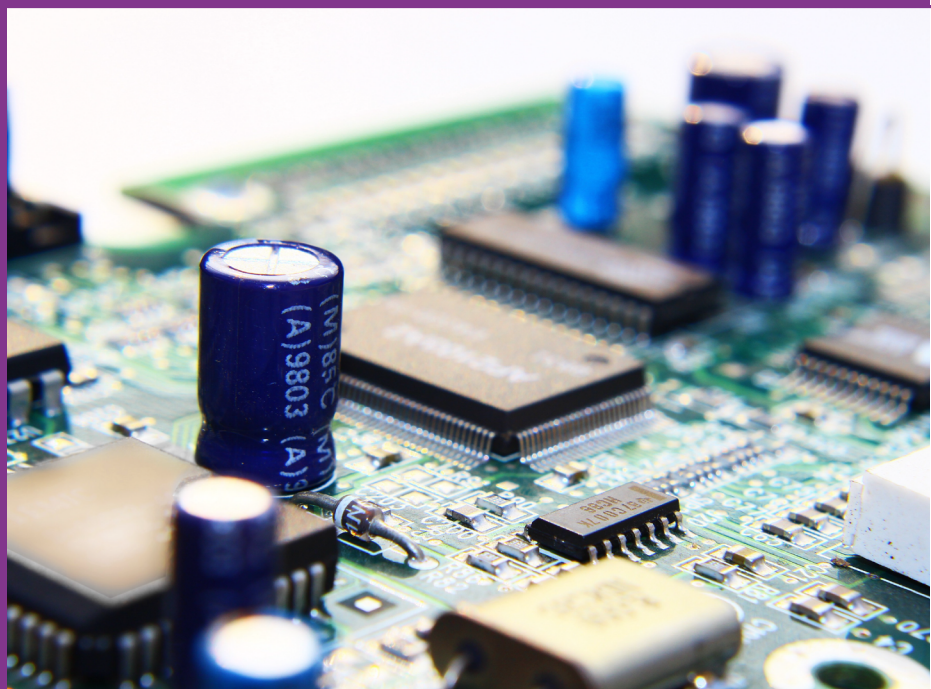


## Reporte de investigación (Informe técnico)



**Empleo del trans-receptor AD936x  
y una plataforma SoC,  
como banco de pruebas,  
para el desarrollo  
de aplicaciones  
Software Defined  
Radio**



Dr. Eduardo Abel Peñolosa Castro  
**Rector General de la UAM**

Dr. José Mariano García Garibay  
**Rector de la UAM Unidad Lerma**

**División de Ciencias Básicas e Ingeniería**

Dr. Edgar López Galván  
**Director de División**

Dr. Carlos Eduardo Díaz Gutiérrez  
**Secretario Académico**

Dr. Gerardo Abel Laguna Sánchez  
**Jefe del Departamento de  
Procesos Productivos**

Dr. Yuri Reyes Mercado  
**Jefe del Departamento de  
Recursos de la Tierra**

Dr. Guillermo López Maldonado  
**Jefe del Departamento de Sistemas  
de Información y Comunicaciones**

**Primera edición 2020**

© Universidad Autónoma Metropolitana Unidad Lerma  
Av. de las Garzas 10, Col. El Panteón, CP 52005,  
Lerma de Villada, México.

Diseño editorial: LDG José Uriel Hernández Pérez



Dirección  
Ciencias Básicas  
e Ingeniería

# **Empleo del trans-receptor AD936x y una plataforma SoC, como banco de pruebas, para el desarrollo de aplicaciones Software Defined Radio**

**Gerardo A. Laguna Sánchez  
Jacobo Sandoval Gutiérrez**

## CONTENIDO

6	<b>I. Resumen</b>
7	<b>II. Introducción</b>
10	<b>III. Metodología empleada</b>
11	<b>Capítulo 1. Marco teórico y conceptual</b>
11	La tecnología de súper-integración altamente flexible
12	Implicaciones de la tecnología HFSI en el desarrollo del concepto SDR
13	La evolución de los trans-receptores de RF
19	<b>Capítulo 2. Sistema propuesto para banco de pruebas SDR</b>
20	Tarjeta de desarrollo ZedBoard
21	Módulo trans-receptor AD-FMCOMMS4-EBZ
22	Descripción funcional de los trans-receptores 936x
25	Configuración del hardware desde Linux
25	Control de oscilador local
26	Control de la máquina de estados
26	Señales de banda base en la ruta de recepción
28	Señales de banda base en la ruta de transmisión
30	Control de los filtros digitales FIR
32	Control del ancho de banda de RF para la recepción
33	Control del ancho de banda de RF en la transmisión
34	<b>Capítulo 3. Puesta a punto del banco de pruebas SDR</b>
34	Pasos para la descarga e instalación de la imagen de Linux en tarjetas de desarrollo con trans-receptores AD-FMCOMMSx-EBZ



39	Instalación de biblioteca Libiio y compilación de códigos de ejemplo
40	El código de referencia ad9361-iostream.c
41	La biblioteca libad9361-iio
42	<b>Capítulo 4. Aplicación semilla para SDR básico con el banco de pruebas propuesto</b>
42	Especificación general
46	<b>IV.- Resultados obtenidos</b>
47	<b>Capítulo 5. Conclusiones</b>
48	Anexo. A. Código principal para aplicación semilla SDR con modulación QAM
58	<b>V.- Bibliografía y referencias para consulta</b>



## I. RESUMEN

El concepto *Software Defined Radio* (SDR), en español *Radio Definido por Software*, es un término genérico que se refiere a sistemas de radio donde la mayor parte de la operación de la capa física, es decir la etapa encargada de la transmisión y recepción de las señales con las que se transportan los datos, se realiza con algoritmos de procesamiento digital de señales (DSP, por sus siglas en inglés). En la actualidad, el avance tecnológico permite contar con diferentes alternativas para la puesta en práctica del concepto SDR a un precio relativamente bajo. En este informe se describe la operación de los módulos trans-receptores de radio frecuencia (RF) de las series ADFMCOMMSx, soportados por los circuitos de la familia AD936x, en conjunto con plataformas de desarrollo para los dispositivos *System on Chip* (SoC) de la familia Zynq, y se presenta una aplicación demostrativa para ilustrar cómo es que este conjunto puede funcionar como banco de pruebas para el desarrollo de aplicaciones en el ámbito de las comunicaciones digitales, en general, y en el contexto del concepto *Software Defined Radio*, en particular.



## II. INTRODUCCIÓN

La expresión *Radio Definido por Software* o SDR, por sus siglas en inglés, es un concepto genérico que se refiere a sistemas de radio donde la mayor parte de la operación de la capa física se realiza con algoritmos de procesamiento digital de señales (DSP, por sus siglas en inglés), en vez de emplear hardware dedicado (Collins et al, 2018; Stewart et al, 2015). Esto es posible hoy gracias al considerable avance tecnológico que se ha logrado para la fabricación de dispositivos de procesamiento de datos, con alto desempeño computacional y gran flexibilidad para su configuración que, además, ocupan pequeñas superficies y tienen un costo relativamente bajo. Esta tecnología fue denominada, por el Dr. Tsugio Makimoto, como tecnología de *súper-integración altamente flexible*, o HFSI por sus siglas en inglés (Laguna, 2015).

Las ventajas de sustituir hardware por software en el diseño de los nuevos sistemas de comunicación digital son evidentes:

- Se minimizan los costos de producción, al eliminar bloques que requieren componentes físicos y ocupan espacio.
- Se maximiza la confiabilidad, pues al disminuir la cantidad de componentes físicos, también se minimiza la probabilidad de falla.
- Se pueden realizar diseños altamente flexibles, en virtud de que el software y los algoritmos empleados son 100% reconfigurables, incluso un equipo que ya está operando en campo.

Un SDR ideal sólo cuenta con un hardware mínimo, para el imprescindible bloque de transmisión y recepción de las señales electromagnéticas, lo que se conoce técnicamente como etapa de *front-end* y se conforma por una antena y un conversor analógico digital de alta velocidad. Las demás etapas, como la de sincronización, demodulación, decodificación, etc., se realizan mediante software, asumiendo que se cuenta con un procesador dedicado de alta velocidad.

En el estado actual de la tecnología, es posible acceder a dispositivos de adquisición de datos para RF de alta velocidad, como lo es el circuito trans-receptor AD9361, y también a plataformas de procesamiento altamente configurables y de alto desempeño computacional, como lo son los SoC de la familia Zynq, a un precio razonable para los individuos y las pequeñas organizaciones, con lo que cada vez está más cerca la posibilidad de concretar el SDR ideal al alcance de todos.

En este informe técnico, se presenta la descripción, con detalle suficiente, sobre el empleo de las tarjetas *mezzanine* trans-receptoras de las series ADFMCOMMS2-3 y ADFMCOMMS4, respectivamente soportadas por los circuitos AD9361 y AD9364, en conjunto con las plataformas de desarrollo para los dispositivos SoC de la familia Zynq, por ejemplo, las tarjetas ZedBoard y ZC702, conformado un banco de pruebas para el desarrollo de aplicaciones SDR.

Este informe técnico es uno de los productos generados como parte del desarrollo del Proyecto de Investigación “*Diseño de la capa física de un sistema de comunicación 5G*”, aprobado por el Consejo Divisional en su Sesión No. 80. El banco de pruebas, que se describe en este documento, tiene como objetivo principal servir de plataforma de experimentación y prueba para los algoritmos necesarios para la realización práctica de los bloques que conforman todo sistema de comunicación digital. Estos bloques incluyen, aunque no se limitan, a los siguientes módulos funcionales de un trans-receptor digital (Sklar, 2001):

- Formateo.
- Codificación.
- Modulación de pulso o banda base.
- Modulación pasa banda.
- Sincronización.
- Igualamiento (*Equalization*).

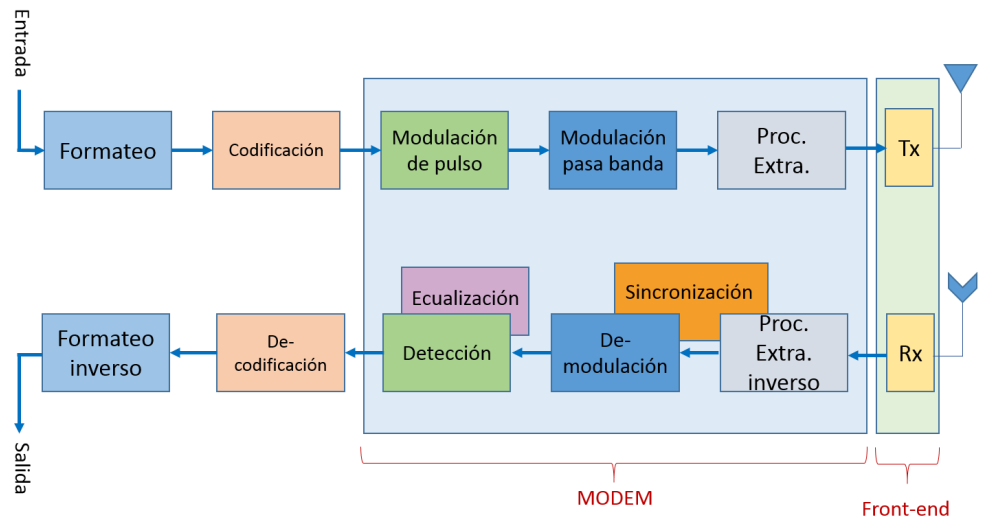


Figura II.1. El trans-receptor digital genérico a bloques

En la figura II.1 se muestra un diagrama a bloques de la configuración básica de un trans-receptor digital genérico, donde se puede apreciar la interrelación de los bloques funcionales mencionados.

Aunque es evidente que el banco de pruebas presentado es una herramienta de gran utilidad para alcanzar los objetivos del proyecto de investigación del que se deriva, es también importante mencionar que este mismo sistema se puede aprovechar, como apoyo didáctico en el proceso de enseñanza-aprendizaje y para las prácticas de laboratorio, dentro de la UEA Comunicaciones Digitales de los Planes y Programas vigentes de la Unidad Lerma y de cursos similares en otras instituciones.





## II. METODOLOGÍA EMPLEADA

Perteneciendo el proyecto de investigación a las ciencias aplicadas, la metodología de trabajo y de investigación está determinada por la naturaleza práctica del mismo. Se trata del diseño, codificación y prueba de los algoritmos desarrollados. Entonces la metodología de investigación comprende una secuencia de avances graduales, en cada uno de los que se deben cumplir las siguientes etapas antes de continuar:

- Desarrollo
- Verificación
- Validación

La verificación y validación de cada bloque funcional se realizó tomando como referencia a las especificaciones de un trans-receptor OFDM de propósito general.

El contenido de este informe técnico se presenta como sigue. En el Capítulo 1, se revisa el marco teórico y conceptual sobre la tecnología altamente integrada y flexible, así como su impacto en el desarrollo del concepto SDR. A continuación, en el Capítulo 2, se describe brevemente la conformación del sistema para el banco de pruebas propuesto, así como las características y los recursos más importantes que ofrecen los dos componentes clave del mismo: la tarjeta *mezzanine* ADFMCOMMSx, que incluye al trans-receptor de RF de alta velocidad de conversión, y la tarjeta de desarrollo con un SoC de la familia Zynq, para la realización del procesamiento digital de señales. En el Capítulo 3, se describe la configuración y la puesta a punto del banco de pruebas. En el Capítulo 4, se demuestra el uso del banco de pruebas para la realización práctica de un sistema básico de comunicación digital. Finalmente, en el Capítulo 5, se presentan nuestras conclusiones.



# Capítulo 1.

## Marco teórico y conceptual

### La tecnología de súper-integración altamente flexible

En los últimos 60 años, el desarrollo de la industria de la electrónica y, en particular, la de los semiconductores vino a confirmar, alcanzando las inmediaciones de los límites físicos impuestos por las leyes del universo sub-atómico, el potencial práctico de la denominada Ley de Moore, que pronosticó que, a más tardar, cada 24 meses se duplicaría el número de transistores en un circuito integrado. Esta tendencia permitió que la tecnología digital se haya vuelto omnipresente en los diversos ámbitos de nuestra vida cotidiana. En particular, la convergencia de los sistemas, a saber los de cómputo con las comunicaciones digitales y los sistemas de comunicación inalámbricos, ha revolucionado la industria y el mercado de las telecomunicaciones, así como la forma en que se conciben y realizan los nuevos equipos transmisores/receptores de radio frecuencia (*trans-receptores de RF*).

