UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE UN ESPECTRÓMETRO DIGITAL CON SEPARACIÓN DE BANDA LATERAL PARA EL RADIOTELESCOPIO MINI

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

JOSÉ ERNESTO VARGAS SALDÍVAR

PROFESOR GUÍA:
WALTER MAX-MOERBECK ASTUDILLO

MIEMBROS DE LA COMISIÓN:
FRANCO CUROTTO MOLINA
FRANCISCO RIVERA SERRANO

Este trabajo ha sido parcialmente financiado por: la Agencia Nacional de Investigación y Desarrollo (ANID) a través de su fondo Quimal ASTRO21-0010

SANTIAGO DE CHILE
2025

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: JOSÉ ERNESTO VARGAS SALDÍVAR
FECHA: 2025
PROF. GUÍA: WALTER MAX-MOERBECK ASTUDILLO

DISEÑO E IMPLEMENTACIÓN DE UN ESPECTRÓMETRO DIGITAL CON SEPARACIÓN DE BANDA LATERAL PARA EL RADIOTELESCOPIO MINI

La radioastronomía ha sido fundamental para el estudio del universo, permitiendo la detección y análisis de señales de radio provenientes de cuerpos celestes. En este contexto, el Radiotelescopio Mini, ubicado en el Observatorio Astronómico Nacional en Cerro Calán, ha sido objeto de actualizaciones tecnológicas para mejorar su capacidad observacional.

En este trabajo se diseñó e implementó un espectrómetro digital con separación de banda lateral utilizando un RFSoC 4x2, con el objetivo de mejorar las capacidades observacionales del radiotelescopio Southern Mini. El sistema desarrollado emplea un híbrido digital para realizar la separación de bandas laterales (LSB y USB). A través del uso de bloques especializados en Simulink y herramientas del grupo CASPER, se desarrollaron modelos capaces de procesar señales en frecuencia intermedia (IF) siguiendo una configuración de separador de banda lateral (2SB), lo que permite una reducción efectiva del ruido y una duplicación del ancho de banda útil.

El diseño incluyó mecanismos de acumulación controlada mediante señales de control del radiotelescopio, así como un sistema de re-cuantización para ajustar los espectros a un sistema de lectura de 32 bits. Se realizaron pruebas tanto en laboratorio como en el radiotelescopio, donde se validó la separación de bandas a través de la métrica SRR (Sideband Rejection Ratio) y se integró el sistema completo con el control del telescopio. Asimismo, se evaluó el desempeño de dos métodos de lectura de datos: uno basado en la librería `casperfpga` y otro mediante un servidor personalizado en C++ con sockets TCP.

Los resultados demostraron que el modelo de 32 bits y 8192 canales espectrales es capaz de obtener valores de SRR superiores a 20 dB, cifra que se considera aceptable considerando que se trata de un híbrido sin calibrar.

Además, el sistema de lectura mediante cliente-servidor usando C++ fue el más eficiente, presentando menor latencia y menor proporción de fallos, por lo que se eligió este método para operar en el telescopio, donde finalmente se logró comunicación entre el FPGA y el sistema completo del telescopio.

A mis padres y hermanas.

Agradecimientos

Quiero agradecer a mi familia por todo el apoyo que me han dado a lo largo de mi vida y durante estos años en la universidad. A mi madre María, por su cariño infinito, por estar siempre dispuesta a apoyarme y enseñarme a ser una mejor persona. A mi padre José, por confiar en mí, compartir sus conocimientos y consejos, y prepararme para la vida. A mi hermana Naty, por ser una gran amiga y acompañarme en tantos momentos. ¡Que nunca falten los conciertos! A mi hermana Jana, quien ha sido como una segunda madre para mí. Gracias por tu cariño y todo el cuidado que me has dado. Estoy muy orgulloso de la linda familia que has formado.

A mis amigos de Limache (y alrededores), especialmente a los cabros de LPJ, gracias por esas noches de Discord, los jueguitos y cada salida y momento épico que hemos compartido.

A los cabros de JooJ SooS, quiero agradecerles por haber sido una parte esencial de mi paso por la universidad. Gracias por las tardes de estudio en los primeros semestres, las risas y las juntas. Con ustedes mis primeros pasos por Santiasco no dieron tanto asco. Espero que nos sigamos viendo seguido.

Quiero hacer mención especial al grupo de Los Tres (no me estoy refiriendo a la banda), los cuales también hicieron más agradable mi paso por la U, especialmente en eléctrica. Hace falta que salga su poolcito.

A toda la gente del MWL. Gracias por el apoyo y por hacer el trabajo más ameno. Especialmente quiero agradecer a Ricardo, por abrirme las puertas y darme la oportunidad para trabajar en Calán. A Max, por su ayuda constante, resolver mis dudas sobre el telescopio y acompañarme cada vez que tenía que realizar alguna prueba en el Mini. A Franco, por la paciencia y buena voluntad para resolver mis dudas, cuya mentoría fue clave para llevar a cabo este proyecto. Finalmente, al profe Walter, por confiar en mí y guiarme en la dirección correcta para darle rumbo a mi trabajo.

Tabla de Contenido

1. Introducción **1**
1.1. Motivación . . . 1
1.2. Objetivos del Trabajo de Título . . . 2

2. Antecedentes **3**
2.1. Marco Teórico . . . 3
2.1.1. Espectro Electromagnético y su Relevancia en Astronomía . . . 3
2.1.2. Radioastronomía . . . 4
2.1.3. Líneas Espectrales . . . 5
2.1.4. El Espectrómetro en Radioastronomía . . . 5
2.1.5. Partes de un receptor astronómico . . . 6
2.1.5.1. Front End . . . 6
2.1.5.2. Back End . . . 6
2.1.6. Radiotelescopio Mini . . . 7
2.1.6.1. Funcionamiento del Mini . . . 8
2.1.6.2. Señal de control y modos de operación del espectrómetro . . . 9
2.1.7. Receptores Heterodinos y Sistema LO . . . 12
2.1.7.1. Receptor SSB . . . 14
2.1.7.2. Receptor DSB . . . 14
2.1.7.3. Receptor 2SB . . . 14
2.1.8. Sideband Rejection Ratio (SRR) . . . 14
2.1.9. Field-Programmable Gate Array . . . 14
2.1.9.1. RFSoC 4x2 . . . 15
2.1.9.2. CASPER . . . 16
2.1.9.2.1 Tutorial 1: Bloques básicos e introducción a Simulink . . . 17
2.1.9.2.2 Tutorial 2: Bloque RFDC y *snapshots* . . . 18
2.1.9.2.3 Tutorial 3: Espectrómetro simple . . . 20
2.2. Estado del Arte . . . 21
2.2.1. Implementaciones de separación de bandas laterales . . . 21

3. Diseño e Implementación del Espectrómetro **26**
3.1. Actualización del Back End del Mini . . . 26
3.2. Diseño del Espectrómetro . . . 27
3.2.1. Funcionamiento como Híbrido Digital . . . 27
3.2.1.1. Control de acumulaciones . . . 29
3.2.2. Modelos desarrollados . . . 30

3.2.2.1. Primeras iteraciones: Modelos de 2048, 8192 y 16384 canales espectrales, y ancho de banda IF de 1.966 GHz 30
3.2.2.1.1 Trabajo adicional: Correlador 34
3.2.2.2. Primera Re-Adaptación: Señal del *RESET* 35
3.2.2.3. Segunda Re-Adaptación: Re-Cuantización de bits 42
3.2.2.4. Tercera Re-Adaptación: Modelo de 8192 canales espectrales con ancho de banda IF de 1.966 GHz + Correlador + Señal de *RESET* + Re-Cuantización de bits 42
3.3. Mediciones en el laboratorio 43
3.3.1. Setup de laboratorio 43
3.3.2. Visualización del espectro 45
3.3.3. Medición de SRR 45
3.3.4. Medición de diferencia de fases para las salidas I y Q 46
3.3.5. Medición del desempeño de la PCB y señal de *RESET* 46
3.3.6. Evaluación de la velocidad de lectura de espectros 47
3.4. Mediciones en el Radiotelescopio 48

4. Resultados y Discusión **51**
4.1. Separación de bandas laterales y funcionamiento del híbrido digital 51
4.1.1. Espectros en LSB y USB 51
4.1.1.1. Modelos de 64 bits 51
4.1.1.2. Modelo de 32 bits 53
4.1.2. Medición de SRR 58
4.1.2.1. Modelos de 64 bits 58
4.1.2.2. Modelo de 32 bits 59
4.1.3. Medición de la diferencia de fase en las señales I y Q 63
4.2. Desempeño de la señal de *RESET* en el laboratorio 65
4.2.1. Verificación del periodo de la señal 65
4.3. Mediciones de velocidad de lectura de espectros 66
4.3.1. Protocolo KATCP y lectura de datos con librería de CASPER en Python 66
4.3.2. Servidor personalizado del RFSoC con C++, Socket programado en C++ y lectura de datos usando Python 67
4.3.3. Resumen con los tiempos de lectura 68
4.3.4. Resumen de lecturas fallidas (> 25 ms) 69
4.3.5. Conclusión de los resultados de velocidad de lectura 70
4.4. Mediciones en el radiotelescopio 70
4.4.1. Evaluación del largo de acumulaciones según la señal *RESET* 70
4.4.2. Comunicación del nuevo espectrómetro con el sistema completo del radiotelescopio y visualización de espectros 71
4.4.3. Prueba de estabilidad en modo calibración y modo de observación . . 77
4.4.3.1. Lectura de una banda 79
4.4.3.2. Lectura de dos bandas 79
4.4.3.3. Conclusión de los resultados de pruebas de estabilidad . . . 79

5. Conclusiones **80**
5.1. Trabajo Futuro 81

Bibliografía **82**

Anexos **84**
A. Desarrollo Matemático de Separación de Bandas Laterales para la configuración 2SB 84
B. Códigos desarrollados 85
C. Resultados adicionales 115
C.1. Espectros adicionales de 64 bits 115
C.2. SRR adicionales de modelos de 64 bits 117
C.3. SRR adicionales de modelos de 32 bits 118
C.4. Mediciones adicionales de diferencia de fase 122
C.5. Gráficos con la latencia en lectura de datos con KATCP y librería de CASPER en Python 124
C.6. Gráficos con la latencia en lectura de datos con servidor y clientes personalizados en C++ e interfaz en Python 126
D. Modelo completo de Simulink 130

Índice de Tablas

4.1. Estadísticas del SRR para modelos de 64 bits con distintas resoluciones espectrales. 59
4.2. Estadísticas del espectro medido con el modelo de 8192 canales, 32 bits, para distintas ganancias y largos de acumulación. 63
4.3. Estadísticas de la diferencia de fase entre señales I y Q en las bandas LSB y USB para modelos de 64 bits. 64
4.4. Tiempos de lectura de espectros desde la RFSoC utilizando distintos métodos, número de canales y cantidad de bandas leídas. 68
4.5. Frecuencia de lecturas fallidas ($>$ 25 ms) para distintas configuraciones. Se incluye el tiempo en que se detectó el primer fallo. 69
4.6. Conteo máximo de acumulaciones completadas entre pulsos de *RESET* para distintos valores de acumulaciones, en modos observación y calibración. 71
4.7. Resumen de pruebas de estabilidad del sistema en distintas configuraciones de lectura de espectros. 78

Índice de Ilustraciones

2.1. Espectro electromagnético [4]. 4
2.2. Opacidad de la atmósfera en función de la longitud de onda. El área sombreada corresponde a la Ventana de Radio [6]. 4
2.3. Espectro rotacional del $^{12}C^{16}O$, donde J representa la transición entre niveles de energía [7]. 5
2.4. Instalaciones del Radiotelescopio Mini 7
2.5. Espectro para la molécula de ^{12}CO 8
2.6. Diagrama general del radiotelescopio Mini 9
2.7. Señal proveniente del RFG para cada modo de operación [10]. 11
2.8. Superposición de bandas laterales [11]. 12
2.9. Esquemas de Receptores Heterodinos [12]. 13
2.10. Tabla comparativa entre FPGA, microcontroladores y circuitos ASIC [15]. . . 15
2.11. Diagrama de bloques con los principales componentes de la RFSoC [16]. 16
2.12. Contador controlable. En la imagen se usa la función FPGA.WRITE_INT() para darle instrucciones al contador. Si se le ingresa un 1, empieza a contar. Si se le ingresa un 0, se detiene el conteo. Finalmente, si se le ingresa un número distinto, como el 2, se reinicia el contador y vuelve a 0. 18
2.13. Sumador controlable. En la imagen se usa la función FPGA.WRITE_INT() para ingresar valores numéricos a dos registros y posteriormente se usa la función FPGA.READ_UINT() para que entregue el resultado de la suma. 18
2.14. Mediciones del bloque *Snapshot* en el Tutorial 2 para distintas frecuencias. . . 19
2.15. Mediciones de los bloques BRAM en el Tutorial 3 para distintas frecuencias. . 20
2.16. Diagrama de bloques del receptor digital separador de banda lateral [11]. . . . 21
2.17. Resultados obtenidos de SRR para un rango de frecuencias de 100.5 a 101.5 GHz. El Oscilador Local está fijado en una frecuencia de 101.0 GHz [11]. . . . 22
2.18. Diagrama de bloques del sistema programado en el FPGA [17]. 23
2.19. SRR del receptor para diferentes LO. (a) Híbrido analógico IF y (b) Híbrido digital IF calibrado [17]. 23
2.20. Diagrama de bloques del sistema de calibración implementado en el FPGA [14]. 24
2.21. Resultados de SRR del receptor. (a) Para banda 7 y (b) para banda 8. Curva roja: sin calibrar (análogico). Curva azul: calibrado (digital) [14]. 25
3.1. Diagrama general actualizado del radiotelescopio Mini 27
3.2. Diagrama del funcionamiento interno de la RFSoC. 28
3.3. Diagrama de Flujo del Sistema de Control de Acumulaciones. 30
3.4. Configuración del bloque RFDC para aumentar el ancho de banda 31
3.5. Bloques Bus, PFB, FFT y sync_gen del espectrómetro en Simulink. 32
3.6. Bloque bram_mult implementado para la multiplicación de constantes de calibración. 33

