UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – PROCESAMIENTO DIGITAL DE SEÑALES

DESARROLLO DE UN SISTEMA DE RECEPCIÓN-TRANSMISIÓN DIGITAL DE PROPÓSITO GENERAL EN UN FPGA

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
ING. MARTHA PATRICIA LARA MENDOZA

TUTOR PRINCIPAL
DR. PABLO ROBERTO PÉREZ ALCÁZAR, FACULTAD DE INGENIERÍA

MÉXICO, D.F. NOVIEMBRE 2015

JURADO ASIGNADO:

Presidente:	Dr. Francisco García Ugalde
Secretario:	Dra. Lucía Medina Gómez
Vocal:	Dr. Pablo Roberto Pérez Alcázar
1 er. Suplente:	Dr. Jesús Savage Carmona
2 d o. Suplente:	M. I. Larry Escobar Salguero

Lugar donde se realizó la tesis: Facultad de Ingeniería, UNAM, México D.F.

TUTOR DE TESIS:

Dr. Pablo Roberto Pérez Alcázar

FIRMA

"El hombre no puede obtener nada sin antes dar algo a cambio, para crear, algo de igual valor debe perderse."

Hiromu Arakawa

Agradecimientos

A mis padres, maestros, amigos y racon.

Se agradece el apoyo proporcionado por DGAPA a través del proyecto PAPIIT IN116014 con título “Estudio de la propagación de ondas acústicas superficiales en niobato de litio y su efecto en gotas de carga”.

Índice general

Índice de figuras

Índice de tablas

Abreviaciones

ADC	**A**nalog to **D**igital **C**converter
AM	**A**mplitude **M**odulation
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
CDMA	**C**ode **D**ivision **M**ultiple **A**ccess
CIC	**C**ascaded **I**ntegrator **C**omb
CLB	**C**onfigurable **L**ogic **B**lock
CPU	**C**entral **P**rocessing **U**nit
CW	**C**ontinuos **W**ave
DAC	**D**igital to **A**nalog **C**onverter
DDC	**D**igital **D**own **C**onverter
DDS	**D**igital **D**irect **S**ynthesizer
DSP	**D**igital **S**ignal **P**rocessing
DUC	**D**igital **U**p **C**onverter
EPR	**E**lectron **P**aramagnetic **R**esonance
FFT	**F**ast **F**ourier **T**ransform
FID	**F**ree **I**nduction **D**ecay
FIR	**F**inite **I**mpulse **R**esponse
FM	**F**requency **M**odulation
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**escription **L**anguage
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngeneers
IF	**I**ntermediate **F**requency
IP	**I**ntelectual **P**roperty
JTAG	**J**oint **T**est **A**ction **G**roup
LUT	**L**ook **U**p **T**able

MAC	**M**ultiplicate **A**ccumulate
MSPS	**M**illions **S**amples **P**er **S**econd
NCO	**N**umerically **C**ontrolled **O**scillator
NMM	**N**on **M**emory **M**apped
ODDR	**O**utput **D**ouble **D**ata **R**ate
PLL	**P**hase **L**ocked **L**oop
PSD	**P**ower **S**pectral **D**ensity
PW	**P**ulse **W**ave
RAM	**R**andom **A**ccess **M**emory
RMN	**R**esonancia **M**agnética **N**uclear
RF	**R**adio **F**recuencia
SDR	**S**oftware **D**efined **R**adio
SNR	**S**ignal to **N**oise **R**atio
UART	**U**niversal **A**synchronous **R**eceiver and **T**ransmitter
USB	**U**niversal **S**erial **B**us
USRP	**U**niversal **S**oftware **R**adio **P**eripheral
VHDC	**V**ery **H**igh **D**ensity **C**able
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage
VHSIC	**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits
WCDMA	**W**ideband **C**ode **D**ivision **M**ultiple **A**ccess

Resumen

En este trabajo se detalla el diseño y la implementación de un sistema para recepción y transmisión de señales de radio frecuencia (RF), se estudian los diferentes elementos que componen los radios digitales y se implementan estos elementos en un dispositivo programable para aumentar la flexibilidad del sistema y reducir el número de componentes del mismo. Los radios digitales nos permiten acceder a los datos de radiofrecuencia generados en diversos experimentos, para poder almacenarlos, analizarlos y, si es el caso, lograr la reconstrucción de imágenes médicas por resonancia magnética y ultrasonido, que son dos de las aplicaciones que comúnmente hacen uso de estos datos. El sistema está basado en una tarjeta de desarrollo con un dispositivo lógico programable o por su nombre en ingles fiel programmable gate array (FPGA), la cual cuenta con un reloj que tiene una frecuencia máxima de operación de 100 MHz; en un convertidor analógico digital con una resolución de 14 bits a 65 millones de instrucciones por segundo y en un convertidor digital analógico con una resolución de 14 bits a 300 millones de instrucciones por segundo. Se usan técnicas tanto de submuestreo como de sobremuestreo, las cuales permiten procesar señales con un ancho de banda de hasta 250 MHz y se creó una interfaz serial con la computadora para la adquisición y el almacenamiento de los datos procesados.

Capítulo 1

Introducción

Los sistemas de comunicaciones digitales nos proporcionan un conjunto de ventajas respecto a los analógicos, pero, cuando se trabaja con frecuencias elevadas es necesario implementar algunas de las etapas de manera analógica. Sin embargo, hoy en día, gracias a las mejoras de los circuitos convertidores analógico a digital o ADC, se puede conseguir una buena representación discreta de la señal en una frecuencia cada vez más elevada, lo mismo sucede con los convertidores digital a analógico o DAC que pueden llevar a cabo interpolación, con lo que se puede transmitir a frecuencias cada vez más altas, esto nos ofrece la oportunidad de implementar nuevas arquitecturas de receptores y transmisores digitales. La base de los radios digitales es que, mientras la digitalización de los mismos esté cada vez más cerca de la antena, los sistemas pueden incrementar su capacidad para manejar mayores frecuencias y anchos de banda [1]. Por lo tanto en el mercado inalámbrico, donde los requerimientos de los distintos usuarios son cada vez más exigentes y se encuentran en constante cambio, los radios digitales se utilizan para solucionar problemas tales como la interacción de sistemas dispares, la adaptación a nuevas tecnologías emergentes y la miniaturización, pues proporcionan una forma de construir radios más eficiente, flexible, fácil de manufacturar y menos costosa, para implementarse pueden usarse combinaciones de tecnologías FPGA, de procesamiento digital de señales o digital signal processing (DSP) y circuitos integrados para aplicaciones específicas o application specific integrated circuit (ASIC), dependiendo de cada caso en particular. Los radios digitales también son llamados radios definidos por software o software defined radio (SDR). Eric Blossom definió el término como la técnica de acercar el código a

la antena tanto como sea posible, convirtiendo los problemas de hardware en problemas de software [2].

1.1. Antecedentes

1.1.1. Motivación

Los trasmisores y receptores inalámbricos de radio han evolucionado a través del tiempo hasta lo que se conoce como un equipo de radio reconfigurable, es decir, un equipo capaz de realizar distintas funciones solo llevando a cabo cambios en su configuración mediante software, lo que es comúnmente llamado radio digital, definido por software o SDR, término acuñado por Joseph Mitola en 1991 [3]. Estos radios surgieron de la necesidad de tener un dispositivo de propósito general para resolver incompatibilidades entre las tecnologías inalámbricas sin recurrir a opciones costosas; pues un mismo equipo es capaz de trasmitir y recibir distintos anchos de banda y frecuencias. La primera implementación importante de un SDR fue en un proyecto militar estadounidense llamado SpeakEas, cuyo objetivo era conjuntar más de 10 tipos de tecnologías de comunicaciones inalámbricas en un equipo programable, operando en una banda de 2 a 200 MHz, y además con un código que debía de poder actualizarse para adaptarse a estándares futuros [4].

Comercialmente existen varias opciones para implementar radios digitales: los universal software radio peripheral (USRP), que se diseñan utilizando computadoras de propósito general, llevan a cabo las tareas de modulación, demodulación, y procesamiento en una unidad central de proceso o central processing unit (CPU) y las operaciones de conversión, decimación e interpolación en un FPGA, lo cual da la libertad de trabajar con relativamente poco esfuerzo y presupuesto. La combinación de un hardware flexible y software de código abierto hacen de los USRP una plataforma ideal para desarrolladores [5]. También podemos encontrar chips que funcionan como receptores, transmisores, sintetizadores y en general cualquier componente que se encuentre en un radio, como por ejemplo el AD6620 de Analog Devices, que incluye un receptor digital en cuadratura completo, con traslación de frecuencia, decimación y etapas de filtrado [6]. Estos chips, a diferencia de los USRP tienen pocas capacidades de reconfiguración, lo que los hace ideales para aplicaciones de propósito específico. Por otro lado, el sistema que se

presenta, basado en un FPGA, solo depende de un CPU para el almacenamiento de la información resultante y no está restringido a un rango de frecuencias ni a una arquitectura en específico, obteniendo así un sistema flexible que podrá procesar señales ya sea de resonancia magnética nuclear, ultrasonido o cualquier otro tipo mientras su frecuencia se encuentre dentro del rango de los convertidores analógico a digital y digital a analógico a utilizar.

La tecnología SDR ha ganado popularidad en áreas tales como la detección y monitoreo del espectro en tiempo real, en gran parte debido a que los FPGAs han hecho posible el procesamiento de alta velocidad; en ellos se puede implementar fácilmente aplicaciones como la trasformada rápida de Fourier o fast Fourier transform (FFT), filtros de respuesta impulso finita o finite impulse response (FIR) y en general cualquier algoritmo que requiera de operaciones de multiplicación y acumulación [7]. Como se mencionó, una de las aplicaciones de los radios digitales está en los sistemas de resonancia magnética nuclear (RMN) de bajo costo, con propósitos educativos y de investigación, en los cuales además de llevar a cabo las tareas de conversión a banda base, se implementan programadores de pulsos para RMN con resolución en el orden de los nano segundos, teniendo así un transceptor completo; si además se usa conversión digital directa, se tiene una alternativa a las arquitecturas superheterodinas convencionales [8]. Como los sistemas para la adquisición de señales para imágenes por resonancia magnética son generalmente costosos, es común recurrir al hardware programable para diseñarlos a la medida [9]. En [10] por ejemplo, se implementa un sistema integrado en un FPGA, para uso en laboratorio, el cual permite llevar a cabo las tareas de cualquier sistema comercial, haciendo uso de un lenguaje de descripción de circuitos electrónicos como el VHSIC hardware description language (VHDL), para describir los módulos digitales encargados de la programación de pulsos, demodulación en cuadratura, filtros duales; así como, una interfaz con la PC. Por otro lado, en los experimentos *pulsed-electron paramagnetic resonance* (EPR), un programador de pulsos basado en tecnología FPGA proporciona ventajas extra, ya que es de bajo costo y alto desempeño en sincronización y control; en estos experimentos se requiere la generación de series constituidas por dos pulsos de ancho variable, separados un cierto tiempo y repitiéndose con una periodicidad determinada [11]. Son muchos los trabajos donde se han aprovechado las ventajas de los radios digitales, y los FPGAs no son los únicos dispositivos empleados, algunos trabajos combinan el uso de los FPGAs con los DSPs, aprovechando las ventajas de cada arquitectura para

obtener un sistema completo [12]. Por lo tanto, se quiere diseñar e implementar un radio digital que sea capaz de reconfigurarse, con el objetivo de trasmitir y recibir señales de distintos anchos de banda y frecuencias. El sistema además contará con la generación de pulsos, lo que permitirá tener un equipo de radio digital útil para investigación.

1.1.2. Radios digitales y técnicas de muestreo

Los transmisores y receptores de radio son interfaces esenciales entre la información digital y las ondas electromagnéticas, en las cuales el procesamiento realizado se puede dividir en *front-end* y banda base. En general el transmisor se encarga de la generación de la forma de onda en banda base (codificación, constelación, etc.), de las conversiones de tasa de muestreo, la traslación de la frecuencia a una frecuencia mayor y, además, analógicamente, de la amplificación y limitación de la señal. El receptor traslada la frecuencia a una menor; la limita; realiza conversiones de tasa de muestreo y conversión a banda base; ecualiza, detecta y decodifica; y, además, extrae la información de sincronización que es una parte importante del proceso de recepción. Como se ha mencionado, debido al crecimiento exponencial de las comunicaciones, se están incrementando los requerimientos de procesamiento; lo cual ha llevado a una búsqueda continua de nuevas tecnologías con mayor capacidad, menor costo y menor tamaño. La arquitectura de los receptores digitales era comúnmente superheterodina, donde se traslada primero la señal a una frecuencia intermedia o intermediate frequency (IF), para ser procesada después. Esta arquitectura ha sido la dominante desde 1930 [13]; sin embargo, la conversión directa ha venido a cambiar esto por su nivel de integración y bajo consumo; teniendo como principal ventaja la traslación de la señal directamente a banda base, lo que se traduce en menos consumo de corriente y filtros más simples [14]. Los transmisores de RF enfrentan otros retos, la modulación por ejemplo, involucra consideraciones entre la eficiencia del ancho de banda y la potencia, lo que determina el tiempo de uso y el máximo rango posible para el transmisor [15].

En cuanto a la implementación de los radios digitales, los FPGAs son ideales, ya que son circuitos que se componen de un gran número de compuertas lógicas programables, de forma que se pueden implementar circuitos digitales con lenguaje de descripción de hardware; como los elementos lógicos pueden ejecutarse en paralelo, los FPGAs ofrecen un procesamiento de alta velocidad. Esta tecnología fue inventada en 1984 por Xilinx,

y desde entonces ha estado evolucionando hasta convertirse en parte importante de las industrias de medición, aeroespacial, de imágenes médicas, consumo, comunicaciones y procesamiento de señales. Una de las principales razones por la cual los FPGAs se han vuelto tan populares, es la cantidad de instrucciones por segundo que es capaz de ejecutar. Por ejemplo, en un DSP la implementación de un filtro de 256 etapas usando una unidad de multiplicación y acumulación o multiply accumulate (MAC), convencional tardaría 256 ciclos de reloj; en un FPGA, usando celdas para procesamiento digital de señales embebidas, se logra alcanzar un desempeño superior, pues el filtro se ejecutaría en un ciclo de reloj, independientemente de la longitud del mismo [16]. La tecnología FPGA es una de las más completas y flexibles para el desarrollo de aplicaciones de alto desempeño, siendo posible diseñar soluciones digitales a la medida, en menos tiempo y a bajo precio.

Una de las especificaciones necesarias para el diseño de un radio digital es el ancho de banda. El ancho de banda es aquel que el convertidor es capaz de digitalizar adecuadamente. La tasa de muestreo del convertidor es el parámetro que define el ancho de banda que es capaz de muestrear el sistema; si nos basamos en Nyquist, por ejemplo, un convertidor de 200 millones de muestras por segundo o million samples per second (MSPS) tendrá un ancho de banda máximo de 100 MHz. El teorema general de muestreo de Nyquist dice que podemos recuperar una señal a partir de sus muestras si estas se obtienen con una tasa de al menos dos veces la componente en frecuencia más grande de la señal. El solapamiento espectral se da en este tipo de muestreo a menos que se impongan condiciones muy estrictas en los filtros, lo que resulta en diseños más complicados, por lo tanto se aplican técnicas alternativas dependiendo del tipo de sistema que tengamos, pudiéndose llevar a cabo un sobremuestreo o un submuestreo. Ajustar la frecuencia de muestreo de la señal de interés es una tarea común en el procesamiento digital de señales y a un sistema que maneja diferentes frecuencias de muestreo se le conoce como multitasa [17]. Por un lado, el sobremuestreo permite relajar las especificaciones del filtrado pues separa las copias del espectro, si bien el filtro es menos complicado, se tienen que trabajar a frecuencias más altas. También podemos obtener una mayor resolución en la conversión analógica a digital si se sobremuestrea y se calcula la media de las muestras, lo cual generalmente es más barato que añadir bits a los convertidores; por ejemplo, un ADC de 12 bits puede funcionar con la precisión de 24 si se sobremuestrea por un factor de 64; finalmente, si la señal no está correlacionada con el ruido, sobremuestrear por un

factor y hacer la media, reduce la varianza del ruido y por lo tanto la relación señal a ruido mejora. Por otra parte, el submuestreo nos permite recuperar la señal original si se muestrea adecuadamente por debajo de la frecuencia de Nyquist, con las ventajas que conlleva el procesamiento a una menor velocidad.

1.1.3. Señales de resonancia magnética y ultrasonido

La RMN es una técnica ampliamente usada en la industria de la imagenología médica, que cuenta con la ventaja de no ser una técnica invasiva. Un sistema típico de RMN consiste en un sistema de campo magnético, un sistema de RF y uno para la reconstrucción de la imagen. El sistema de RF consta de un receptor y un transmisor de RF, con un generador de onda que se encarga de emitir una secuencia de pulsos. En general, las imágenes por RMN se basan en la absorción y emisión de energía en el rango de radio frecuencia del espectro electromagnético, proporcionando detalles acerca del tejido del cuerpo. Entonces, en el receptor, el sistema detecta y procesa las señales generadas cuando los átomos de hidrógeno, que siempre son abundantes en los tejidos, son colocados en un campo magnético fuerte y excitados por pulsos de resonancia magnética. En este caso se aprovecha que los átomos de hidrógeno tienen un momento magnético inherente, como resultado de su spin, el cual, al colocarse en un campo magnético fuerte, tiende a alinearse. Como resultado del no alineamiento perfecto, se presenta un movimiento de precesión a una frecuencia determinada por la fuerza del campo magnético, que se conoce como frecuencia de Larmor. La estimulación adecuada por el campo magnético resonante, a la frecuencia de precesión del núcleo de hidrógeno, puede forzar al momento magnético del núcleo a inclinarse a un plano perpendicular al campo aplicado. Al remover el campo de excitación de RF, los momentos magnéticos del núcleo se realinean, generando una señal de RF a la frecuencia de Larmor, la cual se detecta y se usa para generar la imagen. Para las imágenes de RMN se usan frecuencias desde 1 a 200 MHz.

En la parte del transmisor, las secuencias de pulsos son una serie de pulsos de radiofrecuencia aplicados en el tiempo, de manera secuencial, ordenada y predefinida que provocan una modificación específica sobre la orientación de los spins, que luego se pueden relacionar con algún parámetro molecular a partir del análisis del espectro de RMN resultante. Una de las secuencias básicas es la secuencia spin echo, que se caracteriza por la aplicación inicial de un pulso de radiofrecuencia de 90° seguido por uno de 180°,

luego del doble del tiempo entre estos dos pulsos se recibe una señal o eco proveniente del tejido estimulado [18]. Generar estas señales de alta precisión y pureza desde una representación digital, ayudada de convertidores de alta velocidad, tiene la ventaja de que todos los cambios de frecuencia mantienen la fase y velocidad de conmutación. Estas secuencias en un FPGA se pueden implementar mediante la técnica de síntesis digital directa o direct digital synthesis (DDS).

Por otro lado, los dispositivos médicos para formación de imágenes por ultrasonido usan ondas en el rango de 1-20 MHz y dependiendo del tipo de aplicación, se usan transductores de alta frecuencia o de baja frecuencia [19]. En principio, un sistema de ultrasonido enfoca ondas ultrasónicas en un punto de interés, pero conforme las ondas se propagan, estas se reflejan en cualquier objeto que encuentren a lo largo de su trayectoria, la medición de las ondas reflejadas o eco, contiene información espacial y de contraste para la formación de la imagen [20].

Tanto el campo de la imagenología por ultrasonido como por RMN está en constante progreso, por lo tanto muchos investigadores han construido sus propios receptores, transmisores y programadores de pulsos con FPGAs para desarrollo e investigación, pues con un solo chip se pueden realizar todos los trabajos digitales requeridos, tales como la programación de pulsos, la demodulación en cuadratura, el filtrado paso bajos y también las interfaces con la PC para el control y la transferencia de datos, de aquí la importancia de orientar a este tipo de señales este trabajo [21].

1.2. Objetivo

En este trabajo se propone el desarrollo de un sistema que además de recibir y transmitir señales de frecuencia y ancho de banda reconfigurables, también sea capaz de producir pulsos de excitación con aplicaciones a señales de resonancia magnética nuclear y ultrasonido, dos aplicaciones populares de los radios digitales. Para lograr este objetivo se siguieron los siguientes pasos:

- *Revisión de la información relacionada con las señales a procesar.* Se han elegido señales de resonancia magnética y de ultrasonido para verificar la funcionalidad del sistema, considerando que ambas nos permiten obtener imágenes médicas; sin

embargo, la información que contienen las imágenes resultantes no es la misma, por tanto es importante conocer la naturaleza de cada una de las señales así como determinar los rangos de frecuencia y anchos de banda particulares a usar.

- *Consideración del Hardware disponible.* Análisis de sus alcances y limitaciones: la tarjeta de desarrollo del FPGA tienen un reloj con una frecuencia máxima de 100 MHz; el convertidor analógico digital tiene una resolución de 14 bits y funciona a 65 MSPS, aceptando señales con un ancho de banda de hasta 250 MHz; y en cuanto al convertidor digital analógico, tiene una resolución de 14 bits a 300 MSPS lo que lo hace ideal para aplicaciones de síntesis digital.

- *Definición de la arquitectura del sistema.* En este caso el sistema se compone de 5 subsistemas: transmisor, receptor, generador de pulsos, comunicación y control. La implementación de estos se lleva a cabo con lenguaje de descripción de Hardware y con herramientas de simulación y depuración: Xilinx ISE, ModelSIM, Matlab y System Generator, con el cual se verifica el funcionamiento de cada componente. El análisis de trabajos previos y del trabajo que comenzó con el desarrollo de este proyecto [22] es importante para mejorar el diseño de los diversos componentes.

- *Realización de pruebas.* La verificación del sistema se lleva a cabo utilizando un generador de señales de RF, capaz de producir señales entre 100 KHz y 1 GHz, que además cuenta con funciones de modulación, con lo cual es suficiente para poner a prueba el sistema. La información resultante se enviará a la computadora mediante una interfaz universal asynchronous receiver transmitter (UART) para su almacenamiento.

Los pasos mencionados previamente lograrán que esta tesis aporte fundamentalmente la solución a la traslación de frecuencia de la señal a procesar a banda base y, con una interfaz sencilla, permitir el estudio de dicha señal.

1.3. Organización de esta tesis

En este *primer capítulo* se ha presentado el objetivo de esta tesis, así como algunos de los trabajos realizados sobre radios digitales orientados a la adquisición de datos para

imágenes médicas, con el propósito de mostrar la importancia de la misma. También se ha presentado una introducción general a cada uno de los temas a tratar.

En el *capítulo 2*, se mostrarán los fundamentos teóricos para desarrollar la tesis. La naturaleza de las señales a tratar, las diferentes técnicas para la adquisición de la información, los conceptos básicos tanto de receptores como de transmisores digitales y también una introducción a las arquitecturas reconfigurables, así como las diferentes herramientas para el uso de éstas.

En el *tercer capítulo* se detalla el diseño del sistema, basado en los aspectos teóricos revisados con anterioridad, y se muestra la estructura de los diferentes susbsitemas que componen el radio digital. Debido a la implementación en FPGA se muestran algunas consideraciones especiales a tomar en cuenta cuando se usan estos dispositivos.

En el *capítulo 4* se muestra la implementación del sistema en el FPGA y su funcionamiento con los convertidores analógico digital y digital analógico. Se detallan las pruebas que se llevaron a cabo para verificar la funcionalidad de cada uno de los componentes del sistema así como en general, y los diferentes problemas con los que se lidió.

En el *capítulo 5* se resumen los objetivos logrados y se proponen mejoras para el sistema, de manera que trabajos futuros puedan hacer uso de la experiencia adquirida en este trabajo. Finalmente, en los apéndices se explican aspectos técnicos que serán de utilidad a quien tome este trabajo ya sea para llevar a cabo pruebas o para desarrollar nuevos sistemas.

Capítulo 2

Marco teórico

2.1. Técnicas de muestreo

Los convertidores analógico a digital y digital a analógico con alta tasa de muestreo y rango dinámico permiten desarrollar receptores y transmisores con menos componentes y por lo tanto más sencillos y más baratos. Las técnicas de muestreo que se utilicen, ya sea sobremuestreo o submuestreo, dependerán de la aplicación a la que estos sistemas estén orientados; por ejemplo, para algunas de las aplicaciones de banda estrecha conviene aplicar el sobremuestreo, con el propósito de esparcir el ruido de cuantización sobre todo el espectro del ADC y así permitir al filtrado remover mejor dicho ruido [6].

El teorema de Nyquist dice que cualquier señal puede ser representada por muestras discretas si el muestreo es de al menos el doble del ancho de banda de la señal. La Figura 2.1 muestra un par de señales f_0 y f_a siendo muestreadas a una frecuencia f_s, en la cual se puede observar que para todas las frecuencias debajo de $f_s/2$, como por ejemplo f_0, se cumple el criterio de Nyquist, es decir, cualquier señal presente en la región sombreada se representará correctamente. Pero si tenemos una frecuencia f_a de una señal mayor a $f_s/2$, el muestreo generará una réplica del espectro de f_a en $f_s - f_a$. Una vez que ha ocurrido esto, no hay forma de distinguir si este espectro se ha originado debido a una réplica o si es de alguna componente de la señal original en esa frecuencia, por lo tanto este problema tiene que evitarse. Una forma de resolver esto, es colocar un filtro pasa bajas antes de la conversión analógica digital, para remover todas las componentes en frecuencia por arriba de $f_s/2$ de la señal [23].

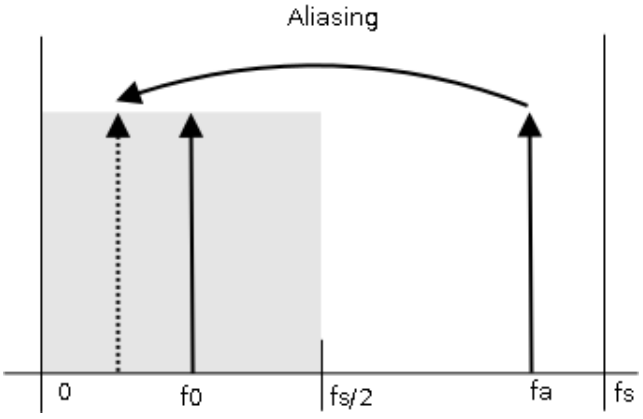


FIGURA 2.1: Efecto del muestreo en el espectro de una señal [23].

Para mostrar las propiedades del muestreo, imaginemos que podemos representar la señal de interés sobre una hoja de papel a la que se le han hecho dobleces como en la Figura 2.2. El eje horizontal representa la frecuencia y se dobla la hoja en múltiplos enteros de la mitad de la frecuencia de muestreo, f_s; entonces, cada sección en la hoja de papel representa lo que podemos llamar una "zona de Nyquist".

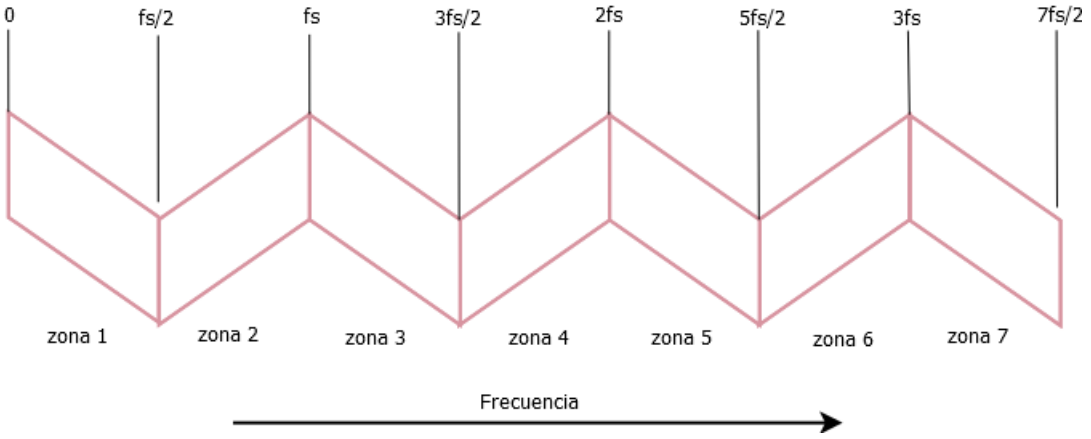


FIGURA 2.2: Zonas de Nyquist representadas sobre hoja de papel [24].

Si plegáramos el papel con una señal distribuida como en la Figura 2.3, lo que veríamos sería una mezcla de todos los segmentos de la señal que se encuentran en las distintas zonas; esto es análogo a muestrear a una frecuencia f_s que no cumple el criterio de Nyquist, lo cual hace que ocurra solapamiento y, una vez que esto ha ocurrido, es imposible volver a separar la señal.

Para prevenir la destrucción de la señal, tenemos que asegurarnos de que ésta se encuentre contenida dentro de la primera zona de Nyquist. Una señal en banda base tiene componentes en frecuencia que comienzan en 0 y terminan en algún máximo, para lograr tener una señal limitada en banda, se insertan filtros pasa bajas que eliminan las

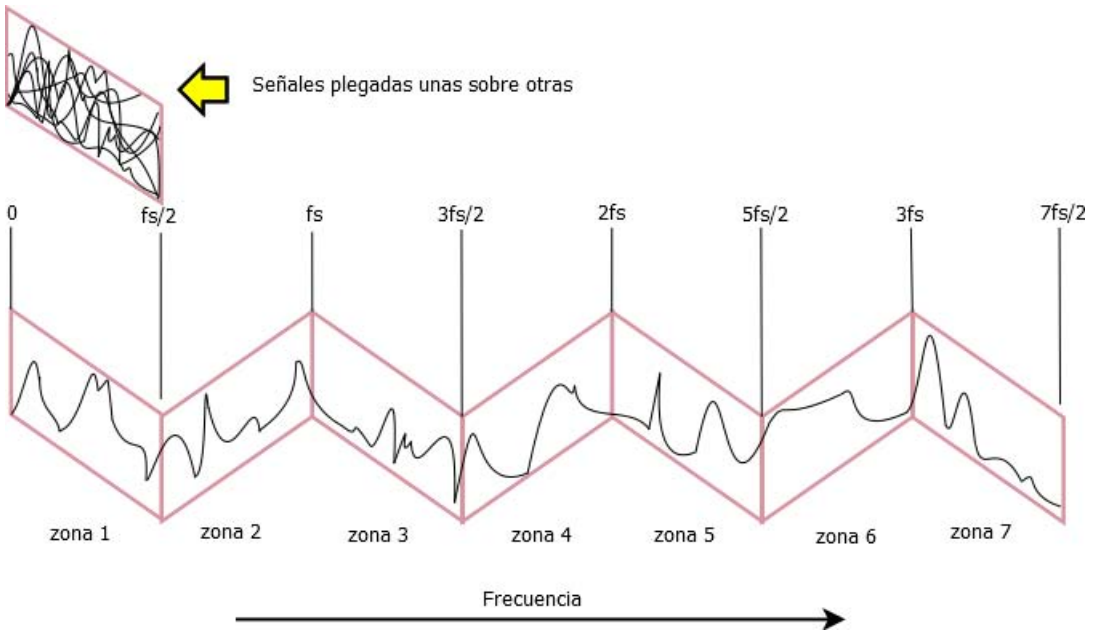


FIGURA 2.3: Efecto de solapamiento sobre hoja de papel [24].

componentes por encima de $f_s/2$, o se incrementa la frecuencia de muestreo de manera que abarque todas las componentes de la señal Figura 2.4.

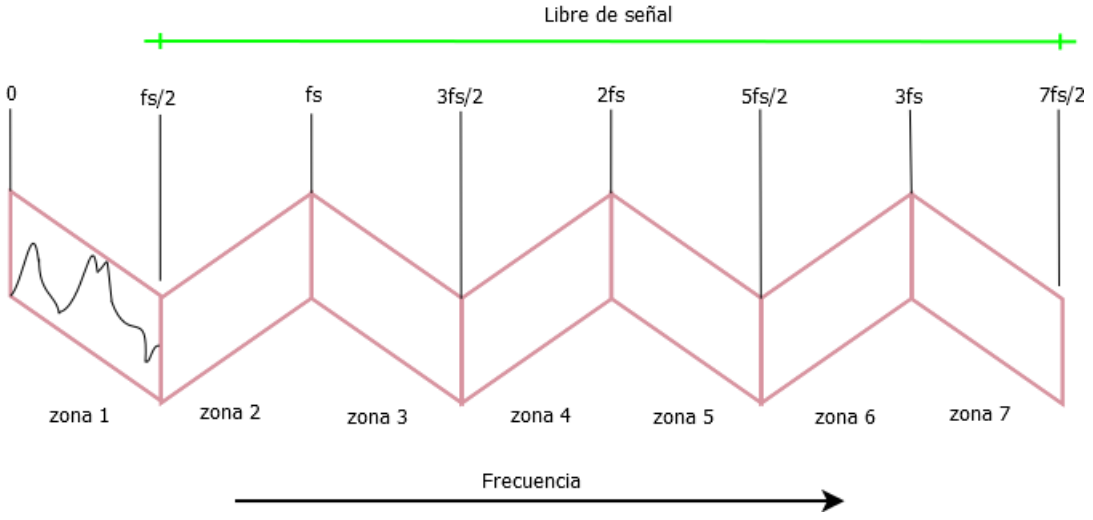


FIGURA 2.4: Efecto del filtrado sobre hoja de papel [24].

