753	root	S	2900	1%	0%	/lib/udev/udev -d
674	2	root	SW	0	0%	0%	[mmcqd/0]
852	2	root	SW	0	0%	0%	[kworker/1:2]
16	2	root	SW	0	0%	0%	[kdevtmpfs]
1027	974	root	S	4872	1%	0%	sshd: root@pts/0
974	1	root	S	4588	1%	0%	/usr/sbin/sshd
1002	1	avahi	S	3404	1%	0%	avahi-daemon: registering [zedboard-zy
1003	1002	avahi	S	3300	1%	0%	avahi-daemon: chroot helper
983	1	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
984	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1019	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1020	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1021	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1018	1	root	S	3132	1%	0%	-sh
1029	1027	root	S	3108	1%	0%	-sh
1032	1030	root	S	3108	1%	0%	-sh
753	1	root	S	2904	1%	0%	/lib/udev/udev -d

Figura 32: Toma pantalla de "top" para ejecución en PS de FFT.

```

Mem: 173672K used, 338060K free, 0K shrd, 4792K buff, 55032K cached
CPU:  23% usr   8% sys   0% nic  68% idle   0% io   0% irq   0% sirq
Load average: 1.45 0.50 0.19 4/74 1088

```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
1081	1029	root	S	118m	24%	31%	python ./FFT4_zed.py
1030	974	root	S	4872	1%	0%	sshd: root@pts/1
1033	1032	root	R	2532	0%	0%	top
845	2	root	SW	0	0%	0%	[kworker/0:2]
1027	974	root	S	4872	1%	0%	sshd: root@pts/0
974	1	root	S	4588	1%	0%	/usr/sbin/sshd
1002	1	avahi	S	3404	1%	0%	avahi-daemon: registering [zedboard-zy
989	1	ntp	S	3308	1%	0%	/usr/sbin/ntpd -u ntp:ntp -p /var/run/
1003	1002	avahi	S	3300	1%	0%	avahi-daemon: chroot helper
983	1	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
984	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1019	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1020	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1021	983	distcc	S N	3144	1%	0%	/usr/bin/distccd --pid-file=/var/run/d
1018	1	root	S	3132	1%	0%	-sh
1029	1027	root	S	3108	1%	0%	-sh
1032	1030	root	S	3108	1%	0%	-sh
753	1	root	S	2904	1%	0%	/lib/udev/udev -d
1064	753	root	S	2900	1%	0%	/lib/udev/udev -d
1065	753	root	S	2900	1%	0%	/lib/udev/udev -d

Figura 33: Toma pantalla de "top" para ejecución híbrida de FFT.

Capítulo 6. Análisis de resultados, conclusiones y recomendaciones.

Se hace en esta sección un análisis de los resultados mostrados en la última sección del Capítulo 4 y el Capítulo 5, en función a los objetivos marcados en el Capítulo 2 se llega a conclusiones y finalmente se dan recomendaciones para proyectos con una temática similar a la tratada en este proyecto.

6.1 Análisis de resultados.

En este proyecto se buscó implementar en hardware de FPGA los bloques de FFT de cuatro puntos y un filtro FIR de varios “taps”. Como en cualquier implementación ya sea de software o de hardware lo primero que se debe asegurar es obtener resultados correctos. Para el caso de las dos implementaciones existe ecuaciones matemáticas claramente definidas de como deben comportarse dichas implementaciones ante determinadas entradas. Se corroboró el correcto funcionamiento de ambas implementaciones de hardware de varias maneras, a saber:

- Se escribe código en C que ejecuta ambos bloques en software. Los bloques de hardware FPGA se estimulan con datos de entrada al igual que los códigos en C; se recolectan los datos de salida tanto de software como de hardware y se corrobora que ambas salidas son iguales, concluyendo que los bloques de hardware funcionan correctamente.
- Se implementa un bloque de GNU Radio para un filtro FIR paso bajo, se obtiene los datos de salida. Luego se implementa en GNU Radio un sistema que sustituye el FIR de software por el FIR de hardware en la FPGA. Se hace gráficas para comparar ambos comportamientos con dos

señales diferentes de entrada. Se corrobora que ambas salidas son iguales con lo que se concluye que el FIR funciona correctamente.

- Similar a la comprobación anterior se guardó los datos de ambos sistemas en un archivo y haciendo uso del software Octave y “scripts” de GNU Radio para extraer estos datos a un arreglo, se compara uno a uno los datos de salida de ambos sistemas. Estos resultan iguales y se concluye que ambos sistemas funcionan de la misma manera.

