



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Cabecera Radio Remota utilizando LimeSDR y ODROID-XU4

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación

AUTOR: Jordi Cerezo Tobias

DIRECTOR: Antoni Gelonch Bosch

FECHA: 2 de febrero del 2018

Título: Cabecera radio remota utilizando LimeSDR y ODROID-XU4

Autor: Jordi Cerezo Tobias

Director: Antoni Gelonch Bosch

Fecha: 2 de febrero del 2018

Resumen

La Radio Definida por Software (SDR), a diferencia de los sistemas de radio convencionales implementados únicamente mediante hardware, ha tenido una gran repercusión gracias a la gran flexibilidad que ofrece el software.

El objetivo de este trabajo es desarrollar una cabecera de radio remota utilizando la LimeSDR, una placa SDR, conectada a una ODROID XU-4. La conexión se realizará mediante el puerto USB 3.0 ya que permite una velocidad de transmisión alta.

Esta cabecera de radio remota se incorporará a un Cloud-RAN para facilitar la virtualización de los recursos de radio. Como la LimeSDR transmite y recibe, la comunicación entre ODROID XU-4 y servidor es full dúplex. Buscando una velocidad de transmisión alta, la conexión entre la ODROID XU-4 y el servidor en el Cloud -RAN se realizará mediante cable Ethernet Gigabit.

Esta comunicación entre ODROID XU-4 y servidor se llevará a cabo mediante sockets. Para ello se han desarrollado dos Virtual Network Functions (VNF) de tal manera que, una se colocará en el servidor de la nube y la otra en la Odroid-XU4.

Conforme va progresando el desarrollo de la cabecera de radio remota se han ido realizando tests. Estos verifican el buen funcionamiento de la placa SDR, el correcto funcionamiento de las VNFs, que la configuración remota de los parámetros más relevantes para la transmisión y recepción de LimeSDR se lleva a cabo correctamente y de que el sistema entero funcione de forma correcta.

El presente trabajo, aparte del desarrollo y verificación, también tiene documentado una introducción a la radio definida por software, otra para la cabecera de radio remota, el planteamiento del diseño, el número total de horas que le he dedicado a este trabajo de final de grado y la tecnología en la que se basa esta cabecera de radio remota, tanto su hardware como su software, para así entender mejor tanto las oportunidades que pueden ofrecer como las limitaciones.

Títol: Capçalera ràdio remota utilitzant LimeSDR i ODROID-XU4

Autor: Jordi Cerezo Tobias

Director: Antoni Gelonch Bosch

Fecha: 2 de febrer del 2018

Resum

La Radio Definida per Software (SDR), a diferencia dels sistemes de ràdio convencionals implantats únicament mitjançant hardware, ha tingut una gran repercussió gràcies a la gran flexibilitat que ofereix el software.

L'objectiu d'aquest treball es desenvolupar una capçalera de ràdio remota utilitzant la LimeSDR, una placa SDR, connectada a una ODROID-XU4. La connexió es realitzarà mitjançant el port USB 3.0 ja que permet una velocitat de transmissió alta.

Aquesta capçalera de ràdio remota s'incorporarà a un Cloud-RAN per facilitar la virtualització dels recursos de ràdio. Com la LimeSDR transmet i rep, la comunicació entre ODROID XU-4 i servidor és full dúplex. Buscant una velocitat de transmissió alta, la connexió entre la ODROID XU-4 i el servidor en el Cloud -RAN es realitzaran mitjançant cable Ethernet Gigabit.

Aquesta comunicació entre ODROID XU-4 i servidor es portarà a terme mitjançant sockets. Per això s'han desenvolupant dos VNF de tal manera que, una es col·locarà al servidor del núvol i l'altra en la Odroid-XU4.

A mesura que va progressant el desenvolupament de la capçalera ràdio remota s'han anat realitzant tests. Aquests verifiquen el bon funcionament de la placa SDR, el correcte funcionament de les VNFs, que la configuració remota dels paràmetres més rellevants per a la transmissió i recepció de LimeSDR es porta a terme correctament i que el sistema sencer funcioni de forma correcta.

El present treball, a part del desenvolupament i verificació, també té documentat una introducció a la ràdio definida per software, una altra per a la capçalera de ràdio remota, el plantejament del disseny, el nombre total d'hores que li he dedicat a aquest treball de fi de grau i la tecnologia en què es basa aquesta capçalera de ràdio remota, tant el seu maquinari com el seu programari, per així entendre millor tant les oportunitats que poden oferir com les limitacions.

Title: Remote Radio Head using LimeSDR and ODROID-XU4

Author: Jordi Cerezo Tobias

Director: Antoni Gelonch Bosch

Date: February 2nd, 2018

Overview

The Software Defined Radio (SDR), unlike conventional radio systems implemented only by hardware, has had a great impact thanks to the great flexibility offered by the software.

The objective of this work has been to develop a remote radio header using the LimeSDR, an SDR board, connected to an ODROID XU-4. The connection will be made through the USB 3.0 port since it allows a high transmission speed.

This remote radio header will be incorporated into a Cloud-RAN to facilitate the virtualization of radio resources. As the LimeSDR transmits and receives, the communication between ODROID XU-4 and server is full duplex. Looking for a high transmission speed, the connection between the ODROID XU-4 and the server in the Cloud-RAN will be made using Gigabit Ethernet cable.

This communication between ODROID XU-4 and server will be carried out through sockets. To do this, two Virtual Network Functions (VNF) have been developed in such a way that one will be placed on the cloud server and the other on the ODROID-XU4.

As the development of the remote radio header progresses, tests have been carried out. These verify the proper functioning of the SDR board, the correct operation of the VNFs, that the remote configuration of the most relevant parameters for transmission and reception of LimeSDR is carried out correctly and that the entire system works correctly.

The present work, apart from the development and verification, also has documented an introduction to the software-defined radio, another for the remote radio headend, the design approach, the total number of hours that I have devoted to this work at the end of degree and the technology on which this remote radio headend is based, both its hardware and its software, in order to better understand both the opportunities it can offer and the limitations.

Dejen que el futuro diga la verdad y evalúe a cada quien acorde a su trabajo y logros. El presente es de ellos; el futuro, por el cual yo ya he trabajado, es mío.

Nikola Tesla

Quiero empezar agradeciendo enormemente al Dr. Antoni Gelonch por acogerme y ser el tutor de mi trabajo. Gracias por tu tiempo, confianza, opiniones, ayudas, atención, sabiduría y sobre todo por la simpatía y actitud mostrada a mi persona. Ha sido un placer poder observar y absorber el modo de resolver los problemas. Te deseo de corazón la mayor suerte para tus futuros trabajos y publicaciones.

A mis padres, porque ellos me dieron “simplemente” la oportunidad de vivir. Modelos a seguir, tanto en lo personal como en lo profesional. Familia trabajadora que me ha formado bajo los valores del respeto, la educación y el amor. Ojalá algún día pueda devolver todo lo que me habéis dado. Gracias por haberme hecho llegar hasta aquí de una forma u otra.

A todos mis abuelos. Personas tolerantes, honestas, bondadosas y generosas con las que he pasado gran parte de mi infancia. Algunos ya no están a mi lado, pero su recuerdo prevalece siempre en mi corazón.

Por otro lado, agradecer a todos los amigos que me han aguantado durante estos duros meses y han estado ayudándome aportándome ideas y dando su punto de vista. Agradecer también a Maria, la persona más incondicional que jamás he conocido, por la ayuda que me ha brindado leyendo y corrigiendo toda la memoria.

Quiero dar las gracias también a todos los profesores que he tenido durante la carrera y me han impregnado de sus conocimientos llegando a desarrollarme como ingeniero. No hace falta ir lejos para encontrar docentes profesionales y, lo más importante, personas como increíbles.

Gracias a todos mis compañeros de clase, no cabe duda de que todos estos años que hemos compartido no los olvidaré.

Gracias a David Ortiz, *Compliance Business Unit Manager de Idneo Technologies S.L.*, por darme la oportunidad de trabajar en el departamento de *Compliance* con un magnífico grupo de trabajo y bellísimas personas en el cual he adquirido muchos conocimientos que también me han servido para este trabajo. Gracias también a Mónica Morales del departamento de *Electrical Safety* por facilitarme la caja empleada para proteger la LimeSDR.

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO 1. RADIO DEFINIDA POR SOFTWARE	2
1.1. Concepto de SDR	2
1.2. Breve historia y evolución de la SDR	3
1.3. Aportación de la SDR	4
CAPÍTULO 2. PLATAFORMA SDR Y ENTORNO SOFTWARE.....	5
2.1. LimeSDR	5
2.1.1. Conexiones RF	6
2.1.2. Conexión de datos y alimentación	8
2.1.3. FPGA	8
2.1.4. FPRF	8
2.2. LimeSuite	11
2.2.1. LMS7 Drivers	12
2.2.2. Board Support.....	12
2.2.3. Lime Suite GUI	13
2.2.4. SDR Interfaces	13
CAPÍTULO 3. REMOTE RADIO HEAD.....	14
3.1. Introducción a la RRH	14
3.2. Diseño de la RRH.....	15
3.2.1. Planteamiento del diseño	15
3.2.2. Comunicación entre Odroid-XU4 y servidor.....	16
3.2.3. Hardware de nuestra RRH	18
3.2.4. Software de nuestra RRH.....	20
3.3. Desarrollo y programación de la RRH.....	22
3.3.1. Sistema Operativo	22
3.3.2. Instalación del software previo	22
3.3.3. Pruebas con LimeSuiteGUI	22
3.3.4. Otros entornos	22
3.3.5. Lime API	23
3.3.6. VNF	23
3.4. Verificación del funcionamiento y la configuración	25
3.4.1. Test de la frecuencia de transmisión de la LimeSDR	25
3.4.2. Test de la ganancia de transmisión de la LimeSDR	27
3.4.3. Test del número de canales de la LimeSDR	29
3.4.4. Test de la habilitación de canales de la LimeSDR	29
3.4.5. Test del Ancho de banda de la LimeSDR	30
3.4.6. Test del factor de sobremuestreo de la LimeSDR	31
3.4.7. Test del LPF de la LimeSDR	32
3.4.8. Test del funcionamiento del TX de la LimeSDR.....	34
3.4.9. Test del funcionamiento del RX de la LimeSDR	36
3.4.10. Test del funcionamiento de las VNFs.....	39
3.4.11. Test del funcionamiento del TX de la RRH	41

CAPÍTULO 4. PLANIFICACIÓN DEL TRABAJO	44
4.1. Estudio previo.....	44
4.2. Diseño.....	44
4.3. Desarrollo	44
4.4. Verificación	44
4.5. Documentación.....	44
CONCLUSIONES	46
Líneas futuras.....	47
REFERENCIAS Y BIBLIOGRAFÍA	48
ANEXOS	51
ANEXO 1: Guía de configuración e instalación para ODROID XU4	51
ANEXO 2: Guía de configuración e instalación para LimeSDR en ODROID XU4	54
ANEXO 3: Comprobación del transceptor LMS7002M	58
ANEXO 4: Trabajar con Pothos.....	61
ANEXO 5: Algunas funciones de la Lime API	67
ANEXO 6: Función para dibujar la FFT mediante MATLAB	69
ANEXO 7: Teorema de muestreo de Nyquist-Shannon.....	70
ANEXO 8: Reconstrucción de la señal analógica	71
ANEXO 9: Aliasing y filtro antialiasing	73
ANEXO 10: Verificación del moduladora de cuadratura	75
ANEXO 11: Guía de configuración y ejecución de las VNFs	76

ÍNDICE DE FIGURAS DE LA MEMORIA

Fig. 1.1	Concepto de Radio Definida por Software [1]	2
Fig. 2.1	Conectores y componentes de LimeSDR [5]	5
Fig. 2.2	Diagrama de bloques de LimeSDR	6
Fig. 2.3	Conexiones RF	7
Fig. 2.4	Diagrama de bloques del transceptor LMS7002M [18]	9
Fig. 2.5	Cadena de filtrado analógico en el receptor [18]	10
Fig. 2.6	Respuesta de amplitud analógica del RX LPF [18]	10
Fig. 2.7	Respuesta de amplitud analógica del TX LPF [18]	11
Fig. 2.8	Arquitectura de Lime Suite [15]	11
Fig. 2.9	Diagrama de bloques de la funcionalidad de Lime Suite y Soapy [8]	13
Fig. 3.1	RRH para estaciones base de telefonía móviles [28]	15
Fig. 3.2	Diagrama de bloques del diseño de la RRH	15
Fig. 3.3	Implementación de la RRH	16
Fig. 3.4	Comunicación entre Servidor y Odroid-XU4	17
Fig. 3.5	Creación de sockets en una VNF	18
Fig. 3.6	LimeSDR con caja de plástico negra	19
Fig. 3.7	Odroid-XU4 con carcasa de plástico	19
Fig. 3.8	Estructura de la VNF colocada en el servidor	21
Fig. 3.9	Estructura de la VNF colocada en la Odroid-XU4	21
Fig. 3.10	Transmisión del tono utilizando como frecuencia central 1.0 GHz	26
Fig. 3.11	Transmisión del tono utilizando como frecuencia central 1.3 GHz	26
Fig. 3.12	Transmisión con una ganancia de 0.1. Potencia de -74.00 dBm	27
Fig. 3.13	Transmisión con una ganancia de 0.4. Potencia de -49.73 dBm	28
Fig. 3.14	Transmisión con una ganancia de 0.9. Potencia de -17.73 dBm	28
Fig. 3.15	Advertencias al configurar que no haya transmisiones	29
Fig. 3.16	Deshabilitación del canal por el cual estábamos enviando una señal a 2.0 GHz	29
Fig. 3.17	Transmisión de canal de ruido con una frecuencia de muestreo y ancho de banda de 60MHz	30
Fig. 3.18	Transmisión de un canal de ruido con una frecuencia de muestreo de 10MHz y con un factor 2 de sobremuestreo	31
Fig. 3.19	Transmisión de un canal de ruido con un ancho de banda de 5MHz con un factor de sobremuestreo de 2 sin LPF	32
Fig. 3.20	Transmisión de un canal de ruido con un ancho de banda de 5MHz con un factor de sobremuestreo de 2 con LPF	33
Fig. 3.21	Escenario del test del funcionamiento del TX de la LimeSDR	34
Fig. 3.22	Tono generado por LimeSDR en tiempo	34
Fig. 3.23	Tono generado por LimeSDR en frecuencia	35
Fig. 3.24	Tono generado por LimeSDR con un coseno en fase y seno cuadratura	35
Fig. 3.25	Tono generado por LimeSDR con un seno en fase y coseno cuadratura	36
Fig. 3.26	Escenario del test del funcionamiento del RX de la LimeSDR	36
Fig. 3.27	Verificación de creación del archivo donde se ha almacenado el buffer de la LimeSDR	37
Fig. 3.28	Respuesta en frecuencia conseguida a partir del archivo	37
Fig. 3.29	Espectro obtenido en RX a partir de la función de MATLAB	38
Fig. 3.30	Espectro del tono TX visualizado en el osciloscopio	38

Fig. 3.31	Valor en hexadecimal de las 10 primeras muestras generadas.	39
Fig. 3.32	Valor en hexadecimal de la primera muestra enviada.....	39
Fig. 3.33	Verificación de que el valor en hexadecimal de la primera recibida coincide con la enviada.....	40
Fig. 3.34	Verificación de los valores en hexadecimal guardados en el array coinciden con los generados.....	40
Fig. 3.35	Verificación del envío de 1024 y parámetros de control por parte de ambas VNFs.	40
Fig. 3.36	Configuración de la RRH desde el servidor.....	41
Fig. 3.37	Envío de datos del servidor a Odroid.	42
Fig. 3.38	Recepción de datos del Odroid.	42
Fig. 3.39	RRH en standby esperando datos desde el servidor.	42
Fig. 3.40	RRH TX.....	43

ÍNDICE DE TABLAS

Tabla 2.1.1.	Figura de ruido del LNA según la banda que se trabaje.....	7
--------------	--	---

ÍNDICE DE ACRÓNIMOS

ADC	Analog to Digital Converter (<i>Conversión analógica-digital</i>)
AGC	Automatic Gain Control (<i>Control automático de ganancia</i>)
API	Application Programming Interface (<i>Interfaz de programación de aplicaciones</i>)
ARM	ARM es una arquitectura RISC de 32 y 64 bits.
ASICs	Application-Specific Integrated Circuit (<i>Circuito integrado de aplicación específica</i>)
BBU	Base Band Unit (<i>Unidad de banda base</i>)
BTS	Base Transceiver Station (<i>Estación base</i>)
BW	BandWidth (<i>Ancho de banda</i>)
CDMA	Code Division Multiple Access (<i>Multiplexación por división de código</i>)
DAB	Digital Audio Broadcasting (<i>Transmisión digital de audio</i>)
DAC	Digital to Analog Converter (<i>Conversión digital-analógica</i>)
DDC	Digital Down Converter (<i>Convertidor digital descendente</i>)
DSPs	Digital Signal Processor (<i>Procesador digital de señales</i>)
DUC	Digital Up Converter (<i>Convertidor digital ascendente</i>)
DVB-T	Digital Video Broadcasting – Terrestrial (<i>Difusión de Video Digital – Terrestre</i>)
FFT	Fast Fourier Transform (<i>Transformada rápida de Fourier</i>)
FM	Frequency Modulation (<i>Modulación de frecuencia</i>)
FPGAs	Field Programmable Gate Arrays (<i>Matriz de puertas programables</i>)
GPIO	General Purpose Input/Output (<i>Entrada/Salida de Propósito General</i>)
GSM	Global System for Mobile communications (<i>Sistema global para las comunicaciones móviles</i>)
GUI	Graphical user interface (<i>Interfaz gráfica de usuario</i>)
I+Q	In-Phase and Quadrature (<i>Fase y cuadratura</i>)
IF	Intermediate Frequency (<i>Frecuencia intermedia</i>)
JTAG	Joint Test Action Group (<i>Grupo de acción conjunta de prueba</i>)
LNA	Low-Noise Amplifier (<i>Amplificador de bajo ruido</i>)
LPF	Low Pass Filter (<i>Filtro Paso Bajo</i>)
LTE	Long Term Evolution (<i>Evolución de la norma 3GPP UMTS</i>)
MIMO	Multiple-input Multiple-output (<i>Múltiple entrada múltiple salida</i>)
PCI	Peripheral Component Interconnect (<i>Interconexión de Componentes Periféricos</i>)
PLL	Phase locked loop (<i>Lazo de fijación de fase</i>)

PSK	Phase-shift keying (<i>Modulación por desplazamiento de fase</i>)
QAM	Quadrature Amplitude Modulation (<i>Modulación de amplitud en cuadratura</i>)
RAM	Random Access Memory (<i>Memoria de acceso aleatorio</i>)
RF	Radiofrequency (<i>Radiofrecuencia</i>)
RFIC	Radio frequency integrated circuit (<i>Circuitos integrados que trabajan en el rango de ondas de radiofrecuencia</i>)
RISC	Reduced Instruction Set Computer (<i>Ordenador con Conjunto Reducido de Instrucciones</i>)
RRH	Remote Radio Head (<i>Cabecera de radio remota</i>)
RRU	Remote Radio Unit (<i>Unidad de radio remota</i>)
RX	Reception (<i>Recepción</i>)
SDR	Software Defined Radio (<i>Radio definida por software</i>)
SPI	Serial Peripheral Interface (<i>Interfície de periféricos en serie</i>)
TI	Information technology (<i>La tecnología de la información</i>)
TX	Transmission (<i>transmisión</i>)
UMTS	Universal Mobile Telecommunications System (<i>Sistema universal de telecomunicaciones móviles</i>)
USB	Universal Serial Bus (<i>Bus universal en serie</i>)
VID/PID	Vendor ID/ Product ID (<i>Identificador de venta/ Identificador del producto</i>)
VNF	Virtual Network function (<i>Función de red virtual</i>)
WCDMA	Wideband Code Division Multiple Access (<i>Acceso múltiple por división de código de banda ancha</i>)

INTRODUCCIÓN

La Radio Definida por Software (SDR), a diferencia de los sistemas de radio convencionales implementados únicamente mediante hardware, ha tenido una gran repercusión gracias a la gran flexibilidad que ofrece el software. Varios ejemplos de utilidades y servicios que puede ofrecer la SDR son Televisión Digital, espionaje, radios FM e implementación de una estación base LTE entre otras.

En este trabajo se ha propuesto, utilizando la plataforma SDR LimeSDR conectada a una ODROID XU-4, diseñar y llevar a cabo una cabecera radio remota que sea capaz de transmitir y recibir, así como analizar la calidad de la señal de esta. Otro punto importante ha sido el estudio y documentación de la tecnología en la que se basa esta cabecera de radio remota, tanto su hardware como su software, para entender de manera precisa su funcionamiento y determinar la viabilidad de apostar por ella, tanto para nuestro caso de uso como para otros.

En el primer capítulo de esta memoria, se define el concepto de radio definida por software para introducirnos en el entorno, se dan unas pinceladas de su funcionamiento y se comenta un poco la historia, evolución y aportación.

En el segundo capítulo, se presenta la plataforma SDR y el entorno software con los que se ha trabajado. Para ambos se han detallado sus características y se ha profundizado un poco. En el caso de la plataforma SDR, se ha comentado los conectores, la FPGA y el FPRF mientras que para el software: los drivers, el soporte, la interfaz gráfica de usuario y la explicación de cómo interactúa el software con el hardware.

El tercer capítulo, se describe el proceso de diseño, desarrollo y programación del sistema realizado fruto de este trabajo, denominado Remote Radio Head. También se hace una pequeña introducción al concepto RRH y se documenta la verificación de la configuración y el correcto funcionamiento del sistema.

En el cuarto capítulo, se quiere reflejar como se ha realizado el conjunto de tareas que se han llevado a cabo, el orden que se ha seguido en el desarrollo de la RRH y el número total de horas que le he dedicado a este trabajo de final de grado.

Finalmente, se hace una pequeña reflexión sobre la consecuencia de los objetivos de este trabajo. Se comentan, asimismo, algunos aspectos a mejorar y desarrollar de nuestra RRH a modo descriptivo de posibles mejoras.

CAPÍTULO 1. RADIO DEFINIDA POR SOFTWARE

Este primer capítulo consiste en presentar y explicar el concepto de Radio Definida por Software, analizar el diagrama de bloques de este concepto explicando brevemente sus funciones y finalmente acabar conociendo mejor la SDR con un breve resumen de su historia junto a las aportaciones de esta tecnología.

1.1. Concepto de SDR

La radio definida por Software (SDR) es un sistema de radiocomunicaciones donde los componentes que normalmente se implementan en hardware se implementan a través de software y son reconfigurables. Es decir, mediante el uso de un ordenador, podemos modificar parámetros de los diferentes componentes que conforman nuestro sistema (filtros, amplificadores, mezcladores, etc.) para lograr trabajar con una modulación deseada o bien transmitir/recibir a una frecuencia (o banda de frecuencia) concreta.

Una característica muy importante de la SDR es poder reconfigurar el transceptor. Esto permitirá que pueda adaptarse a diferentes interfaces de comunicación e incorporar nuevas aplicaciones y servicios.

En la siguiente figura se mostrará el concepto de SDR:

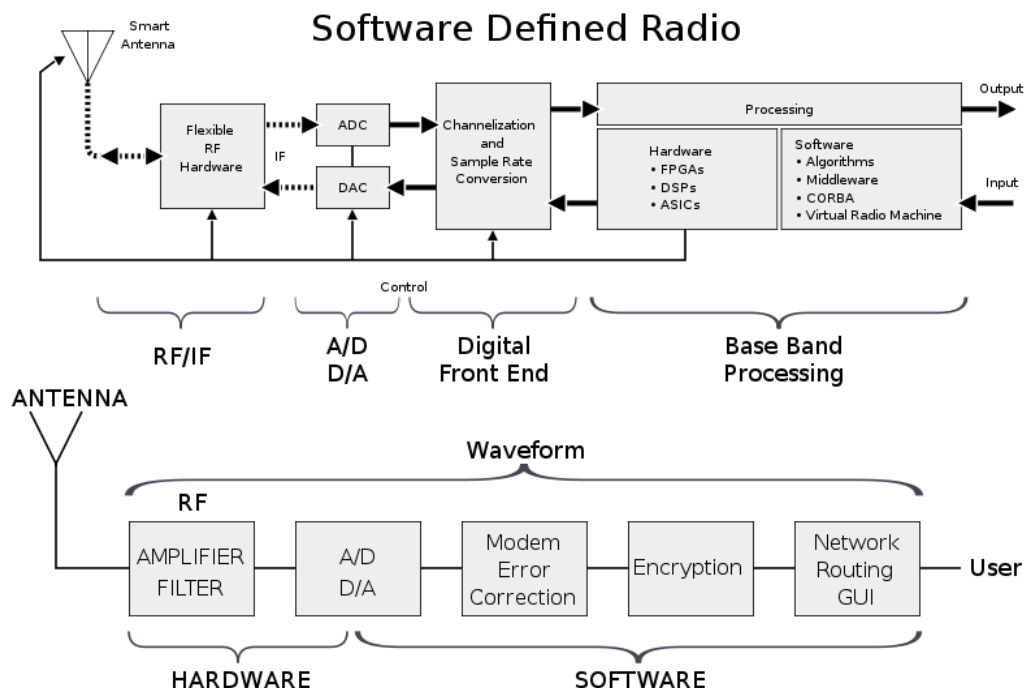


Fig. 1.1 Concepto de Radio Definida por Software [1]

Con el objetivo de aclarar el concepto un poco más, analizaremos brevemente el diagrama de bloques de la parte superior de la figura anterior. En lo primero que nos fijamos es que hay DAC y ADC donde podemos apreciar claramente por las flechas, dos sentidos, transmisión y recepción. Observamos también que hay indicadas secciones.

Empezando por la izquierda, donde está ubicada la antena, nos encontramos la sección de RF/IF. Esta sección tiene la responsabilidad de transmitir/recibir las señales de RF. En el caso de la recepción, prepara y convierte en IF la señal que nos llega, mientras que para el caso de la transmisión, amplifica y modula la señal de IF.

A lo que la parte IF se refiere, en el caso de la recepción, se encarga de digitalizar la señal IF, pasarla a banda base mediante un módulo DDC (Digital Down Converter) y diezmarla para disminuir su tasa de muestreo con el objetivo de adaptarla a la capacidad de transmisión de datos del bus que comunica con la sección de procesado, mientras que para el caso de la transmisión, sería la inversa, pasaríamos la señal de banda base a IF mediante un módulo DUC (Digital Up Converter), interpolamos y haríamos la conversión digital-analógica de la señal para que esta pueda ser transmitida.

Los encargados de la digitalización y el paso a analógico son los módulos ADC o DAC. Todas las operaciones llevadas a cabo en estos módulos se realizan a una velocidad alta la cual viene determinada por la frecuencia de muestreo.

El apartado de software, podríamos decir que es el punto clave de la SDR. Mediante un ordenador se ejecutará un software con el que se manipulará los datos que se han digitalizado en el bloque anterior, consiguiendo así, realizar muchísimas funciones sobre la señal.

El hecho de implementar las funciones mediante software hace que se trabaje con tecnologías de procesado programable como FPGAs, DSPs, ASICs, etc

1.2. Breve historia y evolución de la SDR

Para adentrarnos más en el mundo de la SDR se pretende informar brevemente de su historia y de cómo evolucionará a largo plazo, demostrando la importancia de esta.

En 1984, E-Systems inventó el término de "Software Radio". Utilizaban este término para referirse a un receptor digital de banda base capaz de rechazar interferencias y demodulaciones de señales de banda ancha. Esto se realizaba mediante una serie de procesadores que realizaban un filtrado adaptativo.

A finales de los años 80 se diseñó e implementó el primer transceptor de radio basado en software.

Entre los años 1990 y 1995, la tecnología SDR entró en esplendor al utilizarse para el programa militar de la fuerza aérea de los Estados Unidos, SPEAKEasy.

Este programa consistió en desarrollar una radio donde los componentes estuvieran implementados en software. Detectaba diez protocolos de radio militares diferentes pudiendo operar entre 2MHz y 2GHz.

Entre estos años, concretamente en 1992, Joseph Mitola realizó la primera publicación sobre el tema del software radio: "Software Radio: Survey, Critical Analysis and Future Directions" [1].