3.7. Bloque ACC_CNTRL para el control de acumulaciones 34
3.8. Correlación entre las señales de entrada. 35
3.9. Configuración para programar un puerto PMOD para detectar un canto de bajada. En específico, se programó el PMOD con índice 4 en *GPIO bit index*. . . 36
3.10. PCB diseñada en KiCad para implementar la señal del *RESET* 37
3.11. Modelo en 3D de la PCB diseñada con KiCad para la señal de *RESET*. 38
3.12. Fotos de la PCB diseñada para la señal de *RESET*. 39
3.13. Diagrama de Flujo del Sistema de Control de Acumulaciones con *RESET* integrado. 40
3.14. Sistema lógico del contador de espectros ACC_CNTR 41
3.15. Re-Cuantización de bits con el bloque ROUND0 42
3.16. *Front End* analógico para frecuencias RF 43
3.17. Imagen de una de las RFSoC4x2 disponibles en el *MWL*, en Cerro Calán. . . . 44
3.18. Setup de laboratorio 45
3.19. Setup de medición para evaluar el desempeño de la PCB 47
3.20. RFSoC 4x2 conectado al radiotelescopio Mini 49
4.1. Visualización del espectro de 64 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz. Se observan las bandas LSB y USB. 52
4.2. Visualización del espectro de 64 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz. Se observan las bandas LSB y USB. 52
4.3. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 1. Se observan las bandas LSB y USB. . 53
4.4. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 1. Se observan las bandas LSB y USB. . 54
4.5. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 2^{10}. Se observan las bandas LSB y USB. 55
4.6. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 2^{10}. Se observan las bandas LSB y USB. 55
4.7. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 2^{15}. Se observan las bandas LSB y USB. 56
4.8. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 2^{15}. Se observan las bandas LSB y USB. 56
4.9. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 2^{20}. Se observan las bandas LSB y USB. 57
4.10. Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 2^{20}. Se observan las bandas LSB y USB. 57
4.11. SRR para modelo de 64 bits y 8192 canales espectrales. 59
4.12. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 1 y cantidad de acumulaciones 2^{10}. 60
4.13. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^9 y cantidad de acumulaciones 2^{10}. 60
4.14. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{10} y cantidad de acumulaciones 2^{10}. 61
4.15. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{11} y cantidad de acumulaciones 2^{10}. 61
4.16. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{10} y cantidad de acumulaciones 2^9. 62

4.17. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{11} y cantidad de acumulaciones 2^{8}. 62
4.18. Diferencia de fase entre las señales I y Q para el modelo de 8192 canales espectrales. 64
4.19. Medición del tiempo entre flancos de bajada del *RESET* para un periodo de 53 ms. 65
4.20. Medición del tiempo entre flancos de bajada del *RESET* para un periodo de 60 ms. 66
4.21. Espectro visualizado en el modo observación usando ROACH 72
4.22. Espectro visualizado en el modo calibración usando ROACH 72
4.23. Espectro de las bandas LSB y USB con modelo de 64 bits en RFSoC. 73
4.24. Espectro de las bandas LSB y USB con modelo de 32 bits en RFSoC. Se usó una ganancia de 2^{12}. 74
4.25. Espectro de las bandas LSB y USB con modelo de 32 bits en RFSoC. Se usó una ganancia de 2^{13}. 74
4.26. Espectro de la banda USB en modo observación. Se presentan 512 canales espectrales, tomando como punto de partida el canal 768. 75
4.27. Espectro de la banda USB en modo calibración. Se presentan 512 canales espectrales, tomando como punto de partida el canal 768. 75
4.28. Espectro visualizado desde el computador MAC en el modo observación usando RFSoC. 76
4.29. Espectro visualizado desde el computador MAC en el modo calibración usando RFSoC. 77
4.30. interfaz del MAC donde se monitorean características de operación del Radiotelescopio Mini. 78
A.1. Circuito de la configuración 2SB 84
C.1. Visualización del espectro de 64 bits para 2048 canales, con una frecuencia de entrada RF de 2500 MHz. Se observan las bandas LSB y USB. 115
C.2. Visualización del espectro de 64 bits para 2048 canales, con una frecuencia de entrada RF de 3500 MHz. Se observan las bandas LSB y USB. 116
C.3. Visualización del espectro de 64 bits para 16384 canales, con una frecuencia de entrada RF de 2500 MHz. Se observan las bandas LSB y USB. 116
C.4. Visualización del espectro de 64 bits para 16384 canales, con una frecuencia de entrada RF de 3500 MHz. Se observan las bandas LSB y USB. 117
C.5. SRR para modelo de 64 bits y 2048 canales espectrales. 117
C.6. SRR para modelo de 64 bits y 16384 canales espectrales. 118
C.7. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{6} y cantidad de acumulaciones 2^{10}. 118
C.8. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{7} y cantidad de acumulaciones 2^{10}. 119
C.9. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{8} y cantidad de acumulaciones 2^{10}. 119
C.10. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{11} y cantidad de acumulaciones 2^{9}. 120
C.11. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{12} y cantidad de acumulaciones 2^{10}. 120
C.12. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{12} y cantidad de acumulaciones 2^{7}. 121

C.13. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{13} y cantidad de acumulaciones 2^{10}. 121
C.14. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{13} y cantidad de acumulaciones 2^{6}. 122
C.15. SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{20} y cantidad de acumulaciones 2^{10}. 122
C.16. Diferencia de fase entre las señales I y Q para el modelo de 2048 canales espectrales. 123
C.17. Diferencia de fase entre las señales I y Q para el modelo de 16384 canales espectrales. 123
C.18. Latencia de lectura de una banda de 512 canales espectrales usando el servidor de KATCP y `casperfpga`. 124
C.19. Latencia de lectura de una banda de 1024 canales espectrales usando el servidor de KATCP y `casperfpga`. 124
C.20. Latencia de lectura de una banda de 2048 canales espectrales usando el servidor de KATCP y `casperfpga`. 125
C.21. Latencia de lectura de una banda de 4096 canales espectrales usando el servidor de KATCP y `casperfpga`. 125
C.22. Latencia de lectura de una banda de 8192 canales espectrales usando el servidor de KATCP y `casperfpga`. 126
C.23. Latencia de lectura de una banda de 512 canales espectrales usando servidor en C++ y Sockets de Python. 126
C.24. Latencia de lectura de una banda de 1024 canales espectrales usando servidor en C++ y Sockets de Python. 127
C.25. Latencia de lectura de una banda de 2048 canales espectrales usando servidor en C++ y Sockets de Python. 127
C.26. Latencia de lectura de una banda de 4096 canales espectrales usando servidor en C++ y Sockets de Python. 128
C.27. Latencia de lectura de dos bandas de 512 canales espectrales usando servidor en C++ y Sockets de Python. 128
C.28. Latencia de lectura de dos bandas de 1024 canales espectrales usando servidor en C++ y Sockets de Python. 129
C.29. Latencia de lectura de dos bandas de 2048 canales espectrales usando servidor en C++ y Sockets de Python. 129
C.30. Latencia de lectura de dos bandas de 4096 canales espectrales usando servidor en C++ y Sockets de Python. 130
D.1. Modelo del Espectrómetro Digital con Separación de Banda Lateral diseñado en Simulink para 8192 canales espectrales. Incluye bloques con correlación para medir la diferencia de fase, señal de control RESET y Re-Cuantización de bits para obtener espectros de 32 bits. 131

Capítulo 1

Introducción

1.1. Motivación

La exploración del universo a través de la radioastronomía ha permitido desentrañar muchos de los misterios que rodean la formación y evolución de galaxias, estrellas y estructuras moleculares en el espacio interestelar. En particular, un fenómeno de estudio clave son las líneas espectrales, que consisten en patrones del espectro electromagnético provenientes de fuentes gaseosas e ionizadas. Estas líneas son producidas por transiciones electrónicas en átomos o moléculas, las cuales emiten o absorben señales de radiofrecuencia en bandas estrechas. Las líneas espectrales son esenciales para entender la composición, estructura y dinámica del universo.

Dentro de este contexto, los radiotelescopios han jugado un rol crucial al detectar y analizar las emisiones de diferentes compuestos químicos en el cosmos. El radiotelescopio denominado *Southern Millimeter-Wave Telescope* (*SMWT* por sus siglas en inglés), o *Southern Mini*, fue parte de un proyecto en conjunto entre la Universidad de Chile y la Universidad de Columbia durante los años 80, desde el Observatorio de Cerro Tololo, por estudiar el gas molecular en la Galaxia visible desde el Hemisferio Sur. Junto con su telescopio gemelo en la Universidad de Columbia, y ahora en la Universidad de Harvard, el *Northern Mini*, estos equipos lograron obtener el mapa más extenso y uniforme del gas molecular en la Vía Láctea, logro obtenido analizando la línea rotacional del monóxido de carbono (CO) a 115 GHz [1] [2] [3].

Desde el 2011, es operado por el Laboratorio de Ondas Milimétricas (*Millimeter-Wave Laboratory*, *MWL* por sus siglas en inglés) en el Observatorio Astronómico Nacional en Cerro Calán, perteneciente al Departamento de Astronomía de la Universidad de Chile. Desde entonces, el telescopio ha sido utilizado tanto para labores de docencia en pregrado de Astronomía, como para desarrollos tecnológicos avanzados junto al Departamento de Ingeniería Eléctrica.

Dentro de los desarrollos tecnológicos que se han llevado a cabo, se encuentra la implementación de un receptor y espectrómetro digital para realizar separación de banda lateral. Este espectrómetro utiliza una configuración de separación de banda lateral (2SB). Esta actualización es importante porque los receptores 2SB distinguen entre las bandas laterales de señal e imagen, proporcionando una reducción de ruido atmosférico en comparación con la configuración de doble banda lateral (DSB), y duplican el ancho de banda en comparación

a los receptores de banda lateral única (SSB). Este avance tecnológico se realizó utilizando dispositivos FPGA (*Field Programmable Gate Array*), particularmente la ROACH (*Reconfigurable Open Architecture Computing Hardware*).

Actualmente, el radiotelescopio, también conocido comúnmente como "Mini" debido a su pequeño tamaño, es parte de un proyecto de actualización de su *front-end* y *back-end*, con el objetivo de ser utilizado en docencia y aumentar su potencial para la investigación científica en la banda entre 67 a 116 GHz, que corresponde a las bandas 2 y 3 del Atacama Large Millimeter/submillimeter Array, lo que permitirá el estudio de múltiples líneas simultáneamente.

Hoy en día, existen circuitos FPGA más avanzados y especializados, como el RFSoC (*Radio Frequency System-on-Chip*), que integran convertidores analógico-digitales (ADC) y digital-analógicos (DAC) de alta velocidad. Esta integración permite manejar señales de radiofrecuencia (RF) de manera más eficiente y compacta. El presente trabajo propone diseñar e implementar un espectrómetro digital que realice separación de banda lateral utilizando este nuevo tipo de tecnología. Este desarrollo no solo mejorará la capacidad observacional del radiotelescopio, sino que también proporcionará una plataforma para la formación práctica de estudiantes y el desarrollo de tecnologías de vanguardia en el ámbito de la radioastronomía.

1.2. Objetivos del Trabajo de Título

El objetivo general de este Trabajo de Título, es diseñar e implementar un espectrómetro digital con separación de banda lateral para el Radiotelescopio Mini. Dicho trabajo, se llevó a cabo en el Laboratorio de Ondas Milimétricas de la Universidad de Chile usando como herramienta principal un dispositivo FPGA especializado para el procesamiento de señales de radiofrecuencia.