Habiendo comprobado que los bloques de hardware funcionan de forma correcta se hacen pruebas de rendimiento computacional de ambos bloques, los resultados fueron mostrados en el Capítulo 5. Se ejecutó 3 filtros FIR con diferente cantidad de “taps” (5, 13 y 25), la Tabla 1 muestra los resultados de uso de CPU según lo muestra el comando “top” de Linux cuando se ejecuta los filtros FIR enteramente en PS.

Los resultados varían de muestra a muestra, esto se debe a que el sistema operativo gestiona los recursos computacionales con todos los proceso activos y en este caso esta gestión no puede ser controlada por el usuario. Se tomó un total de 120 medidas del uso de CPU para cada uno de los filtros, obteniendo para el filtro de 5 “taps” el uso de CPU en 84.52% de valor medio con una desviación estándar de 6.02%. Para el filtro FIR de 13 “taps” los datos resultaron ser 77.73% con una desviación estándar de 9.50%. Finalmente el filtro FIR de 25 “taps” mostró una media de 66.92% de uso de CPU con una desviación estándar de 4.85%

Los resultados anteriores se comparan con los mismos filtros pero ejecutados con un bloque híbrido que hace uso de ambos sistemas PS y PL. Para el primero de los filtros los porcentajes bajaron a

15.26% de media con desviación estándar de 0.58%, el FIR de 13 “taps” tuvo uso del CPU en porcentaje de 8.68% de media con 0.58% de desviación estándar y el filtro de 25 “taps” presentó un uso de CPU en 4.87% de media con desviación estándar de 0.55%.

El uso de CPU para el caso del sistema híbrido es sensiblemente más bajo al compararlos con el sistema enteramente de software PS. Esto se debe a que la parte computacional más compleja (el cálculo de las multiplicaciones de los “taps” con los datos de entrada) se hace en la FPGA, liberando el sistema PS de esta tarea y por tanto bajando el porcentaje de uso de CPU.

Pruebas similares se hicieron para el cálculo de la FFT, los datos se muestran en la Tabla 3, para el caso del sistema PS el uso del CPU fue de entre 96.00% de media con 0.82% de desviación estándar, mientras que para el sistema híbrido PS-PL el uso de CPU se redujo hasta 32.25% de media con 2.44% de desviación estándar.

De nuevo, el sistema híbrido mostró un uso mucho menor del CPU al compararlo con el sistema PS. En este caso las variaciones del uso del CPU% no fueron tan acusadas como en los filtros FIR. También, el uso de CPU para esta FFT fue casi de la totalidad del tiempo de procesador para PS. Al comparar los resultados de la FFT con los filtros FIR se obtiene que la FFT es bastante más compleja computacionalmente que el filtro FIR.

No se obtuvo diferencias significativas en el uso de memoria %VSZ entres los sistemas PS e híbrido PS-PL para ninguno de los dos bloques. Para los filtros FIR ejecutados en PS el uso de memoria fue

consistente en un valor de 25% y para el sistema híbrido este porcentaje apenas bajó al 24% de VSZ. Algo similar ocurrió con la FFT, que pasó del 29% de uso de memoria para el sistema PS a 24% del bloque híbrido.

6.2 Conclusiones.

En función de los resultados obtenidos y según los objetivos planteados originalmente, se llega a las siguientes conclusiones:

- Se logró implementar con éxito la plataforma GNU Radio en la tarjeta de desarrollo Zedboard según lo mostrado en la sección 3.3. Para esto se hace uso de Xilinx de Xillybus [26] y la distribución Linux para Zynq7000 de GNU Radio[27].
- La implementación de un canal de comunicación entre el procesamiento en software (del lado PS) y el hardware (PL) fue exitosa. De esta forma se logra acceder a los bloques de hardware como periféricos conectados al procesador presente en el SoC Zynq 7000.
- Se describió el hardware para la FPGA del Zynq7000 que calcula filtros FIR y una transformada FFT de 4 puntos. Estos bloques fueron implementados de tal forma que puedan usar Xillybus como canal de comunicación con el PS.
- Fue posible integrar exitosamente los bloques de hardware FIR y FFT con la plataforma GNU Radio. Estos bloques se utilizaron luego para sustituir bloques de software en GNU Radio. Una vez que se integraron los bloques de hardware se demuestra que el sistema logra ejecutar el procesamiento de señal usando solo bloques en software o utilizando una combinación software y hardware.