Entre 2010 y 2012 se descubrió que algunos receptores tenían la capacidad de decodificar las componentes de la señal en fase y cuadratura, concretamente todo empezó en marzo de 2010 cuando, capturando paquetes USB del software de Windows en modo FM y DAB con el objetivo de sacar un equivalente para Linux, se dieron cuenta de que algunos de los receptores de radiodifusión de video digital terrestre (DVB-T) comerciales tenían la capacidad de decodificar las componentes de la señal en fase y cuadratura (I+Q). La comunidad de código abierto descubrió que suprimiendo los bloques demoduladores DVB-T el controlador envía la señal en banda base digitalizada en fase y cuadratura al ordenador, donde mediante software puede ser tratada. Realtek y Osmocomo trabajaron independientemente desarrollando software o adaptando existentes con el objetivo de potenciar la SDR. Finalmente, en 2012 se unieron y crearon RTLSDR [22].

Como se ha visto anteriormente, a medida que se ha ido investigando, la SDR ha ido evolucionando y potencializando. Por esto se prevé que, a largo plazo, las radios definidas por software se convertirán en la tecnología dominante en radiocomunicaciones.

1.3. Aportación de la SDR

Las SDR son útiles tanto para aplicaciones de uso militar como para aplicaciones de uso comercial gracias al manejo de varios estándares de radiocomunicaciones.

A diferencia de los sistemas de radio convencionales implementados únicamente mediante hardware, nos damos cuenta de que la SDR ha tenido una gran repercusión ya que gracias a la gran flexibilidad que ofrece el software, un mismo equipo puede añadir prestaciones nuevas o bien puede implementar diferentes sistemas de radiocomunicación.

La aportación de esa gran flexibilidad realizándose de forma dinámica y eficiente a un precio razonable hace que la SDR sea muy interesante.

También aporta al medioambiente ya que, desde el punto de vista ecológico, al sustituir gran parte del hardware gracias a la implementación mediante software, se reducen residuos industriales. Por otro lado, desde el punto de vista legal, aspectos regulatorios como certificaciones de terminales pueden ser más fáciles ya que lo único que debería certificarse es el software [24].

CAPÍTULO 2. PLATAFORMA SDR Y ENTORNO SOFTWARE

A continuación, veremos una descripción detallada de la plataforma hardware utilizada, como funciona y su entorno software.

2.1. LimeSDR

LimeSDR proporciona una plataforma hardware para desarrollar y crear prototipos de sistemas SDR utilizando el FPGA Altera Cyclone IV y el transceptor LMS7002M.

En ella podemos distinguir tres tipos de conectores: Alimentación, RF y de datos y depuración (conector USB3.0, FPGA GPIO y JTAG).

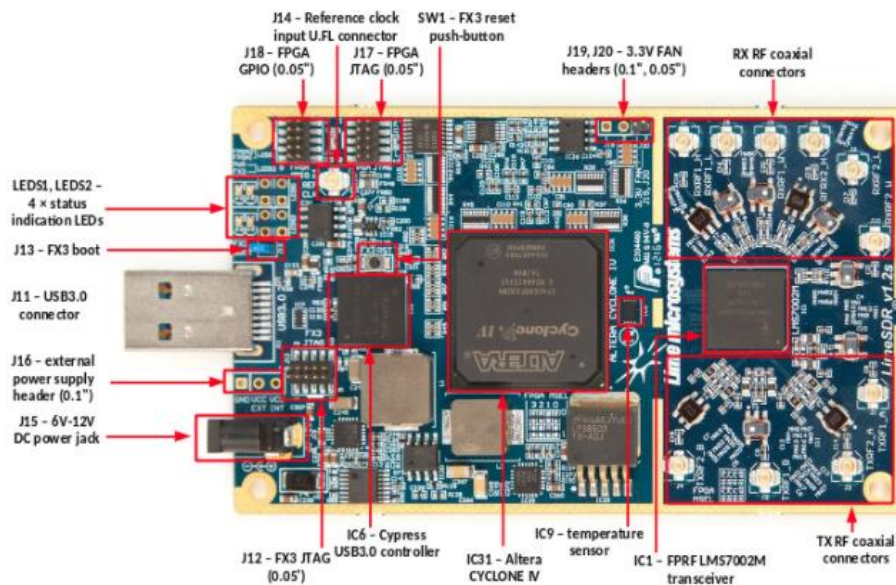


Fig. 2.1 Conectores y componentes de LimeSDR [5].

En LimeSDR encontramos ya implementados mezcladores, convertidores analógicos/digital, moduladores y demoduladores, filtros, etc. Una de las ventajas que presenta esta plataforma es que es de código abierto y todo el hardware y software está disponible para su estudio y modificación. Como dato extra, los diseños, esquemas, lista de materiales y software están disponibles bajo licencias Creative Commons y Apache 2.0.

Como ya se ha introducido en el capítulo 1, gracias a la configuración mediante software, ganamos flexibilidad a la hora de desarrollar sistemas inalámbricos.

El FPRF (explicado más adelante en el punto 2.1.4) de LimeSDR, LMS7002M, acepta flujo de datos en fase y cuadratura.

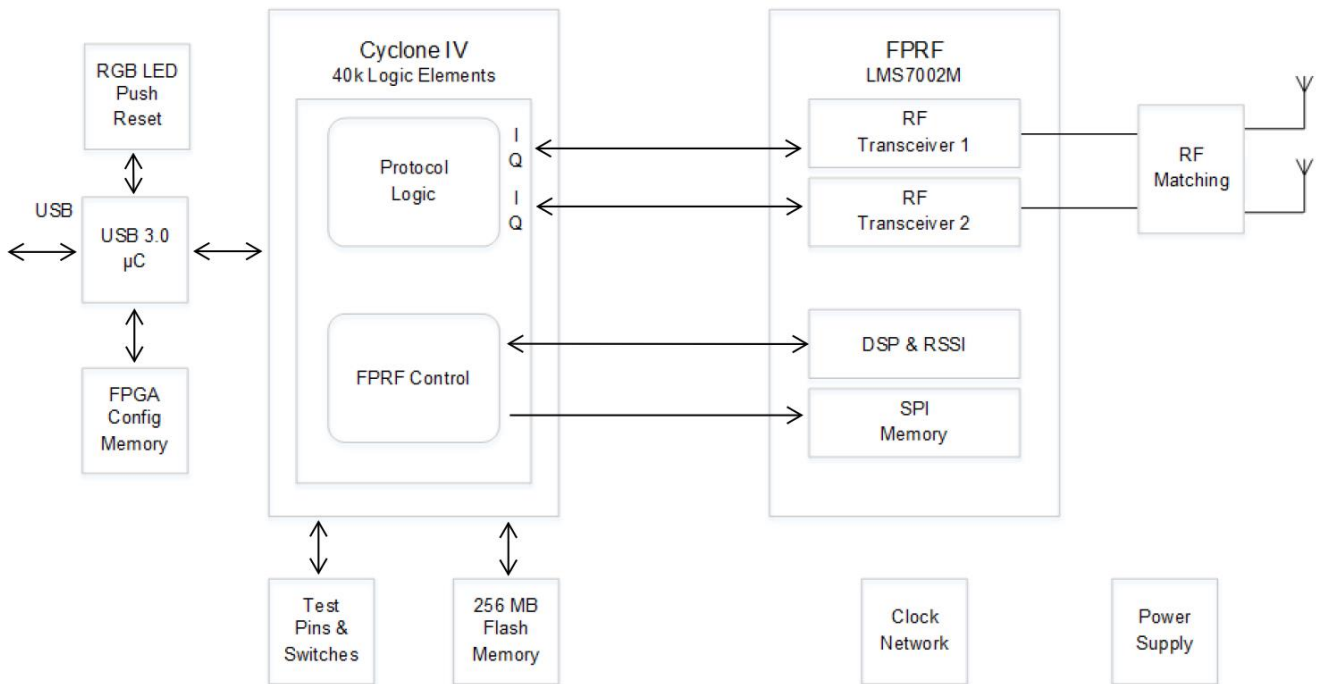


Fig. 2.2 Diagrama de bloques de LimeSDR.

LimeSDR, tiene un abanico amplio de utilidades ya que puede trabajar dentro del rango de frecuencias comprendidas entre 100 kHz y 3.8 GHz: Radio, televisión, telefonía, Wi-Fi, Bluetooth, protocolos IoT, navegación información meteorológica, aeronáutica, marina y comunicación espacial.

A continuación, se detallarán los dispositivos y las utilidades que ofrece: conexiones RF, datos y alimentación, FPGA, FPRF.

2.1.1. Conexiones RF

LimeSDR tiene 2 pares de conectores recepción/transmisión. Esto permite aprovecharlos y desarrollar un sistema MIMO con dos antenas recibiendo y dos antenas transmitiendo.

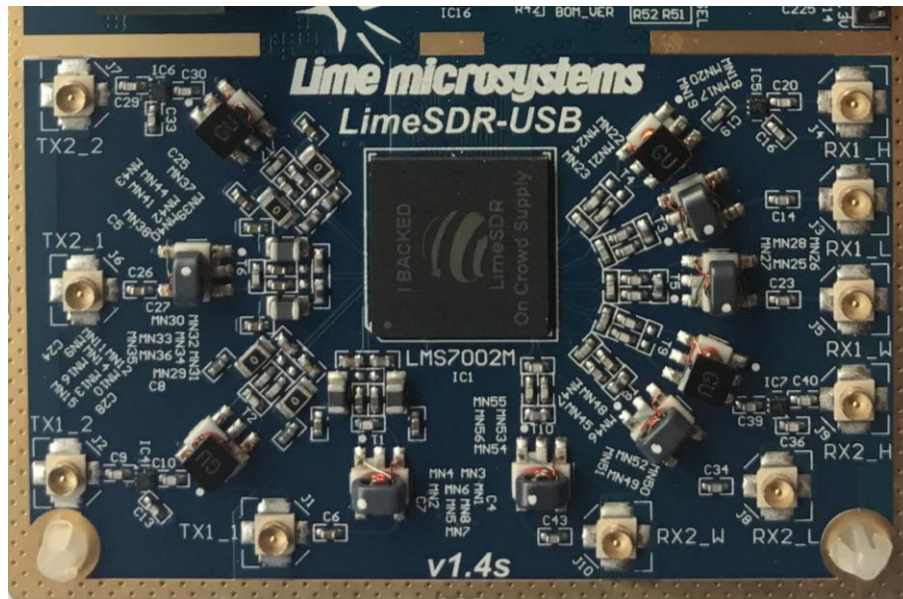


Fig. 2.3 Conexiones RF.

Observando la placa, podemos ver que conectores de RF hay 10. Esto se debe a una búsqueda de mejora en el rendimiento, por lo que, para ciertos rangos de frecuencias encontramos puertos para Transmitir/Recibir en banda baja y alta. Además, también encontramos un canal de recepción de banda ancha. A continuación, se detallarán los conectores y su perteneciente banda por defecto:

- TX1_1/ TX2_1 = Banda ancha.
- TX1_2/ TX2_2 = Banda ancha.
- RX1_1/ RX2_1 = Banda Baja (inferior a 1.5GHz).
- RX1_2/ RX2_2 = Banda Alta (superior a 1.5GHz).
- RX1_3/ RX2_3 = Banda ancha de 100kHz a 3.8GHz.

Siempre que se conozca la frecuencia con la que se va a trabajar debe utilizarse el puerto específico adecuado. Esto se debe a que, dependiendo la banda, el LNA añade una figura de ruido u otra.

Tabla 2.1.1. Figura de ruido del LNA según la banda que se trabaje.

Banda del amplificador LNA	Figura de ruido que añade
LNAL	<2dB
LNAH	<3dB
LNAW	5-7dB

Por esto, haciendo una correcta elección obtendremos un rendimiento del puerto de recepción mejor.

2.1.2. Conexión de datos y alimentación

A parte de las conexiones de RF también encontramos un puerto USB 3.0 y un conector de alimentación. Se aprovechará la alimentación mediante el puerto USB por lo que se prescindirá del conector de alimentación.

Dispone del dispositivo Cypress USB 3.0 (EZ-USB FX3) que es un controlador de dispositivos y será el encargado de detectar la LimeSDR (programado con VID/PID). Ver la Figura 4 del Anexo.

También dispone de un conector J19 el cual consiste en un conjunto de 8 GPIO genéricos que pueden ser controlados desde la API como entradas y salidas.

2.1.3. FPGA

Una FPGA es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad se puede configurar.

Nuestra plataforma utiliza Altera Cyclone IV. Esta FPGA mejora la eficiencia de procesamiento en la capa física y también disminuye la carga de procesamiento a la Odroid. Las señales muestreadas por los convertidores analógico digital serán procesadas en la FPGA como señales reales o como una única señal compleja (I-Q).

El código para el FPGA se proporciona de forma gratuita y bajo la licencia Apache 2.0.

Para más información sobre FPGA se pueden consultar las referencias [33], [34] y [35].

2.1.4. FPRF

LMS7002M es un RF IC que se puede reconfigurar sobre la marcha a cualquier frecuencia en el rango de 100kHz a 3.8GHz.

A continuación, se muestra el diagrama de bloques de LMS7002M:

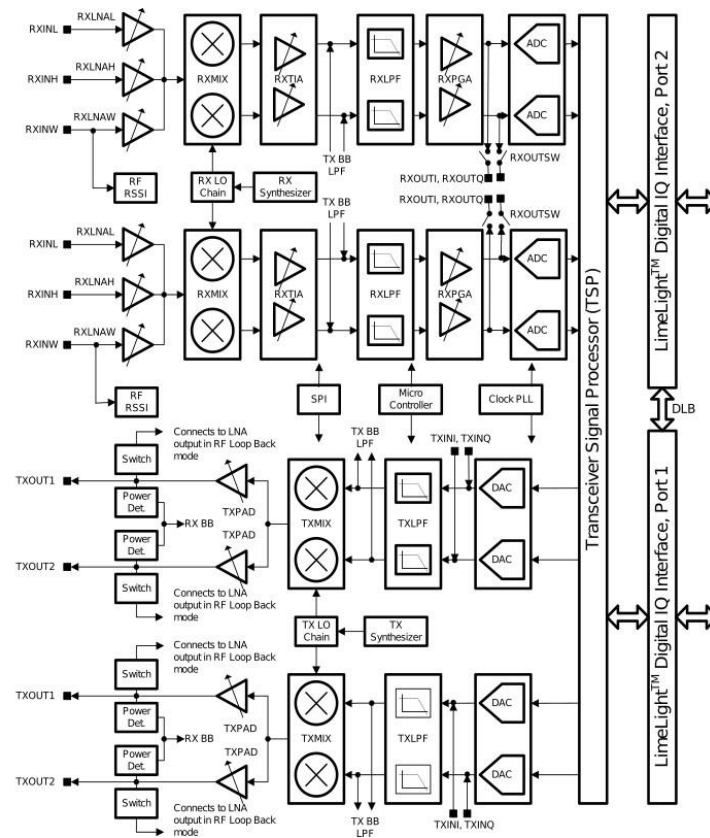


Fig. 2.4 Diagrama de bloques del transceptor LMS7002M [18].

Se puede observar que hay 3 entradas con sus correspondientes LNA y en función de la frecuencia a la que se trabaje, se elegirá una. Estos amplificadores LNA tienen la característica de amplificar la señal añadiendo el mínimo ruido. Esto se hace porque la señal recibida es débil e interesa amplificarla lo antes posible, ya que a medida que añadimos componentes estos añaden ruido y si se coloca después el amplificador también amplificaría este ruido.

El siguiente bloque en la ruta de recepción es el mezclador, la señal se mezcla con el RXPLL para obtener la señal en banda base (down converted). El RXPLL es sintetizado a partir del reloj (clock) del sistema a través de un sintetizador con el ruido de fase bajo.

Una vez convertida la señal a banda base hay un segundo amplificador de transimpedancia /LNA. Los objetivos de este amplificador son:

- Proporcionar un filtro paso bajo primario de la señal en banda base.
- Proporcionar un desplazamiento DC para evitar la saturación de la señal manteniendo el rango dinámico del ADC.
- Finalmente, una ganancia configurable para acondicionar la señal antes de filtrar.

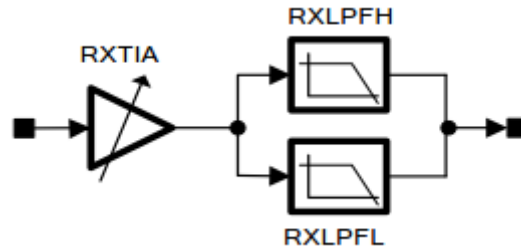


Fig. 2.5 Cadena de filtrado analógico en el receptor [18].

Cuando la señal ya ha pasado por RXTIA, pasará a un filtro configurable de banda alta o baja (RXLPFL o RXLPFH).

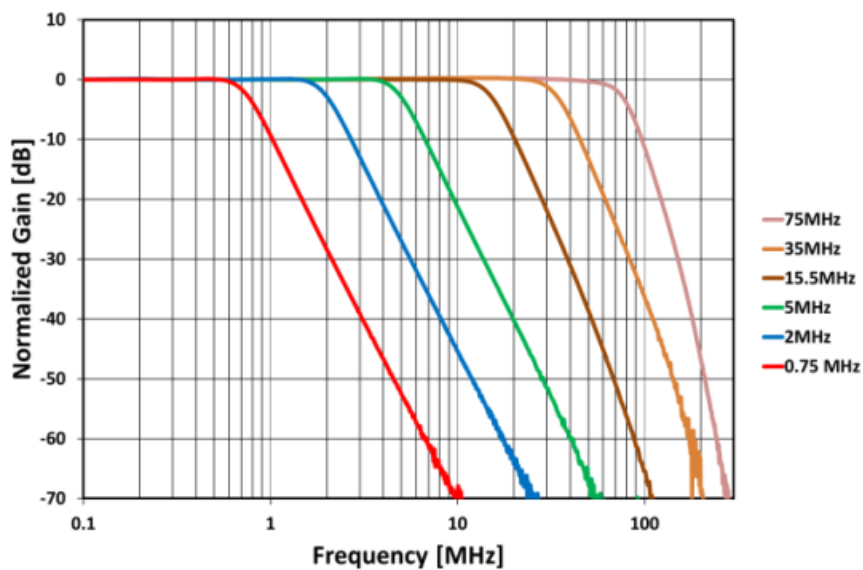


Fig. 2.6 Respuesta de amplitud analógica del RX LPF [18].

Con el objetivo de optimizar el número de bits en la etapa de cuantificación del ADC, se hace una tercera etapa. Se añade un LNA final, RXPGA. Con este amplificador de bajo ruido se conseguirá una señal correcta en el ADC, aumentando o disminuyendo la ganancia.

Para el caso de la transmisión, podemos observar que a lo que amplificadores se refiere, hay menos ya que como la señal la generamos nosotros conocemos los niveles. Solo debemos amplificar la señal en la salida para asegurarnos que allí donde esté el receptor le llegará con una calidad aceptable.

Puesto que necesitamos señales de calidad buena y que no interfieran en otras bandas, necesitamos tener el filtrado correcto. Es por esto, que en el transmisor se le da un papel importante al filtrado.

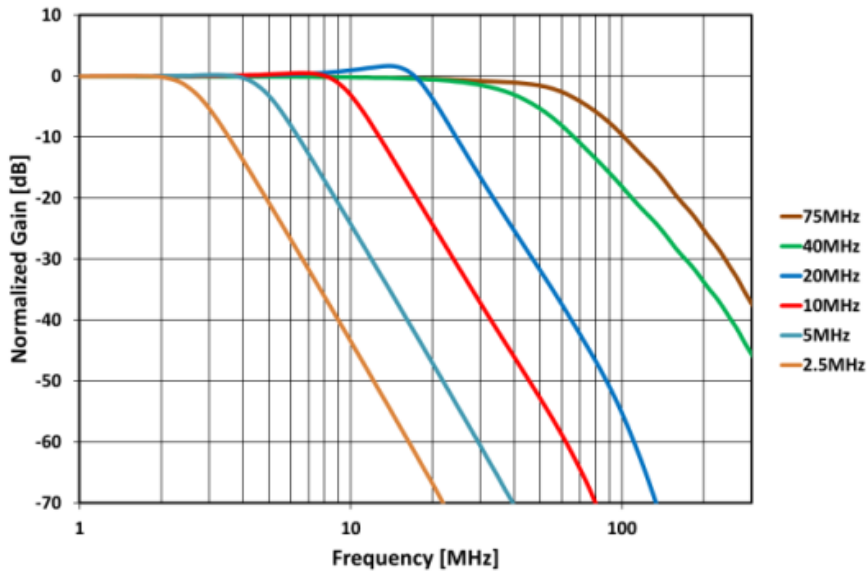


Fig. 2.7 Respuesta de amplitud analógica del TX LPF [18].

También es importante saber la potencia a la que transmitiremos, ya hay que unas normas y deben cumplirse.

2.2. LimeSuite

Podemos definir Lime Suite como una colección de software que soporta diferentes plataformas de hardware. En este trabajo se utiliza de plataforma hardware una que soporta Lime Suite, LimeSDR. También soporta controladores para el transceptor LMS7002M y varias herramientas para desarrollar hardware basado en LMS7.

Es decir, LimeSuite se encarga de unir todo el hardware basado en RFIC con el ecosistema de aplicaciones de SDR.

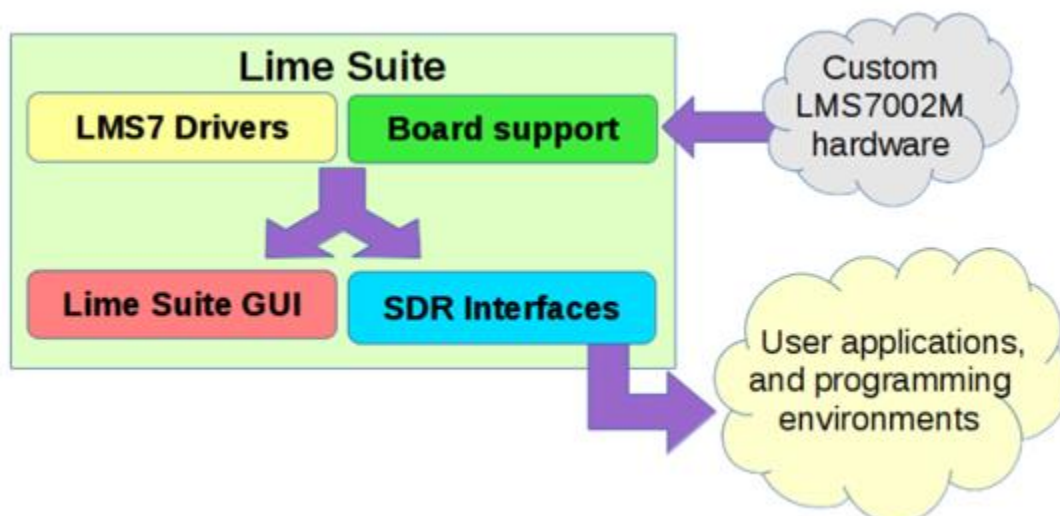


Fig. 2.8 Arquitectura de Lime Suite [15].

Como podemos ver en la figura anterior, al unir todo permite que usuarios puedan interactuar con la LimeSDR en aplicaciones SDR existentes y que se desarrollen software en una variedad de API y entornos de programación.

En LimeSDR, el software ofrece mucha flexibilidad y diversas opciones para la programación en función de cuál sea el nivel.

Por ejemplo, encontramos Lime Suite GUI. Podemos decir que es simple, intuitivo y no necesitas manipular código. Perfecto para gente que busque inicializarse.

Para usuarios más familiarizados con SDR, encontramos niveles en los que es posible la manipulación de código (se considera nivel intermedio o experto). Para estos niveles, sobre todo experto, se requiere saber bien las características del transceptor LMS7002M.

Este trabajo se podría considerar de nivel intermedio-experto. Esto se debe, básicamente, a que todo el trabajo se basa en el desarrollo de código. Esto no quita que para introducirme me familiarizara con Lime Suite GUI.

A continuación, se detallarán los bloques de LMS7 Drivers, Board support, Lime Suite GUI y SDR Interfaces.

2.2.1. LMS7 Drivers

Lime Suite ofrece para la interfaz con el transceptor LMS7002M RFIC diferentes opciones de controlador.

Estos controladores actúan como un puente entre las transacciones del bus SPI de bajo nivel y las llamadas de alto nivel. Ejemplos de estas llamadas son la sintonización, controladores de ganancia y configuración de la interfaz de datos.

Cabe destacar que Lime Suite utiliza el controlador LMS7 con el objetivo de soportar el hardware y también de que los desarrolladores creen diseños basados en el transceptor LMS7002M, ya que, este controlador LMS7, dispone de API C o C++.

2.2.2. Board Support

Lime Suite tiene varios módulos de soporte hardware, uno de ellos es el de LimeSDR.

Debido a su flexibilidad de hardware, Lime Suite proporciona todo tipo de herramientas como controladores, GUI y soporte de aplicaciones sin importar el hardware subyacente.

Tanto la interfaz como el registro de conexión es en C++. La interfaz se encarga de abstraer los detalles del hardware mientras que el registro maneja la enumeración de dispositivos disponibles.

2.2.3. Lime Suite GUI

Lime Suite proporciona LimeSuiteGUI, una interfaz gráfica para realizar de manera más intuitiva y fácil funciones como seleccionar dispositivos disponibles, depurar, visualizar el espectro de señales y actualizar firmware e imágenes de la FPGA para el dispositivo utilizado.

Como dato de interés, los archivos de configuración son extensión “.ini”, al cargarlos, se interactúa gráficamente con LMS7 usando botones, deslizadores y menús desplegables.

2.2.4. SDR Interfaces

Con el objetivo de interactuar con el hardware, Lime Suite ofrece a los usuarios muchas opciones.

Gracias a SDR Soapy podemos trabajar con API de nivel bajo y alto. Esta flexibilidad para todos los usuarios permite la integración de aplicaciones en el ecosistema SDR.

Lime Suite viene con un módulo de soporte llamado SoapyLMS7 que es el encargado de vincular la conexión de Lime Suite y la API de controladores con la librería SoapySDR. Con esto damos a entender que SoapySDR actúa como un puente entre los controladores, las API y las aplicaciones SDR.

Soapy proporciona API en los lenguajes de C, C++ y Python, acceso remoto para usar en redes locales y aplicaciones graficas SDR.

Es decir, se encarga de enlazar el controlador LimeSDR y aplicaciones SDR.

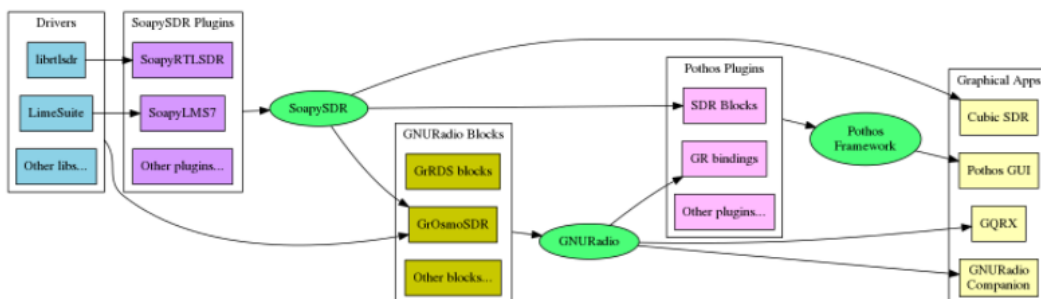


Fig. 2.9 Diagrama de bloques de la funcionalidad de Lime Suite y Soapy [8].

CAPÍTULO 3. REMOTE RADIO HEAD

Una vez vistos los dispositivos y el entorno con el que se trabaja, es hora de ver como se ha desarrollado la RRH.

En este capítulo se introducirá el concepto de RRH llegando a exponer un ejemplo teórico para la aclaración de este. También se detallará la idea, diseño, desarrollo y configuración de la RRH que se ha llevado a cabo.

3.1. Introducción a la RRH

Remote Radio Head, o en español, cabecera de radio remota, también se puede llamar en redes inalámbricas, unidad de radio remota (RRU).

Una RRH es un transceptor de radio remoto que, para el caso de telefonía móvil, se conecta a un panel de control radio del operador a través de una interfaz eléctrica o inalámbrica.

La RRH es remota a la BTS en tecnologías de sistemas inalámbricos como GSM, CDMA, UMTS o LTE. En estas tecnologías, por ejemplo, se utilizaría la RRH para extender la cobertura de una BTS en entornos rurales o bien túneles [27].