El Dr. Tsugio Makimoto, uno de los personajes más influyentes en la industria de los semiconductores, pronosticó que, alrededor del año 2017, surgiría lo que él denominó *tecnología de super-integración altamente configurable* o HFSI, por sus siglas en inglés (Laguna, 2015). Para propósitos de este documento, lo relevante es que, de acuerdo con el pronóstico de Makimoto, la era de la tecnología de super-integración altamente configurable ya inició. Y es que, de hecho, esto es lo que confirma la tendencia observada en los productos de los fabricantes más importantes de circuitos FPGA (por las siglas en inglés para la tecnología de *arreglos de compuertas lógicas programables en campo*) en los últimos años, así como en los desarrollos de la comunidad del SDR que ha aprovechado al máximo esta tecnología.

## Capítulo 1.

### Marco teórico y conceptual

Actualmente, nos encontramos en la tercera generación de los FPGA. Esta etapa inició alrededor del año 2010 y se ha caracterizado por incluir con estos dispositivos todos los elementos constructivos para realizar un sistema microprocesado completo, de tal forma que actualmente los FPGA son, de hecho, dispositivos SoC. Entonces, los FPGA de última generación incluyen, además de la lógica programable de aglutinamiento, dispositivos dedicados para procesamiento, memoria, periféricos y dispositivos de entrada/salida. Esto es precisamente el caso de los dispositivos pertenecientes a la familia Zynq del fabricante Xilinx, que cristalizan notablemente el concepto HFSI prefigurado por Makimoto. Estos dispositivos son descritos por el fabricante como dispositivos “SoC completamente programables” (*All programmable SoC*) y es que, literalmente, en estos dispositivos todo es programable, tanto el hardware, incluyendo los periféricos de E/S, como el propio software que se ejecuta sobre el hardware.

Así es como un dispositivo SoC típico de la familia Zynq es equivalente a un FPGA del tipo Artix o Kintex, con alrededor de 440,000 celdas lógicas, 2,020 bloques DSP, periféricos de E/S y un procesador de 32 bits, tipo RISC, de la familia ARM Cortex.

### **Implicaciones de la tecnología HFSI en el desarrollo del concepto SDR**

Actualmente, existe una amplia variedad de tarjetas de experimentación para desarrollar sistemas a partir de los dispositivos SoC de la familia Zynq. Estas tarjetas se encuentran en el mercado con especificaciones y configuraciones, que van desde las más básicas y económicas hasta las más sofisticadas y caras. Afortunadamente, para su aplicación en el desarrollo de sistemas SDR, funcionan muy bien las tarjetas de la gama más económica, entre las que se encuentra la tarjeta de desarrollo ZedBoard y, por supuesto, las tarjetas de gama media, entre las que se encuentra la tarjeta de desarrollo ZC702.

Ya se mencionó que los dos componentes clave del banco de pruebas propuesto en este documento son la tarjeta *mezzanine* ADFMCOMMSx, que incluye el trans-receptor RF de alta velocidad, y una tarjeta de desarrollo con un FPGA de la familia

## Capítulo 1.

### Marco teórico y conceptual

Zynq, que constituye al sistema de DSP. De acuerdo con lo expuesto, es evidente el papel que la tecnología HFSI ha tenido para que hoy contemos con plataformas de desarrollo, como las tarjetas Zedboard y ZC702, que permiten acceder a sistemas computarizados completos, embebidos en un solo chip, para el procesamiento de los datos recolectados, mediante tarjetas *mezzanine* de adquisición de datos que pueden conectarse directamente a estas tarjetas.

Por otro lado, respecto de la tarjeta trans-receptora RF de alta velocidad ADFMCOMMSx, el segundo componente fundamental del banco de pruebas propuesto, los beneficios de la tecnología HFSI tampoco son despreciables. El avance en el ámbito de los dispositivos de adquisición de datos no sólo se tradujo en el aumento exponencial de la densidad de integración, además del incremento en la flexibilidad para la configuración de estos dispositivos, sino que también se ha incrementado notablemente la velocidad de muestreo de los mismos. El notable incremento en la velocidad de muestreo en estos dispositivos es lo que, de hecho, permite acercarnos cada vez más al sistema SDR ideal.

### **La evolución de los trans-receptores de RF**

En el origen, todos los componentes y bloques funcionales de un trans-receptor de radio frecuencia se conformaban por elementos discretos de hardware analógico. Posteriormente, fue posible incluir parte de los circuitos analógicos en circuitos integrados, con los que el tamaño de los equipos disminuyó a la par que la confiabilidad aumentó. Finalmente, cuando el poder computacional disponible de los microprocesadores digitales se incrementó lo suficientemente, fue posible delegar parte del trabajo, realizado originalmente por componentes analógicos, a su contraparte digital. En este proceso, el papel de los convertidores, tanto el analógico-digital como el digital-analógico, ha sido fundamental en términos del progresivo incremento en la velocidad de muestreo.

Un sistema SDR ideal es aquel que cuenta con convertidores y velocidades de muestreo suficientemente altas para cumplir con el criterio de Nyquist, a fin de poder procesar las señales recibidas prácticamente desde el punto donde se conecta la

## Capítulo 1. Marco teórico y conceptual

antena. En la Figura 1.1 se muestra un diagrama a bloques del radio SDR ideal.

Nótese como el dominio analógico termina justo después de los amplificadores de RF y de los filtros anti-alias. Asumiendo que se cuenta con convertidores cuya velocidad de muestreo sea, por lo menos, dos veces la máxima frecuencia de RF, es decir, que se cumple con el criterio de muestreo de Nyquist, entonces es posible trabajar en el dominio digital sin necesidad de transportar o convertir las señales recibidas a una frecuencia menor.

En realidad, los primeros receptores digitales emplearon velocidades de muestreo muy bajas y progresivamente se ha venido avanzando en este terreno. Por ejemplo, en la actualidad son normales las comunicaciones de telefonía celular en el orden de magnitud de los GHz, por lo que un convertidor adecuado para concretar el SDR ideal tendría que muestrear en el orden de las decenas de los GHz, lo que tecnológicamente es un gran reto. Así, durante una parte importante del desarrollo del concepto SDR, se contó con opciones intermedias, en donde el principal recurso para solventar las limitaciones tecnológicas consistió en convertir las señales de interés a una frecuencia menor. A este proceso se le conoce como *heterodinización* y a los radios que pertenecen a esta modalidad se les conoce como radios *heterodinos*.

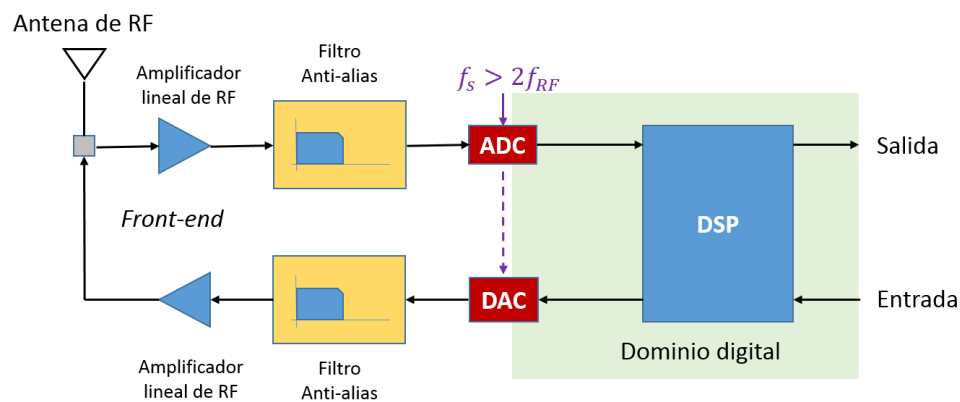


Figura 1.1. Sistema SDR ideal.



# Capítulo 1. Marco teórico y conceptual

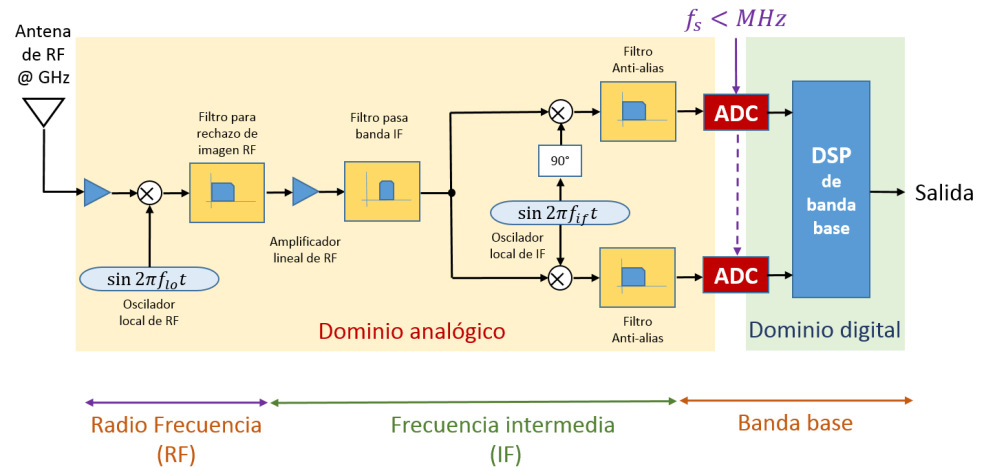


Figura 1.2. Sistema SDR heterodino de banda base.

En la figura 1.2 se muestra el diagrama a bloques de un radio receptor SDR heterodino que extiende el dominio analógico hasta el punto donde se tratan las señales con frecuencias de banda base y, por ello, se le conoce como radio heterodino de banda base. Esta configuración fue la más socorrida cuando las velocidades de muestreo de los convertidores no rebasaban el orden de los MHz. Se puede observar que el dominio analógico incluye un par de etapas con mezcladores. La primera etapa de mezcla, que se encuentra en la sección de radio frecuencia (RF), tiene como propósito de sintonizar la señal útil y recupera la señal de una frecuencia intermedia que transporta la información. La segunda etapa de mezcla, que se encuentra en la sección de frecuencia intermedia, forma parte de un receptor en cuadratura y permite recuperar las componentes de banda base. En este caso, es justo después del mezclador del receptor en cuadratura, que se colocan los convertidores para continuar con el procesamiento digital de las señales a partir de este punto.

Conforme la velocidad de muestreo de los convertidores aumentó, se pudo iniciar el procesamiento digital sobre señales de frecuencia intermedia, es decir entre RF y la banda base. En la Figura 1.3 se muestra el diagrama a bloques de un receptor heterodino de frecuencia intermedia. Nótese como el dominio digital comienza una vez que se recupera la señal de frecuencia intermedia, después del primer mezclador. Las

# Capítulo 1. Marco teórico y conceptual

etapas de recepción en cuadratura y el procesamiento de los datos quedan, en esta configuración, en el dominio digital.

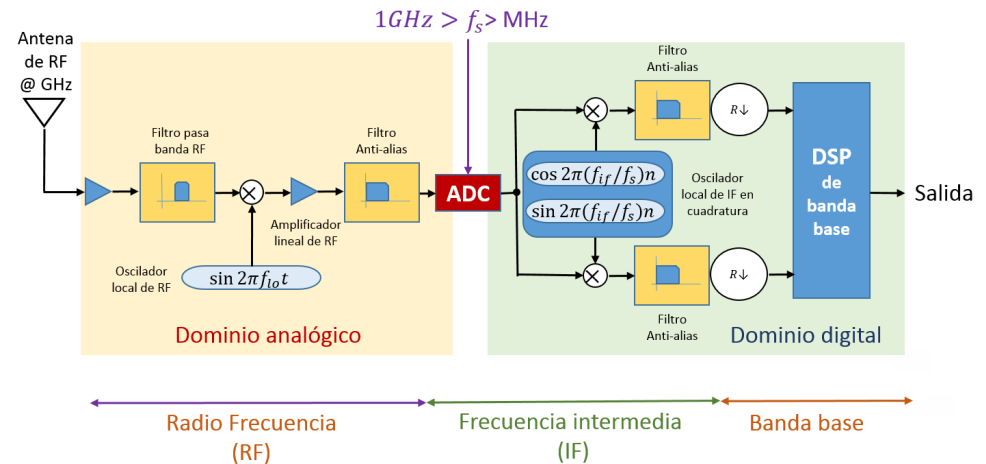


Figura 1.3. Sistema SDR heterodino de frecuencia intermedia.

Cuando el avance tecnológico lo permita y se cuente con convertidores suficientemente rápidos, llegará el momento en que, para un segmento importante del espectro de radio comunicaciones de muy alta frecuencia, ya no será necesario recurrir a técnicas de frecuencia intermedia y se procesará directamente la señal de RF. Este tipo de radio se denomina de conversión directa o, bien, radios de *cero IF*, también conocidos como homodinos. En la figura 1.4 se muestra el diagrama a bloques de un receptor de este tipo.

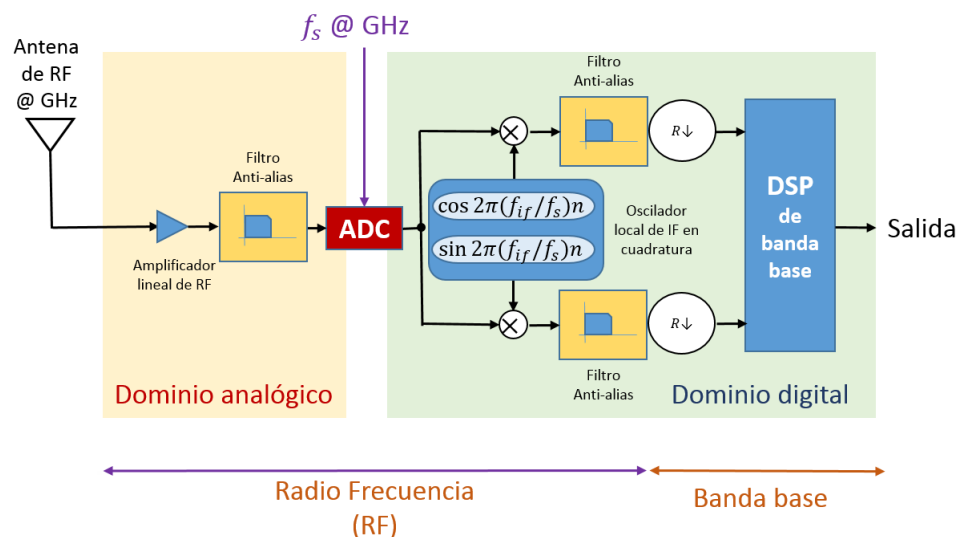


Figura 1.4. Sistema SDR homodino.