Por otro lado las señales paso-banda tienen un espectro diferente de cero solo sobre un rango de frecuencias $f_L < |f| < f_H$, donde f_L y f_H son la menor y la mayor frecuencia respectivamente. El ancho de banda de este tipo de señales se define como $B = f_H - f_L$. Usando el criterio de Nyquist se necesitaría una frecuencia de muestreo de $2f_H$, sin embargo, sabemos que el proceso de muestreo genera imágenes del espectro de la señal espaciados en armónicos de la frecuencia de muestreo usada, y por lo tanto podemos

tomar ventaja de esto relacionando la señal paso-banda con una de sus imágenes y así muestrear a una frecuencia menor a $2f_H$ determinada por $B << f_L$. Como la señal se puede muestrear a una frecuencia menor sin perdida de información, se dice que la señal ha sido submuestreada; podemos submuestrear una señal sin solapamiento espectral siempre que la frecuencia de muestreo f_s satisfaga el teorema del muestreo paso-banda [25].

2.1.1. Submuestreo y sobremuestreo

Un submuestreo es una forma de disminuir los tiempos de adquisición en un receptor digital ya que tomamos menos muestras de las que necesitaríamos normalmente haciendo uso de los efectos del muestreo sobre el espectro de la señal. Para ilustrar esta técnica, considere una señal paso-banda con centro en 70 MHz y ancho de banda de 10 MHz, normalmente para el muestreo se usaría una frecuencia de 150 MHz; sin embargo, usando submuestreo podemos proponer una frecuencia menor, supóngase f_s de 40 MHz.

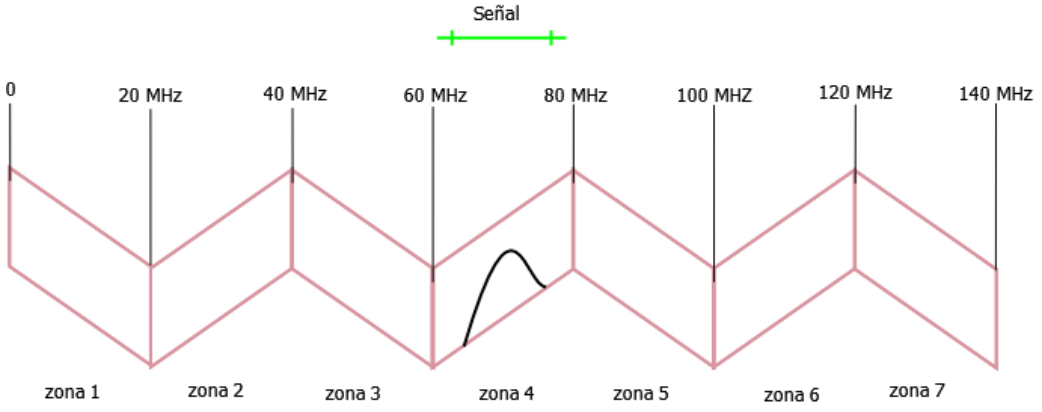


FIGURA 2.5: Señal paso-banda sobre hoja de papel [24].

Si representamos la señal de interés sobre la hoja de papel, Figura 2.5, vemos que esta se encuentra contenida en la zona de Nyquist 4, de $3f_s/2$ a $2f_s$, es decir, en las frecuencias desde 60 MHz hasta 80 MHz; al plegar el papel, el espectro que buscamos se mantendrá pues no hay señal en ninguna otra zona. Sin embargo, un factor que hay que tener en cuenta, es que dependiendo en qué zona se encuentra la señal, al plegar el papel, el espectro correspondiente se conservará o se invertirá. En el caso de la zona 4, éste se invierte, lo cual no es difícil de corregir en un receptor digital. También, para asegurarnos que no haya otras señales fuera de nuestro rango de interés, se necesita insertar un filtro paso-banda, que, debido a efectos reales en la implementación, es conveniente rebajarlo

a un ancho de banda de $0.8f_s/2$. De esta forma es posible reconstruir la señal deseada usando una frecuencia de muestreo menor, en este caso 40 MHz.

Para señales paso banda, con límite inferior f_L y límite superior f_H, la condición para un submuestreo correcto se reduce a hacer que los desplazamientos, a múltiplos enteros de la frecuencia de muestreo f_s, tanto para la banda positiva como para la negativa no se superpongan. Para lograr esto, se debe cumplir con el teorema del muestreo paso banda, el cual indica que la frecuencia de muestreo debe cumplir con la condición:

$$\frac{2f_H}{n} \leq f_s \leq \frac{2f_L}{n-1} \tag{2.1}$$

Cuando n satisface:

$$1 \leq n \leq \frac{f_H}{f_H - f_L} \tag{2.2}$$

Cuanto mayor es n menor puede ser la frecuencia de muestreo f_s. Dependiendo de la frecuencia de muestreo que se escoja, las réplicas más cercanas a la frecuencia cero pueden resultar invertidas (el componente de las frecuencias positivas en la banda de las frecuencias negativas y a la inversa) es decir se ha generado una inversión del espectro. Siguiendo el estudio gráfico realizado por O. Brigham en [25], se puede establecer que cuando n es par el espectro se recupera invertido y si n es impar no invertido. Con la nueva frecuencia de muestreo, $\hat{f}_s$, el número de muestras se reduce en un factor de $f_s/\hat{f}_s$ con lo que la velocidad de procesamiento se reduce; así como el orden del filtro y el almacenamiento de información. Otra ventaja es que los polos del filtro que se implemente con una menor tasa de muestreo están más lejos del circulo unitario, es decir los efectos de la longitud de palabra en el filtro digital son menores.

Muestrear a una tasa más elevada que el ancho de banda de la señal se conoce como un sobremuestreo. Este tipo de muestreo permite separar las réplicas espectrales de la señal y facilitar las condiciones de filtrado; sin embargo, por ejemplo, si se lleva a cabo un sobremeustreo por un factor de 10, entonces también se tendrá una tasa de datos 10 veces mayor para almacenar y procesar, con la ventaja de que obtenemos una mejor calidad de la señal sin una precisión muy alta de los componentes diseñados.

La técnica de sobremuestreo es útil cuando se necesita capturar bordes rápidos, transitorios y eventos que solo ocurren una vez, además de que permite reducir el ruido de cuantización. El ruido de cuantización total del ADC o varianza σ^2, es el valor del bit menos significativo o q, elevado al cuadrado y sobre 12.

$$\sigma^2 = \frac{q^2}{12} \tag{2.3}$$

Si se considera el ruido de cuantización como ruido aleatorio, entonces, en el dominio de la frecuencia tiene un espectro plano; es decir esta distribuido igualmente desde $-f_s/2$ a $+f_s/2$, Figura 2.6. Por lo tanto la densidad espectral de potencia o power spectral density (PSD) del ruido de cuantización es:

$$PDS_{noise} = \frac{q^2}{12f_s} \tag{2.4}$$

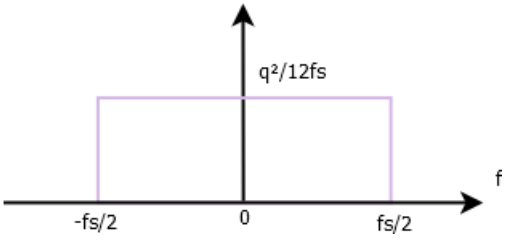


FIGURA 2.6: Densidad espectral de potencia del ruido de cuantización.

Para reducir el ruido, hay que reducir q, lo cual puede lograrse usando un ADC con mayor número de bits, pero ya que esto a veces resulta poco práctico, se puede también aumentar el denominador de la ecuación 2.5, es decir, incrementar f_s a $\hat{f}_s$. De esta forma esparcimos el ruido sobre todo el espectro. La mejora se muestra en:

$$SNR_{A/D} = 10log_{10}(\hat{f}_s/f_s) \tag{2.5}$$

Entonces existen dos formas de alterar la tasa de muestreo: reducir el número de muestras, incrementando el periodo de muestreo, y aumentando las muestras, reduciendo el periodo de muestreo. Un sobremuestreador con factor de muestreo de L, o expansor por L, inserta $L-1$ muestras nulas entre muestras consecutivas, haciendo que el espectro de salida sea una versión comprimida del espectro de la entrada; esta compresión de las réplicas centradas en los múltiplos de 2π hace que aparezcan $L-1$ copias nuevas (o

imágenes) en el intervalo $[-\pi, \pi]$. Un submuestreador, con factor de muestreo de M o compresor, es un sistema que selecciona una de cada M muestras de la señal de entrada y desecha las otras $M-1$ muestras, con lo cual la señal se comprime por un factor M. El proceso de eliminar muestras en el dominio del tiempo añade componentes espectrales de alta frecuencia, teniéndose en el dominio de la frecuencia una versión expandida por un factor M del espectro de la señal de entrada [26].

2.1.2. Muestreo en cuadratura

Debido a la limitación de recursos espectrales, la mayoría de los sistemas de comunicaciones son paso-banda, es decir, trabajan en rangos de frecuencia bien definidos. Además, el ancho de banda usado se puede considerar muy estrecho en comparación con el valor de la frecuencia central del canal. Estas propiedades tienen implicaciones importantes, puesto que podemos obtener representaciones paso bajo equivalentes de las señales preservando toda la información. En esas condiciones, la señal paso-banda $s(t)$ se puede escribir según dos señales reales paso bajo:

$$s(t) = s_I(t)cos(\omega_c t) - s_Q(t)sin(\omega_c t) \tag{2.6}$$

Donde ω_c es la frecuencia portadora. Según la ecuación 2.6, se puede construir una señal paso-banda, de banda estrecha, modulando dos señales reales paso bajo con dos portadoras de la misma frecuencia y desfasadas 90°. Como una rotación angular de 90° corresponde con un ángulo recto, se suele decir que estas señales están en cuadratura, y la componente $s_I(t)$ recibe el nombre de componente en fase y $s_Q(t)$ componente en cuadratura. La frecuencia portadora suele ser mucho más grande que el ancho de banda de las señales paso bajo $s_I(t)$ y $s_Q(t)$. Entonces este tipo de muestreo resulta conveniente ya que la tecnología actual permite procesar señales en banda base sin dificultades, al contrario de lo costoso que resulta procesar directamente la señal paso-banda; por eso, disponer de representaciones banda base de $s_I(t)$ y $s_Q(t)$ tiene un alto interés práctico [27]. Las características de estas representaciones son complejas y pueden ser procesadas en equipos digitales como microprocesadores, dispositivos de lógica programable FPGAs, procesadores digitales de señales DSPs, o circuitos integrados de propósito específico ASIC.

2.2. Resonancia magnética y ultrasonido

Resonancia magnética, RM

La resonancia magnética es una técnica que se basa en el magnetismo y la electricidad. Para comprender la teoría y práctica de la RM, y sus aplicaciones, hay que entender los fundamentos del magnetismo y la electricidad. Los fenómenos magnéticos son fundamentalmente de naturaleza eléctrica, lo cual se aprecia al considerar que la corriente eléctrica creará un campo magnético y un campo magnético genera movimiento de electrones. Si tenemos un campo magnético oscilante alrededor de un alambre, se induce un voltaje y existirá corriente eléctrica; en los equipos de RM se crea una corriente de este tipo. Las señales de RM son voltajes o corrientes eléctricas variables en el tiempo, que se manifiestan como ondas que son inducidas por un campo magnético oscilante, cuya forma de onda refleja la información de la señal. En RM la radiación se transmite de manera fragmentada, en pulsos que se pueden agrupar en secuencias. Existen diferentes tipos de secuencias que se utilizan para crear distintos cortes en el objeto de estudio. En cuanto a la forma de los pulsos, un pulso rectangular invariante no tiene relevancia en la práctica de la RM; para dar aplicabilidad a los pulsos es necesario darles forma y variar su amplitud en el tiempo, las formas más usadas son la Gaussiana y la sinc.

En resonancia magnética es muy importante considerar diversos aspectos relacionados con la materia. La materia se compone de átomos, los cuales difieren unos de otros por su estructura interna, que da lugar a distintas propiedades físicas. Las propiedades magnéticas de los núcleos atómicos son la base del fenómeno de resonancia magnética. Cuando los núcleos atómicos con propiedades magnéticas se colocan en un campo magnético pueden absorber algunas ondas electromagnéticas con frecuencias características. Esta frecuencia característica depende del tipo de núcleo, la intensidad del campo y el entrono del núcleo. La absorción y reemisión de estas ondas es el fenómeno básico en el que se basa la RM. El átomo de hidrógeno es el más utilizado en RM debido a que los dos componentes principales del cuerpo humano, agua y grasa, contienen hidrógeno.

Una masa giratoria (o spin) con carga eléctrica crea un pequeño momento magnético que generará un movimiento de precesión alrededor de la dirección del campo externo. La frecuencia de este movimiento de precesión ω viene dada por la Ecuación 2.7, llamada ecuación de Larmor.

$$\omega = \gamma \, B_0 \tag{2.7}$$

Donde ω es la frecuencia angular de Larmor, γ es la relación giromagnética que viene dada por la relación entre las propiedades magnéticas y mecánicas del núcleo y B_0 es la fuerza del campo magnético en Tesla. La resonancia se producirá cuando una onda electromagnética con la frecuencia apropiada (igual a la frecuencia de Larmor o frecuencia natural) alcance los núcleos. Estos núcleos absorberán energía y pasarán del estado de menor energía al estado de energía más alta. La Tabla 2.1 muestra algunas frecuencias de excitación típicas en RM para núcleos de hidrógeno (H), flúor (F), fósforo (P) y sodio (Na) ante diferentes intensidades de campo magnético [28].

Intensidad de campo magnético	Frecuencia (MHz)			
(T)	1H	19F	31P	23Na
0.1	4.3	4	1.7	1.1
0.3	12.8	12	5.1	3.4
0.5	21.3	20	8.6	5.6
1.0	42.6	40	17.2	11.3
1.5	63.9	60	25.9	16.9
2.0	85.2	80	34.5	22.5
3.0	127.8	120	51.8	33.8
4.7	200	188	81	52.9
9.4	400	376	162	105.8
11.7	500	470	203	131.6

Tabla 2.1: Dependencia entre la intensidad de campo y la frecuencia.

Entonces, en presencia de un campo externo, los núcleos con propiedades magnéticas similares y que tienen un movimiento de precesión alrededor de la dirección del campo externo, dan lugar a un momento magnético único o magnetización neta. Para detectar esta magnetización neta es necesario alejarla del eje del campo principal, lo cual se consigue con un impulso electromagnético a la frecuencia de resonancia. El pulso de RF provocará una inclinación de la magnetización neta, cuyo valor dependerá de la intensidad y duración del pulso. Cuando los espíns empiezan a regresar a su equilibrio, emiten una señal que tiende a ser cero con el tiempo, llamada caída libre de la inducción o free induction decay (FID). Si el campo magnético no es homogéneo, las diferentes partes de la muestra se ven sometidas a diferentes intensidades de campo y por tanto tendrán diferentes frecuencias de Larmor. Cuando la señal es digitalizada, se pueden analizar sus componentes en frecuencia y determinar la intensidad y fase de cada componente.

Existen dos parámetros importantes en la RM. El proceso de vuelta al estado de equilibrio desde un estado excitado, que se denomina proceso de relajación spin-red o relajación longitudinal, el cual viene caracterizado por el tiempo de relajación T_1, que es el tiempo requerido para que el sistema recupere el 63 % de su valor de equilibrio después de ser expuesto a un pulso de 90 °. El parámetro T_1 refleja las propiedades físico químicas del entorno de los núcleos observados y por lo tanto permite caracterizar muestras biológicas complejas constituidas por diferentes compuestos químicos. Después de que un sistema de spín se haya excitado por un pulso de RF, inicialmente se comporta como un sistema coherente donde los componentes tendrán un movimiento de precesión en fase, con el tiempo la señal observada disminuye y los movimientos de precesión dejan de estar en fase debido a las diferencias de las frecuencias de Larmor inducidas por diferencias en los campos magnéticos estáticos en distintas localizaciones de la muestra. Este proceso se conoce como relajación spin-spin o relajación transversal y se caracteriza por el parámetro T_2

Ultrasonido

El sonido es una serie de vibraciones mecánicas capaces de producir una sensación auditiva. Los infrasonidos son vibraciones con frecuencias por debajo del umbral de sensibilidad humana (menores a 20 Hz) y, por otro lado, los ultra sonidos son ondas acústicas inaudibles con frecuencias por encima del umbral de sensibilidad humana (mayores a 20 KHz). El generador ultrasónico general se puede considerar constituido de un elemento primario y un elemento secundario, donde el primario o transformador convierte la señal eléctrica del elemento secundario en energía mecánica, haciendo vibrar el medio circundante y provocando ondas de presión a altas frecuencias. El elemento secundario es el que proporciona la señal de excitación (eléctrica, magnética, mecánica). El ultrasonido es una técnica de imagen sencilla, no invasiva y accesible, que permite la evaluación del sistema musculoesquelético en tiempo real. Cuando la energía acústica interactúa con los tejidos corporales, las moléculas del mismo se alteran levemente y la energía se transmite de una molécula a otra adyacente. De esta manera la energía acústica se mueve a través del tejido mediante ondas longitudinales y las moléculas del medio de transmisión oscilan en la misma dirección que la onda. Estas ondas sonoras corresponden básicamente a la rarefacción y compresión periódica del medio en el cual se desplazan.

La velocidad de propagación del sonido varía dependiendo del tipo de material que atraviese, siendo los factores relevantes la densidad y la comprensibilidad. A medida que las ondas ultrasónicas se propagan a través de las diferentes capas de una sustancia, las ondas pierden potencia y su intensidad disminuye progresivamente. Cuando un haz ultrasónico se propaga en un medio, una parte de él se refleja a manera de eco y otra parte pasa a través del segundo medio. El eco al llegar al transductor se convierte en una pequeña onda de voltaje. El circuito receptor puede determinar la amplitud de la onda sonora de retorno y el tiempo de propagación total, con lo cual se puede calcular la profundidad del tejido usando además la velocidad del sonido. La amplitud de la onda permitirá establecer la tonalidad de gris que debe asignarse a la imagen [29].

Como ya se refirió antes, los sistemas de ultrasonido trabajan en frecuencia mayores a los 20 KHz y los sistemas médicos de ultrasonido usan frecuencias en el rango de 1 a 20 MHz. Las frecuencias bajas ofrecen imágenes con poca resolución, pero tienen un poder de penetración mayor y las frecuencias altas se atenuarán fácilmente, por lo que se usan transductores de entre 10 y 15 MHz para imágenes superficiales y frecuencias de entre 2 y 5 MHz para estructuras más profundas. A veces las ondas de ultrasonido se generan en pulsos que comúnmente consisten en dos o tres ciclos de la misma frecuencia, los cuales tienen una frecuencia de repetición (pulse repetition frequency, PRF) determinada de tal manera que haya suficiente tiempo entre pulsos para permitir a la señal llegar a la región de interés y reflejarse, generalmente para dispositivos de imágenes médicas se usan PRF de entre 1 a 10 KHz [19].

EL ultrasonido es una de las modalidades más usadas en imagen médica, debido a que provee imágenes de alta resolución sin el uso de radiación ionizante. Un sistema de ultrasonido esta conformado por diversos elementos. Entre estos se tiene una unidad de control que se encargará de sincronizar la generación de las ondas de sonido y de la medición de las ondas reflejadas. Este controlador debe conocer la región de interés en ancho y profundidad, para trasladar la región en una serie de líneas de exploración. La unidad de control debe decir a cada elemento del transductor cuando activarse y cuanto tiempo para cubrir toda la región de interés. Dependiendo de la modalidad para la formación de la imagen pueden llevarse a cabo varios procesos. El sistema puede estar basado en una onda pulsada o pulse wave (PW); pero también existe la alternativa donde se usa una onda continua o continous wave (CW). Los sistemas CW se usan generalmente cuando se requiere de una medición más exacta de la información de la

velocidad, sin embargo tienen el inconveniente de que se usan transductores separados para transmisión y recepción.

Las funciones esenciales en el procesamiento de sistemas de ultrasonido son: filtrado, detección y compresión. El filtrado reduce el ruido en las frecuencias fuera del rango de interés. La detección puede usar varios métodos. Si se usa una representación analítica de la transformada de Hilbert, se tiene la ventaja de que no importa que tipo de frecuencia o método de imagen se use; sin embargo es más complicada de implementear; si se usa una demodulación compleja seguida de un filtro, sin duda es una operación más simple, pero se necesita saber la frecuencia de operación. El rango dinámico de la señal recibida depende de los bits de resolución del ADC, generalmente se usa un compresor para ajustar el rango a uno que pueda desplegarse visualmente. Cabe destacar que se puede agregar procesamiento más complicado al sistema, como compensación en frecuencia, combinaciones de frecuencia, control de ganancia, promediado del eco, suavizado, detección de bordes, entre otros [30].

2.3. Receptor y transmisor digital

La arquitectura clásica en los receptores digitales fue por mucho tiempo la superheterodina, mostrada en la Figura 2.7, donde se pueden ver etapas de mezclado y filtrado analógicas; sin embargo, como se ha mencionado, los nuevos convertidores de alta velocidad permiten llevar a cabo los receptores digitales con arquitectura directa, tal como lo muestra la Figura 2.8, adquiriendo así las ventajas de tener un mayor número de componentes digitales.

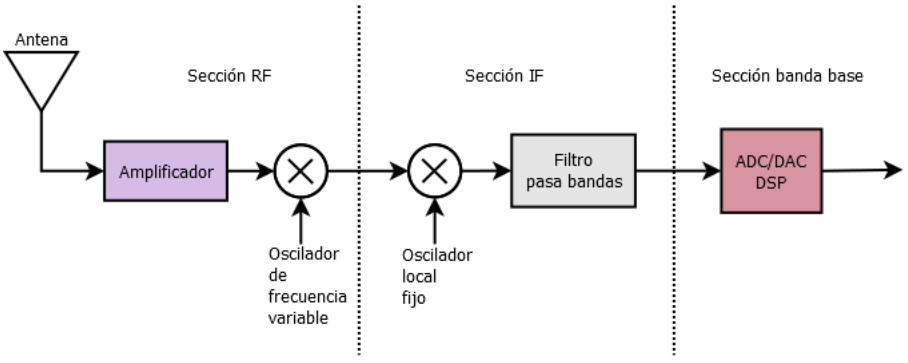


FIGURA 2.7: Arquitectura superheterodina.

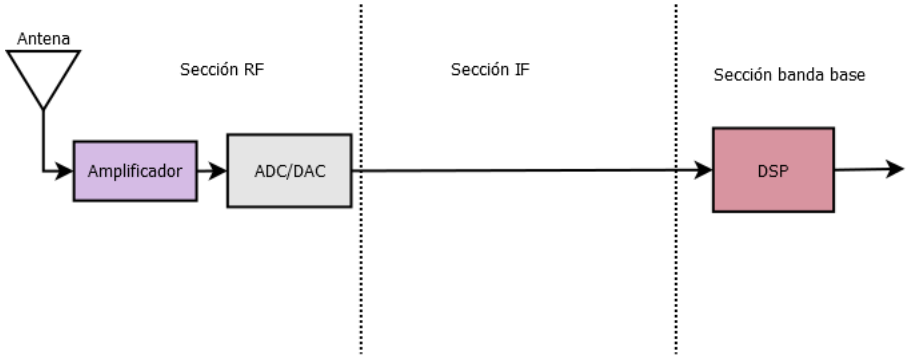


FIGURA 2.8: Arquitectura directa.

En la Figura 2.9 tenemos un diagrama de bloques de un receptor digital, donde el bloque principal corresponde al digital down converter (DDC). Los DDC son un componente esencial en cualquier sistema con base en *software radio*, simplificando el diseño de la sección de RF del receptor. Actualmente con la tendencia de los sistemas de manejar un ancho de banda cada vez mayor, se requiere que los DDC se implementen en FPGAs colocados después del ADC [7].

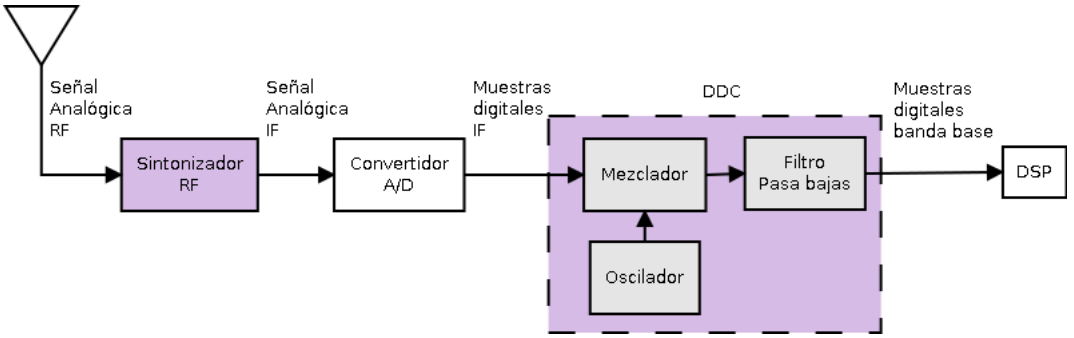


FIGURA 2.9: Sistema de recepción [24].

La primera etapa de un DDC es el mezclador digital complejo, mostrado en la Figura 2.10, el cual traslada la frecuencia de interés a banda base, utilizando un par de multiplicadores y un sintetizador digital directo o direct digital synthesizer (DDS), que hace la función de oscilador numéricamente controlado o numerically controlled oscillator (NCO) para la generación de la señales senoidal y cosenoidal necesarias. La segunda etapa reduce la frecuencia de muestreo de la señal, utilizando comúnmente filtros cascaded integrator comb (CIC) para decimar la información y agregar una etapa de ganancia. Para alimentar al mezclador, el DDS genera muestras digitales de dos ondas senoidales, desfasadas por 90°, un seno y un coseno. La función principal de esta etapa es la de convertir la señal de RF, como se ve en la Figura 2.11, cuando se tienen dos señales $f1$ y f_2, el mezclador generará $f_1 \pm f_2$.

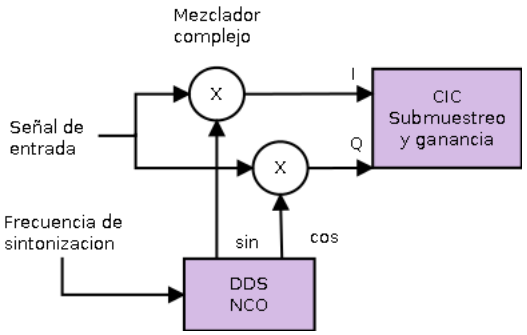


FIGURA 2.10: Mezclador digital [23].

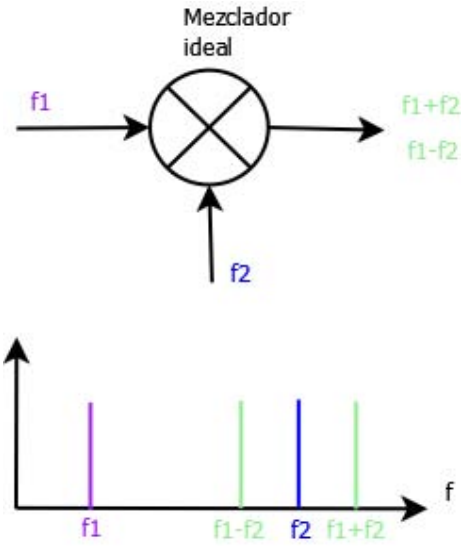


FIGURA 2.11: Mezclador ideal.

Entonces las muestras del ADC se multiplican por las señales digitales seno y coseno del oscilador, generando las salidas en cuadratura, I y Q, que son importantes para mantener la información de fase contenida en la señal. El objetivo es sintonizar el oscilador para centrar la señal de interés alrededor de 0 Hz de tal forma que el filtro pasa bajas deje pasar solo a esta señal. El filtro se programa con el llamado factor de decimación para limitar el ancho de banda de la salida [23]. Los DDC permiten recorrer a la señal de su frecuencia portadora a banda base, lo que reduce el procesamiento subsecuente. Hay dos clases de DDC, de banda ancha y de banda estrecha, cuya diferencia radica en sus factores de decimación; si el factor es menor de 32 se considera de banda ancha y si es mayor a 32 de banda estrecha. El filtrado es diferente dependiendo del tipo de DDC, sin embargo el decimador es igual para ambos. Con señales de banda ancha, el principal reto en un receptor es el de tener suficiente procesamiento para filtrar la señal; esto generalmente se lleva a cabo mediante una separación del filtro, de manera que los filtros subsecuentes operen a una tasa mucho menor. Con señales de banda estrecha los

retos en los DDC son diferentes, ya que en este caso necesitamos filtros que permitan grandes factores de decimación. Una buena opción son los filtros CIC, los cuales pueden implementar decimación y proporcionar buena frecuencia de corte con pocas etapas, usando solamente sumadores y retardos; el inconveniente de los "ripples" generados en su banda de rechazo, se puede compensar con un filtrado extra.

El transmisor digital presenta típicamente una estructura como la mostrada en la Figura 2.12, teniendo una concepción similar a la del receptor, pero en este caso con un digital upconverter (DUC) que traslada la señal en banda base a IF, para luego alimentar al DAC. La señal de salida de éste requiere además de amplificación para obtener la señal RF final.

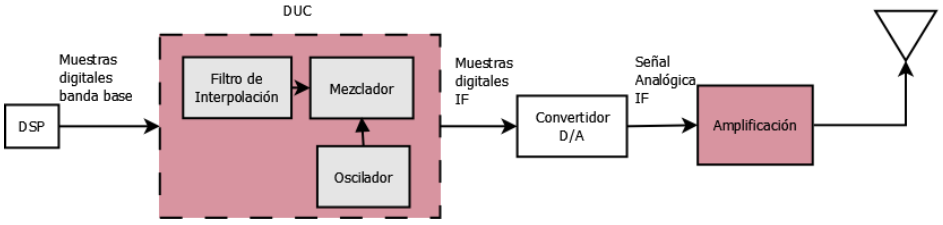


FIGURA 2.12: Sistema de transmisión [24].

En el DUC, el mezclador y el oscilador trasladan la señal de banda base a frecuencia IF. La frecuencia se determina con el oscilador y debe igualar a la frecuencia de muestreo f_s del convertidor. Como la señal en banda base viene a una frecuencia mucho menor, se necesita un filtro de interpolación, el cual debe de llevar la señal de entrada que se encuentra a una frecuencia f_s/N a la frecuencia que requiere el mezclador f_s. El filtro de interpolación incrementa la frecuencia de muestreo de la señal en banda base por un factor N conocido como factor de interpolación [24].

Entonces, con la interpolación y la decimación se reducen considerablemente los requisitos del filtrado y además el ruido de cuantización se esparce sobre una región amplia con respecto al ancho de banda de la señal original, con lo que también se logra una mejora en la relación señal a ruido [31].

2.4. Hardware

2.4.1. Arquitecturas reconfigurables

Los ASICs y los DSPs han sido usados para procesamiento asociado con las radiocomunicaciones, sin embargo, los FPGAs están haciéndose presentes en aplicaciones de radio digital debido a su capacidad de trabajar en paralelo. Actualmente los vendedores de FPGAs han incluido librerías de propiedad intelectual o intelectual property (IP) para manejar tareas como modulación, decodificación, síntesis, entre otras, lo cual reduce considerablemente el tiempo de diseño. Por lo tanto los sistemas basados en FPGA son plataformas ideales cuando se necesitan características fuera de lo comercial, o que sean variables; sin embargo, también hay que tener en cuenta sus desventajas, como poca memoria y que ciertas operaciones pueden consumir muchos recursos, como las divisiones, las matriciales y el uso de punto flotante.

A continuación se describe la estructura básica de un FPGA. Los tres elementos básicos son: los bloques lógicos configurables o configurable logic block (CLB), las interconexiones y los bloques de entrada/salida, como se muestra en la Figura 2.13.

