UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE UN ESPECTRÓMETRO BASADO EN FPGA, DE ANCHO DE BANDA SELECCIONABLE PARA APLICACIONES ASTRONÓMICAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA

EDGARDO ANTONIO HUARACÁN DURÁN

PROFESOR GUÍA:

RICARDO ALBERTO FINGER CAMUS

MIEMBROS DE LA COMISIÓN:

FAUSTO PATRICIO MENA MENA

JORGE FELIPE SILVA SANCHEZ

Este trabajo fue realizado gracias al apoyo de CONICYT a través del Centro de Astrofísica y Tecnologías Afines (PBF 06), ALMA-CONICYT 31120005, Gemini-CONICYT 32120004 y FONDECYT 11140428.
Agradecemos a Xilinx Inc. Por la donación de circuitos integrados y licencias de software.

SANTIAGO DE CHILE

2014

RESUMEN DE LA MEMORIA

PARA OPTAR AL TÍTULO DE

INGENIERO CIVIL ELECTRICISTA

POR: EDGARDO HUARACÁN DURÁN

FECHA: 8 DE SEPTIEMBRE DE 2014

PROF. GUIA: Sr. RICARDO FINGER CAMUS

"DISEÑO E IMPLEMENTACIÓN DE UN ESPECTRÓMETRO BASADO EN FPGA, DE ANCHO DE BANDA SELECCIONABLE PARA APLICACIONES ASTRONÓMICAS"

El auge que ha presentado en los últimos años la Astronomía observacional en Chile, producto de las inigualables condiciones ambientales del desierto de Atacama, ha generado la necesidad de desarrollar tecnologías afines con este gran desafío. Este trabajo propone soluciones en el área de procesamiento digital de señales haciendo uso de procesadores de alta velocidad. En concreto, se diseña e implementa un espectrómetro de ancho de banda seleccionable para aplicaciones en espectrometría de alta resolución como lo es el estudio de estructuras moleculares híper finas. Además se diseña un bloque de control de datos, útil entre otras cosas para mejorar las mediciones de rechazo de banda lateral.

El uso de procesadores FPGA (*Field Programmable Gate Array*) permite implementar espectrómetros digitales. Estos diseños son aplicables en procesadores cada vez más veloces. Existen arquitecturas prediseñadas para estos propósitos como por ejemplo la plataforma ROACH (*Reconfigurale Open Architecture Computing Hardware)* creada por CASPER (*Collaboration for Astronomy Signal Processing and Electronic Research*) y que se utiliza en este trabajo de título.

Utilizando diseños en ambiente MATLAB Simulink se crea un ejecutable que da instrucciones al FPGA. Así, estos diseños son controlados y comandados por códigos en lenguaje Python, los que realizan un post procesamiento de los datos para su posterior análisis.

Los diseños creados para mejorar el rechazo de banda lateral demuestran ser efectivos limitando la contaminación de banda adyacente a menos de -50dB. El bloque de control asegura el buen funcionamiento del sistema, incluso en presencia de alto ruido de fase. Por otro lado al realizar acercamiento a una porción del ancho de banda se logra una mejor resolución espectral logrando distinguir variaciones de energía en rangos pequeños de frecuencia.

A futuro, todos los diseños creados en este trabajo pueden implementarse a mayor velocidad, utilizando procesadores más avanzados. Es posible enriquecer el zoom, reduciendo el ancho de banda desde dos hasta ocho veces, y utilizar mezcladores digitales para seleccionar una zona arbitraria del espectro.

Agradecimientos

Quiero agradecer a mi familia por el importante apoyo que me han brindado en todos los aspectos.

Agradecimientos para los miembros del Laboratorio de Ondas Milimétricas, en especial a Ricardo Finger y Leonardo Bronfman por darme la oportunidad de ser parte de este proyecto. A la comunidad de CASPER por su disposición a ayudar, y a Xilinx por la donación de licencias.

Tabla de contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

El gran incremento en la velocidad de procesadores de última generación en los últimos años, ha hecho posible el procesamiento digital de señales con un ancho de banda considerable en tiempo real. Esto es especialmente interesante si se trata de procesamiento de señales radioastronómicas.

Por otra parte el gran auge de la Astronomía en Chile, con proyectos como el VLT (*Very Large Telescope*) en cerro paranal o ALMA (*Atacama Large Millimeter-sub millimeter Array*) en el llano de Chajnantor, y la llegada de nuevos súper proyectos astronómicos posicionan a Chile como la capital mundial de la Astronomía observacional.

Todo lo anterior evidencia la necesidad de realizar investigación y desarrollo en lo que respecta a tecnologías de punta aplicadas a estos proyectos. Es por esto que se han creado incentivos desde distintos organismos para la innovación en instrumentación astronómica. Es en este contexto en donde se posiciona este trabajo de título, en particular aportando directamente a mejorar la etapa digital de receptores que se utilizan típicamente en radioastronomía, agregando características nuevas y aprovechando todos los recursos disponibles.

1.2. Alcances y Objetivos

En términos generales este proyecto pretende entregar herramientas de ingeniería que permitan manejar los datos captados por un receptor de una manera más apropiada. En particular se diseñará un circuito que pretende mejorar el rechazo de banda lateral y otro que permita realizar un acercamiento a distintas líneas en un espectro aumentando la resolución espectral.

Los resultados expuestos en este trabajo servirán para enriquecer las características actuales del telescopio Mini en el Cerro Calán, pero en general son mejoras aplicables a cualquier radiotelescopio con etapa digital programable.

1.3. Estructura

La estructura de este documento es la siguiente:

Capítulo 2. Marco Teórico: Aquí se exponen los principales conceptos teóricos que se requieren para entender el trabajo realizado.

Capítulo 3. Objetivos y Metodología: Es donde se entrega una descripción del ambiente de trabajo, los distintos instrumentos y software utilizados. También se explica con detalle qué es lo que se intenta realizar en este proyecto y cómo se pretende lograr los objetivos.

Capítulo 4. Implementación: En este capítulo se detallan todas las etapas de diseño de los distintos circuitos a realizar, desde las especificaciones de diseño, hasta el diseño final y detalles de funcionamiento. También se explican las rutinas que se realizan para tomar los datos con los sistemas diseñados.

Capítulo 5. Resultados: Se muestran los datos tomados con los nuevos diseños y se analizan las mejoras producidas.

Capítulo 6. Conclusiones: Se entregan conclusiones generales respecto al trabajo realizado, y se discuten posibles mejoras a los diseños.

Capítulo 2

Contextualización

En este capítulo se revisarán los conceptos previos para poder entender e interiorizase en el tema central de esta memoria.

2.1. La Radioastronomía

2.1.1. Una breve reseña histórica

Antiguamente la única forma en que los astrónomos recopilaban datos, era observando visualmente el cielo, fijándose en los colores, brillo o forma de los objetos. Sin embargo, a mediados del siglo XIX el físico teórico escocés James Clerk Maxwell (1831-1879) revolucionó la física con su teoría del electromagnetismo basado en experimentos previos realizados por Michael Faraday (1791-1867). Maxwell formalizó matemáticamente estos resultados llegando a un conjunto de ecuaciones que le permitieron concluir que un cambio en el campo eléctrico producía un campo magnético y viceversa, pero además se dio cuenta que estos campos se propagaban en forma de onda perpendiculares entre sí y además oscilaban perpendicularmente con la dirección de propagación. Con estas ecuaciones pudo también concluir que la velocidad de propagación de estas ondas era de 3.0×10^8 [m/s] lo cual considerando errores experimentales, era igual a la velocidad de la luz. Con estos indicios era fácil sospechar que la luz era una onda electromagnética. Sin embargo esta teoría no fue completamente aceptada hasta varios años después de su descubrimiento.

Heinrich Hertz (1857-1894) demostró la teoría de Maxwell ya que en 1888 produjo ondas electromagnéticas en el laboratorio, de frecuencias mucho más baja que la luz (5 metros de longitud de onda). El mundo científico comenzó a interesarse por esta clase de ondas invisibles ya que con ellas podrían diseñar sistemas de inalámbricos. Sin duda un gran salto considerando que en esa época se utilizaba el telégrafo como principal medio de comunicación a larga distancia.

No pasó mucho tiempo para que se comenzara a desarrollar tecnología para procesar estas ondas. En 1932 un joven ingeniero eléctrico llamado Karl Guthe Jansky (1905-1950) que trabajaba para *Bell Telephone* realizaba un estudio de fuentes de interferencia para comunicaciones transatlánticas [1]. Para esto Jansky construyó una antena de 30 metros de longitud y 4 metros de alto que operaba a 20,5 MHz. Sin quererlo captó una señal que volvía a aparecer cada 23 horas y 56 minutos, lo que corresponde al día sideral. A finales de ese mismo año Jansky concluyó correctamente que esa señal de radio provenía del centro de la vía láctea. Este descubrimiento abrió camino a una nueva disciplina, la radioastronomía. Pese a que los resultados de Jansky no fueron tomados demasiado en cuenta por la comunidad de

astrónomos (ya que estaban centrados en la astronomía óptica), fue cosa de tiempo para que un ingeniero eléctrico norteamericano llamado Grote Reber (1911-2002) tomara en cuenta los resultados de Jansky y construyera su propia antena parabólica de 10 metros de diámetro para apuntarla especialmente al cielo. Esta fue la primera antena radioastronómica y Reber se convirtió en el primer radio astrónomo de la historia (de hecho fue el único durante diez años en el mundo entero). Con el descubrimiento de la radiación en 21 centímetros (1420 MHz) del Hidrogeno, se demostró empíricamente que un gas podía emitir en ondas de radio con lo cual la radioastronomía pasó a ser una sub área indispensable de la astronomía. Hoy en día la radioastronomía es responsable de al menos 3 grandes descubrimientos importantes: los cuásares, la radiación de fondo cósmico y los pulsares [2].

2.1.2. Radioastronomía desde la tierra

Las ondas de radio (desde $\lambda = 20\,\text{m}$ hasta $\lambda = 0{,}2\,\text{mm}$) son especialmente interesantes para la astronomía desde la tierra, ya que la atmosfera es casi transparente para este tipo de radiación [3], tal como se muestra en la figura 2.1. La frecuencia superior de corte de esta banda es producida por la presencia de las moléculas de H_2O y O_2 en la atmosfera. Para evitar la absorción de radiación producida por estas moléculas, se suelen construir radiotelescopios a gran altura y en lugares secos.

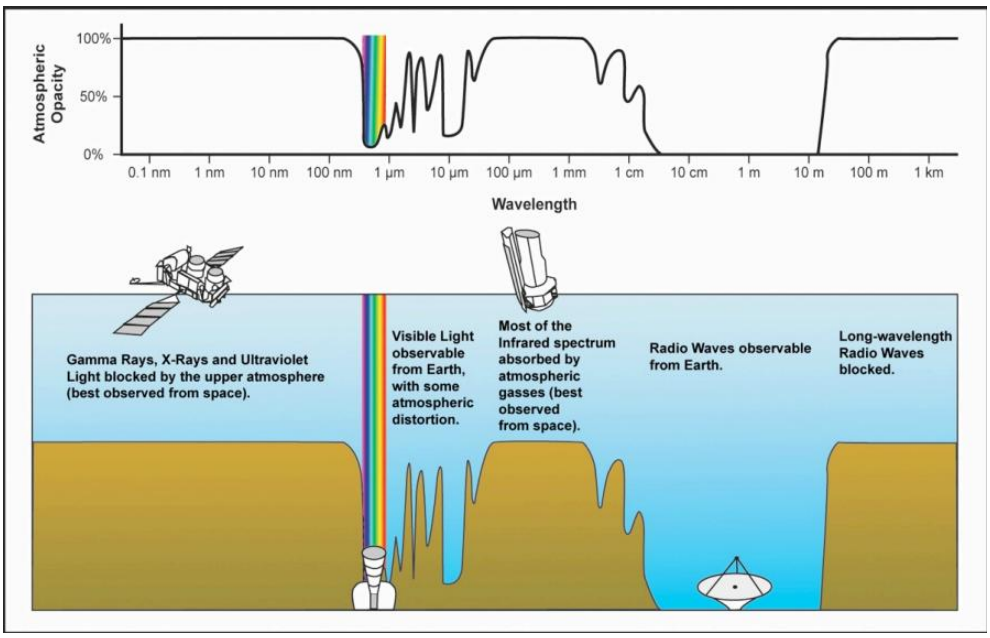


Figura 2.1: Opacidad de la Atmosfera terrestre a altitudes cercanas al nivel del mar en función de la longitud de onda de la señal proveniente del espacio exterior: [4].

Se observan pequeñas ventanas atmosféricas en el óptico y el infrarrojo. La señal de radio frecuencia es por lo general de muy baja potencia, es por esto que es muy vulnerable a ruidos propios de los receptores o componentes electrónicos diseñados para el procesamiento. Por esta razón estos instrumentos deben estar especialmente diseñados para evitar contaminaciones en la señal de interés [5].

El área de la radio astronomía ha tenido grandes avances en los últimos años, y se han construido numerosos radiotelescopios de distinta envergadura. Sin ir más lejos, a 5000 msnm en el desierto Chileno de Atacama se ha instalado el observatorio radioastronómico ALMA (*Atacama Large Millimeter and Sub-Millimeter Array*). Este observatorio es el más grande del mundo, y consta de 66 antenas de radio con discos de 12 y 7 metros de diámetro, con 10 bandas de observación, conformando de esta manera un gran ojo gigante de señales de longitud de onda milimétrica [6].

Un ejemplo ilustrativo de la aplicación de la radioastronomía es el estudio de la emisión de monóxido de carbono relacionado con la formación estelar. Intentar entender la formación de estrellas sin conocer el estado físico del denso gas molecular interestelar del que las estrellas están hechas es casi una tarea imposible [7]. Así es como por ejemplo se utilizó el radiotelescopio SMWT (1.2 m *Southern Millimeter-Wave Telescope*) más conocido como Mini, para realizar un mapeo completo de la emisión del ^{12}CO en la vía láctea [8].

2.2. El receptor heterodino

El tipo de receptor típicamente utilizado en radioastronomía, es muy similar a los usados en telecomunicaciones. Estos se llaman receptores heterodinos. Captan una señal llamada RF (*Radio Frequency*) y la convierten a una frecuencia mucho más baja llamada IF (*Intermediate Frequency*), manteniendo sus características de amplitud y fase. En la figura 2.2 se muestra la estructura general de este tipo de receptores también conocido como DSB (*Double Side Band*) [9].

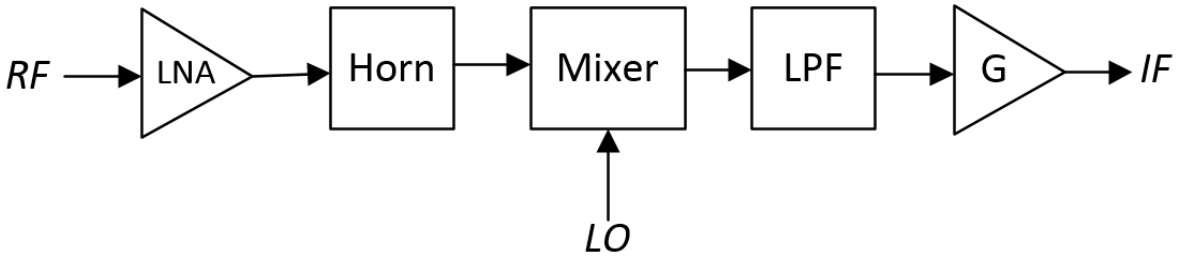


Figura 2.2: Diagrama de un Receptor Heterodino con todas sus componentes. La señal RF que entra por la antena es multiplicada por un oscilador local LO para bajar la frecuencia de RF a IF y así poder procesarla digitalmente.

Este tipo de configuración consta de una antena receptora llamada *horn* (bocina), especialmente diseñada para captar señales de alta frecuencia, que se encarga de trasformar la señal a una señal de voltaje y corriente, que por lo general es de muy baja potencia. Para trabajar con una señal de baja potencia, es necesario tener un elemento que amplifique la señal adhiriendo el menor ruido posible. Este elemento se llama amplificador de bajo ruido o LNA (*Low Noise Amplifier)* el cual se conecta a la antena [5]. Luego de amplificada, la señal se mezcla con un tono puro usualmente llamado oscilador local (LO: *Local Oscilator*) mediante un elemento no lineal llamado mezclador o *mixer*, que básicamente se encarga de entregar en su puerto de salida la multiplicación entre LO y RF en el dominio del tiempo. El mezclador al ser un elemento no lineal (basado en diodos) produce armónicos y productos de intermodulación los cuales deben ser filtrados. De esta manera el espectro en el espacio de Fourier, de la señal que sale del *mixer* es el mismo de la señal RF pero duplicado (convertido hacia arriba y hacia abajo). Lo anterior es evidente al considerar el análisis en el dominio de la frecuencia para un mezclador que realiza una multiplicación ideal (sin productos de intermodulación), como en la siguiente igualdad:

$$\mathfrak{F}[RF(t) \times \cos(2\pi\, f_{LO} t)] = \frac{1}{2}[\mathfrak{F}_{RF}(f - f_{LO}) + \mathfrak{F}_{RF}(f + f_{LO})] \qquad (2.1)$$

En (2.1) $\mathfrak{F}$ es el símbolo empleado para la transformada de Fourier, f_{LO} la frecuencia del oscilador local y $\mathfrak{F}_{RF}(f)$ la transformada de Fourier de $RF(t)$. En resumen la mitad de la energía de la señal RF se concentra en la frecuencia $f - f_{LO}$, mientras que la otra mitad se concentra en la frecuencia $f + f_{LO}$ como lo indica la ecuación (2.1). Luego $\mathfrak{F}_{RF}(f - f_{LO})$, correspondiente a la porción convertida hacia arriba, es filtrada por un filtro pasa bajo (LPF: *Low Pass Filter*) como se muestra en la figura 2.2, quedando solo la señal $\mathfrak{F}_{RF}(f + f_{LO})$ que corresponde a la señal IF. Este proceso se denomina *conversión hacia abajo* puesto que se baja la frecuencia de la señal RF, manteniendo su espectro [6].

2.3. El radiotelescopio

El radiotelescopio es un sistema receptor-antena, especialmente diseñado para captar y procesar ondas electromagnéticas de baja potencia venidas del espacio, especialmente en la frecuencia de las ondas de radio. Su diseño está focalizado a intentar contaminar lo menor posible la señal de radiofrecuencia [5]. Está normalmente conformado por un espejo parabólico capaz de reunir la radiación en un punto (foco). Existen distintos tipos de antenas parabólicas reflectoras, sin embargo en esta sección la discusión se centra en el reflector Cassegrain clásico. Este último es uno de los más utilizados, puesto que presenta mejor relación focal que otros como por ejemplo el reflector de foco primario [10]. Como se muestra en la figura 2.3, este receptor consta de un espejo primario de superficie parabólica que refleja la luz hacia un espejo llamado secundario y de geometría hiperbólica, que a su

vez se encarga de colectar la luz en el foco [11], donde se sitúa la antena de tipo *horn* para el posterior procesamiento de la señal. A todo el procesamiento analógico se le denomina *front end*.

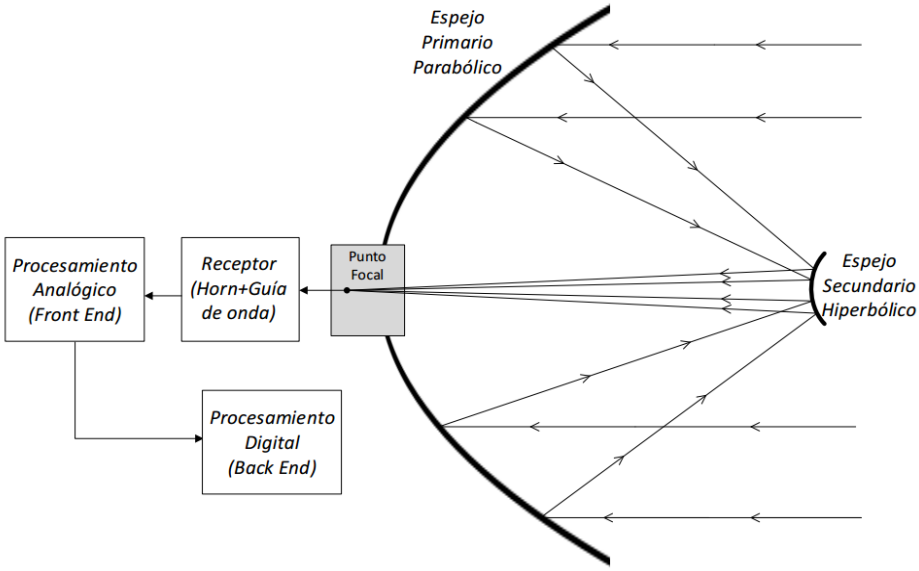


Figura 2.3: Principio de funcionamiento de un reflector Cassegrain.

Después de amplificarse, la señal es pasada a través de un receptor heterodino situado en la etapa de procesamiento analógico, el que se encarga de bajar la frecuencia de la señal para así poder digitalizarla y realizar el posterior procesamiento por medio del *back end*.

En la figura 2.4 se aprecia el radiotelescopio *Effelsberg*, uno de los telescopios más modernos existentes. Pese a que comenzó sus operaciones en 1972, se ha ido mejorando continuamente con tecnología de punta. Este telescopio es empleado entre otras cosas para observar pulsares, nubes moleculares y materia expulsada por agujeros negros [12].

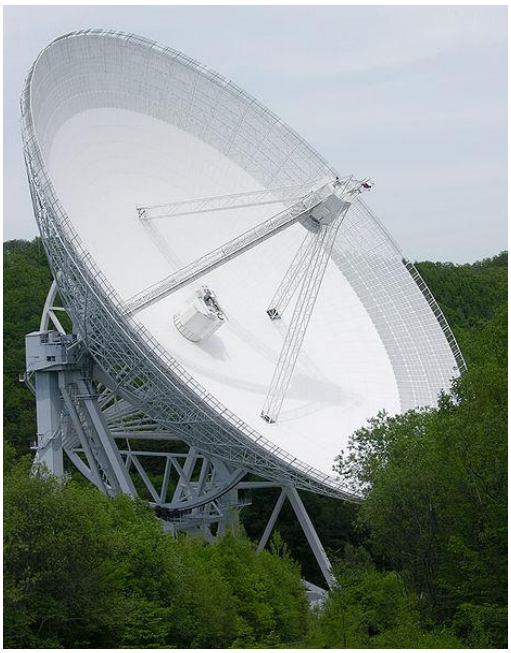


Figura 2.4: Fotografía del Radio Telescopio Alemán Effelsberg de 100 metros de diámetro. [http://en.wikipedia.org/wiki/Effelsberg_100-m_Radio_Telescope, Julio de 2014]]

2.4. Separación de banda lateral

Cuando la señal RF pasa por un receptor heterodino, esta se convierte hacia abajo. El problema que se presenta en el receptor DSB es que al pasar por el proceso de conversión hacia abajo ocurrirá que la porción LSB (*Lower Side Band*) se reflejará sobre la porción USB (*Upper Side Band*) como lo muestra la figura 2.5 [9], siendo LSB y USB las bandas laterales o porciones del espectro de RF que están por debajo y por arriba del oscilador local LO respectivamente. De esta manera no es posible distinguir en la banda IF a priori cuál es el LSB ni el USB. Para separar las bandas en la señal IF se aplica una técnica llamada separación de banda lateral [13] [6].

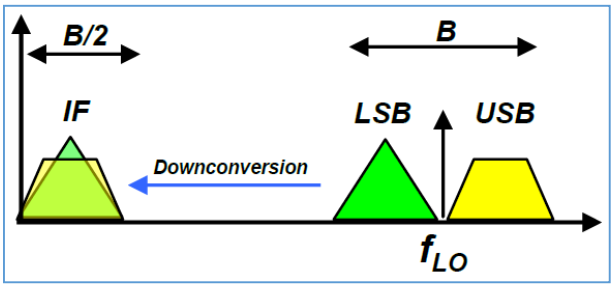


Figura 2.5: Gráfico que ilustra el proceso llamado conversión hacia abajo. Al convertirse hacia abajo, la porción LSB queda traslapada con la USB en la banda de IF: [9].

La figura 2.6 muestra el procedimiento seguido para separar las bandas laterales en una configuración denominada mezclador separador de banda lateral o SSM (*Sideband*

Separating Mixer). La señal RF es dividida en dos rutas. En una ruta la señal es desfasada en 90°, mientras que por la otra se mantiene con desfase de 0°. Esto es posible con un elemento lineal llamado Hibrido RF. Luego ambas rutas son mezcladas con el oscilador local mediante mezcladores tal como se explicó anteriormente. El último paso es volver a desfasar en 90° por una ruta, mantener por la otra y luego sumar. Este último paso se realiza en el Hibrido IF.

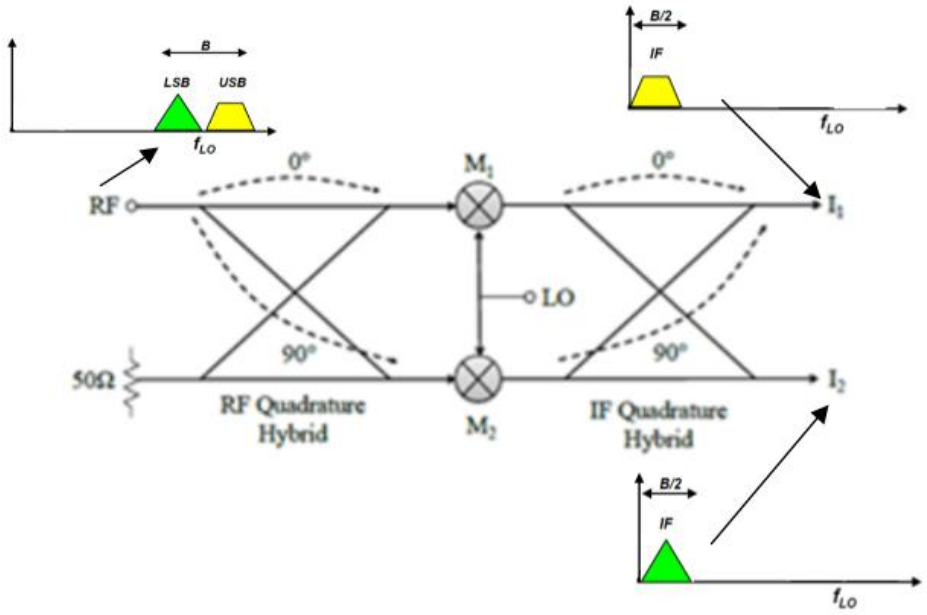


Figura 2.6: Configuración SSM utilizada para realizar separación de las bandas laterales. Esta consta de una etapa hibrida RF, dos Mixers y otra etapa Hibrida IF: [9].

Una forma cuantitativa de medir la separación de banda lateral o rechazo de imagen (*Image Rejection)* está dada por la relación de rechazo o SRR (*Sideband Rejection Ratio*). En la figura 2.7 se muestra la misma configuración SSM de la figura 2.6, pero se muestran las ganancias de cada línea. La entrada RF USB y LSB son físicamente la misma guía de onda, sin embargo por simplicidad se muestran como dos entradas distintas.

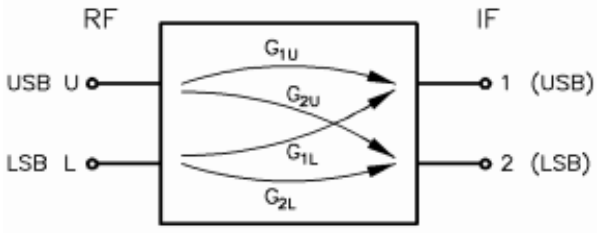


Figura 2.7: Ganancias de la distintas rutas en el receptor SSM. Los dos puertos de entrada RF son físicamente la misma guía de onda [14].

Se definen los rechazos de banda lateral SRR_1 y SRR_2 en los puertos de salida IF 1 y 2, de la figura 2.7, respectivamente como [14]

$$SRR_1 = \frac{G_{1U}}{G_{1L}} \tag{2.2}$$

$$SRR_2 = \frac{G_{2L}}{G_{2U}}. \tag{2.3}$$

A su vez el máximo rechazo de imagen teórico que se puede lograr está dado por

$$SRR = -10\log\left(\frac{1 + A^2 - 2A\cos(\theta)}{1 + A^2 + 2A\cos(\theta)}\right), \tag{2.4}$$

donde A es la suma de los desbalances de amplitud en los híbridos RF e IF y en los *mixers* de la figura 2.6, mientras que el factor θ representa el desbalance total de fase debido a la suma de los desbalances individuales de las cuadraturas en los híbridos RF e IF y el desbalance de fase de los dos *mixers* [15]. En la figura 2.8 se pueden apreciar distintos rechazos de imagen alcanzables para distintos desbalances de fase en función del desbalance de amplitud.