## Capítulo 1. Marco teórico y conceptual

El trans-receptor, que proponemos aquí para conformar el banco de pruebas, es el correspondiente al circuito integrado AD9361 o alguna de sus variantes. Se trata de una configuración intermedia entre el sistema SDR ideal y un SDR de banda base, pero sin recurrir a la frecuencia intermedia. Consiste en un trans-receptor en cuadratura, cuyo mezclador sintoniza a la señal de RF que trasporta la información y permite la conversión a las frecuencias de banda base. En la Figura 1.5 se muestra a un diagrama a bloques del circuito referido.

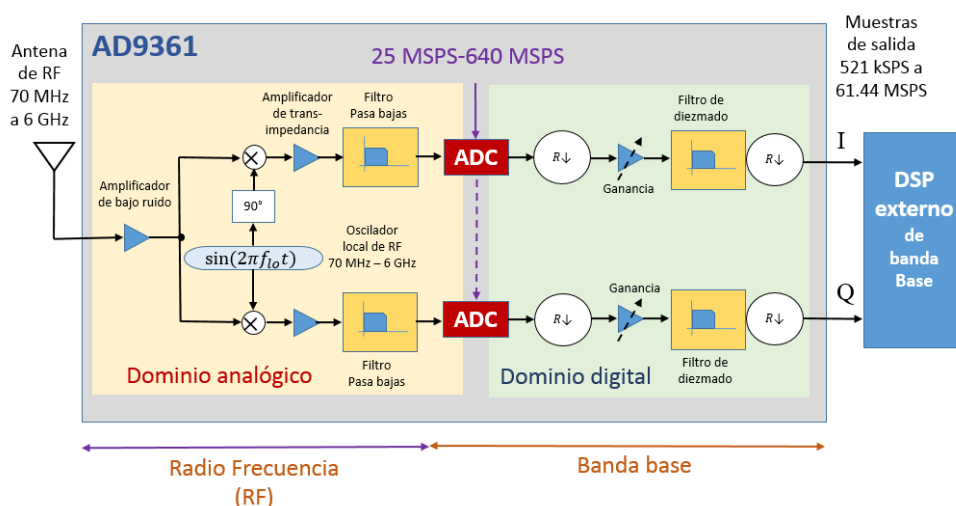


Figura 1.5. Diagrama a bloques del bloque receptor del circuito AD9361.

Como podemos apreciar en la Figura 1.5, se trata de un sistema de recepción en cuadratura, donde el oscilador local sintoniza la frecuencia de la portadora de RF y los mezcladores permiten recuperar las componentes moduladoras de banda base, tanto en fase, I, como en cuadratura, Q. Aquí, en concordancia con los diagramas anteriores, sólo se presenta la parte correspondiente a la sección de recepción, sin embargo, es importante mencionar que el circuito AD9361 también cuenta con la contraparte transmisora, dado que este dispositivo es realmente un trans-receptor.

Es importante hacer notar el rango de operación del circuito AD9361, dado que permite trabajar con señales de RF en el intervalo que va de los 70 MHz a los 6 GHz, en la sección de receptora, y de los 47 MHz a los 6 GHz, en la sección transmisora, así como señales de banda base que van de los

## Capítulo 1. Marco teórico y conceptual

260.5 kHz a 30.72 MHz, dado que este dispositivo cuenta con distintos bloques de división de frecuencia y diezmado (*decimation*), completamente configurables por el usuario, a fin de lograr la combinación óptima.

El circuito AD9361, en sí mismo, es un ejemplo palpable de la tecnología HFSI ya que integra, en un mismo dispositivo, de tan solo 1cm<sup>2</sup>, 144 terminales, la sección analógica, la sección digital, incluyendo los convertidores de alta velocidad y los osciladores necesarios, y más de 1000 registros de configuración para la puesta a punto del trans-receptor.

## Capítulo 2.

### Sistema propuesto para banco de pruebas SDR

Como ya se mencionó, la propuesta para el banco de pruebas, a fin de desarrollar aplicaciones SDR, se conforma por una tarjeta de desarrollo con un dispositivo FPGA-SoC de la familia Zynq, a fin de contar con un sistema de cómputo para el procesamiento digital de las señales (DSP) de banda base, y una tarjeta *mezzanine*, con el trans-receptor AD9361, que pueda conectarse al sistema DSP a fin de conformar la etapa front-end del sistema SDR. Así, el conjunto posibilita al usuario para recibir y transmitir señales de RF, realizando las correspondientes conversiones de alta velocidad, analógicas-digitales y digitales-analógicas, para finalmente, mediante procesos de diezmado e interpolado, intercambiar con el sistema de procesamiento de datos las secuencias correspondientes a las señales de banda base. En específico, se propone una tarjeta de desarrollo ZedBoard, o una tarjeta ZC702, para el sistema DSP, y una tarjeta *mezzanine* AD-FMCOMMS4-EBZ, que se monta directamente en uno de los conectores FMC LPC de la tarjeta de desarrollo, para fungir como *front-end* del sistema SDR (Beltrán-Obio, 2018). Esta configuración se presenta en la imagen de la Figura 2.1.

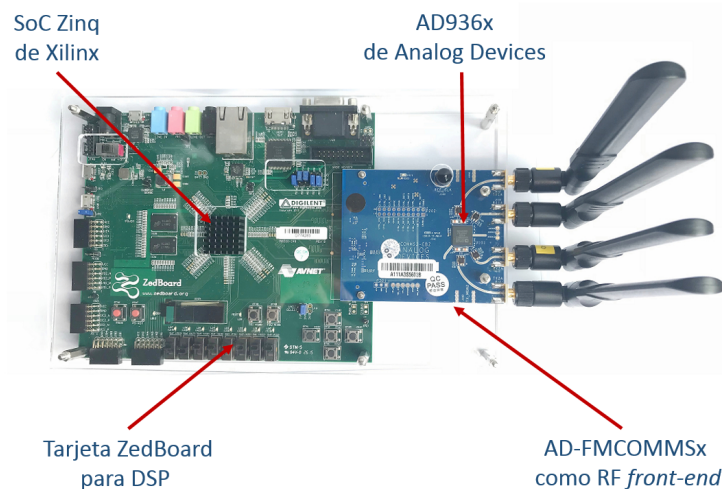


Figura 2.1. Imagen del banco de pruebas SDR a partir de una tarjeta de desarrollo ZedBoard y una tarjeta *mezzanine* AD-FMCOMMSx-EBZ montada sobre su conector FMC LPC.



## Tarjeta de desarrollo ZedBoard

La tarjeta ZedBoard es una plataforma de desarrollo de bajo costo (aproximadamente \$450.00 dólares) soportada por un dispositivo SoC de la familia Zynq-7000, del fabricante Xilinx, que incluye un sistema de procesamiento ARM además de lógica programable de última generación. Esta tarjeta contiene todo lo necesario para crear un sistema de procesamiento, sobre el que se puede montar un sistema operativo, obteniendo una computadora de propósito general que, en nuestro caso, permitirá la realización del procesamiento digital de señales. Esta tarjeta también cuenta con diferentes conectores para que el sistema de procesamiento DSP pueda acceder a dispositivos de entrada/salida.

Las características resumidas de la tarjeta son las siguientes (AVNET, 2014):

- Dispositivo Zynq-7000 SoC con el número de parte XC7Z020-CLG484-1.
- 512MB de memoria RAM.
- 256 MB de memoria Flash.
- Sistema de programación USB-JTAG incluido.
- Interfaz Ethernet.
- Conectores de expansión incluyendo conectores de bajo perfil FMC.
- Salidas para despliegue que incluyen HDMI, VGA y pantalla OLED.
- CODEC para audio.
- Socket para tarjeta de memoria externa SD.

Así mismo, la tarjeta cuenta con múltiples recursos de apoyo y diseños de referencia para el desarrollo de aplicaciones. Todo el material de consulta se encuentra disponible en línea y ha sido compartido por la comunidad de desarrolladores, por ejemplo, en el sitio **[www.zedboard.org](http://www.zedboard.org)**, así como por los fabricantes de tarjetas *mezzanine* compatibles. Tal es el caso de la compañía Analog Devices, que es precisamente la que produce y da soporte a los módulos trans-receptores AD-FMCOMMS2-EBZ y AD-FMCOMMS4-EBZ a través de su sitio **<https://wiki.analog.com>**

## Módulo trans-receptor AD-FMCOMMS4-EBZ

La tarjeta mezzanine AD-FMCOMMS4-EBZ es una herramienta que facilita el desarrollo y prueba de diseños con el circuito AD9364. Esta tarjeta posibilita su acceso a una plataforma de procesamiento de datos mediante su conector LPC FMC y cuenta con el hardware mínimo que la habilita para fungir como etapa *front-end* en el desarrollo de un sistema RF de comunicaciones digitales (Analog Devices, 2019).

La principal característica de este módulo trans-receptor es su capacidad para operar en un amplio espectro de frecuencias que va, para propósitos prácticos, de los 70 MHz a los 6 GHz. Aunque esta tarjeta es similar a los modelos AD-FMCOMMS2-EBZ y AD-FMCOMMS3-EBZ, se diferencia en los siguientes aspectos:

1. En vez de emplear el circuito AD9361, que cuenta con dos canales de recepción y dos de transmisión, utiliza el circuito AD9364, que sólo cuenta con un canal receptor y un canal transmisor.
2. Cuenta con dos tipos de transformadores de acoplamiento de RF, denominados *baluns*: uno diseñado para aplicaciones con un amplio intervalo de sintonización y otro con desempeño óptimo en la frecuencia de 2.4 GHz.

Los principales componentes de la tarjeta AD-FMCOMMS4-EBZ son los siguientes:

- Circuito trans-receptor AD9364.
- Circuitos de alimentación ADP1755 y ADP2164.
- Circuito EEPROM M24C02 para almacenamiento de la configuración.
- Circuito ADC AD7291, con 8 canales, interfaz I2C y sensor de temperatura.
- Cristal de 40MHz como el reloj del sistema.
- Control SPI, directamente cableado al conector FMC.

La tarjeta cuenta con un amplio contenido de recursos de apoyo en línea en el sitio <https://wiki.analog.com>. En particular, es de gran utilidad la imagen con el sistema operativo Linux, ofrecido gratuitamente por la compañía Analog Devices, para dar soporte al control y acceso del módulo trans-receptor, desde la plataforma de desarrollo en la que se monta, por

ejemplo, las tarjetas de desarrollo ZedBoard y ZC702, entre otras. Este sistema operativo no sólo incluye los controladores específicos para el hardware de la tarjeta de desarrollo en el que se ejecuta sino, además, los controladores para las tarjetas AD-FMCOMMSx-EBZ. También se encuentra disponible la biblioteca **Libiio**, que permite controlar y acceder al circuito trans-receptor AD9364, mediante una interfaz estándar de software conocida como *Industrial I/O* (IIO), que hace abstracción de los detalles de bajo nivel del hardware y proporciona una interfaz de programación simple para el desarrollador de aplicaciones avanzadas en un tiempo relativamente corto.

### Descripción funcional de los trans-receptores 936x

En la Figura 2.2 se presenta un diagrama a bloques con los principales módulos funcionales de los trans-receptores de las series 936x de Analog Devices. Se puede apreciar que el dispositivo cuenta con dos rutas de datos principales:

- La ruta de recepción (*Rx Path*). El recorrido que realiza la señal recibida y las correspondientes secuencias de datos digitales.
- La ruta de transmisión (*Tx Path*). El recorrido que realizan las secuencias de datos a transmitir y las señales correspondientes.

También, se puede apreciar que cada ruta cuenta con varios canales (*Channels*). Así, por ejemplo, el dispositivo AD9361 cuenta con dos canales para la ruta de recepción y dos canales para la ruta de transmisión, mientras que el AD9364 sólo cuenta con uno para cada ruta.

## Capítulo 2. Sistema propuesto para banco de pruebas SDR

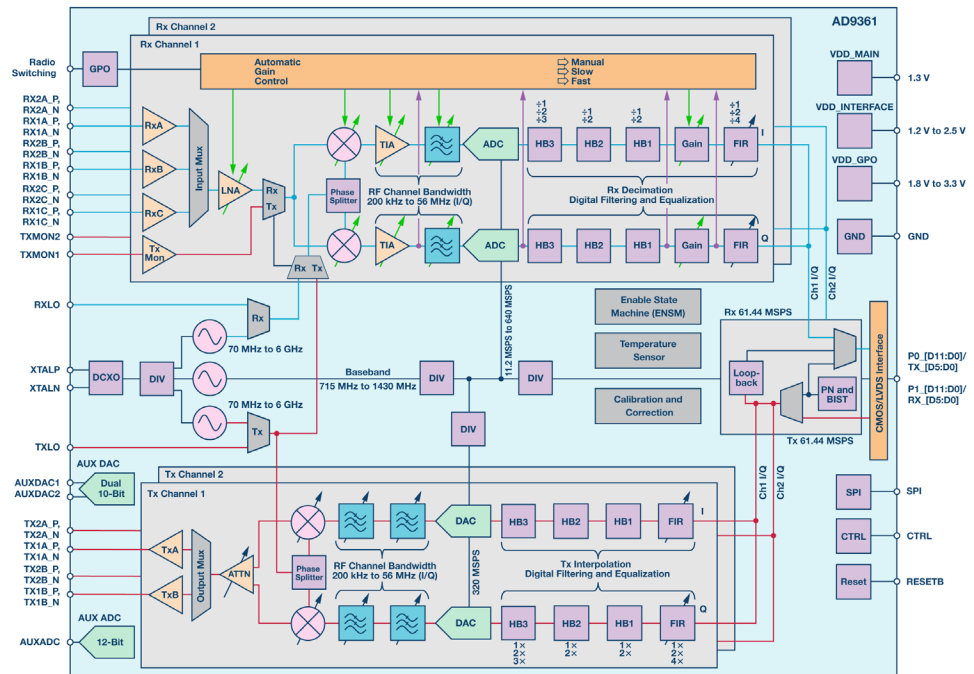


Figura 2.2. Diagrama a bloques del trans-receptor AD9361 y variantes (Pu, 2015).