Dentro de los objetivos específicos del proyecto se encuentran:

- Diseñar y compilar un modelo en Simulink de un espectrómetro digital con separación de banda lateral usando constantes de calibración ideales, utilizando librerías especializadas y programarlo en una placa RFSoC 4x2.
- Realizar mediciones de *Sideband Rejection Ratio* para evaluar la calidad de separación de bandas y desempeño del espectrómetro.
- Aumentar el número de canales espectrales para abordar distintas resoluciones de observación.
- Implementar comunicación entre la RFSoC y el sistema de control del telescopio, y realizar mediciones de prueba del espectrómetro.

Capítulo 2

Antecedentes

Para tener una mayor comprensión del proyecto llevado a cabo, es necesario poner en contexto el trabajo que se realizó y conocer los principales fundamentos teóricos relacionados para su desarrollo, además de repasar los principales proyectos ligados a la separación digital de bandas laterales aplicados en receptores astronómicos.

2.1. Marco Teórico

2.1.1. Espectro Electromagnético y su Relevancia en Astronomía

La astronomía es la ciencia que estudia los cuerpos celestes y los fenómenos que ocurren fuera de la atmósfera terrestre. Su propósito es observar, analizar y comprender el origen, evolución y características de objetos como estrellas, planetas, galaxias y el universo en su conjunto. Desde tiempos antiguos, la observación del cielo ha sido una herramienta clave para la humanidad, permitiendo a las civilizaciones, por ejemplo, medir el tiempo, crear calendarios, navegar los océanos, entre otras aplicaciones. A lo largo de la historia, la astronomía ha evolucionado y, en tiempos recientes, se ha consolidado como una disciplina fundamental para buscar respuestas a las preguntas más profundas sobre el origen y la estructura del universo.

Aunque la astronomía comenzó siendo estudiada principalmente a través de observaciones en el espectro visible, el desarrollo de instrumentos electrónicos, como los receptores especializados, ha permitido detectar radiación electromagnética más allá de lo que el ojo humano puede percibir. Este avance ha revolucionado la ciencia, ampliando significativamente nuestra capacidad para explorar el cosmos y entender la complejidad y vastedad del universo.

En la Figura 2.1, se muestra cómo se distribuye el espectro electromagnético en función de la longitud de onda y la frecuencia. Se puede observar que la luz visible corresponde solamente a una pequeña fracción de dicho espectro.

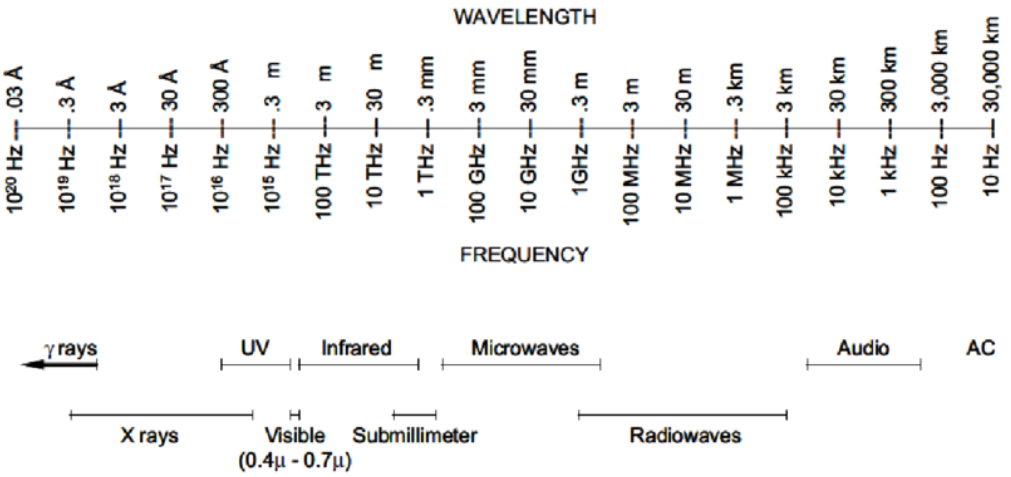


Figura 2.1: Espectro electromagnético [4].

2.1.2. Radioastronomía

La expansión del rango de frecuencias observables ha dado lugar a la radioastronomía, definida como el estudio de las emisiones de radio provenientes de fuentes celestes. Afortunadamente, desde la superficie de la Tierra, la atmósfera es mayormente transparente a las ondas de radio en ciertos rangos de frecuencia.

El rango de frecuencias en radioastronomía está determinado por dos factores principales: la opacidad atmosférica y el ruido cuántico en amplificadores coherentes. En el extremo superior, la frontera entre la radioastronomía y la astronomía del infrarrojo lejano se encuentra aproximadamente en $\nu \sim 1\,\mathrm{THz}$, correspondiente a una longitud de onda de $\lambda = c/\nu \sim 0.3$ mm, donde c es la velocidad de la luz en el vacío. En el extremo inferior, la ionosfera terrestre impone un límite de baja frecuencia al reflejar las ondas de radio extraterrestres con frecuencias menores a $\nu \sim 10\,\mathrm{MHz}$ (o $\lambda \sim 30$ m) [5].

La Figura 2.2 muestra la opacidad de la atmósfera en función de la longitud de onda, destacando la región de frecuencias en la que las ondas de radio pueden atravesar la atmósfera sin ser absorbidas o dispersadas significativamente. A esta región se le conoce como la Ventana de Radio, y es fundamental para la observación astronómica desde la Tierra.

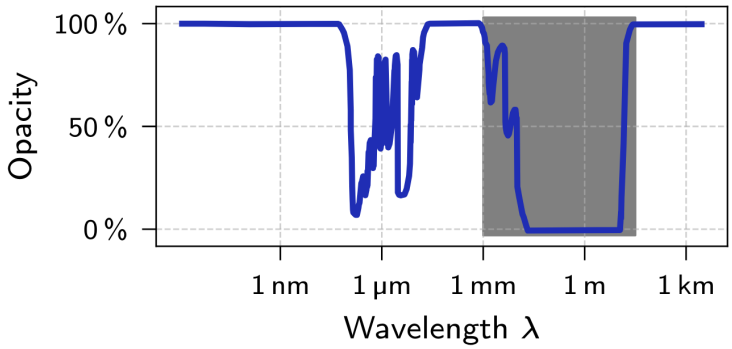


Figura 2.2: Opacidad de la atmósfera en función de la longitud de onda. El área sombreada corresponde a la Ventana de Radio [6].

2.1.3. Líneas Espectrales

Gran parte del conocimiento actual sobre la estructura de la Vía Láctea proviene de observaciones en radioastronomía. Uno de los aspectos clave de estas observaciones es el estudio de las líneas espectrales, las cuales corresponden a patrones característicos en el espectro electromagnético generados por transiciones de energía en átomos y moléculas.

Las líneas espectrales pueden clasificarse en líneas de emisión y líneas de absorción. Las primeras ocurren cuando un electrón en un átomo o molécula desciende a un nivel de energía más bajo, emitiendo un fotón con una frecuencia característica. En contraste, las líneas de absorción aparecen cuando un electrón absorbe energía para ascender a un nivel superior.

Un ejemplo relevante en radioastronomía es la emisión de la molécula de monóxido de carbono ($^{12}C^{16}O$). En su transición fundamental (del nivel rotacional $J = 1$ al $J = 0$), emite un fotón con una frecuencia de 115.3 GHz [7]. En la Figura 2.3 se presenta el espectro rotacional de emisión de esta molécula.

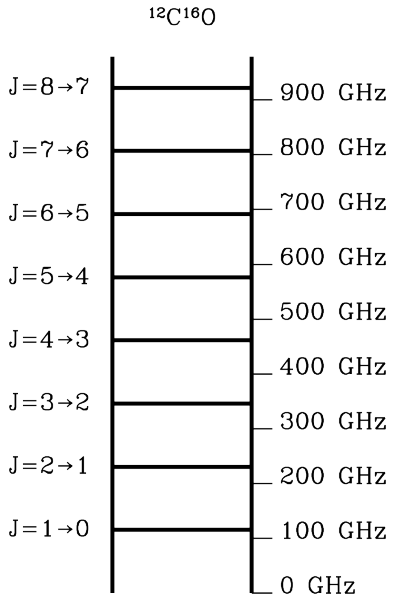


Figura 2.3: Espectro rotacional del $^{12}C^{16}O$, donde J representa la transición entre niveles de energía [7].

En particular, la banda de 67-116 GHz (Banda 2+3 de ALMA) alberga numerosas líneas de emisión de moléculas de alto interés científico. Entre ellas, el monóxido de carbono es especialmente relevante, ya que se utiliza como trazador del hidrógeno molecular, el componente más abundante de las nubes moleculares y una fracción significativa de la masa molecular en nuestra galaxia.

2.1.4. El Espectrómetro en Radioastronomía

Un espectrómetro es un dispositivo utilizado para analizar el contenido espectral de una señal, es decir, medir cómo se distribuye su potencia en función de la frecuencia. En radioastronomía, los espectrómetros permiten identificar líneas de emisión o absorción asociadas a transiciones atómicas o moleculares, lo que resulta esencial para estudiar la composición,

dinámica y condiciones físicas del medio interestelar.

Específicamente, un espectrómetro mide la densidad espectral de potencia (*Power Spectral Density*, PSD), que representa la potencia presente por unidad de ancho de banda. Esta medida permite, por ejemplo, detectar líneas espectrales como la emisión de hidrógeno neutro a 21 cm, que proporciona información clave sobre la estructura galáctica [8].

En términos prácticos, un espectrómetro toma una señal en el dominio temporal y calcula su densidad espectral de potencia, que representa cuánta energía contiene la señal en cada componente de frecuencia. Este proceso es esencial en observaciones espectroscópicas, donde se busca identificar y caracterizar señales de interés en medio de un entorno ruidoso.

2.1.5. Partes de un receptor astronómico

En el contexto de la radioastronomía, es importante distinguir dos etapas claves: el *Front End* y el *Back End*.

2.1.5.1. Front End

El *Front End* constituye la primera etapa del sistema receptor, cuya función es captar, amplificar y procesar las señales de radio provenientes del espacio antes de enviarlas a las siguientes etapas. Esta etapa es fundamental, ya que las señales astronómicas son extremadamente débiles y están inmersas en ruido, por lo que el *Front End* debe operar con alta sensibilidad y con un mínimo nivel de ruido añadido.

Una etapa clave dentro del *Front End* es el proceso de mezclado, donde la señal captada se combina con una señal generada localmente, proveniente del Oscilador Local (*Local Oscillator*, LO por sus siglas en inglés). Este proceso permite convertir la señal de su frecuencia original, típicamente muy alta, a una Frecuencia Intermedia (*Intermediate Frequency*, IF por sus siglas en inglés) más manejable para su posterior procesamiento en etapas digitales.

2.1.5.2. Back End

El *Back End* es la etapa del sistema receptor encargada del procesamiento y análisis de las señales previamente captadas y acondicionadas por el *Front End*. El *Back End* se ocupa de su manipulación, digitalización y almacenamiento para su posterior análisis científico.

En particular, la implementación del espectrómetro digital forma parte de esta etapa. Este módulo es responsable de realizar el análisis espectral detallado en tiempo real, a partir de las señales ya amplificadas y convertidas. En la actualidad, la mayoría de los espectrómetros utilizados en radioastronomía son digitales, aprovechando los avances en electrónica y procesamiento digital de señales.

El espectrómetro desarrollado en este proyecto está completamente integrado en el *Back End* del sistema y ha sido implementado sobre la plataforma RFSoC 4x2, la cual se detallará más adelante.

2.1.6. Radiotelescopio Mini

En el presente trabajo de título se va a trabajar con el *Southern Millimeter Wave Telescope* (*SMWT*), también conocido como *Southern Mini*, o simplemente como Mini. Actualmente este radiotelescopio se encuentra en la cima del cerro Calán, en la comuna de Las Condes, siendo parte del Departamento de Astronomía de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. En la Figura 2.4, se tiene una imagen del telescopio, así como el edificio en el que se encuentra.

(a) Radiotelescopio Mini

(b) Edificio en el que se encuentra el Mini

Figura 2.4: Instalaciones del Radiotelescopio Mini

El Mini fue construido originalmente para la observación de líneas de monóxido de carbono, enfocando su estudio en las transiciones del $^{12}C^{16}O$ en 115.3 GHz y del $^{13}C^{16}O$ en 110.2 GHz [1] [2]. Actualmente, el *SMWT* opera en una banda de 84 a 116 GHz, aunque se espera que en el futuro su rango de observación se amplíe hasta cubrir toda la banda 2+3 de ALMA.