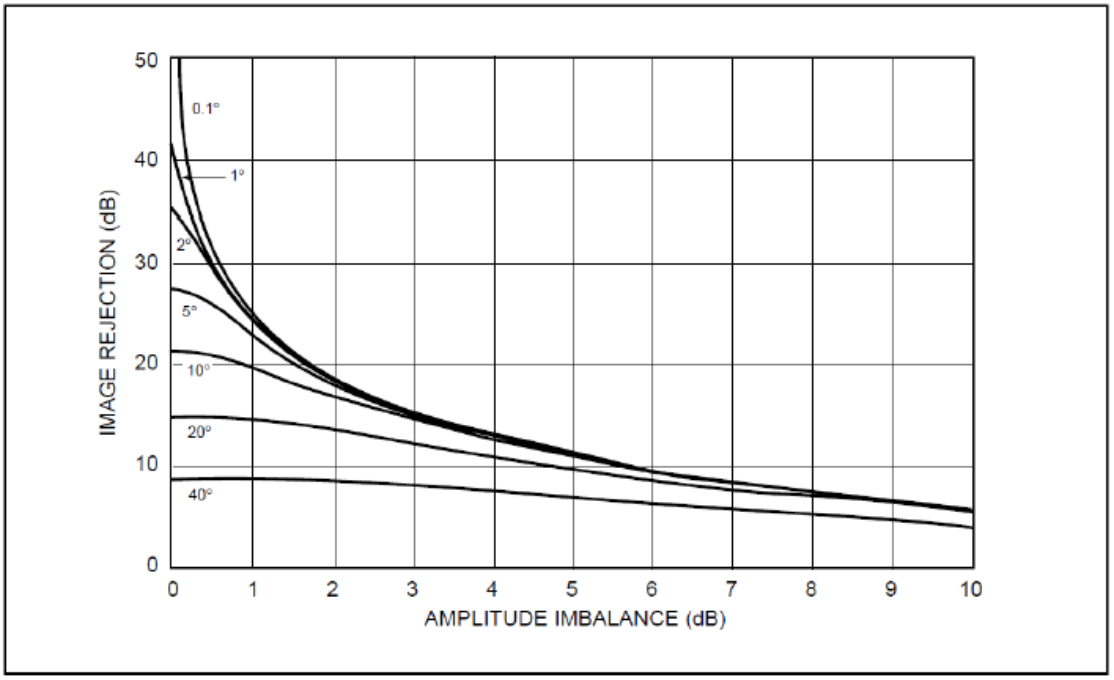


Figura 2.8: Máximo SRR alcanzable en función del desbalance de amplitud y de fase. Se puede observar que el rechazo es mayor cuando los desbalances de amplitud y/o de fase son bajos [15].

2.5. Procesamiento digital de señales

2.5.1. Teorema de muestreo de Shannon & Nyquist

El teorema expuesto en esta sección es de gran importancia para el procesamiento digital de señales ya que establece las restricciones necesarias para garantizar una correspondencia única entre las señales analógicas y sus respectivas señales discretas. De esta manera se puede realizar el procesamiento digital de una señal en un computador para luego reconstruir la señal analógica sin (o con la menor) perdida de información. Este teorema fue demostrado matemáticamente por Claude E. Shannon en 1949, sin embargo fue propuesto por Harry Nyquist en 1928, por esa razón el teorema es identificado con el nombre de ambos [16].

Las señales son en gran parte de naturaleza continua o analógica. Es decir existe un valor determinado de la señal para cualquier instante de tiempo. Esto implica que para procesar una señal continua $x_a(t)$ en un computador, se necesitaría infinita memoria, incluso si se analiza la señal en un intervalo acotado de tiempo. Entonces es necesario tomar muestras de la señal en el intervalo de tiempo en que esta se desea analizar. Estas muestras se toman en general cada cierto tiempo T, que se conoce como tiempo de muestreo, generando así una señal discreta $x(n)$ como sigue:

$$x(n) = x_a(nT) \quad \forall\, n = 0,1 \dots, N-1 \tag{2.5}$$

donde N es el número de muestras de la señal analógica $x_a(t)$.

Para realizar el análisis de $x_a(t)$ con sólo algunas muestras de ésta, primero es necesario asegurar que pueda reconstruirse sin ambigüedad a partir de sus muestras. El teorema de Nyquist indica la relación entre tiempo de muestreo y la frecuencia de la señal que asegura que la señal analógica se pueda reconstruir unívocamente. En concreto, una señal $x_a(t)$ de banda base, con un ancho de banda BW puede ser reconstruida a partir de sus muestras tomadas con una frecuencia de muestreo F_s si se cumple la siguiente desigualdad.

$$F_s \geq 2BW \tag{2.6}$$

En otras palabras si F_s es menor que el doble del ancho de banda de $x_a(t)$, la señal analógica reconstruida a partir de las muestras $x(n)$ será distinta de $x_a(t)$. Este efecto de ambigüedad se conoce como efecto *aliasing*, y en algunos casos podría hacer perder información. Por ejemplo si una señal de banda base se muestrea a una tasa menor que el doble de su frecuencia de muestreo, al procesarla en un computador y luego reconstruirla, habrá ruido adherido a la señal. En estos casos hay que evitar este efecto. Sin embargo en otros casos produce una

conversión hacia abajo de la señal sin pérdida de información en cuyo caso su efecto es beneficioso para ciertas aplicaciones [17].

2.5.2. Transformada Discreta de Fourier (DFT)

Entre los años 1670 y 1672 Isaac Newton mostró que un prisma puede descomponer la luz del sol en colores. A esta banda de colores Newton le dio el nombre de espectro. Este proceso de descomposición de la luz blanca en colores es llamado análisis de frecuencia. La transformada de Fourier es una herramienta matemática que en cierto sentido realiza lo mismo que el prisma, ya que descompone la señal analizada en todas sus frecuencias, permitiendo así conocer por ejemplo como se distribuye la energía de la señal.

La gran mayoría de las señales de interés práctico se pueden descomponer en una suma de sinusoides. Cuando la señal a descomponer es periódica, se habla de serie de Fourier. Para señales aperiódicas, en cambio, este proceso se denomina transformada de Fourier. Si la señal a ser analizada es discreta, la transformada de Fourier pasa a llamarse transformada discreta de Fourier o DFT por sus siglas en inglés (*Discret Fourier Transform*). Si la señal analógica $x_a(t)$ es muestreada satisfaciendo el criterio del teorema de Shannon y Nyquist para obtener la señal discreta $x(n)$, entonces la DFT de $x(n)$ se puede obtener a partir de la transformada de Fourier de $x_a(t)$. De esta manera el análisis realizado a partir de cualquiera de las dos transformadas es equivalente.

En general una señal aperiódica discreta $x(n)$ de largo infinito, tiene transformada de Fourier $X(\omega)$ continua y periódica de periodo 2π (equivalentemente $X(f)$ es periódica de periodo 1, siendo $\omega = 2\pi f$), dada por

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega n}. \tag{2.7}$$

Al ser periódica, $X(\omega)$ queda completamente determinada en un periodo. La siguiente ecuación muestra que si se toman N muestras de $X(\omega)$ entre 0 y 2π lo que resulta es la transformada discreta de Fourier $X(k)$ (asumiendo condiciones de Shannon-Nyquist).

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \quad \forall\, k = 0,1,2,\dots,N-1 \tag{2.8}$$

La ecuación (2.8) es conocida como ecuación de análisis. Equivalentemente si se quiere reconstruir $x(n)$ a partir de $X(k)$ se tiene la siguiente ecuación conocida como ecuación de síntesis.

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k)\, e^{j2\pi kn/N} \quad \forall\, n = 0,1,2,\dots,N-1 \tag{2.9}$$

Así, si se tiene una secuencia $x(n)$ de largo N, la cuál puede estar formada por valores complejos, entonces la DFT de esa secuencia es también una secuencia $X(k)$ de N valores complejos. En general se tiene que todos los valores son complejos como lo muestran las dos ecuaciones escritas más abajo.

$$x(n) = x_R(n) + jx_I(n) \tag{2.10}$$

$$X(\omega) = X_R(\omega) + jX_I(\omega) \tag{2.11}$$

Si $x(n)$ es real (i.e., $x_I(n) = 0$) se cumple que $X(\omega)$ es simétrica. Es decir, asumiendo que la señal es real y utilizando esto en la ecuación (2.7) se llega al siguiente par de relaciones.

$$|X(\omega)| = |X(-\omega)| \tag{2.12}$$

$$\sphericalangle X(-\omega) = -\sphericalangle X(\omega) \tag{2.13}$$

Por lo tanto se puede concluir que si $x(n)$ es real entonces $X(\omega)$ queda completamente caracterizada en la mitad de su periodo. En concreto cualquier intervalo de largo π es suficiente para obtener toda la información contenida en $X(\omega)$ o lo que es lo mismo cualquier intervalo de largo ½ caracteriza por completo a $X(f)$. Esto equivale a decir, como lo indica la siguiente ecuación, que solo $N/2$ valores de $X(k)$ son necesarios para obtener toda la información.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \quad \forall\, k = 0,1,\dots,\frac{N}{2} - 1 \tag{2.14}$$

De la ecuación (2.14) se deduce que la frecuencia angular ω esta dada por:

$$\omega = 2\pi f = 2\pi k/N \tag{2.15}$$

donde f se define como,

$$f = F/F_s \tag{2.16}$$

De la ecuación (2.15) se desprende la siguiente relación,

$$f = k/N \tag{2.17}$$

que en conjunto con (2.16) resultan en una expresión para la frecuencia analógica F correspondiente a cada valor k como se muestra a continuación.

$$F = F_s k/N \tag{2.18}$$

siendo $F_s = 1/T$ la frecuencia de muestreo. Lo que significa que los $N/2$ valores $X(k)$ corresponden cada uno a una frecuencia f entre 0 y $1/2$, o lo que es lo mismo a una frecuencia F entre 0 y $F_s/2$. Específicamente si F_s se mide en *Hertz* se tiene que a cada punto de la DFT le corresponde una frecuencia específica dada por las siguientes relaciones.

$$X(0) \rightarrow 0\ [Hz]$$

$$X(1) \rightarrow F_s * \frac{1}{N}[Hz]$$

$$X(2) \rightarrow F_s * \frac{2}{N}[Hz]$$

$$\vdots$$

$$X\left(\frac{N}{2}-1\right) \rightarrow F_s * \frac{\frac{N}{2}-1}{N} = \frac{F_s}{2} * \frac{N-2}{N}\ [Hz] \tag{2.19}$$

Estos $N/2$ puntos de la DFT, definidos en (2.19), son también conocidos como canales en un espectrómetro digital y cada uno representa una frecuencia específica de la señal analógica [17].

2.5.3. Transformada Rápida de Fourier (FFT)

Existen algoritmos óptimos para realizar el cálculo de la DFT. Estos algoritmos utilizan propiedades matemáticas de la función exponencial compleja $e^{-j2\pi kn/N}$ en la ecuación (2.8) reduciendo significativamente el costo en operaciones matemáticas necesarias. Los algoritmos que ocupan estas propiedades de simetría y periodicidad se llaman FFT (*Fast Fourier Transform*) y basan su funcionamiento en realizar el cálculo de la DFT en etapas o *stages*, en donde cada etapa consiste en una DFT más pequeña. Cuando el largo de la DFT es una potencia de 2 este algoritmo se llama radix-2 FFT. Por ejemplo una DFT de largo N =1024 requiere de N^2 =1048576 multiplicaciones para el cálculo directo, mientras que para la FFT se requieren solo $(N/2)log_2(N)$ =5120 multiplicaciones, lo que implica una mejora en la velocidad de computo de 204,8 veces [17].

2.5.4. Energía de una señal

Es muy importante en el análisis de una señal conocer la energía contenida en esta. Sobre todo si se trata de un espectrómetro. En términos generales se define la energía E_x de una señal $x_a(t)$ como

$$E_x = \int_{-\infty}^{\infty} |x_a(t)|^2 \, dt. \tag{2.20}$$

El teorema de Parseval muestra dos formas distintas de calcular E_x,

$$E_x = \int_{-\infty}^{\infty} |x_a(t)|^2 \, dt = \int_{-\infty}^{\infty} |X(f)|^2 \, df. \tag{2.21}$$

Por este motivo a la cantidad $|X(f)|^2$ se le denomina densidad espectral de potencia o PSD (*Power Spectral Density*). A partir de la definición anterior se tiene que la energía contenida en cada canal de frecuencia de la DFT está dada por

$$E_f = |X(f)|^2 = Re\{X(f)\}^2 + Im\{X(f)\}^2. \tag{2.22}$$

2.5.5. Zonas de Nyquist

Al estudiar el teorema de Nyquist se podría pensar erróneamente que la señal a ser muestreada solo puede ser banda base, es decir que su contenido energético se encuentre entre 0 Hz y un valor máximo usualmente denominado ancho de banda o BW (*Band Width*), como se indica en el teorema de Nyquist y en la ecuación (2.6) (Es posible generalizar la definición de ancho de banda como simplemente la diferencia entre la frecuencia máxima y la mínima contenida en una señal). Sin embargo en realidad el teorema de Nyquist se aplica también para señales pasa banda que se encuentren en las zonas o bandas de Nyquist. En efecto, si la frecuencia de muestreo satisface la ecuación (2.6), entonces BW define las zonas de Nyquist como se muestra en la figura 2.9.

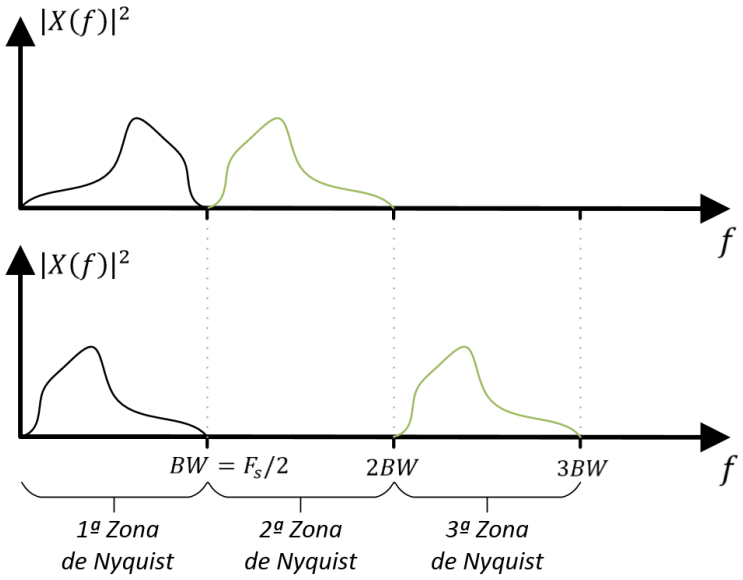


Figura 2.9: Conversión digital hacia abajo para señales pasa banda en zonas de Nyquist.

Si la señal pasa banda de color verde en la figura 2.9 está en la segunda zona de Nyquist (o en cualquier zona par), la señal tendrá un "alias" en la primera zona, pero con espectro invertido. De la misma manera, si la señal pasa banda se encuentra en la tercera zona de Nyquist (o cualquier zona impar), la señal tendrá un "alias" en la primera zona de Nyquist con el espectro en el mismo orden (véase figura 2.9). Este proceso define una conversión digital hacia abajo en banda base.

2.5.6. Filtros ideales y Filtro FIR.

En procesamiento digital de señales es muy común encontrarse con la necesidad de diseñar un filtro ya sea para asegurar un ancho de banda limitado y así evitar efecto *aliasing* o para cambiar la respuesta en fase de una señal. Esta sección se focaliza en filtros pasa bajo o LPF (*Low Pass Filters*), pasa banda o PB (*Pass Band*) y pasa alto o HPF (*High Pass Filters*). Un ejemplo de un filtro pasa bajo ideal se muestra en la figura 2.10.

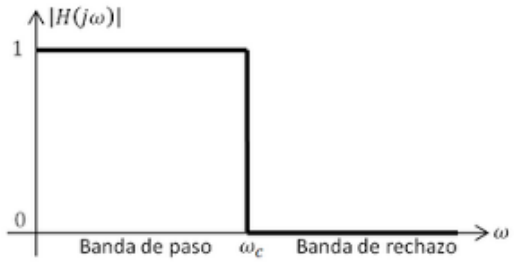


Figura 2.10: Respuesta en magnitud de un Filtro Pasa Bajo.

Estos filtros se caracterizan por una banda de paso, una banda de rechazo y una frecuencia de corte tal como se muestra en la figura 2.10. En este ejemplo las frecuencias mayores que ω_c son completamente atenuadas por lo tanto no se presentan a la salida del filtro. Las frecuencias menores que ω_c pasan sin ninguna atenuación por el filtro. Este tipo de filtros se denominan ideales y no son realizables físicamente, puesto que se requieren infinitas muestras en el dominio del tiempo para construirlos.

Los filtros ideales pueden ser vistos como una función rectangular cuya transformada inversa de Fourier es un *sinc* en el tiempo,

$$sinc(x) = \frac{\sin(\pi x)}{\pi x}. \tag{2.23}$$

La función *sinc* definida en la ecuación (2.23) es infinita en el tiempo. Si se utiliza una cantidad finita de puntos para representarla, entonces su transformada de Fourier será distinta a una función rectangular generando lo que se conoce como efecto *Gibbs*. Es por esto que se suelen utilizar otros tipos de filtros llamados FIR (*Finite Impulse Response*) que se pueden implementar en hardware y que se pueden asemejar a un filtro ideal tanto como se desee implicando mayor costo de hardware.

Un filtro FIR de orden M pertenece a una clase de sistemas lineales invariantes en el tiempo o LTI (*Lineal Time Invariant*) discretos que satisfacen la ecuación de diferencia dada por

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) \tag{2.24}$$

, y cuya función de transferencia está dada por

$$H(z) = \sum_{k=0}^{M} b_k z^{-k}. \tag{2.25}$$

La respuesta de estos sistemas lineales depende solo de M entradas pasadas y queda caracterizado por sus coeficientes $\{b_k\}$. Los coeficientes de un filtro FIR definen su respuesta en frecuencia, y corresponden a la respuesta al impulso $h(n)$ del filtro, es decir,

$$h(n) = \begin{cases} b_n, & 0 \le n \le M-1 \\ 0, & \sim \end{cases}. \tag{2.26}$$

La realización directa en hardware de un filtro FIR de orden M se muestra en la figura 2.11. La implementación de la figura 2.11 consiste en M delays, $M+1$ multiplicaciones y M sumas para el cálculo de $y(n)$ [17].

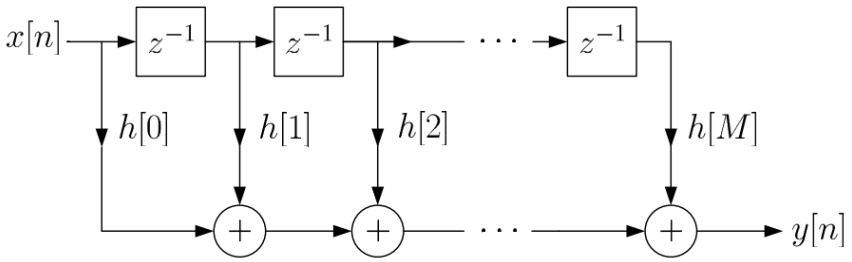


Figura 2.11: Esquemático de la implementación directa de un filtro FIR.

2.6. El espectrómetro

En Radioastronomía se habla de *back end* para referirse a aquellos dispositivos diseñados para analizar la polarización, estructura temporal o propiedades espectrales de la radiación. De todos los tipos de *back end* utilizados para propósitos especializados, los espectrómetros son probablemente los más usados [3]. La función que cumple un espectrómetro es calcular la densidad espectral de potencia de una señal. Estos dispositivos son implementados en un hardware analógico especialmente diseñado, pero en la actualidad está haciéndose cada vez más común el uso de procesadores. El cálculo del PSD se puede efectuar de dos maneras tal como se muestra en la Figura 2.12.

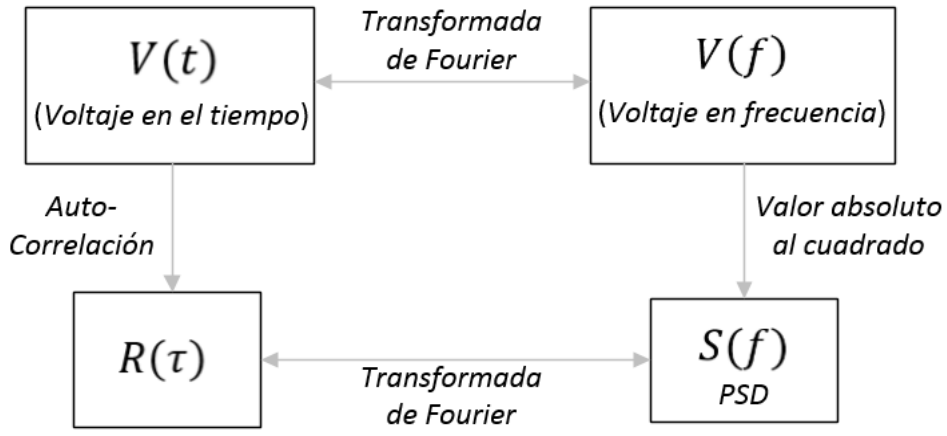


Figura 2.12: Diagrama de flujo del cálculo del PSD donde puede proceder de dos maneras distintas, pero equivalentes.

Los espectrómetros de auto correlación calculan la auto-correlación de la señal para luego realizar la transformada de Fourier y así obtener el PSD (flujo en sentido anti horario en la figura 2.12). Los espectrómetros de Fourier en cambio realizan la transformada de Fourier directamente y luego calculan el PSD (sentido horario en la figura 2.12). En este trabajo se diseñará un espectrómetro de Fourier digital.

2.6.1. Resolución espectral

Un parámetro importante dentro de un espectrómetro digital es la resolución espectral Δf, definida en la siguiente ecuación, que relaciona el ancho de banda total de la señal que se está procesando con la cantidad de canales del espectrómetro [18].

$$\Delta f = \frac{BW}{N} \tag{2.27}$$

Donde:

- BW= Ancho de banda procesado [Hz].
- N= Número de canales del espectrómetro.

2.6.2. Resolución Temporal

La resolución temporal es simplemente el tiempo que transcurre entre un espectro y otro. Este tiempo depende principalmente de las distintas etapas de procesamiento de los datos. Usualmente los datos se someten a una etapa de acumulación donde son promediados para mejorar la relación señal a ruido. Esta etapa es la que tiene el mayor efecto sobre la resolución temporal. Para observar eventos en escalas cortas de tiempo, como por ejemplo los pulsares se requerirá de una mayor resolución temporal, en cambio para observar líneas débiles se requiere de un tiempo de acumulación mayor y por ende una resolución temporal menor [18].

2.7. FPGA y sus aplicaciones al procesamiento de señales radio astronómicas

Un FPGA *(Field Programmable Gate Array)* es un arreglo de compuertas lógicas programables. Como se muestra en la Figura 2.13, el FPGA consiste en un circuito integrado digital con bloques lógicos configurables donde las conexiones entre estos bloques son también configurables. De esta manera se puede programar este tipo de dispositivos para implementar una gran variedad de tareas [19].

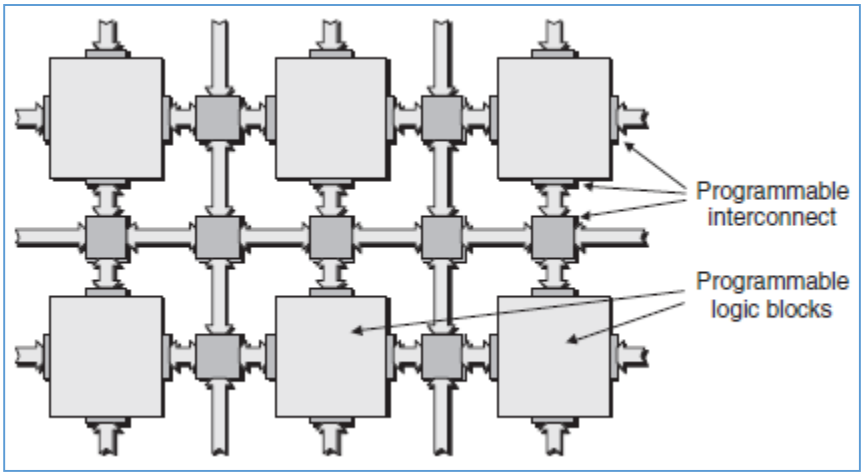


Figura 2.13: Arquitectura general de un FPGA en donde se aprecian bloques con lógica programable e interconexiones entre ellos también programables [19].

Se llama *Field Programmable* porque es programado en el "campo" es decir se está cambiando su funcionalidad interna fuera de la fábrica.

Actualmente los FPGA se utilizan en aplicaciones de procesamiento de señales radio-astronómicas, debido a su alta capacidad de procesamiento paralelo, y la gran cantidad de operaciones que es capaz de realizar en tiempos muy pequeños, dígase algunos pocos nanosegundos. Estas características la convierten en el candidato número uno para diseños de espectrómetros, ya que los datos son captados, procesados y guardados en memoria para su posterior análisis y todo esto en tiempo real, sin prácticamente ningún retraso en los datos ni tampoco acumulación indeseada de estos (buffers).

Capítulo 3

Objetivos y Metodología

En este capítulo se describe, el hardware y software utilizados para la realización del proyecto, y también se detallan las etapas de diseño del trabajo.

3.1. Antecedentes

A continuación se entregan los antecedentes generales que permiten entender el ambiente de trabajo. Se explicará en detalle todos los elementos con los que se trabaja y que se pueden apreciar en la figura 3.1. El trabajo presentado fue realizado en el Laboratorio de Ondas Milimétricas del Observatorio Astronómico Nacional, perteneciente a la Universidad de Chile.

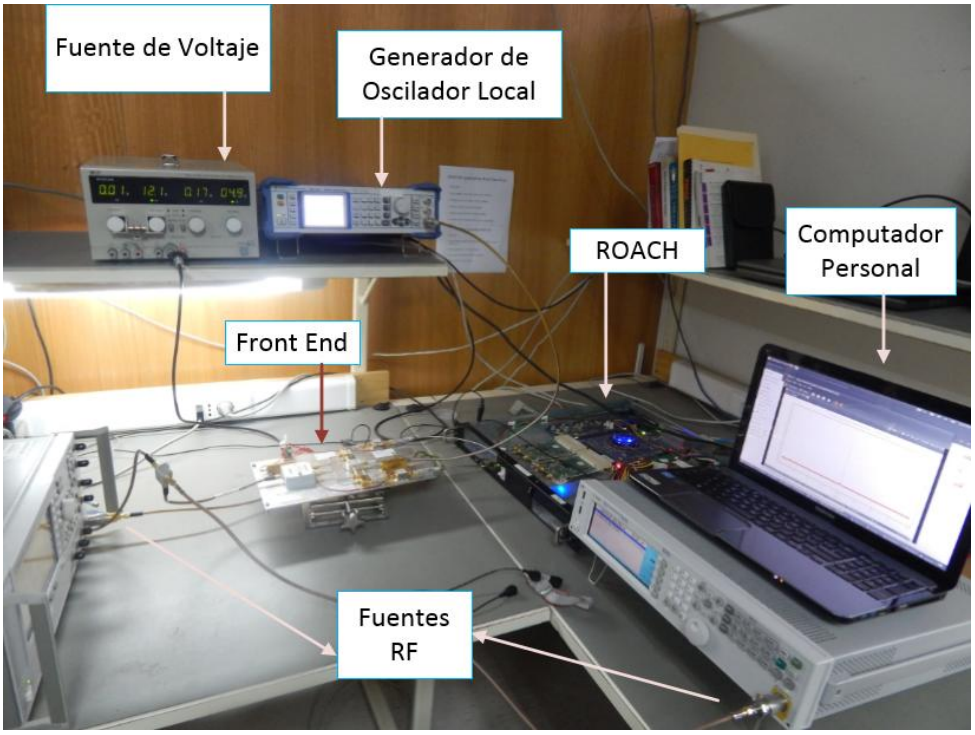


Figura 3.1: Fotografía que muestra el ambiente de trabajo con los principales instrumentos utilizados.

3.1.1. Descripción de la parte analógica o Frontend

Para implementar el mezclador de rechazo de banda lateral o SSM primero se utiliza un receptor de una entrada correspondiente al *front end* del sistema, que procesa analógicamente una señal de dos maneras distintas, generando así dos salidas que son

idénticas en magnitud, pero distintas en fase. Como se indica en la figura 3.2, el receptor toma una señal RF y genera dos señales desfasadas nominalmente 90º entre sí. Luego estas dos señales son pasadas por dos mezcladores idénticos que realizan conversión hacia abajo a ambas señales. Estas dos señales IF paralelas definen dos rutas o ramas que para efectos de simplicidad denominaremos i y q. A su vez las señales IF generadas en los puertos de salida de este receptor se denominarán IF_i e IF_q. En la figura 3.3 se muestra el *front end* utilizado en este trabajo. Fue diseñado en el Laboratorio de Ondas Milimétricas y puede procesar una señal RF con frecuencias entre 2 y 4 GHz. Este trabajo se focaliza en procesar una señal RF de 1 GHz de ancho de banda centrada en cerca de 3 GHz, de esta manera la conversión hacia abajo genera 2 IF de 500 MHz de ancho de banda cada una, las que posteriormente son procesadas por el *back end*.

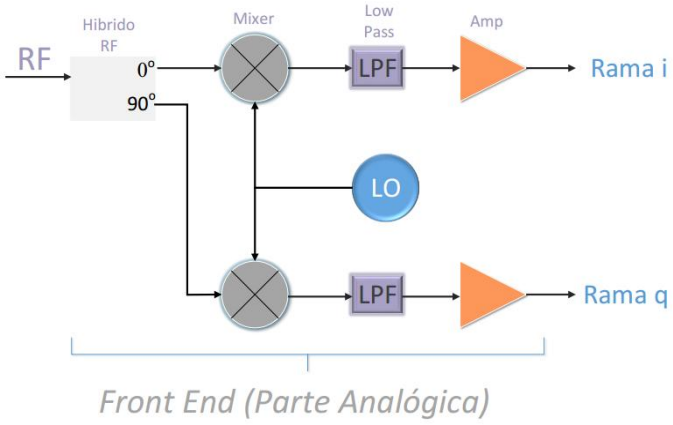


Figura 3.2: Diagrama del receptor analógico utilizado. Ambas ramas son receptores heterodinos.