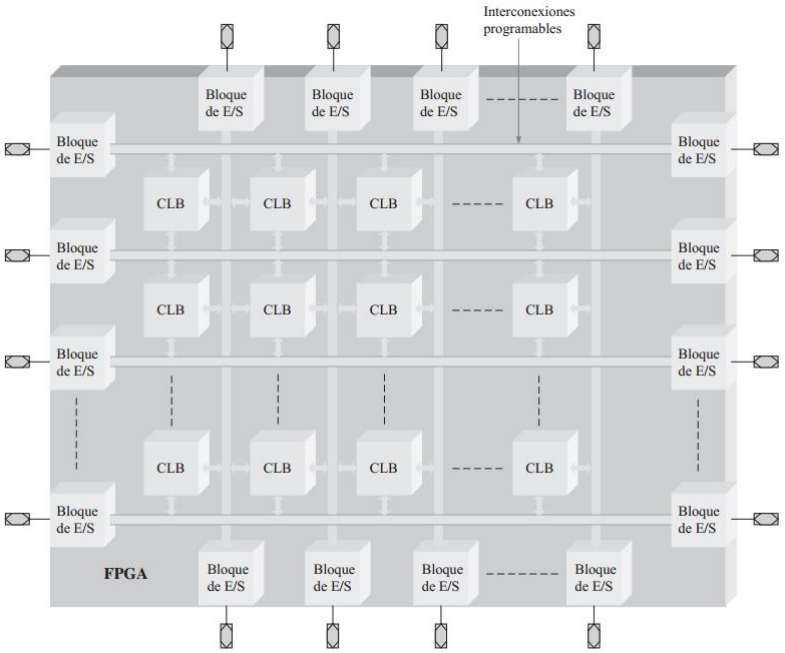


FIGURA 2.13: Estructura básica de un FPGA [32].

Cada CLB está formado por múltiples módulos lógicos mas pequeños y por una serie de interconexiones locales programables. Un módulo lógico basado en tablas de búsqueda

o look up table (LUT), que es un tipo de memoria programable utilizada para generar funciones combinacionales [32]. Los FPGAs también contienen módulos hardware, los cuales son una parte de la lógica dentro de un FPGA, que el fabricante incluye para proporcionar una función específica y que no puede reprogramarse; la ventaja de estos módulos es que se consumen menos recursos comparado con la programación de la aplicación por el diseñador y por lo tanto se requiere menos tiempo de diseño. Algunos de estos módulos son bloques de memoria, multiplicadores y buffers globales para reloj.

Los FPGAs pueden manejar una carga computacional grande cuando se implementan arquitecturas paralelas, siendo ideales para diseños multicanal. Si además se cuenta con celdas dedicadas para procesamiento digital, como las DSP *slice*, que permiten implementar en un ciclo de reloj operaciones que en un DSP necesitarían múltiples ciclos de reloj, entonces tendremos una plataforma ideal para programar radios digitales.

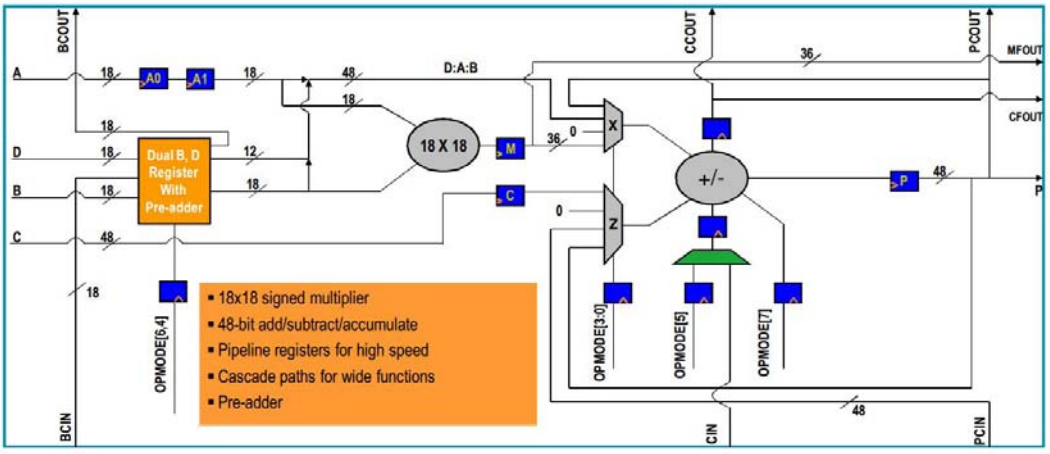


FIGURA 2.14: Segmento DSP de un Spartan 6 [33].

La familia de FPGAs con la que se cuenta, Spartan6 de Xilinx, es de bajo consumo, tanto dinámico como en sus entradas y salidas y en la funcionalidad de su IP. Los bloques de memoria son de 18 Kb y pueden funcionar hasta 300 MHz y con diferentes configuraciones para los puertos de lectura y escritura.

Algunas de las razones por las cuales se usan FPGAs para la implementación de radios digitales son: se puede aumentar la velocidad de los filtros usando arquitectura paralela como se ve en la Figura 2.15; los FPGAs se pueden usar para diseños multi-canal que corren en paralelo; y la arquitectura configurable compensa el costo con el desempeño obtenido. Por otro lado, la ventaja de usar los *XtremeDSP Slice* se aprecia en el desempeño, pues por ejemplo, con una implementación normal, un filtro de 32 etapas

consumirá 1461 celdas lógicas [33], ya que el direccionamiento se implementará con lógica y además se tendría una latencia variable debido al ruteado.

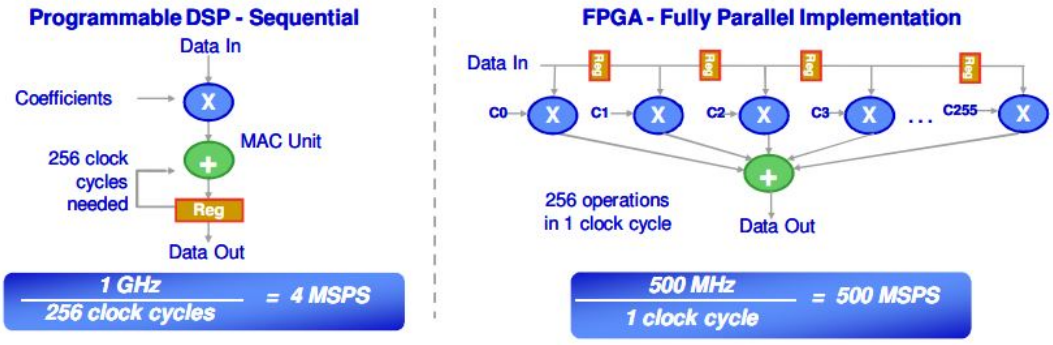


FIGURA 2.15: Comparación de un filtro digital en un DSP y en un FPGA [33].

2.4.2. Herramientas de diseño

Para programar FPGAs, generalmente, se usa lenguajes de descripción de hardware o hardware description language (HDL), que permiten especificar a un nivel más alto que la lógica de compuertas. VHDL es uno de los lenguajes más utilizados para describir sistemas electrónicos digitales y fue estandarizado en 1987 por el Instituto de ingenieros eléctricos y electrónicos o Institute of electrical and electronics engineers (IEEE). Este lenguaje permite una descripción funcional o de comportamiento del circuito o una descripción de la estructura del diseño, declarando de forma jerárquica las entidades y subentidades; además permite simular el diseño y sintetizarlo [34].

Xilinx provee diversas herramientas para la programación de los FPGAs. ISE Design Suite es la plataforma de Xilinx para la programación con HDL de FPGAs, permitiendo usar tanto VHDL como Verilog. Un método de diseño basado en simulación es muy útil al programar FPGAs, pero a medida que los sistema se vuelven más complejos, los requerimientos de tiempo para las simulaciones pueden ocasionar que estas fallen, por lo tanto es mejor llevar acabo un monitoreo de la implementación directa en el FPGA. Para este tipo de pruebas, Xilinx cuenta con ChipScope, que permite cargar los diseños en el FPGA y analizarlos con entradas de prueba. Por otro lado, mientras que VHDL permite especificar el comportamiento de un circuito a alto nivel, System Generator es una herramienta de aun más alto nivel, con la cual podemos programar en el entorno gráfico Simulink de Matlab. System Generator provee bloques con los que

podemos construir modelos y además de un generador de hardware, que se encarga de transformar estos modelos en HDL. Además, con Simulink podemos generar bloques de prueba y visualizar las salidas del diseño. En la Figura 2.16 se muestra el esquema general de un diseño con System Generator.

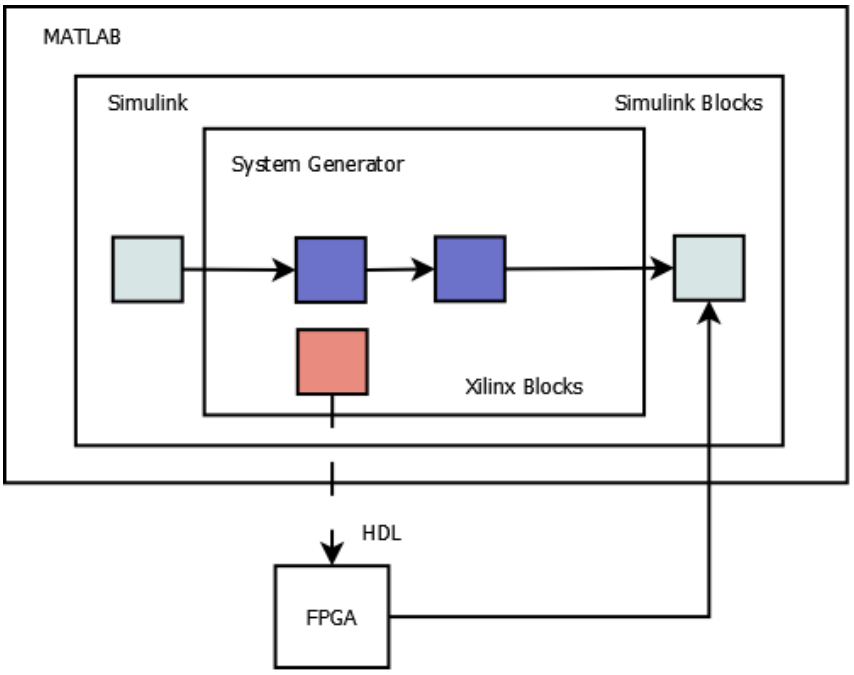


FIGURA 2.16: Esquema de diseño en System Generator [35].

Los bloques de Simulink nos sirven de apoyo tanto para generar estímulos para el diseño como para visualizar, analizar y almacenar los datos resultantes; sin embargo estos bloques no se programan en el FPGA. Solo los bloques de System Generator son sintetizables y por lo tanto son los únicos que se transforman en hardware en el FPGA. Desde Simulink podemos llevar a cabo simulaciones y co-simulaciones; permitiendo con las simulaciones comprobar el funcionamiento del sistema antes de llevarlo al FPGA, mientras que con las co-simulaciones, se programa el sistema en el FPGA y se monitorea a través de la interfaz joint test action group (JTAG) para verificar el correcto funcionamiento.

2.4.3. Convertidores de alta velocidad

En los sistemas de comunicaciones, los convertidores son elementos clave, que sirven de puente entre la transmisión analógica y los procesadores digitales. Los convertidores de alta velocidad se usan ampliamente, debido a que su costo ha bajado, y su poder y

velocidad han incrementado; además, algunos pueden incluso configurarse para cambiar su comportamiento según se necesite. En la actualidad la mayoría de los convertidores tienen un bajo consumo de potencia y un amplio rango dinámico [36].

Generalmente los convertidores de alta velocidad cuentan con configuraciones diferenciales, tanto para el reloj como para la señal analógica de entrada o salida, según sea el tipo de convertidor. Este tipo de configuración reduce el ruido, debido a la cancelación de las señales de modo común, y la presencia de armónicos pares, ya que la señal está desplazada 180°. Las señales diferenciales tienen un mejor desempeño armónico ya que su amplitud es la mitad de la señal en su equivalente *single-ended* y teniendo señales más pequeñas se tiene un mayor rango para operar en la región lineal del convertidor, lo que se ve reflejado en los armónicos [37]. Por estas razones es que se usan entradas diferenciales para el reloj de los convertidores, pues éste es un aspecto crítico; si no cumple con las características de ruido necesarias, el funcionamiento del convertidor en el rango dinámico y la linealidad puede verse afectada reduciéndose el SNR [38].

El AD6644 es un convertidor analógico digital, que se usa comúnmente en sistemas de banda ancha como code division multiple access (CDMA) y wideband code division multiple access (WCDMA); su uso combinado con el sobremuestreo puede poner los armónicos fuera del ancho de banda de interés, facilitando el uso de receptores con decimación, lo cual permite que el ruido de fondo se reduzca. Este convertidor cuenta con entradas diferenciales [39]. Mientras que el AD9755 es un convertidor de alta velocidad digital a analógico, con un puerto de datos dual multiplexado y que cuenta con un phase locked loop (PLL) para generar la frecuencia interna necesaria para muestrear ambos canales. También cuenta con entradas y salidas diferenciales [40].

Capítulo 3

Diseño del sistema

3.1. Consideraciones de diseño

En el diseño de radios digitales, se quiere poner lo más cerca de la antena la conversión analógica a digital y digital a analógica, mediante la implementación en circuitos digitales reconfigurables. En la práctica, este objetivo es difícil debido a los requerimientos de costo, consumo y tamaño de los sistemas actuales. Algunas de las principales características que se buscan en los equipos comerciales son: bajo consumo, banda reconfigurable, suficientes recursos para implementar distintos tipos de procesamiento, capacidad para múltiples antenas, desplegado de la información, interfaz Ethernet y bus universal en serie o universal serial bus (USB), rango amplio de temperatura de operación y costo moderado.

Para lograr reconfiguración y flexibilidad en los radios digitales, la solución más común es la de usar tanto FPGAs como DSPs. Con esto, se combina la plataforma que ofrecen los DSPs para implementar algoritmos complejos, con las altas tasas de procesamiento que ofrecen los FPGAs. Como en este trabajo el objetivo principal es el de llevar la señal de RF a banda base y generar secuencias de pulsos sencillas, y no el de implementar algoritmos para el procesamiento de dichas señales, se hace uso únicamente de un FPGA apoyado en un CPU para el almacenamiento de la información. Para la selección del ADC, debido a que se llevará a cabo el uso del submuestreo, se pueden recibir frecuencias mayores a si se usara Nyquist, así, aunque el convertidor usado tiene una tasa de muestreo de 65 MSPS, el único factor que lo limita es el ancho de banda; en cuanto a la resolución,

de 12 a 14 bits es generalmente suficiente para los requerimientos de los radios actuales. Para el DAC, en las aplicaciones típicas de radio definido por software, donde el ancho de banda es menor a 25 MHz, una transmisión de 100 MHz es suficiente, y 14 bits de resolución satisfacen los requerimientos de desempeño de los radios modernos. En cuanto a el FPGA, la familia Spartan6 es de bajo consumo y ofrece capacidades para procesamiento como multiplicadores, celdas dedicadas a DSP y bloques de memoria embebidos que amplían las capacidades del FPGA[41].

Una de las principales características del sistema a tener en cuenta a la hora del diseño es el ancho de banda, el cual es una medida del rango de frecuencia entre la mayor y la menor permitida en una señal. El ancho de banda del receptor tiene una relación directa con el SNR, un ancho de banda menor mejora el SNR pero causa distorsiones espaciales, mientras que uno mayor reduce el SNR pero permite mayores velocidades. En este trabajo el ancho de banda de trabajo para el receptor viene dado por el convertidor analógico a digital usado, en este caso es de 250 MHz. Mientras que para el ancho de banda del transmisor, que se refiere al pulso de excitación requerido para la secuencia de pulsos, se tiene que, como el convertidor digital a analógico usado está diseñado para soportar hasta una tasa de datos de 150 MSPS, pero la salida se actualiza dos veces por cada flanco de subida del *latch* de entrada, entonces se obtiene una tasa de hasta 300 MSPS de salida. El FPGA utilizado, puede generar, a partir de su reloj de 100 MHz, frecuencias de hasta 370 MHz, para usarse tanto internamente como externamente, de forma que, puede manejar ambos convertidores sin problemas.

Cuando se diseñan DUC y DDC en FPGAs, hay diversos parámetros que tener en cuenta. El cambio total de la tasa de muestreo es uno de estos factores; cundo es bajo, por ejemplo menor de 32, la interpolación y la decimación puede llevarse a cabo efectivamente con filtros FIR; en cambio, cuando la tasa de cambio es alta, los filtros FIR no llegan a ser tan eficientes debido al costo en recursos de la implementación de los multiplicadores necesarios en los FPGAs, y es mejor una combinación con filtros CIC. Los filtros CIC o Hogenauer destacan ya que su implementación en hardware es sencilla, pues se usa una combinación entre derivadores e integradores en cascada, y pueden utilizarse ya sea para subir o bajar la tasa de muestreo de la señal. Entonces, la técnica de diseño para los filtros dependerá de la tasa de muestreo, los coeficientes, el reloj y los recursos disponibles.

Con los DDS, a partir de una frecuencia de reloj fija, se puede producir señales de frecuencia ajustable, en este caso se busca la mayor calidad posible en la generación de las ondas necesarias. Los DDS contienen un registro de fase, un acumulador de fase y una LUT. El registro de fase almacena la palabra de control que define el salto que da el acumulador de fase en cada ciclo de reloj. El acumulador de fase es un contador que computa la dirección a ser leída en la LUT. Y la LUT transforma la fase en amplitud, ya que para cada una de las direcciones dadas por el acumulador, provee a la salida, el valor digital de la amplitud de la señal que tiene guardada en correspondencia con el valor de la fase; este valor será convertido por el DAC. El registro acumulador de fase contiene un valor numérico de una longitud de n bits, y con cada ciclo de reloj a este valor se le suma un incremento de fase m. Si a cada valor numérico del acumulador de fase le asociamos un valor de ángulo de forma lineal, es decir, un valor de cero en el acumulador equivale a $0°$ y un valor de $111\ldots111$ equivale a $360°$, entonces, tenemos un oscilador digital cuya frecuencia de salida está dada por la ecuación:

$$f_{salida} = m\ f_{clk}/2^n \tag{3.1}$$

El número de puntos de fase discretos contenidos, está determinado por la resolución del acumulador de fase (n). Como la teoría de muestreo de Nyquist dice que necesitamos al menos 2 muestras por ciclo de reloj para reconstruir una señal periódica y continua, entonces la frecuencia máxima de salida será de $f_{clk}/2$.

3.1.1. *Xilinx LogiCORE IPCores y System Generator*

El diseño de los diversos componentes de los radios digitales, tales como los DUC y DDC se puede lograr fácilmente con FPGAs, sin embargo, los programadores deben de tener un profundo conocimiento de la arquitectura del chip para obtener el mayor potencial del dispositivo. Se puede reducir el costo y el tiempo de desarrollo haciendo uso de las herramientas de diseño y simulación de Xilinx, tales como *System Generator*, y de los bloques de propiedad intelectual *LogiCORE IP*, incrementando así la productividad [42]. Por un lado, los *LogiCORE IP* nos permiten mediante una interfaz gráfica, generar diversos tipos de componentes, mediante la especificación de sus principales características y si se tiene un conocimiento más profundo de la arquitectura del FPGA, es posible

ajustar parámetros de la implementación, por ejemplo, para crear una memoria, se puede desde simplemente dar la longitud de palabra, la profundidad y la configuración de los puertos de lectura y escritura, hasta especificar que tipo de celdas de memoria se quieren usar. Por otro, lado *System Generator* en conjunto con Simulink permite simular y programar los componentes del radio digital en un ambiente gráfico amigable, y por lo tanto es una plataforma ideal para llevar a cabo un primer acercamiento al diseño del sistema; sin embargo, presenta algunas desventajas para aplicaciones en tiempo real de alta velocidad, debido a que se requiere del uso de bloques extras para la comunicación y sincronización con Matlab. Algunas de las ventajas de trabajar con *System Generator* para simulación y depuración, son que el tiempo de programación en VHDL se reduce en un poco más del 80 %, la interfaz gráfica es de fácil compresión, se soportan simulaciones en Modelsim y Chipscope y se puede acceder a herramientas de Matlab para llevar a cabo co-simulaciones; es decir, un monitoreo del programa corriendo en el FPGA con bloques de Simulink para visualizar y almacenar los resultados [35].

Al usar *System Generator* que trabaja también con Simulink, hay que tener cuidado con aspectos como la representación de los números y la sincronización de los tiempo de muestreo entre bloques. Por ejemplo, Simulink usa el formato double, es decir un número flotante de 64 bits con complemento a dos. Ya que el punto binario se puede mover, se puede representar aproximadamente números entre $\pm 10^{-323}$ a 10^{308}, un rango deseable pero no eficiente para FPGAs. Los bloques de Xilinx por otra parte, usan un formato de n-bits con punto fijo y complemento a dos opcional, por lo tanto se requiere de una conversión para comunicar los bloques de Simulink con los bloques de Xilinx. En cuanto al muestreo, cada bloque tiene el atributo *sample period*, que corresponde a que tan seguido la función del bloque es calculada. Este periodo se relaciona con el reloj que se implementará en el hardware.

En este trabajo el uso de *System Generator* se restringió a la co-simulación de componentes clave del sistema, para así tener un mejor conocimiento de su funcionamiento y desempeño fuera de la teoría de las simulaciones. Esto también tiene la ventaja de que se pueden usar datos generados desde Matlab para probar los componentes implementados y así poder hacer pruebas previas al uso de señales de RF. La implementación de todos los bloques del radio digital está hecha en lenguaje VHDL y apoyada en los bloques de propiedad intelectual *LogiCORE IP*.

3.2. Arquitectura del sistema

El sistema consiste de cuatro módulos principales y de su respectivo módulo de control y sincronización. El módulo receptor, el transmisor, el generador de pulsos y el bloque de comunicación. En la Figura 3.1 se muestra un diagrama general de los diferentes bloques implementados en el FPGA y como se conectan con el resto del sistema.

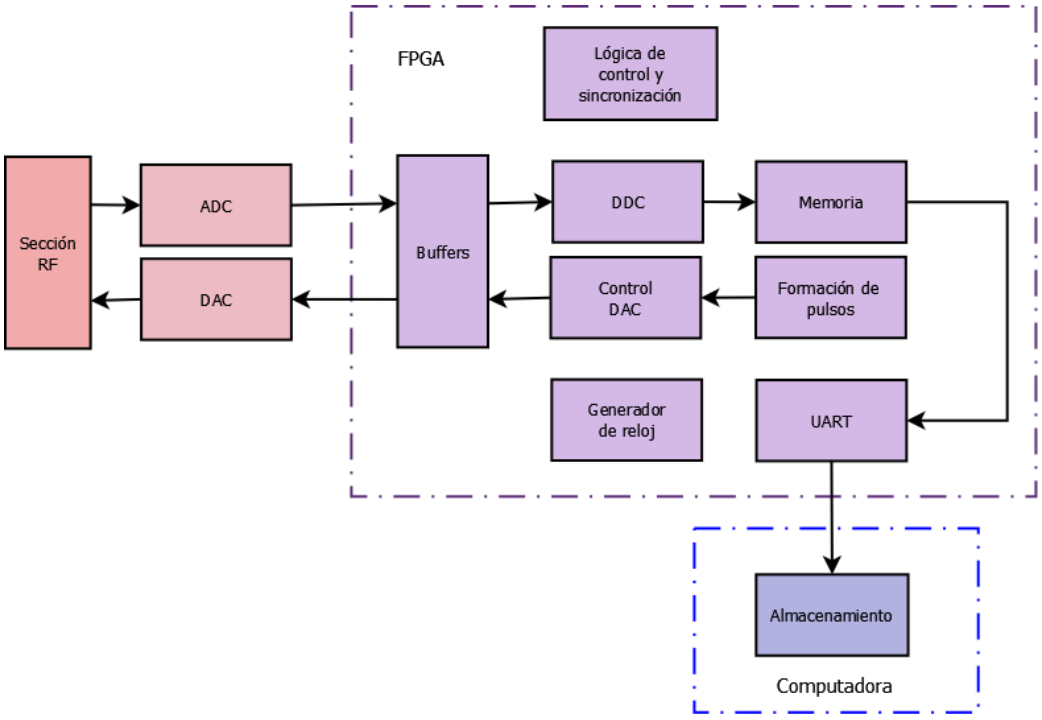


FIGURA 3.1: Arquitectura del Sistema.

3.2.1. Receptor

Este módulo esta basado en un DDC, cuyo funcionamiento en términos generales es el siguiente: se recibe una muestra desde el convertidor analógico a digital y se multiplica por una senoidal y una cosenoidal que se generan internamente con un DDS, cuya frecuencia corresponde a la frecuencia de la señal portadora; por ejemplo, para el caso de resonancia magnética, a la frecuencia de Larmor. Luego, estas señales pasan por un proceso de decimación y filtrado. La tasa de submuestreo R se puede estimar a través de:

$$R = f_{ADC}/kf \tag{3.2}$$

Donde f_{ADC} es el muestreo del convertidor, f es la frecuenica de la portadora y k el el factor de submuestreo [43]. Los datos resultantes se almacenan en una memoria de acceso aleatorio o random access memory (RAM) para ser enviados a la computadora, ya que los datos se generan más rápido de lo que pueden ser enviados a través del puerto USB y por lo tanto transmitir en tiempo real no es posible.

3.2.1.1. Adquisición de los datos

El convertidor analógico a digital funciona con un reloj de 66 MHz. En este convertidor, cuando una nueva salida digital está disponible, se activa el pin del convertidor *data ready*, para indicar que se puede tomar la muestra, y el pin *overload* si la señal de entrada al convertidor está fuera del rango de funcionamiento. Estas dos señales se usan para llevar a cabo la interfaz con el convertidor para la adquisición de los datos. En la Figura 3.2 se muestra el diagrama de flujo del programa que se encarga de la recepción de los datos.

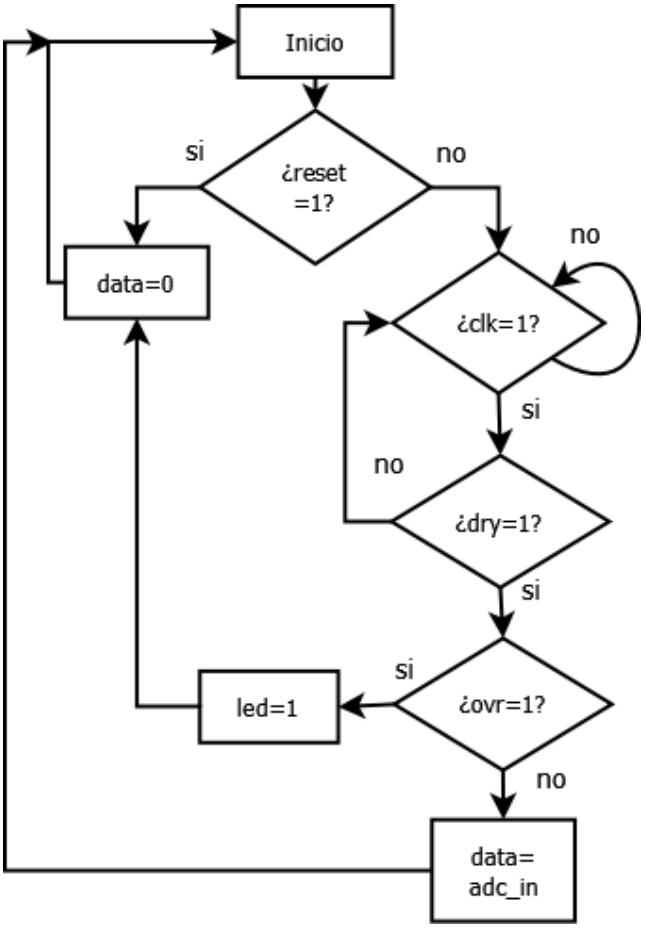


FIGURA 3.2: Diagrama de flujo, adquisición de datos.

La ejecución del proceso se llevará a cabo mientras la señal de reset se encuentre desactivada, entonces, con cada flanco de subida del reloj, se verificará el estado del pin *data ready*, para saber cuando podemos tomar la muestra. Si el pin *overload* indica que la señal esta fuera del rango del convertidor se encenderá un led en la tarjeta como aviso. El dato adquirido se guarda temporalmente en una variable para que pueda ser usado en los siguientes procesos. Cabe mencionar que el reloj con el que funciona este proceso tiene que ser de una frecuencia al menos dos veces mayor que la frecuencia con la que el convertidor genera la señal *data ready*, esto para asegurar que no se perderán datos.

3.2.1.2. Mezclador

El mezclador se conforma de un par de multiplicadores y un bloque DDS. Para generar las señales senoidal y cosenoidal, se utilizan bloques *LogiCORE IP* para aumentar el desempeño y la velocidad. Los bloques funcionan a una velocidad de 66 MHz, que es la misma velocidad a la que se reciben las muestras del ADC.

Bloque DDS

El bloque *LogiCORE IP DDS* consiste de un generador de fase y de una tabla de búsqueda (LUT) para formas de onda sinusoidales. El bloque puede generar ya sea una señal senoidal, una cosenoidal o en cuadratura, siendo esta última la configuración que se usa. La LUT se implementó con bloques de memoria BRAM de 18 K, usando un total de 4. La frecuencia de salida del bloque DDS se calcula de la siguiente manera:

$$f_{out} = \frac{f_{clk}\Delta\theta}{2^{B_{\theta(n)}}} \tag{3.3}$$

Donde f_{out} es la frecuencia de salida deseada, f_{clk} es la frecuencia del reloj del sistema, ambas frecuencias dadas en Hz, $B_{\theta(n)}$ es el ancho de fase, es decir el número de bits del acumulador de fase y $\Delta\theta$ es el incremento de fase [44]. Para calcular el incremento de fase necesario, según la frecuencia de salida deseada, se puede despejar de la Ecuación 3.3 a:

$$\Delta\theta = \frac{f_{out}2^{B_{\theta(n)}}}{f_{clk}} \tag{3.4}$$

El número de bits del acumulador de fase afecta directamente a la resolución de la frecuencia de salida, Δf, que esta dada por la Ecuación 3.5 [44].

$$\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}}} \tag{3.5}$$

La elección del número de bits para el acumulador de fase se hace tomando en cuenta que mientras más bits se tengan, mayor es la resolución pero también se necesitarán más recursos para implementar el DDS. Por lo tanto, se escogieron 32 bits para el acumulador de fase, con lo cual las ondas generadas tienen una resolución de:

$$\Delta f = \frac{66\ MHz}{2^{32}} = 0.01536 \tag{3.6}$$

La frecuencia de salida del DDS dependerá de la señal que se reciba, según el tipo de experimento. Para calcular el incremento de fase y poder generar las señales en cuadratura necesarias, tenemos la Ecuación 3.7.

$$\Delta\theta = \frac{f_{out}2^{32}}{66\ MHz} \tag{3.7}$$

Las salidas del DDS son de 14 bits cada una y cada nuevo par de salidas se estarán generando cada 66 MHz; así se obtienen las señales senoidal y cosenoidal para los multiplicadores. Cabe destacar que el bloque DDS tiene una latencia de 6 ciclos de reloj, por lo tanto hay que esperar este tiempo antes de comenzar a multiplicar la señal recibida.

Bloque multiplicador

El bloque *LogiCORE IP Multiplier* se encarga de llevar acabo multiplicaciones con alto desempeño en velocidad y en ahorro de recursos, utilizando *DSP Slices* para su implementación, una para cada multiplicador del mezclador. El dato generado por el ADC tienen una longitud de 14 bits y es signado con complemento a dos; además el dato generado por el bloque DDS tiene estas mismas características, por lo tanto, ambos pueden entrar directamente al multiplicador. La salida tiene una longitud de 28 bits y se producirá un ciclo de reloj después de haber suministrado las entradas al bloque.

3.2.1.3. Decimación

La manera más simple para reducir la tasa de muestreo consiste en eliminar muestras, es decir, un submuestreo; sin embargo, se necesita tener cuidado con los efectos que esto conlleva, ya que las copias espectrales de la señal submuestreada estarán modificadas en frecuencia y en amplitud por el factor de submuestreo. Para evitar efectos de solapamiento espectral se usa un filtro antes del proceso de submuestreo. La combinación del filtrado y reducción de tasa se le conoce como decimación. La frecuencia de corte del filtro es de:

$$0.8 * (F_s/2)/R \tag{3.8}$$

Donde R es el factor de decimación y F_s la frecuencia de muestreo. En banda ancha reducimos por un factor pequeño y en este caso funcionan mejor los filtros FIR para implementar el decimador, pues cuando estos se usan simétricamente se puede reducir la cantidad de recursos usados. Con banda estrecha, los filtros necesitan manejar factores de decimación muy grandes, por lo que se usan comúnmente los filtros CIC, cuya estructura consiste sólo de sumadores y retrasos. La función de transferencia de estos filtros viene dada por la ecuación 3.9.

$$H(z) = \left[\frac{1 - z^{-R*M}}{1 - z^{-1}}\right]^N \tag{3.9}$$

Siendo R el factor de decimación (o interpolación), N el número de etapas en el filtro y M el retraso diferencial. La respuesta en frecuencia se obtiene evaluando la ecuación 3.9 en $z = e^{2j\pi f}$, con f la frecuencia normalizada a la mayor frecuencia en la tasa del filtro; la frecuencia de entrada en caso de decimación y la frecuencia de salida, en el caso de interpolación. Así obtenemos la respuesta en magnitud dada por la Ecuación 3.10.

$$|H(f)| = \left[\frac{sin(\pi RMf)}{sin(\pi f)}\right]^N \tag{3.10}$$

El *LogiCORE IP CIC Compiler* soporta tanto decimación como interpolación y se puede programar una tasa de cambio de muestreo de entre 4 y 8192; las etapas del filtrado también son programables, entre 3 y 6; y, acepta datos en complemento a dos de hasta

24 bits. El bloque se implementa con *DSP Slices* y el número dependerá de los parámetros elegidos; por ejemplo, para 24 bits de entrada, 3 etapas de filtrado y un factor de decimación de 10, se necesitarán 4 *DSP Slices*.