Como ejemplo de las observaciones realizadas con el Mini, en la Figura 2.5, se presenta el espectro del monóxido de carbono $^{12}C^{16}O$ (abreviado como 12CO en la imagen). En este gráfico, el eje horizontal representa la velocidad radial de la fuente celeste, mientras que el eje vertical muestra la temperatura de brillo medida por el radiotelescopio.

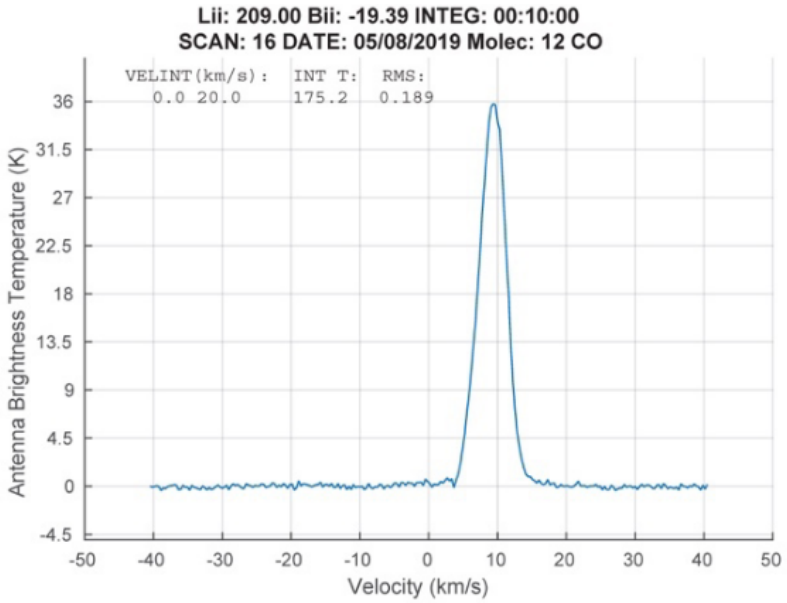


Figura 2.5: Espectro para la molécula de ^{12}CO

En radioastronomía espectroscópica, es común representar el eje horizontal en unidades de velocidad (km/s) en lugar de frecuencia (Hz). Esta conversión se realiza utilizando el efecto Doppler, que relaciona el desplazamiento en frecuencia con la velocidad radial de la fuente astronómica respecto al observador. La forma clásica del efecto Doppler se expresa como:

$$\frac{\Delta f}{f} = \frac{\Delta v}{c} \tag{2.1}$$

donde Δf representa la diferencia entre la frecuencia emitida y la observada, Δv es la velocidad relativa entre la fuente y el observador, f es la frecuencia emitida y c es la velocidad de la luz [9].

En el contexto del análisis espectral, esta relación permite reinterpretar los canales en frecuencia como canales en velocidad, facilitando el estudio de dinámicas internas en nubes moleculares o galaxias.

2.1.6.1. Funcionamiento del Mini

Actualmente, el Mini sigue el diagrama de bloques de la Figura 2.6. El receptor incluye el *Front End* y la parte analógica del telescopio, que comprende la antena, amplificadores, filtros, entre otros componentes. Por otro lado, el *Back End* está conformado principalmente por un espectrómetro digital implementado en un FPGA de la plataforma ROACH (*Reconfigurable Open Architecture Computing Hardware*). Este dispositivo captura las componentes en frecuencia del receptor, las digitaliza y las procesa para realizar la separación de bandas laterales, procedimiento que se detallará más adelante.

A continuación, se muestran las características técnicas del espectrómetro:

- El espectrómetro actual del Mini opera con un ancho de banda de 500 MHz.
- Utiliza una transformada rápida de Fourier (FFT) de tamaño 2048, lo que resulta en

una resolución espectral de aproximadamente 244 kHz por canal.

- Actualmente, solo se están capturando 512 canales espectrales, lo que limita la cobertura del espectro analizado.

Como interfaz de comunicación, el sistema cuenta con un microcontrolador PIC32[1], el cual actúa como nexo entre el computador MAC que visualiza los espectros (limitado a 512), el espectrómetro y el sistema de control del telescopio.

Finalmente, el sistema de control es una estructura más compleja que manipula tanto las partes mecánicas del telescopio, como su movimiento y posición, así como también componentes analógicos y digitales involucrados en la medición de espectros.

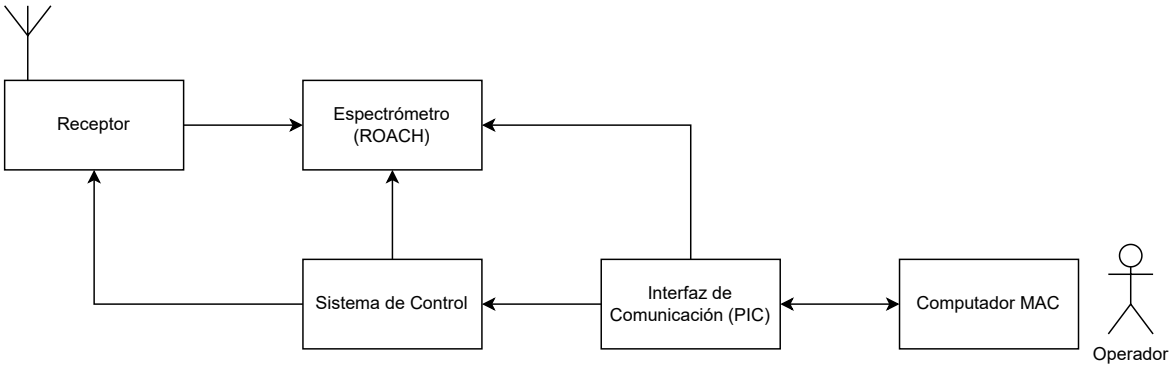


Figura 2.6: Diagrama general del radiotelescopio Mini

2.1.6.2. Señal de control y modos de operación del espectrómetro

Como se mencionó previamente, el sistema de control del *SMWT* abarca distintos aspectos clave para su funcionamiento, incluyendo la operación del espectrómetro. Este espectrómetro trabaja en dos modos: *Spectral Line Observation* (*SPLOBS*, o simplemente modo de observación) y modo de calibración (*CAL*). Ambos modos son gestionados por un Generador de Referencia (RFG), el cual genera señales de control que determinan la frecuencia con la que se mide un espectro [10].

El Generador de Referencia (RFG) es el encargado de controlar los ciclos de integración y retención del banco de filtros del espectrómetro. Su funcionamiento varía dependiendo del modo de operación seleccionado: *Spectral Line Observation* (*SPLOBS*) o *Calibración* (*CAL*).

En el modo *SPLOBS*, el RFG opera de manera autónoma, utilizando su propio reloj interno para generar un ciclo de control con una duración total de 53 ms. Este ciclo se compone de dos fases: una integración de 48 ms, seguida de un período de 5 ms en el que los datos son retenidos para su lectura a través del conversor analógico-digital (A/D).

Por otro lado, en el modo *CAL*, el ciclo de control del RFG está sincronizado con la ro-

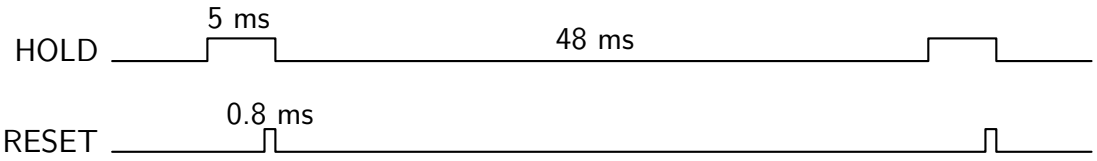
[1] https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/pic32-32-bit-mcus

tación de la *chopper wheel*[2]. En este caso, se genera un ciclo de integración de 25 ms cada vez que la bocina del receptor es completamente cubierta o descubierta por la rueda, lo que ocurre aproximadamente cada 50 ms.

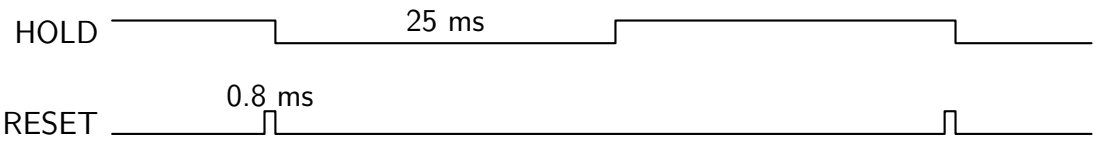
El RFG gestiona el espectrómetro mediante dos señales de control: *HOLD* y *RESET*. La operación de estas señales es la siguiente:

- Cuando *HOLD* y *RESET* están en nivel bajo (*LOW*), el espectrómetro se encuentra en modo de integración, lo que quiere decir que el dispositivo está midiendo espectros.
- Cuando *HOLD* se encuentra en nivel alto (*HIGH*), la integración se detiene y los datos se mantienen estables para su conversión, por lo que el espectrómetro para de medir.
- Cuando *RESET* se activa (*HIGH*), los integradores se vacían, preparando el sistema para un nuevo ciclo de integración y medición del espectrómetro.

En el siguiente diagrama, se tienen las señales del reloj que entrega el Generador de Referencia [10] para el modo *SPLOBS*:

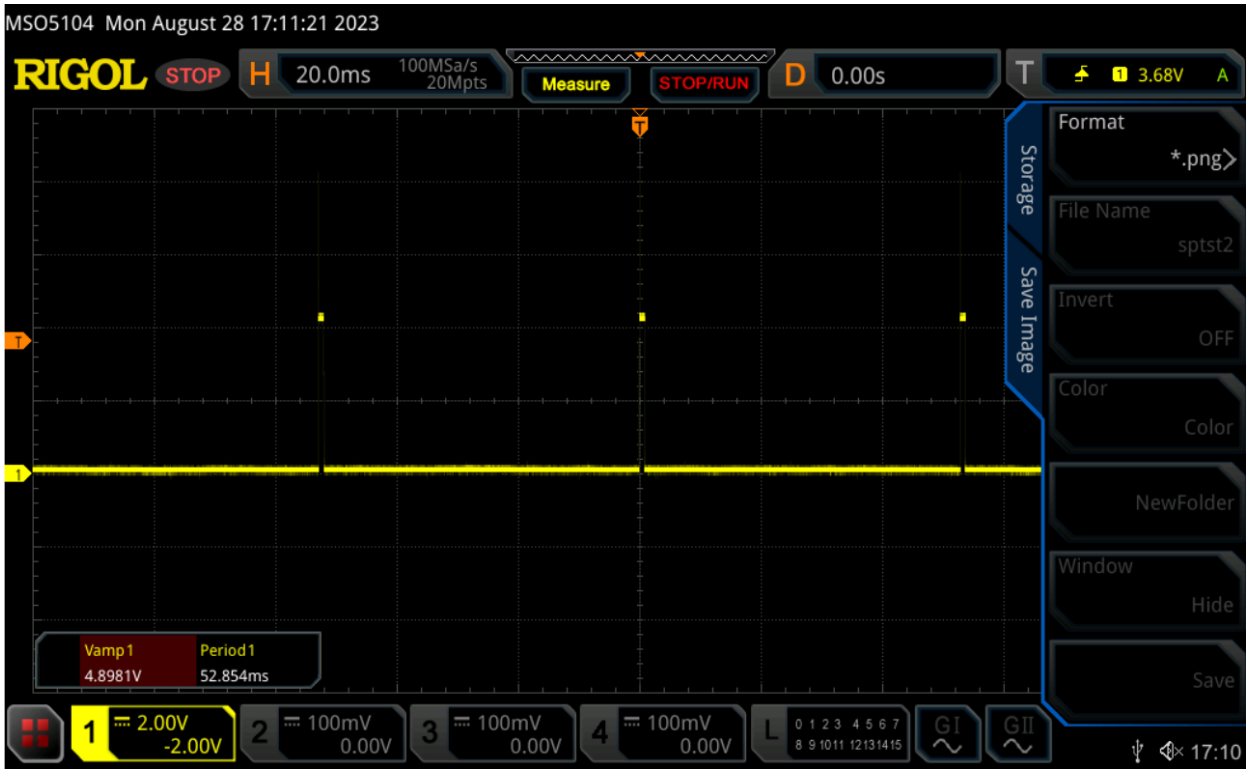


Por otro lado, en el siguiente diagrama se tienen las señales para el modo *CAL*:

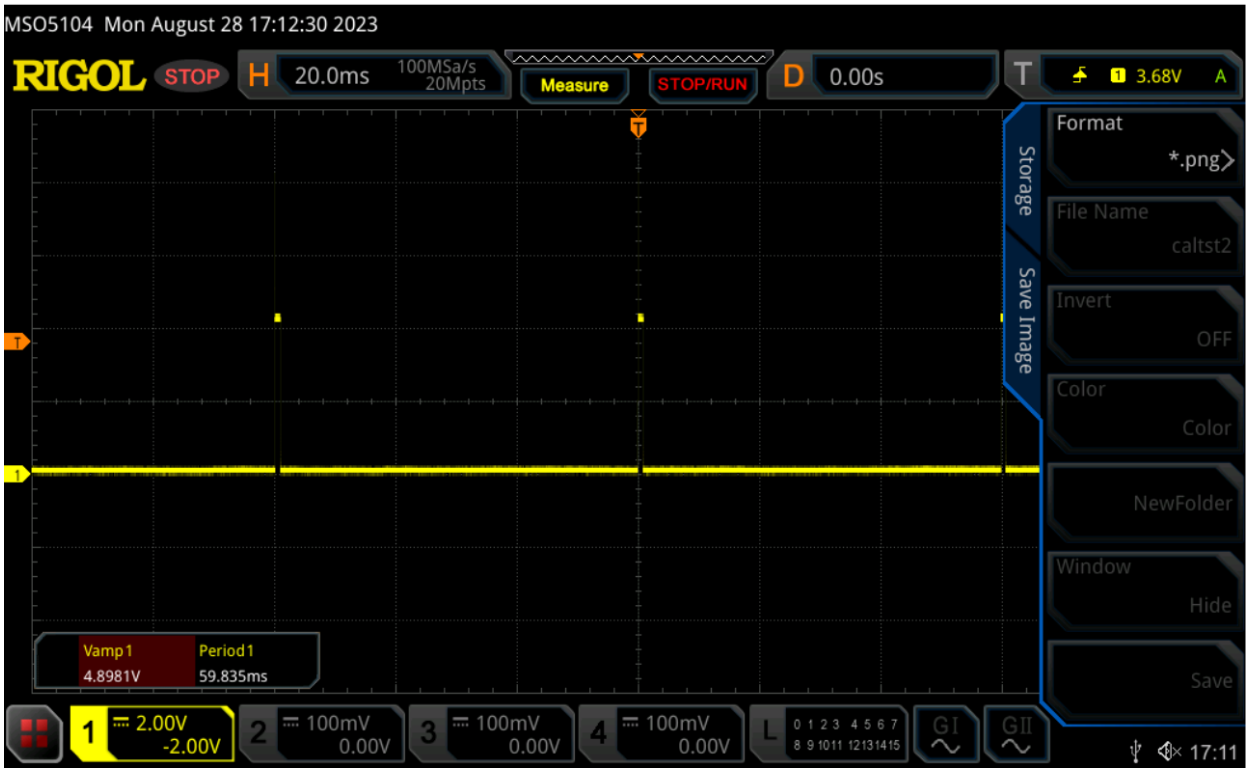


En ambos modos de operación, la señal *RESET* tiene una duración de 0.8 ms y se activa 1 ms antes de que inicie un nuevo periodo de integración. Esta señal es clave para garantizar la sincronización y el correcto funcionamiento del espectrómetro. Finalmente, en la Figura 2.7, se tienen las mediciones en un osciloscopio de la señal del RFG que recibe el ROACH del Mini para cada modo. De estas mediciones se puede notar que el espectrómetro en modo *SPLOBS* recibe una señal con un periodo de 53 ms, mientras que en modo *CAL* cada 60 ms.

[2] La *chopper wheel* es un dispositivo mecánico rotatorio utilizado para la calibración en sistemas de radioastronomía. Consiste en una rueda con sectores opacos y transparentes que alternan entre bloquear y permitir el paso de la radiación hacia el receptor.



(a) Modo *SPLOBS*



(b) Modo *CAL*

Figura 2.7: Señal proveniente del RFG para cada modo de operación [10].

Cabe destacar que el modo *CAL* es particularmente sensible a retrasos o desincronizaciones en la adquisición de datos. Esto se debe a que, si el espectrómetro no logra completar

su medición dentro del tiempo estipulado (25 ms de integración), el sistema considera que ocurrió una falla y detiene la adquisición. No obstante, en la práctica, este modo se activa por períodos breves —generalmente no más de 15 segundos— por lo que el riesgo de interrupción prolongada es bajo. Por el contrario, el modo *SPLOBS* es más tolerante a eventuales fallos de sincronización, ya que no detiene el proceso de adquisición ante errores puntuales. Además, dispone de un mayor tiempo de integración (48 ms), lo que brinda un margen adicional para completar las operaciones internas del espectrómetro sin afectar la continuidad de la observación. Esta diferencia en el tratamiento de errores hace que el sistema en modo observación resulte más robusto ante variaciones temporales en la lectura de datos o latencias en la comunicación.

2.1.7. Receptores Heterodinos y Sistema LO

Los receptores usados en radioastronomía tienen un funcionamiento limitado que no les permite procesar directamente señales de muy alta frecuencia, como las que provienen del cielo. Como se mencionó previamente en la etapa del *Front-End*, para mitigar esto, la señal entrante se convierte a una frecuencia más baja, conocida como IF, para su posterior procesamiento. En estos sistemas se utiliza un dispositivo no lineal llamado mezclador (*mixer*) que combina la señal entrante con una señal de referencia conocida como LO. Este tipo de receptores se conoce como receptores heterodinos, o coherentes, y el proceso de disminución de frecuencia se denomina conversión hacia abajo o *down-conversion* en inglés.

No obstante, en este proceso, las frecuencias por encima del LO, denominadas banda lateral superior (*upper sideband*, USB por sus siglas en inglés), y las frecuencias por debajo del LO, denominadas banda lateral inferior (*lower sideband*, LSB por sus siglas en inglés), se reducen a la misma frecuencia IF. Esto provoca superposición en las bandas resultantes de la heterodinación, introduciendo ruido en la observación de la banda deseada [11]. La Figura 2.8 proporciona una representación visual de este problema.

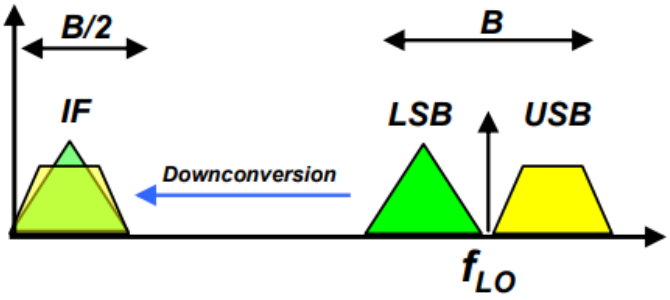
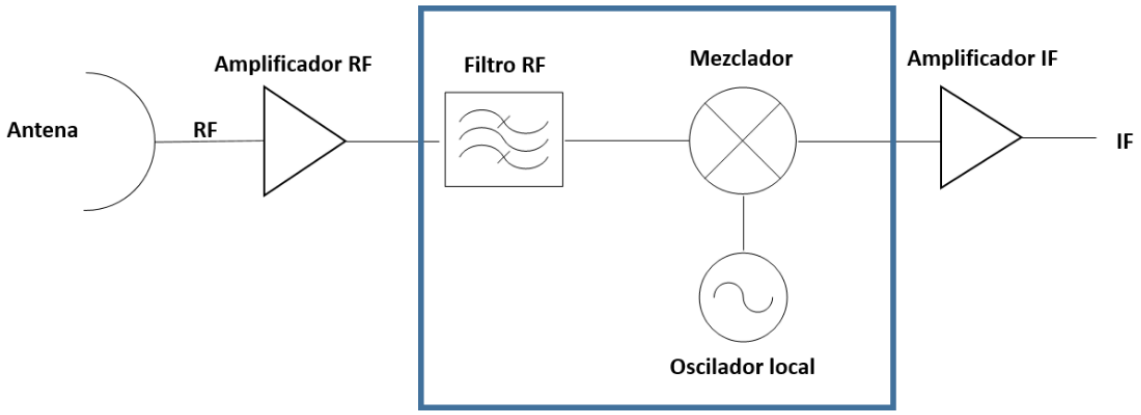
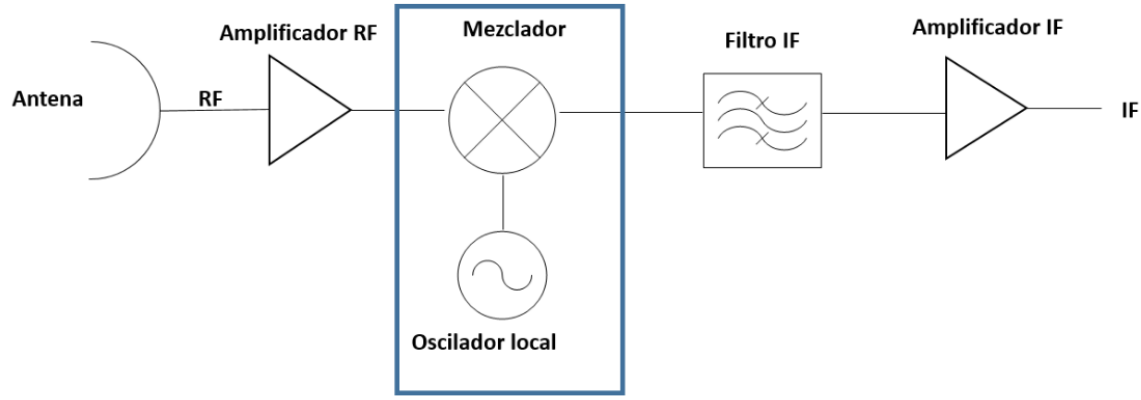


Figura 2.8: Superposición de bandas laterales [11].

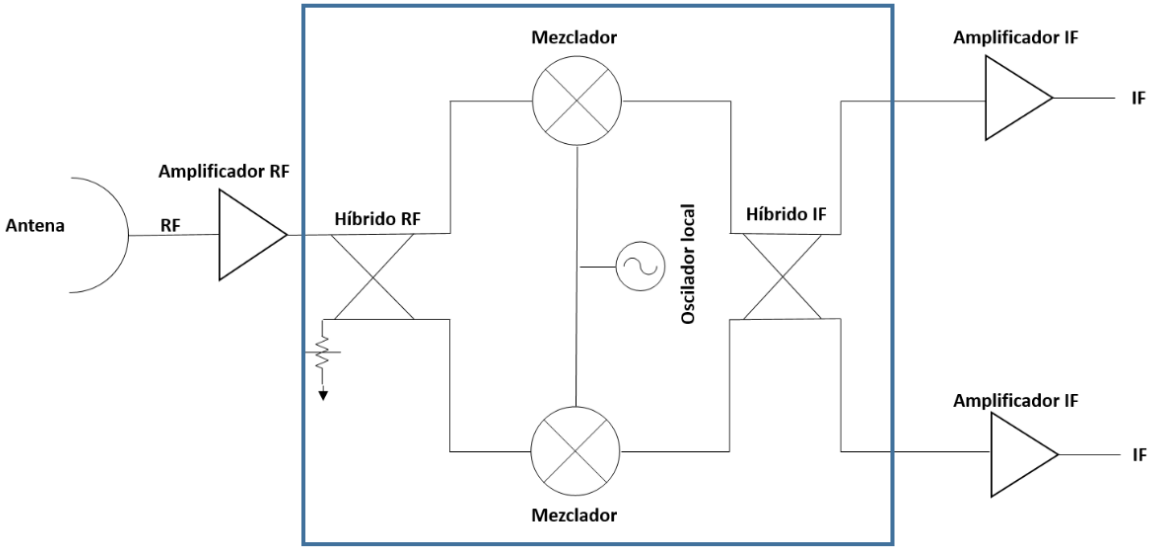
Avances tecnológicos han permitido procesar más de una banda simultáneamente. Según sea el caso, cualquiera de las dos bandas puede o no ser eliminada, dependiendo del tipo de configuración de receptor que se requiera. Entre los receptores que procesan más de una banda lateral se encuentran las configuraciones de banda lateral única (*Single Sideband*, SSB), banda lateral doble (*Double Sideband*, DSB) y separador de banda lateral (*Sideband Separating*, 2SB). En la Figura 2.9 se muestra la configuración correspondiente para cada uno de estos receptores.



(a) Receptor SSB. En esta configuración, se emplea un filtro en la señal RF para eliminar la banda lateral no deseada, permitiendo que solo una de las bandas se convierta a frecuencia intermedia.



(b) Receptor DSB. A diferencia del caso anterior, no se aplica filtrado en la señal RF, por lo que ambas bandas laterales se convierten simultáneamente a la misma frecuencia intermedia, generando superposición.



(c) Receptor 2SB. Esta configuración permite separar ambas bandas laterales aplicando desfases mediante híbridos de cuadratura. Como resultado, se obtienen por separado las señales IF correspondientes a la banda lateral superior e inferior.

Figura 2.9: Esquemas de Receptores Heterodinos [12].

2.1.7.1. Receptor SSB

Para la configuración SSB se elimina la banda RF no deseada. Esto permite evitar la superposición entre las bandas laterales tras disminuir las frecuencias recibidas a la banda IF de salida. En la Figura 2.9.a se puede apreciar este tipo de receptor, específicamente lo que está encerrado en el recuadro azul constituye la configuración SSB, donde se coloca un filtro antes del mezclador.