Las RRH se han convertido en subsistemas importantes. Estas contienen los circuitos de RF de la estación base más los ADC, DAC y convertidores ascendentes y descendentes.

A lo que a ventajas se refieren, facilitan la operación MIMO. Aumentan la eficiencia de una estación base y facilitan la localización física de los problemas de cobertura.

En la figura siguiente podemos ver como ejemplo el uso de RRH para estaciones base de telefonía móviles. Con lo mencionado anteriormente sacamos como conclusión que una RRH solo implementa las funcionalidades del front-end de RF y está conectada a la Unidad de Banda Base (BBU) a través de un enlace bidireccional.

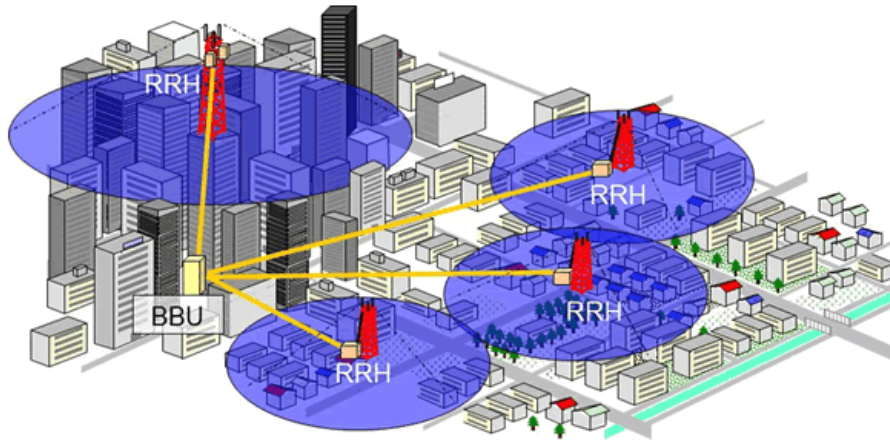


Fig. 3.1 RRH para estaciones base de telefonía móviles [28].

3.2. Diseño de la RRH

Una vez introducidos a las RRH y visto el ejemplo para estaciones base de telefonía móvil, nuestro objetivo será cumplir con las necesidades de transmitir y recibir datos vía radio desde un servidor en la nube. Queriendo cumplir con este objetivo, se ha llevado a cabo el desarrollo de la RRH.

3.2.1. Planteamiento del diseño

Para el desarrollo de la RRH se ha utilizado una Odroid XU-4 conectada a la LimeSDR.

El diseño se basa en esta Odroid que dispone de una interfaz con conexión Ethernet Gigabit con la cual se pretende conectar al servidor para el intercambio de datos mediante socket. La Odroid, a su vez, se conectará también a la LimeSDR mediante un puerto USB 3.0 y en esta tendremos las antenas (MIMO), dos transmitiendo y dos recibiendo.

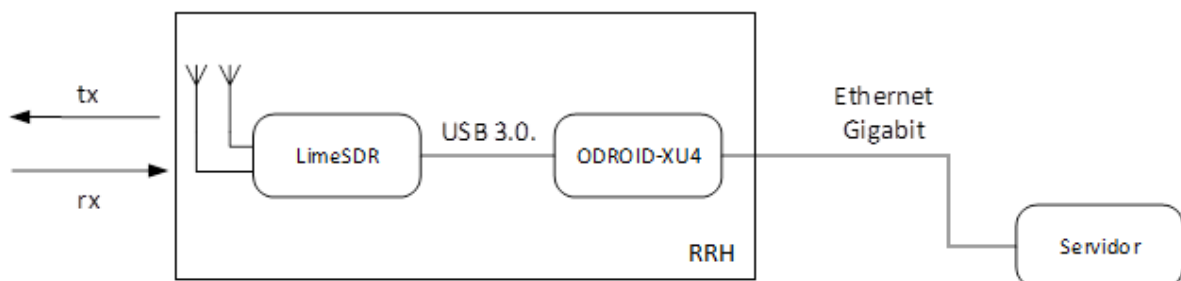


Fig. 3.2 Diagrama de bloques del diseño de la RRH.

El funcionamiento de todo este diseño será el siguiente:

Generaremos datos en el servidor ubicado en la nube, estos datos serán enviados a la Odroid mediante sockets creados por 2 VNF's, una situada en el servidor y otra en la Odroid (esto se verá detallado en el apartado 3.2.2).

Al recibir la Odroid los datos del servidor, se los pasará a la LimeSDR para que sean transmitidos.

A su vez, interesa que la LimeSDR también reciba. Así que, con el objetivo de comprobar también la cadena de transmisión, se configura la frecuencia de recepción igual que la de transmisión. Por lo que, los datos que ha enviado la LimeSDR los recibe y son enviados a la Odroid. Finalmente, esta se los envía al servidor en la nube.

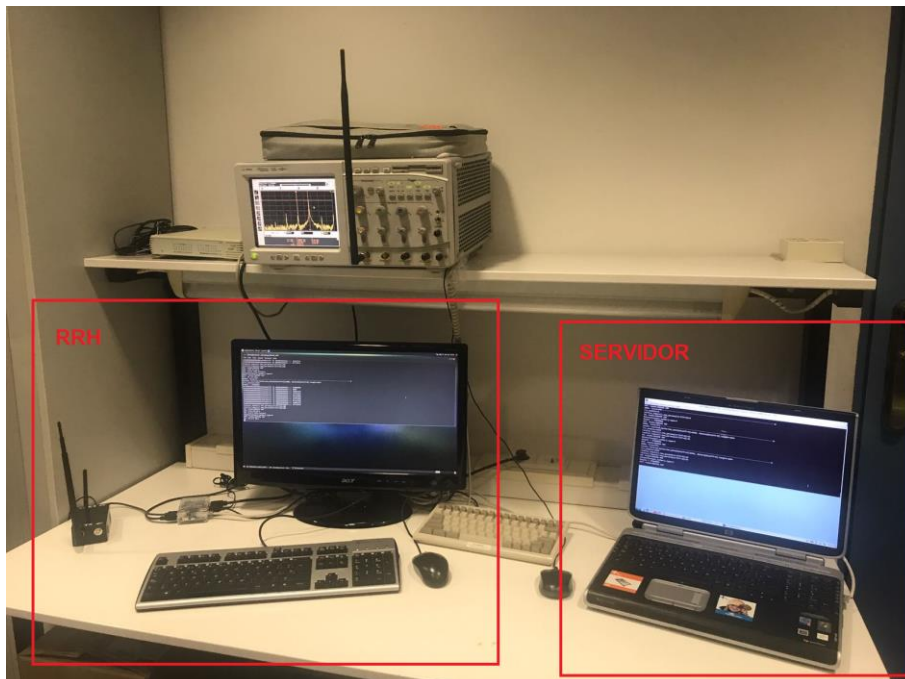


Fig. 3.3 Implementación de la RRH.

3.2.2. Comunicación entre Odroid-XU4 y servidor.

Como se ha introducido anteriormente en el funcionamiento del diseño, esta comunicación se llevará a cabo mediante sockets. Para ello se han desarrollado dos VNF.

VNF es un concepto de arquitectura de red que utiliza las tecnologías de virtualización TI para virtualizar clases enteras de funciones de nodo de red en bloques de construcción que pueden conectarse o encadenarse para crear servicios de comunicación [30].

Se han desarrollado dos VNF de tal manera que, una se colocará en el servidor de la nube y la otra en la Odroid. Estas dos VNFs serán procesos independientes y cada VNF está organizada con 4 interfaces: Entrada de datos, salida de datos, entrada de control, salida de control. Cal destacar que, si fuera necesario para mejoras en un futuro, se pueden crear tantas salidas y entradas como hicieran falta.

Servidor y Odroid estarán en la misma red con sus correspondientes IPs.

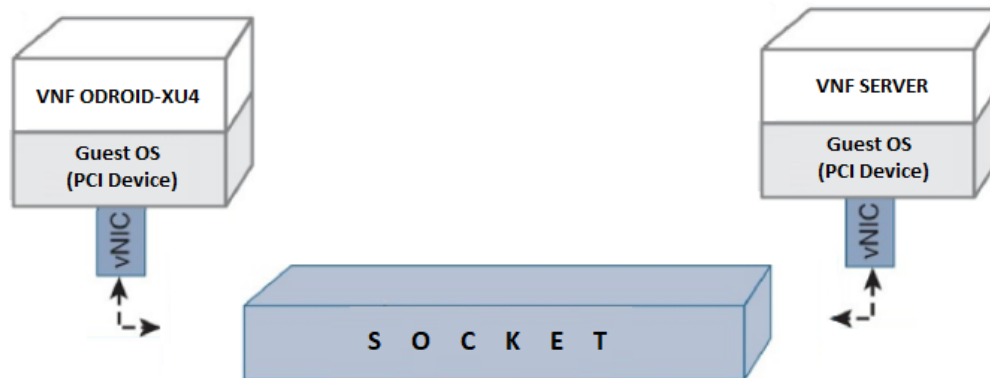


Fig. 3.4 Comunicación entre Servidor y Odroid-XU4.

Estas VNFs están compuestas por varios ficheros y la finalidad es solucionar el problema de comunicación. Al compilar el código y configurar el ejecutable, puesto que están en la misma red, se pasan las direcciones ip y puertos de entrada y salida de cada uno, creando así los sockets de transmisión y de recepción correspondientes entre ambas VNFs.

Estos sockets son UDP, por lo que no están orientados a conexión. Esto quiere decir que el programa puede abrir un socket y ponerse a enviar o recibir información sin necesidad de esperar a que alguien se conecte en el otro extremo del socket. Para más información sobre estos sockets mirar en la referencia [41].

```

-----0
0  SPECIFIC PARAMETERS SETUP: PROVAIN.Initialize()
0  SOCKETS SETUP:
0  INDATA_IP=192.168.10.3, PORT=8889, STAT=1
0  INCTRL_IP=192.168.10.3, PORT=5555, STAT=1
0  OUTDATA_IP=192.168.10.50, PORT=8888, STAT=1
0  OUTCTRL_IP=192.168.10.50, PORT=4444, STAT=1
-----0
RX Socket created: File descriptor = 3, Port = 5555, blocking flag=1
Receive Sockets Created: rxctrl[0]=3
RX Socket created: File descriptor = 4, Port = 8889, blocking flag=0
Receive Sockets Created: rxdata[0]=4
Data Out socket created!!!: IP address=192.168.10.50, PORT=4444
Transmit Sockets Created: txctrl[0]=5
Data Out socket created!!!: IP address=192.168.10.50, PORT=8888
Transmit Sockets Created: txdata[0]=6

```

Fig. 3.5 Creación de sockets en una VNF.

Corriendo las VNFs tanto en el servidor como en la Odroid, tenemos una serie de funciones las cuales son las encargadas de que se proceda al envío de datos por esos sockets. Cuando se envían estos datos se encapsula el paquete en un formato en que se le indica a la VNF receptora el número de datos, el tipo de datos que son y los parámetros de control.

En la VNF ubicada en el servidor, se generan los datos que se quieren enviar. Se encarga también de pasar parámetros de control y de pasar la configuración deseada de la LimeSDR para configurarla remotamente. Estos parámetros siguen una estructura que tanto la VNF TX como la RX tienen que reconocer y se envían por la interfaz de control.

Estas funciones mencionadas anteriormente, “traducen” estos datos que se quieren pasar a IP para así poder enviarlos a la VNF de la Odroid.

Para el proceso de recepción se hace la función inversa, en vez de “traducir” a IP los datos que quiero generar, los datos recibidos por LimeSDR son los que se “traducen”. Es decir, esta VNF de la Odroid tiene un código que es capaz de coger datos desde otro proceso para así enviarlo a la VNF del servidor mediante el socket.

La ejecución de las VNF, con IPs y puertos de entrada y salida, junto a configurar IP estática en cada dispositivo, se puede ver detallado en el ANEXO 11.

3.2.3. Hardware de nuestra RRH

Para el desarrollo de nuestra RRH, el hardware utilizado es la plataforma SDR LimeSDR (explicada en el punto y una Odroid).

Con el fin de prevenir descargas electrostáticas, ambas placas están protegidas. Odroid mediante una carcasa de plástico y LimeSDR mediante una caja de plástico negro ignifuga proporcionada por el departamento de Seguridad Eléctrica de IDNEO, la cual he amoldado a las necesidades de este trabajo.



Fig. 3.6 LimeSDR con caja de plástico negra.

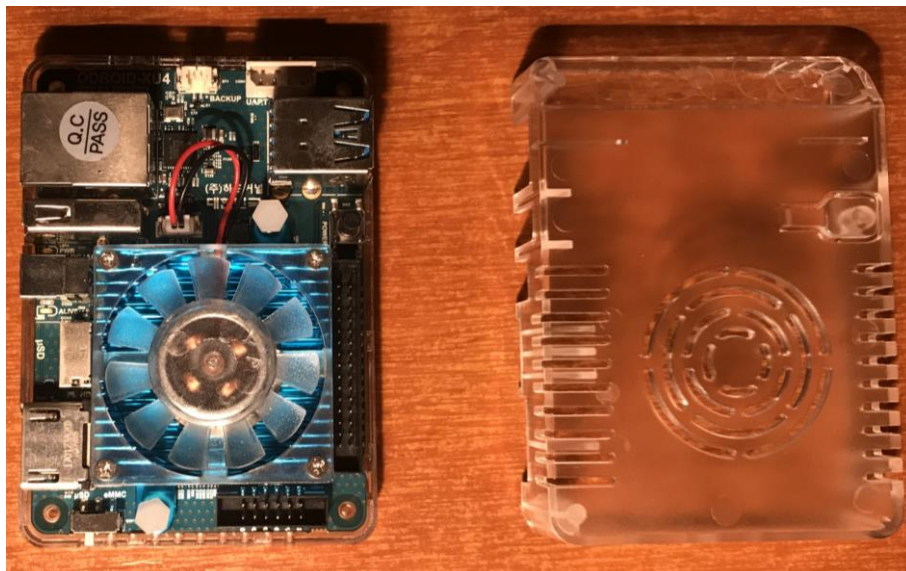


Fig. 3.7 Odroid-XU4 con carcasa de plástico.

Como no pertenece al entorno SDR, anteriormente no se ha detallado nada de la placa de desarrollo Odroid XU-4. A continuación, para saber un poco más y hacernos a la idea de cuál es su potencial, se hará una breve descripción.

Odroid-XU4 es un mini ordenador el cual dispone de un procesador Samsung Exynos 5422 de 8 núcleos con 4 cores ARM Cortex A15 y 4 ARM Cortex A7 que trabajan hasta 2GHz. También integra una GPU ARM Mali-t628 MP6 (compatible con OpenGL ES 3.0/2.0/1.1 y OpenCL 1.1), debido a esto, se ha implementado en la placa refrigeración activa. Incorpora 2Gb de RAM para complementar al procesador.

3.2.4. Software de nuestra RRH

3.2.4.1. Sistema operativo

Como todos los ordenadores, Odroid-XU4 necesita un sistema operativo. Para el desarrollo de la RRH se ha utilizado la distribución Ubuntu MATE 16.04 del sistema operativo GNU/Linux. Esto se debe a que este sistema operativo, entre otras muchas ventajas, ofrece una fácil instalación, repositorios centralizados, alta velocidad de compilación y una gran estabilidad.

Por estas ventajas y porque GNU/Linux es una plataforma de código abierto (más documentación y ayudas), es la opción más recomendable.

3.2.4.2. API

En cuanto a la funcionalidad de RRH nos basamos en una API de alto nivel. En la API hecha para la transmisión y recepción de datos encontramos una librería específica llamada LimeSuite.h y otra LMS7002M_parameters.h, ambas proporcionadas por Lime microsystems y donde hay definidas las funciones del hardware, programación FPGA de alto nivel, el hardware específico, enumeración de puertos, configuraciones, transmisiones y recepciones.

A continuación se verá la estructura de las dos VNFs que serán explicada más adelante en el apartado 3.3.6.

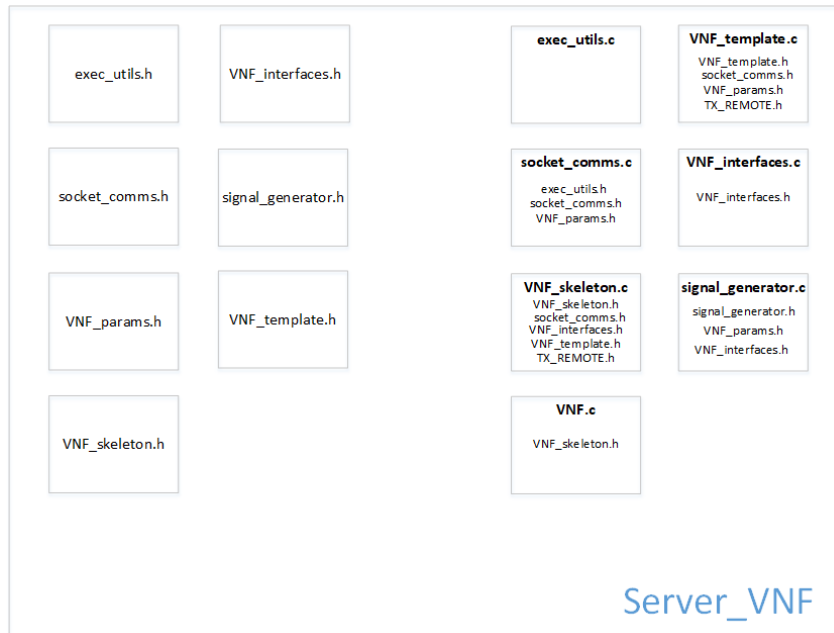


Fig. 3.8 Estructura de la VNF colocada en el servidor.

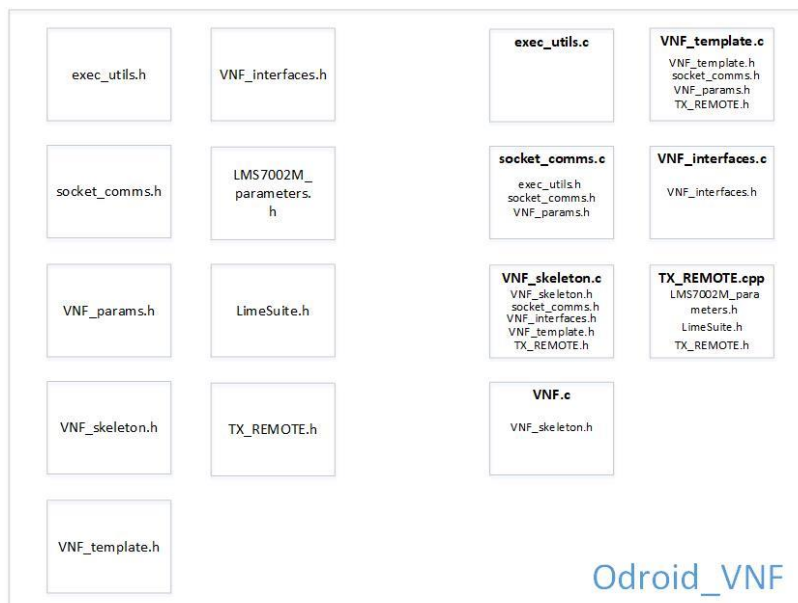


Fig. 3.9 Estructura de la VNF colocada en la Odroid-XU4.

3.2.4.3. Lenguaje de programación

Para el desarrollo de la RRH se ha utilizado las API que proporciona Soapy en los lenguajes C y C++.

También, tanto para desarrollar las 2 VNFs como el código de la LimeSDR, se han utilizado también esos mismos lenguajes de programación.

3.3. Desarrollo y programación de la RRH

Como hemos decidido desarrollar una RRH, en este apartado se explicará cómo se ha llevado a cabo.

Primero hemos configurado y analizado la LimeSDR y la Odroid-XU4. Una vez funcionaban correctamente, hemos procedido mediante código que consiste en abrir un socket, enviar y recibir información, enviar muestras del servidor a la Odroid. Información que la Odroid reenvía a la LimeSDR.

Esta descripción del desarrollo, con el objetivo de entrar más en detalle, dispondrá de referencias a anexos.

3.3.1. Sistema Operativo

Para la instalación de Ubuntu, se ha instalado en una tarjeta SD booteable persistente, por lo que se insertará la tarjeta SD en la Odroid- y este cargará el sistema operativo. Debido a la persistencia, una vez se apague, mantendrá guardado las últimas modificaciones. Ejemplos serían ficheros que se añadan o programas que se instalen. Todos los pasos que se han realizado los encontramos detallados en el ANEXO 1.

3.3.2. Instalación del software previo

Una vez funcionando el Odroid, se ha configurado correctamente la placa LimeSDR en la Odroid. Se ha realizado la configuración y la instalación de todos los paquetes y software necesarios para poder trabajar [ANEXO 2].

3.3.3. Pruebas con LimeSuiteGUI

Mediante la herramienta software LimeSuiteGUI, y gracias a archivos de configuración de ejemplo proporcionados por el Lime microsystems, se han realizado pruebas para comprobar el funcionamiento del transceptor LMS7002M [ANEXO 3].

3.3.4. Otros entornos

Aunque el objetivo del proyecto se ha realizado mediante código en lenguaje c y c++, con el propósito de familiarizarme y ampliar conocimientos, estuve también trabajando con una interfaz SDR, Pothos. [ANEXO 4].

3.3.5. Lime API

El código soporta la recepción y el envío de datos utilizando 2 canales de transmisión y 2 de recepción.

En general, este código lo que hace es conectarse al primer dispositivo LimeSDR detectado, lo configura y recibe y envía muestras.

En el ANEXO 5 se puede ver algunas de las funciones del código desarrollado.

3.3.6. VNF

El formato está diseñado siguiendo una estructura tal que nos va mostrando una fase de inicialización, una fase de ejecución, recibir datos, procesarlos y enviarlos.

Cada VNF contiene sus correspondientes archivos en su carpeta. A continuación, veremos en detalle estas carpetas y sus archivos que, en conjunto, hace que se pueda compilar y ejecutar correctamente.

Los archivos que se encargan de compilar y ejecutar son el Makefile donde está toda la configuración de compilación y el compileALL.sh para compilar y crear el ejecutable.

3.3.6.1. Server VNF

- Signal_generator.c: En este archivo están las funciones que generarán nuestra señal. En el caso de este proyecto las señales que se han probado han sido un tono y un canal de ruido.
- Signal_generator.h: Aquí vienen definidas y declaradas las variables y funciones de su correspondiente archivo c para poder llamarlas en otros archivos.
- Socket_comms.c: Encontramos las funciones que generan nuestros sockets.
- Socket_comms.h: Al igual que el otro fichero .h, este también declara variables y funciones de su correspondiente archivo c para que así puedan ser llamadas en otros archivos.
- VNF.c: Lleva la gestión de funciones de la VNF. Las funciones a las que llama son: Inicializarse, que comience a funcionar, que se pare o bien, si sucediera un error, que lo mostrara.
- VNF_interfaces.c: Se define la función que se encarga de decir por donde entran y salen los datos, así como el tipo que son.

- VNF_interfaces.h: Declara variables y funciones de su correspondiente archivo c para que así puedan ser llamadas en otros archivos.
- VNF_params.h: En este archivo encontramos los parámetros que se pasarán mediante el socket de una VNF a otra.
- VNF_skeleton.c: Están definidas las funciones que son llamadas por VNF.c.
- VNF_skeleton.h: Declara variables y funciones de su correspondiente archivo c para que así puedan ser llamadas en otros archivos.
- VNF_template.c: Están definidas las funciones que llama VNF_skeleton.c, es decir, donde se puede modificar la función de inicializar y de trabajar.
- VNF_template.h: Declara variables y funciones de su correspondiente archivo c para que así puedan ser llamadas en otros archivos.
- Exec_utils.c: Está definida la función cuando muere el proceso.
- Exec_utils.h: Declara variables y funciones de su correspondiente archivo c para que así puedan ser llamadas en otros archivos.

3.3.6.2. Odroid VNF

En esta VNF, encontramos los mismos archivos idénticos, configurados para esta VNF, menos el signal generator ya que, la Odroid no generará nada, sino que se encargará de enviar al servidor lo que recibe la LimeSDR.

Como se tiene que comunicar con LimeSDR, a diferencia de estos ya mencionados, encontramos:

- LimeSuite.h: LMS API library.
- LMS_7002M_parameters.h: librería donde están definidos los parámetros de control del transceptor LMS7002M.
- TX_REMOTE.cpp: La API creada. Streaming API permitiendo configurar frecuencia, filtros, ganancia...
- TX_REMOTE.h: Declara variables y funciones de su correspondiente archivo C++ para que así puedan ser llamadas en otros archivos. En este caso, preparado para ser entendidas por el lenguaje C y combinarse con las VNFs.

3.4. Verificación del funcionamiento y la configuración

El objetivo de este apartado consiste en la realización de varios test para la verificación del correcto funcionamiento de lo que se va desarrollando.

Se han realizado varios test específicos:

- 1) Test de la frecuencia de transmisión de la LimeSDR.
- 2) Test de la ganancia de transmisión de la LimeSDR.
- 3) Test del número de canales de la LimeSDR.
- 4) Test de la habilitación de canales de la LimeSDR.
- 5) Test del Ancho de banda de la LimeSDR.
- 6) Test del factor de sobremuestreo de la LimeSDR.
- 7) Test del LPF de la LimeSDR.
- 8) Test del funcionamiento del TX de la LimeSDR.
- 9) Test del funcionamiento del RX de la LimeSDR.
- 10) Test del funcionamiento de las VNFs.
- 11) Test del funcionamiento del TX de la RRH.

3.4.1. Test de la frecuencia de transmisión de la LimeSDR

Con este test se quiere verificar el funcionamiento de la LimeSDR. Consiste en verificar que la frecuencia de transmisión es generada correctamente por la LimeSDR en función del parámetro introducido. Este parámetro, indica la frecuencia central (portadora) deseada en Hz. La información que se quiere transmitir se enviará a la frecuencia introducida en este parámetro.

Mediante un osciloscopio que hace la FFT se determina la calidad de la señal generada a partir de su visualización tanto en tiempo como en frecuencia.

En las siguientes figuras se observa que, cambiando el valor, se transmite en una frecuencia u otra siendo con un nivel de potencia similar.



Fig. 3.10 Transmisión del tono utilizando como frecuencia central 1.0 GHz.

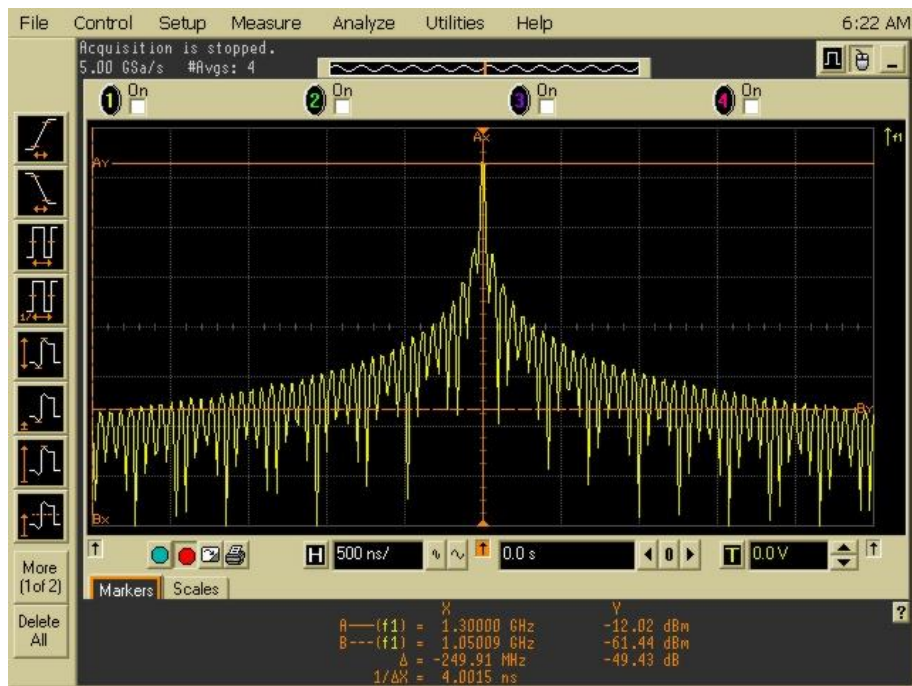


Fig. 3.11 Transmisión del tono utilizando como frecuencia central 1.3 GHz.