Para interactuar con el CORE se usan las señales *New Data* (ND), *Ready for Data* (RFD) y *Filter Output Sample Ready* (RDY). RFD indica que el bloque esta listo para recibir datos y esta señal es activa en alto; ND en activo alto le indica al bloque que hay un nuevo dato en la entrada y RDY indica que hay un nuevo dato en la salida del bloque. El funcionamiento seria el siguiente: primero se espera por la señal RFD en 1, se escribe una nueva entrada en el bloque y al mismo tiempo ND debe ponerse en 1 por un ciclo de reloj, finalmente se puede leer el resultado cuando RDY sea 1 [45]. La rutina programada se muestra en la Figura 3.3.

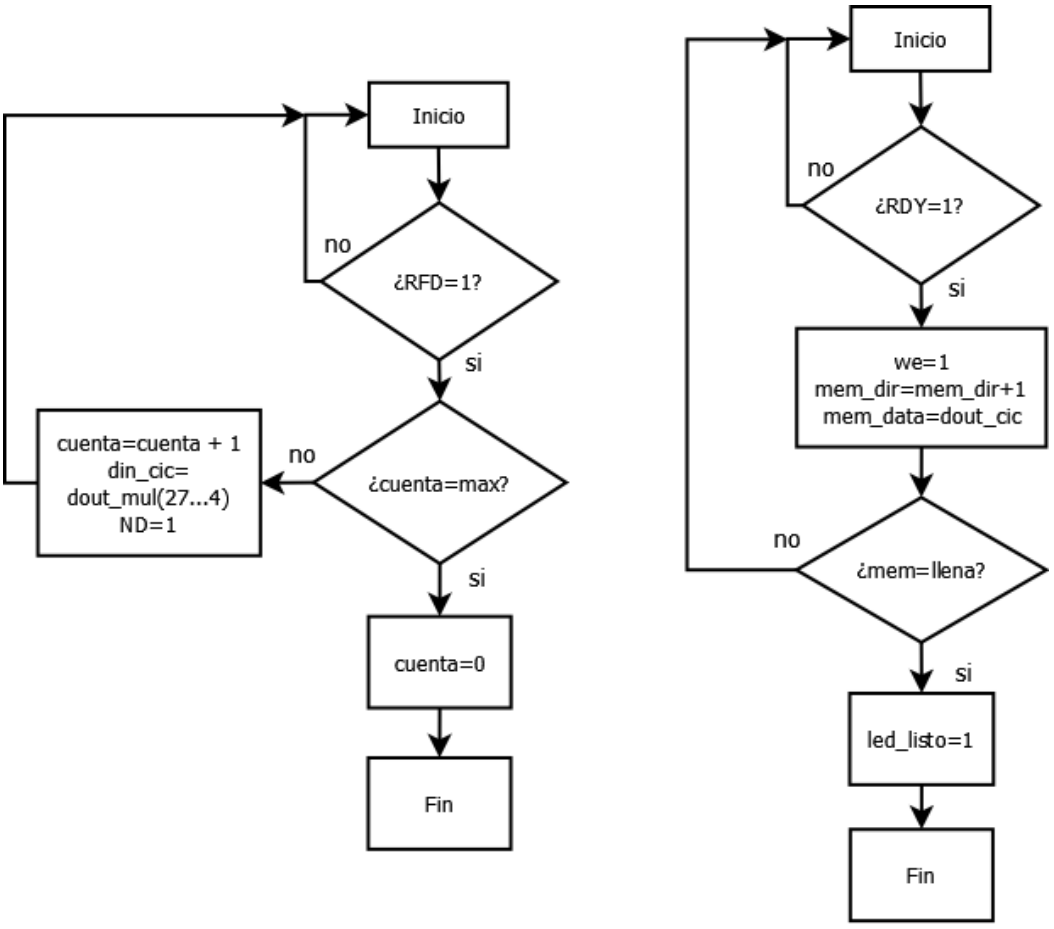


FIGURA 3.3: Diagrama de flujo, control del proceso de decimación.

En el diagrama se presentan simultáneamente dos procesos ya que estos se ejecutan de forma paralela. En el primero, se espera a que el bloque este listo para recibir una entrada, la cual viene de la salida del mezclador, pero, ya que no se soportan más de 24

bits, solo se toman los más significativos de los 28 generados a la salida del mezclador, es decir se descartan los bits menos significativos, $(3...0)$. Como la memoria para almacenar los datos resultantes es limitada, no se envían al bloque más datos, que los necesarios para llenar la memoria. Por ejemplo, si la memoria tienen una profundidad de 500, y se efectúa una decimación por 10, entonces el bloque necesita 5000 datos para llenar la memoria. Con esto se evita que el FPGA continúe procesando datos innecesariamente. Las variables cuenta y max tienen la función de verificar si se han mandado suficientes datos al bloque. Por lo tanto, si es necesario que se sigan enviando datos, activará las señales correspondientes e incrementará el contador, si no, entonces se reiniciarán las variables y terminará el proceso. El segundo proceso está continuamente verificando si el bloque ha producido una salida y, cuando ésta está disponible, habilita la escritura en la memoria. Dependiendo de los parámetros en el bloque de decimación, la longitud de la salida generada cambiará, por lo tanto, también se necesita llevar a cabo un truncamiento de los bits menos significativos en la salida, para ajustar el tamaño de palabra a la longitud de la memoria. En este proceso también se cuenta con una variable que verifica cuando se ha llenado la memoria; cuando esto sucede, se enciende un led en la tarjeta de desarrollo de FPGA que indica al usuario que se puede llevar a cabo el proceso de adquisición de los datos en la computadora.

Almacenamiento

Para llevar a cabo el almacenamiento de los datos se programó una memoria con el *LogiCORE IP Block Memory Generator*, el cual usa bloques RAM embebidos en el FPGA. Se usa una configuración *Simple Dual-port RAM*, en la cual se cuenta con puertos independientes para lectura y escritura, y por lo tanto se pueden llevar a cabo ambas funciones mientras no sea sobre la misma dirección [46]. Para escribir, basta con indicar la dirección y con activar la habilitación, we, como se ve en la Figura 3.3. Como la latencia del bloque es de un ciclo de reloj, para la lectura, al indicar la dirección deseada, hay que esperar hasta el siguiente flanco de subida para poder leer en el puerto de salida. Una de las ventajas de este bloque es que ya que los puertos de lectura y escritura son independientes, el reloj con el que funcionan puede ser distinto, así como la longitud de los datos. La longitud de la memoria se eligió de 32 bits, de manera que como se mencionó, la salida del bloque decimador en algunos casos se excederá de esta longitud, y por lo tanto se tendrá que truncar los bits menos significativos. Aunque la longitud de la memoria es de 32 bits, podemos leer la palabra almacenada en múltiplos de 8, es

decir 8, 16, 24 o 32 bits, comenzando por los menos significativos; esto es útil ya que con la comunicación UART podemos enviar un máximo de 8 bits de datos por paquete, entonces, se escoge el puerto de salida con una longitud de 8 bits tomando en cuenta que se leerán 4 veces más datos de los que se escriben.

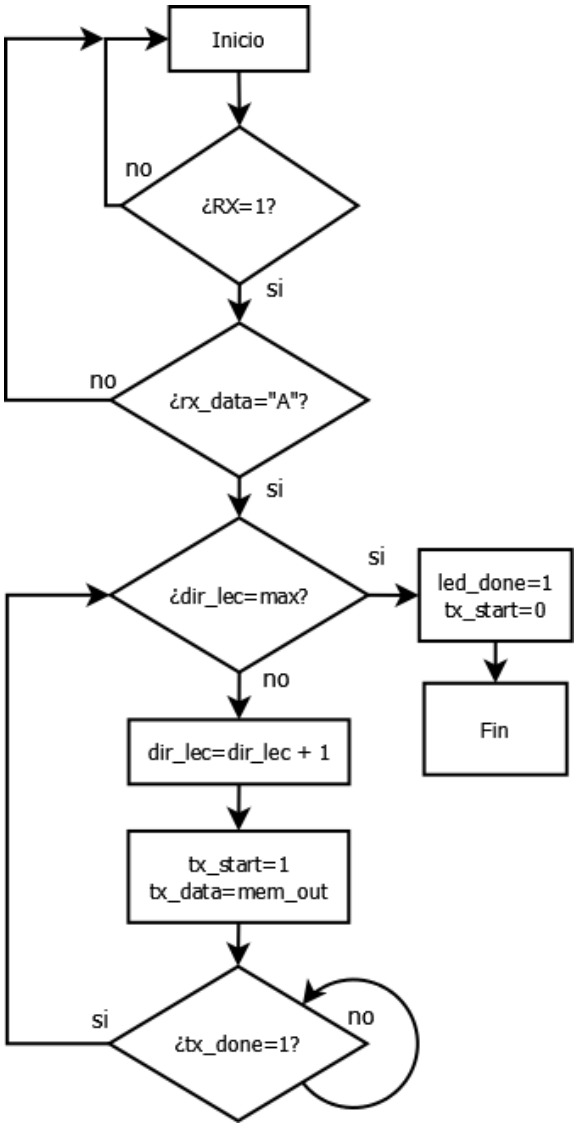


FIGURA 3.4: Diagrama de flujo, envío de datos a computadora.

En el diagrama de la Figura 3.4 se observa el proceso que se encarga de enviar los resultados del DDC a la computadora. Primero se espera por la señal de activación desde Matlab, en este caso es la letra A, y entonces se comienza el envío de los datos. Se manda al bloque de memoria la dirección deseada, después de un ciclo de reloj el dato se manda junto con la activación de envío del bloque TX del UART, luego se espera por la señal del bloque TX que indica que se han terminado de envíar el paquete y entonces

se puede pedir el siguiente dato de la memoria. Cuando se han enviado todos los datos, se cierra el puerto y se enciende un led de aviso en la tarjeta de desarrollo.

3.2.2. Transmisor

El módulo transmisor está compuesto de un bloque encargado de la generación de los pulsos y de uno que lleve a cabo la interfaz con el DAC, es decir, que se encargue de suministrar al convertidor, tanto las muestras como el reloj de trabajo.

3.2.2.1. Generación de reloj

El DAC solo necesita de un reloj de entrada y de los datos a convertir, de manera que se requiere de la generación de una reloj de frecuencia variable según la aplicación, de entre 6.25 y 150 MHz, según las especificaciones del convertidor. Para este propósito se usa el *Clocking Wizard*, con el cual se pueden crear circuitos de reloj. La interfaz permite a partir de una frecuencia de entrada, generar hasta 6 de salida diferentes, ya sea a una menor o una mayor frecuencia, dejando el calculo de los multiplicadores y divisores necesarios al bloque [47]. La frecuencia de entrada es la del reloj de la tarjeta de desarrollo, es decir 100 MHz, y a partir de esta se generan además de la frecuencia para el DAC, las frecuencias para el resto de los bloques del sistema. Las señales de reloj generadas por el *Clocking Wizard*, además de crear una red de reloj interna para el FPGA, también pueden usarse para aplicaciones externas donde se requiera ya sea de señales sencillas o diferenciales, para lo cual se implementa un buffer de retraso cero (Zero Delay Buffer) [48].

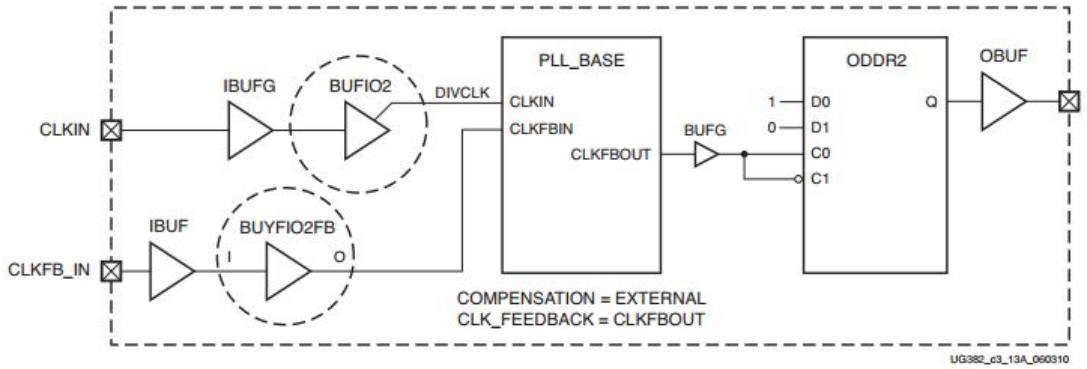


FIGURA 3.5: Buffer de retraso cero para reloj sencillo [48].

En la Figura 3.5, el bloque PLL_BASE se refiere al *Clocking Wizard* donde se genera la nueva frecuencia de reloj, mientras que la primitiva Double Data Rate Output (ODDR), está diseñada para producir señales de doble tasa de datos [49].

3.2.2.2. Secuencia de pulsos

Para las secuencias de pulsos, al igual que en el mezclador, se hizo uso del bloque *LogiCORE IP DDS*, con el cual se genera una señal senoidal que posteriormente será modulada con una secuencia de pulsos cuadrados de ancho configurable. Esta secuencia se programa con un par de contadores que se encargan de calcular el tiempo de repetición entre los pulsos y el tiempo de duración de los mismos. Con el comienzo de la generación de la secuencia, se activa una señal que nos indica cuando podemos iniciar la adquisición de la señal en el receptor, como se muestra en la Figura 3.6.

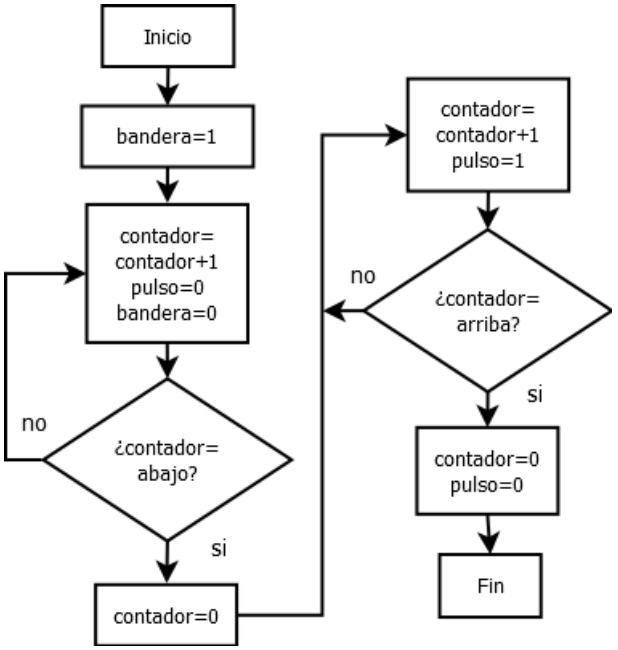


FIGURA 3.6: Diagrama de flujo, máquina de estados del módulo que genera la secuencia de pulsos.

Para las frecuencias a enviar, en el caso de RMN, por ejemplo, los pulsos necesarios para hacer resonar el núcleo de hidrógeno, típicamente necesitan una salida de 1 KHz. Por lo tanto se hace uso de frecuencias en el orden de KHz hasta MHz. Por último, el DAC no trabaja con lógica en complemento a dos, a diferencia del ADC y los distintos módulos programados en el FPGA, por lo tanto es necesario que los datos generados se conviertan a su representación no signada.

3.2.3. Comunicación mediante UART

La comunicación mediante UART envía datos paralelos sobre una línea serie usando el estándar RS-232. La tarjeta de desarrollo Atlys cuenta con un chip que convierte los voltajes definidos en RS-232 al voltaje de los pines de entrada y salida del FPGA, por lo tanto solo queda diseñar la comunicación. Se diseñan un transmisor y un receptor. El transmisor es un registro que carga los datos a transmitir en paralelo y luego los va sacando bit a bit a una determinada velocidad, comenzando por el menos significativo. El receptor recibe bits en serie y se encarga de rearmar el dato completo. Para el protocolo de comunicación se usan dos líneas de datos, una para recibir y otra para enviar, además de la alimentación. El procedimiento es el siguiente:

- Cuando la línea no lleva datos se encuentra en "1".
- La transmisión comienza con un "0"seguido de los bits de datos, el bit de paridad (que es opcional) y el bit de paro.
- No existe señal de reloj en la transmisión.
- El emisor y el receptor deben tener configurada la misma velocidad de transmisión o baud rate y el número de bits que se transmitirá.

En la Figura 3.7 se muestra el diagrama a bloques del sistema implementado, el cual consiste de: Un módulo generador de baud rate, que se encargará de generar una señal que permita decidir cuando el bit de entrada en el receptor puede ser leído, pues se quiere tomar el bit justo en la mitad de su duración; y cuando puede enviarse en el transmisor el bit de salida. El módulo receptor usa esta señal para llevar a cabo un sobremuestreo de la señal de entrada y poder adquirir los datos y reordenarlos en una palabra. El módulo transmisor se encarga de ir sacando los bits de la palabra a enviar a una frecuencia adecuada según la velocidad de transmisión. En cuanto a las memorias, se encargan de hacer una interface para almacenar los datos tanto de entrada como de salida.

Para llevar a cabo el sobremuestreo que nos permite adquirir correctamente los datos, lo más común es generar una frecuencia que sea 16 veces el baud rate. El procedimiento es el siguiente:

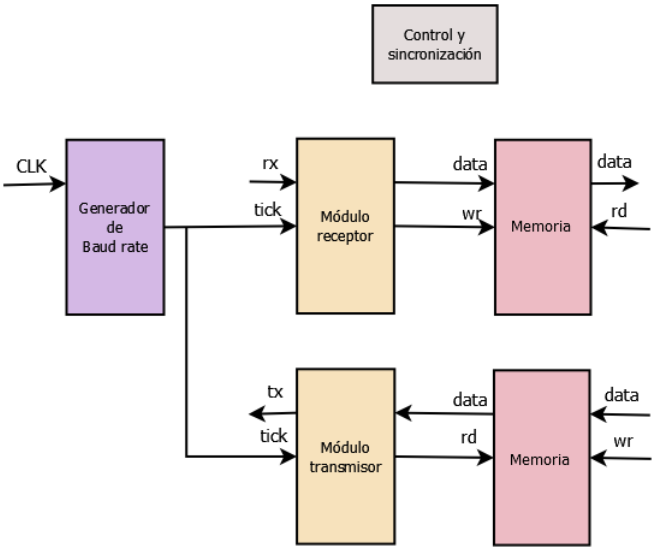


FIGURA 3.7: Diagrama de bloques comunicación UART [50].

1. Esperar que la señal de entrada se ponga en ”0”, es decir cuando recibamos el bit de inicio comenzaremos a contar.

2. Cuando se hayan contado 7 pulsos nos encontraremos en la mitad de la duración del bit de inicio. Es decir, si muestreamos la señal de entrada con una velocidad 16 veces mayor que la velocidad de transmisión, con cada nuevo bit que llegue podemos llevar a cabo una cuenta desde el 0 hasta el 15, donde cuando contemos hasta el 7, estaremos posicionados aproximadamente en la mitad del tiempo de transmisión del bit.

3. Se reinicia el contador, pues no nos interesa el inicio de la transmisión del bit, si no su valor medio donde es estable.

4. Esperamos a que el contador llegue a 15, cuando nos encontremos en la mitad del siguiente bit transmitido. Se lleva a cabo la adquisición del bit y se reinicia el contador.

5. Se repite el paso 4 por cada bit de datos.

6. Se repite el paso 4 una vez más para el bit de paridad, si es que existe, y otra vez para el bit de paro.

Entonces, el modulo generador de baud rate debe generar un pulso a una frecuencia 16 veces mayor que la velocidad de transmisión, para lo cual se ha elegido un baud rate de 115200 bits por segundo, es decir necesitamos un pulso cada 16 X 115200 = 1843200.

El reloj de la tarjeta Atlys es de 100 MHz, por lo tanto hay que generar el pulso cada 55 ciclos de reloj:

$$(100\ X\ 10^6)/(16\ X\ 115200) \approx 55 \tag{3.11}$$

Cabe destacar que esta señal no es propiamente un reloj, si no más bien una señal de habilitación que se llama comúnmente *tick*. Este nombre se debe a que solo valdrá "1" durante un ciclo de reloj [50]. Para el receptor se programó una máquina de estados que lleva a cabo el muestreo de los datos en serie y luego los arma en un byte. Para este sistema se mandan paquetes de 8 bits de datos, sin bit de paridad y con 1 bit de paro. La memoria del receptor se encarga de guardar el dato armado cuando recibe la señal de habilitación del receptor, la cual indica que ya ha adquirido todos los bits. En la Figura 3.8 se muestra el diagrama de flujo de la máquina de estados, en el que el proceso de sobremuestreo se lleva a cabo para el bit de inicio, los bits de datos y el bit de paridad. El transmisor también consta de una máquina de estados, que en este caso desarma la palabra a enviar y la transmite bit a bit, seguida de un bit de paro. Este módulo usa una frecuencia 16 veces menor que la calculada en la Ecuación 3.11, es decir, la velocidad de transmisión; por lo tanto se usa un contador de 0 a 15 para poder aprovechar la señal generada por el módulo del baud rate. La memoria del transmisor guarda los datos a ser trasmitidos y con la señal que genera el transmisor indicando que ha terminado de enviar una palabra, habilita la lectura para tomar otra y enviarla. En la Figura 3.9 se observa el diagrama de flujo de la máquina de estados del módulo transmisor.

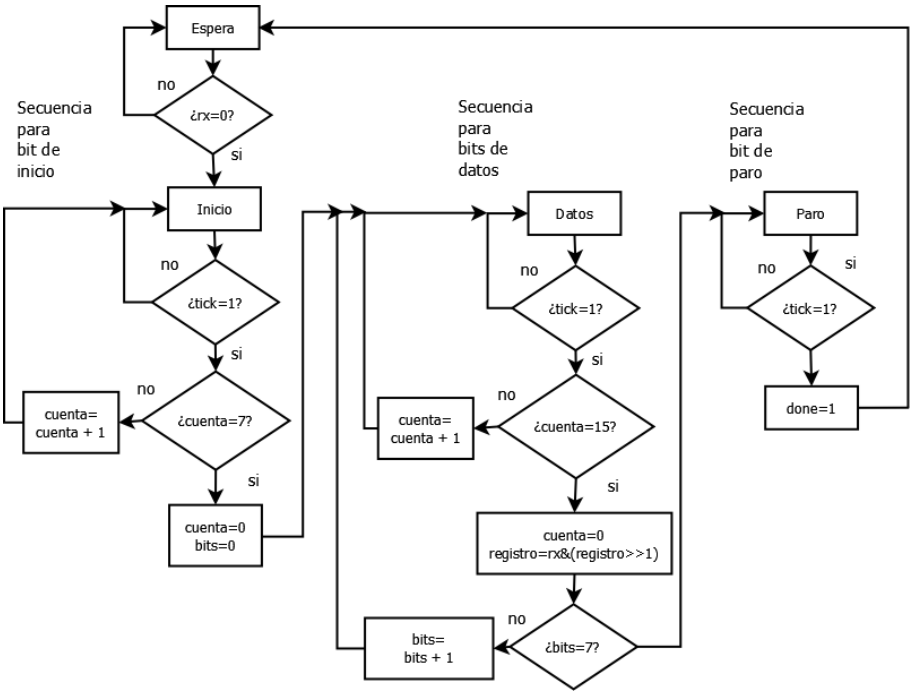


FIGURA 3.8: Diagrama de flujo, máquina de estados del módulo UART receptor.

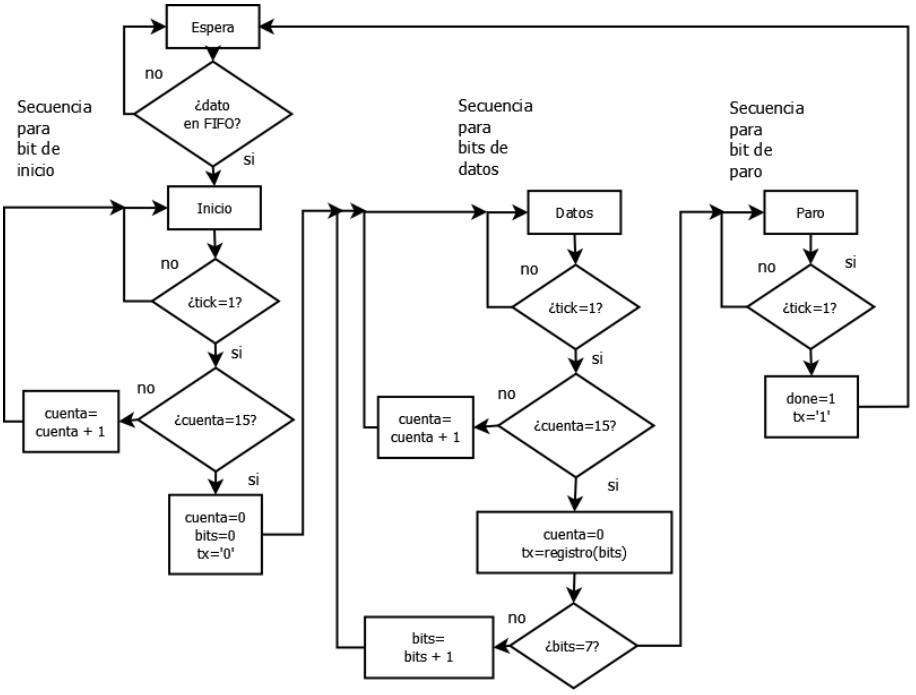


FIGURA 3.9: Diagrama de flujo, máquina de estados del módulo UART transmisor.

Capítulo 4

Implementación y pruebas

4.1. Tarjetas utilizadas

Atlys Spartan-6 FPGA Development Board

La tarjeta de desarrollo, Figura 4.1, cuenta con: Un FPGA Spartan-6 LX45, con 6822 *slices*, 2.1 MBits de memoria en bloques RAM, 58 *DSP Slices* y 6 bloques PLL entre otros. Puertos USB para programación y transferencia de datos. Un oscilador de 100 MHz. Así como LEDS, interruptores y botones para interactuar con la tarjeta.

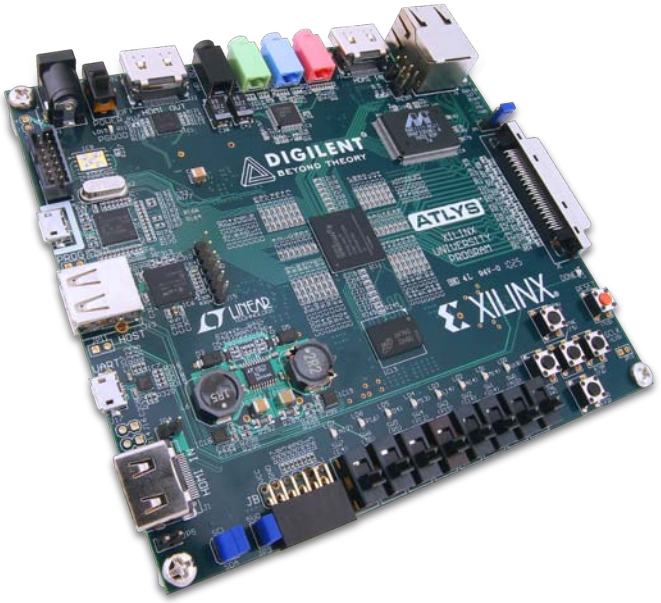


FIGURA 4.1: Atlys Spartan-6 FPGA Development Board.

Para la conexión con otros módulos, la tarjeta de desarrollo Atlys tiene dos conectores. Un conector de muy alta densidad o very high density cable (VHDC) de 68 pines de entrada/salida para transmisión de datos en paralelo o serie de alta velocidad y un Pmod de 8 pines de entrada/salida de baja velocidad. Para el conector VHDC se cuenta con el *Diligent Vmod Breadboard*, el cual conecta 28 de los 40 pines de entrada/salida para datos, 2 de alimentación y 2 de GND del conector VHDC a una placa de pruebas, Figura 4.2, con este módulo se llevan a cabo las conexiones a los pines de datos de los convertidores ADC y DAC.

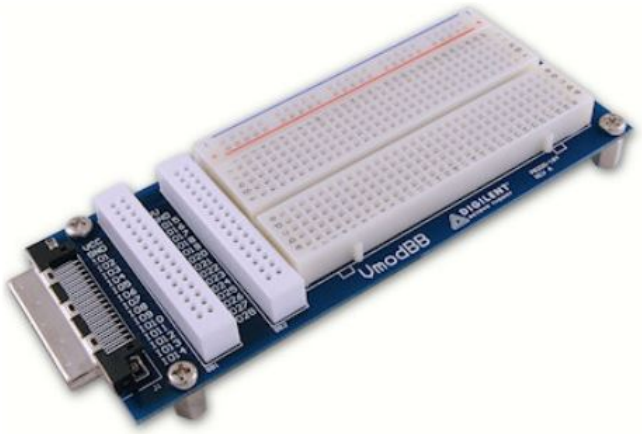


FIGURA 4.2: Diligent Vmod Breadboard.

En la Tabla 4.1 se muestran las conexiones entre los pines del FPGA y los del conector Pmod de la tarjeta Atlys.

PMOD JB	FPGA	PMOD JB	FPGA
1	T3	7	V9
2	R3	8	T9
3	P6	9	V4
4	N5	10	T4
5	GND	11	GND
6	VCC	12	VCC

TABLA 4.1: Pines del FPGA correspondientes al conector Pmod

Y en la Tabla 4.2 las conexiones entre los pines del FPGA, el conector VHDC y el *Diligent Vmod Breadboard*. En base a estas tablas, se puede crear el archivo .ucf (*user constraints file*), necesario para la implementación del sistema, es decir el archivo donde se especifican las conexiones externas del FPGA.

Breadboard	VHDC	FPGA	Breadboard	VHDC	FPGA
VCC	VCC	VCC	VU	VU	VU
GND	GND	GND	GND	GND	GND
IO1	1	U16	IO15	35	V16
IO2	3	U15	IO16	37	V15
IO3	4	U13	IO17	38	V13
IO4	6	M11	IO18	40	N11
IO5	7	R11	IO19	41	T11
IO6	9	T12	IO20	43	V12
IO7	10	N10	IO21	44	P11
IO8	12	M10	IO22	46	N9
IO9	13	U11	IO23	47	V11
IO10	15	R10	IO24	49	T10
IO11	20	U10	IO25	54	V10
IO12	22	R8	IO26	56	T8
IO13	23	M8	IO27	57	N8
IO14	25	U8	IO28	59	V8

TABLA 4.2: Pines del FPGA correspondientes al conector al VHDC

AD6644, A/D Converter

El AD6644 es un convertidor analógico a digital de alta velocidad y rendimiento a 65 MSPS, con resolución de 14 bits en complemento a dos. La tarjeta de evaluación, Figura 4.3, contiene toda la circuitería necesaria para el convertidor, de manera que, las únicas conexiones externas que se requieren son las alimentaciones y la entrada analógica. Se necesita de un voltaje de polarización de 5 V para la parte analógica del convertidor y de 3.3 V para la digital. La entrada analógica se conecta a través de un conector BNC (AIN) y cuenta con un acoplamiento mediante trasformador para adecuar la señales. La señal de reloj proviene de un oscilador montado en la tarjeta, sin embargo, si se requiere se puede remover y usar ya sea el conector SMA (OPT_CLK) o el BNC (EN) para suministrar una señal externa. Los pines de datos de salida pasan a través de unos *latchces* cuya frecuencia de reloj está determinada por la posición del jumper (BUFLAT), que permite elegir ya sea la señal *Data Ready Output* generada por el convertidor, o la señal del reloj [39]. El FPGA se conecta al ADC mediante los 14 bits de datos y las señales DRY (*Data Ready Output*), y OVR (*Overrange Bit*). DRY indica que la salida de datos esta lista y OVR que la señal analógica excede el rango $\pm FS$, es decir, la escala completa del dispositivo. Ambos pines se conectan a través del Pmod en los pines V9 y T9 del FPGA, respectivamente.