2.1.7.2. Receptor DSB

La configuración DSB, sigue el esquema de la Figura 2.9.b. Dentro del recuadro azul, se puede ver que se utiliza un único *mixer* y LO. Si bien esta implementación es simple y económica, no distingue entre las bandas laterales en el proceso de *downconversion*, lo que produce superposición de las bandas en la frecuencia IF, como se vió en la Figura 2.8, e introduce un nivel significativo de ruido en las observaciones.

2.1.7.3. Receptor 2SB

Por otro lado, la configuración 2SB, si bien no elimina ninguna de las bandas RF con ningún filtro, el proceso de separación lo hace implementando dos *mixers* y dos híbridos de cuadratura (*Quadrature Hybrid*), componentes que se pueden apreciar en el diagrama de la Figura 2.9.c, encerrados en el recuadro azul. Cada híbrido, lo que hace, es dividir la señal entrante en otras dos señales de igual potencia y les aplica un desfase en 90º. Las dos señales IF que salen de esta configuración corresponden a las bandas laterales USB y LSB, obteniéndose por separado.

Estudios recientes han demostrado la efectividad de la configuración 2SB para la reducción del ruido atmosférico en radiotelescopios. Más adelante, en la sección de Estado del Arte, se abordarán en detalle implementaciones en espectrómetros digitales. Además, en el Anexo.A se incluye el desarrollo matemático de un circuito con esta configuración [13].

2.1.8. Sideband Rejection Ratio (SRR)

Para evaluar el desempeño de la separación de bandas, se usa como métrica la razón de rechazo de banda lateral (SRR), la cual se define como la relación de potencia en la salida entre una señal proveniente de la banda deseada y la banda rechazada. En términos de desequilibrios de ganancia y fase en un receptor de banda lateral separada, matemáticamente el SRR se calcula (en decibeles) con la Ecuación 2.2

$$\text{SRR (dB)} = -10\log_{10}\left(\frac{1-2\sqrt{G}\cos(\phi)+G}{1+2\sqrt{G}\cos(\phi)+G}\right), \tag{2.2}$$

G representa la relación de potencia entre la señal de la banda deseada y la señal de la banda lateral rechazada en un receptor y ϕ es el error de fase entre las dos vías de la banda lateral rechazada [14].

2.1.9. Field-Programmable Gate Array

Un FPGA, cuyas siglas en inglés corresponden a *Field-Programmable Gate Array*, es un tipo de circuito integrado que puede ser programado para realizar una variedad de funciones lógicas en paralelo. Está compuesto por una matriz con una gran cantidad de bloques

lógicos configurables y conexiones programables entre estos bloques, lo que le otorga una alta velocidad de procesamiento en comparación a otros dispositivos como un microcontrolador. Adicionalmente, estos presentan una alta flexibilidad para ser programados [15]. En la Figura 2.10 se tiene una tabla comparativa entre las FPGA, microcontroladores y circuitos ASIC.

	FPGA	Microcontroller	ASIC
Cost (low quantities)	Moderate	Cheap	Expensive
Cost (high quantities)	Moderate	Cheap	Cheap
Speed	Fast	Moderate	Fast+
Power	Moderate	Low	Low
Flexibility	High	Low	None
Ease of use	Medium	Easy	Difficult

Figura 2.10: Tabla comparativa entre FPGA, microcontroladores y circuitos ASIC [15].

Dentro de los ejemplos de FPGA enfocados en radioastronomía, están los sistemas ROACH y ROACH2, y también los de la familia RFSoC.

2.1.9.1. RFSoC 4x2

La RFSoC 4x2, utilizada en este proyecto, incorpora 4 convertidores analógico-digitales (ADC) con una tasa de muestreo de 5 Gsps[3] y 2 convertidores digital-analógicos (DAC) de 9.85 Gsps (característica que le otorga el apellido "4x2"), lo que permite manejar señales de radiofrecuencia (RF) de forma directa y eficiente.

Además, la RFSoC incluye bloques de procesamiento de señales digitales altamente configurables [16], lo que la hace ideal para aplicaciones en instrumentación, como el diseño de espectrómetros digitales. Su capacidad de combinar procesamiento analógico y digital en un solo *chip* proporciona un entorno de desarrollo compacto y eficiente, facilitando tanto la adquisición como el análisis de datos en proyectos como el espectrómetro del *SMWT*. En la Figura 2.11, se tiene un diagrama de bloques donde se resaltan sus principales componentes.

[3] La unidad Gsps (*Giga samples per second*) indica cuántas muestras de una señal analógica pueden ser adquiridas o generadas por segundo. Por ejemplo, 5 Gsps equivalen a 5 mil millones de muestras por segundo.

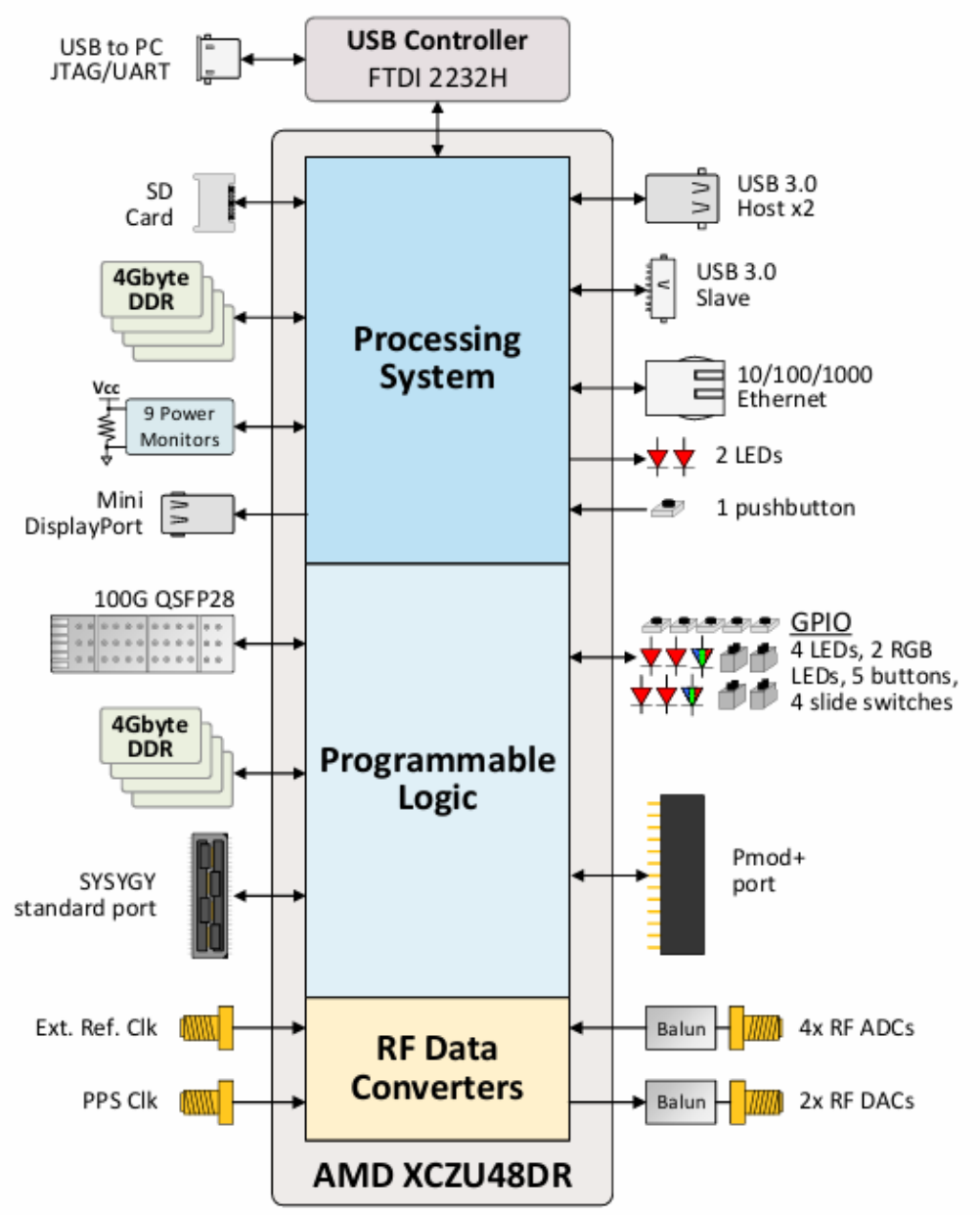


Figura 2.11: Diagrama de bloques con los principales componentes de la RFSoC [16].

2.1.9.2. CASPER

Si bien existen lenguajes de descripción de hardware como *Verilog* y *VHDL* para simular y diseñar circuitos digitales, para el desarrollo de este proyecto, se usaron principalmente las herramientas de desarrollo proporcionadas por *The Collaboration for Astronomy Signal Processing and Electronics Research* (CASPER[4], por sus siglas en inglés). CASPER es una colaboración de científicos e ingenieros que desarrollan herramientas y técnicas avanzadas para el procesamiento de señales en aplicaciones científicas, especialmente en astronomía.

[4] https://casper.berkeley.edu/

Esta colaboración ha desempeñado un papel clave en la creación de plataformas abiertas para el diseño y programación de sistemas de procesamiento de señales utilizando dispositivos FPGA, como los sistemas ROACH y ROACH2, los cuales se mencionarán en el Estado del Arte, y más recientemente, las plataformas RFSoC. El uso de su entorno de desarrollo y bloques prediseñados facilita la integración de componentes como ADC, DAC y unidades de procesamiento de señales digitales (DSP), simplificando significativamente el diseño de sistemas complejos, como espectrómetros y otros equipos de medición en radioastronomía, todo esto dentro de un marco de programación basado en Python, MATLAB y Simulink, lo que facilita la personalización e implementación de algoritmos específicos para procesar señales de radiofrecuencia en plataformas como la RFSoC.

Adicionalmente, CASPER tiene a disposición tutoriales básicos sobre el uso de sus librerías[5]. La primera etapa del proyecto consistió en realizar dichos tutoriales como primer acercamiento al entorno de desarrollo, como los bloques básicos necesarios para implementar el espectrómetro digital.

2.1.9.2.1. Tutorial 1: Bloques básicos e introducción a Simulink

En el primer tutorial[6], se busca un primer acercamiento del usuario al entorno de desarrollo. Uno de los objetivos es programar un LED de la RFSoC para que parpadee con cierta frecuencia utilizando los bloques GPIO, además de programar un contador controlable y un sumador controlable. Aquí se busca que el usuario compile un modelo funcional en la FPGA y pueda interactuar con esta. En la Figura 2.12 y en la Figura 2.13 se tiene una consola en Python, controlada desde una terminal, donde se interactúa con el contador y el sumador, respectivamente.

[5] https://casper-toolflow.readthedocs.io/projects/tutorials/en/latest/tutorials/rfsoc/readme.html
[6] https://casper-toolflow.readthedocs.io/projects/tutorials/en/latest/tutorials/rfsoc/tut_platform.html

```
In [9]: fpga.read uint('software register1')
Out[9]: 0

In [10]: fpga.write int('software register',1)

In [11]: fpga.read uint('software register1')
Out[11]: 341683852

In [12]: fpga.read uint('software register1')
Out[12]: 1513207215

In [13]: fpga.write int('software register',0)

In [14]: fpga.read uint('software register1')
Out[14]: 1622410895

In [15]: fpga.read uint('software register1')
Out[15]: 1622410895

In [16]: fpga.write int('software register',2)

In [17]: fpga.read uint('software register1')
Out[17]: 0
```

Figura 2.12: Contador controlable. En la imagen se usa la función FPGA.WRITE_INT() para darle instrucciones al contador. Si se le ingresa un 1, empieza a contar. Si se le ingresa un 0, se detiene el conteo. Finalmente, si se le ingresa un número distinto, como el 2, se reinicia el contador y vuelve a 0.

```
In [18]: fpga.write int('software register2',15)

In [19]: fpga.write int('software register3',35)

In [20]: fpga.read uint('software register4')
Out[20]: 50
```

Figura 2.13: Sumador controlable. En la imagen se usa la función FPGA.WRITE_INT() para ingresar valores numéricos a dos registros y posteriormente se usa la función FPGA.READ_UINT() para que entregue el resultado de la suma.