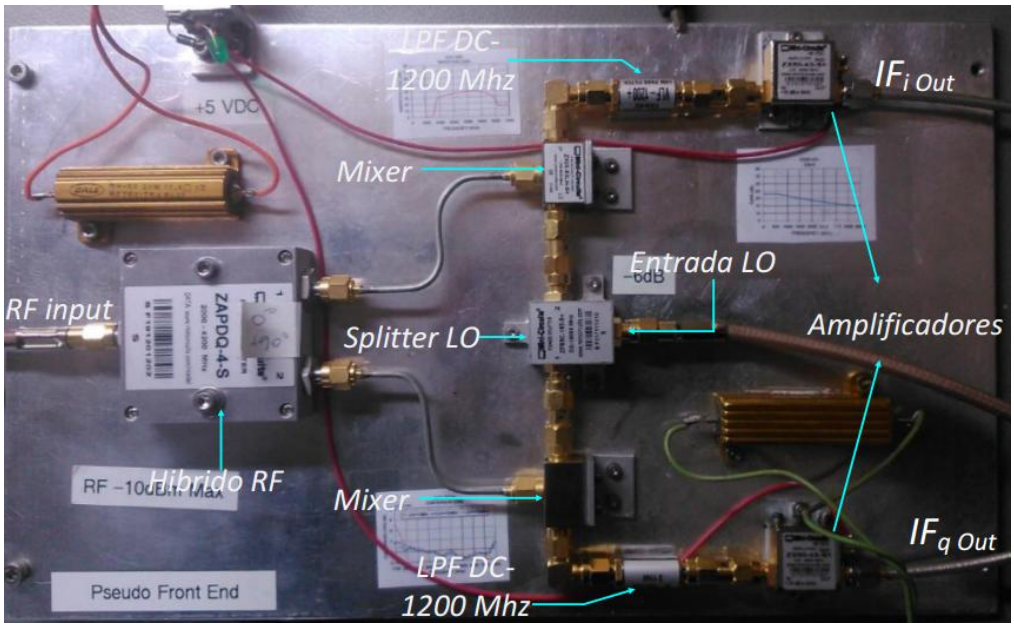


Figura 3.3: Fotografía del receptor analógico o front end utilizado en este trabajo. Se aprecia dos salidas IF de igual magnitud pero de distinta fase.

A continuación se da una breve descripción de cada elemento del *front end*.

Hibrido RF ZAPDQ-4-S minicircuits:

Es un *splitter* que divide la señal en dos caminos desfasados en 90º y con la misma potencia cada uno. Está diseñado para dividir una señal de entrada en un rango de 2000-4000 MHz, y es capaz de recibir una potencia de entrada de hasta 30 dBm.

Amplificador ZX60-43+:

Este elemento amplifica la señal IF entre 22 y 23 dB. Está diseñado para amplificar señales entre 0.5 y 4000 MHz. Debe alimentarse con 5 volts continuos. El *front end* cuenta con 2 de estos amplificadores, uno para la rama i y otro para la rama q (véase la figura 3.3).

Frequency Mixer ZX05-83LH+:

Es un mezclador, es decir toma dos frecuencias como entrada y a la salida se ve la suma de las frecuencias superpuesta con la diferencia de las frecuencias. Como interesa bajar la frecuencia de la señal se toma la frecuencia más baja mientras que la frecuencia alta es filtrada.

Generador de Señales Agilent E8257D PSG

Es el encargado de dar la señal de radio-frecuencia al *Front End* mediante un conector SMA (*SubMiniature version A*). La Señal de RF se entrega con una potencia de -20dBm. Este generador de señales puede generar un tono desde 250 kHz hasta 20 GHz con una resolución de 0,001 Hz. Posee una referencia externa de 10 MHz y esta se utiliza como señal de sincronía para todo el resto de los equipos.

Generador de Señales Rohde & Shwartz

Este generador tiene un rango de RF entre 9 kHz hasta 3.2 GHz. Se utiliza su salida de RF como oscilador local usualmente entre 2 y 3 GHz con 18 dBm de potencia. Se utiliza en modo *external reference* para utilizar la referencia de 10 MHz proveniente desde el generador Agilent. Se utiliza un conector SMA para la salida de radio-frecuencia.

3.1.2. Descripción del Backend

ROACH (Reconfigurable Open Architecture Computing Hardware)

En este trabajo el *back end* se implementa mediante una ROACH que consiste entre otras cosas en una tarjeta o una placa que contiene un FPGA dentro, además de dos convertidores analógicos/digitales (ADC, por sus siglas en inglés: *Analog to Digital Converter*) y un reloj interno que se utiliza como señal de sincronismo. La plataforma ROACH es básicamente la encargada de realizar el procesamiento digital de la información, y guardar los resultados en memorias, de las cuales se puede extraer tal información para su posterior interpretación. Los relojes internos de los ADC corren a 500 MHz. El reloj del FPGA corre a ¼ de la velocidad del reloj de los ADC, por cuanto debe procesar los datos en paralelo. Además la ROACH cuenta con un sintetizador que se enclava a una referencia externa con el fin de realizar los cálculos de manera sincronizada o en fase con las señales de entrada. Esta señal de sincronía es entregada por la fuente de RF.

En la figura 3.4 se muestra la arquitectura de una ROACH con un FPGA como bloque central, dos entradas con conectores de acoplamiento Z-DOK+ donde se pueden conectar tanto ADC como DAC. El subsistema PowerPC se encarga de controlar la ROACH, y entre otras cosas se preocupa de comunicar los registros y memorias RAM dentro del FPGA con dispositivos externos vía *Ethernet*. Ademas el PowerPC posee un sistema operativo llamado BORPH (*Berkeley Operating system for ReProgrammable Hardware*), diseñado especialmente para computadores reconfigurables basados en FPGA.

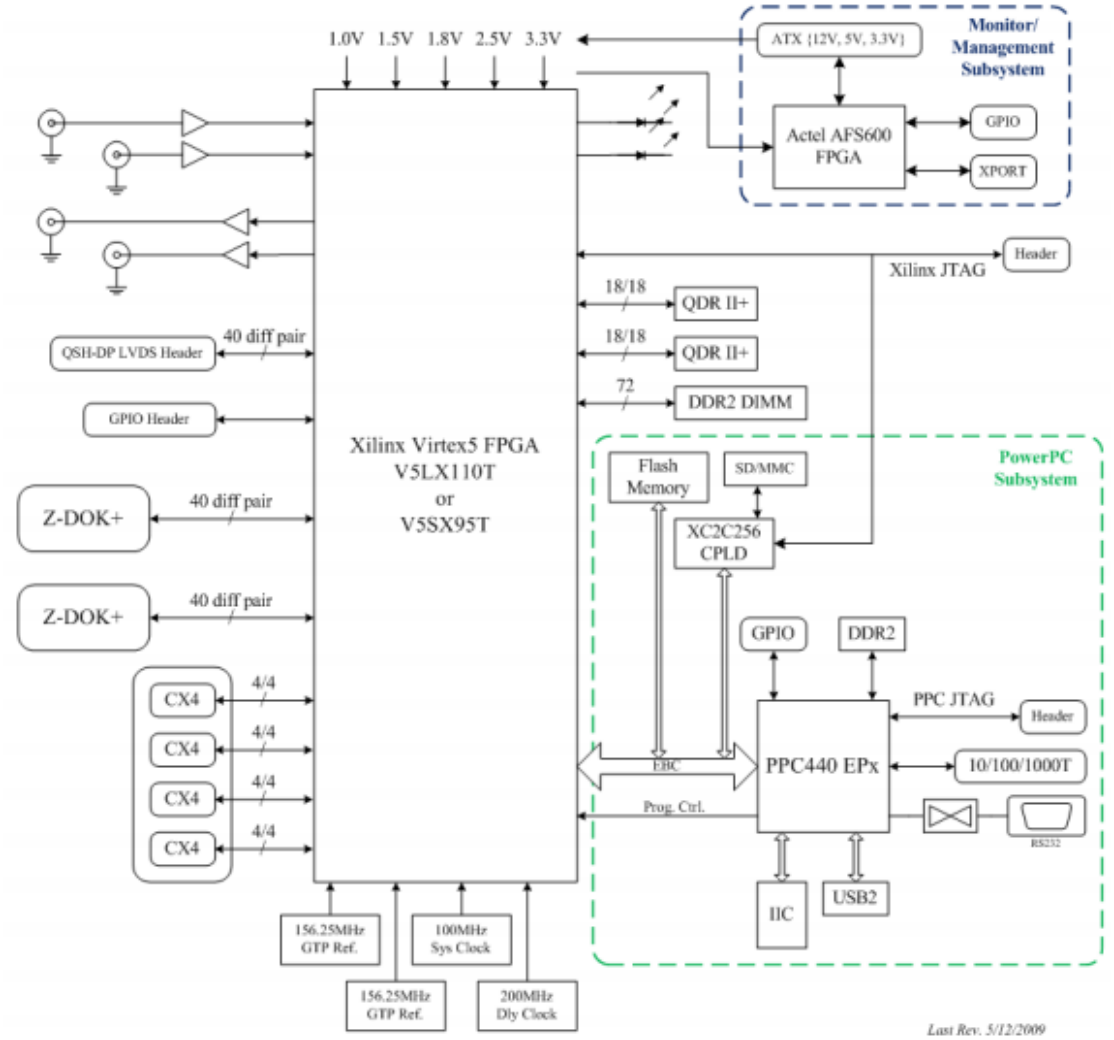


Figura 3.4: Diagrama de bloques de una ROACH que contiene un FPGA Virtex 5.

El ADC

La ROACH *board* contiene 2 ADC *national semiconductor adc083000* de 8 bits y 3 GSPS (*GIGA Samples Per Second*), por lo que puede procesar una señal de hasta 1,5 GHz. Posee 3 entradas vía conectores SMA, una para la señal de reloj, otra para la señal misma y la última para sincronizarse con otra ADC. Posee una salida física denominada Z-DOK+ con 40 conectores par diferencial que contiene tanto los datos de la señal muestreada y de la señal de reloj. Además posee una interfaz en *Simulink* por medio del bloque "*adc083000*". Cada muestra es representada por un dato tipo punto fijo $fix8\_7$.

Como se muestra en la figura 3.5, el reloj del FPGA corre 4 veces más lento que el reloj del ADC. Como este ADC toma muestras tanto en flanco de subida como de bajada de su reloj interno, se tiene que en un periodo del reloj del FPGA se toman 8 muestras de la

señal analógica. Esto implica que la tasa de muestreo f_s de la señal está dada por la ecuación (3.1).

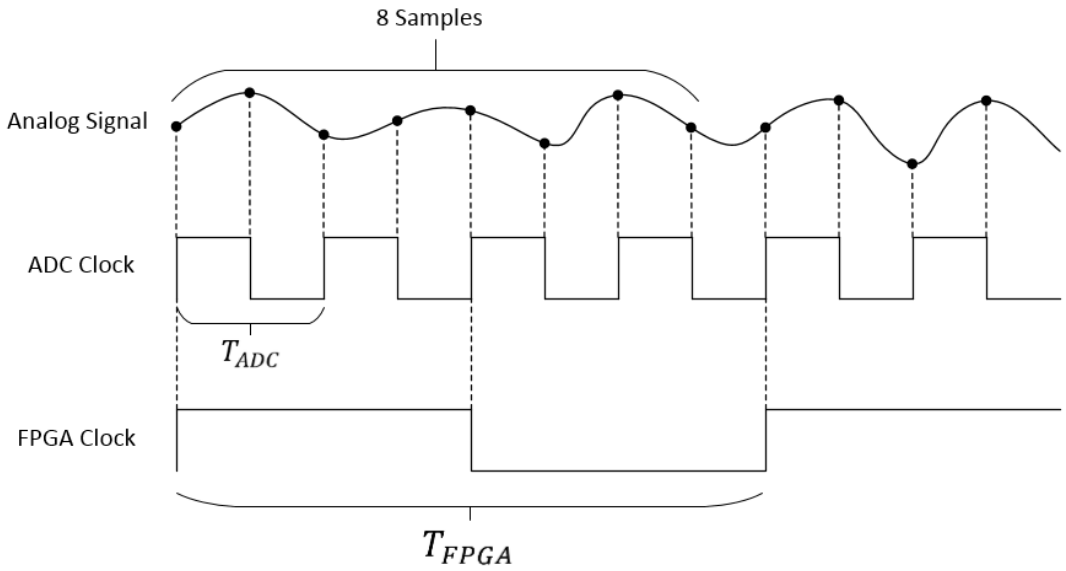


Figura 3.5: Proceso de muestreo del ADC. El reloj del FPGA es 4 veces más lento que el del ADC. El FPGA recibe 8 muestras simultáneas cada ciclo de reloj, las que se procesan en paralelo.

$$f_s = \frac{8}{T_{FPGA}} = \frac{1}{T_{FPGA}/8} = \frac{2}{T_{ADC}} = 2f_{ADC} \tag{3.1}$$

siendo,

T_{ADC}: Periodo del reloj del ADC.

T_{FPGA}: Periodo del reloj del FPGA.

f_s: Tasa de muestreo.

f_{ADC}: Frecuencia del reloj del ADC.

Considerando también la ecuación (2.6) se tiene que la frecuencia de este ADC concuerda con el ancho de banda de la señal procesada (BW), obteniendo la siguiente relación:

$$BW = f_{ADC}. \tag{3.2}$$

Como se desea procesar una señal IF con $BW = 500\,MHz$ se debe correr los ADC a $500\,MHz$. Por lo tanto en un ciclo de su reloj interno, el FPGA recibe 8 muestras (de 8 bits cada una) provenientes de un solo ADC, es decir toma 8 muestras por cada ciclo de su reloj interno. En particular para este trabajo, el ciclo de reloj del FPGA se programa para ser igual a 0.008 µs, por lo tanto el FPGA procesa 2 mil millones de muestras por segundo (2 GSPS) o equivalentemente 2 GB/s (2 GIGA Bytes por segundo).

FPGA

El *hardware* FPGA contenido en la ROACH es un Xilinx Virtex 5 V5SX95T. El sistema operativo en el PowerPC corre un proceso que modifica el hardware mediante la ejecución de un archivo BOF (BORPH *executable File*), que se comporta como cualquier otro proceso. Entre otras cosas este sistema operativo provee acceso a un *Shell* vía ssh, y permite leer ciertas regiones del FPGA, ya sea registros o memorias RAM.

Señal de Referencia

La señal de reloj se implementa con un sintetizador *Valon 5007*, que entrega dos fuentes de frecuencia diseñadas para implementar reloj de alta calidad o para oscilador local. Posee una memoria FLASH que le permite guardar la frecuencia prestablecida y recordarla después de apagar el dispositivo. Este elemento se enclava a la señal de reloj de la fuente *Agilent*, para luego alimentar la entrada de reloj de los ADC en completa sincronía con todo el sistema.

3.1.3. Descripción del toolflow

El *toolflow* de CASPER (*Collaboration for Astronomy Signal Processing and Electronic Research)* facilita un ambiente de diseño para el desarrollo de instrumentos con el hardware de CASPER.

En particular, dentro del software requerido para el diseño se puede mencionar,

- Sistema operativo Ubuntu 12.04 64 bit
- MATLAB Simulink R2012a/b
- Xilinx v14.5
- Librerias MSSGE. (https://github.com/ska-sa/mlib_devel)

Este *toolflow*, posee bloques de lógica implementados en Xilinx, pero también otros bloques creados por CASPER, especializados en el procesamiento de señales radioastronómicas. Una vez terminado el diseño en MATLAB Simulink, se procede a la compilación de este, proceso que termina con la creación de un archivo de extensión *.bof*, el cual contiene la "ruta física" que se requiere implementar en el *hardware* del FPGA considerando restricciones temporales y de recursos en *hardware*.

También es importante mencionar el uso de Python para la creación de *scripts* que se comunican con la ROACH, obteniendo los datos procesados para realizar post procesamiento y graficar los espectros en tiempo real.

3.2. Objetivos y metodología

Este trabajo se divide en dos partes. La primera se basa en perfeccionar la medición de rechazo de banda lateral de un espectrómetro diseñado e implementado previamente en [9]. Este sistema se muestra en detalle en las figuras 3.6 y 3.7. Primero se miden los desbalances de fase y de amplitud como se indica en la figura 3.6. En esta configuración el ADC de la rama i muestrea la IF_i proveniente del receptor analógico y genera $N = 4096$ muestras $x^i(0), x^i(1), x^i(2), \dots, x^i(N-1)$ (lo mismo para la rama q, generando los valores $x^q(0), x^q(1), x^q(2), \dots, x^q(N-1)$). A partir de estos N valores, el bloque FFT calcula $X^i(k)$ y $X^q(k)$ implementando eficientemente la ecuación (2.8), para k=0,1,2, …,2047 (ya que la DFT es simétrica, se ocupa la mitad de los 4096 canales de la FFT). Para medir $X^i(k)$ y $X^q(k)$ en USB, primero se fija el tono RF en una frecuencia mayor al oscilador local, tal que los tonos IF_i e IF_q a la entrada de los ADC correspondan a la frecuencia k-esima dada por la ecuación (2.19). Lo anterior se realiza para los 2048 canales del espectro. Luego el procedimiento se repite pero en LSB, es decir, el tono RF se fija en frecuencias por debajo del oscilador local. El bloque FFT entrega los 2048 valores del espectro en cuatro líneas paralelas, por lo tanto, este bloque entrega un espectro completo cada 512 ciclos de reloj del FPGA (4 microsegundos). Estos datos son guardados en ocho memorias RAM (*Random Access Memory*) de largo 512 (cuatro memorias para la rama i y cuatro para la rama q), las cuales son leídas por un comando en lenguaje *Python*, para realizar los cálculos mostrados en la figura 3.6.

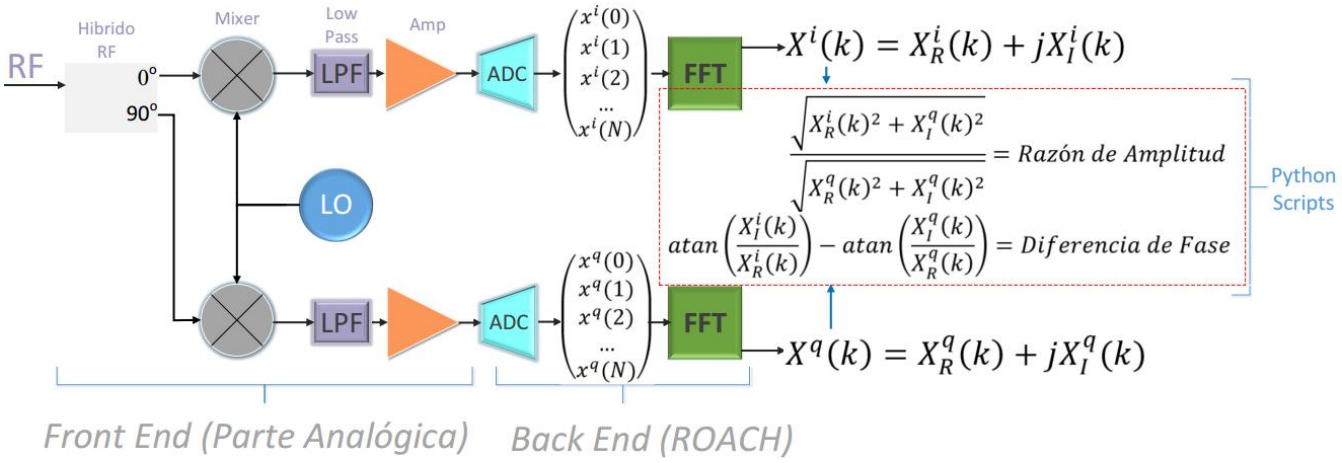


Figura 3.6: Diagrama de bloques del modelo que se utiliza para medir los desbalances de fase y de magnitud entre las ramas del receptor analógico.

La figura 3.7 muestra el diseño realizado en [9], y que es utilizado para realizar el rechazo de banda lateral. Es básicamente la implementación del esquema SSM explicado en la figura 2.6, pero considerando constantes complejas que ponderan los valores en las líneas de la etapa hibrida IF. Las constantes en azul, se obtienen a partir de las mediciones realizadas en la figura 3.6. Al igual que en el diseño mostrado en la figura 3.6, la FFT de este diseño

entrega sus 2048 datos en cuatro líneas paralelas. Luego de pasar por el Hibrido IF, se calcula el modulo al cuadrado de los puntos de la FFT. En seguida los espectros son acumulados y finalmente guardados en memorias RAM de la misma forma que en el diseño de la figura 3.6. Por lo tanto, la tasa de transmisión de un espectro hacia las memorias depende de cuantos espectros sean promediados, y es mucho menor que en el diseño de la figura anterior. La cantidad de espectros promediados se conoce como largo de acumulación.

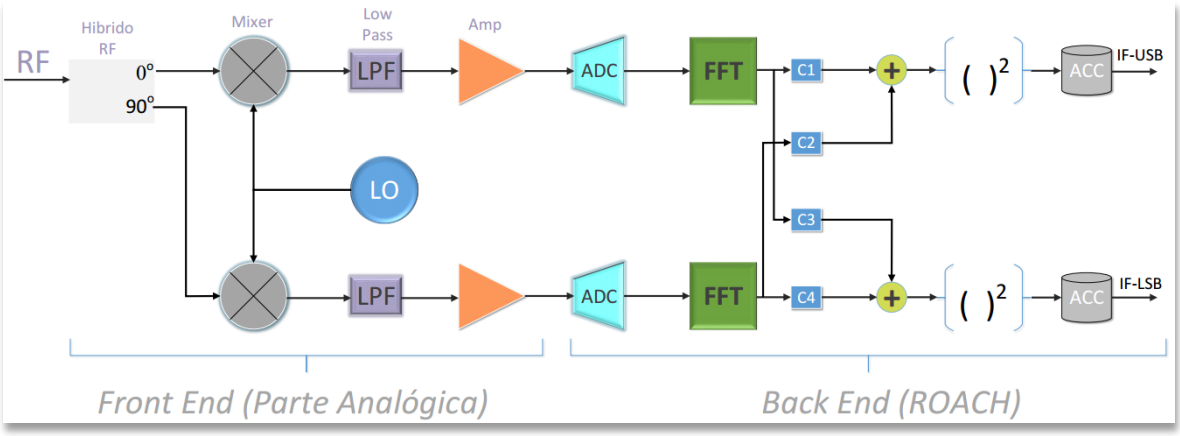


Figura 3.7: Diagrama de bloques del proceso de separación de banda lateral mediante el uso de una ROACH. Este esquema básicamente es la implementación de la configuración SSM explicada en el capítulo 2.

El objetivo entonces, es diseñar un bloque de control de datos que asegure que para cada valor de n, cada uno de los valores $X^i(k)$ y $X^q(k)$ de la figura 3.6 provengan de muestras $x^i(n)$ y $x^q(n)$ que hayan sido tomadas simultáneamente por los ADC de las ramas i y q respectivamente. Se dirá entonces que los espectros son simultáneos.

Como la velocidad de escritura no está sincronizada de ninguna manera con la velocidad de lectura de los códigos, puede darse el caso en que no se cumpla la simultaneidad de los espectros, perdiendo así correlación de fase entre los espectros de ambas ramas. Este efecto de falta de sincronía introduce ruido en las mediciones de rechazo de banda lateral, y su efecto es más notorio cuando se agrega ruido de fase, ya sea a la RF o al oscilador local.

El bloque de control de datos a diseñar, debe recibir una señal externa (proveniente del código *Python* en el computador PC) que señale que las memorias serán leídas. Al momento de recibir esta señal, el bloque debe esperar a que un nuevo espectro llegue a las memorias, y en ese mismo instante debe habilitar la escritura de estas. Inmediatamente después de que el espectro se grabe en las memorias, el bloque deshabilita la escritura para que luego el código extraiga la información de los espectros desde las memorias. Después de esto, el código envía una señal al bloque indicando que la lectura ha finalizado. Esta señal permite que la configuración interna del bloque vuelva al estado inicial, a la espera de una nueva solicitud de lectura por parte del código. Este diseño asegurará la simultaneidad de los espectros i y q.

La segunda parte de este trabajo consiste en el diseño de un espectrómetro con ancho de banda seleccionable. Esto quiere decir que se seleccionará un cierto ancho de banda (una porción de ancho de banda de la señal muestreada) y se utilizarán todos los canales espectrales de la FFT para observar ese ancho de banda.

Para llevar a cabo este espectrómetro, se creará una etapa de procesamiento de las muestras tomadas por el ADC en el dominio del tiempo. Este procesamiento consiste básicamente en realizar una decimación, es decir se filtra la señal para luego realizar un *downsample*. Se utilizará un bloque diseñado por CASPER llamado *dec_fir*. Este bloque implementará un filtro FIR cuyos coeficientes se cargarán en los parámetros de entrada del *dec_fir*. Para diseñar estos filtros se utilizará la herramienta *FDATool* de Xilinx.

Capítulo 4

Implementación

4.1. Diseño de un bloque de control de datos

En esta sección se diseñará un bloque de control de datos que pretende mejorar las medidas de rechazo de banda lateral obtenidas en trabajos previos, disminuyendo la sensibilidad al ruido de fase. Además se hará uso de dos modelos previamente diseñados en [9] (idénticos a los de las figuras 4.9 y 4.10), utilizando MATLAB Simulink. El bloque diseñado en esta sección se agregará en los modelos preexistentes para resolver el problema de no-simultaneidad de los espectros. Un modelo calcula las diferencias de amplitud y de fase entre ambas salidas del receptor analógico haciendo un barrido de frecuencia por los canales de la FFT, tanto en LSB y USB. A este modelo se le llamará en adelante simplemente modelo calibrador. El otro modelo utiliza las medidas tomadas por el modelo calibrador e implementa la configuración mostrada en la figura 3.7. En toda la sección 4.1 a este modelo se le denomina espectrómetro. Las constantes C_2 y C_3 se obtienen de las medidas realizadas por el modelo calibrador y son cargadas en el espectrómetro por medio de comandos *Python*. El espectrómetro contiene una etapa de acumulación en la que se suman sucesivos espectros saliendo de la FFT para luego grabar el resultado en memorias. El modelo calibrador no tiene etapa de acumulación. Ambos modelos fueron programados para procesar una IF de 500 MHz de ancho de banda con 2048 canales.

4.1.1. Especificaciones de diseño

Como se mencionó previamente, se cuenta con un diseño realizado en otro trabajo, que calcula la amplitud y la fase de cada rama del receptor analógico. Este diseño no contiene un bloque de acumulación de datos, por lo tanto la velocidad de entrega de un espectro es muy alta. La figura 4.1 muestra el proceso de calibración en el que es necesario calcular la amplitud y fase de las dos ramas IF.

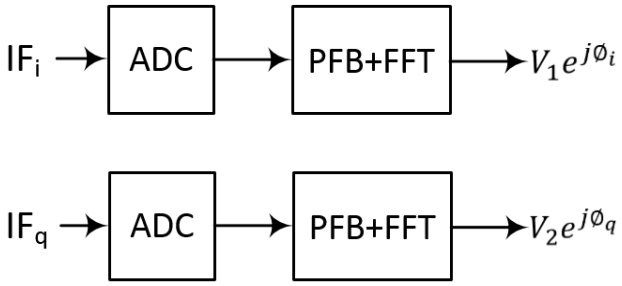


Figura 4.1: Diagrama de flujo del modelo calibrador. Se aprecian dos ramas identificadas con los subíndices i y q.

De esta manera se definen los ángulos $\emptyset_{USB}$ y $\emptyset_{LSB}$ para cada uno de los canales de la FFT según las siguientes relaciones [13].

$$\emptyset_{USB} = \pi - (\emptyset_i - \emptyset_q) \tag{4.1}$$

$$\emptyset_{LSB} = (\emptyset_i - \emptyset_q) - \pi \tag{4.2}$$

Dejando las constantes C_1 y C_4 iguales a la unidad, las constantes C_2 y C_3 se calculan con las siguientes ecuaciones, las cuales después se cargan al espectrómetro, tal como lo muestra la figura 3.7 [13].

$$\frac{1}{C_2} = \frac{1}{X} e^{-j(\emptyset_{LSB} - \pi)} \tag{4.3}$$

$$C_3 = \frac{1}{X} e^{-j(\emptyset_{USB} - \pi)} \tag{4.4}$$

donde: $X = \frac{V_1}{V_2}$.

El problema está en que estos datos están contaminados con ruido de fase, producto de la no sincronización entre lectura y escritura de las memorias en la etapa final del procesamiento digital. Esto provoca que el ruido se introduzca en C_2 y C_3 y por ende en el SRR.

La figura 4.2 muestra las memorias BRAM de color amarillo en las que se guardan los datos procesados por el FPGA. Estas memorias son leídas por medio de un comando en lenguaje *Python*. Estas memoria son de 64 bits y de largo 512. Estas BRAM tienen tres puertos de entrada y uno de salida que no se utiliza. El dato *signed fixed* de 64 bits que entra al puerto *data_in* se guarda en la dirección *addr* de 9 bits, solo si la entrada booleana *we* (*write enable*) es 1. Si *we* es 0, la escritura está bloqueada. La lectura de esta memoria se realiza con el siguiente comando:

*"struct.unpack('>512q',fpga.read('dout0_0',512*8,0))"*,

donde *dout0_0* es el nombre de la BRAM. Este comando está dentro de una función definida en *Python* llamada *get_data()* que lee todas las memorias y genera arreglos con todos los valores indexados en orden correlativo a cada memoria.