El convertidor cuenta con entradas analógicas complementarias AIN y $\overline{AIN}$, cada una

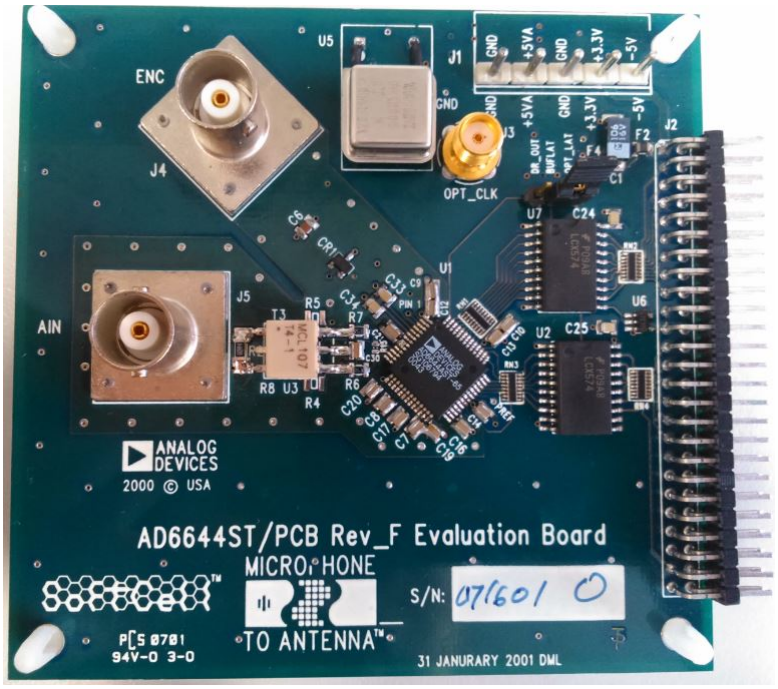


FIGURA 4.3: Convertidor analógico a digital AD6644.

centrada en 2.4 V, que es el voltaje de referencia del convertidor, y oscila ±0.55 V sobre su referencia. Ya que AIN y $\overline{AIN}$ están 180° fuera de fase, la señal analógica diferencial de entrada tiene una amplitud de 2.2 Vpp, de -1.1 a 1.1 V. Como la salida se da en complemento a dos, las representaciones para el número más grande y más pequeño serán 01 1111 1111 1111 y 10 0000 0000 0000, respectivamente. La resolución de 14 bits permite 16384 valores posibles (2^{14}); y como la salida oscila entre $\pm$ 1.1 V, los valores se escalan para que 1.1 V se represente con el valor positivo más grande posible es decir 1FFF(hex), 0 V con 0000(hex) y $-$ 1.1 V con el valor negativo mas grande posible 2000(hex). Por lo tanto, para obtener el voltaje de salida del convertidor a partir del valor digital, se usa la Ecuación 4.1.

$$Voltaje \; = \; salida_{ADC}/8192 \; X \; 1.1 \; V \tag{4.1}$$

Un aspecto a tener en cuenta es que debido a que la entrada del convertidor esta acoplada a 50 Ω, la señal se atenúa en amplitud, aproximadamente la mitad. Es decir si generamos una señal de 2 Vpp, en la entrada del convertidor tendremos 1 Vpp.

AD755, High Speed TxDAC D/A Converter

El AD9755 es un convertidor digital a analógico con puerto dual multiplexado, de alta velocidad y 14 bits de resolución, que ofrece un alto desempeño, soportando tasas de hasta 300 MSPS. La tarjeta de evaluación, Figura 4.4, permite evaluar con facilidad los diferentes modos de operación del convertidor.

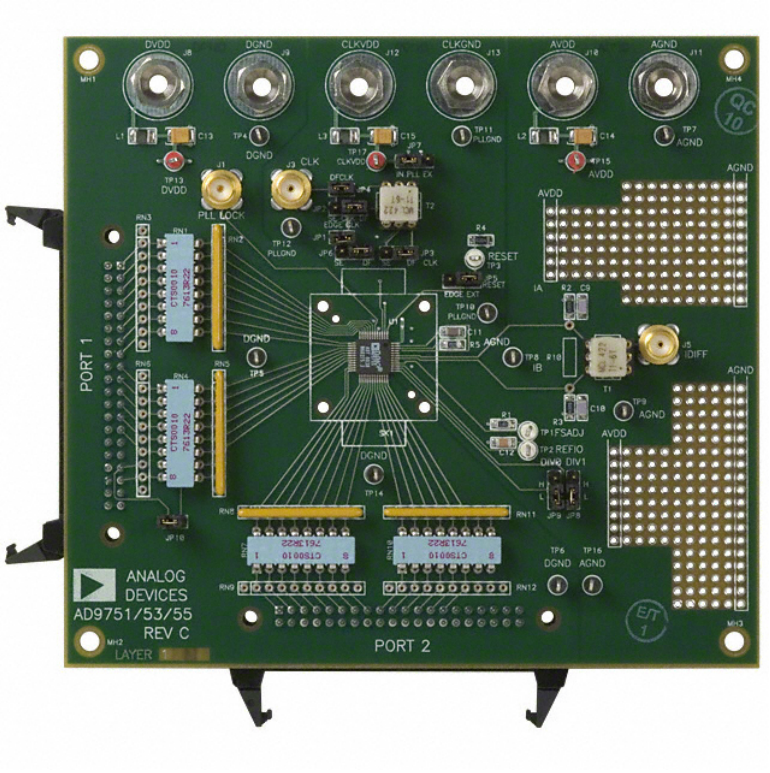


FIGURA 4.4: Convertidor digital a analógico AD9755.

La tarjeta cuenta con 3 alimentaciones de 3.3 V: el voltaje analógico, el digital y el del circuito de reloj. La salida del convertidor puede ser sencilla o diferencial, acoplándola directamente o mediante el uso de un transformador. Los datos digitales llegan a través de los puertos JP1 y JP2 con conectores de 40 pines. La configuración del reloj también puede ser sencilla o diferencial, la cual se escoge mediante la posición de los jumpers JP2, JP3, JP4 y JP6; este reloj puede entrar ya sea al conector CLK o mediante el pin 33 del puerto de datos 1 (P1); para elegir esta última configuración el jumper JP1 debe estar colocado. El convertidor cuenta con un PLL multiplicador interno para doblar la frecuencia de reloj, el cual se habilita con el jumper JP7. Cuando se habilita, un reloj de la mitad de la tasa de salida deseada debe aplicarse y el PLL interno se encargará de la multiplicación de la frecuencia por 2. La posición de los jumpers DIV0 y DIV1 elegirá la razón del divisor, de acuerdo con la Tabla 4.3. Si el PLL está desactivado, la

frecuencia del reloj debe ser de la tasa de salida deseada, internamente este reloj se dividirá por 2. El funcionamiento de los pines DIV0 y DIV1 se muestra en la Tabla 4.4 [40].

Frecuencia de CLK	DIV1	DIV0	Controlador de rango
50 - 150 MHz	0	0	÷ 1
25 - 100 MHz	0	1	÷ 2
12.5 - 50 MHz	1	0	÷ 4
6.25 - 25 MHz	1	1	÷ 8

Tabla 4.3: Configuración de DIV1 y DIV0 con PLL activado

Modo de entrada	DIV1	DIV0
Intercalar (2x)	0	0
Puerto 1	0	1
Puerto 2	1	0
No permitido	1	1

Tabla 4.4: Configuración de DIV1 y DIV0 con PLL desactivado

En la Tabla 4.5 se muestra un resumen de las funciones de los distintos jumpers de la tarjeta de evaluación. Se indica el nombre con el que viene señalado cada uno en la tarjeta, las posiciones que puede tomar (A y B) y el propósito de dicho jumper. En el caso de los jumpers JP6, JP3, JP1, JP2 y JP4; SE y DF se refieren a *Single ended* y *Differential*, que son las dos opciones con las que se puede aplicar el reloj. Ya sea tanto para la señal de reloj como para la salida analógica, si se elige la configuración SE, se deberá remover los trasformadores T2 y T1, respectivamente.

Jumper	A	B	Función
JP10	1O15	1O17	Puente para reloj proveniente de P1
JP5	OUT16	TP3	Reset, puede provenir de un pin externo o aterrizarse
JP8	1	0	DIV1
JP9	1	0	DIV0
JP6	GND	CLK-	CLK SE o DF
JP3	SE	DF	CLK SE o DF
JP1	CLK+	OUT15	CLK SE o DF
JP2	SE	DF	CLK SE o DF
JP4	DF		CLK SE o DF
JP7	VDD	GND	PLL activado o desactivado

Tabla 4.5: Configuraciones de los jumpers de la tarjeta de evaluación AD9755

El AD9755 provee salidas de corriente, complementarias, I_{OUTA} y I_{OUTB}. I_{OUTA} da una salida de corriente cercana a la escala completa de corriente, I_{OUTFS}, cuando todos los bits están en alto (código del DAC=16383) mientras I_{OUTB}, la salida complementaria,

no proporciona corriente. Las corrientes de salida son funciones del código de entrada y de I_{OUTFS}, expresadas en las ecuaciones 4.2 y 4.3.

$$I_{OUTA} = \frac{DAC\ CODE}{16383} X I_{OUTFS} \tag{4.2}$$

$$I_{OUTB} = \frac{16383 - DAC\ CODE}{16384} X I_{OUTFS} \tag{4.3}$$

Donde $DAC\ CODE$ va desde 0 hasta 16383. I_{OUTFS} es una función de la corriente de referencia I_{REF}, dada por el voltaje de referencia V_{REFIO} (1.2 V), y la resistencia externa R_{SET} y se puede expresar como:

$$I_{OUTFS} = 32 X I_{REF} \tag{4.4}$$

Donde:

$$I_{REF} = V_{REFIO}/R_{SET} \tag{4.5}$$

Por lo tanto sustituyendo la Ecuación 4.5 en 4.4.

$$I_{OUTFS} \ = \ 32\ V_{REFIO}/R_{SET} \tag{4.6}$$

Como $V_{REF} = 1.2\ V$ y $R_{SET} = 1.91\ K$ entonces $I_{OUTFS} = 20\ mA$.

Si se requiere de un acoplamiento dc, I_{OUTA} y I_{OUTB} se conectan a resistencias de carga, R_{LOAD}, que están aterrizadas a ACOM. Entonces el voltaje está dado por:

$$V_{OUTA} = I_{OUTA} X R_{LOAD} \tag{4.7}$$

$$V_{OUTB} = I_{OUTB} X R_{LOAD} \tag{4.8}$$

Para mantener las especificaciones de desempeño y linealidad, el valor de escala completa de V_{OUTA} y V_{OUTB} no debe exceder el rango de salida especificado:

$$V_{DIFF} = (I_{OUTA-}I_{OUTB})XR_{LOAD} \tag{4.9}$$

4.2. Adquisición de los datos en CPU

Para almacenar y analizar la información recibida por el FPGA, se usa una interfaz en Matlab. Esta, se encarga de configurar y abrir un puerto de comunicación serial a 115200 baudios. Cuando se quiera recibir la información almacenada en la memoria del FPGA, se manda a escribir en el puerto la letra A para indicar el inicio del envío. Ya que los datos recibidos vienen en paquetes de 8 bits, es necesario armar 4 paquetes recibidos en un solo dato, tomando en cuenta que se reciben los bits menos significativos primero. El valor decimal que obtengamos viene en complemento a dos, así que finalmente tiene que convertirse a su equivalente a voltaje con la ecuación 4.1. Los datos se almacenan en un archivo .dat y el FPGA está programado para enviar un dato nuevo solo cuando el dato enviado previamente se ha recibido correctamente en Matlab, esto para garantizar la correcta recepción de todos los datos.

Como las mediciones en el dominio del tiempo resultan difíciles, se usa el archivo .dat, la frecuencia de muestreo y el factor de decimación empleado, para calcular el espectro de la señal. Así podemos llevar a cabo mediciones tanto de las componentes de la señal como del ruido, que en el dominio de la frecuencia podemos ver si tiene alguna relación con alguna frecuencia en específico. También se calcula la magnitud del espectro, que es la raíz cuadrada de la suma de la parte real al cuadrado con la parte imaginaria al cuadrado; pues es una cantidad real que representa el total de la amplitud de cada frecuencia, independientemente de la fase. En el Apéndice A se muestra el código utilizado.

4.3. Pruebas del sistema

Se llevaron a cabo pruebas de cada uno de los módulos del radio digital implementado para evaluar su desempeño. El generador de funciones 8648A de Agilent, nos permite generar frecuencias desde 100 KHz hasta 1000 MHz con una resolución ajustable de

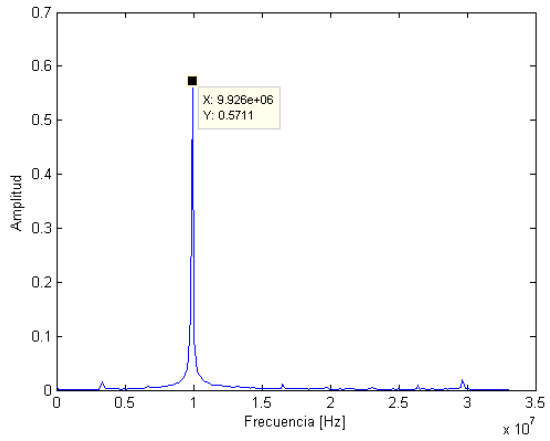
0.0001 Hz. Se puede llevar a cabo modulación en frecuencia o frequency modulation (FM) y en amplitud o amplitude modulation (AM), ya sea con una fuente interna, o con una fuente externa. En el generador se cuentan con 400 Hz y 1 KHz para modular la portadora. Para usar una fuente externa para la modulación, se requiere de una señal de máximo 1 Vpp, con los rangos de frecuencia permitidos dados por la Tabla 4.6 [51].

Modulación	Acoplamiento	Rango
FM	EXT AC	1 Hz a 150 KHz
FM	EXT CC	DC a 150 KHz
Φ	EXT AC	20 Hz a 10 KHz
AM	EXT AC	1 Hz a 25 KHz
AM	EXT CC	DC a 25 KHz

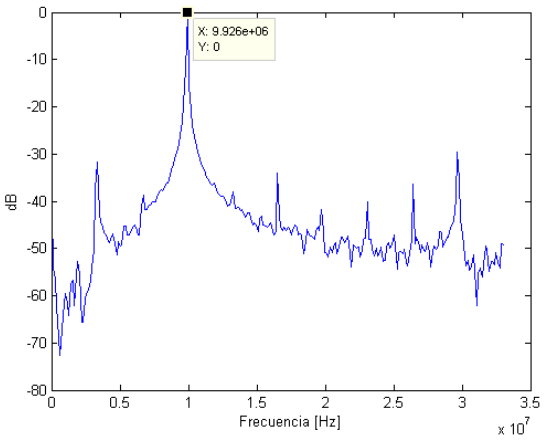
Tabla 4.6: Rangos de frecuencia aceptables para modulación con fuente externa.

4.3.1. Receptor

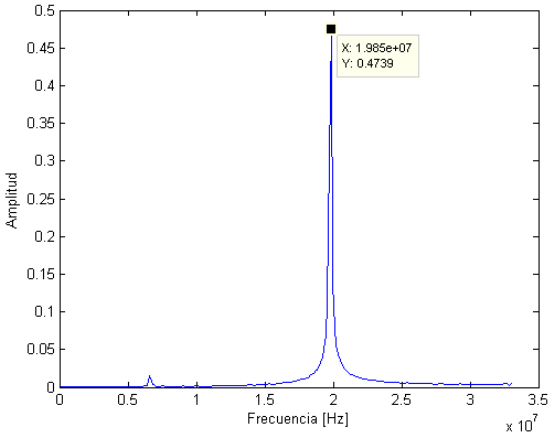
Las primeras pruebas llevadas a cabo para el receptor, fueron las de adquirir la señal del ADC sin procesamiento, usando distintas frecuencias, desde 20 KHz hasta 33 MHz; es decir, siguiendo el criterio de Nyquist. En la Figura 4.5 se muestran algunos resultados, donde para todos los casos, la frecuencia de muestreo F_s es de 66 MHz.



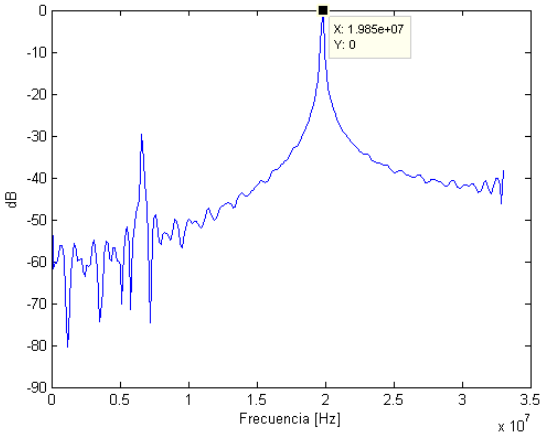
(A)



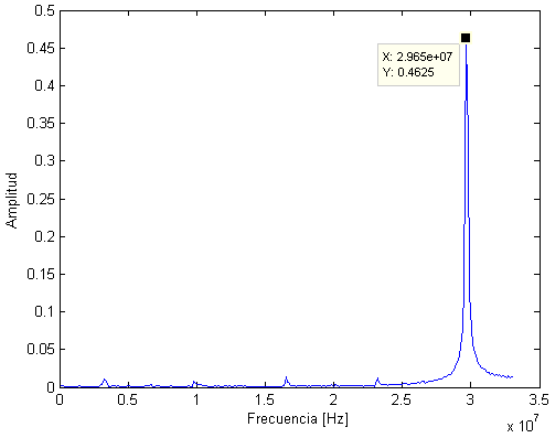
(B)



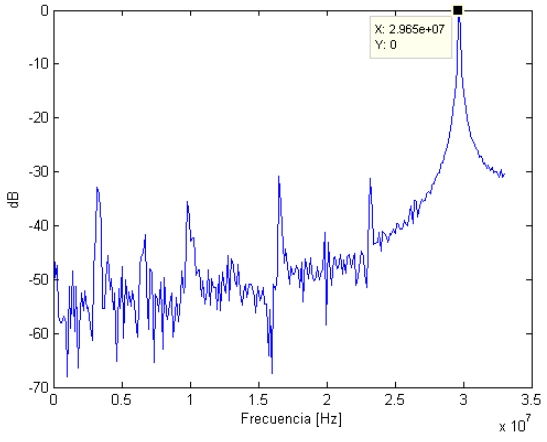
(C)



(D)



(E)

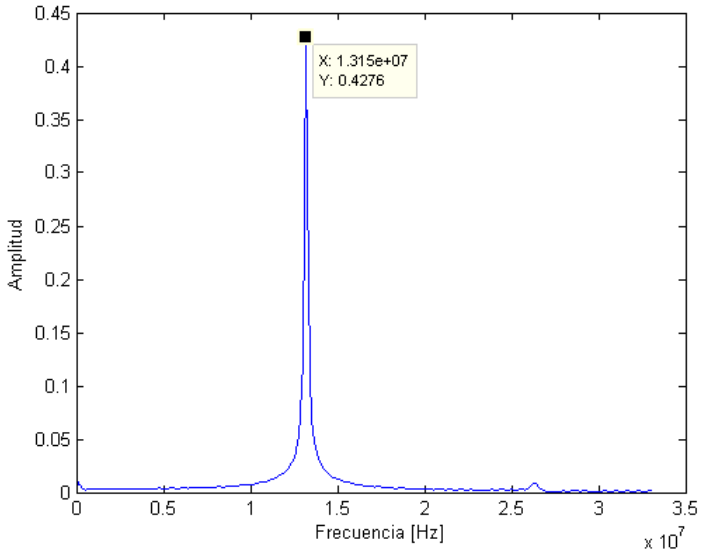


(F)

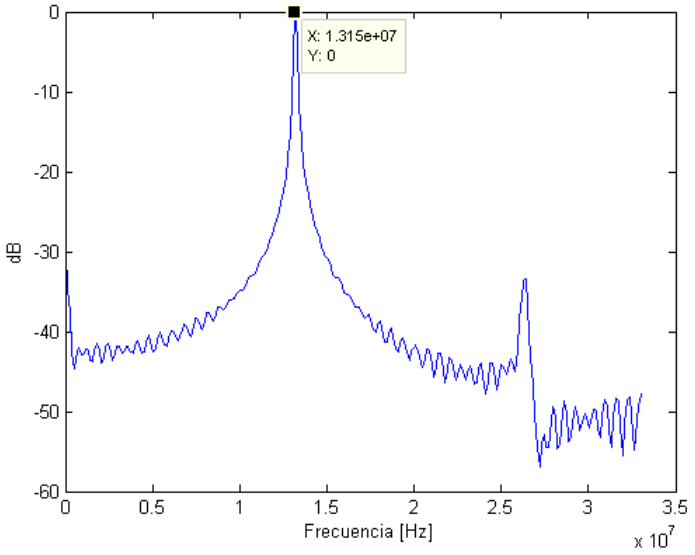
FIGURA 4.5: FFT (lado izquierdo) y densidad espectral de potencia (lado derecho) de una muestra de 500 puntos, para señales con frecuencias de 10 MHZ (A y B); 20 MHz (C y D) y 30 MHz (E y F).

Submuestreo y sobremuestreo

Como se discutió en el Capítulo 2, aún cuando la frecuencia del convertidor es de 66 MHz, podemos adquirir señales de mayor frecuencia. En la Figura 4.6 se muestran los resultados obtenidos al adquirir una señal de 80 MHz, la cual según la teoría, debería aparecer en el espectro alrededor de 13.34 MHz.



(A)



(B)

FIGURA 4.6: FFT (A) y densidad espectral de potencia (B) de una muestra de 500 puntos para una señal de 80 MHZ.

Se puede deducir la frecuencia a la cual se trasladará el espectro, f_{IF}, partiendo de la frecuencia central original f_c, y la frecuencia de muestreo f_s con el conjunto de ecuaciones siguientes, tomadas de [52].

$$Si\ fix\left(\frac{f_c}{f_s/2}\right)\ es \begin{cases} par,\ f_{IF} = rem(f_c, f_s) \\ impar,\ f_{IF} = f_s - rem(f_c, f_s) \end{cases} \tag{4.10}$$

Donde $fix(a)$ redondea a al entero más cercano hacia cero y $rem(a, b)$ es el resto de la división de a y b. Estas ecuaciones las podemos escribir como:

$$f_{IF} = \begin{cases} rem(f_c, f_s) & si\ rem(f_c, f_s) \leq f_s/2 \\ fs - rem(f_c, f_s) & si\ rem(f_c, f_s) > f_s/2 \end{cases} \tag{4.11}$$

Si el resto de la división f_c/f_s es menor o igual a $f_s/2$, hemos trasladado la frecuencia central de la señal en el intervalo $[0, f_s/2]$, de esta forma, el resto de ambas indicará directamente nuestra nueva frecuencia. Si el resto de la división es mayor que $f_s/2$, significa que hemos trasladado la señal en el intervalo $[f_s/2, f_s]$, en este caso, para encontrar la frecuencia a donde se ha trasladado f_c, deberemos de buscar la frecuencia invertida en el intervalo $[0, f_s/2]$, siendo la indicada en la Ecuación 4.11. Una vez calculada f_{IF}, debemos comprobar si la frecuencia de muestreo es válida, es decir si permite trasladar todo el ancho de banda de información B dentro del ancho de banda de muestreo, es decir, se debe cumplir con la Ecuación 4.12, de otra forma, una porción de la información puede solaparse a si misma creando una interferencia destructiva.

$$B/2 < f_{IF} < f_s/2 - B/2 \tag{4.12}$$

Por otro lado para ejemplificar las ventajas que tiene el sobremuestreo en la calidad de la señal resultante, se llevó a cabo la adquisición de una señal de 200 KHz con una frecuencia de muestreo 66 MHz, a la cual después se le aplicó una decimación, con un factor de 100, Figura 4.7.

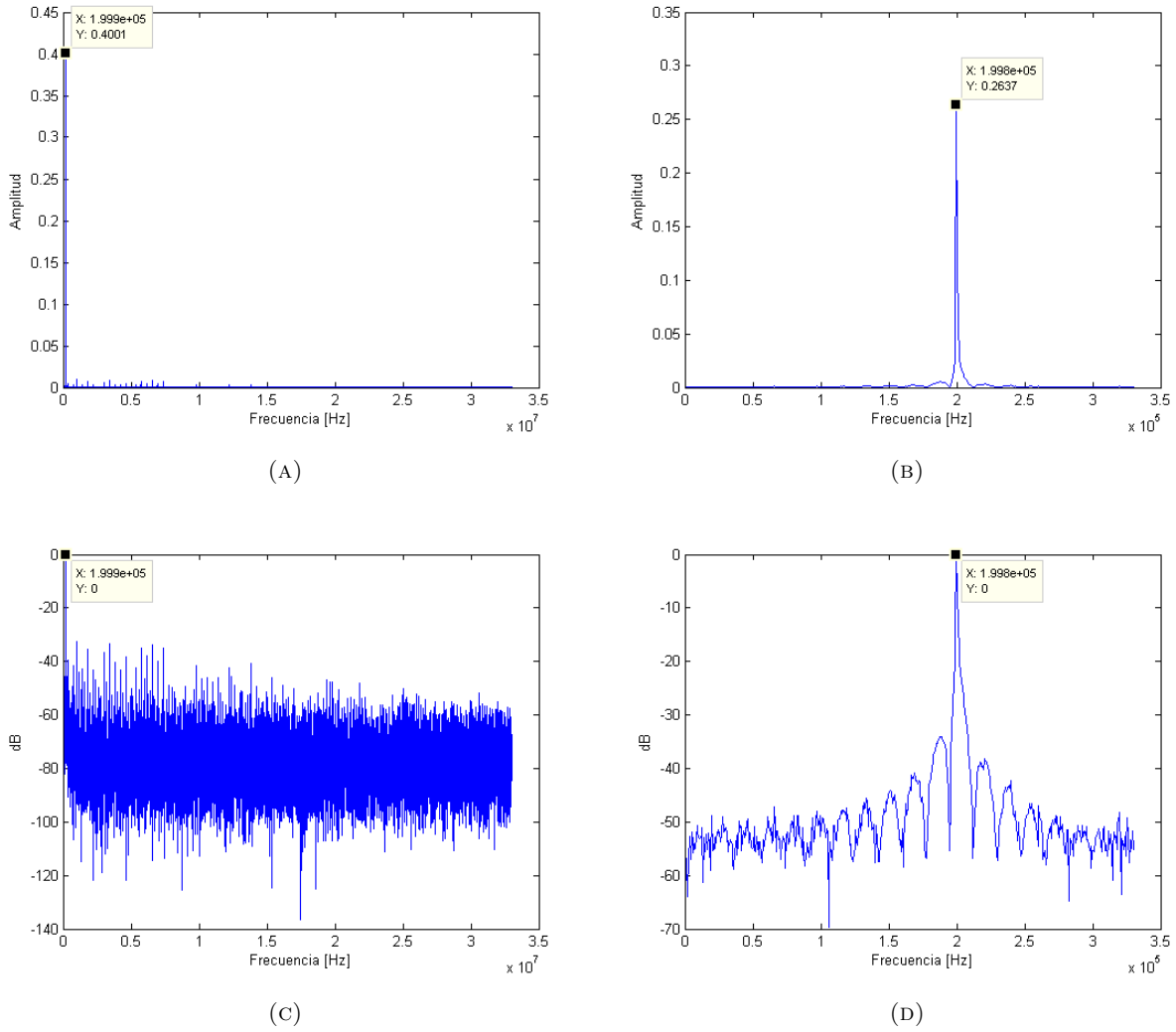
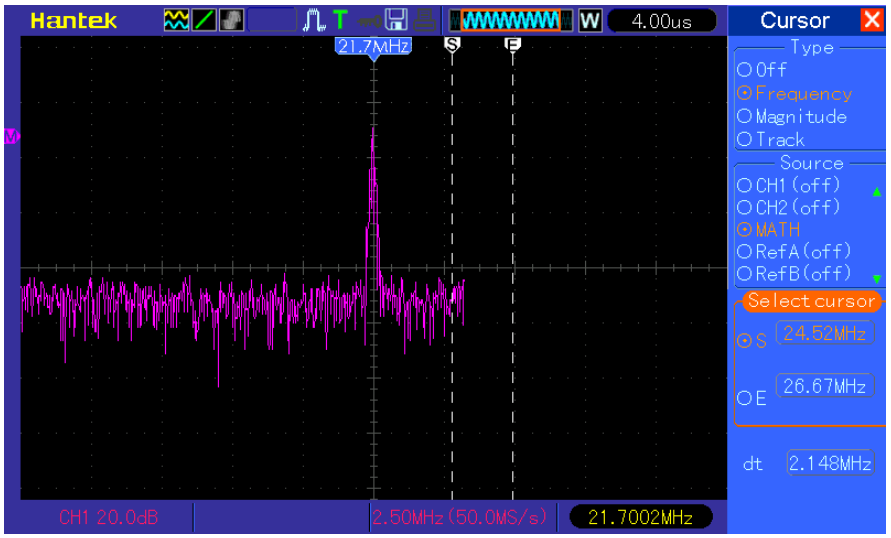


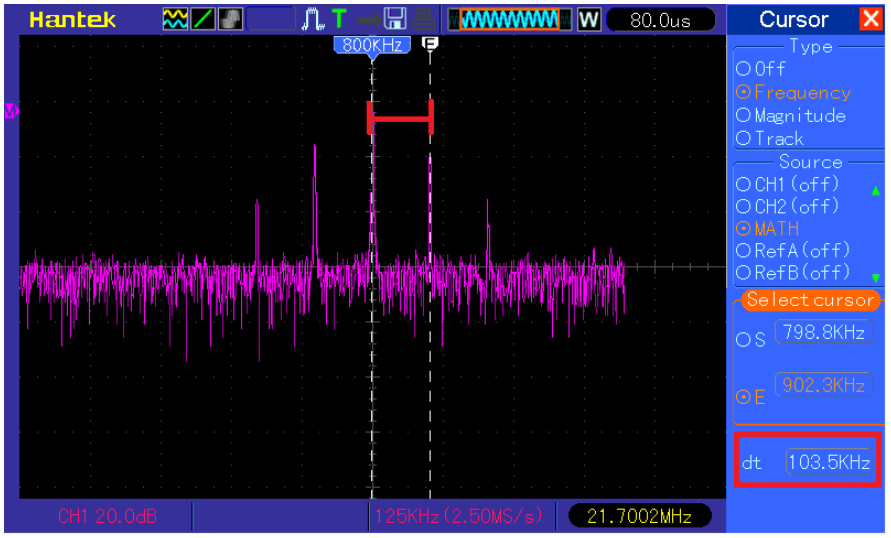
FIGURA 4.7: FFT de la señal original (A) y después de la decimación (B) y densidad espectral de potencia de la señal original (C) y después de la decimación (D), de una muestra de 99000 puntos para una señal de 200 KHz.

Señales moduladas

Se modularon señales portadoras de entre 20 y 100 MHz, usando tanto la modulación interna del generador de funciones 8648A, como también con una señal externa proveniente de otro generador. En la Figura 4.8 se muestra el espectro de una señal de 21.7 MHz, modulada con 100 KHz, obtenido con un osciloscopio digital, y en la Figura 4.9 los resultados obtenidos con el sistema receptor. La separación de las componentes moduladas de la portadora en los resultados obtenidos es de aproximadamente 104.8 KHz, Figura 4.9a, en comparación con los 103.5 KHz, calculados con los cursores del osciloscopio, Figura 4.8b.

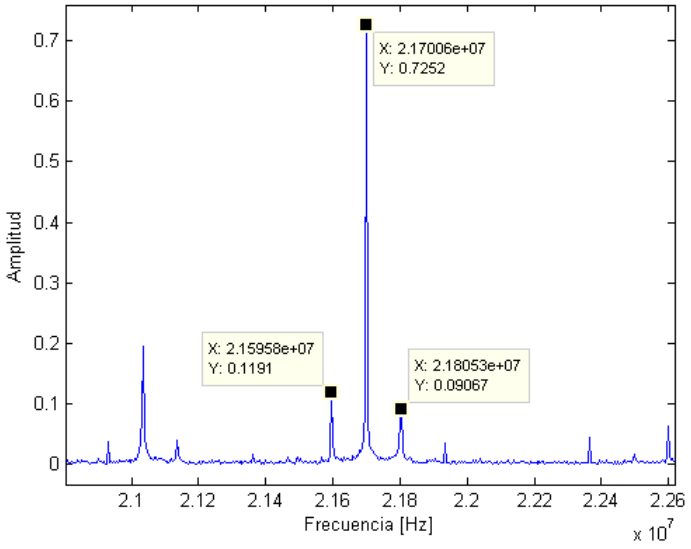


(A)

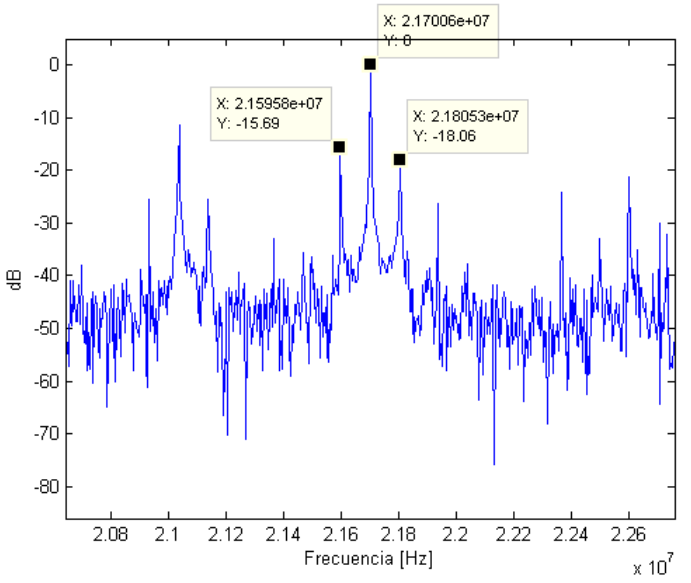


(B)

FIGURA 4.8: FFT de una señal de 21.7 MHZ modulada con 100 KHz, (A) espectro y (B) acercamiento.