3.4.2. Test de la ganancia de transmisión de la LimeSDR

Este test consiste en verificar que, en función del parámetro de ganancia introducido, se transmite a diferentes niveles de potencia. Es un parámetro normalizado (rango de 0 a 1.0 donde 1.0 representa la ganancia máxima). Esto nos permitirá tener un control de la potencia transmitida.

A continuación, en las siguientes figuras, se puede observar que cambiando este valor se transmite la portadora a la frecuencia central configurada (en este caso 1.0GHz) con menor o mayor potencia.

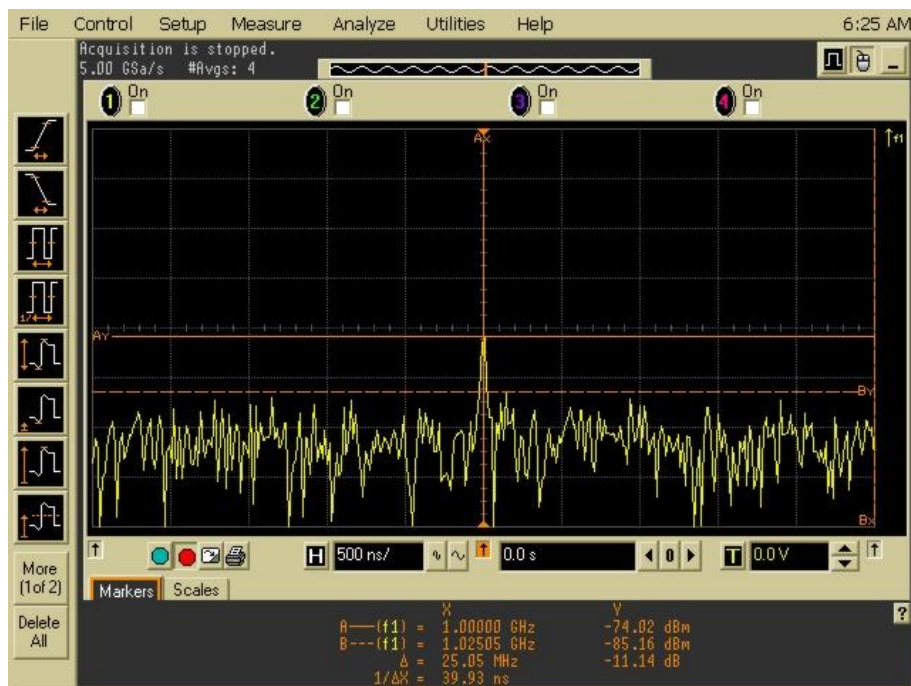


Fig. 3.12 Transmisión con una ganancia de 0.1. Potencia de -74.00 dBm.

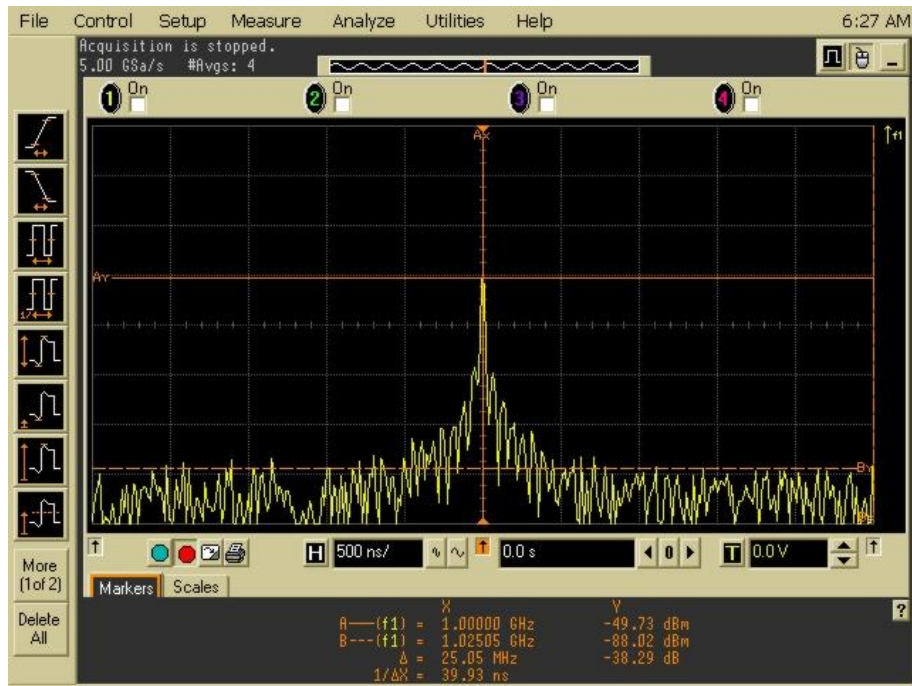


Fig. 3.13 Transmisión con una ganancia de 0.4. Potencia de -49.73 dBm.

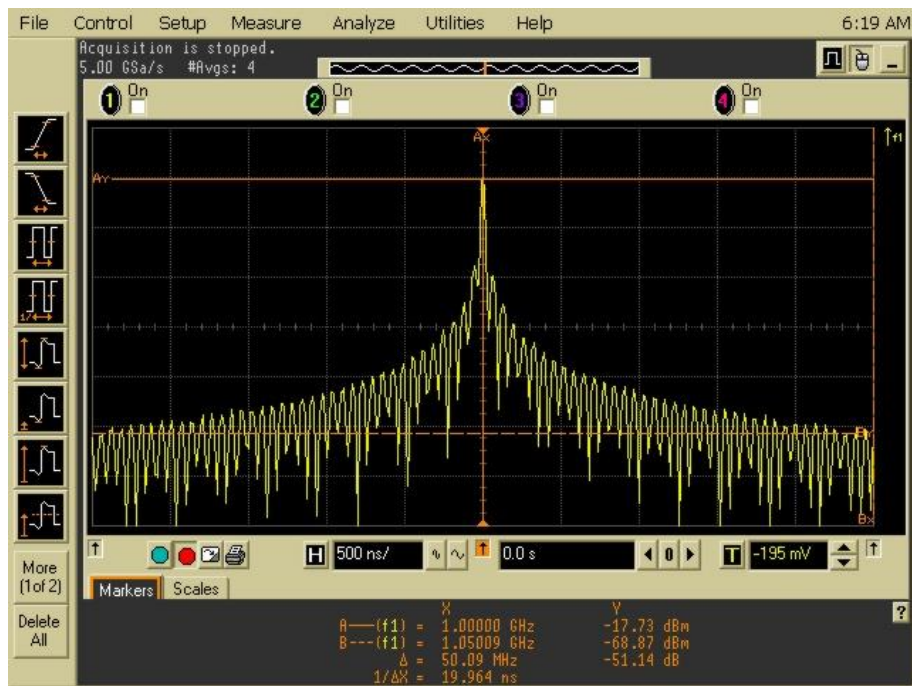


Fig. 3.14 Transmisión con una ganancia de 0.9. Potencia de -17.73 dBm.

3.4.5. Test del Ancho de banda de la LimeSDR

Este test consiste en verificar el ancho de banda configurado. Para este test se configura la LimeSDR para que transmita un canal de ruido (ver el código en ANEXO 5).

LimeSDR se refiere a la frecuencia de muestreo para indicar también el ancho de banda. La API proporcionada por el proveedor utiliza la frecuencia de muestreo configurada que le permite muestrear el ancho de banda de esa misma. Es decir, al utilizar una frecuencia de muestreo de 10 MHz le permitirá muestrear el ancho de banda de 10 MHz.

En las siguientes imágenes observaremos 2 ejemplos en los que se ha configurado la frecuencia de muestreo a 20 y 67 MHz con el objetivo de transmitir esos respectivos anchos de banda. Este ancho de banda se verifica mediante cursores horizontales.



Fig. 3.17 Transmisión de canal de ruido con una frecuencia de muestreo y ancho de banda de 60MHz.

3.4.6. Test del factor de sobremuestreo de la LimeSDR

Este test consiste en verificar el factor de sobremuestreo, el cual indicará por cuanto se multiplicará la frecuencia de muestreo para conseguir muestrear nuestra señal a una frecuencia significativamente superior. Sobremuestrear la señal permite poder trabajar más cómodamente con las especificaciones de filtros [ANEXO 7].

Se configura la LimeSDR para que transmita un canal de ruido a una frecuencia central de 1.0 GHz con una frecuencia de muestreo y un ancho de banda de 10MHz sobremuestreado con un factor 2. Con esta configuración, se encuentran alias de nuestro canal de ruido a cada 20MHz. Esto se puede observar en la siguiente figura.

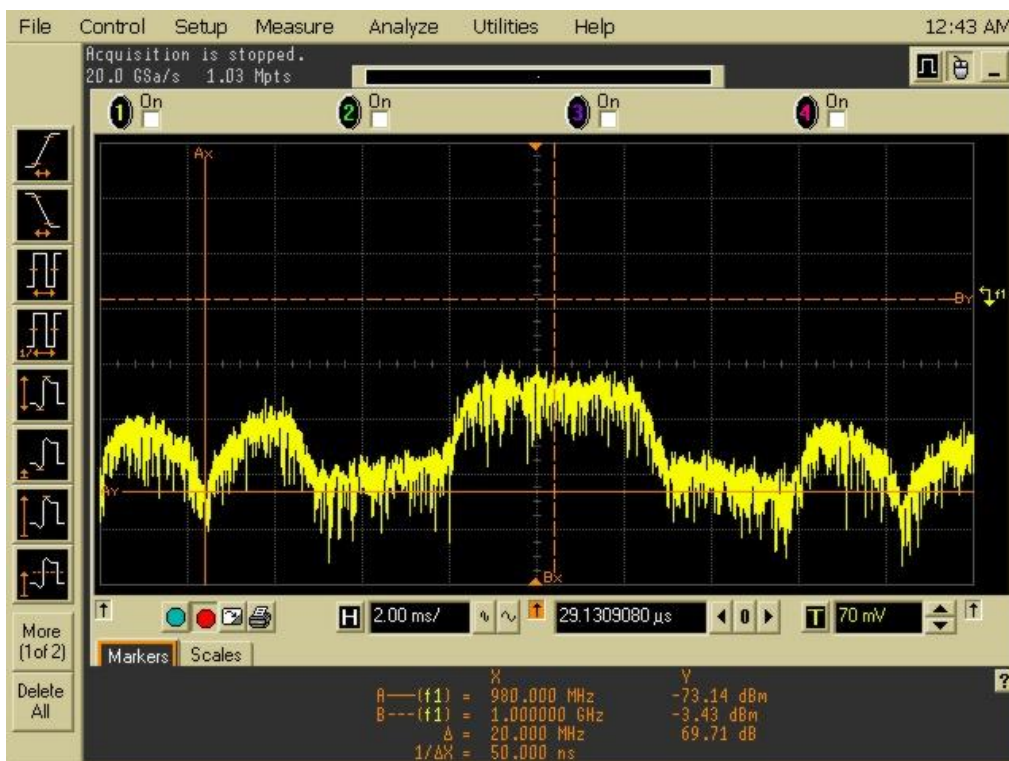


Fig. 3.18 Transmisión de un canal de ruido con una frecuencia de muestreo de 10MHz y con un factor 2 de sobremuestreo.

3.4.7. Test del LPF de la LimeSDR

Este test consiste en verificar el funcionamiento del LPF teniendo en cuenta el sobremuestreo y el teorema de Nyquist.

LMS7002M integra filtros selectivos paso bajo en las rutas de transmisión y recepción. Estos filtros, como se ha explicado en el capítulo 2, tienen una banda de paso programable para proporcionar flexibilidad y también para proporcionar rechazo de canal adyacente en la cadena de recepción.

La función de filtrado completa es una combinación de filtrado analógico y filtrado digital. Los filtros analógicos RX son ajustables desde 0.7 MHz hasta 80 MHz. Los filtros digitales proporcionan una banda de paso inferior de 0,7 MHz. El uso de dicho modo de filtrado mixto (digital y analógico) proporciona 60 dB de rendimiento antialias y 40 dB de rechazo de canal adyacente como el peor de los casos. La banda de paso de la cadena de filtrado TX se puede sintonizar de 2 MHz a 80 MHz.

Indicamos el ancho de banda que queremos de nuestro LPF programable para cumplir con el objetivo deseado.

A continuación, en las siguientes imágenes, veremos la transmisión de una señal con un ancho de banda y frecuencia de muestreo de 2 MHz con un factor 8 de sobremuestreo sin utilizar y utilizando LPF.



Fig. 3.19 Transmisión de un canal de ruido con un ancho de banda de 5MHz con un factor de sobremuestreo de 2 sin LPF.

Podemos observar que, al no utilizar LPF, tenemos alias cada 10 MHz debido al sobremuestreo.

Con el objetivo de eliminar estos alias, se configura el LPF con un ancho de banda de 5MHz.

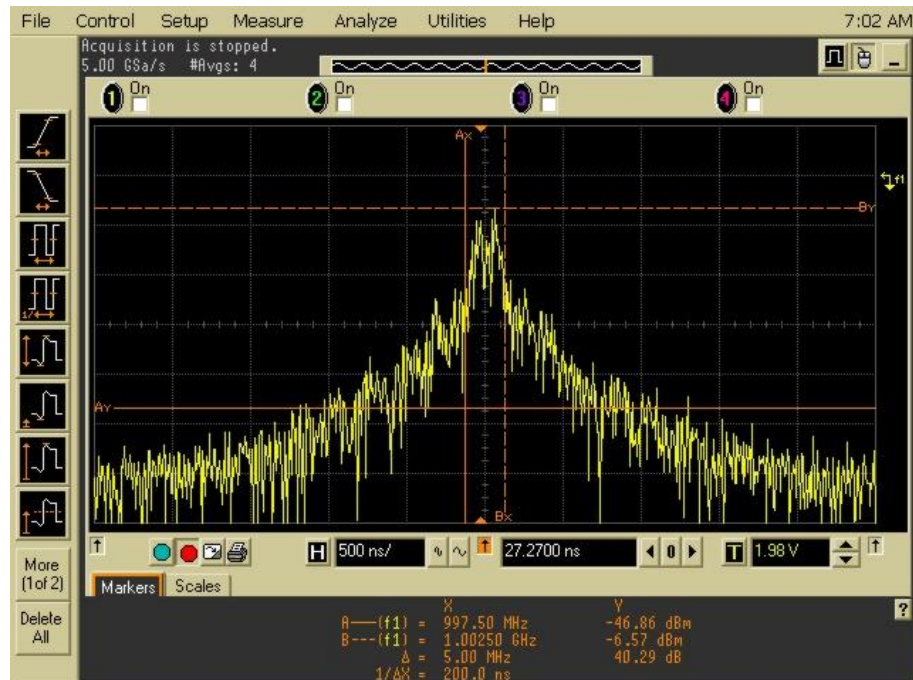


Fig. 3.20 Transmisión de un canal de ruido con un ancho de banda de 5MHz con un factor de sobremuestreo de 2 con LPF.

Podemos observar que, al configurar nuestro LPF, eliminamos los alias.

3.4.8. Test del funcionamiento del TX de la LimeSDR.

Este test consiste en generar un tono (ver el código en ANEXO 5) desde la propia Odroid y enviarlo a la LimeSDR.

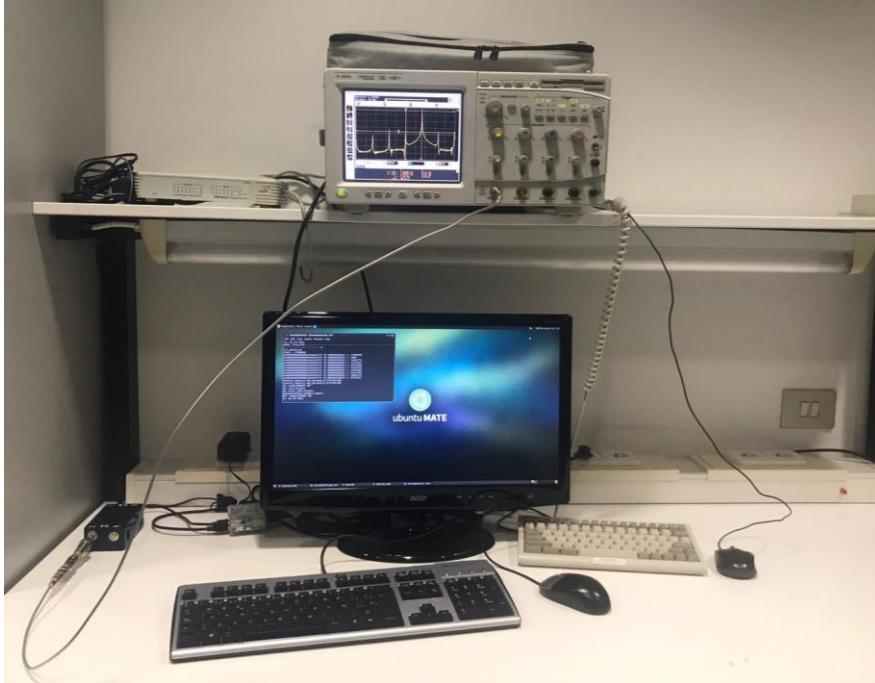


Fig. 3.21 Escenario del test del funcionamiento del TX de la LimeSDR.

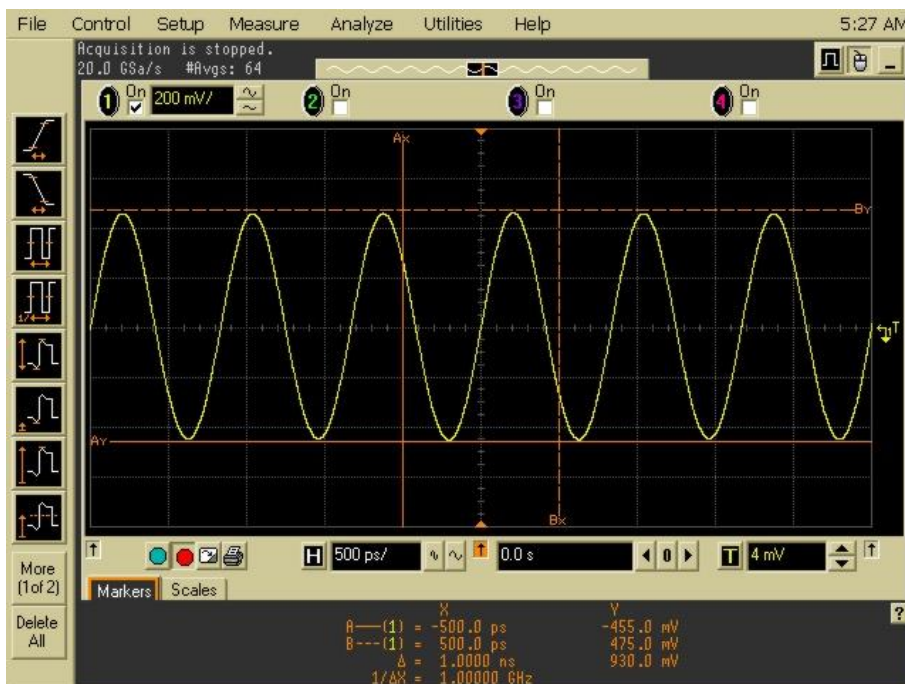


Fig. 3.22 Tono generado por LimeSDR en tiempo.

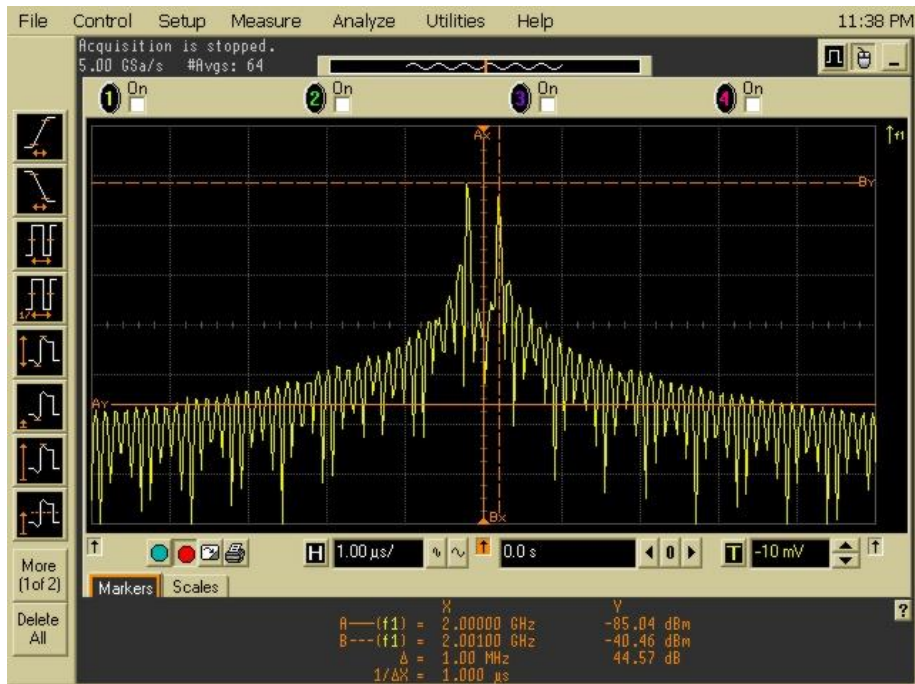


Fig. 3.23 Tono generado por LimeSDR en frecuencia.

Como cuando generamos un tono en banda base, al desplazarlo a FI, aparece su imagen, para conseguir solo tener uno en FI se enviará en los datos de la fase un seno y en los de cuadratura un coseno [ANEXO 10].

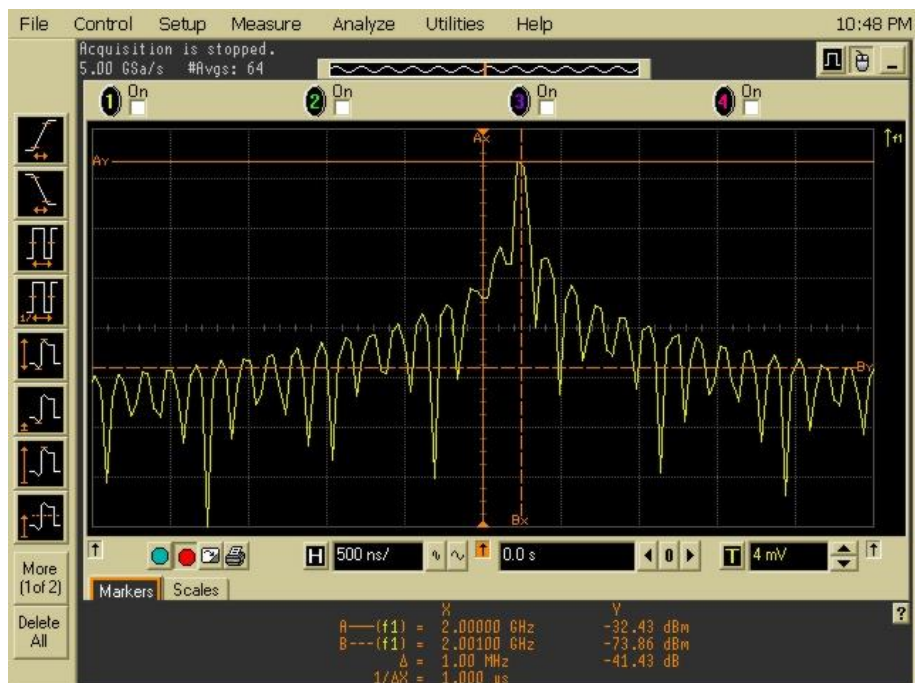


Fig. 3.24 Tono generado por LimeSDR con un coseno en fase y seno cuadratura.

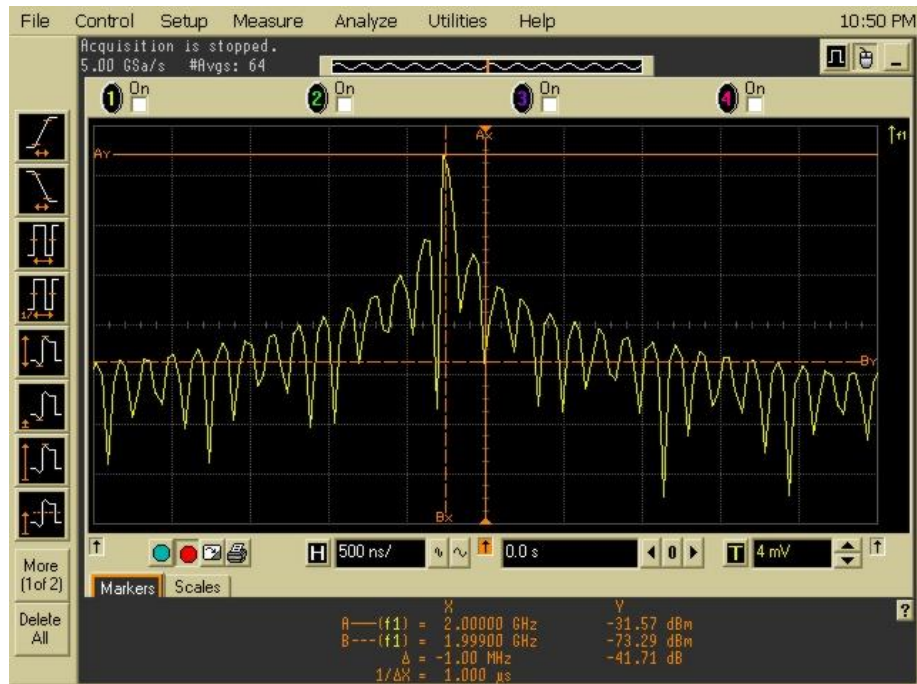


Fig. 3.25 Tono generado por LimeSDR con un seno en fase y coseno cuadratura.

3.4.9. Test del funcionamiento del RX de la LimeSDR

Este test consiste en tener funcionando el transmisor y verificar que el tono generado desde la propia Odroid y transmitido por el medio radio por una antena transmisora de la LimeSDR, es captada correctamente por la antena receptora de esta última.

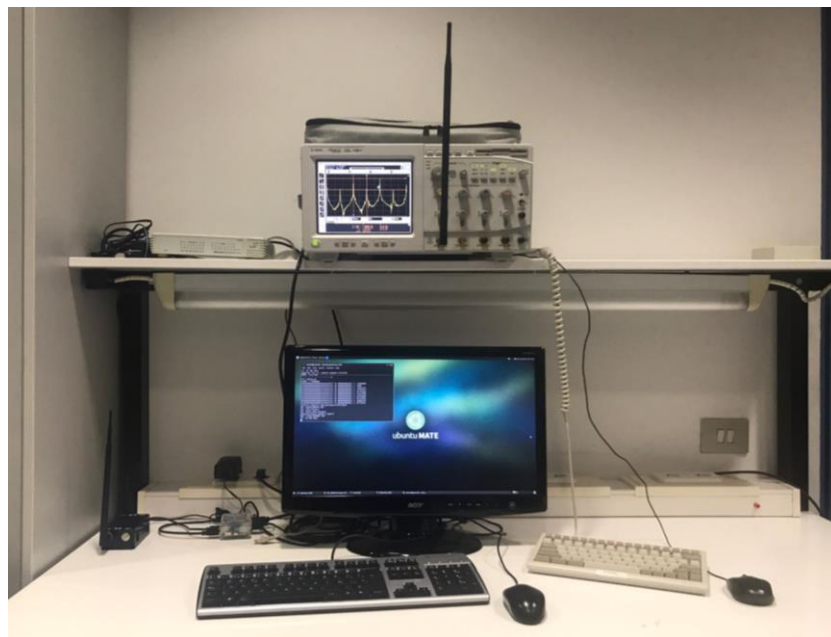


Fig. 3.26 Escenario del test del funcionamiento del RX de la LimeSDR.

A través de una función en el código [ANEXO 5], se guarda los datos que le llegan al buffer del receptor en un fichero (rxdata.dat) y se procede a su análisis mediante MATLAB.

```
Writing filename /home/odroid/LimeSuite/rxdata.dat: wrote data=356352, expected=
50000000
odroid@odroid:~/LimeSuite/tono/bin$
```

Fig. 3.27 Verificación de creación del archivo donde se ha almacenado el buffer de la LimeSDR.

Para confirmar que tanto la parte transmisora como la receptora llevan a cabo su trabajo correctamente se espera observar que la señal que tendremos en el osciloscopio sea igual o muy similar al que obtendremos tras analizar y hacer la FFT del fichero binario analizado en MATLAB.