(Fuente: <http://www.analog.com/en/analog-dialogue/articles/using-model-based-design-sdr-1.html>)

Nótese que ambas rutas, la de recepción y la de transmisión, dependen de los servicios de los módulos de alimentación, los osciladores y los dispositivos de sincronización. En esta parte, fundamental para el funcionamiento del trans-receptor, encontramos al módulo para el control digital del cristal de oscilación (identificado como DCXO en la figura 2.2), los bloques de lazo anclado a fase (PLL, por sus siglas en inglés), así como los divisores de frecuencia (identificados con las siglas DIV) que proporcionan las señales de referencia y tiempo.

Cada canal, a su vez, puede elegir entre diferentes puertos. Así, por ejemplo, el dispositivo AD9361 cuenta con tres puertos disponibles para la ruta receptora (identificados como RxA, RxB y RxC), mientras que la ruta transmisora cuenta con dos puertos (identificados como TxA y TxB).

La ruta de recepción inicia con la selección del puerto de entrada (RxA, RxB o RxC), le sigue un amplificador de bajo ruido (LNA, por sus siglas en inglés) y, a continuación, la etapa del mezclador en cuadratura (*downconversion*), de que se

## Capítulo 2. Sistema propuesto para banco de pruebas SDR

derivan las ramas en fase (I) y en cuadratura (Q). Cada rama cuenta con un amplificador de trans-impedancia (TIA, por sus siglas en inglés) y un filtro pasa bajas, antes del convertidor analógico-digital (ADC, por sus siglas en inglés). Después, cada rama cuenta con tres filtros de media banda en cascada (identificados como HB3, HB2 y HB1), que permiten realizar el trabajo de diezmado (*decimation*) para poder entregar las secuencias de datos de banda base con la velocidad de muestreo especificada. Finalmente, cada rama cuenta con un filtro de respuesta al impulso finita (FIR, por sus siglas en inglés), que es completamente configurable por el usuario y que puede ser usado como etapa de filtrado digital o para propósitos de igualamiento (*equalization*).

La ruta de transmisión inicia con el filtro FIR y tres filtros de media banda en cascada, en este caso, para propósitos de interpolación, hasta alcanzar la velocidad de muestreo requerida para su conversión a señal analógica, mediante el convertidor digital-analógico (DAC, por sus siglas en inglés). A continuación, se encuentra la etapa de filtrado que delimita la banda de RF, justo antes del mezclador de modulación en cuadratura. La señal mezclada se somete a una etapa de atenuación configurable (identificada como ATTN) y se concluye con la selección del puerto de salida (identificados como TxA o TxB).

En la siguiente sección se presentará una breve descripción de la interfaz de control para los trans-receptores de la serie AD936x, mediante mandos en línea, desde el sistema operativo Linux, que forman parte de las herramientas disponibles con el soporte de los controladores conforme al estándar IIO (*Industrial I/O*). La información completa se puede consultar en línea en las siguientes direcciones:

<https://wiki.analog.com/resources/tools-software/linux-drivers/iio-transceiver/ad9361>

y

<https://wiki.analog.com/resources/tools-software/linux-software/libiio>



## Configuración del hardware desde Linux

Para propósitos de control y de configuración, los trans-receptores de la serie AD936x pueden ser vistos como dispositivos IIO (*IIO devices*). En el entorno Linux, cada dispositivo IIO cuenta con una carpeta con la siguiente ruta:

```
/sys/bus/iio/devices/iio:deviceX
```

Donde X es el índice que corresponde al dispositivo. En cada una de estas carpetas se encuentran los archivos con los registros de las configuraciones correspondientes. Un primer parámetro de interés para el usuario es el nombre del dispositivo asociado a una carpeta IIO. Para ello, en la ruta de la carpeta en cuestión, simplemente se recupera el registro del archivo *name* mediante el mando *cat*. Por ejemplo:

```
root: /> cd /sys/bus/iio/devices/  
root:/sys/bus/iio/devices> ls  
iio:device0 iio:device1 iio:device2 iio:device3 iio:device4  
  
root:/sys/bus/iio/devices> cd iio:device1  
root:/sys/bus/iio/devices/iio:device1> cat name  
ad9361-phy
```

En general, la lectura de los parámetros de configuración se realiza con apoyo del mando *cat*, mientras que la escritura se realiza con apoyo del mando *echo*. Para fines demostrativos, en las siguientes subsecciones se hace una rápida revisión de los parámetros de operación más utilizados.

## Control de oscilador local

Los trans-receptores AD936x cuentan con dos sintetizadores idénticos PLL de RF (RFPLL) para la generación de las señales locales de referencia (LO). Uno es para el canal de recepción (Rx) y el otro para el canal de transmisión (Tx). Así, por ejemplo, para la configuración de la frecuencia de recepción tenemos:

```
cat out_altvoltage0_RX_LO_frequency  
2400000000
```

```
root:/sys/bus/iio/devices/iio:device1> echo 2450000000 >  
out_altvoltage0_RX_LO_frequency  
root:/sys/bus/iio/devices/iio:device1> cat out_altvoltage0_  
RX_LO_frequency  
2450000000
```

## Control de la máquina de estados

El trans-receptor opera gracias a una máquina de estados que permite el control del mismo en tiempo real. La máquina de estados permite realizar operaciones, tanto en la modalidad de Duplexado por División de Frecuencia (FDD, por sus siglas en inglés), que permite comunicación *Full-duplex*, como en la modalidad de Duplexado por División de Tiempo (TDD, por sus siglas en inglés), que permite comunicación *Half-duplex*.

## Señales de banda base en la ruta de recepción

Las señales recibidas en cuadratura (I y Q) son recuperadas, mediante el mezclador de bajada (*downconversion*), para ser procesadas en la sección de banda base. La ruta de recepción para las señales de banda base se compone por dos filtros analógicos, pasa bajos y configurables, un ADC de 12 bits y cuatro etapas de filtros de diezmado, para el muestreo descendente (*down sampling*), que pueden configurarse para servir únicamente de paso (puenteadas). Así mismo, puede ser configurada la frecuencia de corte de los filtros pasa-bajos.

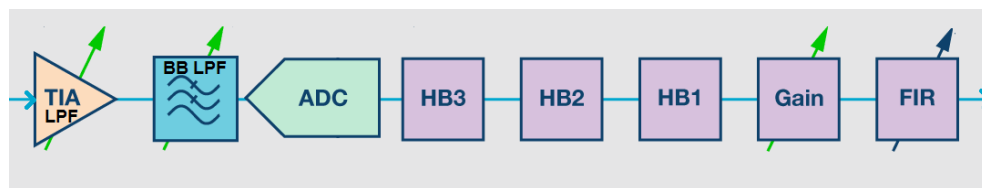


Figura 2.3. Ruta de recepción en banda base.

El parámetro denominado *in\_voltage\_sampling\_frequency* permite definir, con resolución de Hz, la velocidad de muestreo para la señal de banda base en el intervalo de 521 kSPS a 61.44 MSPS. Una vez definido este parámetro, se ajustan en forma automática la frecuencia del PLL de banda base (BBPLL), así como las velocidades de muestreo del ADC y de

## Capítulo 2.

### Sistema propuesto para banco de pruebas SDR

las etapas de diezmado. A continuación tenemos un ejemplo de configuración de la velocidad de muestreo para las señales de banda base en la ruta de recepción:

```
root:/sys/bus/iio/devices/iio:device1> cat in_voltage_
sampling_frequency
30720000
```

```
root:/sys/bus/iio/devices/iio:device1> echo 10000000 > in_
voltage_sampling_frequency
```

```
root:/sys/bus/iio/devices/iio:device1> cat in_voltage_
sampling_frequency
10000000
```

Una vez, que queda definida la velocidad de muestreo de las señales de banda base dentro de la ruta de recepción, el usuario puede conocer las relaciones que guardan todas las velocidades de muestreo mediante la siguiente consulta:

```
root:/sys/bus/iio/devices/iio:device1> cat rx_path_rates
BBPLL:983040000 ADC:245760000 R2:122880000 R1:61440000
RF:30720000 RXSAMP:30720000
```

Donde los parámetros referidos y sus relaciones se muestran en la Tabla 1.

Tabla 1. Velocidades de muestreo en banda base de la ruta de recepción.

ETAPA	DESCRIPCIÓN	RELACIÓN ORIGINARIA
BBPLL	Frecuencia del PLL de banda base.	
ADC	Velocidad de muestreo del ADC.	Divisor del BBPLL= BBPLL/ADC
R2	Tasa de diezmado del filtro HB3.	Factor de diezmado del HB3= ADC/R2
R1	Tasa de diezmado del filtro HB2.	Factor de diezmado del HB2= R2/R1
RF	Tasa de diezmado del filtro HB1.	Factor de diezmado del HB1= R1/RF
RXSAMP	Velocidad de muestreo de banda base en la recepción.	Factor de diezmado del FIR= RF/RXSAMP

## Capítulo 2. Sistema propuesto para banco de pruebas SDR

### Señales de banda base en la ruta de transmisión

Las señales que se alimentan al trans-receptor para ser transmitidas, tanto en la rama I como en la Q, son valores de 12 bits codificados en complemento a 2. Estos datos transitan, a través de los cuatro filtros de interpolación, hasta llegar al DAC de 12 bits. Cada uno de estos filtros también puede ser puenteado. A su vez, la salida del DAC se pasa por dos filtros pasa bajas antes de alimentarse al mezclador de RF. La frecuencia de corte de esta etapa de filtrado también puede ser configurada. Los cuatro filtros de la etapa de muestreo ascendente (*up sampling*) se emplean tanto para limitar el ancho de banda como para interpolar los datos antes de la conversión digital-analógica en el DAC.

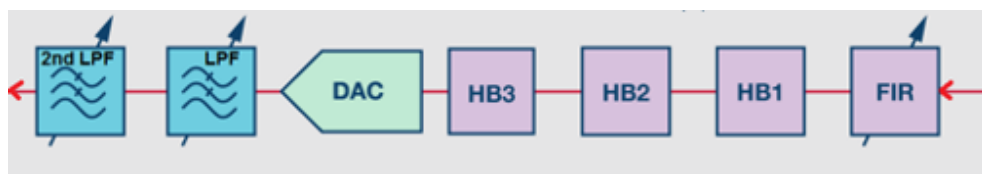


Figura 2.4. Ruta de transmisión en banda base.

En este caso, es el parámetro *out\_voltage\_sampling\_frequency* el que permite definir la frecuencia de muestreo para las señales de banda base de la ruta de transmisión, con resolución de Hz, en el intervalo de 218 kSPS a 61.44 MSPS. Con ello, de manera automática, se ajusta la frecuencia del BBPLL, la velocidad de muestreo del DAC y la correspondiente a las etapas previas de filtrado. Aunque, en principio, la ruta de transmisión puede operar a la mitad de la velocidad de muestreo de la ruta de recepción, la realidad es que esto está limitado por el hardware donde se monta el módulo trans-receptor. A continuación tenemos un ejemplo de configuración de la velocidad de muestreo para las señales de banda base en la ruta de recepción:

```
root:/sys/bus/iio/devices/iio:device1> cat out_voltage_
sampling_frequency
30720000

root:/sys/bus/iio/devices/iio:device1> echo 10000000 > out
_voltage_sampling_frequency

root:/sys/bus/iio/devices/iio:device1> cat out _voltage_
sampling_frequency
10000000
```

Una vez, que queda definida la velocidad de muestreo de las señales de banda base dentro de la ruta de transmisión, el usuario puede conocer las relaciones que guardan todas las velocidades de muestreo, mediante la siguiente consulta:

```
root:/sys/bus/iio/devices/iio:device1> cat tx_path_rates
BBPLL:983040000 DAC:122880000 T2:122880000 T1:61440000
TF:30720000 TXSAMP:30720000
```

Donde los parámetros referidos y sus relaciones se muestran en la Tabla 2.

Tabla 2. Velocidades de muestreo en banda base de la ruta de transmisión.

ETAPA	DESCRIPCIÓN	RELACIÓN ORIGINARIA
BBPLL	Frecuencia del PLL de banda base.	
DAC	Velocidad de muestreo del DAC.	Divisor para el DAC= ADC/DAC
T2	Tasa de diezmado del filtro HB3.	Factor de interpolación del HB3= DAC/T2
T1	Tasa de diezmado del filtro HB2.	Factor de interpolación del HB2= T2/T1
TF	Tasa de diezmado del filtro HB1.	Factor de interpolación del HB1= T1/TF
TXSAMP	Velocidad de muestreo de banda base en la transmisión.	Factor de interpolación del FIR= TF/TXSAMP

## Control de los filtros digitales FIR

Los filtros FIR, tanto en la ruta de recepción como en la de transmisión, son completamente configurables por el usuario pero también se pueden puentear. Estos filtros se aprovechan para realizar etapas de diezmado/interpolación con factores 1, 2 o 4. Así mismo, el número de coeficientes del filtro (*taps*) puede ser definido entre 16 y 128, en saltos de 16. El filtro FIR de la ruta de recepción incluye una etapa de ganancia que puede ser configurada con los valores -12 dB, -6 dB, 0 dB o +6 dB. Dado que este filtro, por sí solo, introduce una ganancia de 6dB, es común que la ganancia del filtro se ajuste a -6 dB.