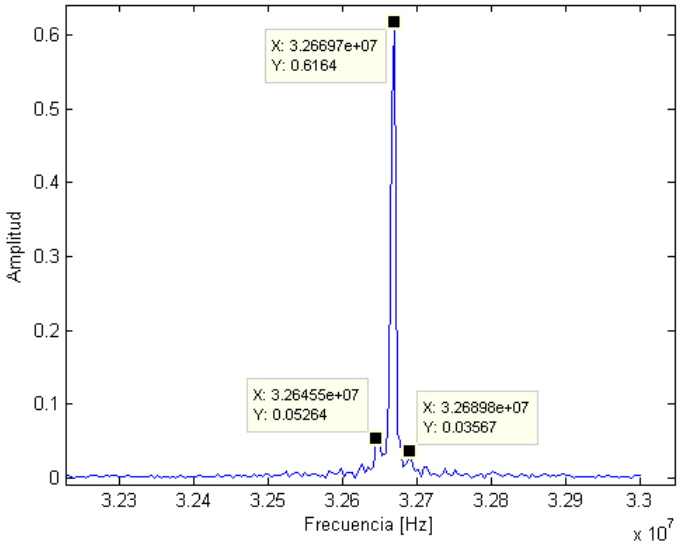
(A)



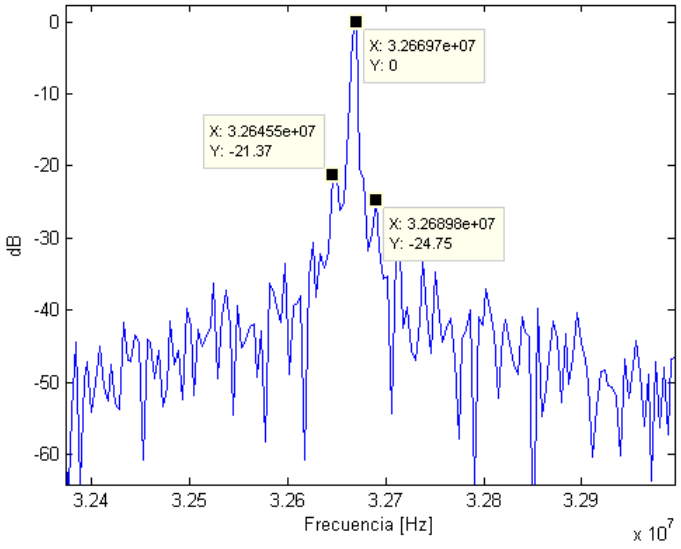
(B)

FIGURA 4.9: FFT (A) y densidad espectral de potencia (B) de una muestra de 16000 puntos para una señal de 21.7 MHZ, modulada con 100 KHz.

En la Figura 4.10 se muestra un ejemplo de submuestreo con una señal modulada. En este caso se muestrea una señal de 100 MHz con 66 MHz, estando la señal modulada con 20 KHz. En los resultados vemos que la componente portadora se encuentra en 32.37 MHz, que corresponde con la teoría, pues el espectro de la señal deseada debería verse en 32 MHz aproximadamente.



(A)



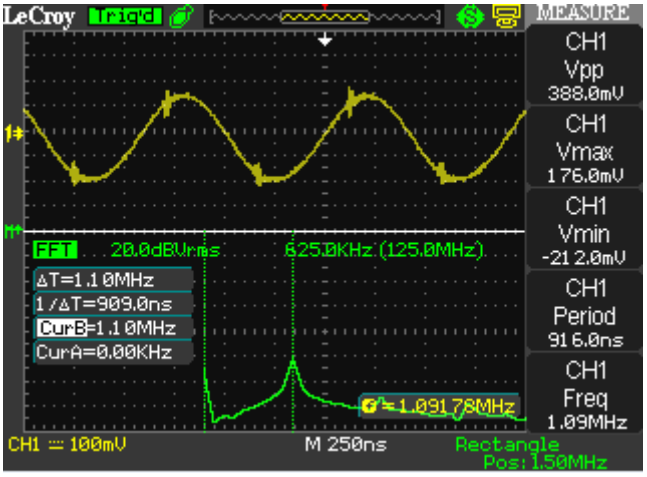
(B)

FIGURA 4.10: FFT (A) y densidad espectral (B) de una muestra de 16000 puntos para una señal de 100 MHZ modulada con 20 KHz.

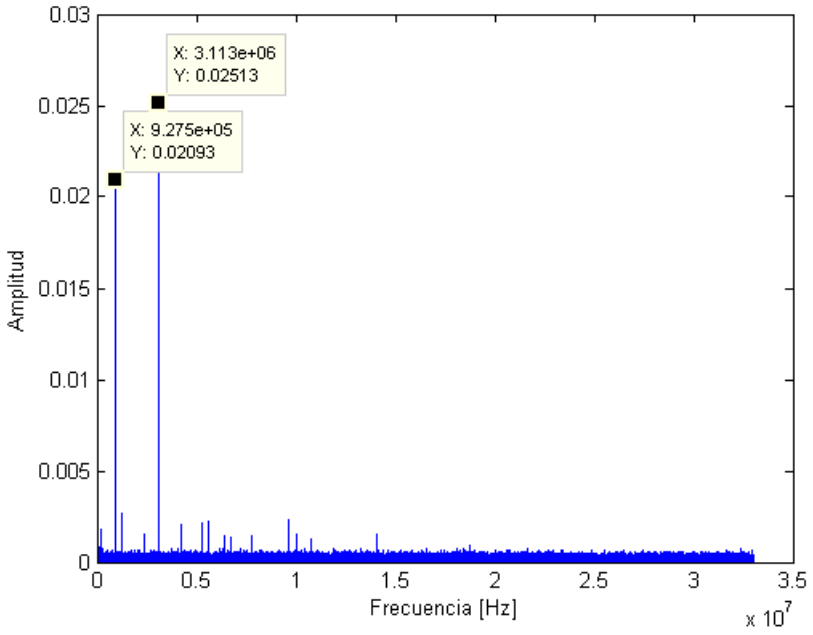
Mezclado

El funcionamiento del módulo mezclador se comprobó llevando a cabo la multiplicación de una señal adquirida desde el ADC por una generada internamente. En la Figura 4.11 se muestran los resultados, donde se multiplicó una señal adquirida de 1.091 MHz, Figura 4.11a, por una senoidal generada de 2.019 MHz. Las componentes resultantes deben

estar en 0.928 MHz y 3.11 MHz, lo que corresponde con los resultados en la Figura 4.11b.



(A)

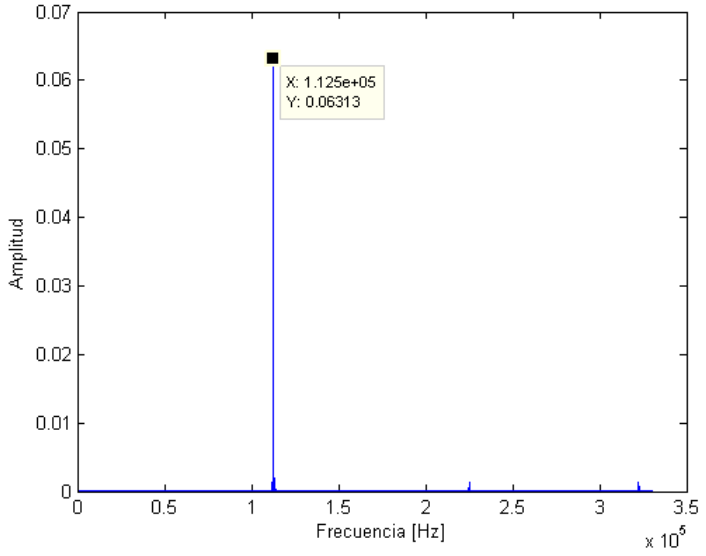


(B)

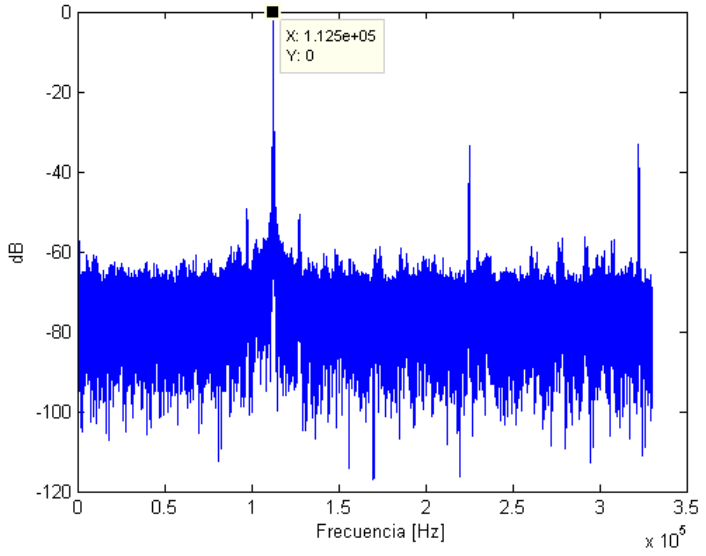
FIGURA 4.11: Señal adquirida (A) y FFT del resultado de multiplicarla por la señal generada (B).

Decimación

El bloque de decimación también se comprobó solo, para asegurar su funcionamiento. En la Figura 4.12 se muestra una decimación por un factor de 100 para una señal de 112 KHz.



(A)



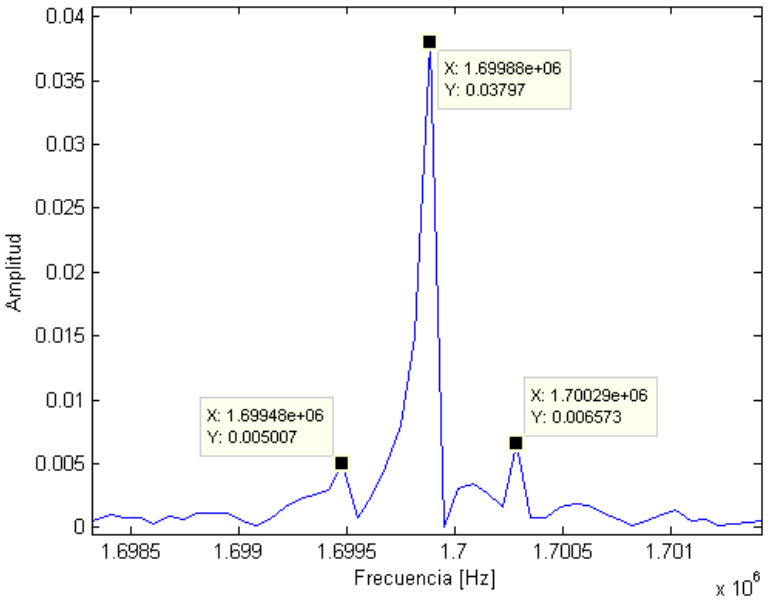
(B)

FIGURA 4.12: FFT (A) y densidad espectral de potencia (B) de una muestra de 99000 puntos para una señal de 112 KHZ con un factor de decimación de 100.

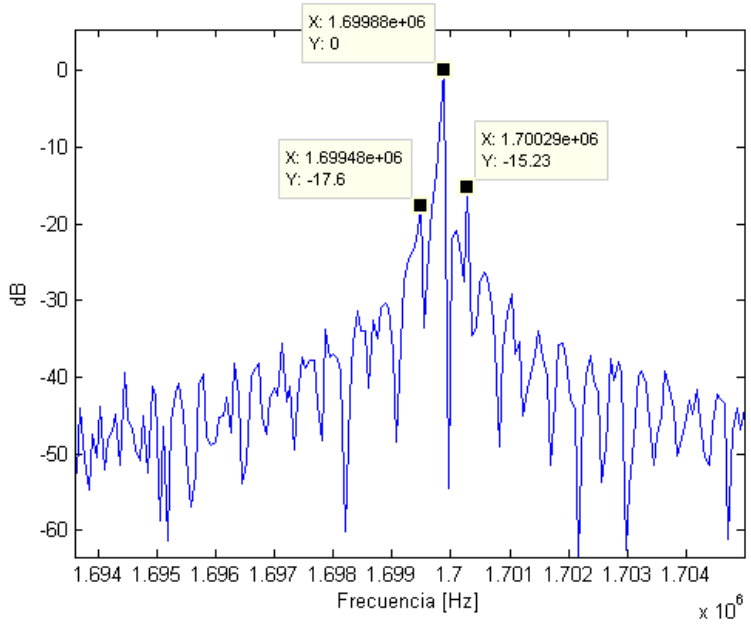
Recepción completa

En la Figura 4.13 se muestran los resultados obtenidos a partir de la integración de todos los bloques que componen el receptor. Se generó una señal de 25.3 MHz y se moduló con 400 Hz. El mezclador se programó con una frecuencia de 27 MHz, por lo que se observa el espectro deseado en 1.7 MHz aproximadamente. Por otro lado para poder ver la modulación, se llevó a cabo una decimación por un factor de 15, con lo cual podemos

guardar en la memoria de recepción 15 veces más muestras que en el caso de que no se realizara la decimación, para ver la modulación necesitamos adquirir un poco mas de 2.5 ms (1/400 Hz), lo que no sería posible si guardáramos todas las muestras generadas.



(A)

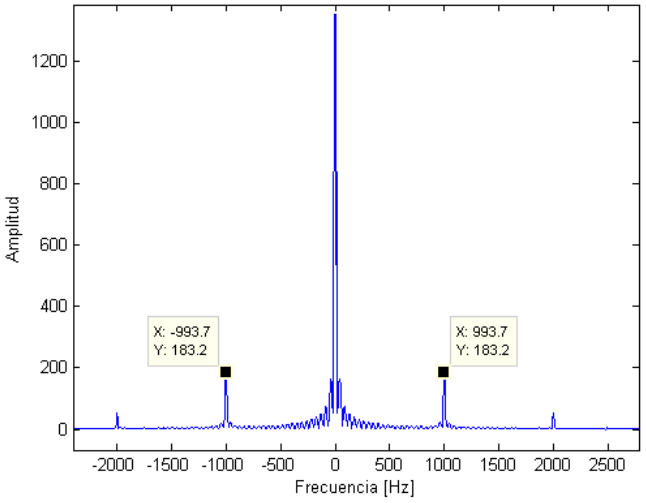


(B)

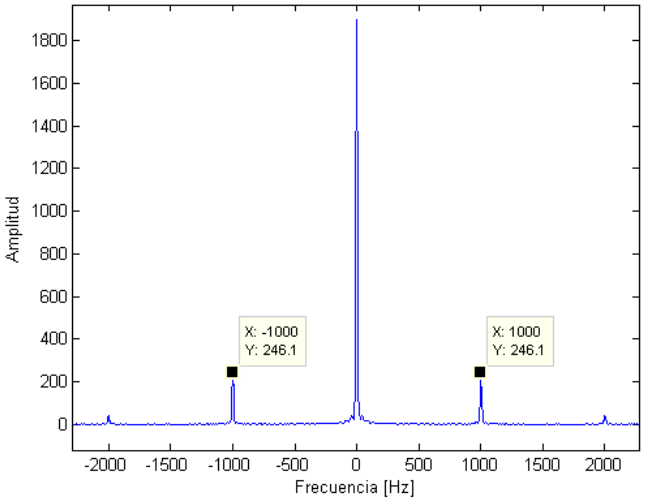
FIGURA 4.13: FFT (A) y densidad espectral de potencia (B)de una muestra de 55000 puntos para una señal de 25 MHZ modulada con 400 Hz.

Recepción en cuadratura

Una vez verificado el funcionamiento correcto de cada uno de los componentes del receptor se puede probar la recepción en cuadratura, en cuyo caso se procesan, almacenan y envían las componentes en fase (I) y en cuadratura (Q). Aunque, con esta recepción podemos almacenar en la memoria programada la mitad de la información que usando un mezclador sencillo, al llevar a cabo decimaciones por factores como 300, 600 y 1000, podemos recibir unos 250 ms de datos. En la Figura 4.14 se muestran las componentes cosenos y seno recibidas de una señal de 21 MHz con una modulación AM de 1 Khz.



(A)

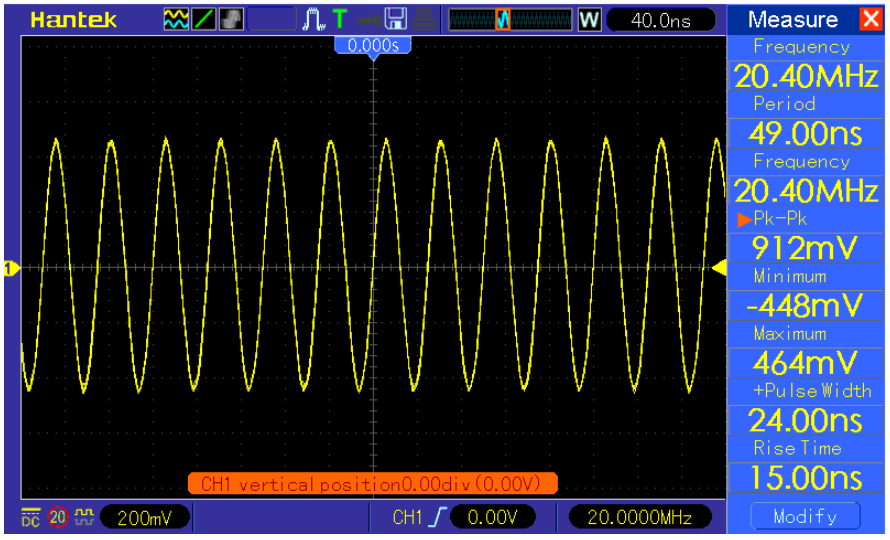


(B)

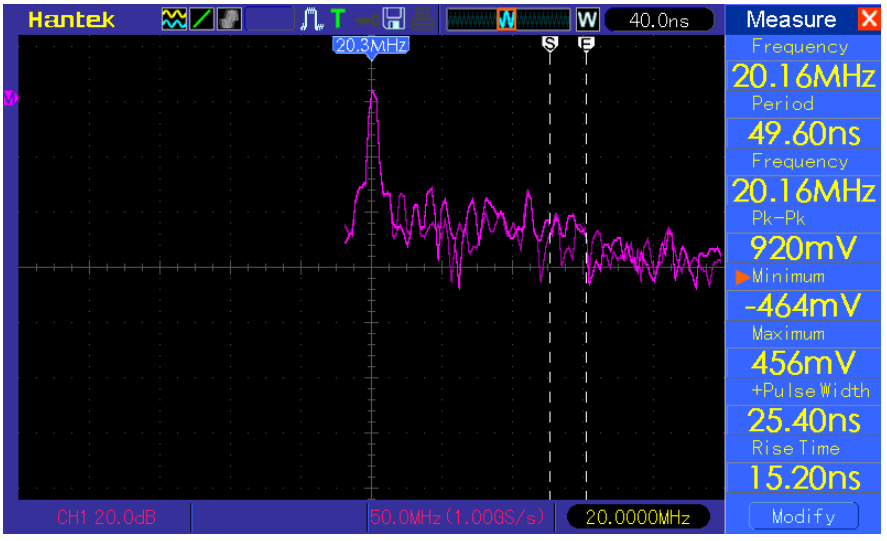
FIGURA 4.14: Componente coseno (A) y seno (B) de una señal portadora de 21 MHz con modulación de 1 KHz y factor de decimación de 300.

4.3.2. Transmisor

En las Figuras 4.15 y 4.16 se muestran, tanto en el dominio del tiempo, como en el dominio de la frecuencia, dos señales generadas con el DAC. Se usó el PLL desactivado y una frecuencia de reloj de 150 MHz. Se muestran 20 MHz y 50 MHz, respectivamente.

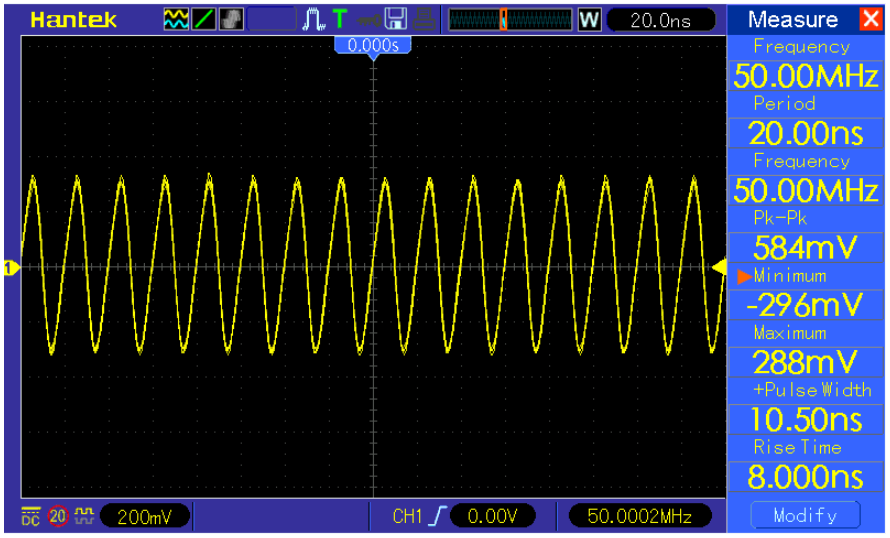


(A)

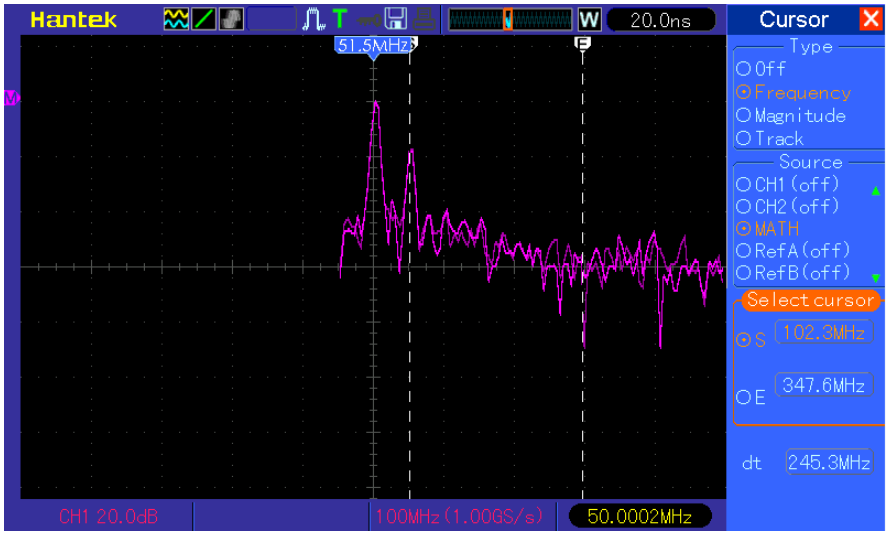


(B)

FIGURA 4.15: Señal generada con DAC de 20 MHz y su FFT.



(A)



(B)

FIGURA 4.16: Señal generada con DAC de 50 MHz y su FFT.

Receptor-transmisor digital completo

Las pruebas anteriores nos permitieron comprobar la calidad de la señal generada, donde, dado que usamos una frecuencia de trabajo de 150 MHz para el DAC, la frecuencia máxima que se puede generar son 75 MHz. En la Figura 4.17 se muestra una señal senoidal de 30 MHz modulada con un pulso cuadrado de ancho de 10 ms. Esta señal nos servirá para verificar el funcionamiento del radio completo mostrado en la Figura 4.18.

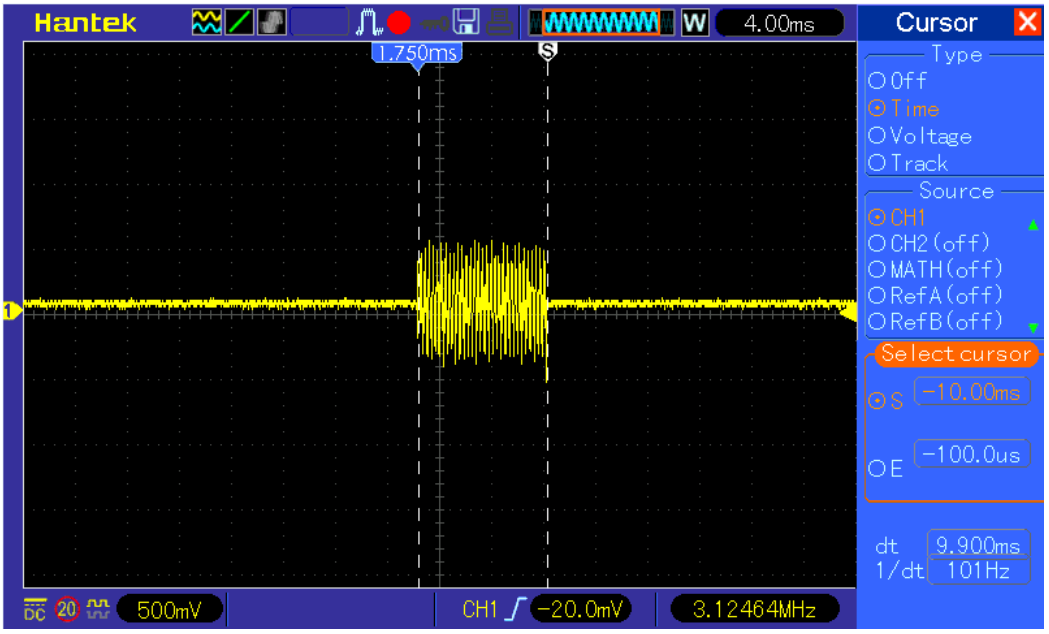


FIGURA 4.17: Pulso generado de 10 ms de duración.

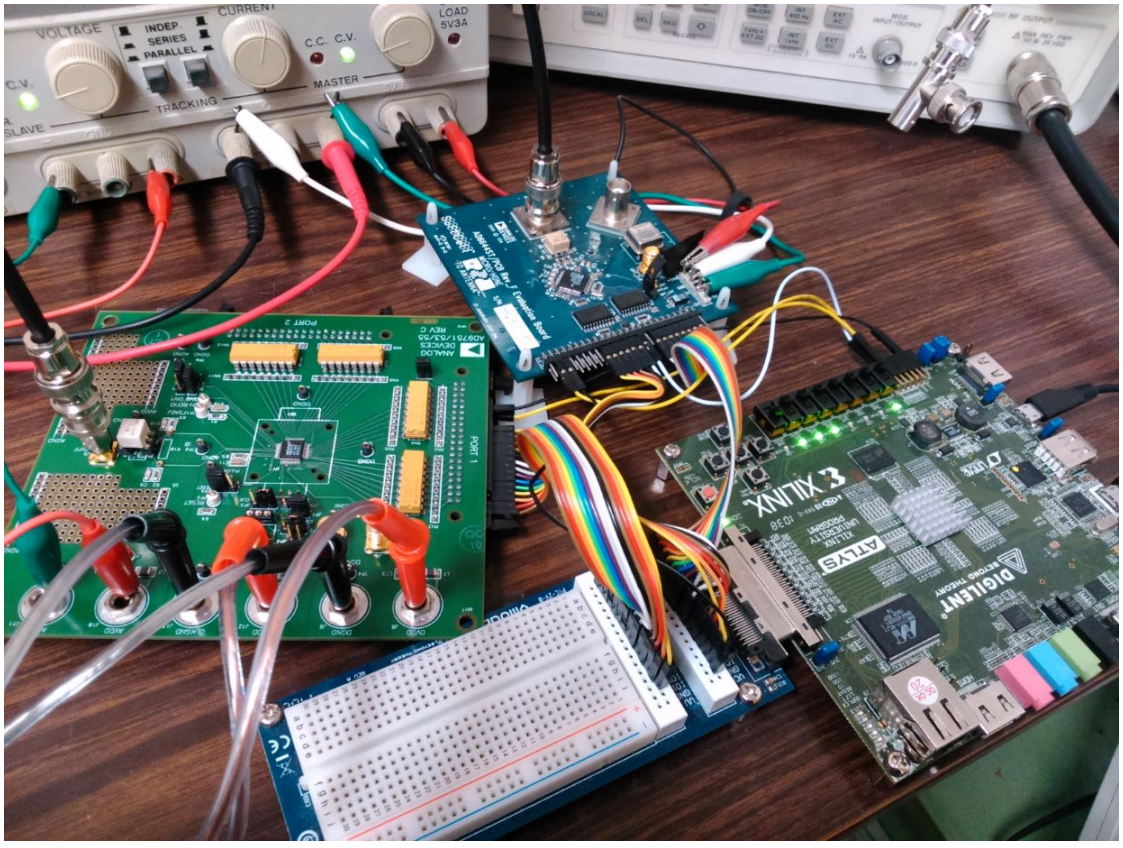
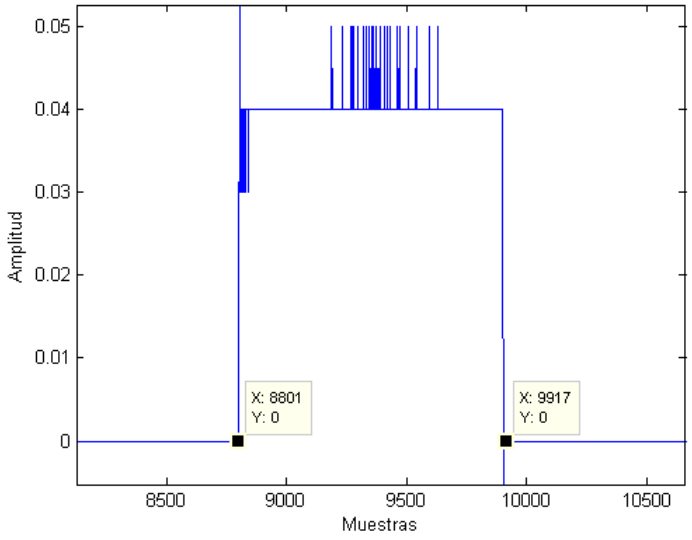


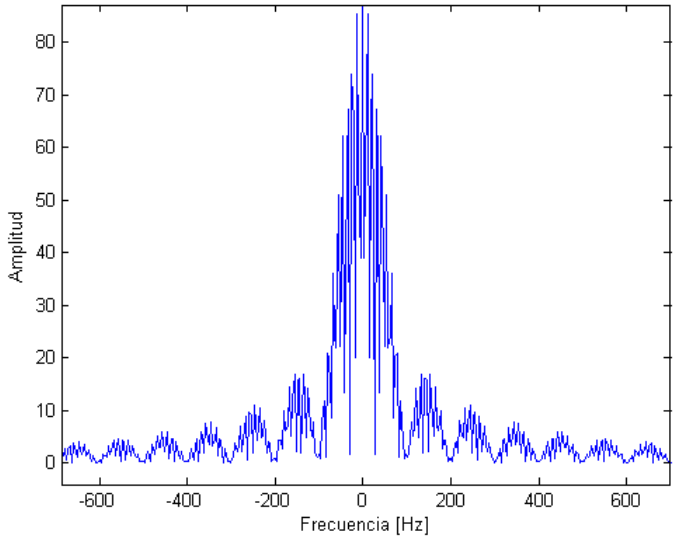
FIGURA 4.18: Sistema receptor-transmisor completo.

En la Figura 4.19 y 4.20 se muestran las componentes en fase y cuadratura, respectivamente, de la adquisición de un pulso de 10 ms. El mezclador esta configurado a la

misma frecuencia de la señal portadora, es decir 50 KHz. El pulso generado tiene un ancho de 10 ms y un intervalo de repetición de 80 ms; como el factor de decimación usado en la prueba es de 600, se están adquiriendo aproximadamente 252 ms, intervalo en el cual se alcanzan a ver dos de los pulsos. En las Figuras 4.19a y 4.20a, se muestran las componente seno y coseno recibidas, donde se aprecia que las señales tienen algunas componentes de frecuencias no deseadas, generadas en el proceso de decimación. Por lo tanto, a las señales se les aplica un filtro FIR paso bajas de orden 40 y con una frecuencia de corte de 1 KHz, con el objetivo de limpiarlas. La respuesta de este filtro se muestra en la Figura 4.21, donde se comparan la respuesta del filtro diseñado en Matlab con su versión cuantizada para el FPGA, el filtro no se implementó en el FPGA debido a los recursos disponibles. En 4.19b y 4.20b, se muestra la FFT de las componentes filtradas. En ambas componentes se puede comprobar el ancho del pulso generado.