- Haciendo uso de la herramienta “top” de Linux se logró medir el rendimiento computacional de la Zedboard, se utilizaron los datos de uso de CPU y porcentaje de uso de memoria (%VSZ).
- Para los filtros FIR el sistema híbrido presenta un mejor rendimiento que solo utilizando bloques de software. El uso de CPU se bajó de promedios de 84.52%, 77.73% y 66.92% para los diferentes filtros FIR ejecutado en PS a usos de 15.27%, 8.68% y 4.87% respectivamente para el sistema híbrido. El uso de memoria se mantuvo similar para ambos casos, siendo 25 y 24% respectivamente.
- Para la transformada FFT el sistema híbrido mostró un rendimiento mejorado con respecto al sistema únicamente de software. El uso de CPU se bajó de 96.00% a 32.25% y el uso de memoria se bajó de 29% a 24%.
- Con lo anterior se cumplió con el objetivo de optimizar sistemas SDR haciendo co-procesamiento con FPGA. Se puso en funcionamiento una plataforma híbrida con la tarjeta Zedboard capaz de ejecutar el SDR GNU Radio y en la que se puede sustituir bloques de software por bloques de hardware en FPGA. La optimización realizada consiste en una reducción significativa del uso de recursos computacionales del sistema híbrido (PS y PL) comparándolo con el sistema de software (PS) para bloques de comunicación SDR.
- Se hace un aporte innovador en la optimización de bloques SDR usando un dispositivo híbrido, se demostró que el uso de CPU y memoria se ve reducido cuando se usa el sistema híbrido PS y PL con respecto al sistema únicamente de software PS.

6.3 Recomendaciones.

Se dan recomendaciones de posibles desarrollos futuros de esta temática aprovechando los avances que se lograron en el desarrollo de este proyecto.

- La distribución Linux denominada como GNURadio-Zynq no posee parte gráfica ni programas básicos para la instalación de paquetes (como apt-get). Se recomienda hacer avances en esta dirección para mejorar la flexibilidad de la distribución.
- Se recomienda instalar en GNU Radio los componentes adecuados para poder hacer uso del USRP con la Zedboard. La transmisión inalámbrica mediante el uso de USRP se logró con la tarjeta DE2i-150 pero no con la Zedboard. Dado que esta última es más flexible para sistemas híbridos, sería de gran valor lograr el funcionamiento del USRP.
- Aprovechando la plataforma se recomienda implementar sistemas de comunicación completos en los que utilicen los bloques FIR y FFT utilizados de este proyecto.
- Explorar la posibilidad de usar herramientas oficiales de Xilinx para la comunicación PS-PL con la finalidad de crear bloques más complejos. Esto porque no fue posible con Xillybus y Vivado HLS crear bloques que realicen operaciones con datos flotantes. Esto abriría la puerta a implementar FFT de mayores tamaños (8, 16, 32, etc).

Bibliografía

- [1] Kenington, P. B. (2005). RF and baseband techniques for software defined radio (p. 177). Norwood: Artech House.
- [2] Welcome to the Wireless Innovation Forum. (n.d.). Retrieved December 10, 2015, from <http://www.wirelessinnovation.org/>
- [3] What is Software Defined Radio. (n.d.). Retrieved December 10, 2015, from <http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf>
- [4] Mitola, J. (1995). The software radio architecture. Communications Magazine, IEEE, 33(5), 26-38
- [5] Overview - GNU Radio - gnuradio.org. (n.d.). Retrieved December 10, 2015, from <http://gnuradio.org/>
- [6] Tucker, D. C., & Tagliarini, G. A. (2009, March). Prototyping with GNU radio and the USRP-where to begin. In Southeastcon, 2009. SOUTHEASTCON'09. IEEE (pp. 50-54). IEEE.
- [7] Ren, Y., Yao, D., & Zhang, X. (2011, November). The implementation of TETRA using GNU Radio and USRP. In Microwave, Antenna, Propagation, and EMC Technologies for Wireless Communications (MAPE), 2011 IEEE 4th International Symposium on (pp. 363-366). IEEE.
- [8] Ivan Rodriguez Abdala, V., & Sanchez Garcia, J. (2014). Empowering Software Radio: IT++ AS A GNU Radio Out-Of-Tree Implementation. Latin America Transactions, IEEE (Revista IEEE America Latina), 12(2), 269-276.
- [9] GNU Radio Manual and C API Reference: Modules. (n.d.). Retrieved December 10, 2015, from <http://gnuradio.org/doc/doxygen/modules.html>