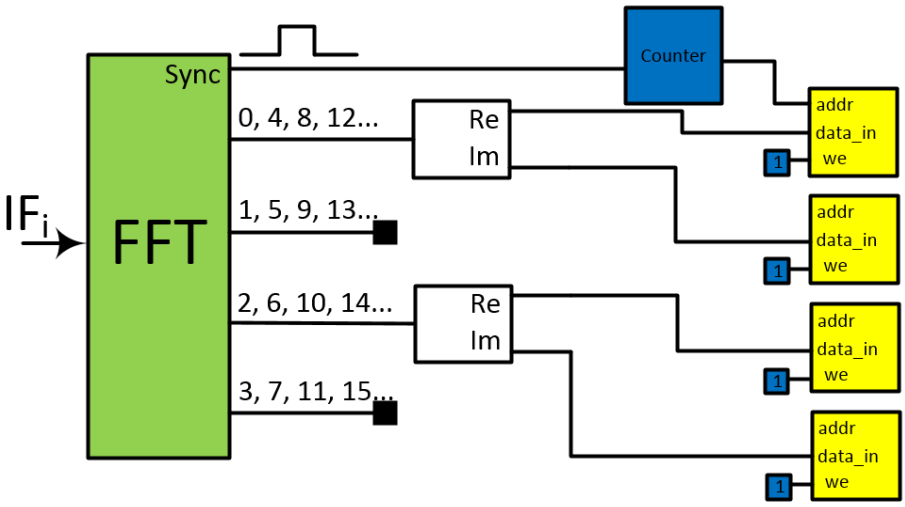


Figura 4.2: Esquema del modelo calibrador sin data_ctrl. Solo se muestra la rama i, sin embargo el modelo calibrador considera ambas ramas (i y q). Se muestra la multiplexación de los canales indicados con números. Solo se utilizan canales pares.

La figura 4.2 muestra cómo funciona el modelo calibrador sin bloque de control de datos. Como se aprecia, existe una señal de sincronismo en forma de pulso llamada *sync* la cual tiene la labor de sincronizar todos los contadores y señales de direccionamiento cada cierto tiempo, y también indica que en el ciclo de reloj siguiente sale un nuevo espectro desde la FFT (en adelante un ciclo de reloj se referirá al FPGA), partiendo por el canal 0. Es decir la FFT entrega 2048 canales e inmediatamente después comienza a entregar los siguientes 2048 canales del siguiente espectro. El pulso *sync* activa un contador de 9 bits que a su vez da direccionamiento a los datos dentro de la memoria. El puerto *we* de las memorias es siempre alimentado con un 1 booleano. Los canales impares no son considerados, sin embargo las constantes de las ecuaciones (4.1) a (4.4) para estos canales son obtenidas como el promedio de las constantes de los canales adyacentes. El procesamiento de la IF_q es idéntico y todo se realiza en el mismo modelo.

La velocidad del FPGA es mucho mayor que la velocidad de lectura del comando de *Python*. Por lo tanto ocurre que en una lectura de memoria, esta se reescribe sucesivas veces insertando ruido de fase en los datos. Esto es equivalente a decir que en una memoria habrá datos de fase en distintos tiempos. Es necesario entonces crear un bloque de control de datos que deshabilite la escritura de la memoria mientras se escribe. En particular se requiere un bloque que cumpla con las siguientes especificaciones.

1) Su entrada debe ser el pulso *sync* o algún derivado de esta.

2) Su salida debe alimentar los *we* de todas las memorias por igual.
3) Debe deshabilitar la escritura mientras las memorias se estén leyendo.
4) Debe tener dos registros para comunicarse con la función *get_data()*. Un registro indicara al bloque si la lectura fue realizada con éxito y otro registro deberá indicar el momento en que la función *get_data()* es llamada por el código de *Python*.
5) Debe poder usarse tanto en el modelo de Calibración (sin acumulador) como en el espectrómetro (con acumulador).

4.1.2. Elección de bloques

Para el diseño del bloque de control de datos se utilizarán diversos bloques de lógica diseñados por Xilinx y algunos diseñados por CASPER.

Registros

Se utilizarán dos registros requeridos en las especificaciones. Uno se denominará *sel_we* y será escrito con un 1 booleano desde *Python* cuando se solicite una nueva lectura de memorias. El otro se denominará *lec_done* y será escrito con un 1 booleano desde *Python* cuando se haya realizado la lectura de manera satisfactoria.

Constantes

Se utilizarán tres constantes booleanas con bloques *Xilinx*. Dos de ellas son 0 y servirán para deshabilitar la escritura en distintos casos. La otra es un 1 y sirve para el modelo con acumulador.

Multiplexores de 2 entradas

Se utilizarán tres multiplexores de dos entradas, en los cuales una señal booleana llamada *sel* seleccionara cuál de las dos entradas pasará por el puerto de salida. Si *sel* es 0 entonces saldrá la entrada *d0*, en caso contrario saldrá *d1*. Estos bloques sirven para deshabilitar el *we* cuando convenga.

Edge detect

Estos bloques generan un pulso cuando detectan un flanco de subida o un flanco de bajada. Tiene la posibilidad de detectar sólo flanco de subida, sólo de bajada o ambos. Servirán para activar o desactivar los contadores cuando ciertos eventos ocurran.

Pulse extender

Este bloque es muy importante ya que básicamente extiende el pulso *sync* para que dure 512 ciclos de reloj. De esta forma el *we* será 1 y grabará los 512 datos que interesan en las memorias.

Slice

Este bloque extrae una cantidad de bits de un número binario. En este caso estos bloques extraerán un bit y así transformarán los datos a booleanos.

Not

Se utilizará un negador de Xilinx para invertir una señal booleana.

And

Se requieren tres bloques and de Xilinx que implementan la función lógica *and*. Básicamente indicarán los rangos de tiempo en que dos eventos ocurren simultáneamente.

Counter

Se utilizarán dos contadores binarios de *Xilinx* para diferenciar dos estados distintos. Un contador cambiará de valor cuando la lectura está hecha, mientras que el otro cambiará de valor cuando la escritura haya terminado.

4.1.3. Detalles de funcionamiento

En esta sección se muestra el detalle del bloque de control de datos, tanto para el modelo calibrador (sin acumulador) como para el espectrómetro. Estos dos diseños son muy similares. Este bloque fue creado con el nombre *data_ctrl*, y maneja solo datos tipo *boolean*. Posee una entrada llamada *valid* que consiste en un pulso de duración igual a un ciclo de reloj del FPGA ($8\ ns$) que indica que 512 datos están entrando a las memorias (uno cada ciclo de reloj). La salida de este bloque se llama *we* y está conectada al puerto de habilitación de escritura de las memorias BRAM como se indica en la figura 4.3. El diseño final de este bloque para el caso sin acumulador se muestra en la figura 4.4.

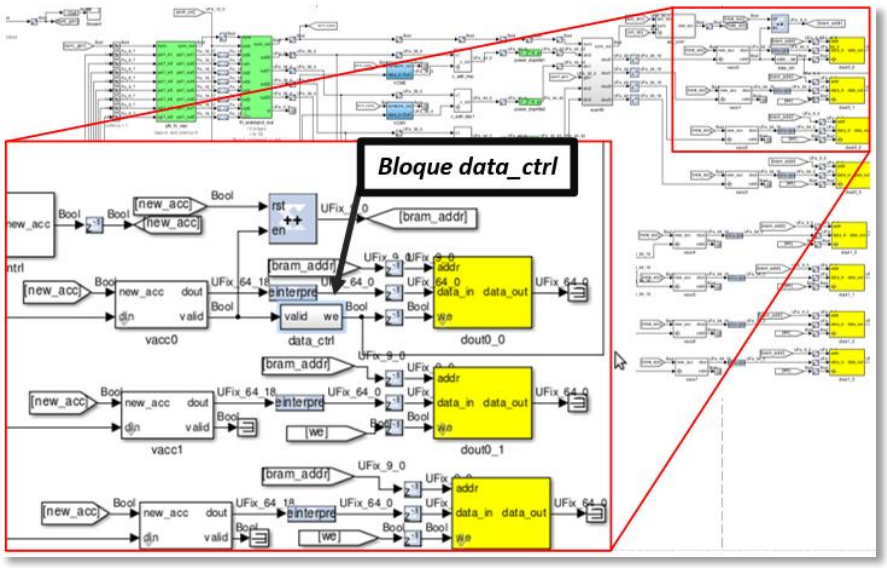


Figura 4.3: Ubicación del bloque data_ctrl en el modelo con acumulador.

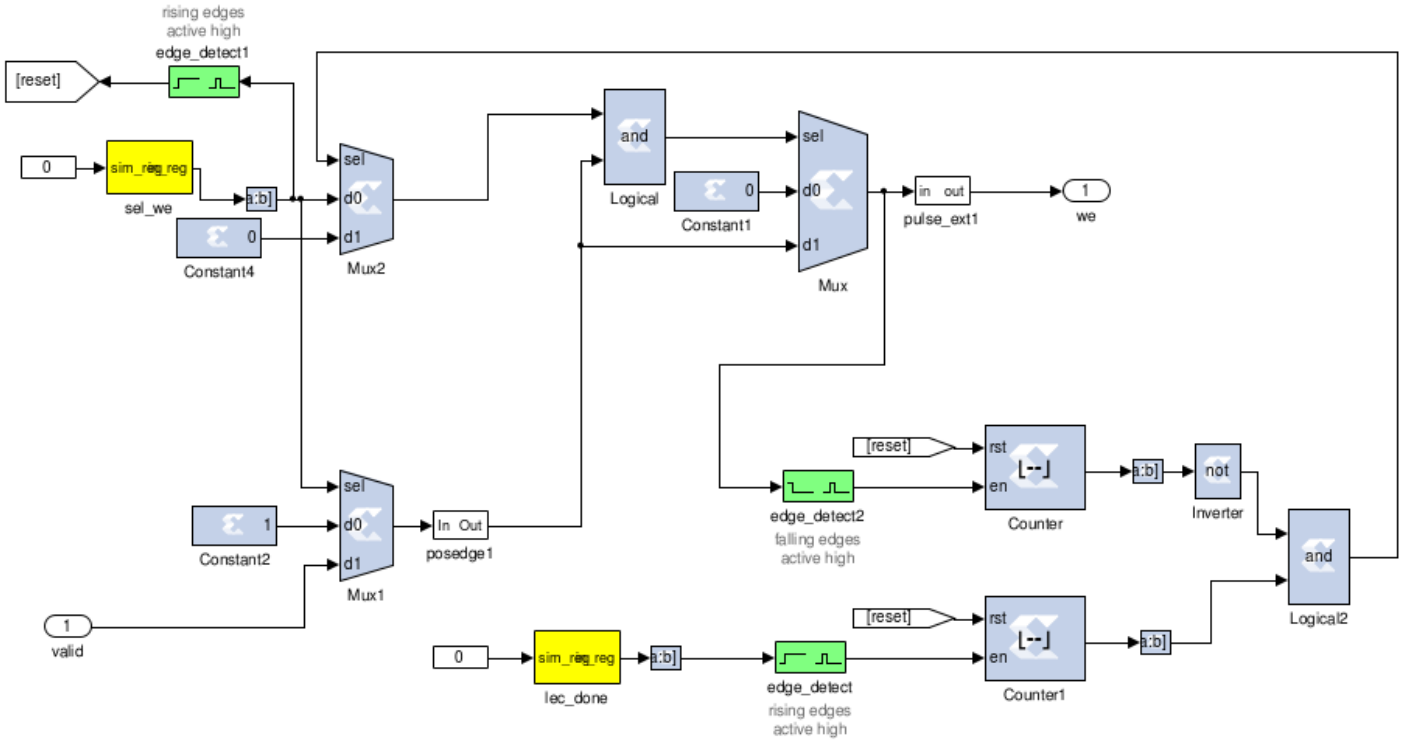


Figura 4.4: Diseño final del bloque data_ctrl para el modelo sin acumulador.

La figura 4.5 muestra un gráfico temporal de casi todas las señales booleanas dentro del *data_ctrl*. Las señales que tienen el mismo nombre que un bloque, representan la señal

de salida de dicho bloque. Por ejemplo la señal *mux1* representa la salida del multiplexor *Mux1* en la figura. En rojo se puede ver la salida del bloque (*we*). Esta señal es un 1 lógico de 512 ciclos de reloj, o equivalentemente 4 microsegundos. Este 1 lógico coincide en el tiempo con la entrada de nuevos datos desde la FFT y se genera sólo después de una solicitud de lectura. Luego de 512 ciclos de reloj esta señal baja a 0 y no vuelve a subir hasta después de que una nueva lectura sea solicitada, asegurando así que los datos escritos en la memoria pertenecerán al mismo espectro de la FFT. El instante en el que se solicita una escritura es aleatorio. Una vez realizada la lectura, los registros vuelven a su estado inicial esperando una nueva solicitud de escritura y así el proceso mostrado en la figura 4.5 se repite.

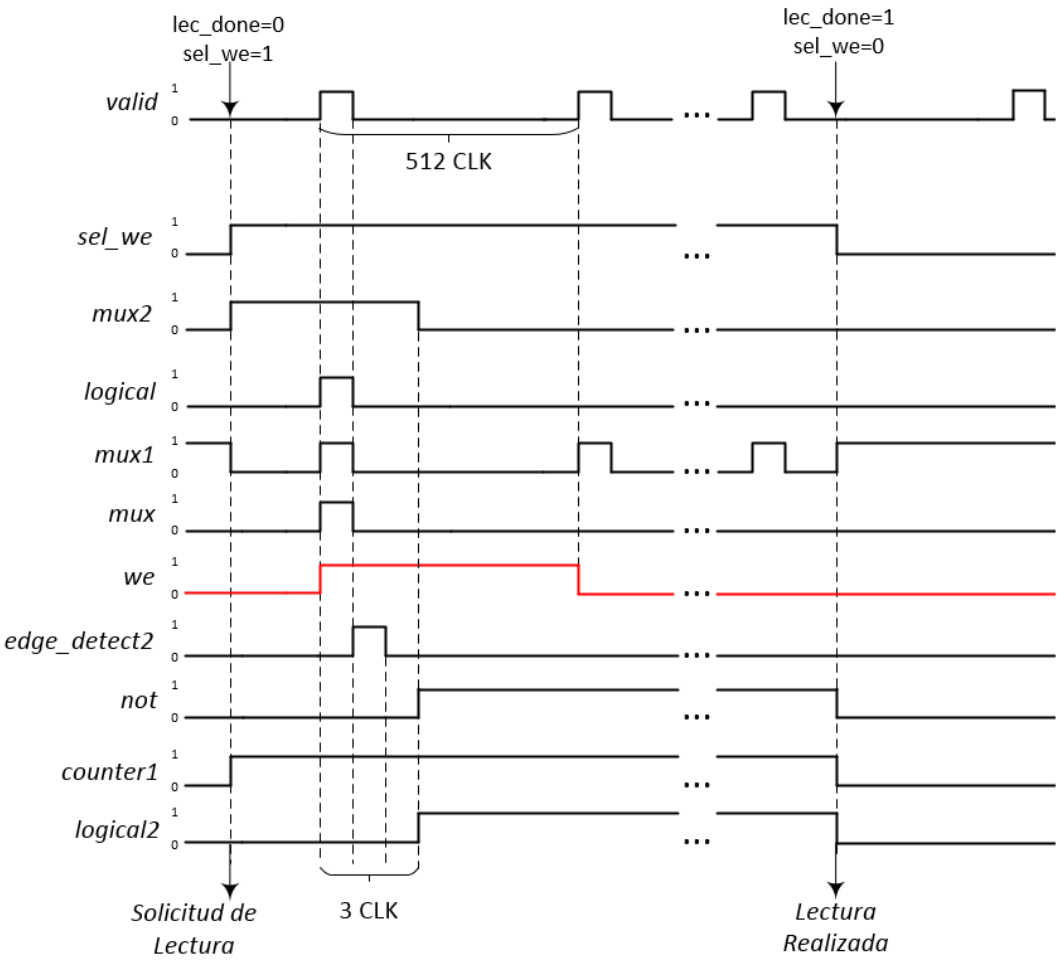


Figura 4.5: Diagrama de tiempo de las señales dentro del data_ctrl sin acumulador. CLK corresponde a un ciclo del reloj del FPGA. 512 datos validos arriban cada pulso valid.

La figura 4.6 muestra el diseño del bloque de control de datos para el caso del modelo con acumulador. En este caso la señal *valid* es un pulso de 512 ciclos de duración cada $l * 512$ ciclos, donde l es el largo de acumulación.

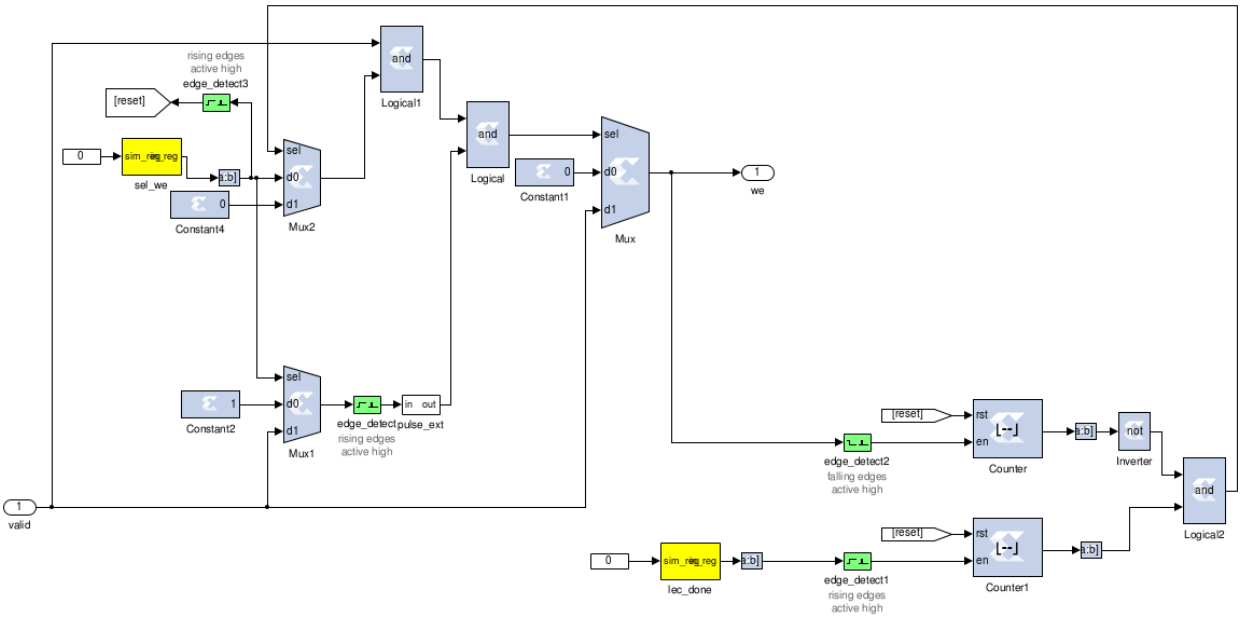


Figura 4.6: Diseño final del bloque data_ctrl para el modelo con acumulador.

La relación entre entrada y salida de este bloque se muestra en la figura 4.7. En esta configuración puede darse el caso en que la solicitud de lectura se envía cuando el acumulador está entregando datos válidos, en cuyo caso el *data_ctrl* esperará al siguiente pulso para habilitar la escritura, tal como se muestra en la figura 4.8.

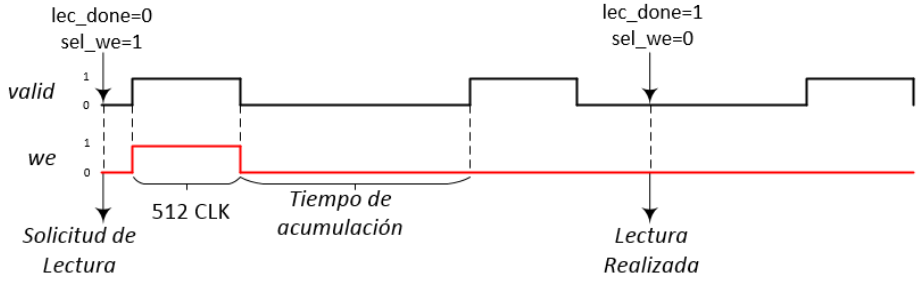


Figura 4.7: Diagrama de tiempo del data_ctrl con acumulador.

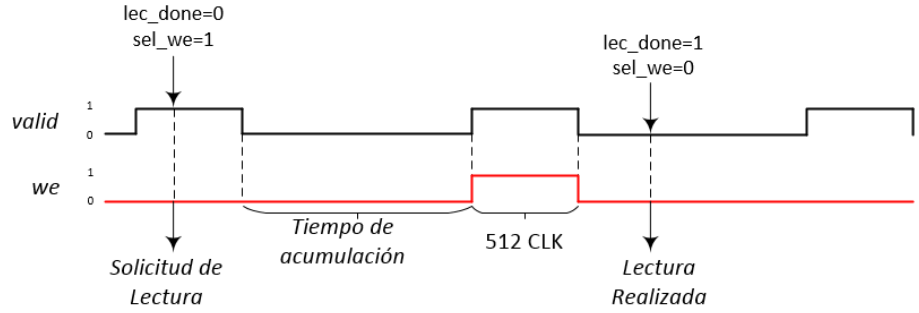


Figura 4.8: Diagrama de tiempo del data_ctrl con acumulador, cuando la solicitud de lectura coincide con el pulso valid de 512 CLK de duración.

La figura 4.9 muestra el modelo calibrador en el que está inserto el diseño del *data_ctrl* de la figura 4.4. Este modelo de Simulink realiza el proceso explicado en las figuras 4.1 y 4.2, pero con control de datos. Se aprecia en amarillo a la izquierda un bloque que representa a los ADC, luego una etapa llamada PFB (*Polyphase Filter Bank)* representada por los bloques verdes, donde se realiza un filtrado de la señal y una posterior transformada de Fourier real de largo 4096 (2048 canales a la salida). Luego se utilizan solo los canales pares, se separa la parte real e imaginaria y estas se guardan en las ocho memorias en amarillo a la derecha (4 memorias de largo 512 para cada IF). Los valores guardados en estas memorias servirán para calcular las constantes dadas desde las ecuaciones (4.1) hasta (4.4).

La figura 4.10 muestra el modelo en Simulink del espectrómetro que realiza el cálculo de rechazo de banda lateral, implementando el proceso explicado en la sección 2.4. Más precisamente este modelo implementa el mismo esquema mostrado en la figura 3.7 en la etapa de *backend*. Las constantes calculadas a partir del modelo calibrador son cargadas en este modelo dentro de los bloques VCM (*Vector Complex Multiplier*) en azul. Antes de grabarse en las memorias, los espectros son acumulados en los bloques *vacc* (*vector accumulator*). La FFT es también real y de largo 4096.

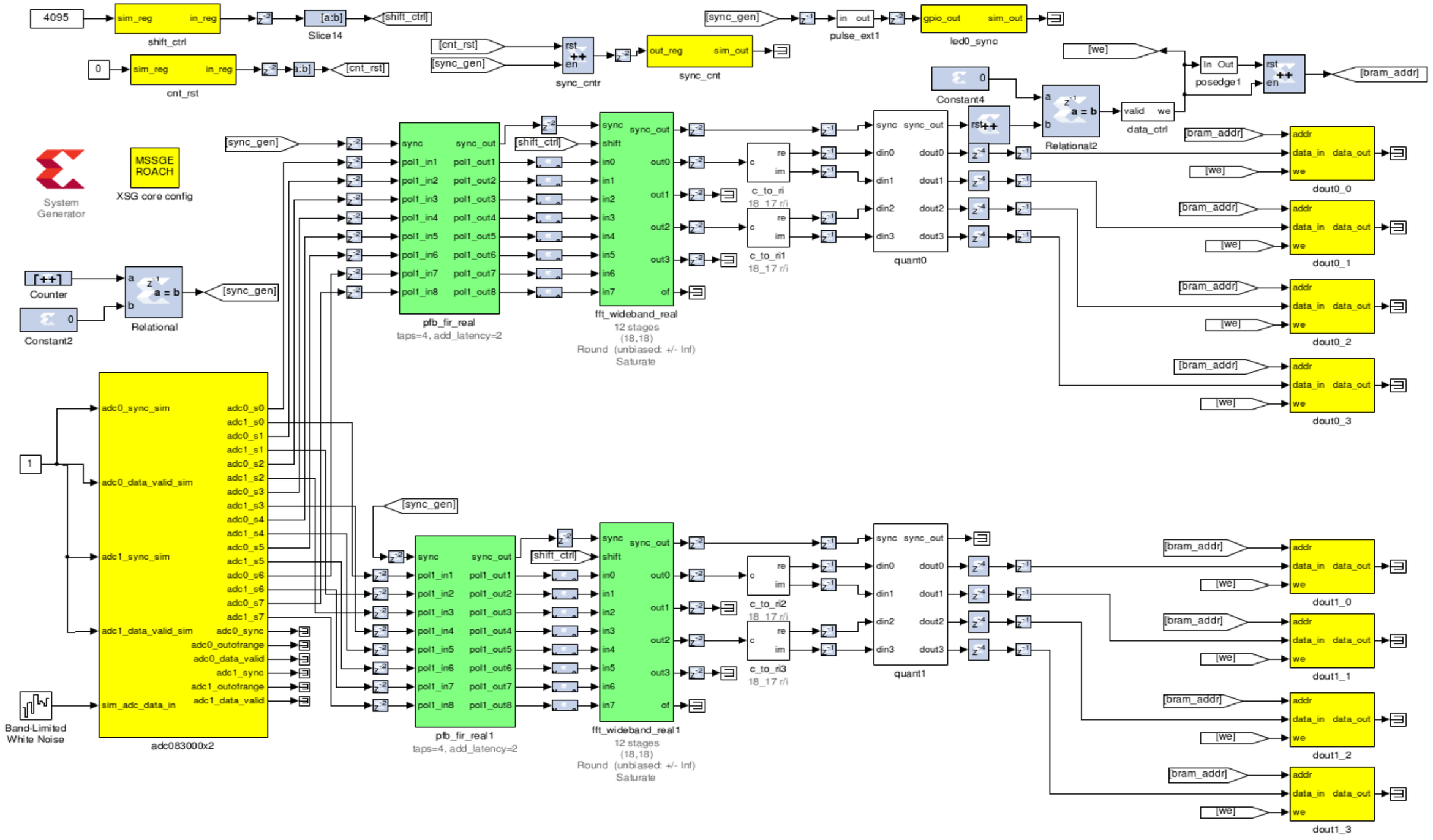


Figura 4.9: Modelo Simulink del calibrador. Los datos a la salida del ADC son procesados en una etapa llamada PFB (Polyphase Filter Bank), en donde se filtran y se transforman al domino de la frecuencia. Luego son amplificados y reinterpretados, para finalmente grabarse en las memorias BRAM dout (data output).

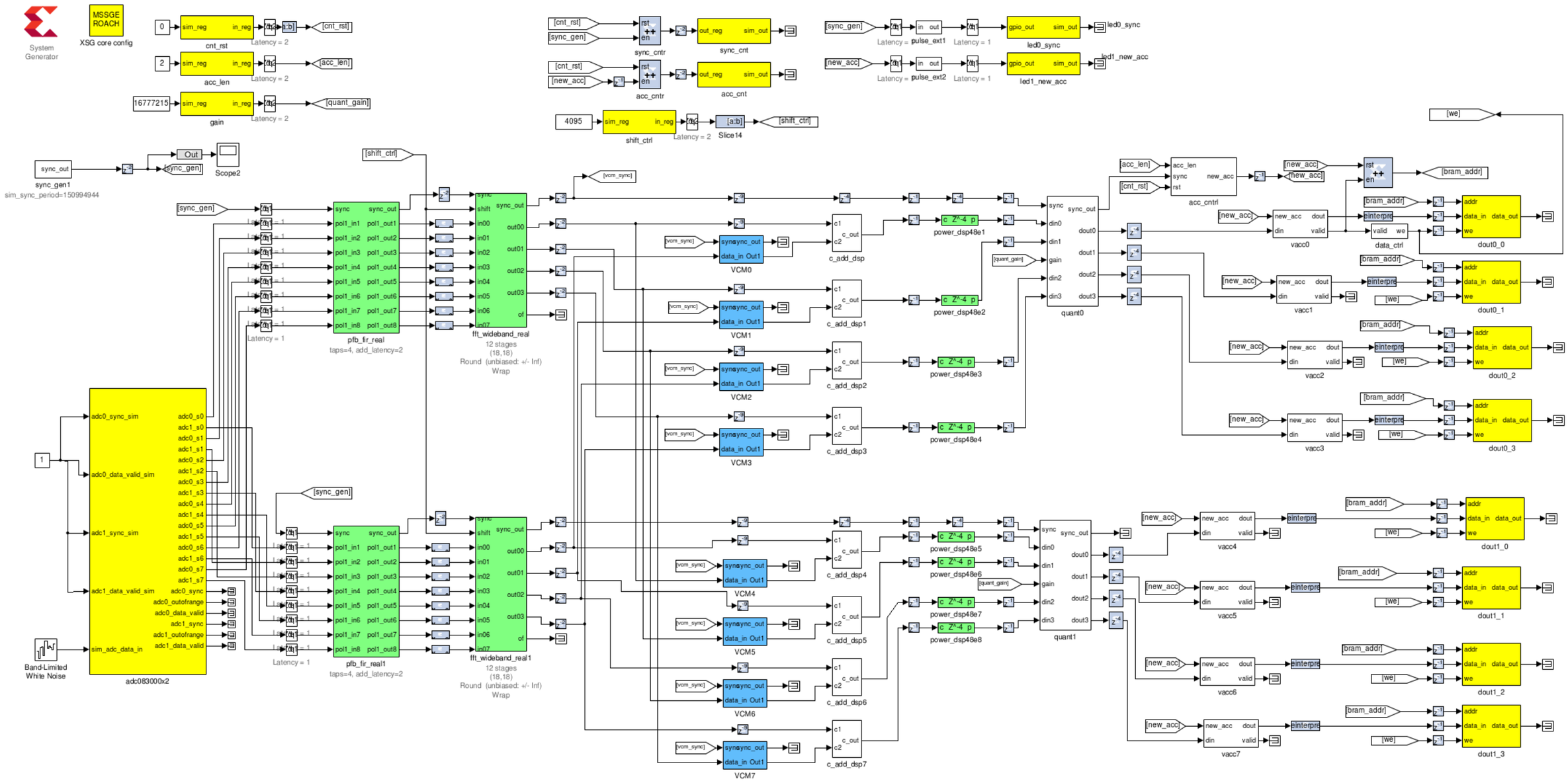


Figura 4.10: Modelo del espectrómetro que realiza rechazo de banda lateral. Luego de pasar por el PFB, los datos son multiplicados por las constantes C_1 y C_3, obtenidas con los datos tomados por el modelo calibrador, que se cargan dentro de los bloques VCM (Vector Complex Multiplier). Luego se calcula el modulo al cuadrado por medio de los bloques power_dsp48. Finalmente se realiza una acumulación de los espectros dentro de los bloques vacc. El largo de acumulación es indicado por el usuario al cargar el modelo en ROACH.