Una vez creado el archivo binario, lo he guardado en una memoria externa y lo he pasado a mi portátil donde he abierto MATLAB y lo he importado.

Los datos de fase y cuadratura capturados sin procesar son datos de 8 bits. Cada valor de fase y cuadratura varía de 0 a 255. Para obtener del rango sin signo (0 a 255), necesitamos restar 127.5 de cada valor, por lo que tendremos como resultado un nuevo rango de -127.5 a 127.5. Entonces los datos complejos son simplemente $y = I + jQ$. Para realizar todo esto utilizando Matlab se aplicarán los comandos proporcionados a continuación:

```
>> fid = fopen('rxdata.dat', 'rb');
>> y = fread(fid,'uint8=>double');
>> y = y-127.5;
>> y = y(1:2:end) + i*y(2:2:end);
```

A continuación, elijo trazar el espectro utilizando la función de MATLAB freqz (se puede encontrar información en la referencia [38]), escribo lo siguiente en el espacio de trabajo de MATLAB:

```
>> freqz(y(1:5000),1,[-2E9:.01E9:2E9],10E9);
```

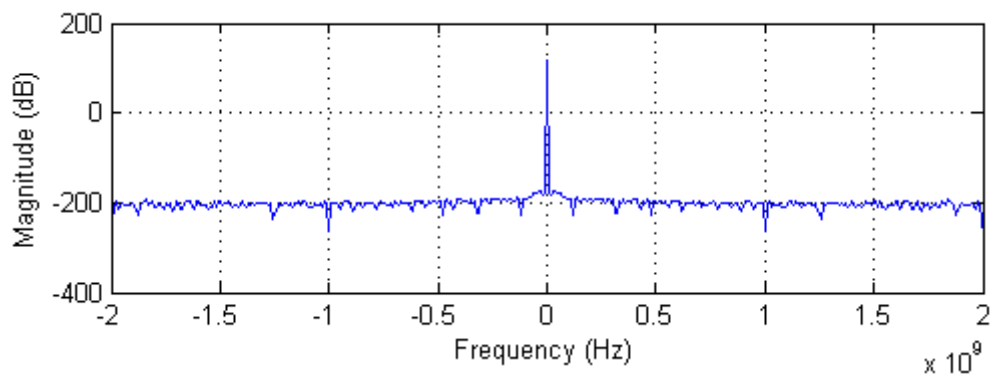


Fig. 3.28 Respuesta en frecuencia conseguida a partir del archivo.

El espectro está centrado en 0 Hz. En nuestra mente, mapeamos 0 Hz a la frecuencia central conocida (este caso, 2GHz).

Ahora, mediante la función `plot_FFT.m` [ANEXO 6] simplificamos el proceso de trazado del espectro, centrándonos en un pequeño segmento de datos, donde el eje de frecuencia se centra en la frecuencia centrada. Una línea vertical roja se dibuja a través del punto de frecuencia central.

```
>> plot_FFT(y,1,1024, 10E3,2E3);
```

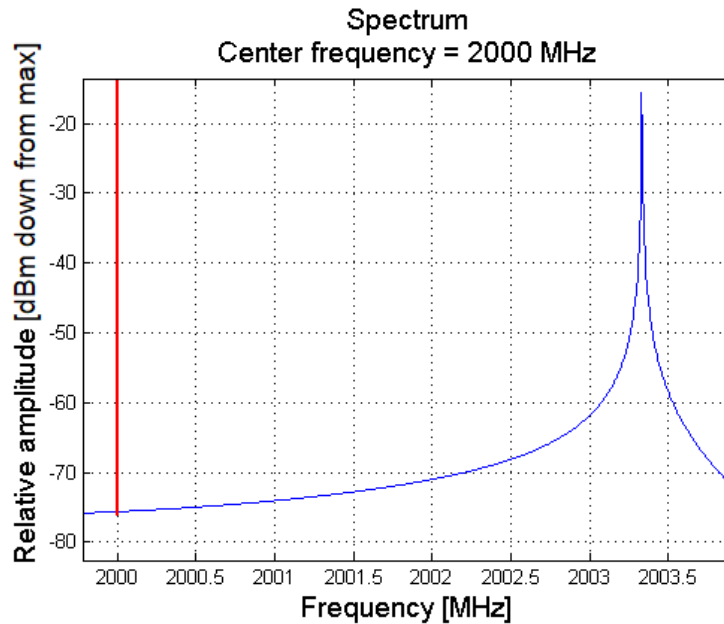


Fig. 3.29 Espectro obtenido en RX a partir de la función de MATLAB.

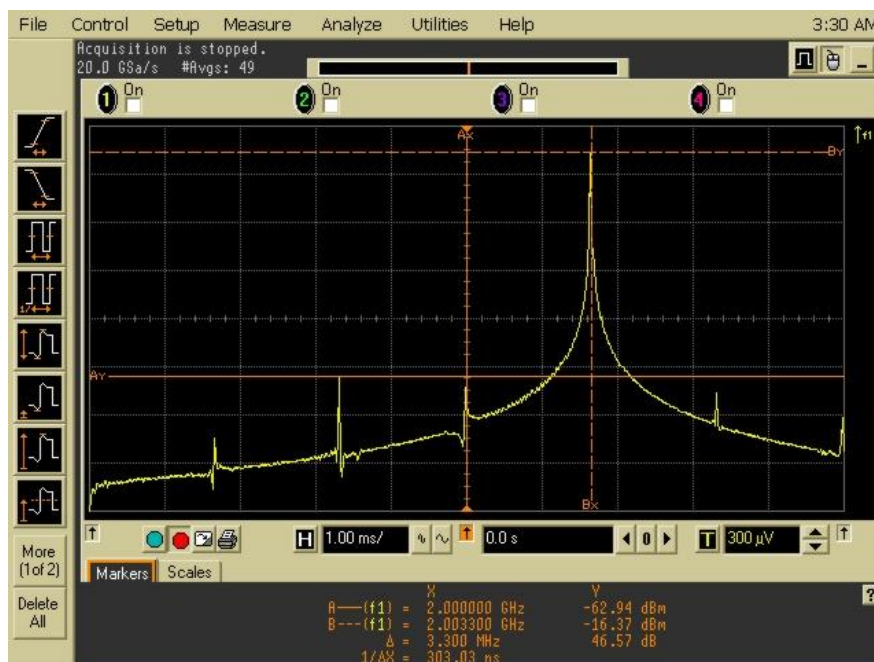


Fig. 3.30 Espectro del tono TX visualizado en el osciloscopio.

Como se esperaba, podemos observar que la señal que obtenemos en el receptor parecida a la enviada, un tono desplazado 3.33MHz.

3.4.10. Test del funcionamiento de las VNFs

Este test consiste en verificar el intercambio de datos y parámetros de las VNFs. Se configuran las dos VNFs en la Odroid para no tener que estar dependiendo de un portátil.

Para este ejemplo se configura de tal manera que las dos VNFs se pasen bloques de 1024 muestras. Estas muestras serán generadas en el signal generator.

```

36
37 int gen_toneCOMPLEX(_Complex float *func_out){
38
39     const int tx_size = 1024;
40     _Complex float *tx_buffer;
41     tx_buffer = func_out;
42     for (int i = 0; i < tx_size; i++)
43     {
44         tx_buffer[*i] = (float)cos(*M_PI*i/5.0);
45         tx_buffer[*i+] = (float)sin(*M_PI*i/5.0);
46     }
47
48     for (int j=0; j<10; j++){
49         printf("0000UUUUUUUUUUUUUUUUUUUU::: %04x\n", tx_buffer[j]);
50     }
51
52
53
54
55     return();
56 }
57
58
59
60
61 float seq_power(int length, _Complex float *incplx){
62
63     int i;
64     float ave_power=0.0;
65
66     for (i=0; i<length; i++){
67         ave_power += *(incplx+i)*conjf(*(incplx+i));
68     }
69     ave_power=ave_power/(float)length;
70     return(ave_power);
71 }

```

```

odroid@odroid: ~/Desktop/18RRH/Server_VNF
File Edit View Search Terminal Help
OUT: mname=ODROID VNF
WORK; Tslot=73-----0
rcv_samples=0
receive_inputs().new_params[ctrlIIF=0]=0
IN: mname=SERVER VNF
0000UUUUUUUUUUUUUUUUUUUU::: 3f800000
0000UUUUUUUUUUUUUUUUUUUU::: 0000
0000UUUUUUUUUUUUUUUUUUUU::: 3e9e377a
0000UUUUUUUUUUUUUUUUUUUU::: 3f737871
0000UUUUUUUUUUUUUUUUUUUU::: bf4f1bbd
0000UUUUUUUUUUUUUUUUUUUU::: 3f167918
0000UUUUUUUUUUUUUUUUUUUU::: bf4f1bbd
0000UUUUUUUUUUUUUUUUUUUU::: bf167918
0000UUUUUUUUUUUUUUUUUUUU::: 3e9e377a
0000UUUUUUUUUUUUUUUUUUUU::: bf737871
dataout[output_number].type=4
OUT: mname=ODROID VNF
WORK; Tslot=74-----0
rcv_samples=0
receive_inputs().new_params[ctrlIIF=0]=0
IN: mname=SERVER VNF
0000UUUUUUUUUUUUUUUUUUUU::: 3f800000
0000UUUUUUUUUUUUUUUUUUUU::: 0000
0000UUUUUUUUUUUUUUUUUUUU::: 3e9e377a
0000UUUUUUUUUUUUUUUUUUUU::: 3f737871
0000UUUUUUUUUUUUUUUUUUUU::: bf4f1bbd
0000UUUUUUUUUUUUUUUUUUUU::: 3f167918
0000UUUUUUUUUUUUUUUUUUUU::: bf4f1bbd
0000UUUUUUUUUUUUUUUUUUUU::: bf167918
0000UUUUUUUUUUUUUUUUUUUU::: 3e9e377a
0000UUUUUUUUUUUUUUUUUUUU::: bf737871
dataout[output_number].type=4

```

Fig. 3.31 Valor en hexadecimal de las 10 primeras muestras generadas.

Una vez tenemos identificadas las muestras, procedemos a enviarlas. Por lo que tendremos localizada el valor de la primera muestra.

```

109 // PROCESS DATA
110 // gen_cosinus_REAL_COMPLEX(output0, tableA, TABLESZ, samplelength, reffreq, tonefreq);
111 gen_toneCOMPLEX(output0);
112 // if(rcv_samples>0)channel_noise(input0, output0, rcv_samples, oVNF_INparams);
113 // snd_samples0=oVNF_OUTparams.block_length;
114
115 // OUTPUT
116 // Set_output_samples(dataIIF_number, snd_samples); ADD such line for each output
117 set_output_samples(0, snd_samples); // Send data samples.
118
119
120 printf("Output = %04x\n", *(output0+0));
121 // myjcllength=write_file(myjclfilename, set_output_samples, FILEMAXLENGTH);
122 // sprintf(oVNF_OUTparams.mname, "%d", (int)Tslot);
123 // printf("OUT: mname=%s\n", oVNF_OUTparams.mname);

```

```

odroid@odroid: ~/Desktop/18RRH/Server_VNF
File Edit View Search Terminal Help
rcv_samples=0
receive_inputs().new_params[ctrlIIF=0]=0
IN: mname=SERVER VNF
dataout[output_number].type=4
Output = 3f800000
OUT: mname=ODROID VNF
WORK; Tslot=1-----0
rcv_samples=0
receive_inputs().new_params[ctrlIIF=0]=0
IN: mname=SERVER VNF
dataout[output_number].type=4
Output = 3f800000
OUT: mname=ODROID VNF

```

Fig. 3.32 Valor en hexadecimal de la primera muestra enviada.

Verificamos que la muestra que enviamos sea la que recibimos. De esta manera nos aseguramos de que el intercambio de datos es correcto.

```

133     printf("-----\n");
134     printf("WARNING NOT INITIALIZED!\n");
135     printf("PLEASE CHECK YOUR ODROID USB PORT AND SEND LIME PARAMS\n");
136     printf("-----\n");
137 }
138
139 #ifdef VNF
140 int main()
141 {
142     int VNF_work();
143
144     int snd_samples=0;
145     rcv_samples=0;
146
147     _Complex float *input0=input_ptr(); //Pointer to data for input if0
148     _Complex float *output0=output_ptr(); //Pointer to data for output if0
149
150     printf("WORK; Tslot=0\n");
151
152     // INPUT
153     // Capture received data samples if0. Do similar for all active data if's
154     rcv_samples=get_input_samples();
155     printf("rcv_samples=%d\n", rcv_samples);
156     printf("Input = %04x\n", *(input0+));
157     //for (int i = 0; i < input0; ++i)
158     //    printf("Long input = %d\n", input0[i]);
159 }
160
161 //myfilelength2=write_file(filename, get_input_samples, FILEMAXLENGTH);
162
163 //send this data to odroid
164 int lengthArray = 10;
165 _Complex float new_array [lengthArray];
166 int pos=0;
167 while (pos<lengthArray){
168     new_array[pos] = input0[pos];
169     pos++;
170 }
171 for (int j=0; j<lengthArray; j++){
172     printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYYYY TO ODDRO0000ID::: %04x\n", new_array[j]);
173 }
174
175 }
176
177 }
178
179 }
    
```

Fig. 3.33 Verificación de que el valor en hexadecimal de la primera recibida coincide con la enviada.

Aunque los datos llegaban correctamente, al pasárselo con un puntero, este excedía el tamaño para enviárselo a la LimeSDR por lo que salía segmentation fault. Como solución se ha optado a guardar los valores del puntero en un array limitándolo a un tamaño. De esta forma como se impone el tamaño no habrá problemas de limitaciones.

```

148     _Complex float *output0=output_ptr(); //Pointer to data for output if0
149     printf("WORK; Tslot=0\n");
150
151     // INPUT
152     // Capture received data samples if0. Do similar for all active data if's
153     rcv_samples=get_input_samples();
154     printf("rcv_samples=%d\n", rcv_samples);
155     printf("Input = %04x\n", *(input0+));
156     printf("Input = %04x\n", *(input0+));
157     //for (int i = 0; i < input0; ++i)
158     //    printf("Long input = %d\n", input0[i]);
159 }
160
161 //myfilelength2=write_file(filename, get_input_samples, FILEMAXLENGTH);
162
163 //send this data to odroid
164 int lengthArray = 10;
165 _Complex float new_array [lengthArray];
166 int pos=0;
167 while (pos<lengthArray){
168     new_array[pos] = input0[pos];
169     pos++;
170 }
171 for (int j=0; j<lengthArray; j++){
172     printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYYYY TO ODDRO0000ID::: %04x\n", new_array[j]);
173 }
174
175 }
176
177 }
178
179 }
    
```

Fig. 3.34 Verificación de los valores en hexadecimal guardados en el array coinciden con los generados.

Para finalizar con este test, se verifica que tanto en una dirección como en otra recibe y se envían datos.

```

odroid@odroid: ~/Desktop/RRH/Odroid_VNF
File Edit View Search Terminal Help
-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=100
IN: mname=ODROID VNF
IN: Lime_Param=1
IN: Start_TXRX_Param=1
dataout[output_number].type=4
OUT: mname=SERVER VNF
WORK; Tslot=46
-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=100
receive_inputs().new_params[ctrlITF=0]=100
IN: mname=ODROID VNF
IN: Lime_Param=1
IN: Start_TXRX_Param=1
dataout[output_number].type=4
OUT: mname=SERVER VNF
WORK; Tslot=47
-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=100
receive_inputs().new_params[ctrlITF=0]=100
IN: mname=ODROID VNF
IN: Lime_Param=1
IN: Start_TXRX_Param=1

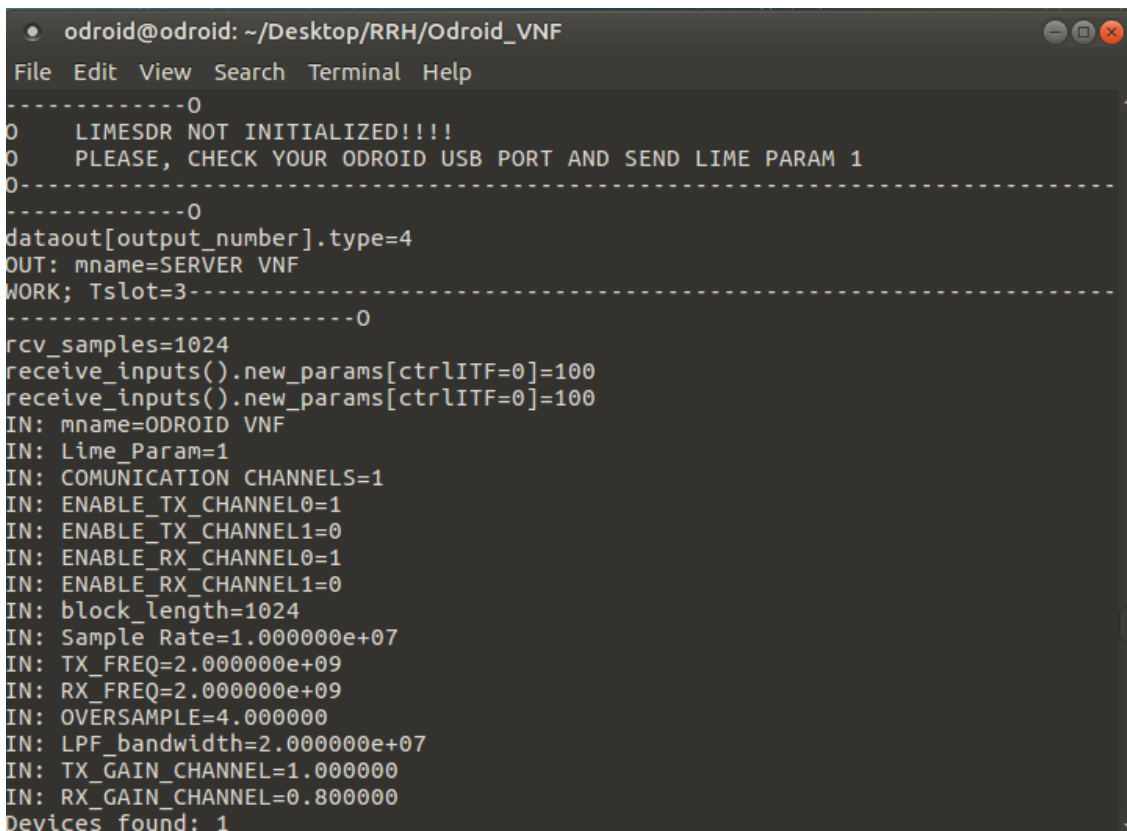
odroid@odroid: ~/Desktop/RRH/Server_VNF
File Edit View Search Terminal Help
dataout[output_number].type=4
OUT: mname=ODROID VNF
WORK; Tslot=45
-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=36
receive_inputs().new_params[ctrlITF=0]=36
IN: mname=SERVER VNF
dataout[output_number].type=4
OUT: mname=ODROID VNF
WORK; Tslot=46
-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=36
receive_inputs().new_params[ctrlITF=0]=36
IN: mname=SERVER VNF
dataout[output_number].type=4
OUT: mname=ODROID VNF
WORK; Tslot=47
-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=36
receive_inputs().new_params[ctrlITF=0]=36
IN: mname=SERVER VNF
dataout[output_number].type=4
OUT: mname=ODROID VNF
WORK; Tslot=48
    
```

Fig. 3.35 Verificación del envío de 1024 y parámetros de control por parte de ambas VNFs.

3.4.11. Test del funcionamiento del TX de la RRH

Este test consiste en generar una señal en la VNF ubicada en el servidor, enviarla a la VNF de la Odroid con sus correspondientes parámetros de configuración y que esta se lo pase a la LimeSDR para que proceda a transmitir.

A continuación, en la siguiente figura, se puede observar como la Odroid-XU4 está con unos parámetros iniciales en los que la LimeSDR está conectada, pero en standby y que, al pasarle los parámetros desde el servidor, estos llegan correctamente y se establece la configuración definida.



```
odroid@odroid: ~/Desktop/RRH/Odroid_VNF
File Edit View Search Terminal Help
-----0
0 LIMESDR NOT INITIALIZED!!!!
0 PLEASE, CHECK YOUR ODROID USB PORT AND SEND LIME PARAM 1
0-----0
-----0
dataout[output_number].type=4
OUT: mname=SERVER VNF
WORK; Tslot=3-----0
-----0
rcv_samples=1024
receive_inputs().new_params[ctrlITF=0]=100
receive_inputs().new_params[ctrlITF=0]=100
IN: mname=ODROID VNF
IN: Lime_Param=1
IN: COMMUNICATION CHANNELS=1
IN: ENABLE_TX_CHANNEL0=1
IN: ENABLE_TX_CHANNEL1=0
IN: ENABLE_RX_CHANNEL0=1
IN: ENABLE_RX_CHANNEL1=0
IN: block_length=1024
IN: Sample Rate=1.000000e+07
IN: TX_FREQ=2.000000e+09
IN: RX_FREQ=2.000000e+09
IN: OVERSAMPLE=4.000000
IN: LPF_bandwidth=2.000000e+07
IN: TX_GAIN_CHANNEL=1.000000
IN: RX_GAIN_CHANNEL=0.800000
Devices found: 1
```

Fig. 3.36 Configuración de la RRH desde el servidor.

Una vez configurada, el siguiente paso que realiza es el envío de los datos. En este caso, como podemos ver en el `block_length` de la figura anterior, le enviamos el tono en bloques de 1024 muestras.

```
lab138@HP0Lab138: ~/DADES/REMOTE RADIO HEAD/23RRH/Server_VNF
File Edit View Search Terminal Help
dataout[output_number].type=4
Output = 3f800000
OUT: mname=ODROID VNF
WORK; Tslot=90-----0
rcv_samples=0
receive_inputs().new_params[ctrlITF=0]=36
receive_inputs().new_params[ctrlITF=0]=36
IN: mname=SERVER VNF
0000UUUUUUUUUUUUUUUUUUUU::: 3f800000
0000UUUUUUUUUUUUUUUUUUUU::: 0000
0000UUUUUUUUUUUUUUUUUUUU::: 3e9e377a
0000UUUUUUUUUUUUUUUUUUUU::: 3f737871
0000UUUUUUUUUUUUUUUUUUUU::: bf4f1bbd
0000UUUUUUUUUUUUUUUUUUUU::: 3f167918
0000UUUUUUUUUUUUUUUUUUUU::: bf4f1bbd
0000UUUUUUUUUUUUUUUUUUUU::: bf167918
0000UUUUUUUUUUUUUUUUUUUU::: 3e9e377a
0000UUUUUUUUUUUUUUUUUUUU::: bf737871
```

Fig. 3.37 Envío de datos del servidor a Odroid.

```
odroid@odroid: ~/Desktop/Odroid_VNF
File Edit View Search Terminal Help
Tx: 39.944 MB/s
WORK; Tslot=10-----0
socket_received_bytes=new_data[dataITF=0]=8200, datain[dataITF=0].length=1024
rcv_samples=1024
Input = 3f800000
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: 3f800000
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: 0000
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: 3e9e377a
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: 3f737871
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: bf4f1bbd
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: 3f167918
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: bf4f1bbd
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: bf167918
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: 3e9e377a
AAARRRRRRRRRRRRRAAAAAAAYYYY TO ODDR00000ID::: bf737871
```

Fig. 3.38 Recepción de datos del Odroid.

En las imágenes verificamos que se envían bloques de 1024 muestras desde el servidor y son recibidos por la Odroid que, a su vez, la Odroid los envía a la LimeSDR y esta los transmite por el medio radio.

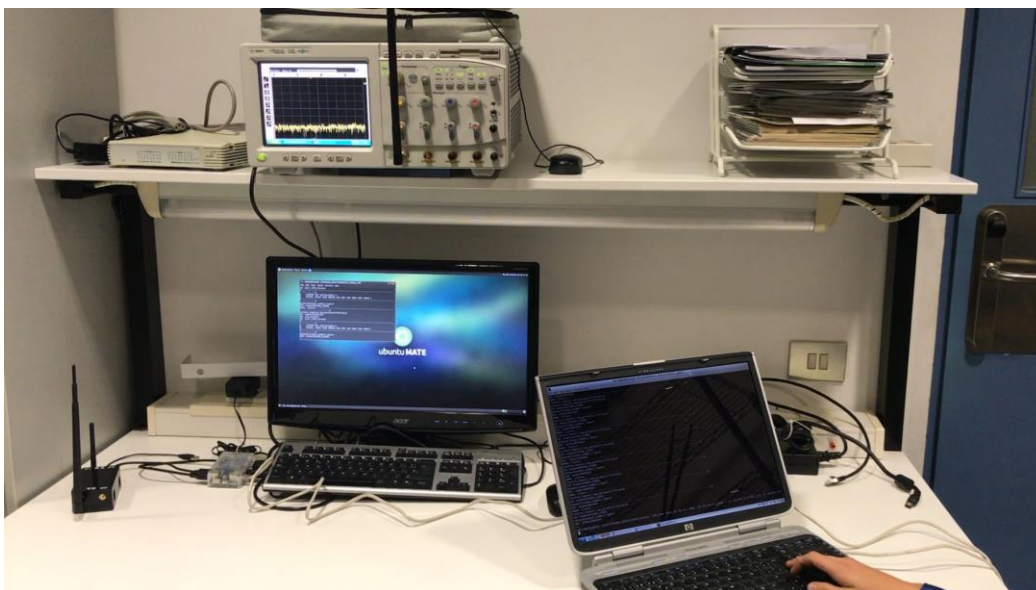


Fig. 3.39 RRH en standby esperando datos desde el servidor.



Fig. 3.40 RRH TX.

CAPÍTULO 4. PLANIFICACIÓN DEL TRABAJO

En este último capítulo se quiere reflejar como se ha realizado el conjunto de tareas que he llevado a cabo, el orden que he seguido en el desarrollo de la RRH y el número total de horas que le he dedicado a este trabajo de final de grado.

4.1. Estudio previo

Se realizó una fase de estudio previo en el que se hizo una búsqueda de plataformas de software radio que cumplieran con unas especificaciones concretas (velocidad de transmisión de datos fuera lo más rápida posible y poder transmitir al menos un ancho de banda de 50MHz entre otras) a un precio razonable.

4.2. Diseño

Durante el diseño, definí el diseño conceptual de la RRH e investigué sobre hardware, software y conceptos nuevos los cuales desconocía o recordaba vagamente de la carrera.

4.3. Desarrollo

Se incluye toda la configuración de dispositivos y programación de la plataforma de software radio.

4.4. Verificación

Se ha comprobado el funcionamiento y la configuración de los parámetros con el objetivo de que funcione correctamente.

4.5. Documentación

Esta fase se ha ido haciendo durante todo el trabajo, es decir, ha sido una fase continuada en la que cada día, o cada fin de semana que le he dedicado, se ha ido actualizando, de tal manera que lo he llevado al día.

A continuación, se presenta una gráfica con las horas totales invertidas en cada fase de forma mensual desde que empecé con este trabajo.

Tabla 1.1. Gráfico de horas mensuales dedicadas al TFG.

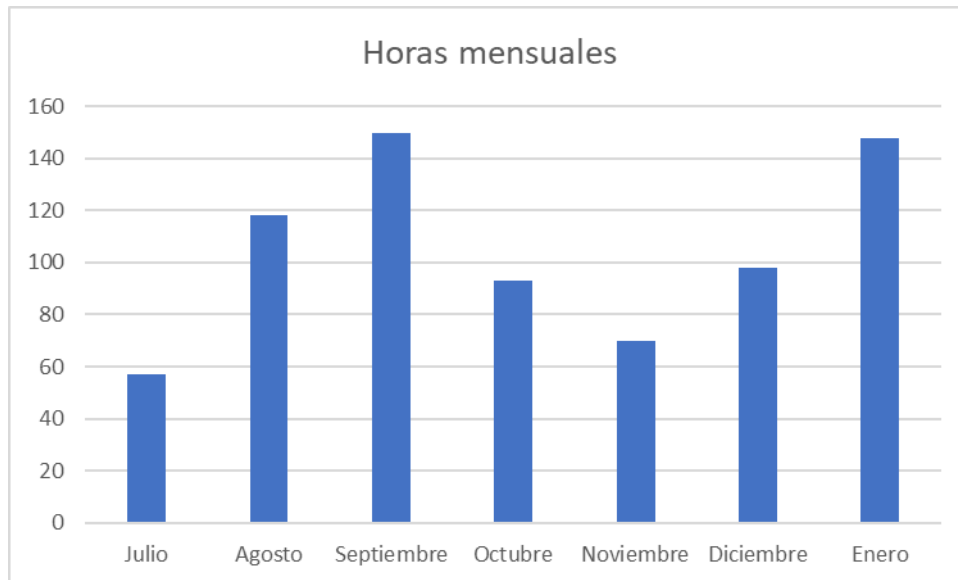


Fig.4.1: Gráfico de horas mensuales dedicadas al TFG.