- [10] A Quick Guide to Hardware and GNU Radio. (n.d.). Retrieved December 10, 2015, from <http://gnuradio.org/redmine/projects/gnuradio/wiki/Hardware>
- [11] Ettus Research. (n.d.). Retrieved December 10, 2015, from <http://home.ettus.com/>
- [12] DE2i-150 FPGA Development Kit. (n.d.). Retrieved December 10, 2015, from <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=11&No=529&PartNo=2>
- [13] Avnet Product Brief Zedboard. (n.d.). Retrieved December 10, 2015, from http://zedboard.org/sites/default/files/product_briefs/PB-AES-Z7EV-7Z020_G-v12.pdf
- [14] Zynq-7000 All Programmable SoCs Product Tables and Product Selection Guide. (n.d.). Retrieved December 10, 2015, from http://www.xilinx.com/publications/prod_mktg/zynq7000/Zynq-7000-combined-product-table.pdf
- [15] ARM Information Center. (n.d.). Retrieved December 10, 2015, from <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html>
- [16] Abirami, M.; Hariharan, V.; Sruthi, M.B.; Gandhiraj, R.; Soman, K.P., "Exploiting GNU radio and USRP: An economical test bed for real time communication systems," in *Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on*, vol., no., pp.1-6, 4-6 July 2013.
- [17] Tucker, D. C., & Tagliarini, G. (2009). Prototyping with gnu radio and the usrp-where to begin. *Southeastcon, 2009. SOUTHEASTCON'09. IEEE*, 50-54.
- [18] Serban, C.; Chadha, R.; Chiang, C.J.; Ge, F.; Gottlieb, Y.; Sapello, A.; Sinkar, K.; Moeltner, K., "A GNU-based packet radio for network management field testing," in *MILITARY COMMUNICATIONS CONFERENCE, 2012 - MILCOM 2012*, vol., no., pp.1-6, Oct. 29 2012-Nov. 1 2012.

- [19] Ren, Y., Yao, D., & Zhang, X. (2011, November). The implementation of TETRA using GNU Radio and USRP. In *Microwave, Antenna, Propagation, and EMC Technologies for Wireless Communications (MAPE), 2011 IEEE 4th International Symposium on* (pp. 363-366). IEEE.
- [20] Marwanto, A., Sarijari, M. A., Fisal, N., Yusof, S. K. S., & Rashid, R. (2009, December). Experimental study of OFDM implementation utilizing GNU radio and USRP-SDR. In *Communications (MICC), 2009 IEEE 9th Malaysia International Conference on* (pp. 132-135). IEEE.
- [21] Berardinelli, G., Zetterberg, P., Tonelli, O., Cattoni, A. F., Sørensen, T. B., & Mogensen, P. (2011, November). An SDR architecture for OFDM transmission over USRP2 boards. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on* (pp. 965-969). IEEE.
- [22] Zitouni, R., Ataman, S., & George, L. (2013, October). RF Measurements of the RFX 900 and RFX 2400 Daughter Boards with the USRP N210 Driven by the GNU Radio Software. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on* (pp. 490-494). IEEE.
- [23] Mate, A., Lee, K. H., & Lu, I. T. (2011, May). Spectrum sensing based on time covariance matrix using GNU radio and USRP for cognitive radio. In *Systems, Applications and Technology Conference (LISAT), 2011 IEEE Long Island* (pp. 1-6). IEEE.
- [24] Ferreira, P. V., Diniz, P., Veiga, A., & Carneiro, M. (2012, September). Frequency Response Acquisition of a Digital Radio Transceiver Using USRP Module and GNU Radio Software. In *Computational Intelligence, Modelling and Simulation (CIMSIM), 2012 Fourth International Conference on* (pp. 243-248). IEEE.