4.2. Diseño de un Zoomming Spectrometer

En esta sección se mostrara el procedimiento para el diseño de un espectrómetro simple de 2048 canales procesando una señal de 500 MHz de ancho de banda a la cual se le desea hacer un acercamiento o zoom a una zona específica de su espectro con ancho de banda 8 veces más pequeño, utilizando los mismos 2048 canales y manteniendo lo mayor posible las características del espectrómetro en términos de rango dinámico.

4.2.1. Especificaciones de diseño

Se tomará como punto de partida un modelo preexistente de un espectrómetro simple con 2048 canales, con acumulador y que contiene el bloque de control de datos diseñado en la sección anterior. Se desea realizar *zoom in* de 8x a una zona del espectro de la señal procesada por el ADC. El ADC procesa una señal de 500 MHz. Como se muestra en la Figura 4.11, en términos generales un zoom toma una fracción BW/M del ancho de banda original y utiliza todos los canales para observar ese ancho de banda.

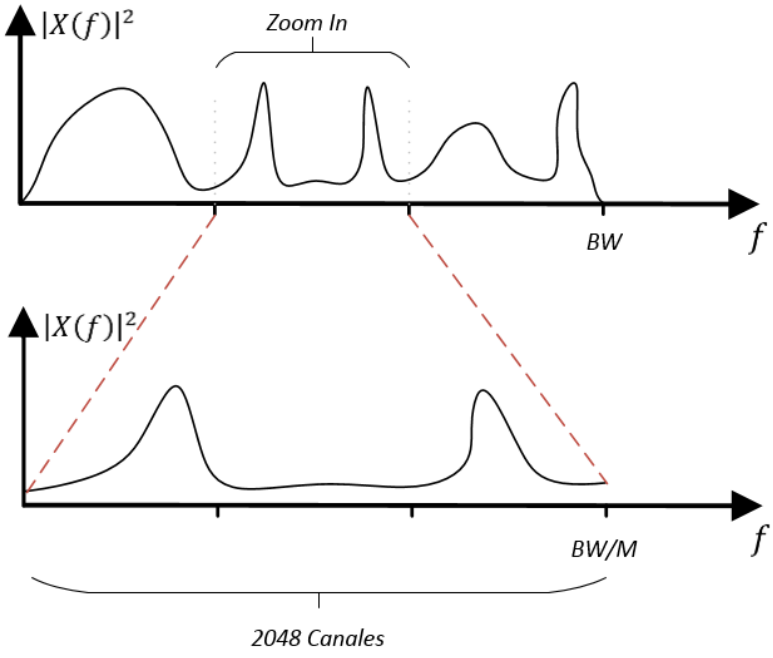


Figura 4.11: Ilustración de un ''zoom in'' a una fracción del espectro ocupando todos los canales de la FFT.

De esta manera, las especificaciones de diseño son:

1) Seleccionar un ancho de banda ocho veces más pequeño que el procesado por el ADC.
2) Ocupar todos los canales espectrales para observar ese ancho de banda.
3) Mantener el rango dinámico del espectrómetro
4) Tener una resolución temporal similar a la obtenida con el espectrómetro original.

4.2.2. Elección de bloques

A continuación se detallan los distintos bloques utilizados para el diseño del *zooming spectrometer*, explicando en detalle el funcionamiento de cada uno y su utilidad dentro del diseño final.

Dec_fir

Este es un bloque que realiza una decimación, es decir filtra la señal por medio de un filtro FIR y luego realiza un *downsampling* de orden 8, reduciendo así el ancho de banda a un octavo del original. Los coeficientes del filtro FIR implementado se cargan dentro de los parámetros de entrada del bloque como un vector. Este bloque pertenece a la librería de CASPER. El número de parámetros del filtro implementado en este bloque debe ser múltiplo del número de entradas de este.

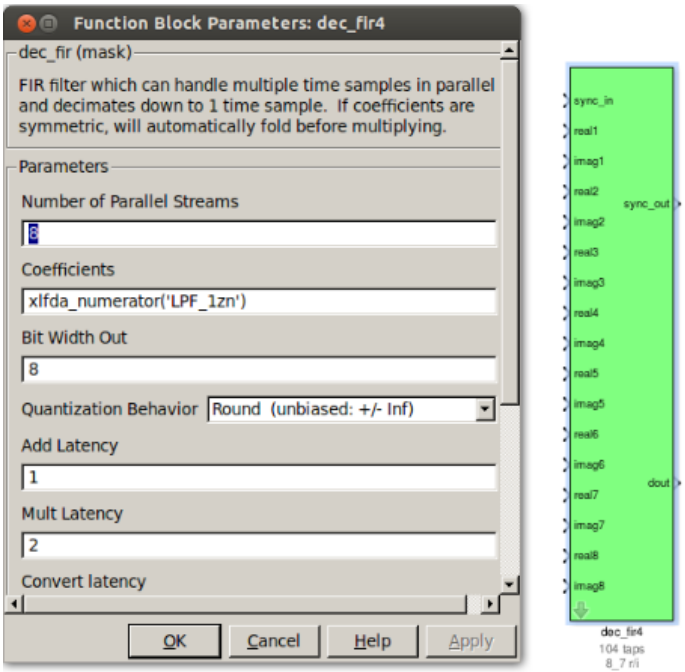


Figura 4.12: Bloque dec_fir que realiza decimación implementando un filtro FIR.

Como se muestra en la figura 4.12, este bloque se utiliza con ocho entradas paralelas reales y ocho imaginarias. Como la señal en cuestión es real, se utilizará solo las entradas reales. En *Coefficients* se indica el vector con los coeficientes del filtro FIR. En el puerto de salida *dout*, se tiene un real y un imaginario concatenados en un *unsigned_fix_16_0*. Solo se utiliza la parte real de este.

FDATool

Este bloque pertenece a *Xilinx* y sirve para diseñar el filtro FIR deseado. Se debe indicar si el filtro a diseñar es pasa bajo, pasa alto o pasa banda. Se indica la frecuencia de muestreo y en base a eso se indican las frecuencias de corte. Además se puede elegir la atenuación deseada en todos los rangos, ya sea en la banda de paso o en las bandas de atenuación. Cuando se ingresan correctamente los parámetros de diseño, se procede a presionar el botón *Design Filter* que generará el filtro deseado (ver Figura 4.13). Las constantes de este filtro se llaman con el siguiente comando en el *workspace* de MATLAB:

"xlfda_numerator('filter_name')",

donde *filter_name* es el nombre del filtro diseñado. Este bloque debe estar en el mismo nivel que el bloque *dec_fir*. En la Figura 4.13 se muestra la interfaz gráfica de usuario (GUI) de este bloque de diseño.

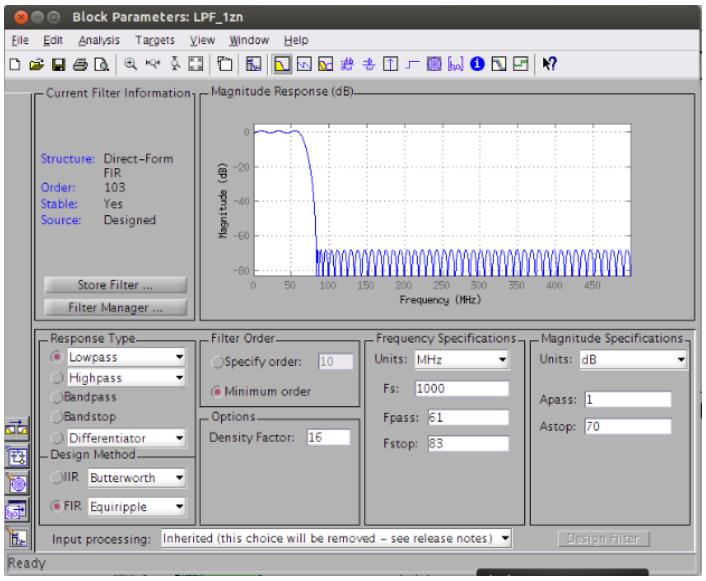


Figura 4.13: GUI del bloque FDATool para diseño de filtros FIR.

C_to_ri

Este bloque simplemente toma un número complejo, usualmente representado en forma de *unsigned_fix*, y es separado en su parte real e imaginaria cada una representada como *signed_fix*. Se utilizará a la salida del bloque *dec_fir*.

Fft_biplex_real_2x

Este bloque es capaz de calcular 4 FFT reales independientemente y las multiplexa en dos puertos de salida. Se utilizará este bloque para mostrar cuatro acercamientos simultáneamente, cada uno en un gráfico distinto.

4.2.3. Diseño final y detalles de funcionamiento

En esta sección se describe específicamente el diseño del *zooming spectrometer*, se explica la interconexión de los bloques, y se muestran los filtros diseñados.

El diseño final consiste en ocho filtros con banda de paso de aproximadamente 62.5 MHz, lo que corresponde a una octava parte del ancho de banda original. Los recursos del FPGA no son suficientes para soportar la implementación simultánea de estos ocho filtros, por lo que se realizaron dos modelos idénticos, cada uno con cuatro filtros. De esta manera el usuario puede visualizar cuatro acercamientos o zoom simultáneos, correspondientes a los cuatro filtros contenidos en el diseño.

En la figura 4.14 se muestra el diseño final del espectrómetro con zoom. Se aprecia en azul un subsistema llamado *dec* que contiene cuatro filtros. Este bloque tiene cinco salidas. Una corresponde a la señal *sync* y las otras corresponden a los espectros de cada uno de los acercamientos. En cada una de las cuatro salidas de datos, se obtiene una reducción de ocho veces el ancho de banda. Luego estas líneas de datos entran a una *FFT_biplex_real*, la cual calcula cuatro transformadas de Fourier (una para cada línea de entrada) y el resultado es entregado en dos líneas paralelas. En la primera línea de salida de la *FFT_biplex_real* se entregan primero los 2048 puntos correspondientes a la entrada *pol0*, y los siguientes 2048 corresponden a la entrada *pol2*. La segunda salida de este bloque FFT es igual que la primera pero considerando las entradas *pol1* y *pol3*. En la etapa final, el procedimiento es similar a un espectrómetro simple, los datos son acumulados de acuerdo al largo indicado en el registro *acc_len* y posteriormente guardados en memoria. Este diseño cuenta con el bloque de control de datos diseñado previamente.

En la figura 4.15, se puede apreciar lo que hay debajo de la máscara del bloque *dec* de color azul. Se alimenta la parte imaginaria en la entrada de los *dec_fir* con ceros, para así utilizar solo la parte real a la salida. Los bloques *FDATool* se encuentran al lado de sus respectivos bloques decimadores, y contienen los coeficientes de cada filtro FIR. A la salida de cada bloque decimador, el bloque *c_to_ri* divide el dato en su parte real e imaginaria. La parte imaginaria es descartada mientras que la parte real corresponde a la entrada de un bloque PFB. Las bandas de paso de los filtros corresponden a las zonas de Nyquist correspondientes a una frecuencia de muestreo de 0,125 GSPS. De esta manera la señal al ser decimada queda en una de las zonas de Nyquist y se produce una conversión digital hacia

abajo. Si la señal decimada queda en una zona par de Nyquist el vector que contiene la información del espectro se invierte en una línea de comando de *Python*. Se diseñaron dos modelos idénticos salvo por los filtros. El primer modelo se llama *espectrometro_8x_zoom_pm.slx* y contiene los filtros correspondientes a las cuatro primeras zonas de Nyquist. El segundo modelo denominado *espectrometro_8x_zoom_sm.slx* contiene los filtros correspondientes desde la quinta hasta la octava zona de Nyquist. Estos dos modelos lucen como se muestra en la figura 4.14. Los filtros diseñados son de orden 104 cada uno.

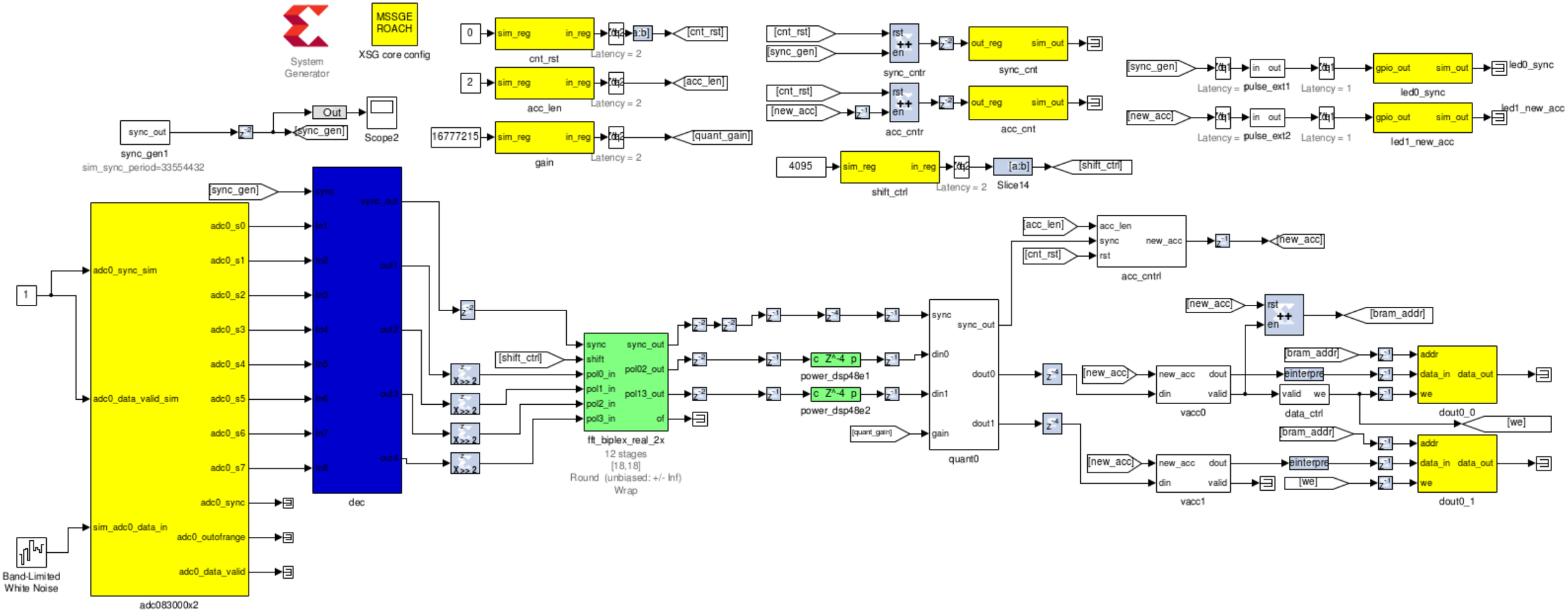


Figura 4.14: Diseño del Zooming Spectrometer, con bloque dec en azul. Se utiliza una fft_biplex en donde los datos son multiplexados en 2 líneas paralelas. Las memorias dout son de largo 4096.

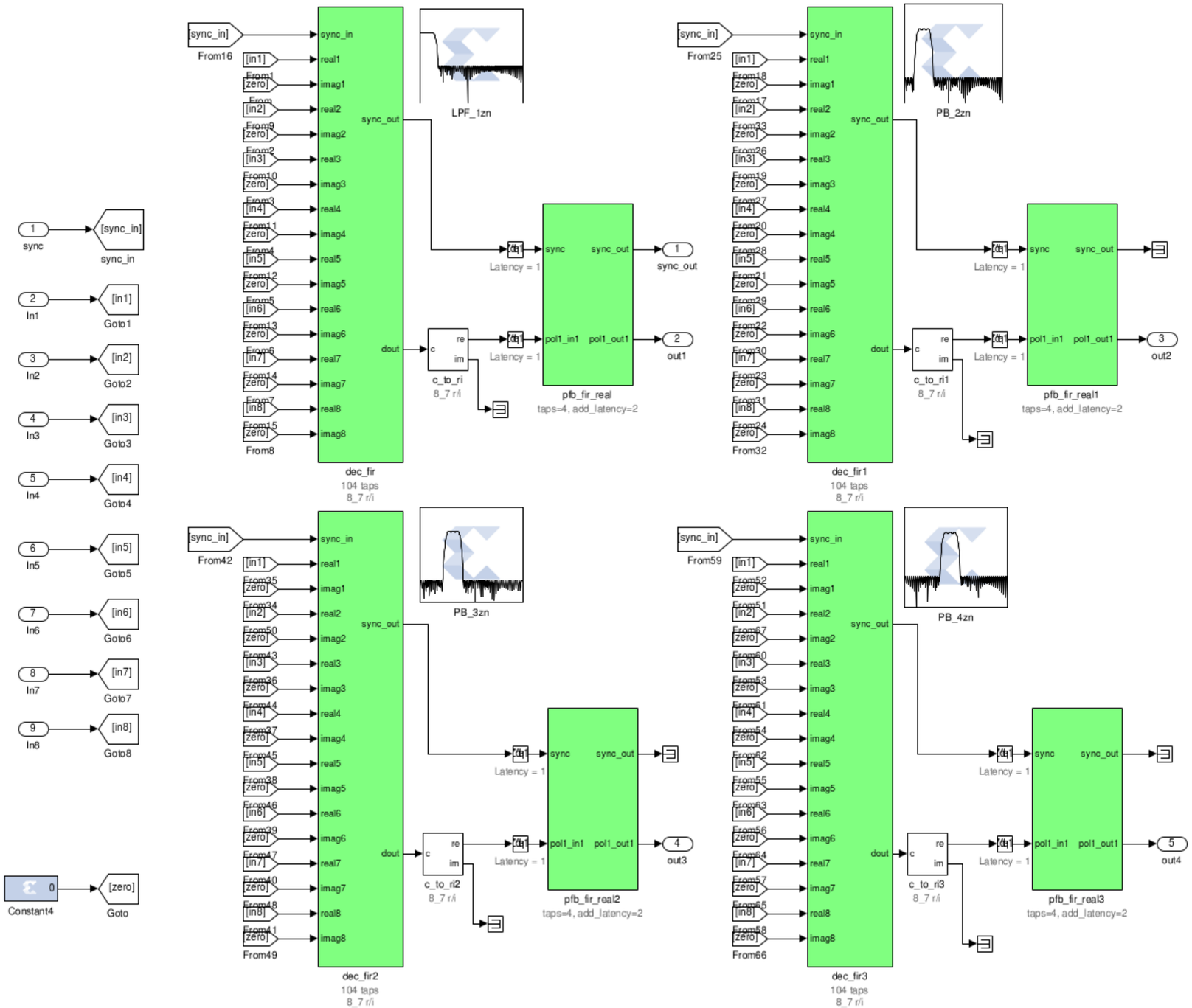
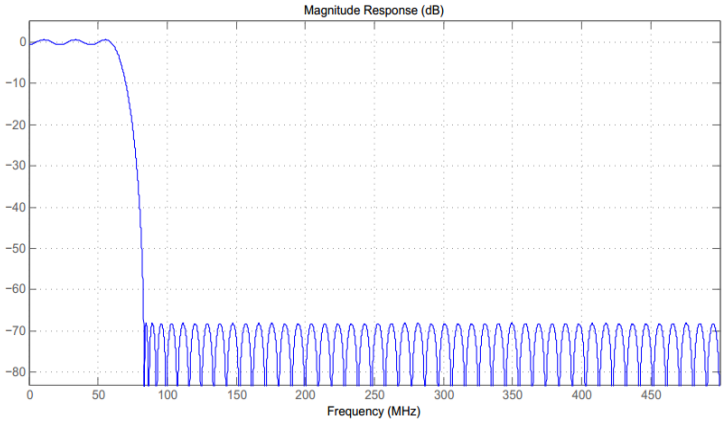


Figura 4.15: Bloque dec de la figura 4.14 que implementa cuatro filtros en la primera mitad del ancho de banda. Para la segunda mitad (desde 250 a 500 MHz) se utiliza otro modelo idéntico implementando los cuatro filtros restantes.

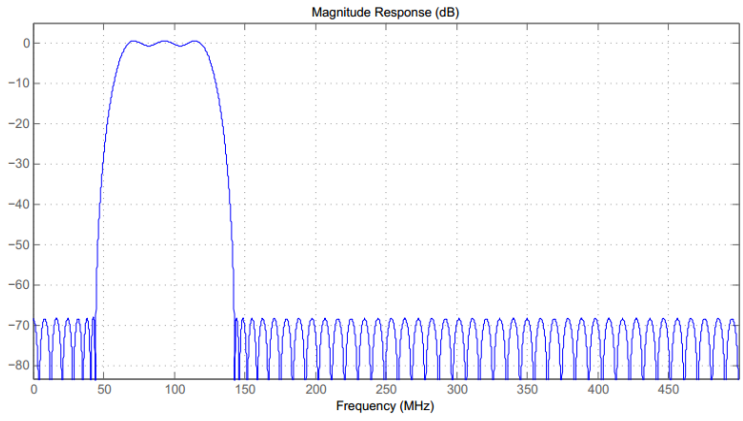
En la tabla 4.1 se muestran las frecuencias de atenuación y de paso, además de las atenuaciones en las bandas de detención. En la figura 4.16 se muestran las respuestas en magnitud de todos los filtros. Debido a que el orden de los filtros debe ser múltiplo de 8, no fue posible encontrar un filtro con 70 dB de rechazo para el filtro HPF_8zn. Por lo tanto, 57 dB de rechazo fue lo mejor que se pudo encontrar para este bloque cumpliendo la restricción de orden de este.

Filtro	$f_{1stop}[MHz]$	$f_{1pass}[MHz]$	$f_{2pass}[MHz]$	$f_{2stop}[MHz]$	$A_{stop1}[dB]$	$A_{stop2}[dB]$
LPF_1zn	-	-	61	83	-	70
PB_2zn	42.5	72.5	120.5	142.5	70	70
PB_3zn	105	135	183	205	70	70
PB_4zn	167.5	197.5	245.5	267.5	70	70
PB_5zn	230	260	308	330	70	70
PB_6zn	292.5	322.5	370.5	392.5	70	70
PB_7zn	355	385	433	455	70	70
HPF_8zn	424	446	-	-	57	-

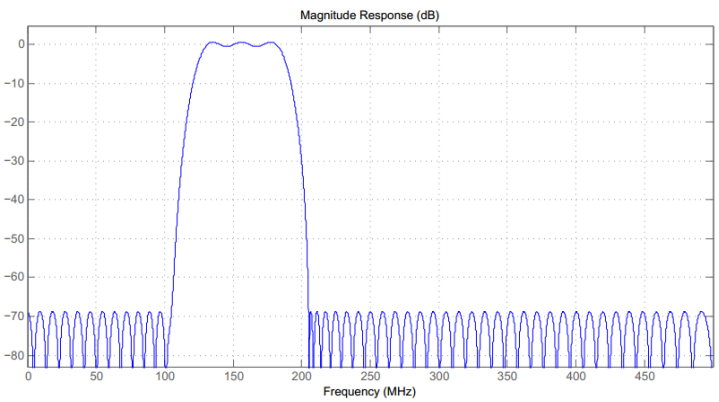
Tabla 4.1: Frecuencias de paso y detención de los ocho filtros diseñados y sus respectivas atenuaciones.



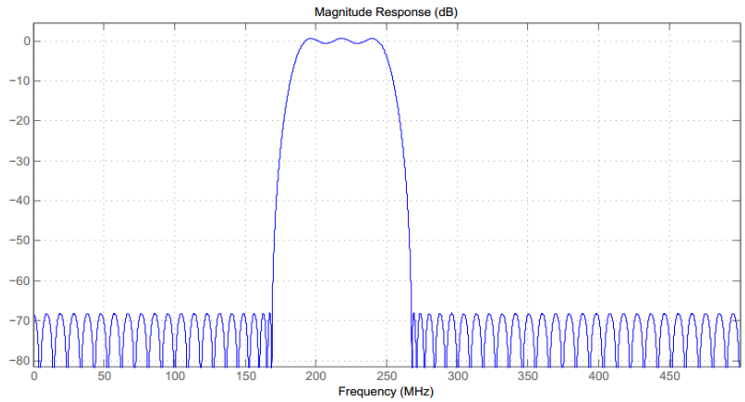
(a)



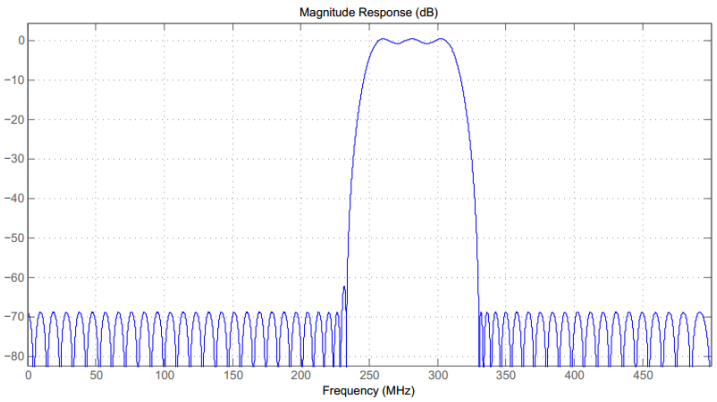
(b)



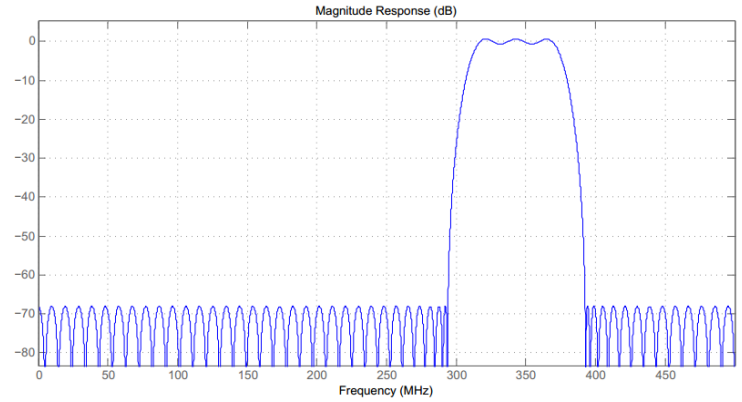
(c)



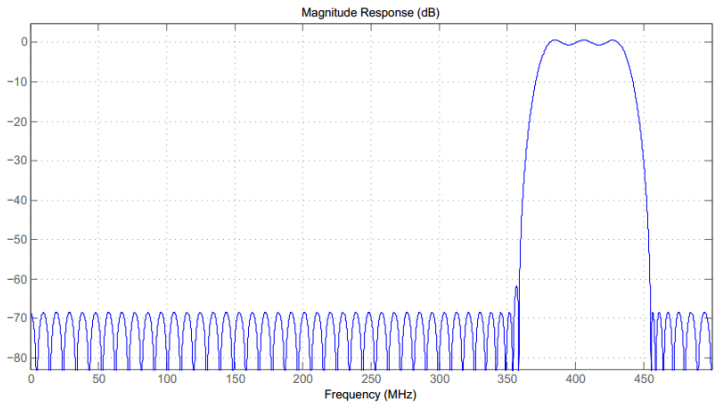
(d)



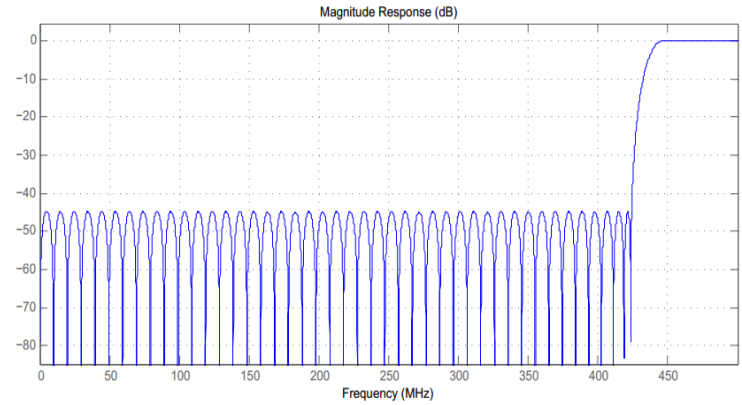
(e)



(f)



(g)



(h)

Figura 4.16: Respuesta en Magnitud de los 8 Filtros diseñados. (a): LPF_2zn, (b): PB_2zn, (c): PB_3zn, (d): PB_4zn, (e): PB_5zn, (f): PB_6zn, (g): PB_7zn, (h): HPF_8zn.