Los coeficientes de estos filtros deben calcularse, en concordancia con las frecuencias de muestreo de las señales de banda base y las frecuencias de corte deseadas. Dado que el diseño de estos coeficientes y las relaciones entre las velocidades de muestreo dependen directamente de los parámetros de operación, de las relaciones de velocidades de muestreo y de las frecuencias de los circuitos PLL, se recomienda que estos coeficientes se calculen con las herramientas autorizadas. Dichas herramientas consideran las restricciones del sistema y generan configuraciones válidas. En este sentido, aquí hacemos referencia a la biblioteca de funciones, en código C, que Analog Devices proporciona y que denomina como proyecto **libad9361-iio**. Las funciones de la biblioteca *libad9361-iio* no solo permiten calcular los coeficientes de los filtros FIR sino, incluso, descargarlos directamente, desde el código en C, hacia el módulo trans-receptor. Los detalles de dicha herramienta se pueden encontrar en el sitio web:

<https://github.com/analogdevicesinc/libad9361-iio>

Por lo pronto, para conocer la configuración que actualmente se encuentra activa, desde la línea de mandos Linux, el usuario puede dar entrada a la siguiente secuencia:

```
root:/sys/bus/iio/devices/iio:device1> cat filter_fir_config
FIR Rx 0,0 Tx 0,0
```



## Capítulo 2. Sistema propuesto para banco de pruebas SDR

Si se desea cargar una nueva configuración, también desde la línea de mandos, el usuario primero debe almacenar la especificación del diseño para los filtros FIR en un archivo de texto con la siguiente sintaxis:

```
RX [1,2,3(both)] GAIN [-12, -6, 0, 6] DEC [1,2,4]
TX [1,2,3(both)] GAIN [-6, 0] INT [1,2,4]
[RRX <BBPLL ADC R2 R1 RF RXSAMP>]
[RTX <BBPLL DAC T2 T1 TF TXSAMP>]
[BWRX <RX RF Bandwidth>]
[BWTX <RX TF Bandwidth>]
<tx-coef0, rx-coef0>
<tx-coef1, rx-coef1>
<..., ...>
<tx-coefN-1, rx-coefN-1>
```

Por ejemplo:

```
#RX [1,2,3(both)] GAIN [-12, -6, 0, 6] DEC [1,2,4]
#TX [1,2,3(both)] GAIN [-6, 0] INT [1,2,4]
#tx,rx
RX 3 GAIN -6 DEC 2
TX 3 GAIN 0 INT 2
RTX 983040000 245760000 245760000 122880000 61440000
30720000
RRX 983040000 491520000 245760000 122880000 61440000
30720000
BWTX 19365438
BWRX 19365514
-5,49
0,217
--continúa...--
```

## Capítulo 2. Sistema propuesto para banco de pruebas SDR

Supongamos que el usuario desea configurar a los filtros FIR del módulo trans-receptor y que el archivo que contiene la especificación de diseño se encuentra en la misma carpeta y se llama *myConfig.txt*. Entonces la descarga y verificación se realiza mediante la siguiente secuencia de mandos:

```
root:/sys/bus/iio/devices/iio:device1> cat myConfig.txt>
filter_fir_config

root:/sys/bus/iio/devices/iio:device1> echo 1 > in_voltage_
filter_fir_en

root:/sys/bus/iio/devices/iio:device1>cat in_voltage_filter_
fir_en
1
root:/sys/bus/iio/devices/iio:device1> cat filter_fir_config
```

### Control del ancho de banda de RF para la recepción

La ruta de recepción, en su sección analógica, incluye un par de filtros analógicos (RX TIA, RX BB LPF), antes del ADC, para eliminar señales espurias, productos de intermodulación y limitar el ancho de banda después del mezclador de bajada (*downconversion*). El filtro pasa bajas del amplificador de trans-impedancia de recepción (RX TIA LPF) es un filtro, de un polo, con una frecuencia de corte de 3dB. Por su parte, el filtro Butterworth, pasa bajas de recepción (RX BB LPF), es un filtro de tercer orden con frecuencia de corte en 3 dB.

Ambos filtros pueden ser configurados con una sola instrucción mediante el parámetro *in\_voltage\_rf\_bandwidth*, como se ilustra a continuación:

```
root:/sys/bus/iio/devices/iio:device1> cat in_voltage_rf_
bandwidth
18000000

root:/sys/bus/iio/devices/iio:device1> echo 9000000 > in_
voltage_rf_bandwidth

root:/sys/bus/iio/devices/iio:device1> cat in_voltage_rf_
bandwidth
9000000
```

## Control del ancho de banda de RF en la transmisión

También en la ruta de transmisión existen un par de filtros analógicos (TX BB LPF, TX *Secondary* LPF), justo después del DAC, para eliminar señales espurias y limitar el ancho de banda antes del mezclador de subida (*upconversion*). El filtro TX BB LPF es un filtro Butterworth, pasa bajas de tercer orden, con frecuencia de corte en 3dB. Por su parte, el filtro TX *Secondary* LPF es un filtro pasa bajas, de un polo, con frecuencia de corte en 3 dB. Ambos filtros se configuran mediante el parámetro *out\_voltage\_rf\_bandwidth*, como se ilustra a continuación:

```
root: / sys / bus / iio / devices / iio:device1 > cat out _ voltage _ rf _  
bandwidth  
18000000  
root:/sys/bus/iio/devices/iio:device1> echo 9000000 > out _  
voltage _ rf _ bandwidth  
root:/sys/bus/iio/devices/iio:device1> cat out _ voltage _ rf _  
bandwidth  
9000000
```

Existen muchos más parámetros y funciones que pueden ser de utilidad, incluyendo las opciones de calibración disponibles, pero las limitaciones de espacio en este informe técnico no permiten tratarlas aquí. El lector puede encontrar más información en el sitio web con la siguiente dirección:

<https://wiki.analog.com/resources/tools-software/linux-drivers/iio-transceiver/ad9361>

## Capítulo 3.

### Puesta a punto del banco de pruebas SDR

#### Pasos para la descarga e instalación de la imagen de Linux en tarjetas de desarrollo con trans-receptores AD-FMCOMMSxEBZ

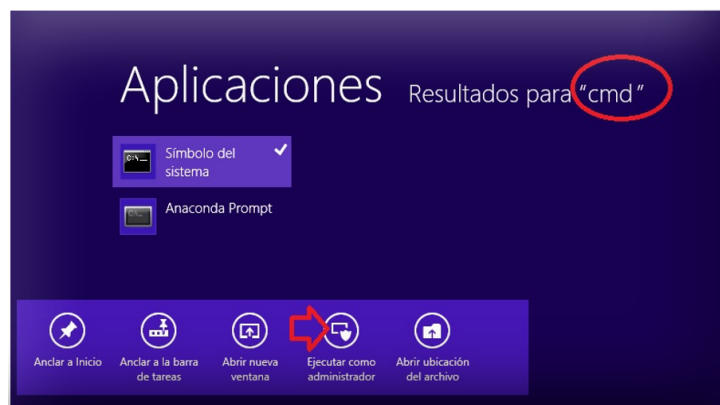
A continuación, se presentan las instrucciones para instalar, en una memoria SD de 8 GB, la imagen de Linux que corresponde a la tarjeta de desarrollo que realizará las funciones de DSP en nuestro banco de pruebas SDR, de tal forma que el usuario pueda controlar y acceder el módulo trans-receptor.

1. Asegurarse de contar con la imagen Linux más reciente, proporcionada por el fabricante Analog Devices, para la tarjeta de desarrollo con la que se cuenta y que normalmente se identifica por su proyecto de hardware (fmcommsx). Las imágenes más recientes, para todos los modelos de tarjeta soportados, se pueden descargar del sitio Wiki de Analog Devices:

[https://wiki.analog.com/resources/tools-software/linux-software/zynq\\_images#staying\\_up\\_to\\_date](https://wiki.analog.com/resources/tools-software/linux-software/zynq_images#staying_up_to_date)

Una vez descargado el archivo con la imagen Linux, se descomprime para obtener el archivo con extensión .img.

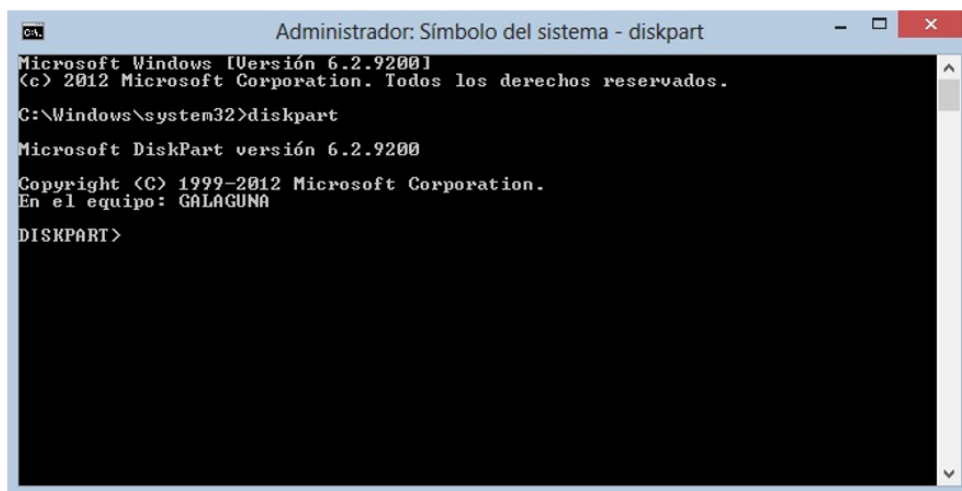
2. Copiar la imagen a la tarjeta SD destino. Se sugiere hacer esto en una máquina con sistema operativo Windows, a fin de poder seguir con mayor claridad el avance del proceso, dado que en Linux es más problemático. Para esto, se ejecuta, como administrador, la ventana de mandos con el símbolo del sistema (CMD).



## Capítulo 3.

### Puesta a punto del banco de pruebas SDR

3. Eliminar toda partición en la tarjeta destino. Para ello, se emplea la herramienta **diskpart**. Iniciamos la herramienta, en la línea de mandos, ejecutando el mando **diskpart**:



```
Administrador: Símbolo del sistema - diskpart
Microsoft Windows [Versión 6.2.9200]
(c) 2012 Microsoft Corporation. Todos los derechos reservados.
C:\Windows\system32>diskpart
Microsoft DiskPart versión 6.2.9200
Copyright (C) 1999-2012 Microsoft Corporation.
En el equipo: GALAGUNA
DISKPART>
```

A continuación, se identifica el número del disco que corresponde a la tarjeta SD. Esto lo hacemos con el mando **list disk**.



```
Administrador: Símbolo del sistema - diskpart
Microsoft Windows [Versión 6.2.9200]
(c) 2012 Microsoft Corporation. Todos los derechos reservados.
C:\Windows\system32>diskpart
Microsoft DiskPart versión 6.2.9200
Copyright (C) 1999-2012 Microsoft Corporation.
En el equipo: GALAGUNA
DISKPART>
DISKPART> list disk

  Núm Disco  Estado      Tamaño  Disp  Din  Gpt
-----
Disco 0      En línea    1863 GB      0 B
Disco 1      En línea    7580 MB    7579 MB
DISKPART> _
```

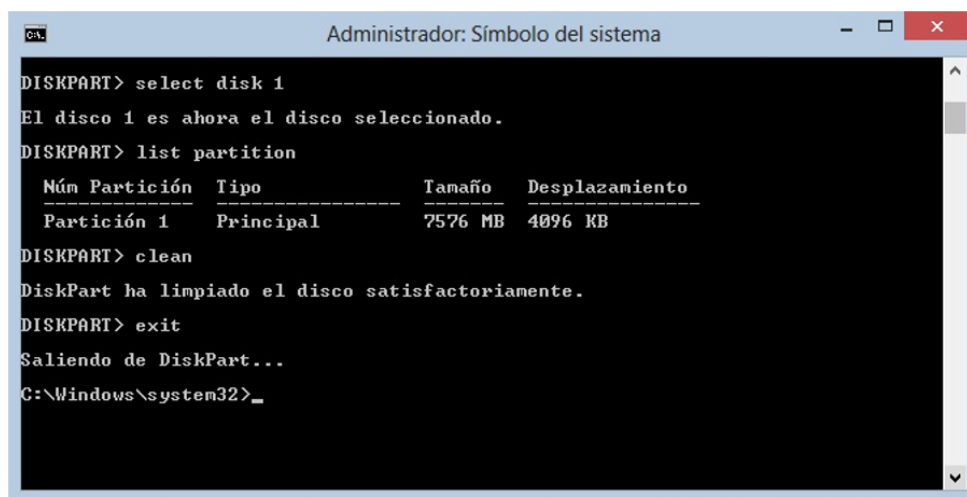
En este ejemplo, se puede apreciar que el disco de interés es el identificado como “Disco 1”, dado que su capacidad es justamente de 8GB. Ahora, se puede seleccionar el disco para trabajar con este. Hay que tener cuidado de seleccionar al disco correcto o, de otro modo, se puede dañar irremediablemente al sistema operativo anfitrión. Se selecciona el disco destino con el mando **select disk n**.

## Capítulo 3.

### Puesta a punto del banco de pruebas SDR

En este punto, es posible listar las particiones existentes mediante el mando **list partition**.

Finalmente, se eliminan las particiones existentes con el mando **clean** y se sale de la herramienta tecleando el mando **exit**.



```
Administrador: Símbolo del sistema

DISKPART> select disk 1
El disco 1 es ahora el disco seleccionado.

DISKPART> list partition

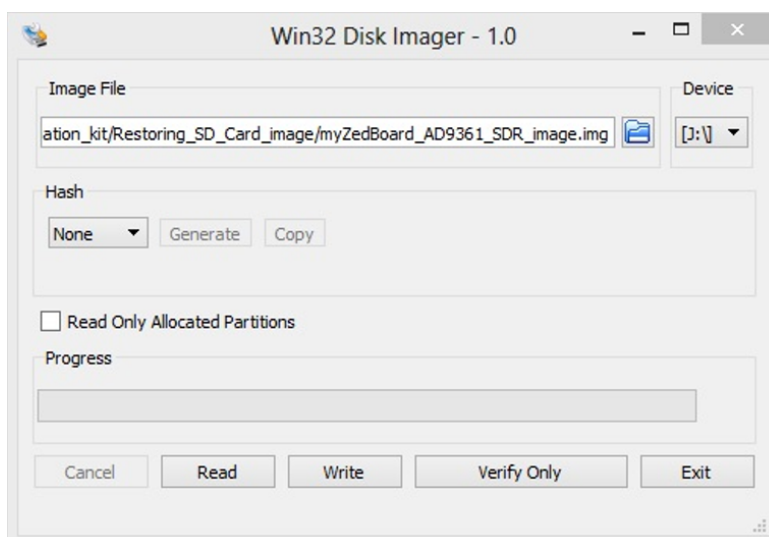
   Núm Partición   Tipo            Tamaño      Desplazamiento
-----
Partición 1       Principal       7576 MB     4096 KB

DISKPART> clean
DiskPart ha limpiado el disco satisfactoriamente.

DISKPART> exit
Saliendo de DiskPart...

C:\Windows\system32>
```

4. Almacenar la imagen en la tarjeta SD. Para ello, se recomienda usar la aplicación *Win32 Disk Imager*. Lo primero es seleccionar el archivo con la imagen y, a continuación, el dispositivo destino.