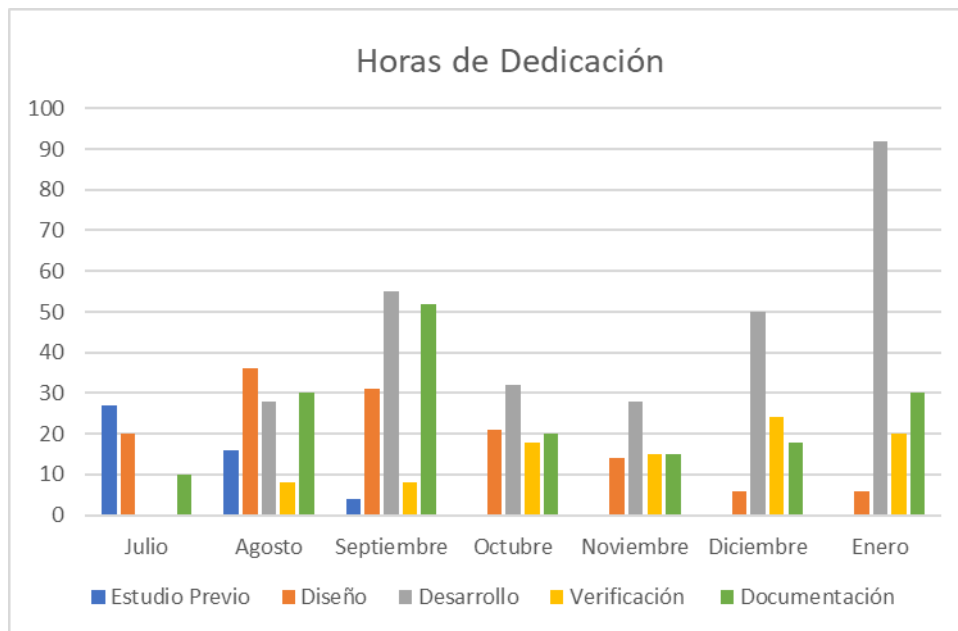


Fig. 4.2: Gráfico del reparto de horas de dedicación.

CONCLUSIONES

El objetivo final del trabajo era desarrollar una cabecera radio remota y realizar pruebas para verificar la señal RF generada.

Se han ido superando muchísimas dificultades. Al principio, aparte de que cuando empecé en julio no había mucha información sobre el funcionamiento de la LimeSDR, los mismos proveedores de la plataforma SDR que te facilitan material, subieron información con la que poder realizar la configuración con algunos errores. Errores en los que, con el nivel de programación que cogí el proyecto, hicieron que le dedicara un número de horas elevadas. Finalmente, con paciencia y razonamiento, solucioné el problema antes de que al tiempo subieran actualizaciones sin el error. La solución de este error y algún otro de caso similar, he decidido dejarlos en sus correspondientes anexos ya que también han formado parte de este trabajo.

Otra de las mayores dificultades ha sido conseguir pasar correctamente mediante una función, los datos recibidos de la VNF del servidor a la LimeSDR. El puntero que llegaba tenía un tamaño elevado y me saltaba segmentation fault. Limité este puntero guardando el número de muestras que yo deseaba en un array. De esta manera conseguí pasar las muestras. Finalmente, el mayor problema es que, estas muestras, a la hora de ser transmitidas, no se genera correctamente la señal.

Se ha documentado la tecnología utilizada con el objetivo de ver lo potente que es y para hacer ver que se debe apostar aún más por ella. No tengo duda que la evolución de esta tecnología irá de la mano de posibles mejoras y se conseguirán lograr cosas impensables. De hecho, yo estoy a la espera de que me llegue la LimeSDR Mini.

Como conclusión personal, la realización de este trabajo me ha aportado aumentar notablemente mi experiencia con la programación en los lenguajes C y C++. Aunque eran lenguajes que ya conocía, hacía tiempo que apenas tocaba. Esto me ha permitido refrescar los conocimientos y consolidarlos.

Por otro lado, utilizar una plataforma SDR como LimeSDR me ha permitido descubrir una forma de programar diferente. Aunque no deja de ser como la asignatura ESR en el 3B donde me introdujo en el mundo del Software Radio, lo que ha hecho diferente y más divertido este trabajo ha sido el trabajar con hardware y poder medir los resultados con equipos.

Finalmente, el hecho de intentar desarrollar una cabecera radio remota de principio a fin me ha aportado, aparte de satisfacción personal, una visión mucho más amplia de todos los factores que se tienen que tener en cuando quieres llevar a cabo algo, como la dedicación, el sacrificio y la complejidad que conlleva.

Líneas futuras

La cabecera radio remota desarrollada no deja de ser una buena base susceptible de mejora. Algunas de las ideas que también se querían realizar no se han podido llevar a cabo. A continuación, se detallarán alguna de ellas i el procedimiento para hacerse.

Me hubiera gustado poder analizar la distorsión de la señal. Para ello se debería enviar dos tonos separados una cierta frecuencia. Se empezaría con una potencia baja y se iría subiendo esta potencia de los tonos y la ganancia de los diferentes amplificadores, pero eso sí, de forma ordenada.

Optimizar el correcto funcionamiento del TX del sistema completo. Debido al nivel de programación i pocas herramientas de la API de la LimeSDR para hacer handle y funciones callback, no se ha podido perfeccionar tanto como me hubiera gustado.

Verificar el correcto funcionamiento del RX del sistema completo. Debido a los problemas encontrados con la parte TX, no se ha podido llegar a realizar un test verificando que la señal deseada enviada, es recibida y reenviada al servidor de nuevo.

REFERENCIAS Y BIBLIOGRAFÍA

- [1] “Software defined radio - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Software-defined_radio. [Accessed: 30-Jul-2017].
- [2] “LimeSDR-USB - Myriad-RF Wiki.” [Online]. Available: <https://wiki.myriardf.org/LimeSDR-USB>. [Accessed: 01-Aug-2017].
- [3] “El Odroid XU4 no viene a competir con la Raspberry Pi sino con tu ordenador de sobremesa.” [Online]. Available: <https://www.xatakahome.com/trucos-y-bricolaje-smart/el-odroid-xu4-no-viene-a-competir-con-la-raspberry-pi-sino-con-tu-ordenador-de-sobremesa>. [Accessed: 23-Jul-2017].
- [4] R. Roy and V. Bommankanti, “User manual odroid-xu4,” p. 83, 2015.
- [5] “LimeSDR Made Simple Part 1: Introduction - Myriad.” [Online]. Available: <https://myriardf.org/blog/limesdr-made-simple-part-1/>. [Accessed: 02-Aug-2017].
- [6] “Lime Suite driver architecture - Myriad.” [Online]. Available: <https://myriardf.org/blog/limesuite-driver-architecture/>. [Accessed: 02-Aug-2017].
- [7] “LimeSDR - Myriad-RF Wiki.” [Online]. Available: <https://wiki.myriardf.org/LimeSDR>. [Accessed: 02-Aug-2017].
- [8] “The LimeSDR application ecosystem - Myriad.” [Online]. Available: <https://myriardf.org/blog/limesdr-application-ecosystem/>. [Accessed: 02-Aug-2017].
- [9] “Lime Suite - Myriad-RF Wiki.” [Online]. Available: https://wiki.myriardf.org/Lime_Suite. [Accessed: 03-Aug-2017].
- [10] “LimeSDR-USB - Myriad-RF Wiki.” [Online]. Available: <https://wiki.myriardf.org/LimeSDR-USB>. [Accessed: 01-Aug-2017].
- [11] “LimeSDR-USB Quick Test - Myriad-RF Wiki.” [Online]. Available: https://wiki.myriardf.org/LimeSDR-USB_Quick_Test. [Accessed: 05-Aug-2017].
- [12] “LimeSDR HF Performance - Myriad-RF Wiki.” [Online]. Available: https://wiki.myriardf.org/LimeSDR_HF_Performance. [Accessed: 06-Aug-2017].
- [13] “LMS7002Mr3 Calibration Using MCU - Myriad-RF Wiki.” [Online]. Available: https://wiki.myriardf.org/LMS7002Mr3_Calibration_Using_MCU. [Accessed: 11-Sep-2017].

- [14] "LimeSDR La plataforma de la siguiente generación La radio cognitiva." [Online]. Available: <http://www.xe1gqp.org.mx/blog/wp-content/uploads/2016/09/LimeSDR-el-más-Flexible-de-la-nueva-generación-por-XE1GPD-Miguel-Gomez.pdf>
- [15] "Lime Suite driver architecture - Myriad." [Online]. Available: <https://myriadr.org/blog/limesuite-driver-architecture/>. [Accessed: 06-Aug-2017].
- [16] "Lime Suite Software -," pp. 1–11, 2017.
- [17] "Lms api -," pp. 1–7, 2017.
- [18] Lime Microsystems, "FPRF MIMO Transceiver IC With Integrated Microcontroller LMS7002M," p. 27, 2015.
- [19] L. M. Limited, "RF Transceiver IC - Programming and Calibration Guide -."
- [20] L. M. Limited, "Stream protocol," 2017.
- [21] J. Mitola, "Software radios-survey, critical evaluation and future directions," in *[Proceedings] NTC-92: National Telesystems Conference*, 1992, p. 13/15-13/23.
- [22] "History and discovery of rtl-sdr." [Online]. Available: http://rtlsdr.org/#history_and_discovery_of_rtlsdr. [Accessed: 31-Aug-2017].
- [23] E.E. Azzouz, A.K.Nandi. Procedure for automatic recognition of analogue and digital modulations, IEE Proc-Commun, Vol. 143 No 5 October 1996
- [24] H. Harada, R. Prasad. Simulation and Software radio for mobilecommunications. Artech House Publishers. Nueva York. 2002. 467p.
- [25] R. I. Lackey y D. W. Upmal. "SpeakEasy: The military Software Radio". IEEE Communications Magazine, Vol.33 No.5. Nueva York, mayo de 1995. p56-61
- [26] J. Mitola. Software Radio Architecture: Object oriented approaches to wireless systems engineering. John Wiley & Sons, Inc. Nueva York. 2000. 561p.
- [27] "Remote radio head - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Remote_radio_head. [Accessed: 28-Sep-2017].
- [28] "Remote Radio Head (RRH) - Fujitsu Global." [Online]. Available: <http://www.fujitsu.com/global/products/network/products/rrh/#id1>. [Accessed: 28-Sep-2017].

- [29] "How to capture raw IQ data from the RTL-SDR dongle and FM demodulate with MATLAB." [Online]. Available: http://www.aaronscher.com/wireless_com_SDR/RTL_SDR_AM_spectrum_demod.html. [Accessed: 27-Oct-2017].
- [30] Network Functions Virtualisation (NFV); Use NFV is present and SDN is future.Cases. .
- [31] D. Emery, Standards, APIs, Interfaces and Bindings. Acm.org.
- [33] "EZ-USB FX3™ SuperSpeed USB 3.0 peripheral controller." [Online]. Available: <http://www.cypress.com/products/ez-usb-fx3-superspeed-usb-30-peripheral-controller>. [Accessed: 25-Jan-2018].
- [34] A. S. Lakehal, "La Radio Definida por Software: Diseño de un receptor de banda aeronáutica VHF."
- [35] Resúmenes de las actas del VII Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica : Madrid, 12, 13 y 14 de julio de 2006. [Universidad Politécnica de Madrid], 2006.
- [36] "Cyclone IV - Overview." [Online]. Available: <https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html>. [Accessed: 18-Jan-2018].
- [37] "Teorema de muestreo de Nyquist-Shannon." [Online]. Available: https://es.wikipedia.org/wiki/Teorema_de_muestreo_de_Nyquist-Shannon. [Accessed: 19-Jan-2018].
- [38] "Frequency response of digital filter - MATLAB freqz - MathWorks España." [Online]. Available: <https://es.mathworks.com/help/signal/ref/freqz.html>. [Accessed: 12-Jan-2018]. de Cataluña], 2015
- [39] "Adquirir una Señal Analógica: Ancho de Banda, Teorema de Muestreo de Nyquist y Aliasing - National Instruments." [Online]. Available: <http://www.ni.com/white-paper/2709/es/#toc3>. [Accessed: 20-Jan-2018].
- [40] "Sources of Error in IQ Based RF Signal Generation - National Instruments." [Online]. Available: <http://www.ni.com/tutorial/5657/en/>. [Accessed: 20-Jan-2018].19-Jan-2018].
- [41] "Sockets UDP en C para Linux." [Online]. Available: <http://www.chuidiang.org/clinux/sockets/udp/udp.php>. [Accessed: 12-Dec-2017].
- [42] Apuntes de la asignatura de procesamiento digital de la señal. [Universidad Politécnica

ANEXOS

ANEXO 1: Guía de configuración e instalación para ODROID XU4

El objetivo de este anexo es dar a conocer el material necesario para el bloque que hará de CPU mediante la Odroid, así como la configuración de esta.

Lo primero de todo, disponer del material esencial como es un cable HDMI para conectar la odroid al monitor o pantalla, periféricos como teclado y ratón para establecer comunicación, cable Ethernet o un adaptador USB WiFi para descargar e instalar paquetes necesarios, conector de alimentación de 5V/4A y tarjeta microSD o módulo eMMC en donde estará el sistema operativo (capacidad mínima de 8GB).

Una vez tenemos el material preparado, se procede a grabar la imagen del sistema operativo en la tarjeta microSD ya que, como cualquier ordenador, requiere de un sistema operativo para arrancar. Puesto que para este proceso aun no estamos trabajando en la Odroid necesitaremos de otro sistema informático para grabar la imagen.

Conectamos la tarjeta microSD mediante un adaptador a nuestro sistema informático y a continuación se instalarán los drivers. Procedemos a ir a buscar la imagen de Ubuntu deseada en la siguiente dirección [Imágenes de Ubuntu]. Una vez descargado, es necesario descomprimir el archivo antes de grabarlo en nuestra microSD. Una vez descomprimido y teniendo el archivo con la extensión .img, utilizaremos el software Win32DiskImager para el proceso de grabación.

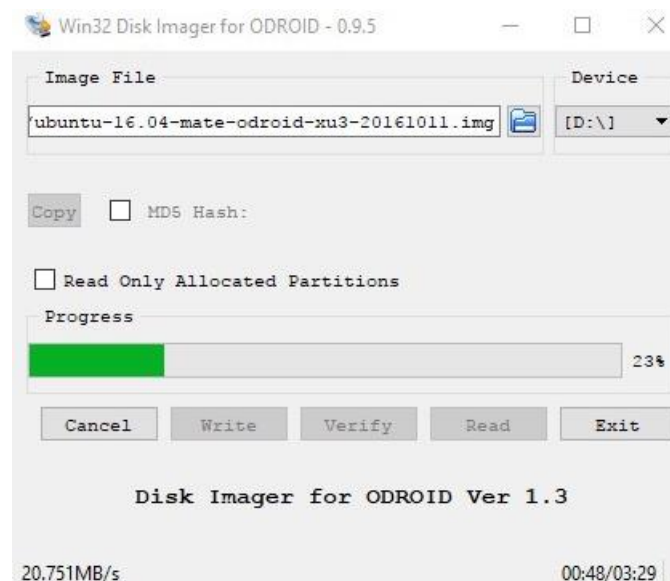


Figura 1: Proceso de grabación de la imagen mediante Win32DiskImager

Cuando acabe el proceso de grabación nos avisará y podremos proceder a retirar la tarjeta y pasar a ponerlo en la Odroid.

Una vez colocada la tarjeta microSD correctamente en la placa, poner el interruptor de soporte de arranque (SD/eMMC) en la posición de SD.

Una vez realizado todo esto, procedemos a conectar en sus correspondientes todo lo necesario dejando por último la alimentación. Cuando tengamos todo conectado, alimentamos la Odroid. Se iluminará el LED rojo y esperamos a que la imagen arranque el escritorio. Dependiendo del sistema operativo y el tipo de soporte de arranque utilizado, tardara más o menos en aparecer.

A continuación, si todo funciona correctamente, el LED azul parpadea. Esto indica que no hay ningún problema.

Una vez en el escritorio abrimos terminal y con el objetivo de actualizar el sistema operativo ejecutamos los siguientes comandos:

```
$ sudo apt-get update  
$ sudo apt-get upgrade  
$ sudo apt-get dist-upgrade
```

Una vez actualizado para conseguir que la hora no se modifique tras apagar la Odroid ejecutamos los siguientes comandos:

```
$sudo hwclock --set --date "2017-07-23 19:18"  
$sudo hwclock --hctosys  
$sudo dpkg-reconfigure tzdata
```

Y por defecto, el teclado no está configurado en español. Para configurarlo, ejecutar:

```
$sudo dpkg-reconfigure keyboard-configuration
```

Elegimos en la primera pantalla de selección Generic 105 (intl) keyboard, una vez seleccionada nos preguntan por el origen del teclado y elegimos Spain. Las opciones a continuación las dejaremos tal cual están.

Como nos interesa no tener nada en el escritorio, lo que vamos a hacer es no mostrar los iconos de boot que se ponen por defecto. Para ello en Applications, nos ponemos encima de System Tools y seleccionamos dconf Editor. A partir de este editor de configuración configuraremos no mostrar los iconos de boot.



Figura 2: Configuración realizada para no mostrar los iconos en el escritorio

Como medida de seguridad se ha realizado un Backup. Esta copia se ha realizado como se indica a continuación:

Primero se ha comprobado la información de la partición de la SD mediante el comando:

```
$ sudo fdisk -l /dev/mmcblk0
```

A continuación, nos sale la información siguiente:

```
odroid@odroid:~$ sudo fdisk -l /dev/mmcblk0
[sudo] password for odroid:
Disk /dev/mmcblk0: 14.9 GiB, 15931539456 bytes, 31116288 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x3cedfd53

Device      Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk0p1  2048    264191    262144    128M  c  W95 FAT32 (LBA)
/dev/mmcblk0p2 264192 31115264 30851073  14.7G  83  Linux
odroid@odroid:~$
```

Figura 3: Información de la partición de la SD

Ahora utilizaremos dd para crear una copia de seguridad que sea tan grande como el espacio utilizado:

```
$ sudo dd if=/dev/mmcblk0 of=MyBackup.img bs=512 count=31115264
```

Le decimos hasta 31115264, ya que fue el último bloque de nuestra partición. Una vez realizado esto tendremos un archivo .img con nuestra copia de seguridad. Si detectáramos algún problema y tuviéramos que utilizarlo, bastaría con cargar la imagen en el Win32DiskImager y seguir los mismos pasos que el procedimiento anterior.

ANEXO 2: Guía de configuración e instalación para LimeSDR en ODROID XU4

En este anexo, se detallarán los pasos realizados para la instalación de paquetes necesarios y la compilación de Lime Suite en el sistema operativo Ubuntu.

Primero de todo, procederemos a instalar los drivers PPA de Ubuntu. En la consola y ejecutaremos los siguientes comandos:

```
$ sudo add-apt-repository -y ppa:myriadrf/drivers
$ sudo apt-get update
$ sudo apt-get install limesuite limesuite-udev limesuite-images
$ sudo apt-get install soapysdr soapysdr-module-lms7
```

Una vez instalados estos drivers, procederemos a instalar todas aquellas dependencias necesarias para la construcción de ficheros (librería principal y de compilación, soporte de hardware y dependencias graficas). Para ello ejecutaremos los siguientes comandos:

```
$ sudo apt-get install git g++ cmake libsqlite3-dev
$ sudo apt-get install libsoapysdr-dev libi2c-dev libusb-1.0-0-dev
$ sudo apt-get install libwxgtk3.0-dev freeglut3-dev
```

El siguiente paso, es comprobar que los controladores necesarios se han instalado correctamente y que las utilidades de LimeSuite encuentran la placa LimeSDR conectada a la Odroid.

Para ello ejecutaremos los comandos:

```
LimeUtil --find
```

```
odroid@odroid:~$ sudo LimeUtil -find
* [LimeSDR-USB, media=USB 3.0, module=STREAM, addr=1d50:6108, serial=0009060B00460D22]
```

Figura 4: Información sobre el dispositivo encontrado

LimeUtil --info

```
#####
## LimeSuite information summary
#####

Version information:
  Library version:      v17.06.0-myriadrf1~xenial
  Build timestamp:     2017-06-21
  Interface version:   v2017.6.0
  Binary interface:    17.06-1

System resources:
  Installation root:   /usr
  User home directory: /home/odroid
  App data directory:  /home/odroid/.local/share/LimeSuite
  Config directory:    /home/odroid/.limesuite
  Image search paths:
    - /home/odroid/.local/share/LimeSuite/images
    - /usr/share/LimeSuite/images

Supported connections:
  * NovenaRF7
  * PCIEXillybus
  * STREAM
  * uLimeSDR
```

Figura 5: Información sobre la instalación de LimeSuite

Cuando este todo configurado, obtendremos el repositorio de LimeSuite, se creará un directorio nuevo de un proyecto de prueba, se configurará este proyecto utilizando cmake y se construirá e instalará el proyecto.

```
$ git clone https://github.com/myriadrf/LimeSuite.git
```

```
$ cd LimeSuite
```

```
$ mkdir builddir && cd builddir
```

```
$ cmake ../
```

```
$ make -j4 *
```

(*) En este paso tuve problemas. Cuando obtenemos LimeSuite mediante el comando git, el fichero CMakeList.txt contiene `-mfpmath=both` que no está definido. Es por esto por lo que se ha procedido a borrarlo y así desaparece el error y compila.

```
$ sudo make install
```

Una vez realizada toda la instalación realizaremos un último paso, instalar las “udev rules” para permitir que usuarios no root accedan a dispositivos basados en USB como LimeSDR:

utilizando cmake y construirán e instalarán el proyecto.

```
$ cd LimeSuite/udev-rules
```

```
$ sudo ./install.sh
```

El siguiente paso es poner en marcha el Lime Suite, ejecutando el siguiente comando dentro de bin [LimeSuite/builddir/bin]:

```
$ sudo ./LimeSuiteGUI
```

Una vez dentro, con el objetivo de configurar el puerto USB, seguimos las siguientes instrucciones:

Options → ConnectionSettings

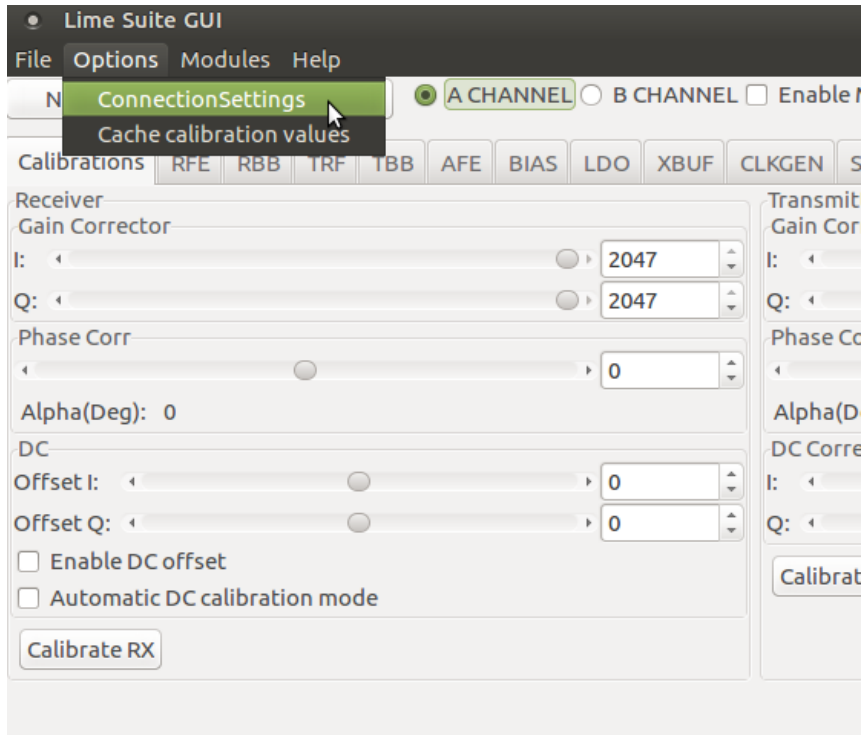


Figura 6: Menú de opciones en el panel de Lime Suite GUI

Y una vez allí seleccionamos el USB 3.0 y le damos al botón connect.

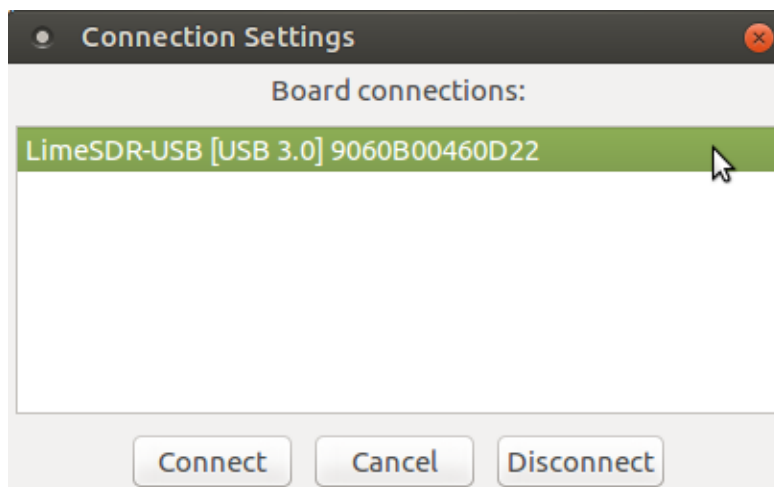


Figura 7: Configuraciones de conexión de Lime Suite GUI

La comunicación entre la LimeSDR-USB y la Odroid, la cual ejecuta el software LimeSuiteGUI, se realiza a través de la interfaz USB 3.0.

La placa LimeSDR-USB viene con el firmware FX3 preprogramado y está lista para usar con una versión baja. Es por esto por lo que procederemos a actualizar las imágenes de firmware y FPGA.

Para ello, las descargaremos mediante los siguientes enlaces: FX3 y FPGA.

Una vez obtenidos y descomprimidos, en el menú de Lime Suite GUI abrimos “Modules” y “Programming”.

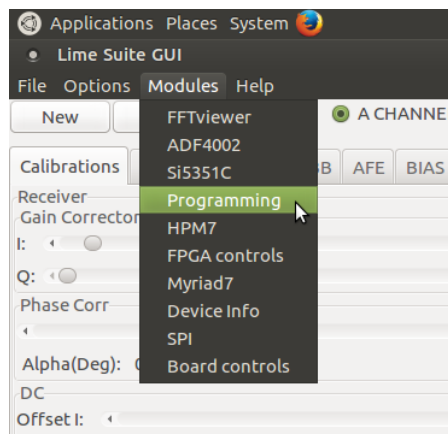


Figura 8: Menú de módulos en el panel de Lime Suite GUI

Por seguir un orden, primero seleccionamos el Device FX3 y en “Open” cargamos el archivo con extensión .img descargado anteriormente y seleccionamos el modo Firmware to Flash.

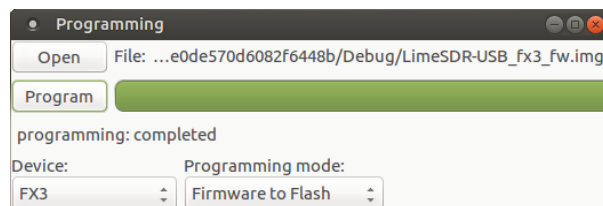


Figura 9: Actualización del FX3 mediante Lime Suite GUI

Una vez actualizado, haríamos lo mismo para la Altera FPGA. Cargamos el archivo descargado con la extensión .rbf y en este caso seleccionamos el modo Bitstream to Flash.