Un aspecto importante en el diseño de estos filtros es que la banda de paso no concuerda exactamente con las zonas de Nyquist de la señal decimada. Como se muestra en la figura 4.17 el filtro PB_2zn que se implementa para observar la segunda zona de Nyquist comienza a atenuar en 72,5 y 120,5 MHz y finaliza atenuando hasta 70 dB en 42,5 y 142,5

MHz. El problema que conlleva esto es que si la señal a ser filtrada tiene contenido energético considerable entre 42,5 y 62,5 MHz, esa porción de la señal no será atenuada completamente y se traspasará a la segunda zona de Nyquist al ser decimada la señal por efecto *aliasing*. El mismo análisis se realiza en la parte superior del filtro donde si la señal tiene contenido energético entre 125 y 142,5 MHz, esos 17,5 MHz se traspasarán a la porción superior de la segunda zona de Nyquist perdiéndose información en esa zona. Es decir si se da el caso anterior existe una porción central en la segunda zona de Nyquist de 25 MHz de ancho en la cual se asegura la no existencia de efecto *aliasing*. Sin embargo si la señal no tiene contenido energético considerable en las bandas anteriormente expuestas, entonces se asegura un zoom sin *aliasing* en toda la zona de Nyquist. La razón de no realizar filtros más angostos que se ajustasen mejor a las zonas de Nyquist se debe únicamente a que el orden del filtro sube rápidamente mientras más angosto se diseñe este, implicando un costo mucho mayor en *hardware*. Todos los filtros pasa banda diseñados en este trabajo cubren su zona de Nyquist correspondiente manteniendo las relaciones dadas en la figura, razón por la cual solo se explica este único caso.

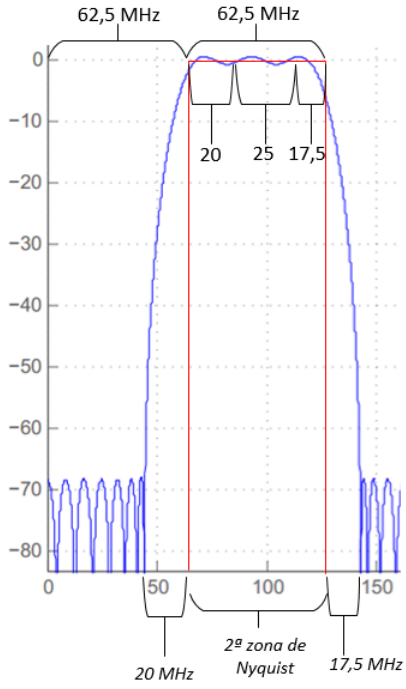


Figura 4.17: Desajuste en magnitud entre el filtro PB_2zn y su respectiva zona de Nyquist.

Otro aspecto importante de estos filtros es que la respuesta de la banda de paso no es plana, obteniéndose distorsiones menores a 1 decibel. Esto puede distorsionar muy levemente la forma del espectro al realizarse el zoom.

4.3. Rutinas de Medición

A continuación se explican los distintos códigos en Python utilizados para realizar las medidas de los espectros, explicando en detalle las rutinas que realizan para tomar las medidas de los datos. Se explicarán también algunas líneas de código muy precisas que son críticas para entender el proceso completo. Todos los códigos y modelos adicionales que se explican en esta sección se pueden encontrar en el apéndice.

a) Rechazo de banda lateral utilizando bloque de control de datos

Para realizar la medición de amplitud y fase de ambas IF se utiliza un código Python llamado *spectrometer_64bits_float_s12_dif_ed_sin_acc_data_upper_low2_dc.py*, que básicamente realiza la medición de las fases $\emptyset_1$ y $\emptyset_2$, y de las magnitudes V_1 y V_2 de la figura 4.1. Esta medición se realiza para cada canal par de la FFT, completando 1024 mediciones en USB y 1024 en LSB. Este código se ejecuta junto con el archivo BOF correspondiente a la figura 4.9 (calibrador). Para esto, el código realiza un barrido de frecuencia con un ciclo *for*. En cada ciclo se introduce un tono puro a la entrada RF del *front end* y se llama a la función *get_data()* encargada de la lectura de memorias. Dentro de esta función se escriben los registros en el diseño del bloque de control de datos. Esta función se resume a continuación.

"def get_data():

fpga.write_int('data_ctrl_lec_done',0) (se escribe el registro *lec_done* con 0 indicando estado de lectura no realizada)

fpga.write_int('data_ctrl_sel_we',1) (Se escribe registro *sel_we* con un 1 indicándole al bloque *data_ctrl* la solicitud de lectura)

Lectura de Memorias. (Se leen las memorias, se reordenan los datos para entregarlos como un arreglo de largo 2048 con los valores del espectro)

fpga.write_int('data_ctrl_lec_done',1) (Se escribe el registro *lec_done* con un 1 indicándole al bloque *data_ctrl* que la lectura fue realizada)

fpga.write_int('data_ctrl_sel_we',0) (Se escribe el registro *sel_we* con un 0 para que todas las señales dentro del *data_ctrl* vuelvan al estado inicial)

return Espectro (La función finaliza retornando un arreglo con los valores del espectro)*"*

Una vez realizado el cálculo de amplitud y fase, otro código de Python llamado *spectrometer_64bits_float_s12_SRR_measurement_agi_prueba.py* carga las constantes calculadas previamente en el modelo de la figura 4.10, empleando las formulas definidas por las ecuaciones (4.3) y (4.4). Una vez cargadas estas contantes en el modelo, se realiza un barrido de frecuencias con tonos puros mediante un *for* desde el LSB hasta el USB,

calculando en cada ciclo el SRR como la diferencia entre el valor máximo en el espectro USB y el valor máximo en el espectro LSB.

b) Espectrómetro con zoom

Para observar el espectro de una señal de 500 MHz se utiliza un modelo previamente diseñado llamado *espectrómetro_simple_2048ch.slx* junto con el código *spectrometer_64bits_float_s12_dctrl.py*. Luego al realizar zoom, se dividen los 500 MHz en 8 zonas de 62,5 MHz de ancho de banda, definiendo las 8 zonas de Nyquist de la señal decimada. Si la zona a la que se le desea realizar zoom se encuentra entre 0 y 250 MHz (primera mitad) se utiliza el modelo *espectrometro_8x_zoom_pm.slx* (figura 4.14) junto con el código *espectrómetro_down8.py*. Si la zona a observar queda por sobre 250 MHz se utiliza el modelo *espectrometro_8x_zoom_sm.slx* (figura 4.14). La correspondencia entre la zona de Nyquist seleccionada y el filtro utilizado se muestra en la tabla 4.2 para zonas entre 0 y 250 MHz y en la tabla 4.3 para zonas entre 250 y 500 MHz.

Zona	Banda Seleccionada (f<250 MHz)	Filtro Implementado
1	0-62,5 [MHz]	LPF_1zn
2	62,5-125 [MHz]	PB_2zn
3	125-187,5 [MHz]	PB_3zn
4	187,5-250 [MHz]	PB_4zn

Tabla 4.2: Correspondencia entre las zonas de Nyquist y los filtros implementados entre 0 y 250 MHz.

Zona	Banda Seleccionada (f>250 MHz)	Filtro Implementado
5	250-312,5 [MHz]	PB_5zn
6	312,5-375 [MHz]	PB_6zn
7	375-437,5 [MHz]	PB_7zn
8	437,5-500 [MHz]	HPF_8zn

Tabla 4.3: Correspondencia entre las zonas de Nyquist y los filtros implementados entre 250 y 500 MHz.

Capítulo 5

Resultados

5.1. Bloque de control de datos

En esta sección se muestran las mejoras introducidas por el bloque de control de datos en el proceso de separación de banda lateral. Se pretende contrastar con los resultados obtenidos sin control de datos para así cuantificar las mejoras producidas.

5.1.1. Pruebas y resultados

En las pruebas realizadas en esta sección se ocupa el receptor *dual sideband* con un oscilador local en 3,2 GHz y una RF entre 2,7 y 3,7 GHz. En la figura 5.1 se muestra un ejemplo del espectro de una de las bandas laterales para una RF de 3329,394 MHz, lo que corresponde a una IF de 3200-3329,394=129,394 MHz. En estas pruebas se utiliza un largo de acumulación igual a 131072, es decir, se acumula o suma esa cantidad de espectros dentro del bloque *vacc* y el resultado es graficado. Se visualiza en la figura 5.1 la presencia de un tono en 250 MHz de unos 30 decibeles. Sin embargo este tono es artificial, es decir se genera en la etapa de digitalización de la señal. La presencia de este tono es indeseada, sobre todo en lo que respecta al cálculo de SRR, puesto que el valor máximo del espectrómetro para la banda suprimida será igual a este tono y no el tono suprimido (como se desea). Por lo tanto se procederá a eliminarlo.

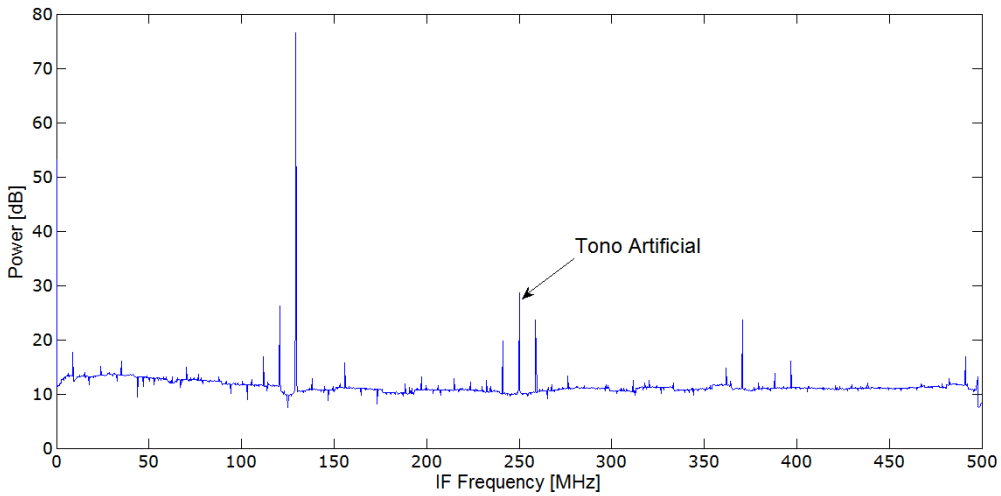


Figura 5.1: Espectro de 500 MHz de ancho con un tono en 129,3 MHz. Se aprecia un tono artificial en la mitad del espectro.

A continuación se muestran los resultados del modelo calibrador, para los casos sin y con bloque de control de datos, procediendo como se indica en la sección 4.3 a). Se focaliza la atención en las mejoras producidas por el bloque de control de datos. En adelante los las sub figuras con sufijo (a) representan resultados tomados sin control de datos mientras que las sub figuras con sufijo (b) representan resultados con control de datos. En los gráficos de diferencia de fase, se grafican en realidad los ángulos $\emptyset_{USB}$ y $\emptyset_{LSB}$ definidos en las ecuaciones (4.1) y (4.2).

En la figura 5.2 se observa la razón de amplitud de ambas ramas calculada por ambos modelos calibradores (sin y con *data_ctrl*) para las frecuencias en LSB y USB. En la figura 5.2 (a) se ve la razón de amplitud entre la rama i y q sin control de datos. En la figura 5.2 (b) se observa la razón de amplitud utilizando control de datos. Se puede notar que ambos gráficos siguen una misma tendencia, variando entre 0,95 y 1,2. Sin embargo en 5.2 (b) se aprecia una curva con menos ruido que en (a).

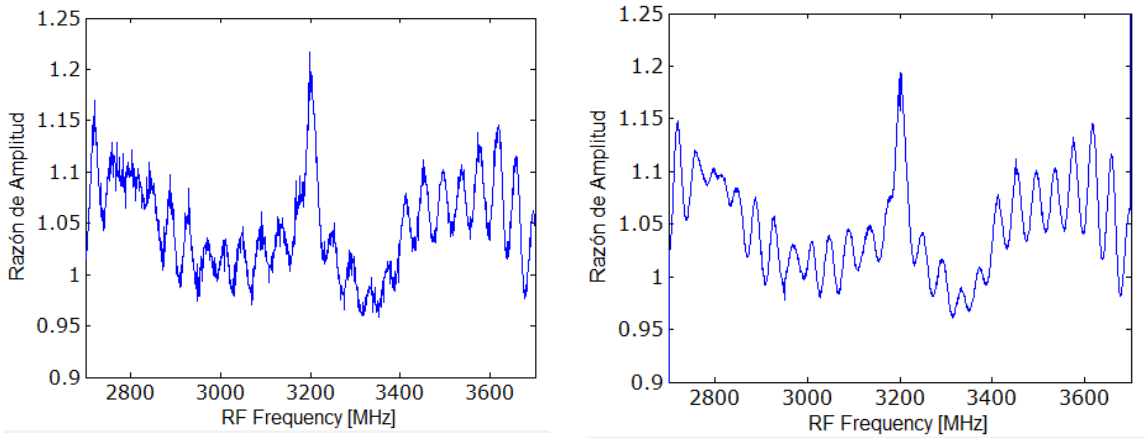
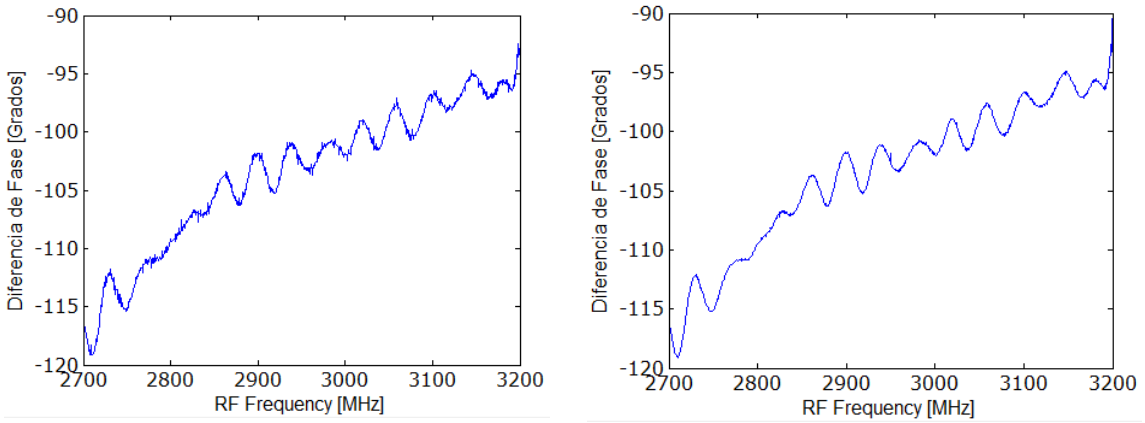


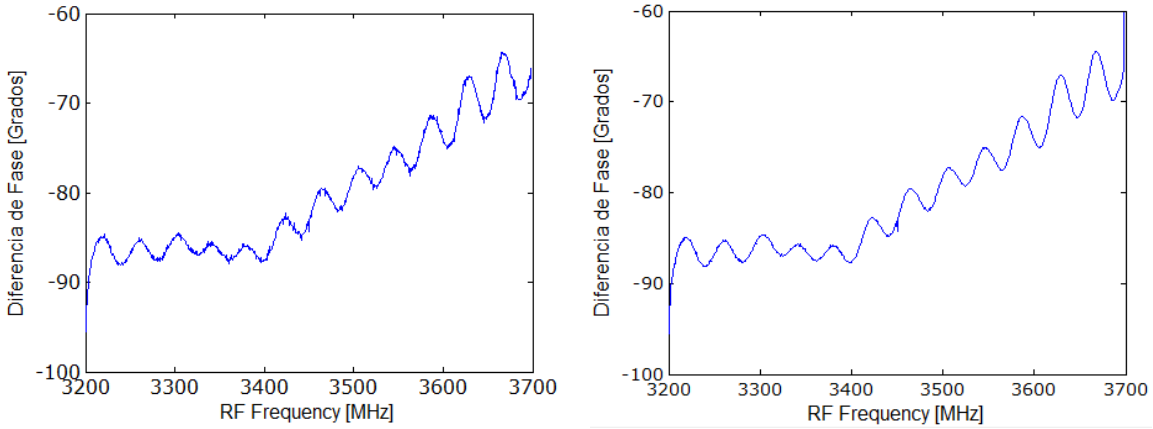
Figura 5.2: Razón de amplitud entre las dos ramas del receptor. (a) Sin control de datos, (b) con control de datos.

En la figura 5.3 se aprecia la diferencia de fase entre las ramas i y q del receptor para LSB. En la figura 5.3 (a) se aprecia más ruido que en (b), y ambas siguen un mismo patrón variando entre -120 y -90 grados. Resultados muy similares se obtienen en USB para la diferencia de fase como se muestra en la figura 5.4.



(a) *(b)*

Figura 5.3: Diferencia de fase en LSB entre las dos ramas del receptor. (a) Sin control de datos, (b) con control de datos.



(a) *(b)*

Figura 5.4: Diferencia de fase en USB entre las dos ramas del receptor. (a) Sin control de datos, (b) con control de datos.

Luego de medidos los datos mostrados previamente, se procede a calcular el SRR con el modelo respectivo cargando los datos previamente. El cálculo de SRR se realiza primero considerando los datos de calibración sin control de datos, y luego con control de datos.

La figura 5.5 muestra el SRR para 1024 puntos. En la figura 5.5 (a) se aprecia el SRR sin control de datos, mientras que la figura 5.5 (b) contiene control de datos. El SRR sin control de datos varía más de 10 decibeles entre distintas frecuencias, mientras que con control de datos la mejora es significativa en ambas bandas laterales.

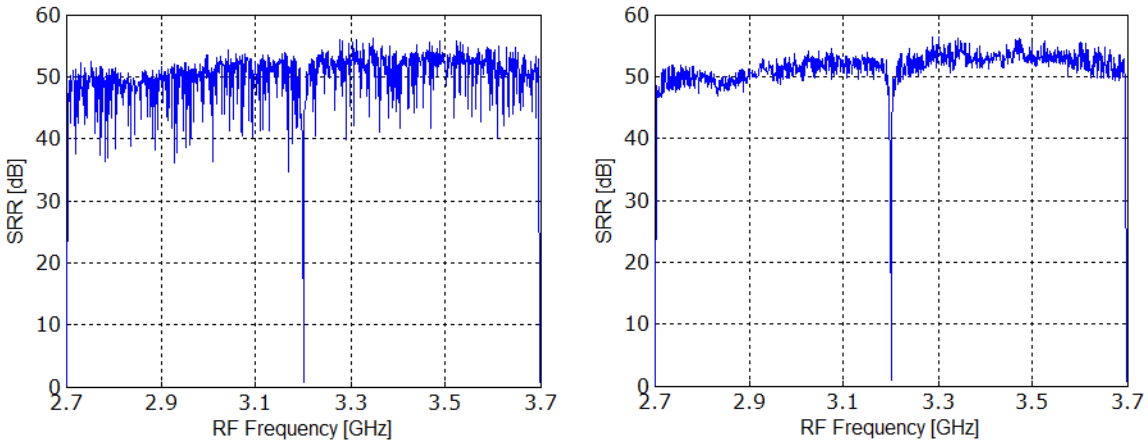


Figura 5.5: Rechazo de banda lateral con oscilador local en 3.2 GHz. (a) Sin control de datos, (b) con control de datos.

A continuación se llevan a cabo nuevamente las calibraciones realizadas anteriormente, pero se adhiere ruido de fase en el oscilador local. Se pretende comprobar que el bloque *data_ctrl* provoca que los datos sean inmunes al ruido de fase, provocando que las diferencias apreciadas en los gráficos anteriores sean agudizadas.

Para agregar ruido de fase en el oscilador local se procede activar la modulación de fase fijando los parámetros de modulación con los siguientes valores.

PhiM Mode: Low noise

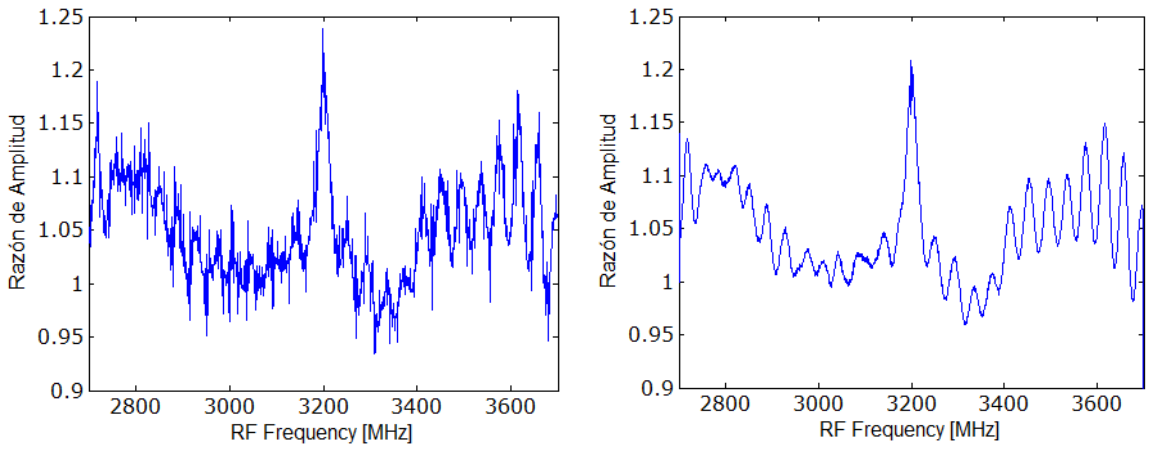
PhiM Deviation = 3 [rad]

PhiM Source LFGen Freq = 1 KHz

LFGen Shape = Sine

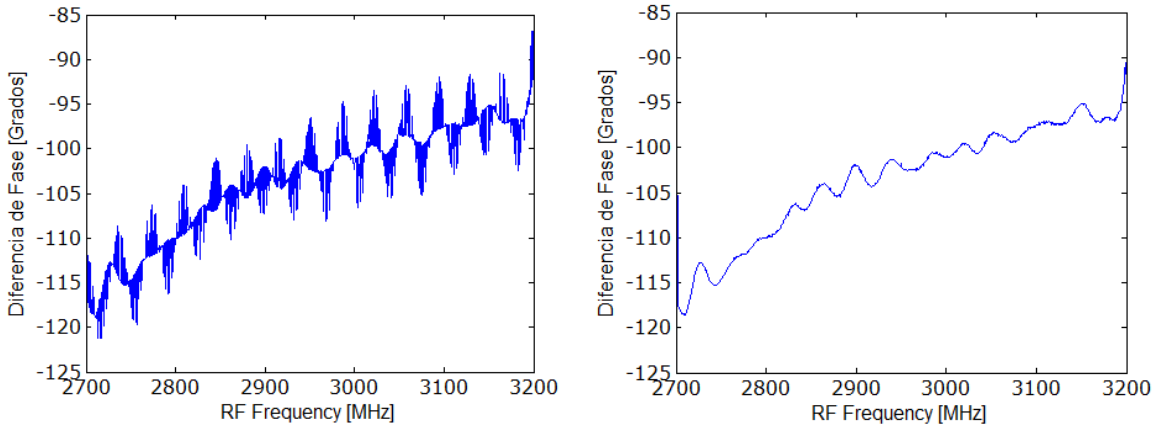
Es decir se contamina la fase de la señal LO con alto ruido de fase, una desviación angular de 3 radianes y una frecuencia de 1 KHz de forma sinusoidal. Se muestran los resultados obtenidos desde la figura 5.6 hasta la figura 5.9.

En la figura 5.6 se aprecia la razón de amplitud. Se ve claramente que la razón de amplitud sin control presenta más ruido que en la figura 5.2 (a). La razón de amplitud para el caso con control de datos se mantiene sin ruido. Algo similar se aprecia en la figura 5.7 y 5.8 pero con respecto a la diferencia de fase en LSB y USB.



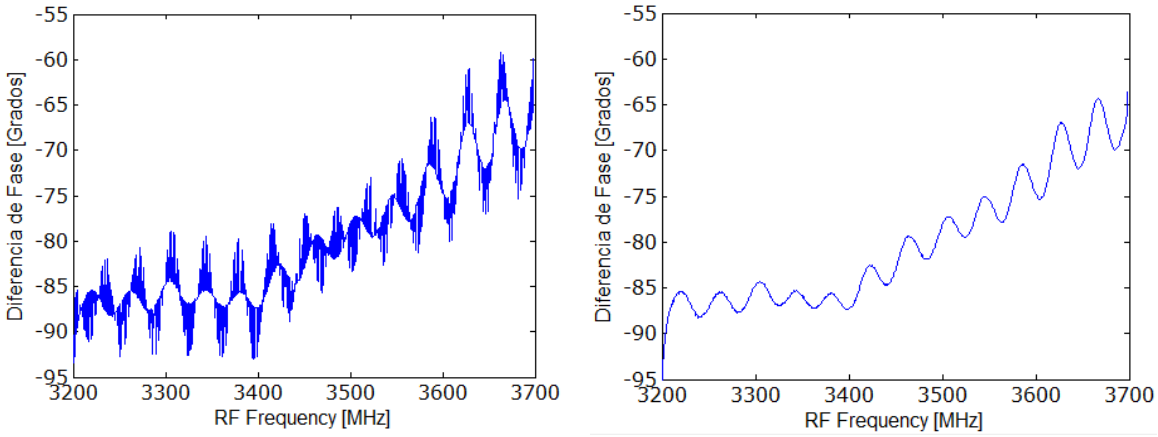
(a) *(b)*

Figura 5.6: Razón de amplitud entre las dos ramas del modelo calibrador con ruido de fase. (a) Sin control de datos, (b) con control de datos.



(a) *(b)*

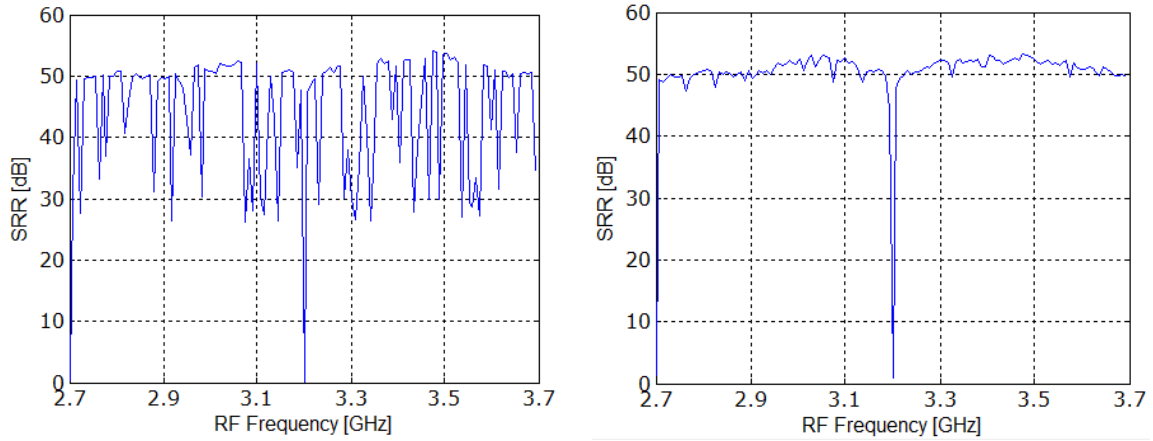
Figura 5.7: Diferencia de fase en LSB entre las dos ramas del modelo calibrador con ruido de fase. (a) Sin control de datos, (b) con control de datos.



(a) *(b)*

Figura 5.8: Diferencia de fase en USB entre las dos ramas del modelo calibrador con ruido de fase. (a) Sin control de datos, (b) con control de datos.

La figura 5.9 muestra el rechazo de banda lateral con 256 puntos. Calculando el SRR con los datos contaminados con ruido de fase, se observa que el control de datos hace que el rechazo de banda sea inmune al ruido de fase, manteniéndose casi igual que en la figura 5.5 (b). Sin embargo comparando la figura 5.9 (a) con 5.5 (a), se hace evidente que el rechazo de banda disminuye en presencia de ruido de fase, variando hasta 20 decibeles entre distintas frecuencias.



(a) *(b)*

Figura 5.9: Rechazo de banda lateral con ruido de fase. (a) Sin control de datos, (b) con control de datos.

5.1.2. Análisis de los resultados

Al haber un bloque de control de datos en el modelo calibrador, los datos entregados en las distintas memorias pertenecen al mismo espectro. Si no hay control de datos entonces los datos en las memorias pueden pertenecer a distintos espectros, por lo tanto se pierde

correlación de fase. Se aprecia claramente que en todos los casos, las calibraciones y SRR logrados con control de datos son mucho menos ruidosas que los resultados sin control de datos.

Al agregar ruido de fase a las señales en la etapa analógica, el bloque de control de datos hace que las mediciones de SRR sean prácticamente inmune a estas variaciones y muestra tener un desempeño óptimo al entregar datos muy limpios y SRR muy regulares.

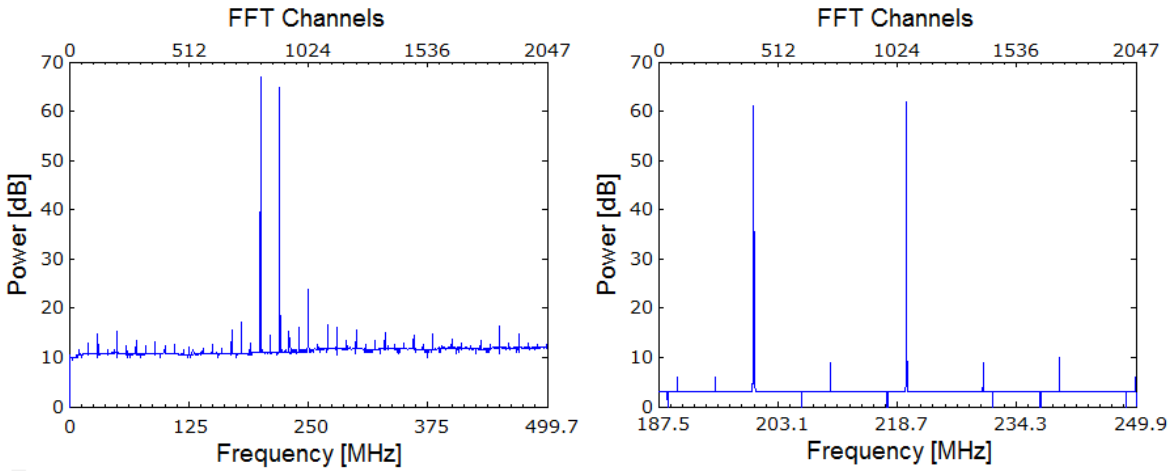
Todo lo anterior demuestra el buen funcionamiento de este bloque, entregando datos simultáneos y con menos ruido. Estos resultados indican la importancia de la sincronía en la comunicación con los distintos comandos dentro de un diseño que corre a alta velocidad. Muchas veces puede suceder que algunos comandos externos corran a velocidades mucho más lentas que el diseño dentro del procesador, y que estas diferencias de velocidad no estén consideradas explícitamente en el diseño.