Para iniciar la carga, se oprime el botón <Write>. El proceso durará varios minutos, es importante estar conscientes de esto para no desesperar y abortar el proceso antes de tiempo.



5. Retirar la tarjeta SD en forma segura. Una vez terminada la transferencia de la imagen Linux, es posible que el mismo sistema Windows no reconozca el contenido de la tarjeta SD. Al sistema Windows se le dificulta identificar las particiones que Analog Devices realizó para su imagen Linux. Sin embargo, se puede explorar el contenido de la tarjeta SD mediante una computadora con sistema Linux, de preferencia Ubuntu. Allí, deberían aparecer dos particiones: *BOOT* y *rootfs*.

6. Personalizar la imagen Linux. Esto, en concordancia con la tarjeta de desarrollo específica que se va a emplear. Para ello, se monta la tarjeta SD en una máquina con sistema operativo Linux, vamos a la partición *BOOT* y buscamos la carpeta cuyo rótulo corresponde a nuestro hardware. Por ejemplo, si se cuenta con una tarjeta ZedBoard y una tarjeta AD-FMCOMMS2-EBZ, la carpeta de interés tendrá un rótulo similar a *zynq-zed-adv7511-ad9361*. Desde la carpeta de interés, en la misma tarjeta SD, se copian a la raíz de la partición *BOOT* los siguientes archivos:

- a. *devicetree.dtb*
- b. *BOOT.BIN*

Finalmente, desde la carpeta *common*, también se copia la imagen *uImage* a la raíz de la partición de *BOOT*. Hecho esto, se puede desmontar la tarjeta SD de la computadora y está lista para ser insertada en de desarrollo.

7. Comprobar el funcionamiento de la imagen Linux. Para ello, es necesario asegurarse de que la tarjeta de desarrollo está configurada para arrancar desde la tarjeta SD (ver la guía o manual específico de la tarjeta de desarrollo). Si todo es correcto y la tarjeta de desarrollo está conectada a un monitor HDMI, un ratón y un teclado (en su caso, mediante un hub USB), se podrá observar, después de algunos segundos, como arranca el sistema Linux de Analog Devices. En el caso de algunas tarjetas de desarrollo, también se puede seguir la secuencia de arranque, al conectar un cable al puerto micro-USB que corresponde al puerto UART de la tarjeta, si se instala un puente USB-serial y se ejecuta un programa terminal con una velocidad de *115200 bps*.

En caso de necesitarse, las contraseñas por defecto son:

## Capítulo 3.

### Puesta a punto del banco de pruebas SDR

USER	PASSWORD
root	analog
analog	analog

Cuando se necesite apagar el sistema de la tarjeta de desarrollo, se debe realizar esto mediante el icono de apagado en la interfaz gráfica de Linux o, en su defecto, mediante los siguientes mandos de consola:

```
$ sudo shutdown -h now
```

ó

```
$ sudo poweroff
```

Es importante enfatizar que el sistema no puede apagarse simplemente con el interruptor de alimentación, pues se corre el riesgo de corromper al sistema.

8. Probar el trans-receptor de la tarjeta AD-FMCOMMSx-EBZ. Para realizar esto, de manera rápida, se puede usar la aplicación gráfica `iio_oscilloscope` que proporciona Analog Device, como parte de la imagen Linux, siguiendo las instrucciones de la guía Quick Start que acompaña al módulo AD-FMCOMMSx-EBZ. Para ello, ver el sitio:

<https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/quickstart/zynq>

Básicamente, se trata de una aplicación con apariencia de osciloscopio, que permite observar las señales transmitidas y recibidas. Para más detalles, ver el sitio:

[https://wiki.analog.com/resources/tools-software/linux-software/iio\\_oscilloscope](https://wiki.analog.com/resources/tools-software/linux-software/iio_oscilloscope)

9. Por último, debe verificarse que la imagen Linux incluye la

biblioteca *Libiio*. Para ello, desde el sistema Linux que corre en la tarjeta de desarrollo, abrir la aplicación Terminal y ejecutar el siguiente mando.

```
$ iio_info -s
```

Debe observarse un mensaje con información sobre los contextos disponibles. Si se marca cualquier error, eso significa que no se encuentra instalada la biblioteca *Libiio*.

A continuación, se detallan los pasos que hay que seguir en caso de que no se cuente con la biblioteca *Libiio*.

### Instalación de biblioteca *Libiio* y compilación de códigos de ejemplo

1. Instalar biblioteca *Libiio*. Para instalar la biblioteca y los ejemplos, es necesario que la tarjeta de desarrollo ya cuente con la imagen de Linux y se encuentre conectada a Internet mediante un cable de Ethernet. Debemos tener instalado *cmake* y otras dependencias. Para ello, teclear:

```
$ sudo apt-get install libxml2 libxml2-dev bison flex libcdk5-dev cmake
```

Luego, ejecutar la siguiente secuencia de mandos:

```
$ cd /usr/local/src
$ sudo git clone https://github.com/analogdevicesinc/libiio.git
$ cd /usr/local/src/libiio/
$ git clean -d -f -x
$ sudo cmake ./
$ sudo make all
$ sudo make install
```

2. Compilar los ejemplos de código C. Estos programas demuestran el uso de la biblioteca *Libiio*. Para ello, ejecutamos la siguiente secuencia de mandos:

```
$ cd /usr/local/src/libiio/examples
$ sudo make
```

Cuando la compilación termina, en el directorio de trabajo se encuentran los archivos ejecutables con los códigos de demostración.

3. Ejecutar un código de ejemplo. En particular, el programa

ad9361-iostream es muy ilustrativo, pues configura el trans-receptor AD9361, recibe muestras, les realiza un procesamiento mínimo y las retransmite:

```
$ ./ ad9361-iostream
```

## El código de referencia ad9361-iostream.c

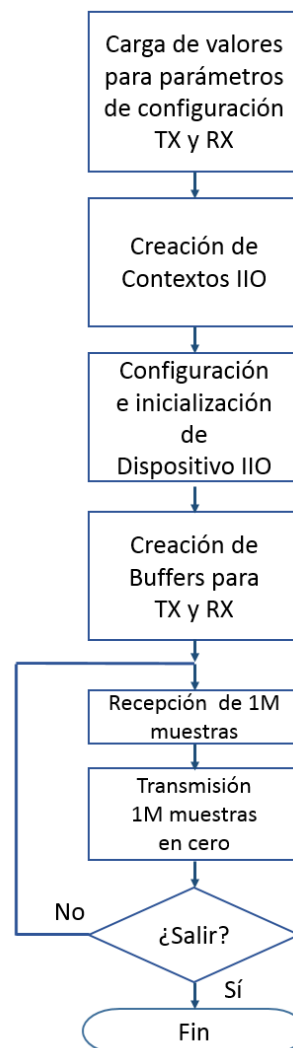
El código *ad9361-iostream.c* sirve muy bien como punto de partida para desarrollar una aplicación SDR. Básicamente, demuestra como inicializar y acceder a los trans-receptores de las serie AD936x mediante código C. El código demostrativo se encuentra en la ruta siguiente:

```
/usr/local/src/libiio/examples
```

Para empezar, en este código se hace la inclusión del archivo de cabecera *iio.h* para poder invocar las funciones IIO. Se definen algunas estructuras necesarias, para representar a los dispositivos con sus configuraciones, y se crean algunas funciones básicas de apoyo que invocan a las primitivas *iio*. Estas funciones básicas permiten identificar, leer y escribir datos al dispositivo *iio*. Estas mismas funciones validan resultados y, en su caso, despliegan los mensajes de error correspondientes.

El cuerpo del programa puede describirse, en forma general, mediante el diagrama de flujo de la Figura 3.1.

**Figura 3.1** Diagrama de flujo para el código *ad9361-*



*iiostream.c*.

## La biblioteca libad9361-iio

Esta colección de funciones permite calcular los coeficientes para los filtros FIR del trans-receptor “sobre la marcha” (“on the fly”), lo que posibilita reconfigurar al dispositivo durante la operación, cosa que es muy conveniente para el concepto SDR. Incluye funciones para determinar automáticamente todas las relaciones de diezmado e interpolación, así como los valores óptimos para los osciladores y divisores de frecuencia involucrados, tanto para la ruta de recepción como para la de transmisión.

El proyecto de la biblioteca, con todos los códigos fuentes y la información asociada para su instalación y empleo, se encuentra en el siguiente sitio:

*<https://github.com/analogdevicesinc/libad9361-iio>*

En general, se encuentran tres tipos de códigos demostrativos:

**a) Auto\_rate\_test\_hw.** Este sub-proyecto demuestra cómo se pueden determinar los valores para las velocidades de muestreo y las frecuencias de los osciladores involucrados, en las rutas de recepción y transmisión, a partir de la velocidad de muestreo requerida para las señales de banda base.

**b) Filter\_designer\_test.** Este sub-proyecto demuestra cómo se pueden calcular los coeficientes de un filtro FIR, a partir de sus especificaciones, empleando diversas técnicas de diseño.

**c) Filter\_designer\_hw.** Este sub-proyecto demuestra cómo se pueden calcular los coeficientes de un filtro FIR y cómo se pueden descargar directamente al trans-receptor.

En el siguiente capítulo se presentará, con más detalle, un ejemplo que ilustra cómo aprovechar todos los recursos presentados hasta ahora, para realizar una prueba de comunicaciones digitales, empleando el banco de pruebas propuesto.

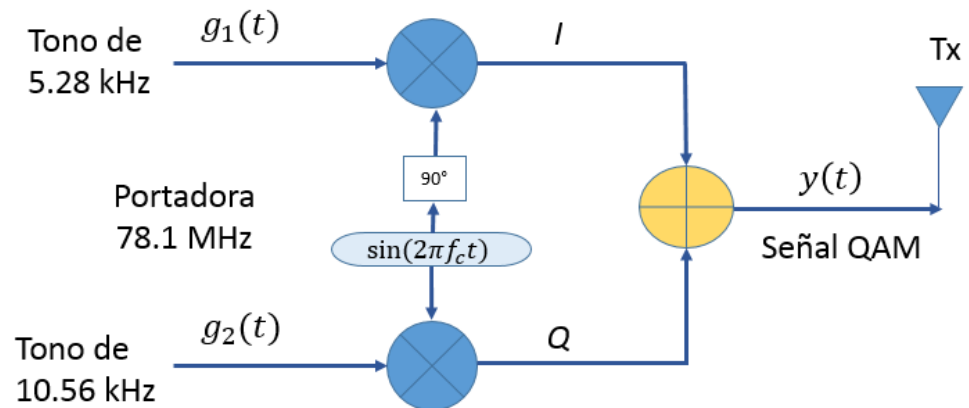
## Capítulo 4.

### Aplicación semilla para SDR básico con el banco de pruebas propuesto

En este capítulo se presenta un proyecto que demuestra cómo se puede aplicar el banco de pruebas propuesto para iniciar una aplicación SDR básica. En esencia se trata del nivel físico de un sistema de comunicación digital con modulación en cuadratura.

#### Especificación general

Se desea realizar, mediante la tarjeta de desarrollo ZedBoard o ZC702 y una tarjeta mezzanine AD-FMCOMMS4-EBZ, un sistema de comunicación digital QAM con frecuencia portadora de 78.1 MHz y dos tonos de banda base: uno tono de 5.28 kHz para la rama en fase (I) y otro de 10.56 kHz para la rama en cuadratura (Q). La idea general se representa en la Figura 4.1.



**Figura 4.1.** Modelo general para la transmisión en un esquema de modulación QAM

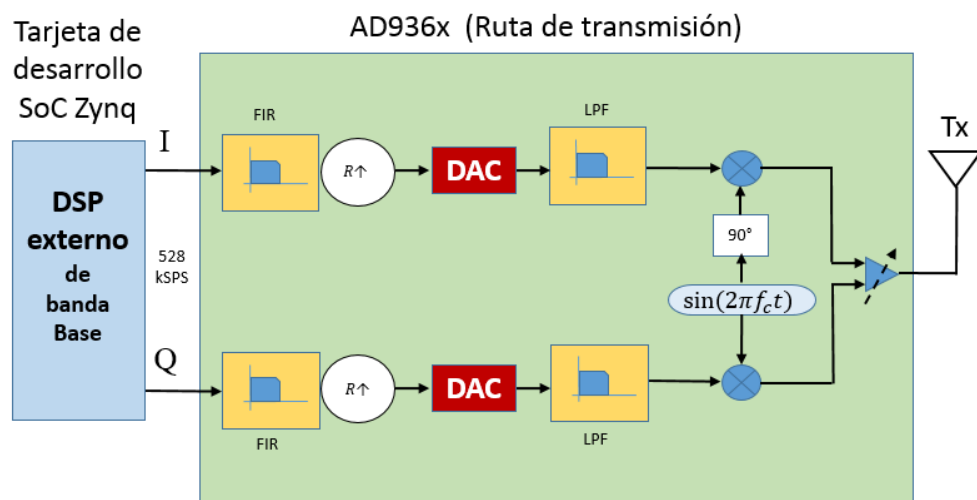
De manera más específica, podemos incorporar en nuestro modelo los componentes de nuestro banco de pruebas. El sistema completo, en lo relativo a la transmisión, se bosqueja en la Figura 4.2, mientras que, en lo relativo a la recepción, se bosqueja en la Figura 4.3.



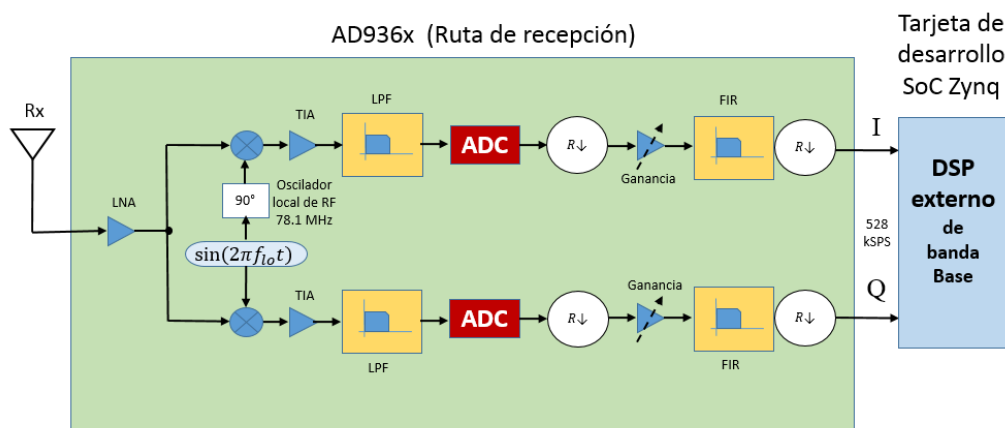
## Capítulo 4.