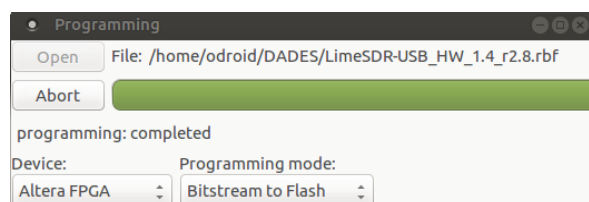


Figura 10: Actualización de la FPGA mediante Lime Suite GUI

ANEXO 3: Comprobación del transceptor LMS7002M

El chip transceptor LMS7002M utiliza Lime Suite GUI como herramienta para configurar y depurar. Como hemos visto en el Anexo 2, Lime Suite GUI permite actualizar el firmware del microcontrolador y la Gateway FPGA, también permite ver tramas FFT, configurar módulos y ver, editar, guardar y cargar el estado de los registros LMS7002M.

Con el objetivo de confirmar el funcionamiento de la placa, procederemos a generar una señal WCDMA y luego visualizarla mediante un loopback.

Empezaremos por descargar dos archivos de configuración para el transceptor y ejemplos de formas de onda: Archivo de configuración y Ejemplos de formas de onda (extraer en un directorio llamado lms7suite_wfm).

Para generar una señal WCDMA y luego visualizarla mediante un loopback utilizaremos el archivo 'self_test.ini'.

Lo que haremos a continuación es enviar una forma de onda desde la Odroid a la LimeSDR, la señal WCDMA se genera en el FPGA y la conexión de transmisor a receptor se lleva a cabo por un interruptor RF de a bordo. Para visualizar esa señal utilizaremos un visualizador de FFT.

Empezamos entrando en Lime Suite GUI y conectando el puerto USB 3.0 como hemos hecho en el anexo dos. A continuación, procedemos a cargar el archivo de configuración 'self_test.ini' descargado anteriormente, para ello le damos a "open".

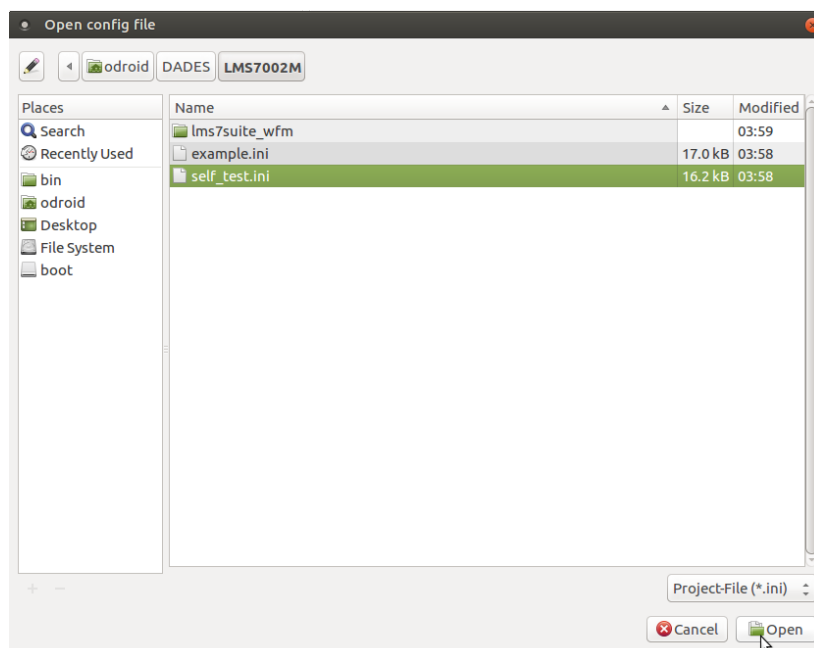


Figura 11: Procedimiento para cargar un archivo de configuración

Una vez cargado seleccionamos la pestaña "GUI →Chip"

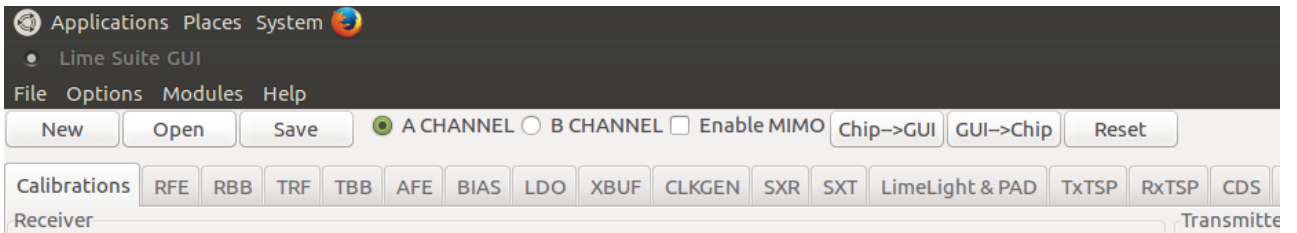


Figura 12: Pestañas de Lime Suite GUI

Para configurar la SXT, vamos a la pestaña SXT, le damos al botón “Calculate” y después al “Tune”.

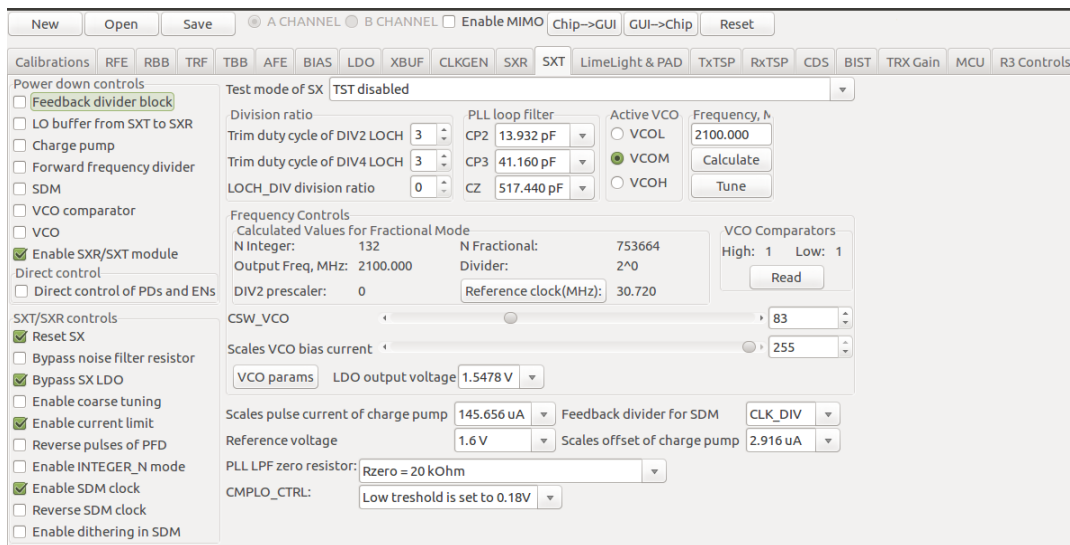


Figura 13: Pestaña SXT de Lime Suite GUI

Para configurar el CLKGEN, vamos a la pestaña CLKGEN, le damos al botón “Calculate” y después al “Tune”.

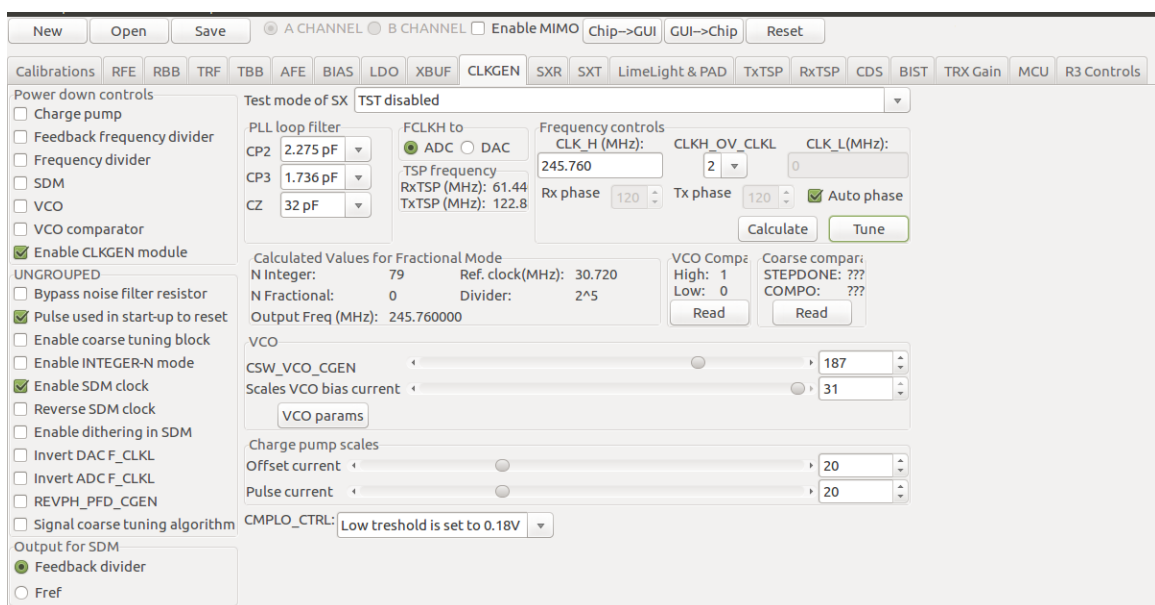


Figura 14: Pestaña CLKGEN de Lime Suite GUI

Podemos ver como Lime Suite GUI muestra por consola la configuración que se aplica.

```
[18:48:29] INFO: Estimated reference clock 30.7196 MHz
[18:48:29] INFO: Selected reference clock 30.720 MHz
[18:48:29] INFO: Connected Control port: LimeSDR-USB FW:2 HW:4 Protocol:1 GW:2 GW_rev:8 Ref Clk: 30.72 MHz
[18:53:08] INFO: SXT frequency set to 2100.000000 MHz
[18:54:28] INFO: CGEN frequency set to 245.760000 MHz
```

Control port: LimeSDR-USB FW:2 HW:4 Protocol:1 GW:2 GW_rev:8 Ref Clk: 30.72 MHz

Figura 15: Consola de Lime Suite GUI

Ahora activaremos el Loopback, para ello vamos a “Modules” y “Board controls” y una vez ahí tildamos solo los loopback de los canales A y B.

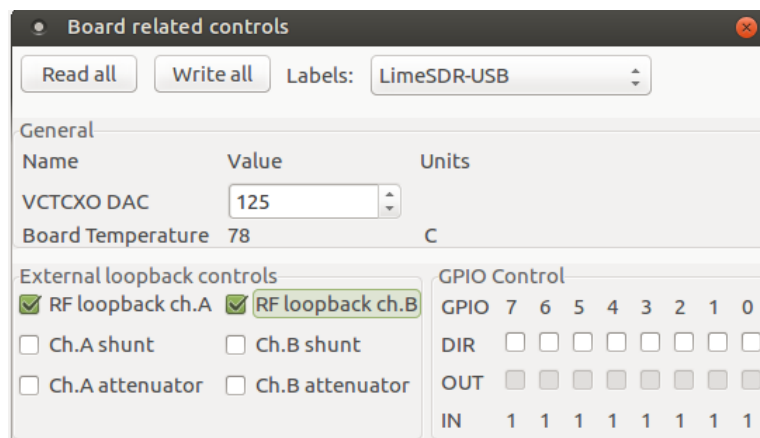


Figura 16: Menú de controles relacionados con la placa

Por último, antes de ver la señal, cargamos la forma de onda, en este ejemplo se utiliza una WCDMA. Como esta se generará en la FPGA, vamos a la pestaña de “Modules” y “FPGA controls”. Estando en la ventana, ahí tildamos “MIMO”, hacemos click en WCDMA y le damos al play.

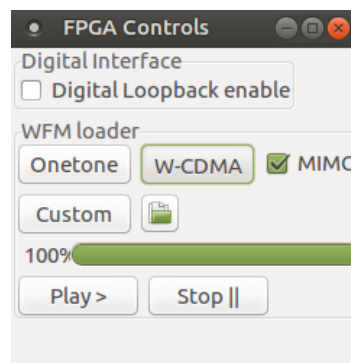


Figura 17: Menú de controles de la FPGA

Para ver la señal seleccionamos “Modules” y “FFT viewer”. Seguidamente, lo configuramos a nuestro interés, es decir, “Data Reading” ponemos LMS MIMO, en Graphs → Display Channel → A&B y le damos a START.

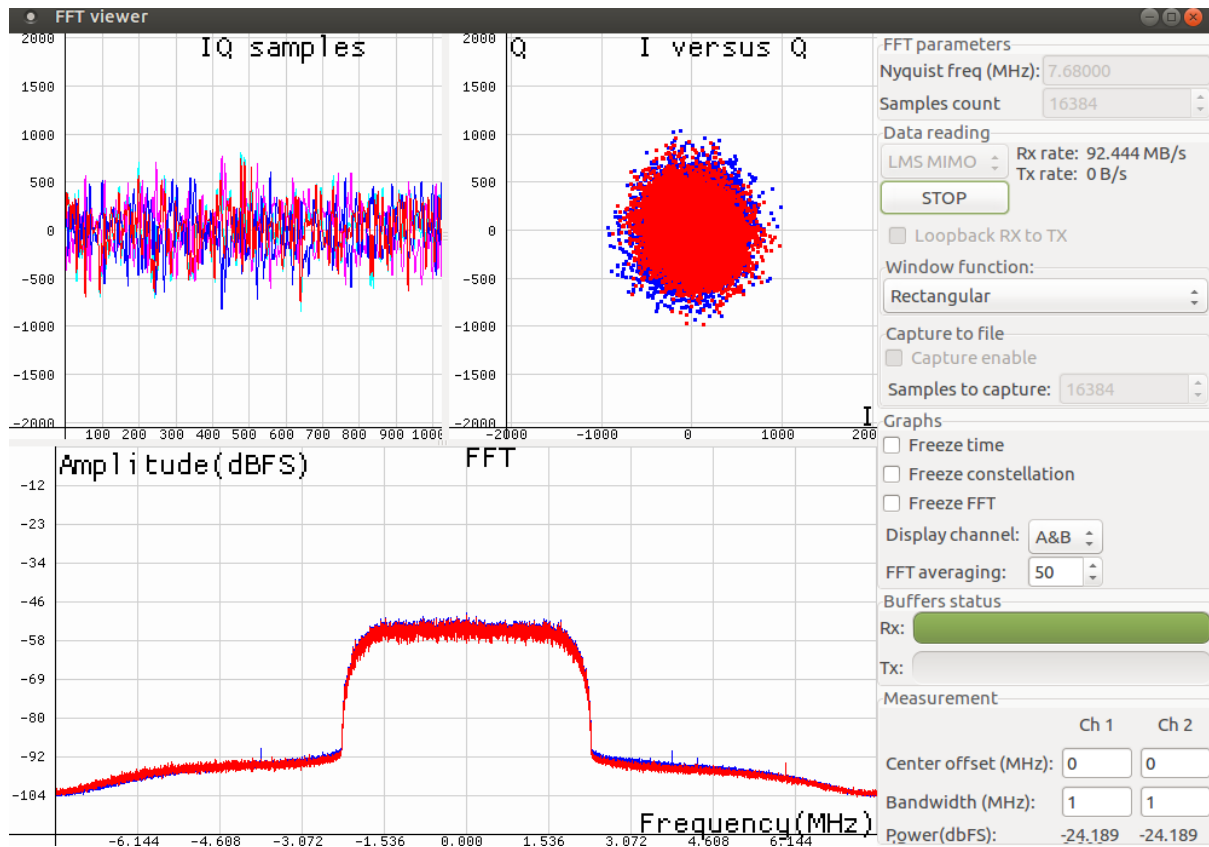


Figura 18: FFT viewer

ANEXO 4: Trabajar con Pothos

En este apartado el objetivo será configurar la Odroid para poder realizar proyectos mediante Pothos.

Los comandos que vienen a continuación, serán los necesarios para realizar una completa configuración e instalación ya que incluyen todas las dependencias necesarias.

```
$ sudo apt-get install libnuma-dev cmake g++
$ sudo apt-get install libpython-dev python-numpy
$ sudo apt-get install qtbase5-dev libqt5svg5-dev libqt5opengl5-dev
$ sudo apt-get install libqwt-qt5-dev
$ sudo apt-get install portaudio19-dev libjack-jackd2-dev
$ sudo apt-get install graphviz
$ sudo add-apt-repository -y ppa:pothosware/support
$ sudo add-apt-repository -y ppa:pothosware/framework
$ sudo add-apt-repository -y ppa:bladerf/bladerf
$ sudo add-apt-repository -y ppa:ettusresearch/uhd
$ sudo apt-get update
$ sudo apt-get install libpoco-dev libspuce-dev nlohmann-json-dev
$ sudo apt-get install libmuparserx-dev
```


Ahora instalaremos las herramientas que necesitaremos, así como todo lo necesario de Soapy SDR:

```
$ sudo add-apt-repository -y ppa:myriadrf/drivers
$ sudo apt-get update
$ sudo apt-get install libsoapysdr-dev soapysdr python-soapysdr python-numpy
$ sudo apt-get install soapysdr-module-remote soapysdr-server
$ sudo apt-get install osmo-sdr soapysdr-module-osmosdr
$ sudo apt-get install rtl-sdr soapysdr-module-rtlsdr
$ sudo apt-get install bladerf soapysdr-module-bladerf
$ sudo apt-get install hackrf soapysdr-module-hackrf
$ sudo apt-get install uhd-host uhd-soapysdr soapysdr-module-uhd
$ sudo apt-get install umtrx uhd-host uhd-soapysdr soapysdr-module-uhd
$ sudo apt-get install miri-sdr soapysdr-module-mirisdr
$ sudo apt-get install soapysdr-module-rfspace
$ sudo apt-get install airspy soapysdr-module-airspy
```

Después de instalar los paquetes, dependencias necesarias, herramientas que necesitaremos y todo lo necesario de Soapy SDR, ahora podemos clonar el repositorio desde git:

```
$ git clone --recursive https://github.com/pothosware/pothos.git
```

Cuando se haya obtenido, miraremos actualizaciones desde el directorio de pothos:

```
$ cd pothos
$ git pull origin master
$ git submodule update --init --recursive --remote
```

Una vez actualizado procedemos a su construcción e instalación:

```
$ mkdir build
$ cd build
$ cmake ..
$ make -j4 (*)
$ sudo make install
$ sudo ldconfig
$ PothosUtil --self-tests
$ PothosGui
```

(*) Al realizar make -j4 me encontré con errores de caminos de librerías que solucioné aplicando el siguiente comando:

```
$ sudo ln -s /usr/lib/arm-linux-gnueabi/mesa-egl/libGLESw2.so.2 /usr/lib/arm-  
linux-gnueabi/libGLESw2.so.2
```

Otros errores o warnings que se han dado al iniciar PothosGui y su solución son los siguientes:

Unable to find dan X11 visual which matches EGL config 1. Could not initialize OpenGL for RasterGLSurface, reverting to RasterSurface.

Solucion: en la ruta `/usr/lib/arm-linux-gnueabi` posiblemente encontremos `mali-egl` y `mesa-egl`. Nos quedamos con `mesa-egl` y borramos `mali-egl`.

Problemas con exynos:

```
$/usr/lib/arm... rm -r libEGL libGLES  
$run ldconfig
```

Finalmente entramos perfectamente al entorno:

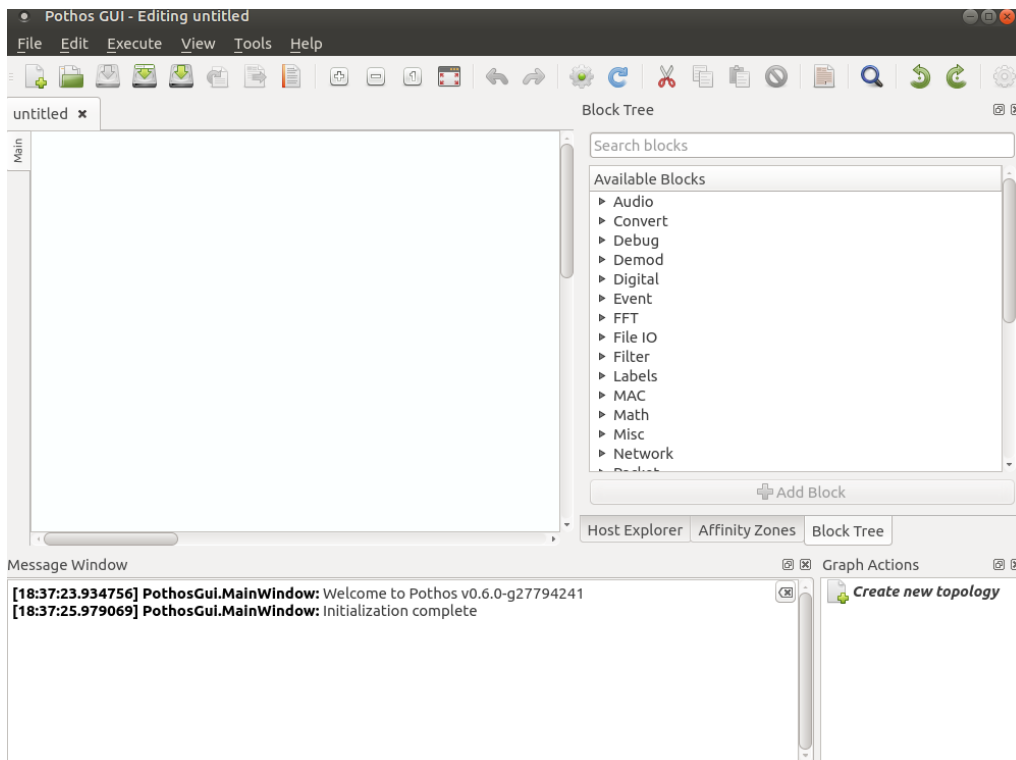


Figura 19: Entorno Pothos GUI

Para comprobar la correcta instalación, ejecutaremos los siguientes comandos que nos proporcionaran información de esta:

```
$ PothosUtil --system-info
```

```

odroid@odroid:~/pothos/build$ PothosUtil --system-info
Lib Version: 0.5.0-0.5.0-ppa1-xenial
API Version: 0.5.0
ABI Version: 0.5
Root Path: /usr
Data Path: /usr/share/Pothos
User Data: /home/odroid/.local/share/Pothos
User Config: /home/odroid/.config/Pothos
Runtime Library: /usr/lib/arm-linux-gnueabi/libPothos.so.0.5.0
Util Executable: /usr/bin/PothosUtil
Dev Include Path: /usr/include
Dev Library Path: /usr/lib/arm-linux-gnueabi

```

Figura 20: Información sobre la instalación de Pothos

\$ SoapySDRUtil --info

```

odroid@odroid:~/pothos/build$ SoapySDRUtil --info
#####
## Soapy SDR -- the SDR abstraction library
#####
Lib Version: v0.6.0-0.6.0-myriadrf1-xenial
API Version: v0.6.0
ABI Version: v0.6
Install root: /usr
Search path: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6
Search path: /usr/local/lib/arm-linux-gnueabi/SoapySDR/modules0.6
Search path: /usr/local/lib/SoapySDR/modules0.6
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libHackRFSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libLMS7Support.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libairspySupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libbladeRFSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libmiriSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libosmosdrSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libremoteSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/librfspaceSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/librtlsdrSupport.so
Module found: /usr/lib/arm-linux-gnueabi/SoapySDR/modules0.6/libuhdSupport.so
Loading modules... linux; GNU C++ version 5.3.1 20151219; Boost_105800; UHD_003.009.002-0-unknown
done
Available factories...airspy, bladerf, hackrf, lime, miri, null, osmosdr, remote, rfspace, rtlsdr, uhd,
odroid@odroid:~/pothos/build$ █

```

Figura 21: Información sobre la instalación de SoapySDR

Tras realizar tutoriales he procedido al desarrollo de cero de un ejemplo simple de transmisión de datos remota.

Gracias a realizar esto, he consolidado muchos conocimientos del software. A diferencia de los ejemplos, cuando empiezas de cero, no todo es ir haciendo, tienes que entrar en LimeSuiteGui, asignar el puerto USB3.0 y asegurarte de haber realizado la configuración que hemos visto anteriormente en el punto X y de haber realizado la calibración ya que cogerá esos datos de la cache.

Colocamos los bloques necesarios para este simple sistema. La SDR Source, Spectrogram, Periodogram y Numeric Entry.

Utilizaremos SDR source para la generación de muestras (Samples) y visualizaremos estas mediante el Spectrogram y el Periodogram.

El Spectrogram lo utilizaremos para visualizar el dominio de frecuencia de un solo canal mientras que el Periodogram el dominio de frecuencia multicanal.

Con el Numeric Entry será con lo que elegiremos el rango de frecuencias en el cual queremos movernos.

Una vez colocado y asignadas las conexiones correspondientes procedemos a activar la topología.

Me he encontrado con varios errores, el Jack y el proxy.

Al activar la topología me saltaba errores del jack los cuales solucione instalando los siguientes comandos:

```
$ sudo apt-get install dbus-x11
$ sudo apt-get install jackd2
```

Una vez completada la instalacion, al empezar a ponerlo en marcha mediante el comando:

```
$ dbus-launch jack_control start
```

Me encontré con el siguiente error:

```
DBus exception:org.jackaudio.Error.Generic:Failed to open server
```

Este error lo solucione como indico a continuación:

```
$ pulseaudio – kill
$ jack_control start
$ jack_control exit
$ pulseaudio --start
$ ps -aux | grep jackd
$ kill .9 jacksPID
```

Por otro lado, al activar la topología, encontraba un error al conectarme con el proxy que decía lo siguiente:

```
Proxy server: No active connections - terminating
```

Al querer hacerlo remoto, hay que activar mediante otro terminal el servidor proxy (0.0.0.0 puerto 16415), esto se hace mediante el comando:

```
$ PothosUtil – proxy-server=""
```

o para confirmar que sea el de por defecto:

```
$ PothosUtil – proxy-server="0.0.0.0:16415"
```

Entonces una vez activado procedemos a añadirlo a Host Explorer y Affinity Zones.

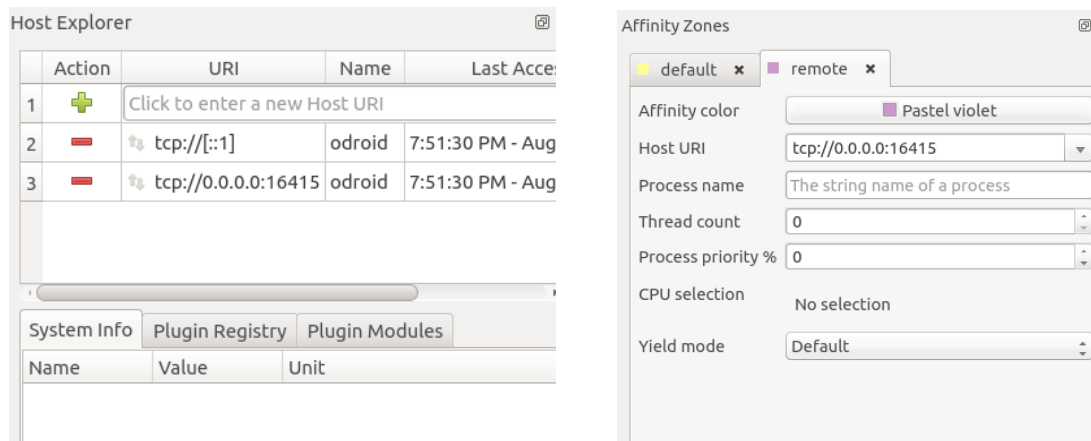


Figura 22: Configuración del Host Explorer y Affinity Zones para control remoto en Pothos

Después de haber realizado esto, al Spectrogram y Periodogram, le aplicamos esa Affinity Zone llamada remote. De tal forma, cuando el proxy no esté conectado, estos no funcionaran.