5.2. Zooming Spectrometer

En esta sección se muestran los resultados del diseño del *zooming spectrometer*. Para esto primero se muestra el espectro de la señal IF de 500 MHz y luego se muestra el acercamiento que se realiza a una zona específica del espectro. En todos los casos el acercamiento se realiza con 2048 puntos, aumentando 8 veces la resolución espectral.

5.2.1. Pruebas y resultados

La figura 5.10 (a) muestra una IF de 500 MHz con 2048 canales, en la que se aprecian 2 tonos en 200 y 220 MHz. El tono en 200 MHz ocupa el canal 819 con 66,8 decibeles y el canal 820 con 39,8 decibeles, mientras que el tono en 220 MHz ocupa el canal 901 y 902 con 64,69 y 25,6 decibeles respectivamente. Al realizar un acercamiento a la zona entre el canal 768 y 1024 (entre 187,5 y 250 MHz) de la figura 5.10 (a), se obtiene el zoom mostrado en la figura 5.10 (b) utilizando un largo de acumulación ocho veces menor que en la figura 5.10 (a), para mantener una resolución temporal similar. En la figura 5.10 (b) el tono en 200 MHz ocupa el canal 408 y 409 con 53 y 61 decibeles respectivamente, mientras que el tono en 220 MHz ocupa el canal 1064 con 61,9 decibeles.



(a) *(b)*

Figura 5.10: Dos tonos en 200 y 220 MHz. (a) IF de 500 MHz de acho, (b) zoom entre 187,5 y 250 MHz. Para que el zoom tenga la misma resolución temporal que el espectrómetro simple, es necesario reducir 8 veces su largo de acumulación.

En la figura 5.11 (a) se muestra un tono en 50 MHz que se encuentra entre los canales 204 y 205, más cerca de este último. Se aprecian también el segundo armónico en 100 MHz, el tercero y cuarto en 150 y 200 MHz respectivamente. Se aprecian también armónicos más altos de menor potencia. Al realizar zoom a la zona entre el canal 256 y 512 (entre 62,5 y 125 MHz), el tono en 50 MHz se fuga a esta zona apareciendo en 75 MHz, pero atenuado más de 20 decibeles (figura 5.11 (b)). En el zoom se aprecia el segundo armónico en 100 MHz correspondiente al canal 1228. Este armónico crece unos 7 dB cuando se realiza el zoom. Lo anterior se explica debido a que en el espectrómetro simple de la figura 5.11 (a) el tono en 100 MHz queda entre medio de dos canales, mientras que al realizar zoom este tono queda muy cerca de un canal, por lo tanto su potencia queda representada casi en su totalidad por ese canal.

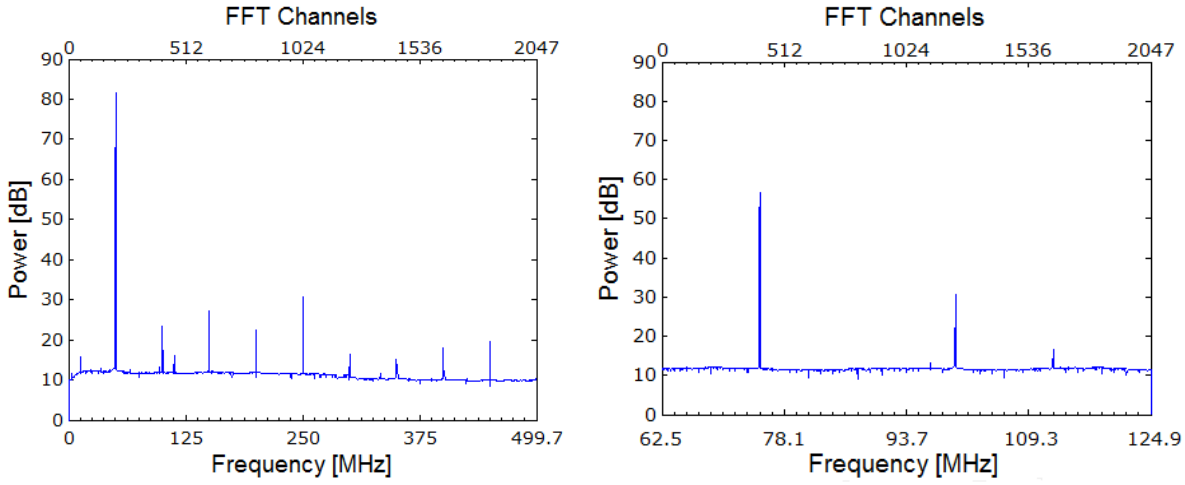


Figura 5.11: Efecto aliasing producto del desajuste entre el filtro y su respectiva zona de Nyquist. (a) IF de 500 MHz de ancho, (b) zoom a la zona entre 62,5 y 125 MHz. En este caso el zoom en (b) se realiza con el mismo largo de acumulación que en (a), manteniendo el piso de ruido, pero disminuyendo 8 veces la resolución temporal.

A continuación se procede a realizar una señal IF pasa banda como la señal mostrada en la figura 5.12 con 2048 canales. Esta señal tiene un contenido energético considerable en ciertas bandas de frecuencia. Se procede a realizar un acercamiento a distintas zonas de esta señal, manteniendo el largo de acumulación, para mostrar el desempeño del *zooming spectrometer*.

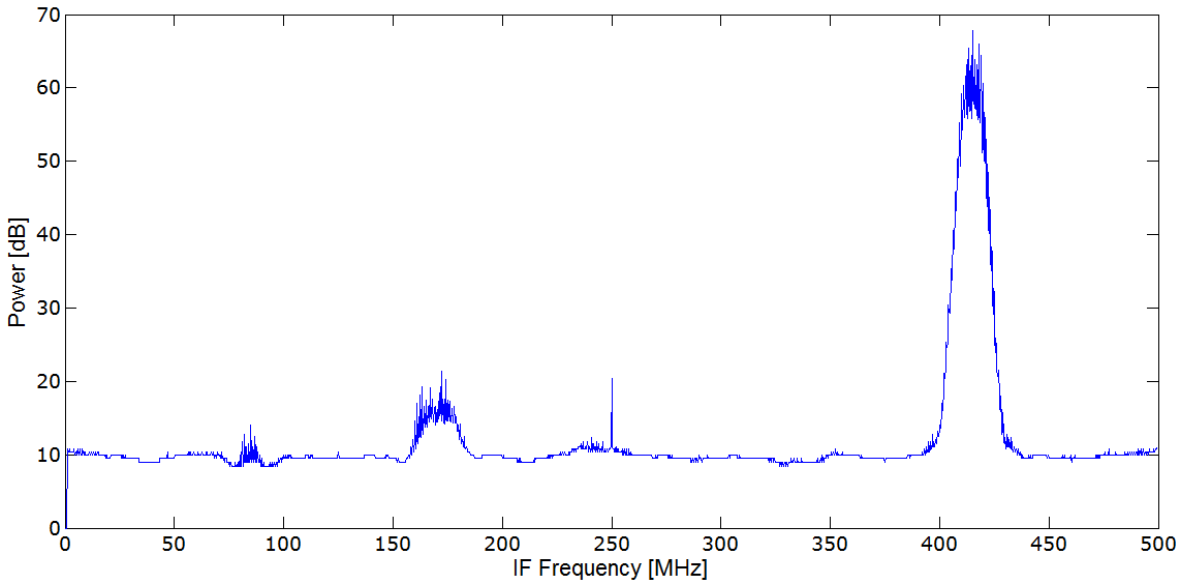


Figura 5.12: Señal de 500 MHz de ancho de banda con contenido energético en distintas bandas de frecuencia utilizando 2048 canales.

La figura 5.13 muestra un zoom a la señal de la figura 5.12 entre 125 y 187,5 MHz. Se aprecia una zona con cierto contenido energético que apenas sobrepasa los 20 decibeles. Al realizar zoom se puede visualizar que la forma de la señal se mantiene llegando también

hasta 20 decibeles pero con una resolución espectral ocho veces mayor que en la figura 5.12. El contenido de la señal entre 400 y 430 MHz apreciado en la figura 5.12 es completamente atenuado por el filtro *PB_3zn*, y por lo tanto se evita completamente la filtración de esa porción hacia el zoom por *aliasing*.

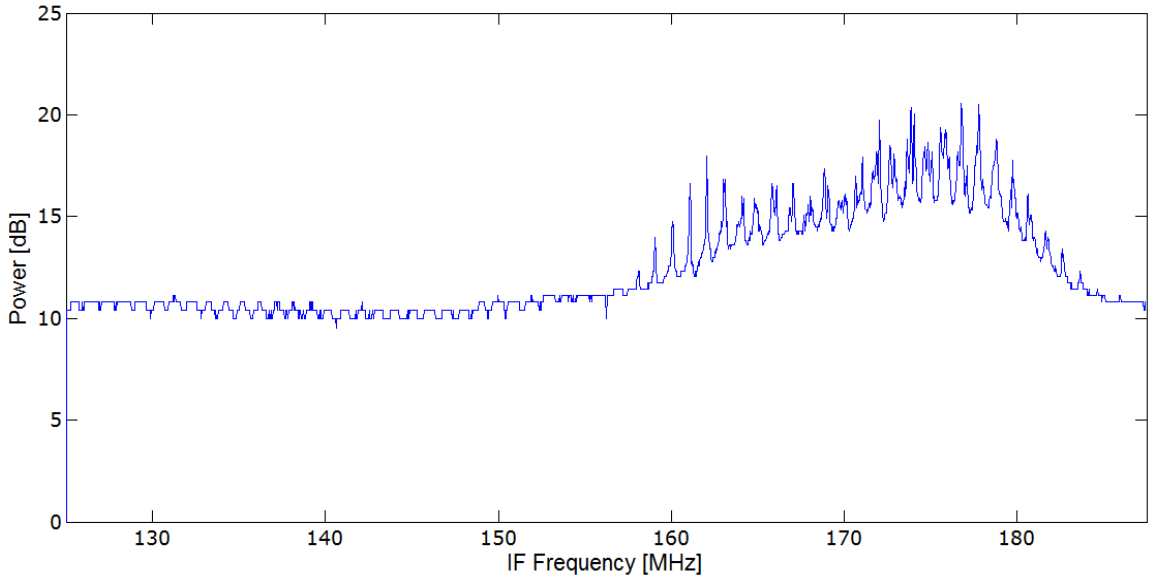


Figura 5.13: Zoom a la señal de la figura 5.12 entre 125 y 187 MHz, utilizando 2048 canales y aumentando ocho veces la resolución espectral.

La figura 5.14 muestra un zoom realizado a la señal de la figura 5.12 entre 375 y 437,5 MHz. En esta zona se ve un contenido energético que sobrepasa los 60 decibeles. Se pueden apreciar más detalles producto del aumento en la resolución.

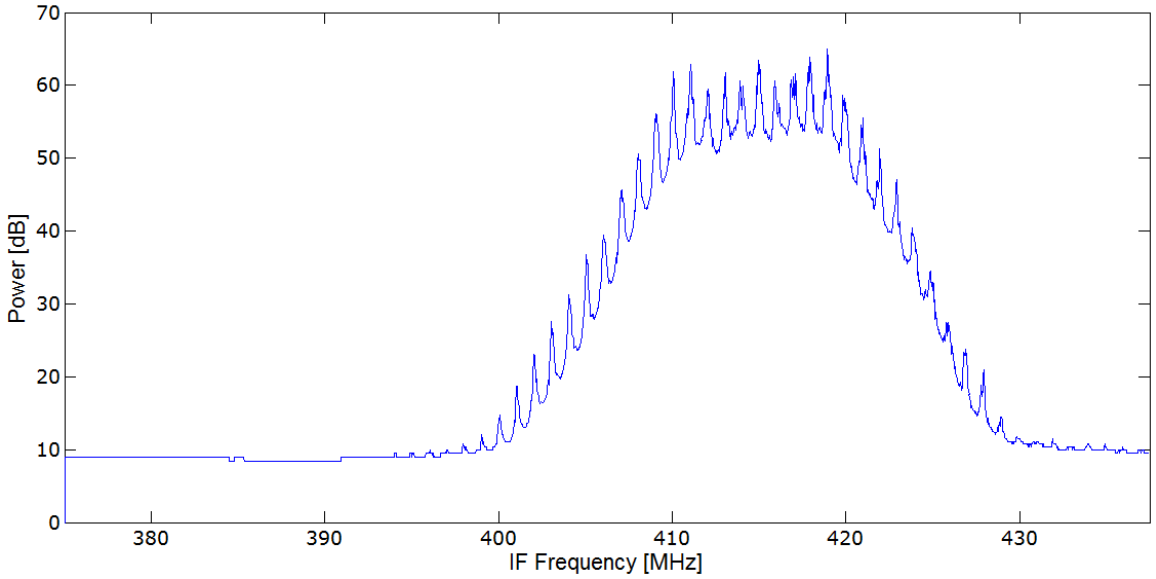


Figura 5.14: Zoom a la señal de la figura 5.12 entre 375 y 437,5 MHz, utilizando 2048 canales y aumentando ocho veces la resolución espectral.

5.2.2 Análisis de los resultados

La resolución espectral aumenta ocho veces en cada acercamiento que se realiza, ya que se ocupa la misma cantidad de canales espectrales, pero en un ancho de banda ocho veces menor (62,5 MHz). En efecto, el espectrómetro de 2048 canales, que procesa una señal de 500 MHz de ancho, tiene un ancho de canal de 500/2048=244,14 KHz, mientras que el *zooming spectrometer* tiene un ancho de canal de 62,5/2048=30,5 KHz, lo que implica que cualquier variación del PSD en un ancho de banda menor que 244,14 KHz no será percibido por el espectrómetro sin zoom. Esto se aprecia en la figura 5.14, ya que hay variaciones de potencia que se hacen evidentes sólo cuando se realiza el zoom en el espectrómetro.

Por otra parte en la figura 5.10 (a) se puede apreciar que un tono es más potente que el otro, sin embargo al realizar el acercamiento mostrado en la figura 5.10 (b), las potencias relativas de cada tono se ven alteradas, es decir el tono menos potente se ve un poco más potente que su vecino al realizar el acercamiento. Esto ocurre porque la respuesta en magnitud del respectivo filtro FIR no es exactamente plana en la zona pasa banda. Además la energía del tono se puede repartir entre dos canales debido a que la frecuencia cae en medio de dos canales, pero al realizar zoom este mismo tono puede caer más cerca de un canal que de otro aumentando así la potencia con la que aparece. Se pueden apreciar diferencias de hasta unos 7 dB entre 2 tonos de la misma potencia analógica si la frecuencia de uno cae justo en la frecuencia representativa de un canal, mientras que la frecuencia del otro tono cae entre 2 canales.

En la figura 5.10 (b), se puede apreciar que el piso de ruido cayó 7 decibeles con respecto a la figura 5.10 (a). Lo anterior se debe a que al hacer zoom, para mantener la resolución temporal, es necesario bajar el largo de acumulación. En este caso se bajó el largo de acumulación de 131072 a 16384, lo que representa una disminución de 8 veces. Esto tiene sentido, puesto que al decimar la señal descartando siete datos de cada ocho que entran a la FFT, se requiere ocho veces más de tiempo para completar el mismo largo de la FFT. En otras palabras la FFT se demora ocho veces más en entregar un espectro cuando se hace zoom. Lo anterior implica que si se requiere observar una fuente que cambia su energía muy rápido en función del tiempo, como por ejemplo un pulsar, al realizar zoom no se puede mantener el largo de acumulación, ya que de lo contrario la resolución temporal podría ser insuficiente para percibir los cambios en la energía de la fuente. En ese sentido el costo de mantener la resolución temporal al realizar un acercamiento es la disminución de la relación señal a ruido.

En la figura 5.11 se puede ver que un tono fuera de la zona de acercamiento puede observarse de igual manera por efecto *aliasing*, debido a que el filtro respectivo no alcanza a atenuar completamente en esa zona. Para resolver este problema se requiere diseñar filtros más angostos.

Capítulo 6

Conclusiones

En este capítulo se entregan las conclusiones generales del trabajo realizado. Se culmina indicando mejoras propuestas para trabajos futuros relacionados con este.

6.1. Conclusiones generales

Las mejoras expuestas por el bloque de control de datos dejan en evidencia la importancia de sincronizar la lectura de datos, con la lógica implementada dentro del FPGA. Esto debido a que las altísimas velocidades de este último pueden hacer que datos supuestamente simultáneos en realidad provengan de distintos espectros, perdiendo correlación temporal.

Las medidas de SRR mejoran en estabilidad al calibrar con datos simultáneos, incluso en condiciones donde se adhiere ruido intencionalmente y a un costo bastante económico, ya que la solución requirió solo de un par de contadores, tres multiplexores, bloques de lógica *and* y *or*, y una pequeña porción de memoria RAM en los registros.

Para el caso del *zooming spectrometer*, una conclusión importante es el hecho de que un mejor desempeño requiere cada vez de más recursos en el FPGA. El principal problema que presenta el diseño del zoom está en el orden de los filtros FIR. Lo que se requiere para evitar *aliasing* es que el filtro esté lo más ajustado posible con la zona en que se realiza el acercamiento, es decir se requiere de un filtro lo más cercano al ideal. Sin embargo, para lograr esto se requiere un filtro de orden muy alto, haciendo imposible que los recursos del FPGA den abasto (muchas multiplicaciones, sumas y retardos simultáneamente). Sin embargo si la señal no tiene contenido energético considerable en los extremos exteriores próximos de cada zona de acercamiento, el desempeño del zoom es bastante bueno, aumentando ocho veces la resolución espectral, lo cual es muy útil en espectrometría de alta resolución aplicada al estudio de estructuras moleculares híper finas. Por otro lado el costo en *hardware* de realizar un zoom utilizando un solo filtro es similar al costo del espectrómetro simple. Esto asegura que si un espectrómetro esta implementado en un FPGA, entonces hay suficiente espacio en *hardware* para implementar al menos un filtro de los que se diseñaron en este trabajo.

La implementación de los distintos diseños en un mismo procesador, demuestra la gran ventaja del FPGA, ya que por ejemplo si se está utilizando un espectrómetro para observar la emisión de una molécula y se desea realizar un acercamiento a una cierta zona

del espectro, el diseño del zoom se puede cargar rápidamente en el procesador en unos pocos segundos, y así en general se pueden cargar distintos modelos sin la necesidad de cambiar el *hardware*.

6.2. Trabajo futuro y mejoras propuestas

Sin duda el rápido avance en las tecnologías de procesamiento digital basadas en FPGA hará posible que en poco tiempo los resultados de este trabajo sean replicados en arquitecturas capaces de procesar un ancho de banda mayor que 500 MHz. En concreto, hay evidencias de que el mismo chip de FPGA virtex 5 con que se realizó este trabajo es capaz de implementar este tipo de espectrómetros para anchos de banda tres veces más grande. Para lograr esto es necesario realizar un *floorplanning* al diseño, es decir optimizar manualmente las rutas físicas y el espaciamiento en el hardware. Mediante este método otros trabajos paralelos reportan aumentos de hasta 1 GHz en el ancho de banda de la IF procesada, utilizando el mismo *hardware*.

En particular para el diseño del zoom se requiere mejorar la respuesta en magnitud de los filtros FIR, en términos de sus frecuencias de corte, para que estos se ajusten de mejor manera a las zonas de Nyquist. También es posible disminuir la atenuación unos 10 decibeles en las bandas de rechazo para obtener filtros de órdenes inferiores. Es muy probable que la implementación de estos filtros en ROACH II pueda tener una forma más ajustada ya que esta plataforma presenta mayor capacidad de cálculo paralelo. Otras mejoras propuestas para este diseño son la implementación de zoom de 2x y 4x. Además es posible realizar zoom en zonas distintas a las zonas Nyquist de la señal decimada. Para esto es necesario encontrar una forma de decimar que permita filtrar y posteriormente mezclar la señal para desplazarla en frecuencia y luego descartar las muestras que se deseen, para así reducir el ancho de banda en la cantidad que se necesite. También es necesario integrar todos estos modelos en una GUI que permita cargar fácilmente cada modelo en la ROACH mediante una interfaz de usuario amigable y fácil de manejar.

Otro punto importante es que se han utilizado algunos bloques prediseñados, los cuales además de realizar los cálculos deseados, realizan otros cálculos que no son relevantes para este trabajo. Así, estos bloques están ocupando lógica innecesaria en el *hardware*. Es posible modificar estos bloques debajo de sus máscaras para que no realicen cálculos superfluos y los diseños realizados en este trabajo ocupen menos espacio en el FPGA.

Bibliografía

[1] Radio Astronomy, John D. Kraus, Ph. D. McGraw-Hill Book Company.

[2] David Levington ‘’A History of Astronomy, From 1890 to the Present”, Springer, Londres, 1996.

[3] Tools of Radioastronomy, T.L. Wilson, K. Rohlfs, S. Huttemeister. Fifth Edition, Springer.

[4]http://commons.wikimedia.org/wiki/File:Atmospheric_electromagnetic_transmittance_or_opacity.jpg, Junio 2014.

[5] J. Fariña, “Diseño y construcción de la etapa analógica de un interferómetro de dos antenas”, Engineering thesis, Universidad de Chile, 2010.

[6] R. Finger, P. Mena, N. Reyes, R. Rodriguez, L. Bronfman, "A calibrated digital sideband separating spectrometer for radio astronomy applications", Publications of the Astronomical Society of the Pacific, 125, 263-269, 2013.

[7] The Milky Way in Molecular Clouds: A New Complete CO Survey. T. M. Dame, Dap Hartmann, and P. Thaddeus.

[8] P. Vasquez “Instalación y puesta en marcha del Radiotelescopio Mini”. Engineering thesis, Universidad de Chile, 2011.

[9] R. Finger, Design and Construction of a Digital Sideband Separating Spectrometer for the 1.2-meter Southern Radio Telescope.

[10] P. Astudillo, “Medición del Patrón de Radiación del Telescópio Mini”, Engineering Thesis, Universidad de Chile 2014.

[11] Stutzman, W. y G. Thiele: Antenna Theory and Design. John Wiley & Sons, Inc., 1981.

[12] http://www.mpifr-bonn.mpg.de/8964/effelsberg

[13] M. Morgan and J. Richard Fisher, Experiments With Digital Sideband-Separating Downconversion, Publications of the Astronomical Society of the Pacific, vol. 122, no. 889, pp. 326-335, March 2010.

[14] http://legacy.nrao.edu/alma/memos/html-memos/alma357/memo357.pdf (August, 2014)

[15] B. Henderson and J.Cook. (1985). Image-Reject and Single-Sideband Mixers.

[16] http://www.historyofinformation.com/expanded.php?id=77 (August, 2014)

[17] Digital Signal Processing: Principles, Algorithms & Applications (3rd Ed.), John G. Proakis, Dimitris G. Manolakis.

[18] https://casper.berkeley.edu/wiki/Wideband_Spectrometer (June, 2014)

[19] The Design Warrior's Guide to FPGAs. Devices, Tools and Flows. Clive Maxfield