- [25] Truong, N. B., Suh, Y. J., & Yu, C. (2013, November). Latency analysis in GNU radio/USRP-based software radio platforms. In *Military Communications Conference, MILCOM 2013-2013 IEEE* (pp. 305-310). IEEE.
- [26] An FPGA IP core for easy DMA over PCIe with Windows and Linux. (n.d.). Retrieved December 10, 2015, from <http://xillybus.com/>
- [27] FPGA Accelerators in GNU Radio with Xilinx's Zynq System on Chip. (n.d.). Retrieved December 10, 2015, from <https://gnuradio.org/redmine/projects/gnuradio/wiki/Zynq>
- [28] Product Downloads. (n.d.). Retrieved December 10, 2015, from <http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html>
- [29] Getting started with Xilinx for Zynq-7000 EPP. (n.d.). Retrieved December 10, 2015, from http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf
- [30] Smith, S. W. (1997). The scientist and engineer's guide to digital signal processing. Capítulo 12.
- [31] Vivado High-Level Synthesis. (n.d.). Retrieved December 10, 2015, from <http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html>
- [32] FPGA coprocessing for C/C programmers. (n.d.). Retrieved December 10, 2015, from <http://xillybus.com/tutorials/vivado-hls-c-fpga-howto-1>
- [33] Woods, R., McAllister, J., Yi, Y., & Lightbody, G. (2008). *FPGA-based implementation of signal processing systems*. John Wiley & Sons. Capítulo 4.
- [34] TFilter - Free online FIR filter design. (n.d.). Retrieved December 10, 2015, from <http://t-filter.engineerjs.com/>
- [35] GNU Octave. (n.d.). Retrieved December 10, 2015, from <https://www.gnu.org/software/octave/>

[36] Octave and Matlab Scripts for GNU Radio. (n.d.). Retrieved December 10, 2015, from <https://gnuradio.org/redmine/projects/gnuradio/wiki/Octave>

[37] Unix Top. (n.d.). Retrieved December 10, 2015, from <http://www.unixtop.org/man.shtml>

Apéndice A. Códigos.

Código C para Vivado HLS 2015.2 para la generación del código de descripción de Hardware para una FFT de 4 puntos con datos de entrada tipo “int”.

```
#include <math.h>
#include <stdint.h>
#include "xilly_debug.h"

void xillybus_wrapper(int *in, int *out) {
#pragma AP interface ap_fifo port=in
#pragma AP interface ap_fifo port=out
#pragma AP interface ap_ctrl_none port=return

    uint32_t tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    int x[4], y[4];
    int zero = 0;

    xilly_puts("Hello, world\n");

    // Handle input data
    tmp0 = *in++;
    tmp1 = *in++;
    tmp2 = *in++;
    tmp3 = *in++;

    x[0] = tmp0;
    x[1] = tmp1;
    x[2] = tmp2;
    x[3] = tmp3;
```

```

// Debug output
xilly_puts("x1=");
xilly_decprint(x[0], 1);
xilly_puts("\n");

// Run the calculations

y[0] = x[0]+x[2]; // This helps HLS in the conversion below
y[1] = x[0]-x[2];
y[2] = x[1]+x[3];
y[3] = x[1]-x[3];

// Handle output data

tmp0 = y[0]+y[2]; // Convert float to uint32_t
tmp1 = y[1];
tmp2 = y[0]-y[2];
tmp3 = y[1];
tmp4 = 0;
tmp5 = -y[3];
tmp6 = 0;
tmp7 = y[3];
*out++ = tmp0;
*out++ = tmp1;
*out++ = tmp2;
*out++ = tmp3;
*out++ = tmp4;
*out++ = tmp5;
*out++ = tmp6;
*out++ = tmp7;
}

```

Código C para Vivado HLS 2015.2 para la generación del código de descripción de Hardware para una FFT de 4 puntos con datos de entrada tipo “int”.

```
#include <math.h>
#include <stdint.h>
#include "xilly_debug.h"

void xillybus_wrapper(int *in, int *out) {
#pragma AP interface ap_fifo port=in
#pragma AP interface ap_fifo port=out
#pragma AP interface ap_ctrl_none port=return

    int coefs[13] = {5, 4, -15, -30, 4, 78, 119, 78, 4, -30, -15, 4, 5};
    static int input[13];

    int i, j;
    int output = 0;
    int tmp_out;

    xilly_puts("Hello, world\n");

    // Handle input data
    for (j = 0; j < 12; j++){
        input[j]=input[j+1];
    }
    input[12] = *in++;

    // Run the calculations
    for (i = 0; i < 13; i++){
        output = output + coefs[i]*input[i];
    }
```

```

// Handle output data
tmp_out = output;
*out++ = tmp_out;
}

```

Código C para la verificación de del funcionamiento de la FFT de 4 puntos. Los datos de entrada son reales 178, 104, 31, 4 y se recibe (real, imaginario) (317, 0), (147, -100), (101, 0), (147, 100).