### Aplicación semilla para SDR básico con el banco de pruebas propuesto

Como se puede apreciar en ambas figuras, el trans-receptor AD936x se complementa con el procesamiento digital de señales realizado en la tarjeta de desarrollo, en nuestro caso una tarjeta ZedBoard o ZC702.



**Figura 4.2.** Sistema de transmisión para el banco de prueba.



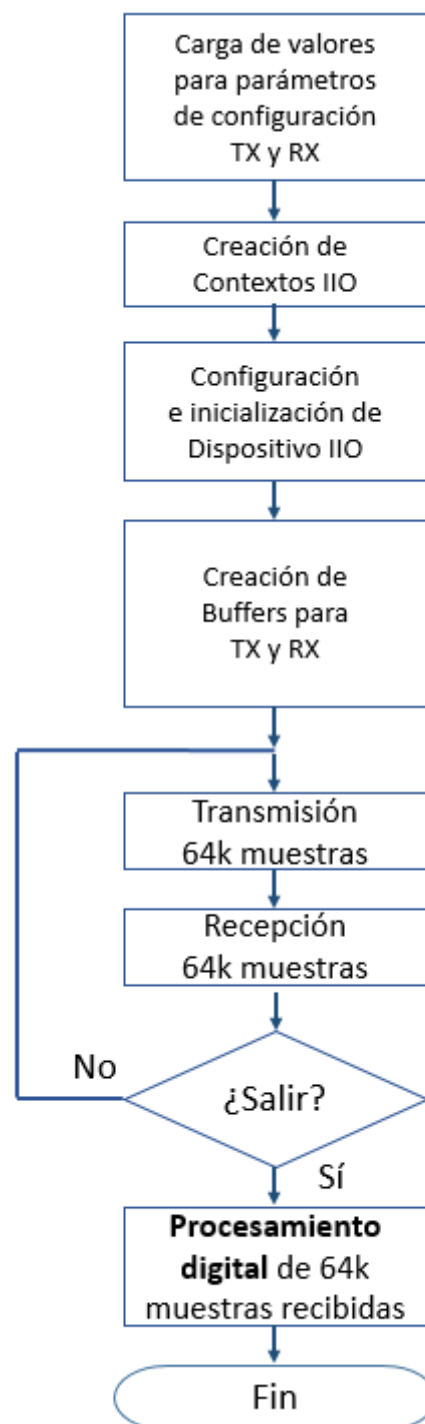
**Figura 4.3.** Sistema de recepción para el banco de prueba

## Capítulo 4.

### Aplicación semilla para SDR básico con el banco de pruebas propuesto

En lo referente al código demostrativo de este capítulo, precisamente el que se ejecuta en la tarjeta de desarrollo, parte del código de referencia *ad9361-iiostream.c* pero se ha adecuado a las necesidades particulares del problema planteado. Esto significa que, además de las secciones de configuración, inicialización y el lazo de transmisión/recepción, se incluye una sección que procesa las señales recibidas de banda base a fin de recuperar los tonos originalmente transmitidos. En la Figura 4.4., se presenta el diagrama de flujo del código principal demostrativo (*main.c*) de este capítulo.

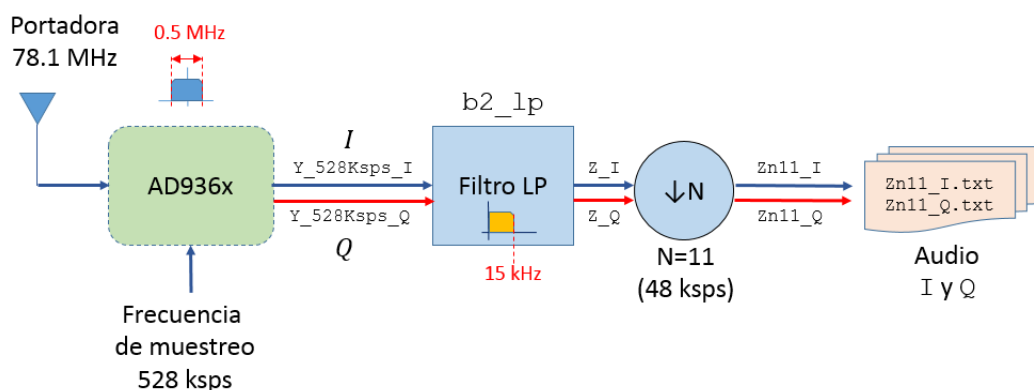
El procesamiento realizado sobre las secuencias de datos asume un par de supuestos de diseño. Se parte del hecho de que el trans-receptor se ha configurado para transmitir y sintonizar una portadora de 78.1 MHz, sobre la que se transmiten dos tonos de audio, uno de 5.28 kHz, para la rama I, y otro de 10.56 kHz, para la rama Q. También se asume que la velocidad de muestreo de banda base es de 528 kSPS. Debido a esto último, es que el ancho de banda de RF se limita a 500 kHz.



**Figura 4.4.** Diagrama de flujo para el código *main.c* de la aplicación demostrativa con QAM.

## Capítulo 4. Aplicación semilla para SDR básico con el banco de pruebas propuesto

Para el caso del código demostrativo presentado aquí, el procesamiento digital de las secuencias recibidas es relativamente simple. El flujo y las etapas de procesamiento se muestran en la Figura 4.5.



**Figura 4.5.** Diagrama a bloques del procesamiento digital realizado sobre las secuencias de datos recibidas.

El propósito principal, del código de procesamiento de las secuencias de datos recibidas, es la recuperación de los tonos originales de modulación QAM y su presentación con una velocidad de muestreo estándar para señales de audio, en este caso 48 kSPS. Para ello, dado que el dispositivo trans-receptor entrega las secuencias de datos con una velocidad de muestreo de 528 kSPS, se recurre a un filtro pasa bajas con frecuencia de corte en 15 kHz y una etapa de muestreo descendente con un factor de diezmado de 11. El código, que integra todo lo mencionado hasta el momento, se presenta en el Anexo A y corresponde tanto al cuerpo del programa principal, bosquejado en la Figura 4.4, como a la sección de procesamiento digital de las secuencias recibidas, bosquejado en la Figura 4.5.

Por supuesto, se invocan funciones para este procesamiento digital, como lo es el filtrado y el diezmado, que no se presentan aquí con detalle, dadas las limitaciones de espacio. Sin embargo, el lector puede implementar sus propias funciones de procesamiento, dado que se trata de técnicas de dominio público y que no deben ser un reto insuperable para el lector especializado en el ámbito de las comunicaciones digitales y el procesamiento digital de señales.

## Resultados obtenidos

El proyecto completo, en código C, fue compilado y ejecutado en la misma tarjeta, todo bajo el entorno Linux. La ejecución del programa demostrativo funcionó de acuerdo a lo previsto y conforme a la especificación de un sistema de modulación QAM donde el sistema transmisor se encuentra en la misma tarjeta de desarrollo que el sistema receptor.

Para fines prácticos, esto permite que el oscilador del transmisor se encuentre perfectamente sincronizado respecto del oscilador local del receptor, lo que elimina la necesidad de la etapa de sincronización. Por otro lado, dada la configuración del banco de pruebas, que emplea al mismo dispositivo trans-receptor para transmitir y recibir las señales, la antena transmisora se encuentra justo a un costado de la antena receptora, lo que reduce los efectos del canal y permite tener un sistema que puede considerarse como de lazo de retroalimentación.

Las muestras transmitidas, correspondientes a los tonos de banda base muestreados a 528 kSPS, se encuentran almacenadas en los siguientes arreglos de enteros:

*-int16\_t SinT[WAV\_TABLE\_SIZE], para la señal de tono de 5.28 kHz.*

*-int16\_t Sin2T[WAV\_TABLE\_SIZE], para la señal de tono de 10.56 kHz.*

Por lo tanto, la comprobación del buen funcionamiento del código consiste simplemente en comparar las secuencias de las señales de audio recibidas, recuperadas y almacenadas, a 48 kSPS, en los archivos Zn11\_I.txt y Zn11\_Q.txt, respecto de las secuencias originales transmitidas.

Las pruebas realizadas mostraron que esta aplicación simple de comunicaciones digitales funcionó de acuerdo a lo esperado y que, en efecto, este código puede funcionar como un código semilla para el desarrollo de un sistema SDR más sofisticado, dado que la parte correspondiente al sistema de modulación en cuadratura quedó suficientemente probado y tomando en cuenta que la mayoría de los sistemas de comunicación digital pueden ponerse en práctica mediante dicho sistema que funge como la etapa de front-end.



## Capítulo 5. Conclusiones

En este informe técnico se presentó una configuración básica para la conformación de un banco de pruebas para el desarrollo de aplicaciones de comunicaciones digitales en el contexto del concepto Software Defined Radio (SDR). El sistema propuesto se compone por un módulo trans-receptor de la serie ADFMCOMMSx y por una tarjeta de desarrollo ZedBoard o ZC702, que se han descrito brevemente en este documento.

El sistema propuesto completo, de conformidad con el concepto SDR, necesariamente incluye código para la realización del procesamiento digital de señales, así como para el control y la configuración del hardware involucrado. Para este fin se recurrió al empleo de un sistema operativo Linux, que corre sobre las tarjetas de desarrollo y permite acceder al dispositivo trans-receptor mediante controladores con interfaz IIO que, para tal propósito, proporciona el fabricante de los circuitos trans-receptores de la familia AD936x, que constituyen el corazón del hardware para la transmisión y la recepción de las señales. Los dispositivos AD936x se caracterizan por su tecnología de conversión, digital-analógica y analógica-digital, de alta velocidad, por su muy alta integración, por su amplio intervalo de operación en el espectro de radio frecuencia, así como su notable flexibilidad en cuanto a configuración y aplicaciones.

En este informe también se presentó una aplicación demostrativa, con un código C semilla, para ilustrar cómo es posible, con base en el banco de pruebas propuesto, desarrollar aplicaciones de comunicaciones digitales donde el procesamiento de las señales de banda base se realiza por completo mediante software de procesamiento digital de señales.

Así, con base en los resultados y los elementos descritos en este informe técnico, se demuestra que es factible contar con una plataforma, relativamente barata, para la experimentación y la prueba de los algoritmos involucrados en la realización práctica de los bloques funcionales que conforman el nivel físico todo sistema de comunicación digital, en general, y de los sistemas SDR, en particular.

## **Anexo. A. Código principal para aplicación semilla SDR con modulación QAM**

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Código principal para banco de prueba SDR (SDR Test bench)
// Transmisión de señal QAM básica (dos tonos en cuadratura)
// sobre una portadora de 78.1 MHz
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Código semilla para proyectos con dispositivos ad936x usando biblioteca libio
// y código para diseño de filtros de Iowa Hills y libad
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Versión de prueba compilada y probada en tarjetas ZedBoard y ZC702 Evaluation Kit
// con tarjeta mezzanine AD-FMCOMMs4-EBZ
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Universidad Autónoma Metropolitana, Unidad Lerma
// Área de Sistemas de Información y Ciencias Computacionales
// Programador integrador: Gerardo A. Laguna-Sánchez
// Fecha: 9 de julio 2019
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Se han tomado como base los siguientes proyectos de acceso libre:
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
 * libio - AD9361 IIO streaming example
 *
 * Copyright (C) 2014 IABG mbH
 * Author: Michael Feilen <feilen_at_iabg.de>
 *
 */
// FIR & IIR filters code:
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
By Daniel Klostermann
Iowa Hills Software, LLC IowaHills.com
If you find a problem, please leave a note at:
http://www.iowahills.com/feedbackcomments.html
May 1, 2016
```

The code in this kit is essentially the same code used in the Iowa Hills IIR and FIR programs, but without the GUI. If you compare the output from this code to the output



from the Iowa Hills programs, you may find small differences in the coefficients. The reasons for this are numerous, and of little concern so long as the filters generated here behave in the desired manner.