Una vez pasado todos estos errores obtendríamos algo como lo que se puede apreciar a continuación en la figura siguiente:

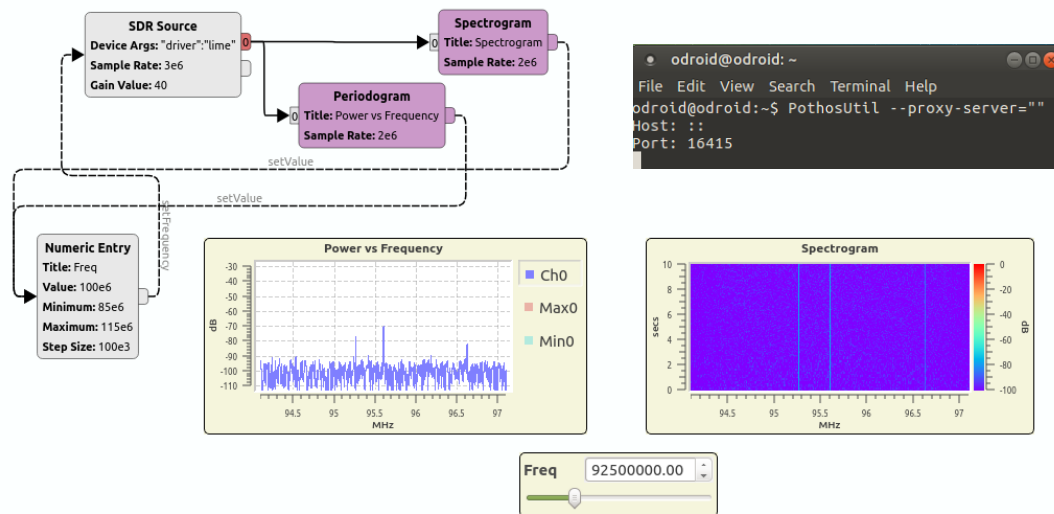


Figura 23: Escenario de una transmisión remota en Pothos

ANEXO 5: Algunas funciones de la Lime API

En este anexo se verán algunas de las funciones utilizadas.

Generación de canal de ruido:

```
float rand_gauss (void);

float rand_gauss (void) {
    float v1,v2,s;

    do {
        v1 = 2.0 * ((float) rand()/RAND_MAX) - 1;
        v2 = 2.0 * ((float) rand()/RAND_MAX) - 1;

        s = v1*v1 + v2*v2;
    } while ( s >= 1.0 );

    if (s == 0.0)
        return 0.0;
    else
        return (v1*sqrt(-2.0 * log(s) / s));
}

void gen_noise_c(_Complex float *x, float variance, int len);

void gen_noise_c(_Complex float *x, float variance, int len) {
    int i;
    for (i=0;i<len;i++) {
        __real__ x[i] = rand_gauss();
        __imag__ x[i] = rand_gauss();
        x[i] *= variance;
    }
}
```

Generación de un tono:

```
int gen_toneCOMPLEX(_Complex float *func_out){

    const int tx_size = 1024*2;
    _Complex float *tx_buffer;
    tx_buffer = func_out;
    for (int i = 0; i <tx_size; i++)
    {
        tx_buffer[2*i] = (float)cos(2*M_PI*i/5);
        tx_buffer[2*i+1] = (float)sin(2*M_PI*i/5);
    }
}
```

Generar un archivo para analizarlo en Matlab:

```
bool write_file(char *filename, void *databuff, long int size){

    long int writelength=0;
    FILE *fp;

    fp=fopen(filename, "w");
    writelength=fwrite(databuff,1,size,fp);
    if(size != writelength){
        printf("Writing filename %s: wrote data=%ld, expected=%ld\n", filename, writelength, size);
    }
    fclose(fp);
    return(writelength);
}
```

Algunas de las funciones de configuración la LimeSDR:

```

if (LMS_Init(device)!=0) //Initialize device with default configuration
    error();

if (LMS_EnableChannel(device,LMS_CH_RX,0,ENABLE_RX_CHANNEL0_IN) != 0) //ENABLE_RX_CHANNEL0
    error();

if (LMS_EnableChannel(device,LMS_CH_RX,1,ENABLE_RX_CHANNEL1_IN) != 0) //ENABLE_RX_CHANNEL1
    error();

if ((n = LMS_GetNumChannels(device, LMS_CH_RX)) < 0) //Get number of RX channels
    error();
cout << "Number of RX channels: " << n << endl;

if (LMS_EnableChannel(device,LMS_CH_TX,0,ENABLE_TX_CHANNEL0_IN)!=0) //ENABLE_TX_CHANNEL0
    error();

if (LMS_EnableChannel(device,LMS_CH_TX,1,ENABLE_TX_CHANNEL1_IN)!=0) //ENABLE_TX_CHANNEL1
    error();

if ((n = LMS_GetNumChannels(device, LMS_CH_TX)) < 0) //Get number of TX channels
    error();
cout << "Number of TX channels: " << n << endl;

// Sample Rate

if (LMS_SetSampleRate(device,SampleRate_IN,OVERSAMPLE_IN)!=0)
    error();

//Set TX center frequency

if (LMS_SetLOFrequency(device,LMS_CH_TX, 0, TX_FREQ_IN)!=0)
    error();

if (LMS_SetLOFrequency(device,LMS_CH_TX, 1, TX_FREQ_IN)!=0)
    error();

//Set RX center frequency

if (LMS_SetLOFrequency(device, LMS_CH_RX, 0, RX_FREQ_IN) != 0)
    error();

if (LMS_SetLOFrequency(device, LMS_CH_RX, 1, RX_FREQ_IN) != 0)
    error();

//Antennas configurations

if (LMS_SetAntenna(device, LMS_CH_TX, 0, LMS_PATH_TX1)!=0) //TX1_1
    error();

if (LMS_SetAntenna(device, LMS_CH_RX, 0, LMS_PATH_LNAH)!=0) //RX1_1
    error();

```

ANEXO 6: Función para dibujar la FFT mediante MATLAB

```
function plot_FFT(x,n0,nf,fs,f0,title_of_plot)

%
% plot_FFT(x,n0,nf)
%
% Plots the FFT of sampled IQ data
%
%     x -- input signal
%     fs -- sampling frequency [MHz]
%     f0 -- center frequency [MHz]
%     n0 -- first sample (start time = n0/fs)
%     nf -- block size for transform (signal duration = nf/fs)
%     title_of_plot--title of plot (string) (optional)
%
%-This extracts a segment of x starting at n0, of length nf, and plots the FFT.
%

x_segment=x(n0:(n0+nf-1)); %extracts a small segment of data from signal

p=fftshift(fft(x_segment)); %find FFT
z = 20*log10(abs(p)/max(abs(p))); %normalize

Low_freq=(f0-fs/2); %lowest frequency to plot
High_freq=(f0+fs/2); %highest frequency to plot

N=length(z);
freq=[0:1:N-1]*(fs)/N+Low_freq;

plot(freq,z);
axis tight
xlabel('Frequency [MHz]','FontSize', 14)
ylabel('Relative amplitude [dB down from max]','FontSize', 14)
grid on
set(gcf,'color','white');

if nargin==6
    title(title_of_plot,'FontSize', 14)
else
    title({'Spectrum','[Center frequency = ' num2str(f0) ' MHz]'},'FontSize', 14)
end
title({'Spectrum','[Center frequency = ' num2str(f0) ' MHz]'},'FontSize', 14)

%Add vertical line
y1=get(gca,'ylim');
hold on;
plot([f0 f0],y1,'r-','linewidth',2);
hold off;
```


ANEXO 7: Teorema de muestreo de Nyquist-Shannon

El teorema de muestreo de Nyquist-Shannon, es un teorema fundamental de la teoría de la información de especial interés en las telecomunicaciones.

Fue desarrollado por Harry Nyquist y demostrado por Claude Elwood Shannon, donde se afirmaba que una señal analógica puede ser reconstruida, sin error, de muestras tomadas en iguales intervalos de tiempo. La teoría de muestreo define que, para una señal de ancho de banda limitado, la frecuencia de muestreo (f_s), debe ser mayor que dos veces su ancho de banda (BW).

$$\omega_s > 2\omega_m \quad (1)$$

Un ejemplo sería la digitalización de la voz. El ancho de banda de la voz es, aproximadamente, 4 MHz. Para reconstruir la señal sin error deberíamos utilizar una frecuencia de muestreo superior a 8 MHz (dos veces el ancho de banda).

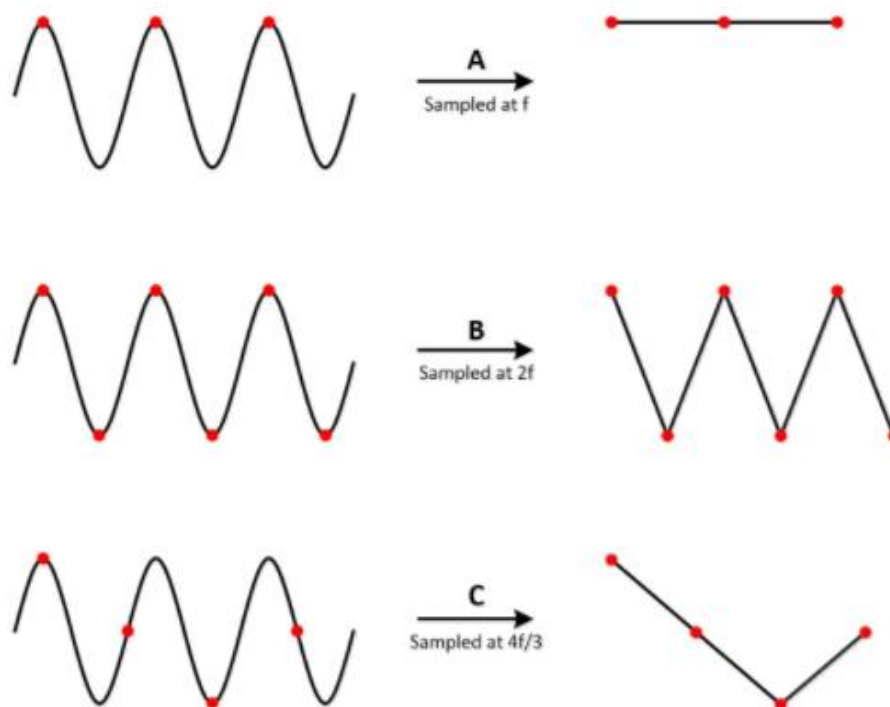


Figura 24: Reconstrucción de la señal con diferentes frecuencias de muestreo [39].

ANEXO 8: Reconstrucción de la señal analógica

Se explicará brevemente un mínimo de teoría de reconstrucción de la señal para acabar de entender porque sucede que, en algunas imágenes, a los múltiplos de las frecuencias de muestreo, tiende a cero.

Las muestras de LimeSDR son convolucionadas por un filtro rectangular en tiempo.

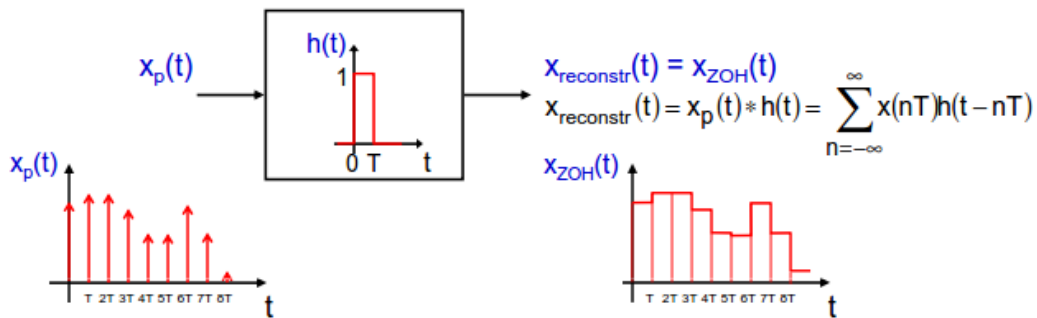


Figura 25: Resultado grafico de convolucionar las muestras.

Lo que en tiempo es convolucionar, haciendo la transformada de Fourier, en frecuencia es el producto.

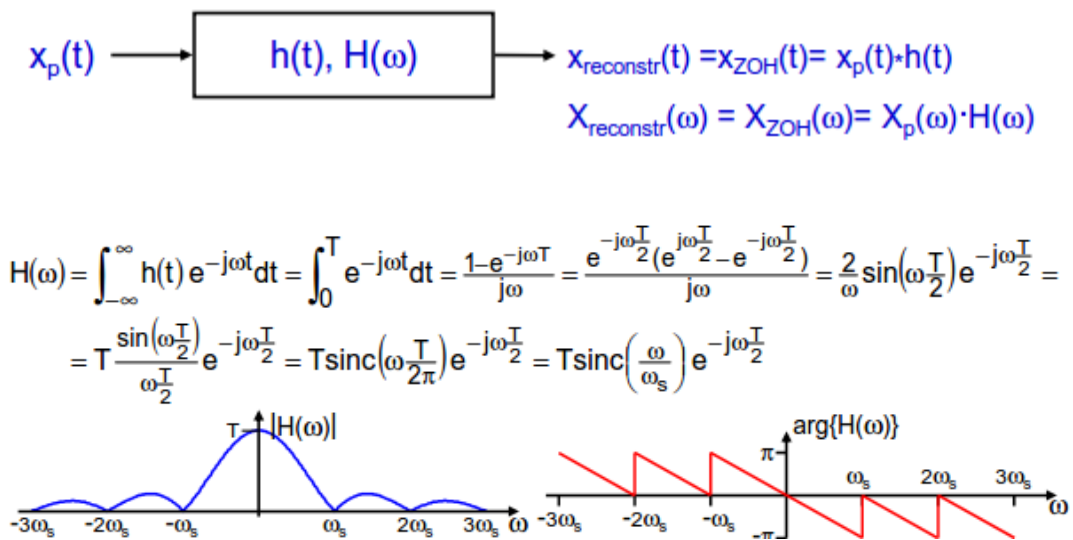


Figura 26: Transformada de Fourier.

Una vez en el dominio frecuencial, se realiza el producto.

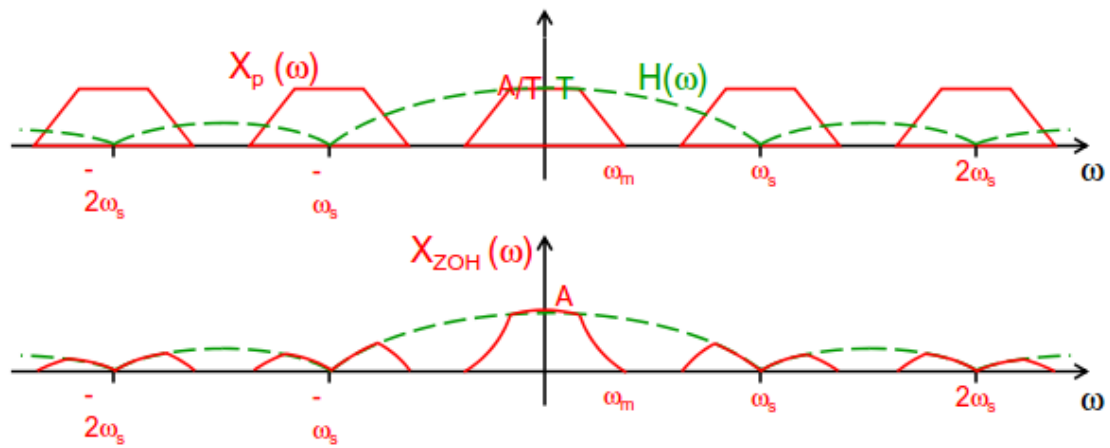


Figura 27: Resultado grafico en el dominio frecuencial.

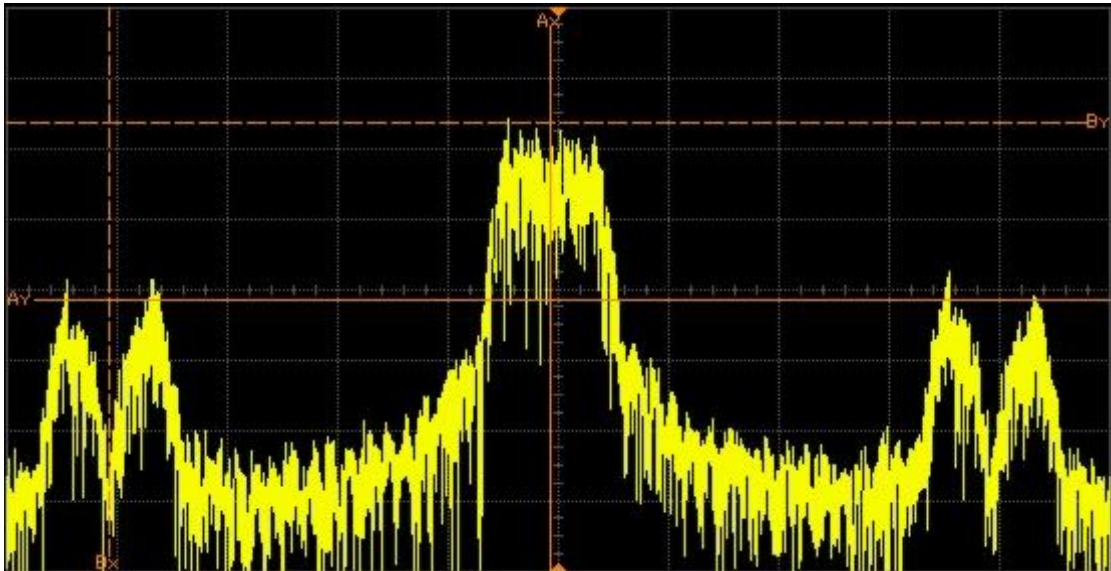


Figura 28: Resultado obtenido del analizador de espectros al enviar un canal de ruido.

ANEXO 9: Aliasing y filtro antialiasing

Para una reconstrucción sin errores, debemos muestrear a una frecuencia de muestreo que cumpla con el teorema de Nyquist. Si muestreamos una señal a una frecuencia de muestreo que no cumpla con el teorema, los componentes de frecuencias falsas más bajas aparecen en los datos muestreados. Esto es lo que se conoce como aliasing.

En la siguiente figura, se muestra una onda sinusoidal de 800 kHz muestreada a 1 MS/s. La línea discontinua indica la señal de alias registrada en esa velocidad de muestreo. La frecuencia de 800kHz usa un alias de vuelta en el pasobanda, apareciendo falsamente como una onda sinusoidal de 200 kHz.

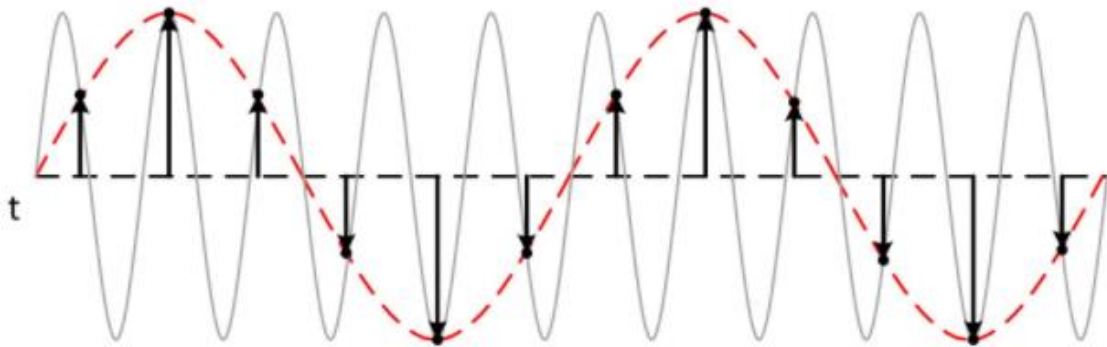


Figura 29: Señal temporal donde se aprecia aliasing por utilizar una frecuencia de muestreo baja [39].

En cuanto a frecuencia, en el espectro veríamos que se solaparía nuestra señal perdiendo información. De esta manera la señal no se reconstruye sin errores.

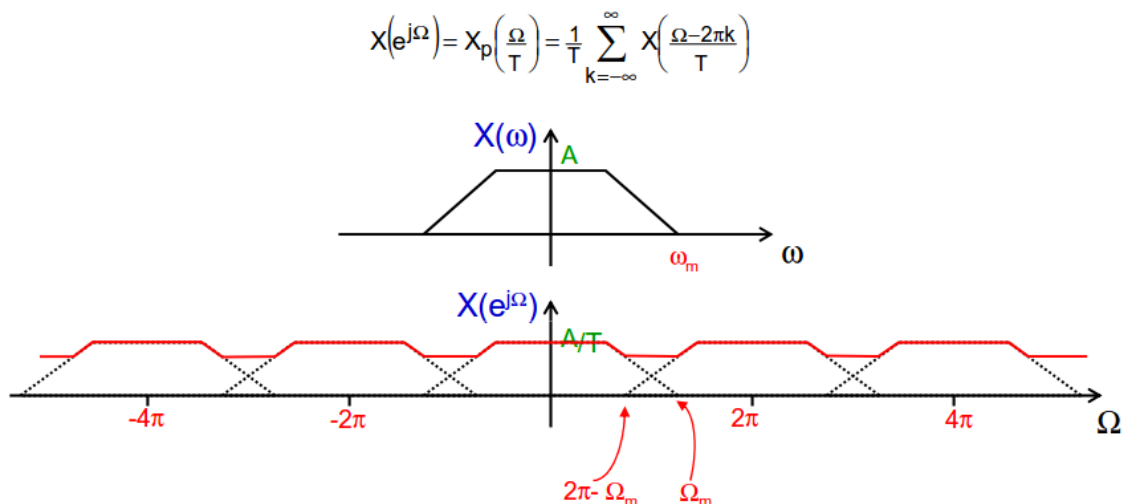


Figura 30: Espectro de la señal muestreada donde se aprecia Aliasing.

Para prevenir el aliasing se utiliza un filtro antialiasing. Este es un filtro paso bajo que atenúa cualquier frecuencia en la señal de entrada que sea mayor a la frecuencia Nyquist y esta antes del ADC para restringir el ancho de banda de la señal de entrada y cumplir con los criterios de muestreo.

A diferencia de antes, que se había mostrado la imagen del espectro de la señal muestreada donde se apreciaba aliasing, ahora, en la siguiente imagen se puede apreciar la corrección de este mediante el filtro antialiasing.

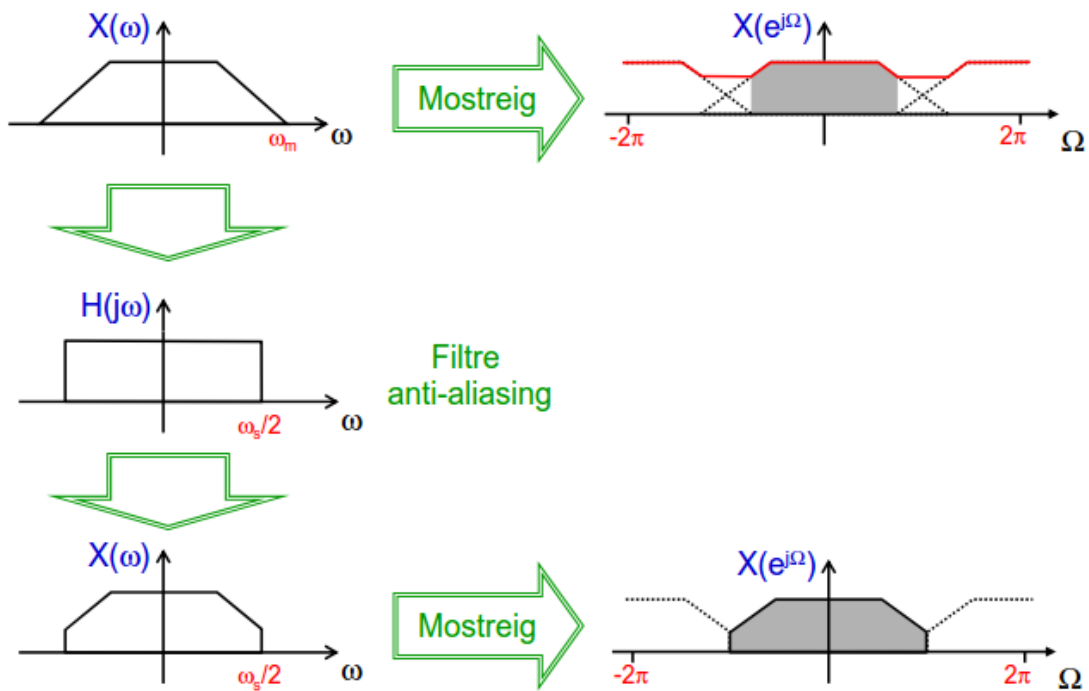


Figura 31: Resultado de utilizar filtro antialiasing.

ANEXO 10: Verificación del moduladora de cuadratura

Para verificar el correcto funcionamiento se procede a enviar tonos únicos. Esto consiste en suministrar tonos que están desfasados 90 grados en las entradas I y Q.

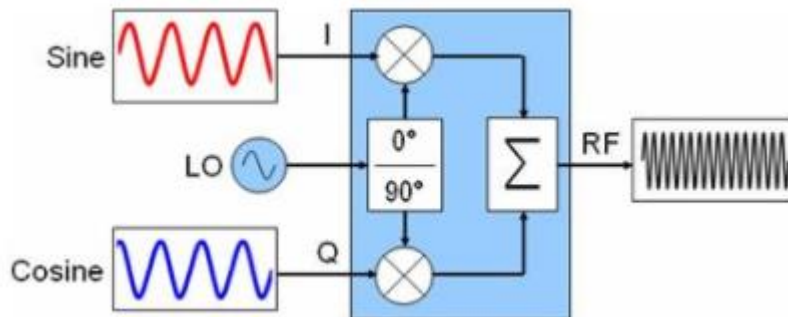


Figura 32: Caracterización del modulador en cuadratura [40].

El resultado de realizar esto debería ser un tono único a la frecuencia que queremos transmitir desplazada sumándole la frecuencia del seno.

Se puede demostrar matemáticamente, pero es suficiente representar esto gráficamente también en la siguiente figura.

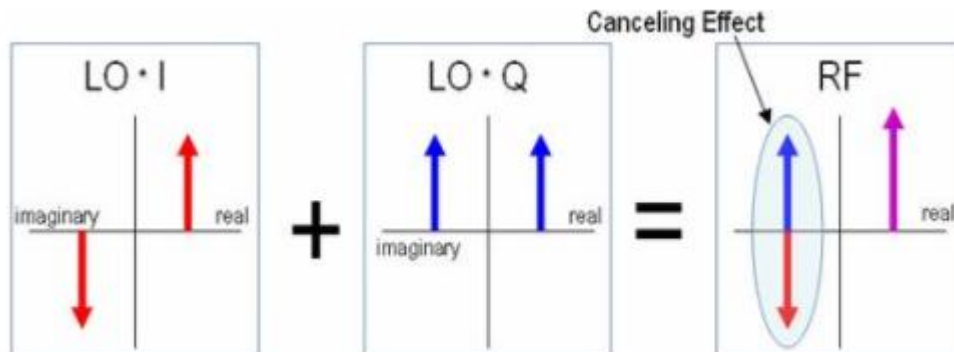


Figura 33: Representación gráfica de la cancelación de la imagen [40].

En sistemas reales, errores tales como desalineamiento de cuadratura y IQ evitarán la cancelación por completo.

ANEXO 11: Guía de configuración y ejecución de las VNFs

1. Ejecutar la conexión entre dos VNFs en un mismo ordenador.

```
REMOTE - Odroid: ./VNF_name -out0 127.0.0.1:8889 -in0 127.0.0.1:8888 -c_out  
127.0.0.1:5555 -c_in 127.0.0.1:4444
```

```
Server: ./VNF_name -out0 127.0.0.1:8888 -in0 127.0.0.1:8889 -c_out 127.0.0.1:4444 -c_in  
127.0.0.1:5555
```

2. Ejecutar la conexión entre dos VNFs en diferentes ordenadores.

```
REMOTE - Odroid: ./VNF_name -out0 192.168.10.3:8889 -in0 192.168.10.50:8888 -c_out  
192.168.10.3:5555 -c_in 192.168.10.50:4444
```

Preparar la configuración para este escenario: `sudo nano /etc/network/interfaces`

```
auto eth0  
iface eth0 inet static  
address 192.168.10.50  
netmask 255.255.255.0  
network 192.168.10.0  
broadcast 192.168.1.255  
gateway 192.168.10.1
```

```
sudo /etc/init.d/networking restart
```

```
Server: ./VNF_name -out0 192.168.10.50:8888 -in0 192.168.10.3:8889 -c_out  
192.168.10.50:4444 -c_in 192.168.10.3:5555
```

Preparar la configuración para este escenario: `sudo nano /etc/network/interfaces`

```
auto eth0  
iface eth0 inet static  
address 192.168.10.3  
netmask 255.255.255.0  
network 192.168.10.0  
broadcast 192.168.1.255  
gateway 192.168.10.1
```

```
sudo /etc/init.d/networking restart
```