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>

int main(int argc, char *argv[]) {

    int fdr, fdw;

    struct {
        int v1;
        int v2;
        int v3;
        int v4;
    } tologic;

    struct {
        int v1;
        int v2;
        int v3;
        int v4;
        int v5;
        int v6;
        int v7;
        int v8;
    } fromlogic;

    fdr = open("/dev/xillybus_read_32", O_RDONLY);
    fdw = open("/dev/xillybus_write_32", O_WRONLY);

    if ((fdr < 0) || (fdw < 0)) {
        perror("Failed to open Xillybus device file(s)");
        exit(1);
    }
}

```

```

tologic.v1 = 178; //
tologic.v2 = 104; //
tologic.v3 = 31; //
tologic.v4 = 4; //

// Not checking return values of write() and read(). This must be done
// in a real-life program to ensure reliability.

write(fdw, (void *) &tologic, sizeof(tologic));
read(fdr, (void *) &fromlogic, sizeof(fromlogic));

printf("Data sent: %d\n, %d\n, %d\n, %d\n", tologic.v1, tologic.v2,
tologic.v3, tologic.v4);
printf("Real part: %d\n, %d\n, %d\n, %d\n", fromlogic.v1, fromlogic.v2,
fromlogic.v3, fromlogic.v4);
printf("Imag part: %d\n, %d\n, %d\n, %d\n", fromlogic.v5, fromlogic.v6,
fromlogic.v7, fromlogic.v8);

close(fdr);
close(fdw);

return 0;
}

//Resultados
entrada (real): 178, 104, 31, 4
Salida (real, imaginario): (317, 0), (147, -100), (101, 0), (147, 100).

```

Código C para la verificación de del funcionamiento de la FIR. Los datos de entrada son un arreglo de 21 datos enteros de entrada y la salida también. Los arreglos de entrada y salida son

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>

int main(int argc, char *argv[]) {

    int fdr, fdw;

```

```

    int input[32]={0, 1, 2, 3, 3, 3, 3, 2, 1, 0, 0, 0, 1, 2, 3, 3, 3, 2, 1,
0, 0};
    int output;

    int i;

    fdr = open("/dev/xillybus_read_32", O_RDONLY);
    fdw = open("/dev/xillybus_write_32", O_WRONLY);

    if ((fdr < 0) || (fdw < 0)) {
        perror("Failed to open Xillybus device file(s)");
        exit(1);
    }

    // Not checking return values of write() and read(). This must be done
    // in a real-life program to ensure reliability.

    for (i=0; i < 21; i++){
        write(fdw, (void *) &input[i], sizeof(input[i]));
        read(fdr, (void *) &output, sizeof(output));
        printf("%d\n",output);
    }

    close(fdr);
    close(fdw);

    return 0;
}

//Resultados

input[32]={0, 1, 2, 3, 3, 3, 3, 2, 1, 0, 0, 0, 1, 2, 3, 3, 3, 2, 1, 0, 0};
output[32]={0, 5, 14, 8, -33, -74, -22, 174, 440, 647, 740, 740, 652, 454,
182, -55, -48, -60, 173, 460, 693};

```

Código de Python a ejecutar en la Zedboard que genera una señal resultante de la suma de dos senoidales de amplitud 200 y 600 Hz respectivamente. Se envía los datos a la FPGA que está ejecutando el hardware de un filtro FIR. Se obtienen los datos del filtrado y se guardan en un archivo.

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: Fir Lowpass Zed
# Generated: Mon Nov 16 10:54:10 2015
#####

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation

```

```

from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser

class FIR_lowpass_zed(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Fir Lowpass Zed")

        #####
        # Variables
        #####
        self.samp_rate = samp_rate = 2000

        #####
        # Blocks
        #####
        self.blocks_file_source_0 = blocks.file_source(gr.sizeof_int*1,
"/dev/xillybus_read_32", False)
        self.blocks_file_sink_1 = blocks.file_sink(gr.sizeof_int*1,
"/home/root/lowpassfile", False)
        self.blocks_file_sink_1.set_unbuffered(False)
        self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_int*1,
"/dev/xillybus_write_32", False)
        self.blocks_file_sink_0.set_unbuffered(False)
        self.blocks_add_xx_0 = blocks.add_vii(1)
        self.analog_sig_source_x_0_0 = analog.sig_source_i(samp_rate,
analog.GR_COS_WAVE, 250, 100, 0)
        self.analog_sig_source_x_0 = analog.sig_source_i(samp_rate,
analog.GR_COS_WAVE, 200, 100, 0)