Apéndice A: Códigos Python

spectrometer_64bits_float_s12_dif_ed_sin_acc_data_upper_low2_dc.py (Calibrador)

```
import corr,time,numpy,struct,sys,logging,pylab,matplotlib,math,Gnuplot, Gnuplot.funcutils,array, telnetlib,
valon_synth

from math import *

bitstream = 'No_bof_file_error'

katcp_port=7147

def creartxt():
    archi=open('datos.dat','w')
    archi.close()
def creartxt():
    archi_teo=open('datos_teo.dat','w')
    archi_teo.close()
archi=open('datos.dat','a')
archi_teo=open('datos_teo.dat','a')
creartxt()
def exit_fail():
    print 'FAILURE DETECTED. Log entries:\n',lh.printMessages()
    try:
        fpga.stop()
    except: pass
    raise
    exit()
def exit_clean():
    try:
        fpga.stop()
    except: pass
    exit()
def get_data():
    fpga.write_int('data_ctrl_lec_done',0)#####
    fpga.write_int('data_ctrl_sel_we',1)  ##### this two lines disable writting on the dout's BRAM
    #Rama I
    re_0i=struct.unpack('>512q',fpga.read('dout0_0',512*8,0))
```

```
    im_0i=struct.unpack('>512q',fpga.read('dout0_1',512*8,0))
    re_2i=struct.unpack('>512q',fpga.read('dout0_2',512*8,0))
    im_2i=struct.unpack('>512q',fpga.read('dout0_3',512*8,0))
    #Rama Q
    re_0q=struct.unpack('>512q',fpga.read('dout1_0',512*8,0))
    im_0q=struct.unpack('>512q',fpga.read('dout1_1',512*8,0))
    re_2q=struct.unpack('>512q',fpga.read('dout1_2',512*8,0))
    im_2q=struct.unpack('>512q',fpga.read('dout1_3',512*8,0))

    fpga.write_int('data_ctrl_lec_done',1)#####
    fpga.write_int('data_ctrl_sel_we',0)  ##### this two lines enable writting on the dout's BRAM

    spec_i=[]
    spec_q=[]
    power_spec_i=[]
    power_spec_q=[]
    for i in range(512):
        spec_i.append(float(re_0i[i])/(2**18))
        spec_i.append(float(im_0i[i])/(2**18))
        spec_i.append(float(re_2i[i])/(2**18))
        spec_i.append(float(im_2i[i])/(2**18))
        spec_q.append(float(re_0q[i])/(2**18))
        spec_q.append(float(im_0q[i])/(2**18))
        spec_q.append(float(re_2q[i])/(2**18))
        spec_q.append(float(im_2q[i])/(2**18))
return spec_i, spec_q

def arcotan(im,re):# define la función arcotangente para los 4 cuadrantes
    tan=0
    if im>=0.0 and re>=0.0:
        if re==0:
                re=10**-20
        tan=atan(im/re)
```

```
    if im>=0.0 and re<=0.0:
        if im==0:
                im=10**-20
        tan=pi/2+atan(abs(re)/im)
    if im<=0.0 and re<=0.0:
        if re==0:
                re=10**-20
        tan=pi+atan(abs(im)/abs(re))
    if im<=0.0 and re>=0.0:
        if im==0:
                im=10**-20
        tan=(3*pi/2)+atan(re/abs(im))
    return tan


def trunca(f, n):# trunca un numero en el decimal deseado.
    '''Truncates/pads a float f to n decimal places without rounding'''
    slen = len('%.*f' % (n, f))
    return str(f)[:slen]


################   START OF MAIN   ################


if __name__ == '__main__':
    from optparse import OptionParser



    p = OptionParser()
    p.set_usage('spectrometer.py <ROACH_HOSTNAME_or_IP> [options]')
    p.set_description(__doc__)
    p.add_option('-l', '--acc_len', dest='acc_len', type='int',default=2*(2**28)/2048,
        help='Set the number of vectors to accumulate between dumps. default is 2*(2^28)/2048, or just under 2 seconds.')
    p.add_option('-g', '--gain', dest='gain', type='int',default=0x00001000,
        help='Set the digital gain (6bit quantisation scalar). Default is 0xffffffff (max), good for wideband noise. Set lower for CW tones.')
```

```
    p.add_option('-s', '--skip', dest='skip', action='store_true',
        help='Skip reprogramming the FPGA and configuring EQ.')
    p.add_option('-b', '--bof', dest='boffile',type='str', default='',
        help='Specify the bof file to load')
    opts, args = p.parse_args(sys.argv[1:])

    if args==[]:
        print 'Please specify a ROACH board. Run with the -h flag to see all options.\nExiting.'
        exit()
    else:
        roach = args[0]
    if opts.boffile != '':
        bitstream = opts.boffile

try:
    loggers = []
    lh=corr.log_handlers.DebugLogHandler()
    logger = logging.getLogger(roach)
    logger.addHandler(lh)
    logger.setLevel(10)

    print('Connecting to server %s on port %i... '%(roach,katcp_port)),
    fpga = corr.katcp_wrapper.FpgaClient(roach, katcp_port, timeout=10,logger=logger)
    time.sleep(1)

    if fpga.is_connected():
        print 'ok\n'
    else:
        print 'ERROR connecting to server %s on port %i.\n'%(roach,katcp_port)
        exit_fail()

    print '------------------------'
    print 'Programming FPGA with %s...' %bitstream,
```

```
    if not opts.skip:
        fpga.progdev(bitstream)
        print 'done'
    else:
        print 'Skipped.'
    print 'Configuring FFT shift register...',
    fpga.write('shift_ctrl','\x00\x00\x0f\xff')
    print 'done'
    print 'Resetting counters...',
    fpga.write_int('cnt_rst',1)
    fpga.write_int('cnt_rst',0)
    print 'done'
    ti=time.time()
    bw=trunc(fpga.est_brd_clk())*4
    lo=input('LO frequency MHZ?)')
    rys = telnetlib.Telnet("172.17.89.49",5025)
    valon=valon_synth.Synthesizer('/dev/ttyUSB0')
    agi = telnetlib.Telnet("172.17.89.54",5023)
    agi.write("freq 2.0ghz\r\n")
    agi.write("power -20dbm\r\n")
    agi.write("output on\r\n")
    def beep():
        print "\a"
    rys.write("output off\r\n")
    rys.write("freq "+str(float(lo)/1000)+"ghz\r\n")
    rys.write("output on\r\n")
    archi.write('#Canal'+' '+'real i'.rjust(7)+' '+'imag i'.rjust(7)+' '+'real q'.rjust(7)+' '+'imag q'.rjust(7)+'
'+'amp_i'.rjust(7)+' '+'amp_q'.rjust(7)+' '+'phase_i'.rjust(7)+' '+'phase_q'.rjust(7)+' '+'amp_rat'.rjust(7)+'
'+'ang_dif'.rjust(7)+' '+'phi'.rjust(7)+' \n')
    ch=input('number of channels(2^?)')
    teo_upper=[]
    for i in range(1*1024/(2**ch),1023,1024/(2**ch)): #comienza el barrido de frecuencia en el USB
            tu=time.time()
            if i/20==float(i)/20:
```

```
                beep()
                        print(str((tu-ti)/60)+' minutos'+' '+str(float(i)*100.0/2044)+'%')
            #valon.set_frequency(valon_synth.SYNTH_A,(2*i)*float(bw)/2048,0.0026)
                agi.write("freq "+ str(lo+(2*i)*float(bw)/2048) +"mhz\r\n")
                    time.sleep(1)
              spec_i, spec_q = get_data()
                    amp_i=((spec_i[2*i])**2+(spec_i[2*i+1])**2)**0.5
                    amp_q=((spec_q[2*i])**2+(spec_q[2*i+1])**2)**0.5
                    angle_i=arcotan(spec_i[2*i+1],spec_i[2*i])*180/(pi) #en grados
                    angle_q=arcotan(spec_q[2*i+1],spec_q[2*i])*180/(pi) #en grados
                    phi=arcotan(spec_q[2*i]*spec_i[2*i+1]-
spec_i[2*i]*spec_q[2*i+1],spec_i[2*i]*spec_q[2*i]+spec_i[2*i+1]*spec_q[2*i+1])*180/(pi)
        amp_ratio=amp_i/amp_q# x del paper
                    phase_dif=(angle_i-angle_q)# phi usb del paper de Fisher y Morgan
              if phase_dif<0:
                phase_dif=360+phase_dif
              teo_upper.append([1/(amp_ratio),lo+(2*i)*float(bw)/2048])
                    teo_upper.append([180-phase_dif,lo+(2*i)*float(bw)/2048])
              amp_i=trunca(amp_i,2)
              amp_q=trunca(amp_q,2)
              angle_i=trunca(angle_i,2)
              angle_q=trunca(angle_q,2)
              amp_rat=trunca(amp_ratio,2)
                ang_dif=trunca(phase_dif,2)
                phi=trunca(phi,2)
                print(str(lo+(2*i)*float(bw)/2048))
              archi.write(repr(2*i).rjust(6)+' '+trunca(spec_i[2*i],2).rjust(7)+' '+trunca(spec_i[2*i+1],2).rjust(7)+'
'+trunca(spec_q[2*i],2).rjust(7)+' '+trunca(spec_q[2*i+1],2).rjust(7)+' '+(amp_i).rjust(7)+'
'+(amp_q).rjust(7)+' '+(angle_i).rjust(7)+' '+(angle_q).rjust(7)+' '+(amp_rat).rjust(7)+' '+(ang_dif).rjust(7)+'
'+phi.rjust(7)+' \n')
    archi.write('lower_sideband'+' \n')
    archi.write('#Canal'+' '+'real i'.rjust(7)+' '+'imag i'.rjust(7)+' '+'real q'.rjust(7)+' '+'imag q'.rjust(7)+'
'+'amp_i'.rjust(7)+' '+'amp_q'.rjust(7)+' '+'phase_i'.rjust(7)+' '+'phase_q'.rjust(7)+' '+'amp_rat'.rjust(7)+'
'+'ang_dif'.rjust(7)+' '+'phi'.rjust(7)+' \n')
    teo_lower=[]
    for i in range(1*1024/(2**ch),1023,1024/(2**ch)):# Comienza el barrido de frecuencia en el LSB
```

```
            tl=time.time()
             if i/20==float(i)/20:
                beep()
                        print(str((tl-ti)/60)+' minutos'+' '+str(float(1022+i)*100.0/2044)+'%')
            #valon.set_frequency(valon_synth.SYNTH_A,(2*i)*float(bw)/2048,0.0026)
                 agi.write("freq "+ str(lo-(2*i)*float(bw)/2048) +"mhz\r\n")
                     time.sleep(1)
             spec_i, spec_q = get_data()
                     amp_i=((spec_i[2*i])**2+(spec_i[2*i+1])**2)**0.5
                     amp_q=((spec_q[2*i])**2+(spec_q[2*i+1])**2)**0.5
                     angle_i=arcotan(spec_i[2*i+1],spec_i[2*i])*180/(pi) #en grados
                     angle_q=arcotan(spec_q[2*i+1],spec_q[2*i])*180/(pi) #en grados
                     phi=arcotan(spec_q[2*i]*spec_i[2*i+1]-
spec_i[2*i]*spec_q[2*i+1],spec_i[2*i]*spec_q[2*i]+spec_i[2*i+1]*spec_q[2*i+1])*180/(pi)
                     #phi=atan((spec_q[2*i]*spec_i[2*i+1]-
spec_i[2*i]*spec_q[2*i+1])/(spec_i[2*i]*spec_q[2*i]+spec_i[2*i+1]*spec_q[2*i+1]))*180/(pi)
        amp_ratio=amp_i/amp_q# x del paper de morgan y fisher
                     phase_dif=(angle_i-angle_q)
            if phase_dif<0:
                phase_dif=360+phase_dif
            teo_lower.append([(amp_ratio),lo-(2*i)*float(bw)/2048])
                     teo_lower.append([phase_dif-180,lo-(2*i)*float(bw)/2048])
            amp_i=trunca(amp_i,2)
            amp_q=trunca(amp_q,2)
            angle_i=trunca(angle_i,2)
            angle_q=trunca(angle_q,2)
                     amp_rat=trunca(amp_ratio,2)
                     ang_dif=trunca(phase_dif,2)
                     phi=trunca(phi,2)
                     print(str(lo-(2*i)*float(bw)/2048))
            archi.write(repr(2046-2*i).rjust(6)+' '+trunca(spec_i[2*i],2).rjust(7)+'
'+trunca(spec_i[2*i+1],2).rjust(7)+' '+trunca(spec_q[2*i],2).rjust(7)+' '+trunca(spec_q[2*i+1],2).rjust(7)+'
'+(amp_i).rjust(7)+' '+(amp_q).rjust(7)+' '+(angle_i).rjust(7)+' '+(angle_q).rjust(7)+' '+(amp_rat).rjust(7)+'
'+(ang_dif).rjust(7)+' '+phi.rjust(7)+' \n')
```

```
    archi_teo.write('#lower_sideband'+' \n')
    for i in range(0,len(teo_lower),2):
        archi_teo.write('0 '+str(teo_lower[i][0])+' '+str(teo_lower[i][1])+' \n') #escribe amp_ratio en LSB
    for i in range(1,len(teo_lower),2):
            archi_teo.write('0 '+str(teo_lower[i][0])+' '+str(teo_lower[i][1])+' \n')     #escribe phase_dif-180 en
LSB
    archi_teo.write('#upper_sideband'+' \n')
    for i in range(0,len(teo_upper),2):
        archi_teo.write('0 '+str(teo_upper[i][0])+' '+str(teo_upper[i][1])+' \n') #escribe 1/amp_ratio en USB
    for i in range(1,len(teo_upper),2):
            archi_teo.write('0 '+str(teo_upper[i][0])+' '+str(teo_upper[i][1])+' \n')     #escribe 180-phase_dif en
USB
    tf=time.time()
    print('tiempo total='+repr((tf-ti)/60)+' minutos')
except KeyboardInterrupt:
    exit_clean()
except:
    exit_fail()
exit_clean()
```

spectrometer_64bits_float_s12_SRR_measurement_agi_prueba.py (SRR)

```
import corr, time, struct, sys, logging, Gnuplot, valon_synth, telnetlib, numpy as np
from math import *
bitstream = 'No_bof_file_error'
katcp_port=7147
t1 = time.time()  #Inicializa cronometro.
def hhmmss(segundostotales):
    hh = segundostotales // 3600
    mm = (segundostotales % 3600)//60
    ss = (segundostotales %3600) %60
    return hh,mm,ss
def exit_fail():
    print 'FAILURE DETECTED. Log entries:\n',lh.printMessages()
```

```
    try:
        fpga.stop()
    except: pass
    raise
    exit()
def exit_clean():
    try:
        fpga.stop()
    except: pass
    exit()
def get_data():
    acc_n = fpga.read_uint('acc_cnt')
   fpga.write_int('data_ctrl_lec_done',0)#####
   fpga.write_int('data_ctrl_sel_we',1)  ##### this two lines disable writting on the dout's BRAM
    a_0l=struct.unpack('>512Q',fpga.read('dout0_0',512*8,0))
    a_1l=struct.unpack('>512Q',fpga.read('dout0_1',512*8,0))
    a_2l=struct.unpack('>512Q',fpga.read('dout0_2',512*8,0))
    a_3l=struct.unpack('>512Q',fpga.read('dout0_3',512*8,0))
    a_0m=struct.unpack('>512Q',fpga.read('dout1_0',512*8,0))
    a_1m=struct.unpack('>512Q',fpga.read('dout1_1',512*8,0))
    a_2m=struct.unpack('>512Q',fpga.read('dout1_2',512*8,0))
    a_3m=struct.unpack('>512Q',fpga.read('dout1_3',512*8,0))
    fpga.write_int('data_ctrl_lec_done',1)#####
    fpga.write_int('data_ctrl_sel_we',0)  ##### this two lines enable writting on the dout's BRAM
    interleave_a=[]
    interleave_b=[]
    interleave_log=[]
    interleave_log_b=[]
    for i in range(512):
        interleave_a.append(float(((float(a_0l[i])+1)/(2**24))))#24 es el original
        interleave_a.append(float(((float(a_1l[i])+1)/(2**24))))
        interleave_a.append(float(((float(a_2l[i])+1)/(2**24))))
        interleave_a.append(float(((float(a_3l[i])+1)/(2**24))))
```

```
        interleave_b.append(float(((float(a_0m[i])+1)/(2**24))))
        interleave_b.append(float(((float(a_1m[i])+1)/(2**24))))
        interleave_b.append(float(((float(a_2m[i])+1)/(2**24))))
        interleave_b.append(float(((float(a_3m[i])+1)/(2**24))))
    for k in range(4*512):
        interleave_log.append(10*log10(interleave_a[k]))
        interleave_log_b.append(10*log10(interleave_b[k]))
    return acc_n, interleave_a, interleave_log, interleave_log_b
srr_datos=open('srr_datos.dat','w')
lsb_usb_testtone=open('lsb_usb_testtone.dat','w')
datos_teo_corregido=open('datos_teo_corregido.dat','w')
################   START OF MAIN   ############################
if __name__ == '__main__':
    from optparse import OptionParser
    p = OptionParser()
    p.set_usage('spectrometer.py <ROACH_HOSTNAME_or_IP> [options]')
    p.set_description(__doc__)
    p.add_option('-l', '--acc_len', dest='acc_len', type='int',default=2*(2**28)/2048,
        help='Set the number of vectors to accumulate between dumps. default is 2*(2^28)/2048, or just under 2 seconds.')
    p.add_option('-g', '--gain', dest='gain', type='int',default=0x00001000,
        help='Set the digital gain (6bit quantisation scalar). Default is 0xffffffff (max), good for wideband noise. Set lower for CW tones.')
    p.add_option('-s', '--skip', dest='skip', action='store_true',
        help='Skip reprogramming the FPGA and configuring EQ.')
    p.add_option('-b', '--bof', dest='boffile',type='str', default='',
        help='Specify the bof file to load')
    opts, args = p.parse_args(sys.argv[1:])
    if args==[]:
        print 'Please specify a ROACH board. Run with the -h flag to see all options.\nExiting.'
        exit()
    else:
        roach = args[0]
    if opts.boffile != '':
```

```
        bitstream = opts.boffile
try:
    loggers = []
    lh=corr.log_handlers.DebugLogHandler()
    logger = logging.getLogger(roach)
    logger.addHandler(lh)
    logger.setLevel(10)
    print('Connecting to server %s on port %i... '%(roach,katcp_port)),
    fpga = corr.katcp_wrapper.FpgaClient(roach, katcp_port, timeout=10,logger=logger)
    time.sleep(1)
    if fpga.is_connected():
        print 'ok\n'
    else:
        print 'ERROR connecting to server %s on port %i.\n'%(roach,katcp_port)
        exit_fail()
    print '------------------------'
    print 'Programming FPGA with %s...' %bitstream,
    if not opts.skip:
        fpga.progdev(bitstream)
        print 'done'
    else:
        print 'Skipped.'
    print 'Configuring FFT shift register...',
    fpga.write('shift_ctrl','\x00\x00\x0f\xff')
    print 'done'
    print 'Configuring accumulation period...',
    fpga.write_int('acc_len',opts.acc_len)
    print 'done'
    print 'Resetting counters...',
    fpga.write_int('cnt_rst',1)
    fpga.write_int('cnt_rst',0)
    print 'done'
    print 'Setting digital gain of all channels to %i...'%opts.gain,
```

```
if not opts.skip:
    fpga.write_int('gain',opts.gain) #write the same gain for all inputs, all channels
    print 'done'
else:
    print 'Skipped.'
g0 = Gnuplot.Gnuplot(debug=1)
g1 = Gnuplot.Gnuplot(debug=1)
g2 = Gnuplot.Gnuplot(debug=1)
bw=trunc(fpga.est_brd_clk())*4
print ('BW obtenido es: '+str(fpga.est_brd_clk()))
g0.clear()
g0.title('ADC1 spectrum using '+bitstream+' | Max frequency = '+str(bw)+' MHz')
g0.xlabel('Channel #')
g0.ylabel('Power AU (dB)')
g0('set style data linespoints')
g0('set yrange [0:120]')
g0('set xrange [-50:2098]')
g0('set ytics 5')
g0('set xtics 256')
g0('set grid y')
g0('set grid x')
g1.clear()
g1.title('ADC0 spectrum using '+bitstream+' | Max frequency = '+str(bw)+' MHz')
g1.xlabel('Channel #')
g1.ylabel('Power AU (dB)')
g1('set style data linespoints')
g1('set yrange [0:120]')
g1('set xrange [-50:2098]')
g1('set ytics 5')
g1('set xtics 256')
g1('set grid y')
g1('set grid x')
rys = telnetlib.Telnet("172.17.89.49",5025)
```

```
    agi = telnetlib.Telnet("172.17.89.50",5023)
    #valon=valon_synth.Synthesizer('/dev/ttyUSB0')
    rys.write("output off\r\n")
    agi.write("output off\r\n")
#setting LO
    LO=input('LO frequency [GHz] ? : ')  #GHz
    rys.write("freq "+str(LO)+"ghz\r\n") # Cambiar frec de LO
    rys.write("power 18dbm\r\n")
    rys.write("output on\r\n")
#setting RF start point
    agi.write("freq 2.0ghz\r\n")
    agi.write("power -20dbm\r\n")
    agi.write("output on\r\n")
    srr=[]
    rf=[]
    tono_usb=[]
    tono_lsb=[]
    canales=[]
    ch=[]
    print ('bw es: '+ str(bw))
    bw=trunc(fpga.est_brd_clk())*4.0/1000
    start=LO-bw
    stop=LO+bw
    print ''
    points=input('Number of points? :') #Numero de puntos para mostrar SRR
    modo_memoria=input('modo memoria? (1,2 o 3)=')
    g2.clear()
    g2.title('Sideband Rejection Ratio using '+bitstream+' | running at '+str(bw)+' GHz')
    g2.xlabel('RF freq (GHz) - shown LSB and USB -')
    g2.ylabel('SSR (dB)')
    g2('set style data linespoints')
    g2('set yrange [0:60]')
    g2('set xrange ['+str(start-0.05)+":"+str(stop+0.05)+']')
```

```
    g2('set ytics 5')

    g2('set xtics 0.1')

    g2('set grid y')

    g2('set grid x')

    ampq=[]

    degq=[]

    ampi=[]

    degi=[]

    matriz_datos_optimos=np.loadtxt('datos_teo.dat')   ##se carga el archivo con las constantes obtenidas en la calibración.

modo_memoria=1 (para completar con datos optimizados o teoricos)

                # modo_memoria=2 (completa con datos 1 de amp y 90 grados)

    if modo_memoria == 1:

        for i in range (0,(len(matriz_datos_optimos)/4)):

            ampi.append(matriz_datos_optimos[i,1])

        for i in range ((len(matriz_datos_optimos)/4),(2*(len(matriz_datos_optimos)/4))):

            degi.append(matriz_datos_optimos[i,1])

        for i in range ((2*(len(matriz_datos_optimos)/4)),(3*(len(matriz_datos_optimos)/4))):

            ampq.append(matriz_datos_optimos[i,1])

        for i in range ((3*(len(matriz_datos_optimos)/4)),(4*(len(matriz_datos_optimos)/4))):

            degq.append(matriz_datos_optimos[i,1])

        for j in range(0,510):# se escriben los datos de la calibración en los bloques
VCM.fpga.write('VCM0_RE_BRAM',struct.pack('>1l',ampi[2*j]*cos((pi/180)*degi[2*j])*2**24),4*j)
#canal 0, 4 ...  (4i)

fpga.write('VCM0_IM_BRAM',struct.pack('>1l',ampi[2*j]*sin((pi/180)*degi[2*j])*2**24),4*j)
fpga.write('VCM1_RE_BRAM',struct.pack('>1l',((ampi[2*j]+ampi[(2*j)+1])/2)*cos((pi/180)*((degi[2*j]+deg
i[(2*j)+1])/2))*2**24),4*j)  #canal 1, 5 ..
fpga.write('VCM1_IM_BRAM',struct.pack('>1l',((ampi[2*j]+ampi[(2*j)+1])/2)*sin((pi/180)*((degi[2*j]+deg
i[(2*j)+1])/2))*2**24),4*j)

fpga.write('VCM2_RE_BRAM',struct.pack('>1l',ampi[(2*j)+1]*cos((pi/180)*degi[(2*j)+1])*2**24),4*j)
#canal 2, 6 ..

fpga.write('VCM2_IM_BRAM',struct.pack('>1l',ampi[(2*j)+1]*sin((pi/180)*degi[(2*j)+1])*2**24),4*j)
fpga.write('VCM3_RE_BRAM',struct.pack('>1l',((ampi[(2*j)+1]+ampi[(2*j)+2])/2)*cos((pi/180)*((degi[(2*j
)+1]+degi[(2*j)+2])/2))*2**24),4*j)
fpga.write('VCM3_IM_BRAM',struct.pack('>1l',((ampi[(2*j)+1]+ampi[(2*j)+2])/2)*sin((pi/180)*((degi[(2*j)
+1]+degi[(2*j)+2])/2))*2**24),4*j)

fpga.write('VCM4_RE_BRAM',struct.pack('>1l',ampq[2*j]*cos((pi/180)*degq[2*j])*2**24),4*j)  #canal 0, 4
...        fpga.write('VCM4_IM_BRAM',struct.pack('>1l',ampq[2*j]*sin((pi/180)*degq[2*j])*2**24),4*j)
```

```
fpga.write('VCM5_RE_BRAM',struct.pack('>1l',((ampq[2*j]+ampq[(2*j)+1])/2)*cos((pi/180)*((degq[2*j]+degq[(2*j)+1])/2))*2**24),4*j)  #canal 1, 5 ...
fpga.write('VCM5_IM_BRAM',struct.pack('>1l',((ampq[2*j]+ampq[(2*j)+1])/2)*sin((pi/180)*((degq[2*j]+degq[(2*j)+1])/2))*2**24),4*j)
fpga.write('VCM6_RE_BRAM',struct.pack('>1l',ampq[(2*j)+1]*cos((pi/180)*degq[(2*j)+1])*2**24),4*j)
#canal 2, 6 ...
fpga.write('VCM6_IM_BRAM',struct.pack('>1l',ampq[(2*j)+1]*sin((pi/180)*degq[(2*j)+1])*2**24),4*j)


fpga.write('VCM7_RE_BRAM',struct.pack('>1l',((ampq[(2*j)+1]+ampq[(2*j)+2])/2)*cos((pi/180)*((degq[(2*j)+1]+degq[(2*j)+2])/2))*2**24),4*j)
fpga.write('VCM7_IM_BRAM',struct.pack('>1l',((ampq[(2*j)+1]+ampq[(2*j)+2])/2)*sin((pi/180)*((degq[(2*j)+1]+degq[(2*j)+2])/2))*2**24),4*j)

        print('--- Escritura de memorias terminada  ---')

            for i in range(points):

        rf.append(start+((stop-start)*i/points))

    modo=1   # Se elige el MODO  1

    if modo == 1:

        srr_datos.write('#Valores SRR    #Frecuencia   \n')

        for i in range(points):

            agi.write("freq " + str(rf[i]) + "ghz\r\n")

            time.sleep(1)

            acc_n, interleave_a, interleave_log, interleave_log_b = get_data()

            time.sleep(0.1)

            interleave_log[0]=interleave_log[1]

            interleave_log_b[0]=interleave_log_b[1]

            g0.plot(interleave_log_b)

            g1.plot(interleave_log)

            srr.append([rf[i],abs(max(interleave_log)-max(interleave_log_b))])# Se calcula la supresión en decibeles.

            g2.plot(srr)

            ### Escribe datos SRR en archivo .dat

            srr_datos.write(str(srr[i][1])+' '+str(rf[i])+' \n')

    lsb_usb_testtone.close()

    t_ejec = time.time()-t1

    hh,mm,ss=hhmmss(t_ejec)

    print('El tiempo de ejecucion del programa (Grafica SRR) fue: '+str(hh)+'[h] ,'+str(mm)+'[min] ,'+str(ss),'[seg].')

    srr_datos.write('# FIN de datos SRR \n')
```

```
    #srr_datos.write('# Tiempo de ejecucion : '+str(hh)+'[h] ,'+str(mm)+'[min] ,'+str(ss),'[seg].')
    srr_datos.close()
    print ('')
    print ('Measurement finished!  - '+str(points)+' points')
    print ('')
    out=raw_input('Enter a filename to save data:(none to exit)')
    if out!='':
        f = open(str(out), 'w')
        f.write("RF (GHz),SSR (dB)\n")
        for i in range (points):
            f.write(str(srr[i][0])+","+str(srr[i][1])+"\n")
except KeyboardInterrupt:
    exit_clean()
except:
    exit_fail()
exit_clean()
```

espectrometro_down8.py

```
import corr,time,numpy,struct,sys,logging,pylab,matplotlib,math,Gnuplot, Gnuplot.funcutils,array
from math import *
bitstream = 'No_bof_file_error'
katcp_port=7147
def exit_fail():
    print 'FAILURE DETECTED. Log entries:\n',lh.printMessages()
    try:
        fpga.stop()
    except: pass
    raise
    exit()
def exit_clean():
    try:
        fpga.stop()
```

```
    except: pass
    exit()
def get_data():
    #get the data...
    acc_n = fpga.read_uint('acc_cnt')
    fpga.write_int('data_ctrl_lec_done',0)#####
    fpga.write_int('data_ctrl_sel_we',1)  ##### this two lines disable writting the dout's BRAM

    a_0l=struct.unpack('>4096Q',fpga.read('dout0_0',4096*8,0))
    a_1l=struct.unpack('>4096Q',fpga.read('dout0_1',4096*8,0))

    fpga.write_int('data_ctrl_lec_done',1)#####
    fpga.write_int('data_ctrl_sel_we',0)  ##### this two lines enable writting the dout's BRAM

    interleave_a=[]
    interleave_b=[]
    interleave_c=[]
    interleave_d=[]
    interleave_log_a=[]
    interleave_log_b=[]
    interleave_log_c=[]
    interleave_log_d=[]

    for i in range(2048):
        interleave_a.append(float((a_0l[i]/(2**18))+1))
        interleave_b.append(float((a_1l[i]/(2**18))+1))
        interleave_c.append(float((a_0l[i+2048]/(2**18))+1))
        interleave_d.append(float((a_1l[i+2048]/(2**18))+1))

    interleave_b.reverse()     #####reverse the 2nd or 6th Nyquist zone Spectra
    interleave_d.reverse()     #####reverse the 4th or 8th Nyquist zone Spectra

    for k in range(4*512):
```

```
        interleave_log_a.append(10*log10(interleave_a[k]))
        interleave_log_b.append(10*log10(interleave_b[k]))
        interleave_log_c.append(10*log10(interleave_c[k]))
        interleave_log_d.append(10*log10(interleave_d[k]))


    return acc_n, interleave_a, interleave_b, interleave_c, interleave_c, interleave_log_a, interleave_log_b, interleave_log_c, interleave_log_d

def continuous_plot(fpga):
        ok=1
        bw=trunc(fpga.est_brd_clk())*4
            g0.clear()
        g0.title('zone 1 '+bitstream+' | Max frequency = '+str(bw)+' MHz')
        g0.xlabel('Channel #')
        g0.ylabel('Power AU (dB)')
        g0('set style data linespoints')
            g0('set yrange [0:100]')
            g0('set xrange [-50:2098]')
            g0('set ytics 5')
            g0('set xtics 256')
            g0('set grid y')
            g0('set grid x')
        g1.clear()
        g1.title('zone 2 '+bitstream+' | Max frequency = '+str(bw)+' MHz')
        g1.xlabel('Channel #')
        g1.ylabel('Power AU (dB)')
        g1('set style data linespoints')
        g1('set yrange [0:100]')
        g1('set xrange [-50:2100]')
        g1('set ytics 5')
        g1('set xtics 256')
        g1('set grid y')
        g1('set grid x')


        g2.clear()
```

```
        g2.title('zone 3 '+bitstream+' | Max frequency = '+str(bw)+' MHz')
        g2.xlabel('Channel #')
        g2.ylabel('Power AU (dB)')
        g2('set style data linespoints')
        g2('set yrange [0:100]')
        g2('set xrange [-50:2100]')
        g2('set ytics 5')
        g2('set xtics 256')
        g2('set grid y')
        g2('set grid x')


        g3.clear()
        g3.title('zone 4 '+bitstream+' | Max frequency = '+str(bw)+' MHz')
        g3.xlabel('Channel #')
        g3.ylabel('Power AU (dB)')
        g3('set style data linespoints')
        g3('set yrange [0:100]')
        g3('set xrange [-50:2100]')
        g3('set ytics 5')
        g3('set xtics 256')
        g3('set grid y')
        g3('set grid x')
        while ok==1 :
            acc_n, interleave_a, interleave_b, interleave_c, interleave_c, interleave_log_a, interleave_log_b,
interleave_log_c, interleave_log_d = get_data()
#                   analog_freq = linspace(0, bw, spec_len)
            #g0.plot(interleave_log_b)
                    #time.sleep(0.3)
            g0.plot(interleave_log_a)
            g1.plot(interleave_log_b)
            g2.plot(interleave_log_c)
            g3.plot(interleave_log_d)
            time.sleep(0.3)
################   START OF MAIN   ################
```

```
if __name__ == '__main__':
    from optparse import OptionParser



    p = OptionParser()
    p.set_usage('spectrometer.py <ROACH_HOSTNAME_or_IP> [options]')
    p.set_description(__doc__)
    p.add_option('-l', '--acc_len', dest='acc_len', type='int',default=2*(2**28)/2048,
        help='Set the number of vectors to accumulate between dumps. default is 2*(2^28)/2048, or just under 2 seconds.')
    p.add_option('-g', '--gain', dest='gain', type='int',default=0x00001000,
        help='Set the digital gain (6bit quantisation scalar). Default is 0xffffffff (max), good for wideband noise. Set lower for CW tones.')
    p.add_option('-s', '--skip', dest='skip', action='store_true',
        help='Skip reprogramming the FPGA and configuring EQ.')
    p.add_option('-b', '--bof', dest='boffile',type='str', default='',
        help='Specify the bof file to load')
    opts, args = p.parse_args(sys.argv[1:])


    if args==[]:
        print 'Please specify a ROACH board. Run with the -h flag to see all options.\nExiting.'
        exit()
    else:
        roach = args[0]
    if opts.boffile != '':
        bitstream = opts.boffile
try:
    loggers = []
    lh=corr.log_handlers.DebugLogHandler()
    logger = logging.getLogger(roach)
    logger.addHandler(lh)
    logger.setLevel(10)
    print('Connecting to server %s on port %i... '%(roach,katcp_port)),
```

```
fpga = corr.katcp_wrapper.FpgaClient(roach, katcp_port, timeout=10,logger=logger)
time.sleep(1)
if fpga.is_connected():
    print 'ok\n'
else:
    print 'ERROR connecting to server %s on port %i.\n'%(roach,katcp_port)
    exit_fail()
print '------------------------'
print 'Programming FPGA with %s...' %bitstream,
if not opts.skip:
    fpga.progdev(bitstream)
    print 'done'
else:
    print 'Skipped.'
print 'Configuring FFT shift register...',
fpga.write('shift_ctrl','\x00\x00\x0f\xff')
print 'done'
print 'Configuring accumulation period...',
fpga.write_int('acc_len',opts.acc_len)
print 'done'
print 'Resetting counters...',
fpga.write_int('cnt_rst',1)
fpga.write_int('cnt_rst',0)
print 'done'
print 'Setting digital gain of all channels to %i...'%opts.gain,
if not opts.skip:
    fpga.write_int('gain',opts.gain) #write the same gain for all inputs, all channels
    print 'done'
else:
    print 'Skipped.'
g0 = Gnuplot.Gnuplot(debug=1)#set up the figure with a subplot to be plotted
g1 = Gnuplot.Gnuplot(debug=1)
g2 = Gnuplot.Gnuplot(debug=1)
```

```
    g3 = Gnuplot.Gnuplot(debug=1)
    continuous_plot(fpga)
    print 'Plot started.'
except KeyboardInterrupt:
    exit_clean()
except:
    exit_fail()
exit_clean()
```