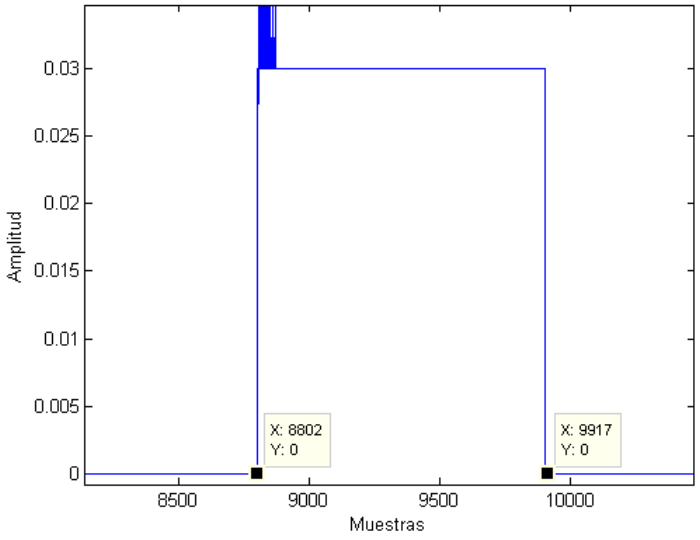


(A)

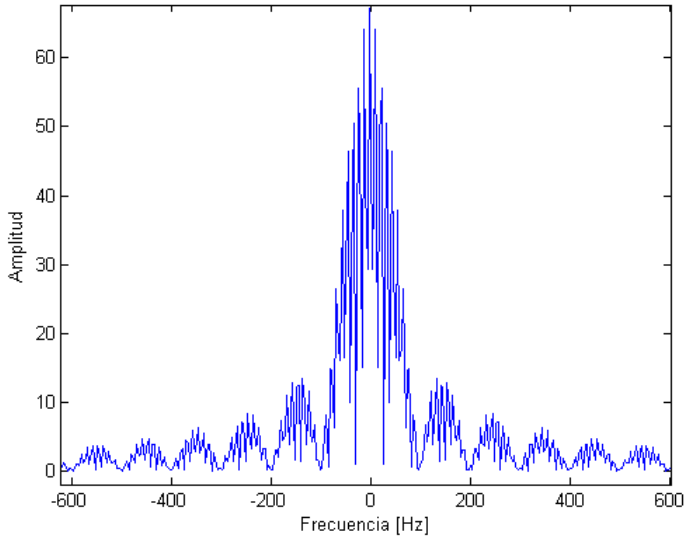


(B)

FIGURA 4.19: Componente coseno recibida (A) y FFT (B) después del filtrado de un pulso de 10 ms con portadora de 50 KHz y factor de decimación 600.



(A)



(B)

FIGURA 4.20: Componente seno recibida (A) y FFT (B) después del filtrado de un pulso de 10 ms con portadora de 50 KHz y factor de decimación 600.

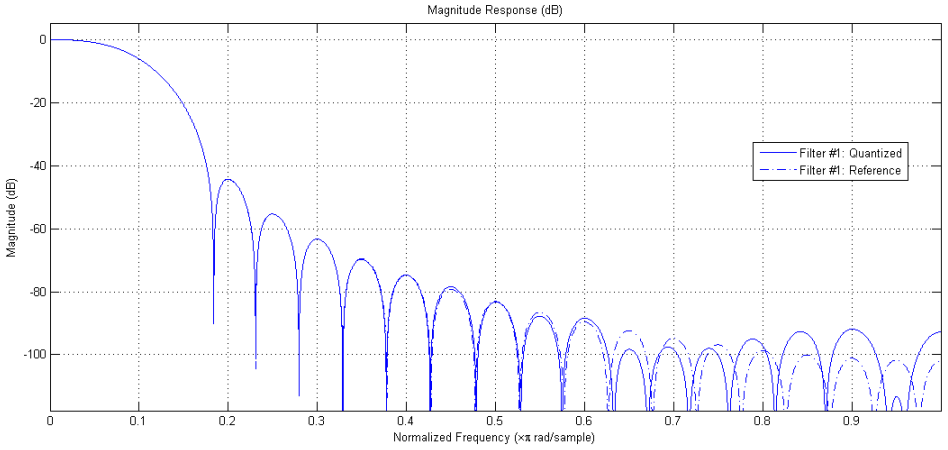


FIGURA 4.21: Respuesta normalizada del filtro pasa bajas, antes y después de la cuantización.

4.3.3. Recursos hardware

A continuación se muestra un resumen de los recursos utilizados en el FPGA para la implementación completa del sistema.

```
1  Design Summary
2  --------------
3  Number of errors:      0
4  Number of warnings:    1
5  Slice Logic Utilization:
6    Number of Slice Registers:                1,040 out of  54,576    1%
7      Number used as Flip Flops:              1,040
8      Number used as Latches:                     0
9      Number used as Latch-thrus:                 0
10     Number used as AND/OR logics:               0
11   Number of Slice LUTs:                       693 out of  27,288    2%
12     Number used as logic:                     544 out of  27,288    1%
13       Number using O6 output only:            246
14       Number using O5 output only:             91
15       Number using O5 and O6:                 207
16       Number used as ROM:                       0
17     Number used as Memory:                    105 out of   6,408    1%
18       Number used as Dual Port RAM:             0
19       Number used as Single Port RAM:           0
20       Number used as Shift Register:          105
```

```
21          Number using O6 output only:             8
22          Number using O5 output only:             0
23          Number using O5 and O6:                 97
24      Number used exclusively as route-thrus:     44
25        Number with same-slice register load:     21
26        Number with same-slice carry load:        23
27        Number with other load:                    0
28  Slice Logic Distribution:
29    Number of occupied Slices:                  364 out of   6,822    5 %
30    Number of MUXCYs used:                      200 out of  13,644    1 %
31    Number of LUT Flip Flop pairs used:         964
32      Number with an unused Flip Flop:          173 out of     964   17 %
33      Number with an unused LUT:                271 out of     964   28 %
34      Number of fully used LUT-FF pairs:        520 out of     964   53 %
35      Number of unique control sets:             25
36      Number of slice register sites lost
37        to control set restrictions:             70 out of  54,576    1 %
38  IO Utilization:
39    Number of bonded IOBs:                      39 out of     218   17 %
40      Number of LOCed IOBs:                     39 out of      39  100 %
41      IOB Flip Flops:                            2
42  Specific Feature Utilization:
43    Number of RAMB16BWERs:                    112 out of     116   96 %
44    Number of RAMB8BWERs:                       1 out of     232    1 %
45    Number of BUFIO2/BUFIO2_2CLKs:              1 out of      32    3 %
46      Number used as BUFIO2s:                   1
47      Number used as BUFIO2_2CLKs:              0
48    Number of BUFIO2FB/BUFIO2FB_2CLKs:          1 out of      32    3 %
49      Number used as BUFIO2FBs:                 1
50      Number used as BUFIO2FB_2CLKs:            0
51    Number of BUFG/BUFGMUXs:                    2 out of      16   12 %
52      Number used as BUFGs:                     2
53      Number used as BUFGMUX:                   0
54    Number of DCM/DCM_CLKGENs:                  0 out of       8    0 %
55    Number of ILOGIC2/ISERDES2s:                1 out of     376    1 %
56      Number used as ILOGIC2s:                  1
57      Number used as ISERDES2s:                 0
58    Number of IODELAY2/IODRP2/IODRP2_MCBs:      0 out of     376    0 %
59    Number of OLOGIC2/OSERDES2s:                1 out of     376    1 %
60      Number used as OLOGIC2s:                  1
61      Number used as OSERDES2s:                 0
```

```
62   Number of BSCANs:                          0 out of       4    0 %
63   Number of BUFHs:                           0 out of     256    0 %
64   Number of BUFPLLs:                         0 out of       8    0 %
65   Number of BUFPLL_MCBs:                     0 out of       4    0 %
66   Number of DSP48A1s:                       10 out of      58   17 %
67   Number of ICAPs:                           0 out of       1    0 %
68   Number of MCBs:                            0 out of       2    0 %
69   Number of PCILOGICSEs:                     0 out of       2    0 %
70   Number of PLL_ADVs:                        1 out of       4   25 %
71   Number of PMVs:                            0 out of       1    0 %
72   Number of STARTUPs:                        0 out of       1    0 %
73   Number of SUSPEND_SYNCs:                   0 out of       1    0 %
74
```

Donde se observa que la memoria es el recurso que se agotó completamente.

Capítulo 5

Conclusiones

5.1. Metas logradas

En este trabajo se mostró el diseño y la implementación de un radio digital. Se aplicaron para el receptor, satisfactoriamente, técnicas tanto de submuestreo como de sobremuestreo; para el receptor se generaron secuencias de pulso sencillas; y se verificó el correcto funcionamiento de cada uno de los componentes del sistema, así como el funcionamiento completo. Se lograron adquirir señales de hasta 100 MHz y con modulaciones desde 400 Hz hasta 25 KHz. La interfaz con Matlab permitió la correcta adquisición de los datos generados por el receptor para poder analizarla, adquiriendo hasta 250 ms para la recepción en cuadratura y el doble para un receptor sencillo. Para el transmisor se generaron señales de distintas frecuencias con una tasa de 150 MHz, las cuales nos permitieron retroalimentar el sistema. Con esto, se pudo obtener un sistema de recepción y transmisión digital básico con el cual se logró llevar a banda base señales de RF y, observar de una manera más tangible cada una de las etapas del sistema, siendo ideal para aplicaciones con propósitos educativos. Debido a que la implementación se llevó a cabo con VHDL, puede trasladarse fácilmente a otro dispositivo para incrementar la funcionalidad y con la interfaz con la computadora, que permite el almacenamiento de los resultados, se puede llevar a cabo un análisis completo así como procesamiento posterior. Durante el desarrollo del proyecto el principal obstáculo que se tuvo fue el del almacenamiento de los datos en el receptor, debido a que se adquieren a una frecuencia de 66 MHz, incluso llevando a cabo decimación, no es posible, usando solo los recursos de memoria internos del FPGA, almacenar suficientes datos como para llevar a cabo pruebas en el orden de

segundos, ni en tiempo real, debido a la velocidad de transmisión de la interfaz UART. Esta es una limitación importante, consecuencia del uso de memoria interna del FPGA para la implemntación de la memoria.

Ya que los sitemas de RF están presentes en casi todos los equipos que usamos diariamente, contar con un sistema completo de recepción y transmisión, que funciones como base para el desarrollo de diversos proyectos es de suma importancia, pues las aplicaciones de los radios digitales van más allá de la formación de imágenes por ultrasonido y RMN. Con este sistema se resuelve parte del problema que implica el uso de un sistema comercial, el cual es el acceso restringido a los datos de radiofrecuencia generados, pues generalmente en los equipos comerciales se tienen que comprar licencias de alto costo.

5.2. Trabajos por realizar

Como se mencionó, la principal limitación del sistema es el almacenamiento de los datos generados en el receptor. La tarjeta de desarrollo Atlys, tiene una memoria MT47H64M16-25E de 128 MByte DDR2 de 16-bits, la cual puede comunicarse con el FPGA a una tasa de hasta 800 MHz. El uso de la memoria junto con el envío continuo de los datos al CPU, permitiría llevar a cabo recepciones en el orden de segundos incrementando las capacidades del sistema, tanto en el tiempo de adquisición, como en la variedad de señales y experimentos a realizar. En el caso del transmisor, la arquitectura completa de un DUC no se implementó debido a las mismas limitaciones que con la memoria, es decir los recursos del FPGA, el uso de los *XtremeDSP Slice* nos facilita el diseño y aumenta el desempeño de los componentes, pero se tiene un número limitado de estos, restringiendo considerablemente el diseño. Por esta misma razón, la última etapa de filtrado para el receptor, tampoco se implemento en el FPGA. Por lo tanto, la utilización de los *Xilinx LogiCORE IPCores* facilitó la implementación del sistema, lo cual es ideal para obtener un primer sistema base, sin embargo la elaboración de algoritmos propios para los distintos bloques del radio, puede aumentar el desempeño del mismo. Un ejemplo de esto, es que el bloque de decimación no acepta entradas mayores a 24 bits, por lo que se tuvo que llevar a cabo un truncamiento de bits que afecta la resolución del dato, además, a pesar de que necesita menos recursos para su implementación, en comparación con los filtros FIR, los CIC tienen la desventaja de que son propensos a tener una caída en amplitud de su banda de paso y rizos en su banda de paro, y es por esto que generalmente

se implementa un filtrado extra en receptor, que al tener una entrada de menor tasa, no consumirá tantos recursos y tendrá especificaciones más relajadas. Por otro lado, la arquitectura combinada hardware-software es actualmente el futuro de muchos sistemas de procesamiento digital de señales que requieran altas velocidades de procesamiento, por lo tanto, si además de extender las capacidades de almacenamiento, se trabaja conjuntamente con un DSP, se podrán implementar algoritmos de procesamiento en banda base.

Apéndice A

Códigos Matlab

A continuación se muestra el código utilizado para la recepción de los datos en Matlab.

```
%Limpia variables y pantalla
clear all
close all
clc
%Borrar conexiones previas
delete(instrfind({'Port'},{'COM5'}));
%Crear una conexion serie
s = serial('COM5','BaudRate',115200);
%Abrir el puerto
fopen(s);
%Mandar A
fprintf(s, 'A');
%Leer el puerto serie
mem=55498;
n=(mem*4)+1;
val=zeros(1,n);
for i=1:n
val(1,i) = fread(s,1,'uchar');
end
%Cierra puerto
fclose(s);
delete(s);
clear s;
val=val';
```

```
25 a=val;
26 t=length(a);
27 %Quitar primer dato basura
28 a=a(2:t);
29 t=length(a);
30 %Juntar 4 bytes en dato
31 a1=dec2hex(a);
32 g=4:4:t+2;
33 h=3:4:t+1;
34 i=2:4:t;
35 j=1:4:t-1;
36 for k=1:t/4
37     v=g(1,k);
38     w=h(1,k);
39     x=i(1,k);
40     y=j(1,k);
41     a2(k,:)=strcat(a1(v,:),a1(w,:),a1(x,:),a1(y,:));
42 end
43 b=hex2dec(a2);
44 b2=b;
45 t2=length(b);
46 %Equivalente a voltaje
47 for w=1:t2
48 if (b(w) ≥ 2147483648)
49     b2(w)=b(w)-4294967296;
50 end
51 end
52 b3=(b2/2147483648)*1.1;
53 t3=length(b3);
54 %Guarda en archivo.dat
55 dlmwrite('archivo.dat', b3, 'delimiter', '\n', 'precision', '%.2f')
```

Apéndice B

Códigos VHDL

A continuación se muestran los códigos implementados en VHDL. El siguiente código es el archivo principal que define y conecta a los demás.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  -- para oddr2
7  Library UNISIM;
8  use UNISIM.VComponents.all;
9  entity uart_test is
10     Port ( clk : in  STD_LOGIC;
11            reset : in  STD_LOGIC;
12            rx : in  STD_LOGIC;
13               ovr : in std_logic;
14               datain : in std_logic_vector(13 downto 0);
15            tx : out  STD_LOGIC;
16               ovr_led: out std_logic;
17               led1 : out STD_LOGIC;
18               led2 : out STD_LOGIC;
19               led3: out STD_LOGIC;
20               clk_out : out std_logic;
21               dataout : out std_logic_vector(13 downto 0);
22               enable : in std_logic
23              );
```

```
24 end uart_test;
25 architecture arch of uart_test is
26 -- inicializa memoria ram
27 COMPONENT ramyo
28   PORT (
29     clka : IN STD_LOGIC;
30     wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
31     addra : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
32     dina : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
33     clkb : IN STD_LOGIC;
34     addrb : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
35     doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
36   );
37 END COMPONENT;
38 -- inicializa pll
39 component clkm
40 port
41  ( CLK_IN1           : in     std_logic;
42    CLK_OUT1          : out    std_logic;        -- 66 mhz
43    CLK_OUT2          : out    std_logic     -- 150 mhz
44  );
45 end component;
46 -- inicializa cic
47 COMPONENT cicyod
48   PORT (
49     aclk : IN STD_LOGIC;
50     s_axis_data_tdata : IN STD_LOGIC_VECTOR(23 DOWNTO 0);
51     s_axis_data_tvalid : IN STD_LOGIC;
52     s_axis_data_tready : OUT STD_LOGIC;
53     m_axis_data_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
54     m_axis_data_tvalid : OUT STD_LOGIC
55   );
56 END COMPONENT;
57 -- inicializa cicd
58 COMPONENT cicyodd
59   PORT (
60     aclk : IN STD_LOGIC;
61     s_axis_data_tdata : IN STD_LOGIC_VECTOR(23 DOWNTO 0);
62     s_axis_data_tvalid : IN STD_LOGIC;
63     s_axis_data_tready : OUT STD_LOGIC;
64     m_axis_data_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
```

```
65      m_axis_data_tvalid : OUT STD_LOGIC
66    );
67  END COMPONENT;
68  -- inicializa dds
69  COMPONENT ddsyo
70    PORT (
71      clk : IN STD_LOGIC;
72       cosine : OUT STD_LOGIC_VECTOR(13 DOWNTO 0);        -- 20 mhz (66)
73      sine : OUT STD_LOGIC_VECTOR(13 DOWNTO 0)
74    );
75  END COMPONENT;
76  -- inicializa dds dac
77  COMPONENT ddsdac
78    PORT (
79      clk : IN STD_LOGIC;
80      sine : OUT STD_LOGIC_VECTOR(13 DOWNTO 0)            -- 20 mhz (150)
81    );
82  END COMPONENT;
83  -- senales internas
84      -- uart
85      signal tx_inicia: std_logic;
86      signal tx_datos_in: std_logic_vector(7 downto 0);
87      signal tx_listo: std_logic;
88      signal rx_datos_out: std_logic_vector(7 downto 0);
89      signal rx_listo: std_logic;
90      -- ram
91      signal wea_sig: std_logic_vector(0 downto 0);
92      signal addra_sig: std_logic_vector(15 downto 0);
93      signal dina_sig: std_logic_vector(31 downto 0);
94      signal addrb_sig: std_logic_vector(17 downto 0);
95      signal doutb_sig: std_logic_Vector(7 downto 0);
96      -- pll
97      signal clk_mu: std_logic;
98      signal clk_dac: std_logic;
99      signal inv_clk_dac: std_logic;
100     -- cic
101     signal din_cic_sig: std_logic_vector(23 downto 0);
102     signal nd_cic_sig: std_logic;
103     signal dout_cic_sig: std_logic_vector(31 downto 0);
104     signal rdy_cic_sig: std_logic;
105     signal rfd_cic_sig: std_logic;
```

```
106     -- cicd
107     signal din_cic_sig_d: std_logic_vector(23 downto 0);
108     signal nd_cic_sig_d: std_logic;
109     signal dout_cic_sig_d: std_logic_vector(31 downto 0);
110     signal rdy_cic_sig_d: std_logic;
111     signal rfd_cic_sig_d: std_logic;
112     -- dds y mul
113     signal coseno: std_logic_vector(13 downto 0);
114     signal seno: std_logic_vector(13 downto 0);
115     signal senodac: std_logic_vector(13 downto 0);
116 begin
117 -- conexiones uart
118     uart_unit: entity work.uart(str_arch)
119         port map(clk=>clk_mu, reset=>reset,
120                     rx=>rx, tx=>tx,
121                     tx_start => tx_inicia,
122                     tx_data_in => tx_datos_in,
123                     tx_done_tick => tx_listo,
124                     rx_data_out => rx_datos_out,
125                     rx_done_tick => rx_listo);
126 -- conexiones controlador adc memoria y uart
127     contram_unit: entity work.contram(ramcarq)
128         port map(clk => clk_mu, reset => reset,
129            tx_start => tx_inicia,
130               tx_data_in => tx_datos_in,
131               tx_done_tick => tx_listo,
132               rx_data_out => rx_datos_out,
133               rx_done_tick => rx_listo,
134              addrb => addrb_sig,
135              doutb => doutb_sig,
136               ovr => ovr,
137               ovr_out => ovr_led,
138            datain => datain,
139               wea => wea_sig,
140              addra => addra_sig,
141              dina => dina_sig,
142               ------------------------
143               din_cic => din_cic_sig,
144               nd_cic => nd_cic_sig,
145               dout_cic => dout_cic_sig,
146               rdy_cic => rdy_cic_sig,
```

```
147               rfd_cic => rfd_cic_sig,
148               -------------------------
149               din_cic_d => din_cic_sig_d,
150               nd_cic_d => nd_cic_sig_d,
151               dout_cic_d => dout_cic_sig_d,
152               rdy_cic_d => rdy_cic_sig_d,
153               rfd_cic_d => rfd_cic_sig_d,
154               ------------------------
155               led1i => led1,
156               led2i => led2,
157               led3i => led3,
158               cosenoidal => coseno,
159               senoidal => seno
160               );
161 -- conexiones ram
162     ramyo_unit : ramyo
163         PORT MAP (
164          clka => clk_mu,
165          wea => wea_sig,
166          addra => addra_sig,
167          dina => dina_sig,
168          clkb => clk_mu,
169          addrb => addrb_sig,
170          doutb => doutb_sig);
171 -- conexiones cic
172 cicyod_unit : cicyod
173   PORT MAP (
174     aclk => clk_mu,
175     s_axis_data_tdata => din_cic_sig,
176     s_axis_data_tvalid => nd_cic_sig,
177     s_axis_data_tready => rfd_cic_sig,
178     m_axis_data_tdata => dout_cic_sig,
179     m_axis_data_tvalid => rdy_cic_sig
180   );
181 -- conexiones cicd
182 cicyodd_unit: cicyodd
183   PORT MAP (
184     aclk => clk_mu,
185     s_axis_data_tdata => din_cic_sig_d,
186     s_axis_data_tvalid => nd_cic_sig_d,
187     s_axis_data_tready => rfd_cic_sig_d,
```

```
188     m_axis_data_tdata => dout_cic_sig_d,
189     m_axis_data_tvalid => rdy_cic_sig_d
190   );
191 -- conexiones pll
192     clkm_unit : clkm
193       port map
194         (CLK_IN1 => clk,
195          CLK_OUT1 => clk_mu,        -- 66 mhz
196          CLK_OUT2 => clk_dac        -- 150 mhz
197          );
198  -- conexiones dds
199  ddsyo_unit : ddsyo
200   PORT MAP (
201     clk => clk_mu,              -- 66
202      cosine => coseno,          -- 20 Mhz
203     sine => seno
204   );
205  -- conexiones ddsdac
206  ddsdac_unit : ddsdac
207   PORT MAP (
208     clk => clk_dac,             -- 150
209     sine => senodac             -- 50 khz
210   );
211 -- conexiones dac
212 contdac_unit: entity work.contdac(contdacarq)
213     port map(clk=>clk_mu, reset=>reset,
214         clk66 => clk_mu,
215         datai => seno,              --- de dds
216         datao => dataout,
217         habilitadac => enable
218         );
219 -- conexiones ODDR2
220 inv_clk_dac <= NOT clk_mu;
221    -- Clock forwarding circuit using the double data-rate register
222    --        Spartan-3E/3A/6
223    -- Xilinx HDL Language Template, version 14.6
224    ODDR2_inst : ODDR2
225    generic map(
226       DDR_ALIGNMENT => "NONE", -- Sets output alignment to "NONE", "C0", "C1"
227       INIT => '0', -- Sets initial state of the Q output to '0' or '1'
228       SRTYPE => "SYNC") -- Specifies "SYNC" or "ASYNC" set/reset
```

```
229    port map (
230       Q => clk_out, -- 1-bit output data
231       C0 => clk_mu, -- 1-bit clock input
232       C1 => inv_clk_dac, -- 1-bit clock input
233       CE => '1',  -- 1-bit clock enable input
234       D0 => '0',   -- 1-bit data input (associated with C0)
235       D1 => '1',   -- 1-bit data input (associated with C1)
236       R => '0',    -- 1-bit reset input
237       S => '0'     -- 1-bit set input
238    );
239 end arch;
```

El siguiente código se encarga de la interfaz con el ADC, el DDC, almacenamiento y envío de la información a la computadora.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  entity contram is
7      Port ( clk : in  STD_LOGIC;
8             reset : in  STD_LOGIC;
9                -- piner para uart
10               tx_start: out std_logic;
11               tx_data_in: out std_logic_vector(7 downto 0);
12               tx_done_tick: in std_logic;
13               rx_data_out: in std_logic_vector(7 downto 0);
14               rx_done_tick: in std_logic;
15               -- pines para ram
16              addrb: out std_logic_vector(17 downto 0);
17              doutb: in std_logic_vector(7 downto 0);
18               wea: out std_logic_vector(0 downto 0);
19              addra: out std_logic_vector(15 downto 0);
20              dina: out std_logic_vector(31 downto 0);
21               -- pines adc
22               ovr : in  STD_LOGIC;
23               ovr_out : out std_logic;
24               datain : in  STD_LOGIC_VECTOR (13 downto 0);
25               -- pines para cic
```

```
26              din_cic: out std_logic_vector(23 downto 0);
27              nd_cic: out std_logic;
28              dout_cic: in std_logic_vector(31 downto 0);
29              rdy_cic: in std_logic;
30              rfd_cic: in std_logic;
31              -- pines para cicd
32              din_cic_d: out std_logic_vector(23 downto 0);
33              nd_cic_d: out std_logic;
34              dout_cic_d: in std_logic_vector(31 downto 0);
35              rdy_cic_d: in std_logic;
36              rfd_cic_d: in std_logic;
37              -- leds indicadores
38              led1i : out STD_LOGIC;
39              led2i : out STD_LOGIC;
40              led3i : out STD_LOGIC;
41              -- pines dds
42              cosenoidal: in std_logic_vector(13 downto 0);
43              senoidal: in std_logic_vector(13 downto 0)
44              );
45 end contram;
46 architecture ramcarq of contram is
47 -- inicializa multiplicador
48 COMPONENT muluyo
49   PORT (
50     clk : IN STD_LOGIC;
51     a : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
52     b : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
53     p : OUT STD_LOGIC_VECTOR(27 DOWNTO 0)
54   );
55 END COMPONENT;
56 -- inicialliza multiplicador dos
57 COMPONENT muluyod
58   PORT (
59     clk : IN STD_LOGIC;
60     a : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
61     b : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
62     p : OUT STD_LOGIC_VECTOR(27 DOWNTO 0)
63   );
64 END COMPONENT;
65 -- para maquina de estados que lee  memoria
66 signal e: std_logic_vector(4 downto 0):="00000";
```

```
67  constant s0: std_logic_vector(4 downto 0):="00000";
68  constant s1: std_logic_vector(4 downto 0):="00001";
69  constant s2: std_logic_vector(4 downto 0):="00010";
70  constant s3: std_logic_vector(4 downto 0):="00011";
71  constant s4: std_logic_vector(4 downto 0):="00100";
72  constant s5: std_logic_vector(4 downto 0):="00101";
73  constant s6: std_logic_vector(4 downto 0):="00110";
74  constant s7: std_logic_vector(4 downto 0):="00111";
75  constant s8: std_logic_vector(4 downto 0):="01000";
76  constant s9: std_logic_vector(4 downto 0):="01001";
77  constant s10: std_logic_vector(4 downto 0):="01010";
78  constant s11: std_logic_vector(4 downto 0):="01011";
79  constant s12: std_logic_vector(4 downto 0):="01100";
80  constant s13: std_logic_vector(4 downto 0):="01101";
81  constant s14: std_logic_vector(4 downto 0):="01110";
82  constant s15: std_logic_vector(4 downto 0):="01111";
83  constant s16: std_logic_vector(4 downto 0):="10000";
84  constant s17: std_logic_vector(4 downto 0):="10001";
85  constant s18: std_logic_vector(4 downto 0):="10010";
86  constant s19: std_logic_vector(4 downto 0):="10011";
87  constant s20: std_logic_vector(4 downto 0):="10100";
88  constant s21: std_logic_vector(4 downto 0):="10101";
89  constant s22: std_logic_vector(4 downto 0):="10110";
90  constant s23: std_logic_vector(4 downto 0):="10111";
91  constant s24: std_logic_vector(4 downto 0):="11000";
92  constant s25: std_logic_vector(4 downto 0):="11001";
93  constant s26: std_logic_vector(4 downto 0):="11010";
94  constant s27: std_logic_vector(4 downto 0):="11011";
95  constant s28: std_logic_vector(4 downto 0):="11100";
96  constant s29: std_logic_vector(4 downto 0):="11101";
97  constant s30: std_logic_vector(4 downto 0):="11110";
98  constant s31: std_logic_vector(4 downto 0):="11111";
99  -- para maquina de estados que escribe memoria
100 signal ed: std_logic_vector(2 downto 0):="000";
101 constant sd0: std_logic_vector(2 downto 0):="000";
102 constant sd1: std_logic_vector(2 downto 0):="001";
103 constant sd2: std_logic_vector(2 downto 0):="010";
104 constant sd3: std_logic_vector(2 downto 0):="011";
105 constant sd4: std_logic_vector(2 downto 0):="100";
106 constant sd5: std_logic_vector(2 downto 0):="101";
107 constant sd6: std_logic_vector(2 downto 0):="110";
```

```
108 constant sd7: std_logic_vector(2 downto 0):="111";
109 -- para escribir en tx
110 signal tx_start_s: std_logic:= '0';
111 signal tx_data_in_s: std_logic_vector(7 downto 0):=x"00";
112 signal addrb_s: std_logic_vector(17 downto 0):="000000000000000000";
113 -- para leer de ram
114 signal addra_s: std_logic_vector(15 downto 0):="0000000000000000";
115 signal dina_s: std_logic_vector(31 downto ...
        0):="00000000000000000000000000000000";
116 signal wea_s: std_logic_vector(0 downto 0):= "0";
117 -- para cic
118 signal din_cic_s: std_logic_vector(23 downto 0):= "000000000000000000000000";
119 signal bandera: std_logic:= '0';
120 -- para cic_d
121 signal din_cic_s_d: std_logic_vector(23 downto 0):= ...
        "000000000000000000000000";
122 -- para verificar cic
123 signal cic_fin: std_logic_vector(1 downto 0):= "00";
124 -- para multiplicador
125 signal pse: std_logic_vector(27 downto 0):="0000000000000000000000000000";
126 -- para multiplicador dos
127 signal pse_d: std_logic_vector(27 downto 0):="0000000000000000000000000000";
128 -- para latencia del dds
129 signal laten: std_logic_vector(4 downto 0):="00000";
130 -- para cont muestras
131 signal cuenta: std_logic_vector(23 downto 0):="000000000000000000000000";
132 begin
133 ----------------------------------------------------------------------------------------
134 -- solo lee ram
135 state_machine:process(clk, reset, tx_done_tick, rx_data_out, ...
        rx_done_tick, doutb, addrb_s, e, rfd_cic, rfd_cic_d, bandera, datain, ...
        senoidal, laten, ovr)
136 begin
137     if reset='1' then
138         e <= s0;
139         tx_start_s <= '0';
140         tx_data_in_s <= x"00";
141         addrb_s <= "000000000000000000";
142         din_cic_s <= "000000000000000000000000";
143         din_cic_s_d <= "000000000000000000000000";
144         nd_cic <= '0';
```

```
145         nd_cic_d ≤ '0';
146         led1i ≤ '0';
147         led2i ≤ '0';
148         led3i ≤ '0';
149         laten ≤ "00000";
150         cuenta ≤ "000000000000000000000000";
151         ovr_out ≤ '0';
152     elsif (clk'event and clk='1') then
153         ovr_out ≤ ovr;
154         case e is
155         ---------------------------------- ADQUISICION Y MULTIPLICACION ...
    -------------------------------------
156             when s0 =>                      -- espera a que pasen los ...
    ciclos de latencia mul y dds
157                 led1i ≤ '0';
158                 led2i ≤ '0';
159                 led3i ≤ '0';
160                 if laten = "10000" then
161                     laten ≤ "00000";
162                     e ≤ s1;
163                 else
164                     laten ≤ laten + 1;
165                     e ≤ s0;
166                 end if;
167             ---------------------------------- ENVIO DE DATOS A CIC ...
    -------------------------------------
168             when s1 =>                      -- si cic este listo para ...
    recibir el dato y no se ha enviado el tope los manda
169                 if rfd_cic = '1' then
170                     if rfd_cic_d = '1' then
171                         if cuenta = "111111100000111100010000" then -- ...
    16650000 (55500*600)/2
172                             cuenta ≤ "000000000000000000000000";
173                             nd_cic ≤ '0';
174                             nd_cic_d ≤ '0';
175                             e ≤ s2;
176                         else
177                             cuenta ≤ cuenta +1;
178                             nd_cic ≤ '1';
179                             nd_cic_d ≤ '1';
```

```
180                         din_cic_s <= pse(27 downto 4);       -- tienen ...
    que quitar los 4 menos significativos  sin
181                         din_cic_s_d <= pse_d(27 downto 4);       -- ...
    tienen que quitar los 4 menos significativos  cos
182                         e <= s1;
183                     end if;
184                     --nd_cic <= '1';
185                 else
186                     e <= s0;
187                 end if;
188             else
189                 e <= s0;
190             end if;
191         when s2 =>                      -- termino de dar datos a ...
    cic, espera que se guarden los resultados
192             led1i <= '1';
193             if bandera = '1' then
194                 led2i <= '1';           -- cic termino de guardar ...
    salidas en memoria
195                 e <= s3;
196             else
197                 e <= s2;
198             end if;
199         ---------------------------------- ENVIO DE DATOS PROCESADOS A ...
    PC -------------------------------------
200         when s3 =>                      -- espera el dato de matlab ...
    para comenzar envio A
201             if rx_done_tick = '1' then
202                 if rx_data_out = x"41" then
203                     e <= s4;
204                 else
205                     e <= s3;
206                 end if;
207             else
208                 e <= s3;
209             end if;
210         when s4 =>
211             if addrb_s = "110110001100101000" then --- si llego al ...
    tope termina 55500*4
212                 e <= s7;
213             else
```

```
214                     addrb_s ≤ addrb_s + 1;      -- envia direccion para ...
    lee ram
215                     e ≤ s5;
216                 end if;
217             when s5 =>                      -- escribe en tx el valor leido
218                 tx_start_s  ≤ '1';
219                 tx_data_in_s ≤ doutb;
220                 e ≤ s6;
221             when s6 =>                      -- espera a que se haya ...
    escrito el dato en tx
222                 if tx_done_tick = '1' then
223                     e ≤ s4;
224                 else
225                     e ≤ s6;
226                 end if;
227             when s7 =>                  -- termino de escribir en tx ...
    todos los datos de la memoria
228                 addrb_s ≤ "0000000000000000"; -- reinicia contador de ...
    lectura
229                 led3i ≤ '1';
230                 e ≤ s8;
231             when s8 =>
232                 tx_start_s  ≤ '0';
233                 e ≤ s8;
234             when others =>
235                 e ≤ s0;
236         end case;
237     end if;
238 end process;
239 ----------------------------------------------------------------------------------------
240 -- solo escribe ram
241 state_machined:process(clk, reset, ed, addra_s, dout_cic, rdy_cic, ...
    dout_cic_d, rdy_cic_d)
242 begin
243 if reset='1' then
244         ed ≤ sd0;
245         wea_s ≤ "0";
246         addra_s ≤ "0000000000000000";
247         dina_s ≤ "00000000000000000000000000000000";
248         bandera ≤ '0';
249         cic_fin ≤ "00";
```

```
250     elsif (clk'event and clk='1') then
251         case ed is
252              ---------------------------------- ALMACENAR RESULTADOS DE ...
        CIC -------------------------------------
253             when sd0 =>                 -- espera salida de cic
254                 if cic_fin = "11" then
255                     ed <= sd1;
256                 else
257                     if rdy_cic = '1' then
258                         cic_fin(0) <= '1';
259                     end if;
260                     if rdy_cic_d = '1' then
261                         cic_fin(1) <= '1';
262                     end if;
263                     ed <= sd0;
264                 end if;
265             when sd1 =>                 -- guarda en memoria primero seno
266                 cic_fin <= "00";
267                 wea_s <= "1";
268                 addra_s <= addra_s + 1;
269                 dina_s <= dout_cic;
270                 ed <= sd2;
271             when sd2 =>                 -- guarda en memoria luego coseno (d)
272                 wea_s <= "1";
273                 addra_s <= addra_s + 1;
274                 dina_s <= dout_cic_d;
275                 ed <= sd3;
276             when sd3 =>                 -- cuando sean 55500 activa bandera
277                 wea_s <= "0";
278                 if addra_s = "1101100011001010" then
279                     bandera <= '1';
280                     ed <= sd4;
281                 else
282                     ed <= sd0;
283                 end if;
284             when sd4 =>
285                 ed <= sd4;
286             when others =>
287                 ed <= sd0;
288         end case;
289     end if;
```

```
290 end process;
291 ----------------------------------------------------------------------------------------------
292 -- conexiones multiplicador
293 muluyo_unit : muluyo
294   PORT MAP (
295     clk => clk,
296     a => datain,
297     b => senoidal,
298     p => pse
299   );
300 -- conexiones multiplicador dos
301 muluyod_unit : muluyod
302   PORT MAP (
303     clk => clk,
304     a => datain,
305     b => cosenoidal,
306     p => pse_d
307   );
308 -- salidas
309 tx_start ≤ tx_start_s;
310 tx_data_in ≤ tx_data_in_s;
311 addrb ≤ addrb_s;
312 addra ≤ addra_s;
313 dina ≤ dina_s;
314 wea ≤ wea_s;
315 din_cic ≤ din_cic_s;
316 din_cic_d ≤ din_cic_s_d;
317 end ramcarq;
```