```

*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//System includes
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <iio.h>

//Application includes
#include "FFTCODE.h" // The FFT code and the windowing code
                    // (Kaiser and Sinc windows, etc.)
#include "IIRFilterCode.h" // The IIR filter code.
#include "FIRFilterCode.h" // The FIR filter code.
#include "NewParksMcClellan.h" // The Parks McClellan FIR algorithm code.
#include "FreqSamplingCode.h"

#include "defs2.h"
#include "matopc.h"
#include "fft_gv2.h"
#include "myfilters2.h"
#include "myfm.h"
#include "mypilot_pll.h"
#include "ad936x_iio.h"

/*****
/*Global variables */
*****/

#define WAV_TABLE_SIZE 100
#define DATA_SIZE 65536

//For customized FFT:
#define CONV_FFT_SIZE DATA_SIZE
pyramidarray *conv_gre=new pyramidarray(CONV_FFT_SIZE,1);

```

```

piramidarray *conv_gro=new piramidarray(CONV_FFT_SIZE/2,1);
piramidarray *conv_gWd=new piramidarray(CONV_FFT_SIZE,1);
piramidarray *conv_gWi=new piramidarray(CONV_FFT_SIZE,1);

//*****
//External variables for iio (in ad936x_iio.cpp):
//*****

/* IIO structs required for streaming */
extern struct iio_context *ctx;
extern struct iio_channel *rx0_i;
extern struct iio_channel *rx0_q;
extern struct iio_channel *tx0_i;
extern struct iio_channel *tx0_q;
extern struct iio_buffer *rxbuf;
extern struct iio_buffer *txbuf;

extern bool stop;

//*****
//Customized code:
//*****

void loadintarray(const char *FileName, int size,int *Array)
{
    FILE *fp;
    char buffer[255];
    fp = fopen(FileName, "r");

    for(int i=0;i< size; i++)
    {
        fgets(buffer, 255, (FILE *)fp);
        Array[i] = atoi(buffer);
    }
    fclose(fp);
}

```

```

//*****
//Main code:
//*****

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

    printf("Cuerpo de prueba para proyecto SDR Test bench.\n");
    printf("Autor: Gerardo Laguna, UAM-L, 9 julio 2019.\n\n");

    printf("Prueba de lazo streaming con AD9361 y modulacion QAM basica con dos tonos.\n");
    printf("Portadora de 78.1 MHz.\n\n");

    //*****
    /* Initial customized FFT/IFFT weights configuration */
    //*****

    pyramid_Wcoef(conv_gWd);
    pyramid_invWcoef(conv_gWi);

    //*****
    /* Initial iio configuration */
    //*****

    long long fs = MHZ(0.528); // sample rate

    //Sine table (aprox. 5.28 KHz):
    int16_t SinT[WAV_TABLE_SIZE]={0, 64, 128, 191, 254, 316, 376, 435, 493, 548,
    601, 652, 700, 746, 789, 828, 864, 897, 926, 952,
    973, 991, 1005, 1015, 1021, 1024, 1021, 1015, 1005, 991,
    973, 952, 926, 897, 864, 828, 789, 746, 700, 652,
    601, 548, 493, 435, 376, 316, 254, 191, 128, 64,
    -1, -65, -129, -192, -255, -317, -377, -436, -494,
    -549, -602, -653, -701, -747, -790, -829, -865, -898, -927,
    -953, -974, -992, -1006, -1016, -1022, -1024, -1022, -1016, -1006,
    -992, -974, -953, -927, -898, -865, -829, -790, -747, -701,
    -653, -602, -549, -494, -436, -377, -317, -255, -192, -129, -65};

    //Sine table (aprox. 10.56 KHz):
    int16_t Sin2T[WAV_TABLE_SIZE]={0, 128, 254, 376, 493, 601, 700, 789, 864, 926,
    973, 1005, 1021, 1021, 1005, 973, 926, 864, 789, 700,
    601, 493, 376, 254, 128, -1, -129, -255, -377, -494,
    -602, -701, -790, -865, -927, -974, -1006, -1022, -1022, -1006,

```

```

-974, -927, -865, -790, -701, -602, -494, -377, -255, -129,
0, 128, 254, 376, 493, 601, 700, 789, 864, 926,
973, 1005, 1021, 1021, 1005, 973, 926, 864, 789, 700,
601, 493, 376, 254, 128, 0, -129, -255, -377, -494,
-602, -701, -790, -865, -927, -974, -1006, -1022, -1022, -1006,
-974, -927, -865, -790, -701, -602, -494, -377, -255, -129};

printf("Acquiring IIO context\n");
ASSERT((ctx = iio_create_default_context()) && "No context");
ASSERT(iio_context_get_devices_count(ctx) > 0 && "No devices");

//*****
/* Rx/Tx Signal path filter design (libad) configuration */
//*****

int ret;
unsigned long Fpass, Fstop, wnomTX, wnomRX, rate;
struct iio_device *dev;

rate = (unsigned long) fs;

dev = iio_context_find_device(ctx, "ad9361-phy");

printf("Testing rate: %lu\n", rate);
Fpass = 100000;
Fstop = Fpass * 1.7;
wnomTX = 1.6 * Fstop;
wnomRX = 1.4 * Fstop;

ret = ad9361_set_bb_rate_custom_filter_manual(dev, rate, Fpass, Fstop,
wnomTX, wnomRX);
if (ret < 0)
return ret;

//*****
//iio streaming configuration:
//*****

// Streaming devices
struct iio_device *tx;
struct iio_device *rx;

```

```

// RX and TX sample counters
size_t nrx = 0;
size_t ntx = 0;

// Stream configurations
struct stream_cfg rxcfg;
struct stream_cfg txcfg;

// Listen to ctrl+c and ASSERT
signal(SIGINT, handle_sig);

// RX stream config
rxcfg.bw_hz = 500000; // 500 KHz rf bandwidth
rxcfg.fs_hz = fs; // rx sample rate
rxcfg.lo_hz = MHZ(78.1); // Rx rf frequency
rxcfg.rfport = "A_BALANCED"; // port A (select for rf freq.)

// TX stream config
txcfg.bw_hz = 500000; // 500 KHz rf bandwidth
txcfg.fs_hz = fs; // tx sample rate
txcfg.lo_hz = MHZ(78.1); // Tx rf frequency
txcfg.rfport = "A"; // port A (select for rf freq.)

printf("* Acquiring AD9361 streaming devices\n");
ASSERT(get_ad9361_stream_dev(ctx, TX, &tx) && "No tx dev found");
ASSERT(get_ad9361_stream_dev(ctx, RX, &rx) && "No rx dev found");

printf("* Configuring AD9361 for streaming\n");
ASSERT(cfg_ad9361_streaming_ch(ctx, &txcfg, RX, 0) && "RX port 0 not found");
ASSERT(cfg_ad9361_streaming_ch(ctx, &txcfg, TX, 0) && "TX port 0 not found");

printf("* Initializing AD9361 IIO streaming channels\n");
ASSERT(get_ad9361_stream_ch(ctx, RX, rx, 0, &rx0_i) && "RX chan i not found");
ASSERT(get_ad9361_stream_ch(ctx, RX, rx, 1, &rx0_q) && "RX chan q not found");
ASSERT(get_ad9361_stream_ch(ctx, TX, tx, 0, &tx0_i) && "TX chan i not found");
ASSERT(get_ad9361_stream_ch(ctx, TX, tx, 1, &tx0_q) && "TX chan q not found");

printf("* Enabling IIO streaming channels\n");
iio_channel_enable(rx0_i);
iio_channel_enable(rx0_q);
iio_channel_enable(tx0_i);

```

```

iio_channel_enable(tx0_q);

printf("* Creating non-cyclic IIO buffers with DATA_SIZE\n");
rxbuf = iio_device_create_buffer(rx, DATA_SIZE, false);
if (!rxbuf) {
    perror("Could not create RX buffer");
    shutdown_iio_dev();
}
txbuf = iio_device_create_buffer(tx, DATA_SIZE, false);
if (!txbuf) {
    perror("Could not create TX buffer");
    shutdown_iio_dev();
}

//*****
/* Streaming */
//*****
vector *y_528Ksps_I=new vector(1,DATA_SIZE,0); //For I Rx samples (AD936x @ 528 Ksps)
vector *y_528Ksps_Q=new vector(1,DATA_SIZE,0); //For Q Rx samples (AD936x @ 528 Ksps)

printf("* Starting IO streaming (press CTRL+C to cancel)\n");

ssize_t nbytes_rx, nbytes_tx, cnt;
char *p_dat, *p_end;
ptrdiff_t p_inc;
while (!stop)
{
    int i, Ival, Qval;

    // WRITE: Get pointers to TX buf and write IQ to TX buf port 0
    p_inc = iio_buffer_step(txbuf);
    p_end = (char *)iio_buffer_end(txbuf);

    cnt=0;
    i=0;
    for (p_dat = (char *)iio_buffer_first(txbuf, tx0_i); p_dat < p_end; p_dat +=
p_inc) {
        // Example: fill with val
        // 12-bit sample needs to be MSB aligned so shift by 4

        Ival = SinT[i] << 4; // Real (I)

```



```

    Qval = Sin2T[i] << 4; // Imag (Q)
    ((int16_t*)p_dat)[0] = Ival; // Real (I)
    ((int16_t*)p_dat)[1] = Qval; // Imag (Q)

    cnt++;
    i++;
    if (i==WAV_TABLE_SIZE) i=0;
}

// Schedule TX buffer
nbytes_tx = iio_buffer_push(txbuf);
if (nbytes_tx < 0) { printf("Error pushing buf %d\n", (int) nbytes_tx);
shutdown_iio_dev(); }

// Refill RX buffer
nbytes_rx = iio_buffer_refill(rxbuf);
if (nbytes_rx < 0) { printf("Error refilling buf %d\n", (int) nbytes_rx);
shutdown_iio_dev(); }

// Sample counter increment and status output
nrx += nbytes_rx/ iio_device_get_sample_size(rx);
ntx += nbytes_tx/ iio_device_get_sample_size(tx);
printf("\tRX %li muestras, TX %li muestras\n", (long)nrx, (long)ntx);
} //END while (!stop)

// READ: Get pointers to RX buf and read IQ from RX buf port 0
p_inc = iio_buffer_step(rxbuf);
p_end = (char *)iio_buffer_end(rxbuf);

cnt=0;
for (p_dat = (char *)iio_buffer_first(rxbuf, rx0_i); p_dat < p_end; p_dat
+= p_inc) {
    // Example: swap I and Q
    const int16_t i = ((int16_t*)p_dat)[0]; // Real (I)
    const int16_t q = ((int16_t*)p_dat)[1]; // Imag (Q)

    y_528Ksps_I->putRe(cnt,i);
    y_528Ksps_Q->putRe(cnt,q);
    cnt++;
}

```

```

printf("Leido el buffer de recepcion del puerto 0.\n");

distroy_iio_dev();

/*****
// DSP filter coefficients
*****/

float fpass;           //Pass frequency for LPF
float fcenter;         //Center frequency for BPF
TFIRPassTypes PassType; // firLPF, firHPF, firBPF, firNOTCH, firALLPASS
int NumTaps = 64;      // 4 <= NumTaps < 256 for windowed FIR
float BW;              // 0.0 < BandWidth < 1.0
float ParksWidth;      // 0.01 <= ParksWidth <= 0.3 The transition bandwidth
float OmegaC;          // 0.0 < OmegaC < 1.0. (3dB attenuation corner)

////////////////////////////////////
// LPF (FIR) and its b2_lp coefficients:
////////////////////////////////////

float b2_lp[MAX_NUMTAPS]; // FIR filter coefficients. MAX_
                           NUMTAPS = 256

PassType = firLPF;
fpass=0.057;              // 15/(528/2) [KHz]
BW = fpass;
ParksWidth = 0.1;
OmegaC = fpass+(ParksWidth/4.0);
// Use ParksMcClellan algorithm to calculate the FIR Coefficients:
NewParksMcClellan(b2_lp, NumTaps, PassType, OmegaC, BW, ParksWidth);

/*****
// Downsampling to get 48Ksps
*****/

printf("Procesando %li muestras complejas...\n", (long)cnt);
printf("(Sea paciente!!!)\n");

vector *z=new vector(1,DATA_SIZE,0);
vector *dxre=new vector(1,DATA_SIZE,0);
vector *dxim=new vector(1,DATA_SIZE,0);
vector *work_x=new vector(1,DATA_SIZE,0);
vector *gImpResp=new vector(1,DATA_SIZE,0);
vector *work_h=new vector(1,DATA_SIZE,0);

```

```

vector *work_X=new vector(1,DATA_SIZE,1);
vector *work_H=new vector(1,DATA_SIZE,1);
vector *work_A=new vector(1,DATA_SIZE,1);
vector *work_B=new vector(1,DATA_SIZE,1);

float a[1]={1.0};

vector *z_I=new vector(1,DATA_SIZE,0);
fast_filter_gv(y_528Ksps_I,b2_lp,NumTaps,a,1,z_I,work_x,gImpResp,
work_h,conv_gre,conv_gro,conv_gWd,conv_gWi,work_X,work_H,
work_A,work_B);

vector *z_Q=new vector(1,DATA_SIZE,0);
fast_filter_gv(y_528Ksps_Q,b2_lp,NumTaps,a,1,z_Q,work_x,gImpResp,
work_h,conv_gre,conv_gro,conv_gWd,conv_gWi,work_X,work_H,
work_A,work_B);

vector *zn11_I=new vector(1,DATA_SIZE/11,0);
downsample(z_I,11,0,zn11_I); //I audio samples @ 48 Ksps

vector *zn11_Q=new vector(1,DATA_SIZE/11,0);
downsample(z_Q,11,0,zn11_Q); //Q audio samples @ 48 Ksps

zn11_I->fsave("zn11_I.txt");

zn11_Q->fsave("zn11_Q.txt");

printf("Adios!!!\n");

return 0;
}

```



## BIBLIOGRAFÍA Y REFERENCIAS PARA CONSULTA

- **AVNET.** (2014). ZedBoard (Zynq™ Evaluation and Development) Hardware User's Guide. USA: AVNET.
- **Analog Devices.** (2019). AD9361 high performance, highly integrated RF Agile Transceiver™ Linux device driver. Consultado 05/07/2019, de Analog Devices, Inc. Sitio web: <https://wiki.analog.com/resources/tools-software/linux-drivers/iio-transceiver/ad9361>
- **Beltrán-Obio Salvador.** (2018). Remote radio head for C-RAN (Master Thesis). Consultado 05/07/2019, de Universitat Politècnica de Catalunya - Barcelona Tech. Sitio web: <https://upcommons.upc.edu/handle/2117/119623?show=full>
- **Collins T. F., Getz R., Pu D., and Wyglinski A. M.** (2018). Software-Defined Radio for Engineers. USA: Artech House. Disponible en línea: <https://www.analog.com/en/education/education-library/software-defined-radio-for-engineers.html>
- **Laguna, G.** (2015). Del transistor al SoC y más allá: La industria de los semiconductores en el pronóstico de Tsugio Makimoto. Contactos, Revista de educación en ciencias e ingeniería, Vol1, No. 98, pp. 12-24. Consultado el 27 de marzo de 2020, de UAM-I, sitio web: <http://www2.izt.uam.mx/newpage/contactos/revista/98/pdfs/soc.pdf>
- **Pu D., Cozma A., and Hill T.** (2015). Four Quick Steps to Production: Using Model-Based Design for Software-Defined Radio (Part 1) The Analog Devices/Xilinx SDR Rapid Prototyping Platform: Its Capabilities, Benefits, and Tools. Consultado 05/07/2019, de Analog Devices, Inc. Sitio web: <https://www.analog.com/en/analog-dialogue/articles/using-model-based-design-sdr-1.html>
- **Sklar, B.** (2001). Digital Communications: Fundamentals and Applications. USA: Prentice Hall.
- **Stewart R.W, Barlee K, W., Atkinson D.S.W., Crockett L.H.** (2015). Software Defined Radio using MATLAB & Simulink and the RTL-SDR. USA: Strathclyde Academic Media. Disponible en línea: <https://www.mathworks.com/campaigns/offers/download-rtl-sdr-ebook.html>.