        #####
        # Connections
        #####
        self.connect((self.blocks_file_source_0, 0),
(self.blocks_file_sink_1, 0))
        self.connect((self.blocks_add_xx_0, 0), (self.blocks_file_sink_0,
0))
        self.connect((self.analog_sig_source_x_0_0, 0),
(self.blocks_add_xx_0, 1))
        self.connect((self.analog_sig_source_x_0, 0),
(self.blocks_add_xx_0, 0))

# QT sink close method reimplementation

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate

```

```

        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
        self.analog_sig_source_x_0_0.set_sampling_freq(self.samp_rate)

if __name__ == '__main__':
    parser = OptionParser(option_class=eng_option, usage="%prog:
[options]")
    (options, args) = parser.parse_args()
    tb = FIR_lowpass_zed()
    tb.start()
    raw_input('Press Enter to quit: ')
    tb.stop()
    tb.wait()

```

Apéndice B. Fotografías.

Se muestra figuras de diferentes dispositivos de hardware utilizado en la realización de este proyecto.

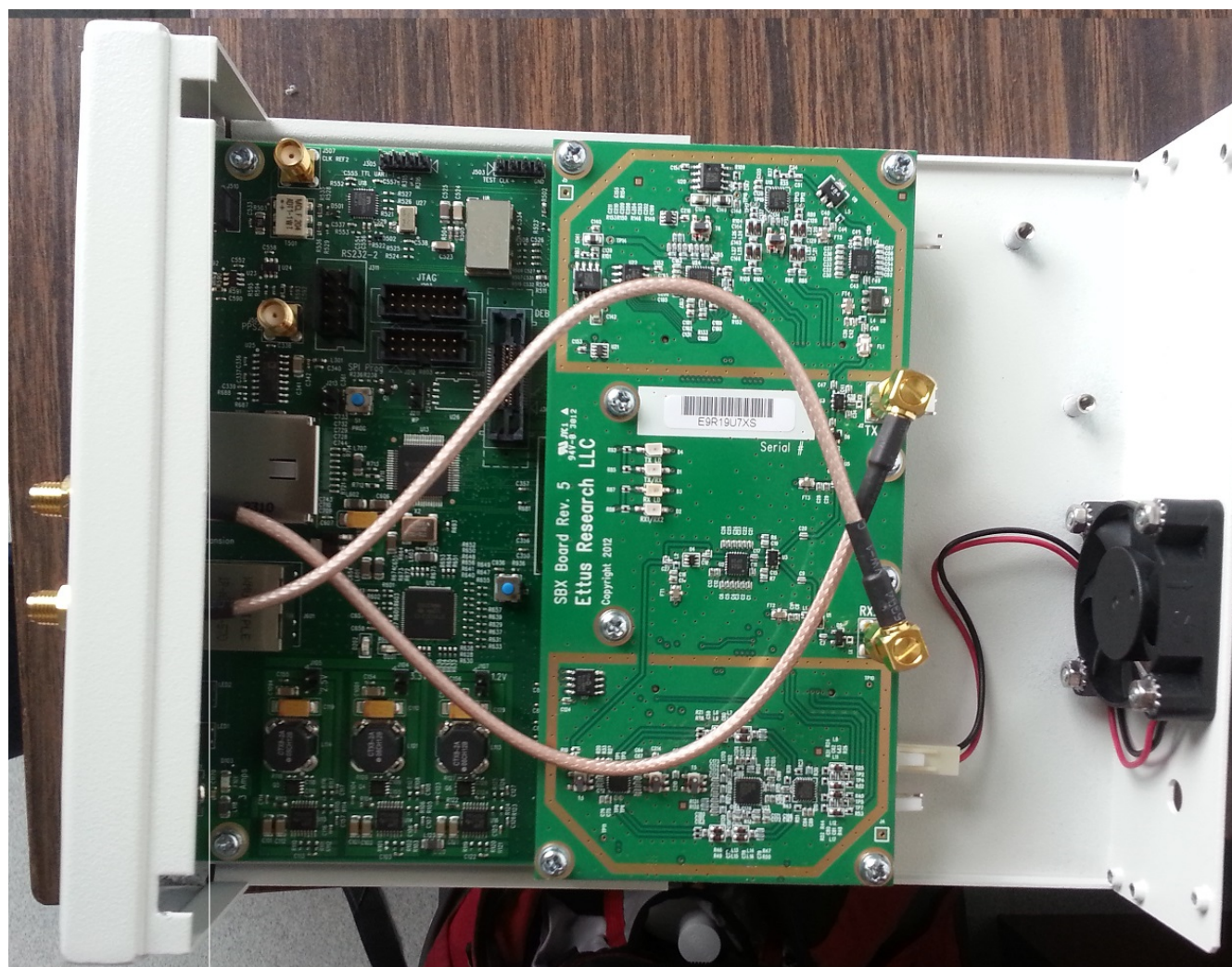


Figura 34: USRP y SBX, vista interna del equipo.



Figura 35: Sistema USRP con su respectiva antena, conectado a la computadora mediante Ethernet.



Figura 36: DE2i-150 ejecutando Ubuntu 12.04.

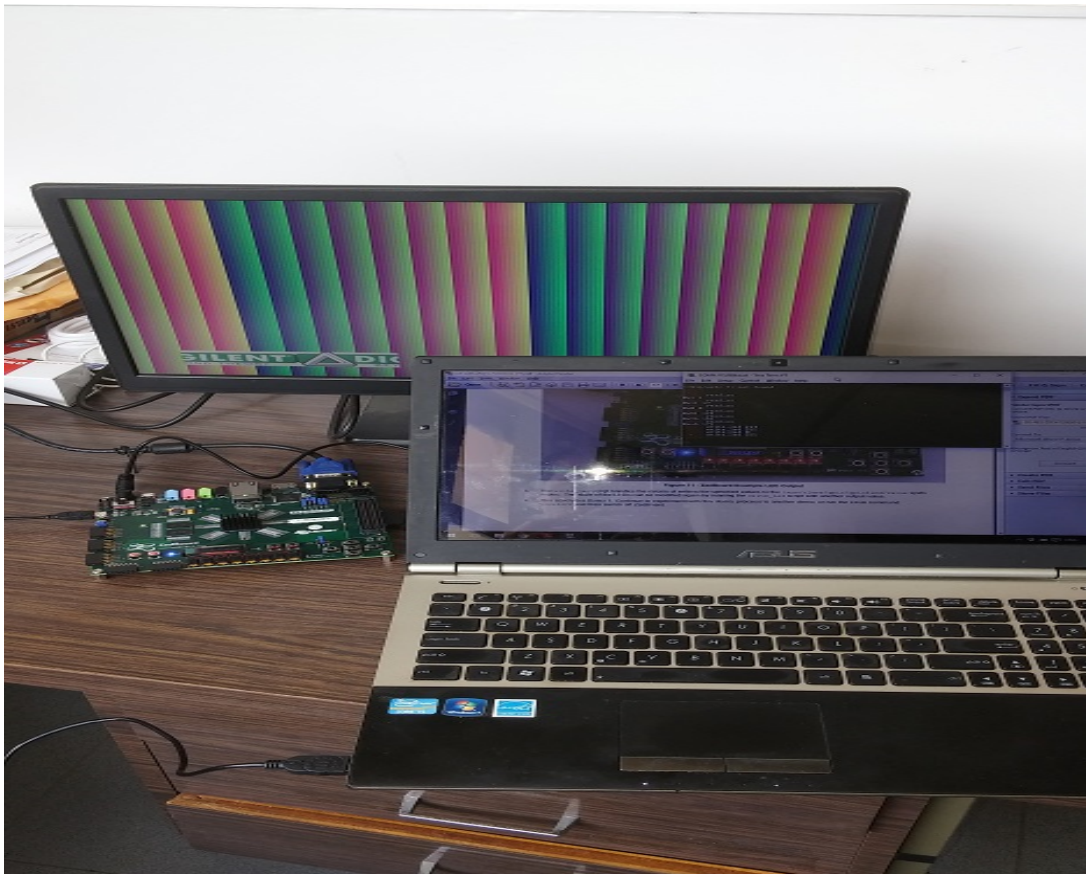


Figura 37: Zedboard ejecutando imagen de “booteo” de fábrica.