El código siguiente es el que se encarga de generar la secuencia para el DAC.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 use IEEE.STD_LOGIC_ARITH.ALL;
5 use IEEE.STD_LOGIC_UNSIGNED.ALL;
6 entity contdac is
7     Port ( clk : in  STD_LOGIC;
8                clk66 : in STD_LOGIC;
9            reset : in  STD_LOGIC;
```

```
10            datai : in  STD_LOGIC_VECTOR (13 downto 0);
11            datao : out  STD_LOGIC_VECTOR (13 downto 0);
12               habilitadac : in std_logic
13               );
14 end contdac;
15 architecture contdacarq of contdac is
16 signal tempoo: std_logic_vector(13 downto 0):= "00000000000000";
17 signal contadort: std_logic_vector(22 downto 0):= "00000000000000000000000";
18 signal pulso: std_logic := '0';
19 -- para maquina de estados que genera la secuencia
20 signal e: std_logic_vector(2 downto 0):="000";
21 constant s0: std_logic_vector(2 downto 0):="000";
22 constant s1: std_logic_vector(2 downto 0):="001";
23 constant s2: std_logic_vector(2 downto 0):="010";
24 constant s3: std_logic_vector(2 downto 0):="011";
25 constant s4: std_logic_vector(2 downto 0):="100";
26 constant s5: std_logic_vector(2 downto 0):="101";
27 constant s6: std_logic_vector(2 downto 0):="110";
28 constant s7: std_logic_vector(2 downto 0):="111";
29
30 begin
31 ---------------------------------------------------------------------------
32 -- para quitar el complemento a dos
33 pa:process(clk, reset, datai, habilitadac, tempoo, pulso)
34 begin
35     if reset='1' then
36         tempoo <= "00000000000000";
37     elsif (clk'event and clk='1') then
38         if (habilitadac = '1') and (pulso = '1') then
39             tempoo <= datai + "10000000000000";      --- 8192
40         else
41             tempoo <= "00000000000000";
42         end if;
43     end if;
44 end process;
45 ---------------------------------------------------------------------------
46 -- para generar el pulso cuadrado  10 ms
47 pb:process(clk66, reset, contadort)
48 begin
49     if reset='1' then
50         contadort <= "00000000000000000000000";
```

```
51      elsif (clk66'event and clk66='1') then
52          case e is
53              when s0 =>              --   pulso abajo
54                  pulso <= '0';
55                  if contadort = "10100001001000100000000" then --- 80 ms
56                      contadort <= "00000000000000000000000";
57                      e <= s1;
58                  else
59                      contadort <= contadort +1;
60                      e <= s0;
61                  end if;
62              when s1 =>              --    pulso arriba
63                  pulso <= '1';
64                  if contadort = "00001010000101001000100000" then --- 10 ms
65                      contadort <= "00000000000000000000000";
66                      e <= s0;
67                  else
68                      contadort <= contadort +1;
69                      e <= s1;
70                  end if;
71              when others =>
72                  e <= s0;
73          end case;
74      end if;
75  end process;
76  ---------------------------------------------------------------------------
77  datao <= tempoo;
78  end contdacarq;
```

El código para la comunicación UART, así como sus submódulos se muestran a continuación. El siguiente, es el bloque que define las configuraciones generales e interconecta los módulos.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  entity uart is
7      -- baudrate 115200, 8 bits datos, 1 paro
```

```
8     Generic(    DBIT: integer :=8;      -- bits de datos
9                 SB_TICK: integer :=16;  -- ticks 16/24/32 para 1/1.5/2 ...
      bits de paro
10                DVSR: integer :=36;     -- divisor baudrate = ...
      6M/(16*115200)=  36
11                DVSR_BIT: integer :=6   -- bits para DVSR
12                );
13    Port ( clk : in  STD_LOGIC;
14           reset : in  STD_LOGIC;
15              rx : in  STD_LOGIC;
16              tx_start : in std_logic;
17              tx_data_in : in std_logic_vector(7 downto 0);
18              tx_done_tick : out std_logic;
19              rx_data_out : out std_logic_vector(7 downto 0);
20              rx_done_tick : out std_logic;
21           tx : out  STD_LOGIC
22              );
23 end uart;
24 architecture str_arch of uart is
25     signal tick: std_logic;
26 begin
27     baud_gen_unit: entity work.mod_m_counter(arch)
28         generic map(M=>DVSR, N=>DVSR_BIT)
29         port map(clk=>clk, reset=>reset,
30                     q=>open, max_tick=>tick);
31     uart_rx_unit: entity work.uart_rx(arch)
32         generic map(DBIT=>DBIT, SB_TICK=>SB_TICK)
33         port map(clk=>clk, reset=>reset, rx=>rx,
34                     s_tick=>tick, rx_done_tick=>rx_done_tick,
35                     dout=>rx_data_out);
36     uart_tx_unit: entity work.uart_tx(arch)
37         generic map(DBIT=>DBIT, SB_TICK=>SB_TICK)
38         port map(clk=>clk, reset=>reset,
39                     tx_start=>tx_start,
40                     s_tick=>tick, din=>tx_data_in,
41                     tx_done_tick=>tx_done_tick, tx => tx);
42 end str_arch;
```

El siguiente código es para el receptor UART.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4      Generic (DBIT: integer:=8;          -- # bits datos
5                  SB_TICK: integer:=16);  --  # ticks
6     Port (clk : in  STD_LOGIC;
7              reset : in  STD_LOGIC;
8           rx : in  STD_LOGIC;
9           s_tick : in  STD_LOGIC;        -- senal del generador de baud rate
10          rx_done_tick : out  STD_LOGIC; -- se activa cuado se ha ...
       completado la recepcion
11          dout : out  STD_LOGIC_VECTOR (7 downto 0));
12 end uart_rx;
13 architecture arch of uart_rx is
14     type state_type is (idle, start, data, stop);
15     signal state_reg, state_next: state_type;
16     signal s_reg, s_next: unsigned(3 downto 0); -- contador ticks
17     signal n_reg, n_next: unsigned(2 downto 0); -- contador bits de datos
18     signal b_reg, b_next: std_logic_vector(7 downto 0); -- datos salida
19 begin
20     -- estado FSMD y registros de datos
21     process(clk,reset)
22     begin
23         if reset='1' then
24             state_reg ≤ idle;
25             s_reg ≤ (others=>'0');
26             n_reg ≤ (others=>'0');
27             b_reg ≤ (others=>'0');
28         elsif (clk'event and clk='1') then
29             state_reg ≤ state_next;
30             s_reg ≤ s_next;
31             n_reg ≤ n_next;
32             b_reg ≤ b_next;
33         end if;
34     end process;
35     -- logica siguiente estado y trayectoria de datos
36     process(state_reg, s_reg, n_reg, b_reg, s_tick, rx)
37     begin
38         state_next ≤ state_reg;
39         s_next ≤ s_reg;
40         n_next ≤ n_reg;
```

```
41          b_next ≤ b_reg;
42          rx_done_tick ≤ '0';
43          case state_reg is
44              when idle =>
45                  if rx='0' then
46                      state_next ≤ start;
47                      s_next ≤ (others=>'0');
48                  end if;
49              when start =>
50                  if (s_tick='1') then
51                      if s_reg=7 then
52                          state_next ≤ data;
53                          s_next ≤ (others=>'0');
54                          n_next ≤ (others=>'0');
55                      else
56                          s_next ≤ s_reg + 1;
57                      end if;
58                  end if;
59              when data =>
60                  if (s_tick='1') then
61                      if s_reg=15 then
62                          s_next ≤ (others=>'0');
63                          b_next ≤ rx & b_reg(7 downto 1);
64                          if n_reg=(DBIT-1) then
65                              state_next ≤ stop;
66                          else
67                              n_next ≤ n_reg + 1;
68                          end if;
69                      else
70                          s_next ≤ s_reg + 1;
71                      end if;
72                  end if;
73              when stop =>
74                  if (s_tick ='1') then
75                      if s_reg=(SB_TICK-1) then
76                          state_next ≤ idle;
77                          rx_done_tick ≤'1';
78                      else
79                          s_next ≤ s_reg + 1;
80                      end if;
81                  end if;
```

```
82          end case;
83      end process;
84      dout ≤ b_reg;
85  end arch;
```

El siguiente código es para el transmisor UART.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  entity uart_tx is
5      Generic(    DBIT: integer :=8;      -- # bits datos
6                  SB_TICK: integer :=16); -- # ticks
7     Port ( clk : in  STD_LOGIC;
8            reset : in  STD_LOGIC;
9            tx_start : in  STD_LOGIC;
10           s_tick : in  STD_LOGIC;
11           din : in  STD_LOGIC_VECTOR(7 downto 0);
12           tx_done_tick : out  STD_LOGIC;
13           tx : out  STD_LOGIC);
14 end uart_tx;
15 architecture arch of uart_tx is
16     type state_type is (idle, start, data, stop);
17     signal state_reg, state_next: state_type;
18     signal s_reg, s_next: unsigned(3 downto 0);
19     signal n_reg, n_next: unsigned(2 downto 0);
20     signal b_reg, b_next: std_logic_vector(7 downto 0);
21     signal tx_reg, tx_next: std_logic;
22 begin
23     -- maquina FSMD y registros de datos
24     process(clk,reset)
25     begin
26         if reset='1' then
27             state_reg ≤ idle;
28             s_reg ≤ (others=> '0');
29             n_reg ≤ (others=> '0');
30             b_reg ≤ (others=> '0');
31             tx_reg ≤ '1';
32         elsif (clk'event and clk='1') then
33             state_reg ≤ state_next;
```

```
34              s_reg ≤ s_next;
35              n_reg ≤ n_next;
36              b_reg ≤ b_next;
37              tx_reg ≤ tx_next;
38          end if;
39      end process;
40      -- siguiente estado y flujo
41      process(state_reg, s_reg, n_reg, b_reg, s_tick,
42                  tx_reg, tx_start, din)
43      begin
44          state_next ≤ state_reg;
45          s_next ≤ s_reg;
46          n_next ≤ n_reg;
47          b_next ≤ b_reg;
48          tx_next ≤ tx_reg;
49          tx_done_tick ≤ '0';
50          case state_reg is
51              when idle =>
52                  tx_next ≤ '1';
53                  if tx_start ='1' then
54                      state_next ≤ start;
55                      s_next ≤ (others =>'0');
56                      b_next ≤ din;
57                  end if;
58              when start =>
59                  tx_next ≤ '0';
60                  if (s_tick='1') then
61                      if s_reg=15 then
62                          state_next ≤ data;
63                          s_next ≤ (others=>'0');
64                          n_next ≤ (others=>'0');
65                      else
66                          s_next ≤ s_reg + 1;
67                      end if;
68                  end if;
69              when data =>
70                  tx_next ≤ b_reg(0);
71                  if (s_tick = '1') then
72                      if s_reg=15 then
73                          s_next ≤ (others=>'0');
74                          b_next ≤ '0' & b_reg(7 downto 1);
```

```
75                          if n_reg=(DBIT-1) then
76                              state_next <= stop;
77                          else
78                              n_next <= n_reg + 1;
79                          end if;
80                      else
81                          s_next <= s_reg + 1;
82                      end if;
83                  end if;
84              when stop =>
85                  tx_next <= '1';
86                  if (s_tick='1') then
87                      if s_reg=(SB_TICK-1) then
88                          state_next <= idle;
89                          tx_done_tick <= '1';
90                      else
91                          s_next <= s_reg + 1;
92                      end if;
93                  end if;
94          end case;
95      end process;
96      tx <= tx_reg;
97  end arch;
```

El contador del siguiente código se encarga de la generación del baudrate.

```
1  -- contador que cuenta de 0 hasta m-1 y comienza de nuevo
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5      Generic (N: integer := 9;       -- numero de bits N=log2M
6               M: integer := 326); -- mod-M
7      Port (clk : in  STD_LOGIC;
8            reset : in  STD_LOGIC;
9            max_tick : out  STD_LOGIC;
10           q : out  STD_LOGIC_VECTOR(N-1 downto 0));
11 end mod_m_counter;
12 architecture arch of mod_m_counter is
13     signal r_reg: unsigned(N-1 downto 0);
14     signal r_next: unsigned(N-1 downto 0);
```

```
15 begin
16     -- registro
17     process(clk,reset)
18     begin
19         if (reset='1') then
20             r_reg ≤ (others=>'0');
21         elsif (clk'event and clk='1') then
22             r_reg ≤ r_next;
23         end if;
24     end process;
25     -- logica siguiente estado
26     r_next ≤ (others=>'0') when r_reg=(M-1) else
27         r_reg + 1;
28     -- logica de salida
29     q ≤ std_logic_vector(r_reg);
30     max_tick ≤ '1' when r_reg=(M-1) else '0';
31 end arch;
```

Apéndice C

Configuración de *System Generator*

En este apéndice se muestra la configuración de la tarjeta de desarrollo Atlys para llevar a cabo *Hardware Co-Simulation*. *System Generator* cuenta con plugins para algunas tarjetas en el mercado, pero, si se tiene una tarjeta que no sea soportada o, que necesite alguna característica especial, como puertos no mapeados en memoria o non memory mapped (NMM), entonces se puede crear un plugin personalizado. El primer paso es abrir el bloque de System Generator, y en **Compilation**, escoger **Hardware Co-Simulation** y luego **New Compilation Target**.

Se completan los campos como aparece en la Figura C.2 y después de completar **Target Board Information**, **System Clock** y **JTAG Options**, se oprime el botón **Detect**. La tarjeta deberá estar conectada y encendida. Luego se agrega el dispositivo en **Targetable Devices**.

Una vez que se ha reconocido exitosamente la tarjeta, se agregan en **Non-Memory-Mapped Ports**, los pines de la tarjeta que se vayan a utilizar, como se muestra en las Figuras C.3 y C.4.

Cuando todas las configuraciones estén hechas y se hayan agregado todos los pines, en este ejemplo se muestran dos, Figura C.5, se presiona el botón **Install** para generar el plugin.

Saldrá un aviso de que la instalación del plugin ya se ha llevado a cabo, Figura C.6.

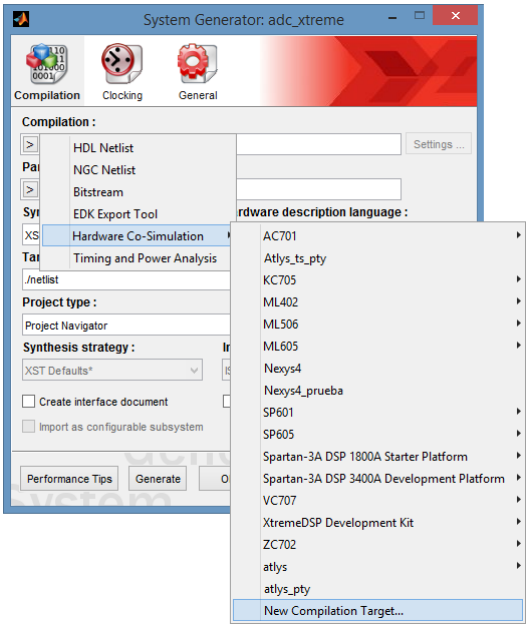


FIGURA C.1: Menú *Compilation* del bloque *System Generator*.

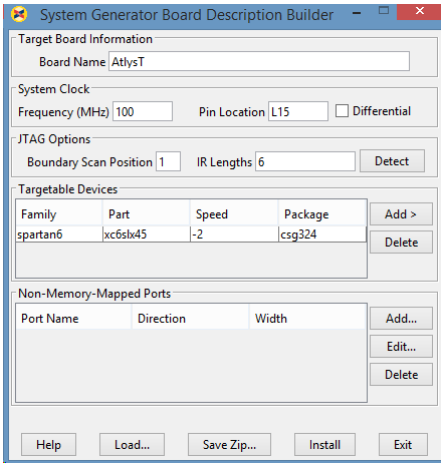


FIGURA C.2: Campos en *Board Description Builder*.

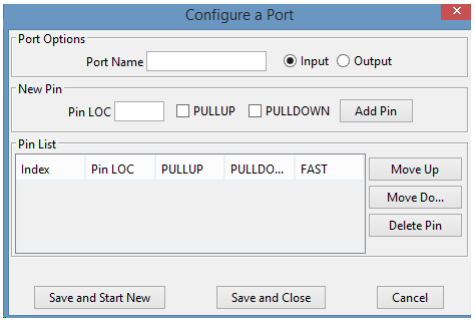


FIGURA C.3: Menú al desplegar **Add...** en **Non-Memory-Mapped Ports**.

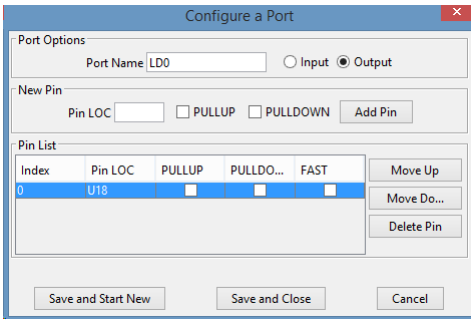


FIGURA C.4: Ejemplo agregando una salida.

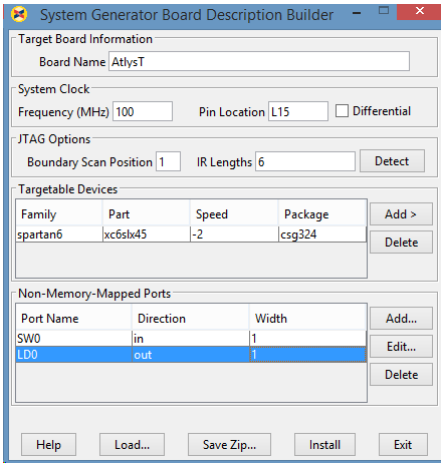


FIGURA C.5: Descripción en *Board Description Builder* completa.

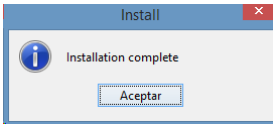


FIGURA C.6: Aviso de instalación completa.

Cuando el proceso se haya completado se abrirá la librería personalizada del plugin, donde se mostrarán los bloques creados para los pines que agregamos como NMM ports, Figura C.7.

Ahora, cuando en el bloque de System Generator escojamos **Hardware Co-Simulation** aparecerá el nombre que le dimos a nuestra tarjeta, y podremos usar los bloques de la librería personalizada, Figuras C.8 y C.9.

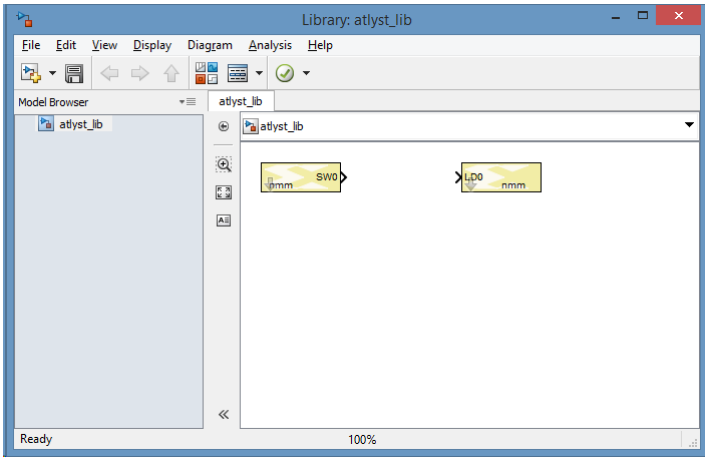


FIGURA C.7: Librería con NMM ports.

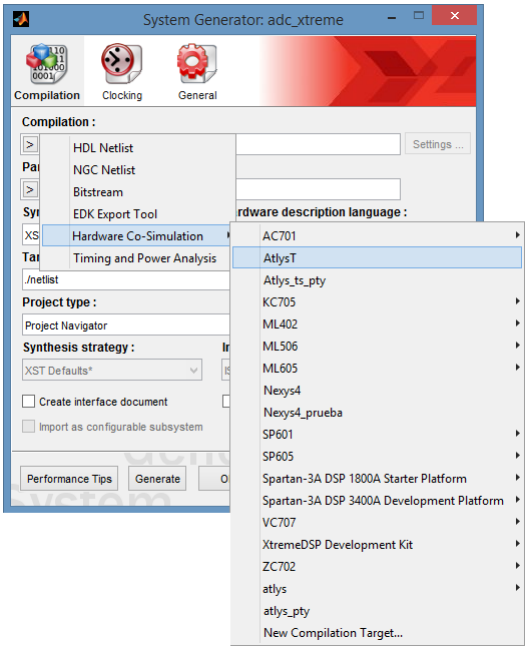


FIGURA C.8: Plugin instalado correctamente.

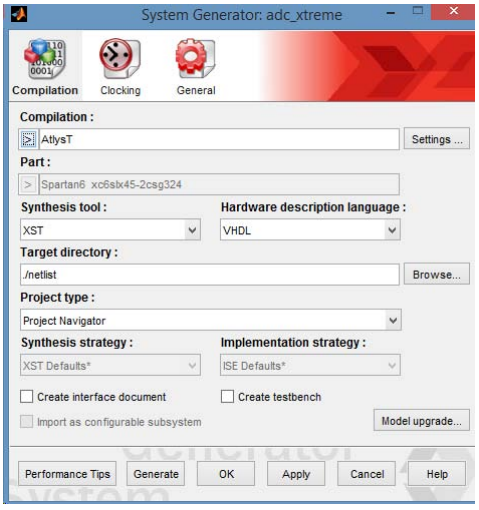


FIGURA C.9: Uso del plugin para hardware co-simulation.

Bibliografía

[1] Jeffery Alan Wepman and Randy Hoffman. RF and IF Digitization in Radio Receivers: Theory, Concepts and Examples. Technical report, U. S. DEPARTMENT OF COMMERCE, 1996.

[2] Jose Angel Amador Fundora and Nestor Alonso Torres. RDS (Radio Definido por Software) consideraciones para su implementación en hardware. *Revista Telem@tica*, 12(2), mayo-agosto 2013.

[3] Alexander Galvis Quintero, Christian A. Ceballos Betancour, and Lukas De Sanctis Gil. SDR: La alternativa para la evolución inalámbrica a nivel físico. *Conference and Workshop*, 2006.

[4] Iván Pinar Domínguez and Juan José Murillo Fuentes. *Laboratorio de Comunicaciones Digitales Radio Definida por Software*. Universidad de Sevilla, 1 edition, 2011.

[5] Firas Abbas Hamza. The USRP under 1.5x Magnifying Lens! 2008.

[6] AD6620, 67 MSPS Digital Receive Signal Processor. Analog Devices, 2001.

[7] Angsuman Rudra. FPGA-based applications for software radio, 2004. URL http://mobiledevdesign.com/news/fpga-based-applications-software-radio. [Fecha de consulta: Junio 2015].

[8] Cecil Accetti R. de A. Melo and Ricardo E. de Souza. FPGA-based Digital Direct-Conversion Transceiver for Nuclear Magnetic Resonance Systems. *Integrated Circuits and Systems Design*, pages 1–5, 2012.

[9] Zhengmin Liu, Cong Zhao, Heqin Zhou, and Huanquing Feng. A Novel Digital Magnetic Resonance Imaging Spectrometer. In *28th IEEE EMBS Annual international Conference*, 2006.

[10] Kazuyuki Takaeda. OPENCORE NMR: Open-source core modules for implementing an integrated FPGA-based NMR spectrometer. *Journal of Magnetic Resonance*, 2008.

[11] Li Sun, Joshua J. Savory, and Kurt Warncke. Design and implementation of an FPGA-Based Timing Pulse Programmer for Pulsed-Electron Paramagnetric Resonance Applications. *Concepts in Magnetic Resonance. Part B, Magnetic Resonance Engineering*, 2014.

[12] Weinan Tang and Weimin Wang. A single-board NMR spectrometer based on a software defined radio architecture. *Measurement Science and Technology*, 2010.

[13] Solomon Lule Workneh, Raghavendra, Vuda Sreenivasarao, and Babu Reddy. Advanced Receiver Architectures in Radio - Frequency Applications. *Global Journal of Researches in Engineering Electrical and Electronics Engineering*, 13, 2013.

[14] Adad Abidi. Direct-Conversion Radio Transceivers for Digital Communication. *IEEE Journal of Solid-State Circuits*, 30(12), 1995.

[15] Behzad Razavi. RF Trasmitter Architectures and Circuits. In *Custom integrated circuits conference*, 1999.

[16] Sean Gallagher. Systems DSP algorithms into FPGAs, November 2010.

[17] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, 3 edition, 2007.

[18] Hiroshi Yoshioka, Philipp Schlechtweg, and Katsumi Kose. *Magnetic Resonance Imaging*. 2003.

[19] Vincent Chan and Anahi Perlas. *Basics of Ultrasound Imaging*. Springer, 2011.

[20] Masayuki Tanabe, editor. *Ultrasound Imaging*. InTech, 2011.

[21] Kazuyuki Takaeda. A highly integrated FPGA-based nuclear magnetic resonance spectrometer. *Journal of Applied P*, 2007.

[22] Gabriel Augusto Zebadúa García. Diseño de un receptor digital de señales de radiofrecuencia mediante un FPGA. 2012.

[23] Rodger H. Hosking. *Digital Receiver Handbook: Basics of Software Radio*. Pentek, 5 edition, 2006.

[24] Rodger H. Hosking. *Software Defined Radio Handbook*. Pentek, 11 edition, 2013.

[25] E. Oran Brigham. *The fast fourier transform and its applications*. Pretince Hall, 1988.

[26] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, 3 edition, 2009.

[27] Pere Martí I Puig. *Los sistemas de comunicaciones digitales*. Universitat Oberta de Catalunya.

[28] Rinck P. Magnetic Resonance in Medicine. The Basic Textbook of the European Magnetic Resonance Forum., July 2013. URL www.magnetic-resonance.org. [Fecha de consulta: Agosto 2015].

[29] Angélica Vargas, Luis Amescua, Araceli Bernal, and Carlos Pineda. Principios físicos básicos del ultrasonido, sonoanatomía del sistema musculoesquelético y artefactos ecográficos. *Medigraphic*, 2008.

[30] Murtaza Ali, Dave Magee, and Udayan Dasgupta. Signal processing overview of ultrasound systems for medical imaging. Texas Instruments, 2008.

[31] Walt Kester. Oversampling interpolating dacs. Analog Devices, 2009.

[32] Thomas L. Floyd. *Fundamentos de sistemas digitales*. Pearson, 9 edition, 2006.

[33] DSP Design Flow using System Generator. Xilinx, 2012. Course.

[34] Robert Joachim Schweers. Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC. 2002.

[35] Dan McLane and Billy Kao. *FrameWork Logic and MATLAB Board Support Package*. Innovative Integration, 2007.

[36] Walt Kester, editor. *Mixed-signal and DSP Design Techniques*. Analog Devices, 2003.

[37] Chris Pearson. High-Speed, Analog-to-Digital Converter Basics. Texas Instruments, January 2011.

[38] The Impact of Clock Generator Performance on Data Converters. Hittite Microwave Corporation, 2012. URL https://www.hittite.com/literature/clocks_timing_web.pdf. [Fecha de consulta: Junio 2015].

[39] AD6644, 14-Bit, 65 MSPS Analog-to-Digital Converter. Analog Devices, 2007. Rev.

[40] AD9755, 14-Bit, 300 MSPS High Speed Digital-to-Analog Converter. Analog Devices, 2003.

[41] Michael Parker. Architecture and Component Selection for SDR Applications. Altera, June 2007.

[42] Helen Tarn, Kevin Neilson, Ramon Uribe, and David Hawke. Designing Efficient Wireless Digital Up and Down Converters Leveraging CORE Generator and System Generator. Xilinx, October 2007. Application Note: Virtex-5, Spartan-DSP FPGAs.

[43] Stephen Creaney and Igor Kostarnov. Designing Efficient Digital Up and Down Converters for Narrowband Systems. Xilinx, Noviembre 2008. Application Note: Virtex-5 Family.

[44] LogiCORE IP DDS Compiler. Xilinx, March 2011. DS558.

[45] LogiCORE IP CIC Compiler. Xilinx, March 2011.

[46] LogiCORE IP Block Memory Generator. Xilinx, December 2012. PG058.

[47] LogiCORE IP Clocking Wizard. Xilinx, July 2012.

[48] Spartan-6 FPGA Clocking Resources. Xilinx, Jue 2015. UG382.

[49] Spartan-6 Libraries Guide for HDL Designs. Xilinx, January 2012. UG615.

[50] Pong P. Chu. *FPGA Prototyping by VHDL examples.* John Wiley & Sons, 2008.

[51] 8648A Signal Generator Operation and Service Guide. Angilent Technologies, March 2001.

[52] Dennis M. Akos. *A software radio approach to global navigation satellite system receiver design.* PhD thesis, College of Engineering and Technology, Ohio University, August 1997.