2.1.9.2.2. Tutorial 2: Bloque RFDC y *snapshots*

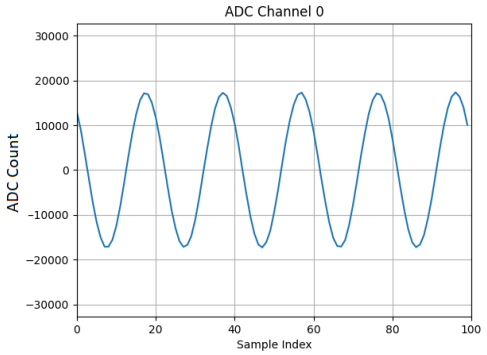
En este tutorial[7], se indica cómo programar el bloque RFDC de las librerías de CASPER en Simulink. El bloque RFDC (*RF Data Converter*) es un componente que permite configurar y controlar los ADC y DAC integrados en la plataforma RFSoC. En el tutorial se enseña a programar los ADC para que reciban una señal de entrada, y se utiliza el bloque de *snapshots* para poder leer directamente la señal capturada por los ADC.

Debido a que la configuración por defecto de la RFSoC 4x2 usa una tasa de muestreo de 3932.16 Msps y una decimación de un factor de 2, con estos datos, el ancho de banda se puede calcular con la frecuencia de Nyquist, como se muestra en la Ecuación 2.3.

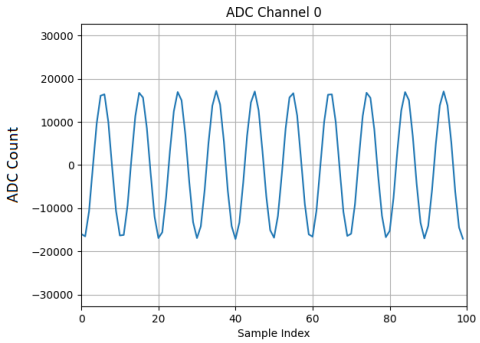
[7] https://casper-toolflow.readthedocs.io/projects/tutorials/en/latest/tutorials/rfsoc/tut_rfdc.html

$$BW = \frac{sampling\ rate}{2} = \frac{3932.16/2}{2} = 983.04 \text{ MHz} \tag{2.3}$$

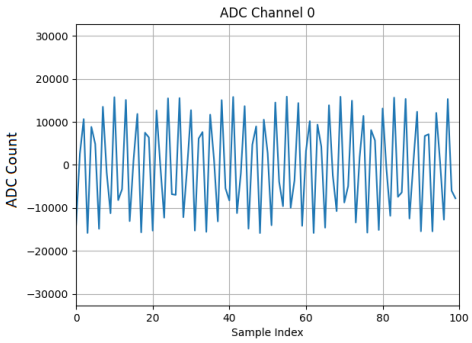
En la Figura 2.14 se muestra la señal leída por el bloque *Snapshot* para distintas frecuencias de entrada. Se puede notar que, a medida que aumenta la frecuencia, la sinusoide va perdiendo definición. Esto se debe a que, para representarla correctamente, es necesario aumentar la resolución utilizando más puntos de muestreo en el eje x del gráfico. Durante la realización de este tutorial no se aplicó dicho ajuste, pero la figura sirve para ilustrar que es posible utilizar el bloque *Snapshot* para visualizar una señal. Finalmente, para una frecuencia de entrada de 1200 MHz no se observa ninguna señal, ya que esta se encuentra fuera del límite impuesto por la frecuencia de Nyquist calculada previamente.



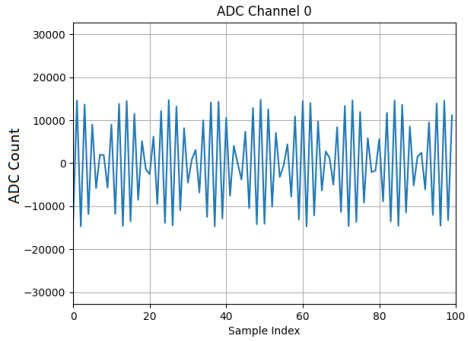
(a) 100 MHz



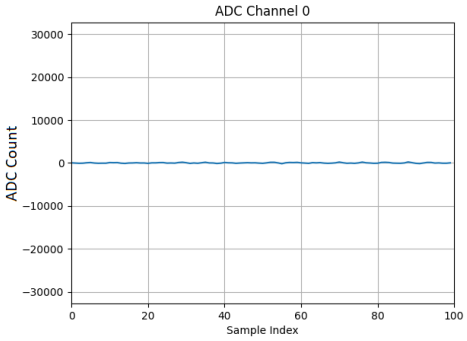
(b) 200 MHz



(c) 700 MHz



(d) 900 MHz



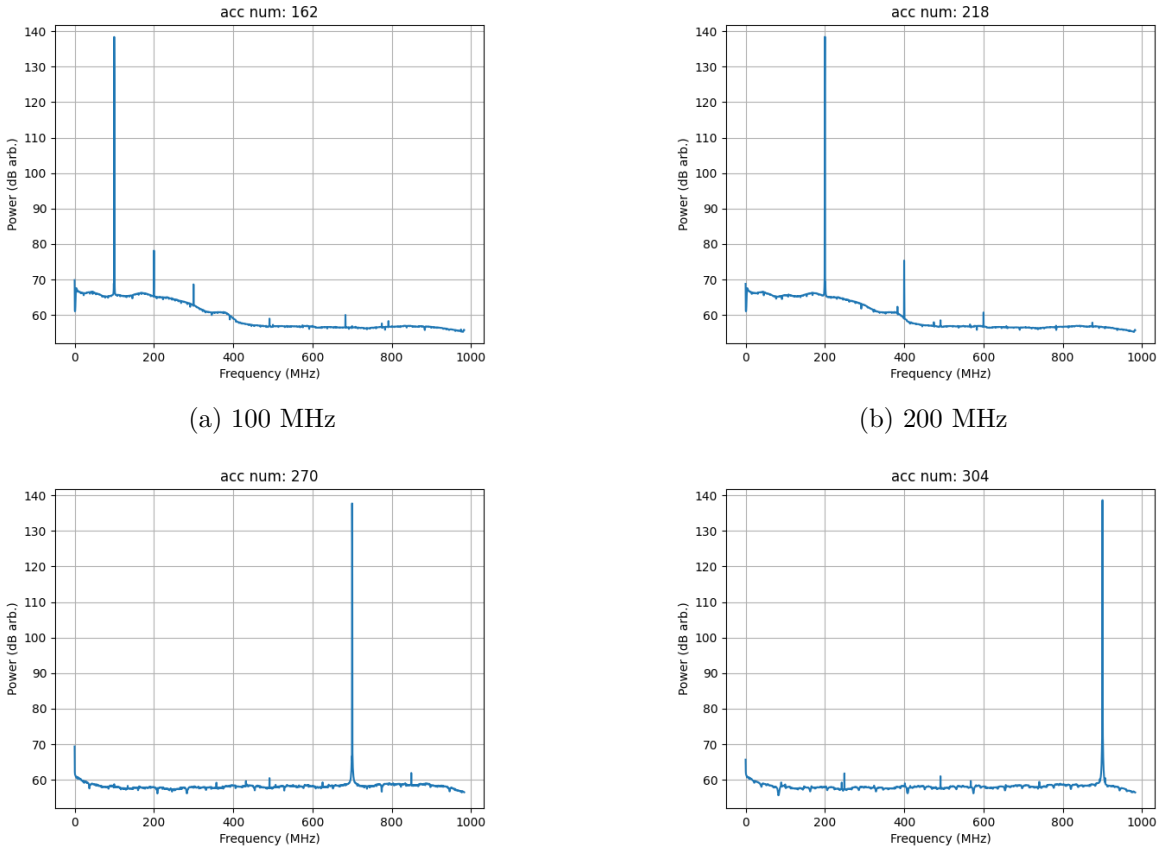
(e) 1200 MHz

Figura 2.14: Mediciones del bloque *Snapshot* en el Tutorial 2 para distintas frecuencias.

2.1.9.2.3. Tutorial 3: Espectrómetro simple

En el tercer tutorial[8], se presenta cómo construir un espectrómetro simple utilizando bloques de procesamiento de señales digitales de CASPER. La base del funcionamiento de este espectrómetro es el uso del algoritmo de la transformada rápida de Fourier (FFT), el cual permite convertir las señales entrantes, que se encuentran en el dominio del tiempo, al dominio de la frecuencia. Una vez realizada la FFT, se calcula la potencia espectral de cada componente de frecuencia, lo que consiste en obtener el cuadrado del módulo de la salida de la FFT. Estos valores de potencia se acumulan sumándose una cierta cantidad de veces, según un parámetro definido por el operador, y luego esta información es almacenada en bloques de lectura BRAM de 64 bits, desde donde se accede posteriormente para visualizar los tonos del espectro.

El ancho de banda con el que opera el espectrómetro es el mismo que se calculó en la Ecuación 2.3. En la Figura 2.15 se presentan capturas del espectrómetro midiendo señales a distintas frecuencias. La potencia de los tonos está expresada en decibeles.



(a) 100 MHz

(b) 200 MHz

(c) 700 MHz

(d) 900 MHz

Figura 2.15: Mediciones de los bloques BRAM en el Tutorial 3 para distintas frecuencias.

Este tutorial sienta las bases para el desarrollo del espectrómetro diseñado.

[8] https://casper-toolflow.readthedocs.io/projects/tutorials/en/latest/tutorials/rfsoc/tut_spec.html

2.2. Estado del Arte

La literatura revisada para abordar este proyecto, se centra en trabajos pasados sobre implementaciones de receptores y espectrómetros digitales para realizar la separación de banda lateral, utilizando tecnología basada en FPGA. A continuación, se presentan las principales publicaciones que se han revisado como referencia para el proyecto. Dichas implementaciones se basan en la configuración mostrada en la Figura 2.9.c.

2.2.1. Implementaciones de separación de bandas laterales

En [11], la tesis se centra en el diseño y construcción de un espectrómetro de separación de banda lateral digital calibrado para el uso en radioastronomía. Se aborda la necesidad de una alta separación de bandas laterales en receptores de radioastronomía y se propone un enfoque basado en tecnología digital para superar las limitaciones de la tecnología analógica actual. Aquí se presenta la arquitectura de receptores 2SB y se discuten las limitaciones de la tecnología analógica en la separación de bandas laterales. Se describe la implementación de un espectrómetro de Transformada Rápida de Fourier (FFT) basado en FPGA, que logra una mejora significativa en la relación de rechazo de banda lateral, superando los 40 dB en todo el ancho de banda de recepción.

En la Figura 2.16 se presenta un diagrama de bloques del sistema implementado, cuya parte encerrada con línea punteada es el híbrido digital que realiza la separación de bandas. El método consiste en digitalizar las salidas del mezclador para luego calcular la transformada de Fourier. Posteriormente, cada canal de frecuencia (de ambos *mixers*) se duplica y se multiplica por unas constantes complejas C_i, con $i = 1, ..., 4$, denominadas constantes de calibración, las cuales, si se eligen adecuadamente, pueden replicar un híbrido de cuadratura para la señal IF. En particular, se tiene que $C_1 = 1$ y $C_4 = 1$, por lo que solamente es necesario calcular C_2 y C_3 para calibrar. En la Figura 2.17 se puede apreciar el resultado de su desempeño de forma gráfica.

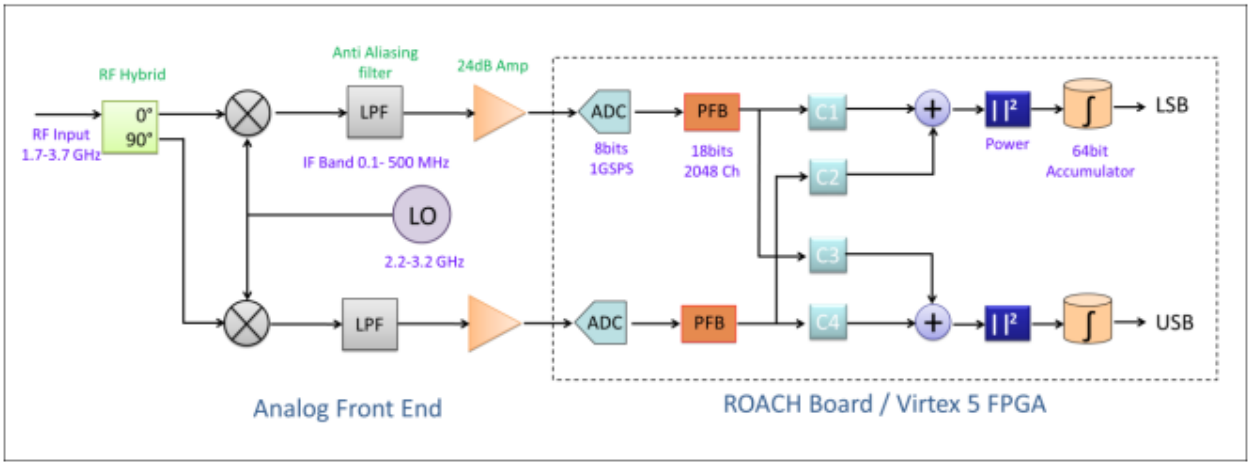


Figura 2.16: Diagrama de bloques del receptor digital separador de banda lateral [11].

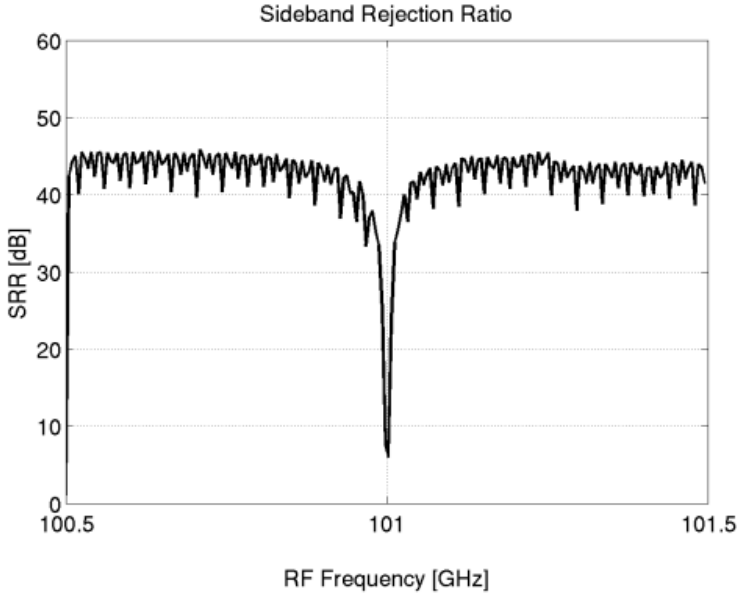


Figura 2.17: Resultados obtenidos de SRR para un rango de frecuencias de 100.5 a 101.5 GHz. El Oscilador Local está fijado en una frecuencia de 101.0 GHz [11].

En la implementación de la tesis, se utilizó un FPGA Xilinx Virtex 5. Este FPGA forma parte de la plataforma ROACH, utilizada para el procesamiento de señales en el espectrómetro de separación de banda lateral digital. El FPGA Xilinx Virtex 5 proporciona la capacidad de realizar algoritmos de procesamiento de datos en paralelo, lo que es fundamental para el funcionamiento eficiente del espectrómetro en aplicaciones de radioastronomía.

De esta implementación, se destaca la importancia de la tecnología digital en la mejora del rendimiento de los receptores de radio, demostrando su viabilidad y eficacia por sobre la analógica para aplicaciones en radioastronomía.

En [17] se presenta la implementación de un receptor de radioastronomía en la banda milimétrica que incorpora un espectrómetro con un híbrido digital IF calibrado. En esta publicación se utiliza un *backend* digital, basado en una placa ROACH, para calibrar desequilibrios de amplitud y fase en componentes individuales. Esta plataforma integra dos ADC con una tasa de muestreo de 3 Gsps y un FPGA para el procesamiento de datos. Además, se emplea un procesador PowerPC (*Performance Optimization With Enhanced RISC – Performance Computing*) para controlar el flujo de datos y gestionar la interfaz con sistemas informáticos a través de Ethernet. En la Figura 2.18, se muestra la configuración del sistema, la cual sigue el mismo procedimiento descrito en la publicación anterior, donde se extrae la FFT y, mediante el uso de constantes de calibración, se replica el híbrido digital que realiza la separación de bandas laterales. Aquí, C_1 y C_4 también tienen valor 1, y C_2 y C_3 se ajustan para calibrar el sistema.

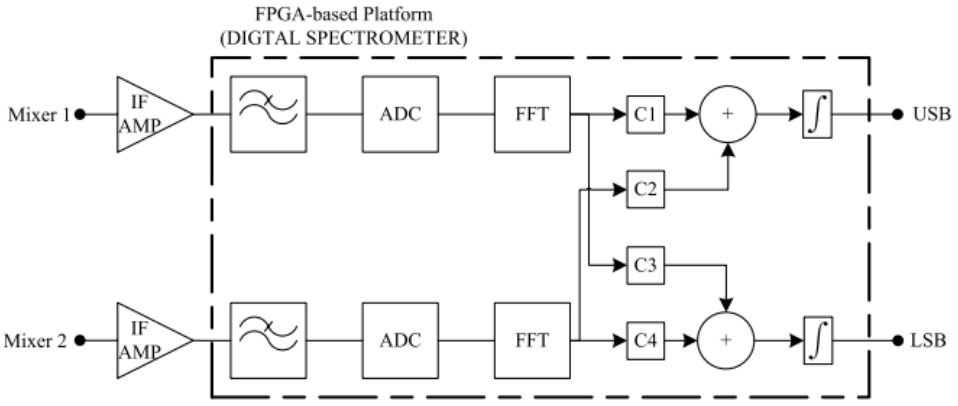
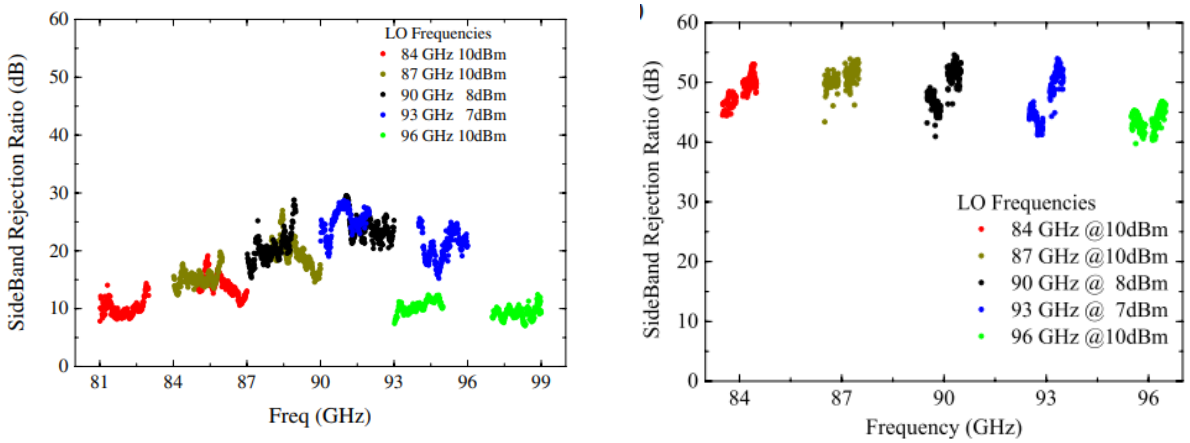


Figura 2.18: Diagrama de bloques del sistema programado en el FPGA [17].

Con dicha configuración, se lograron razones de rechazo de bandas laterales superiores a 40 dB en toda la banda de frecuencia de radiofrecuencia. En la Figura 2.19 se pueden ver los resultados del desempeño de la implementación para distintas frecuencias de LO. En el gráfico de la izquierda, se tiene el SRR usando un híbrido analógico en la IF, mientras que, en el gráfico de la derecha, se muestra los resultados de medición de SRR usando un híbrido digital calibrado en la IF.



(a) (b)

Figura 2.19: SRR del receptor para diferentes LO. (a) Híbrido analógico IF y (b) Híbrido digital IF calibrado [17].

La relevancia de este trabajo radica en la mejora de la calidad de las mediciones en radioastronomía mediante el uso de tecnología digital. Nuevamente, se destaca la importancia de los receptores que separan las bandas laterales en la radioastronomía y se aborda el desafío de mantener una alta razón de rechazo de bandas laterales (SRR).

Finalmente, en [14] se presentan los resultados de la aplicación de una técnica de calibración digital a un receptor astronómico de radiofrecuencia (RF) de banda ancha, que separa las bandas laterales para el *Atacama Large Millimeter/submillimeter Array* (ALMA). Siguiendo las definiciones de ALMA, dicha técnica se aplicó a las bandas 7 y 8, siendo la banda 7 de 275–373 GHz y la banda 8 de 385–500 GHz.

La técnica de calibración consiste en medir la relación de ganancia y la diferencia de fase entre las salidas del receptor, calcular constantes de corrección y aplicarlas a las señales digitalizadas para mejorar el rendimiento del receptor.

Para la implementación del sistema de calibración digital se utilizó la plataforma ROACH2, que consiste en dos ADC EV8AQ160 y un FPGA Virtex-6. Esta configuración permitió implementar dos espectrómetros de 2048 *bins* con un ancho de banda de 1.08 GHz cada uno. La plataforma ROACH2 es una segunda versión de la ROACH mencionada en las publicaciones anteriores, siendo una plataforma de hardware flexible y potente que facilita la implementación de sistemas digitales complejos para aplicaciones como la calibración de receptores astronómicos de radiofrecuencia de banda ancha. En la Figura 2.20, se tiene el diagrama de bloques del híbrido digital con el sistema de calibración. Si bien este diagrama solo presenta dos constantes de calibración, es equivalente a los mostrados anteriormente, ya que omite las dos constantes de calibración que tenían valor 1.

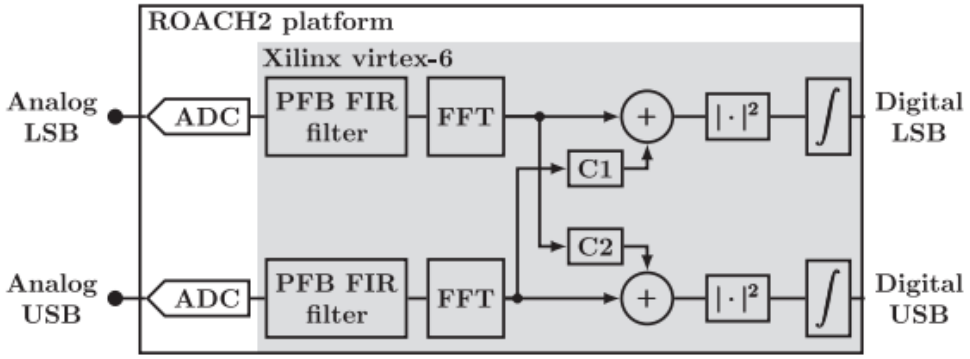


Figura 2.20: Diagrama de bloques del sistema de calibración implementado en el FPGA [14].

La calibración logró aumentar la razón de rechazo de las bandas laterales en aproximadamente 30 dB en promedio, mejorando significativamente la supresión de las bandas laterales y reduciendo el ruido atmosférico. En la Figura 2.21, se tienen los resultados de mediciones del SRR del receptor.

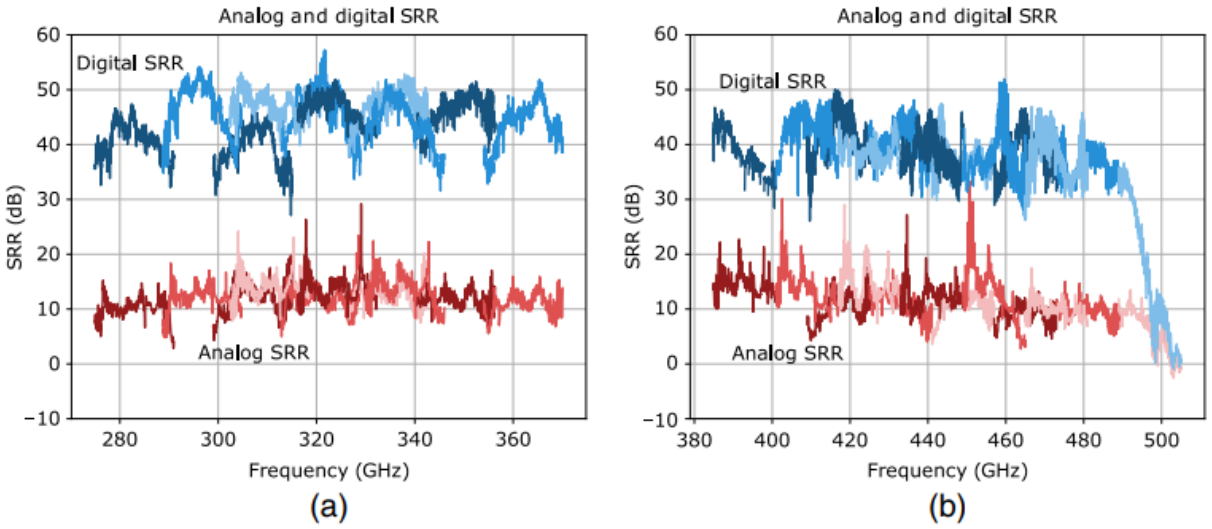


Figura 2.21: Resultados de SRR del receptor. (a) Para banda 7 y (b) para banda 8. Curva roja: sin calibrar (análogico). Curva azul: calibrado (digital) [14].

Esta publicación destaca por su contribución significativa a la mejora del rendimiento de receptores astronómicos de radiofrecuencia de banda ancha, mediante la aplicación de técnicas de calibración digital usando FPGA, lo que lo posiciona como un importante punto de referencia en el estado del arte de la separación de bandas laterales en receptores astronómicos.

De manera similar a los trabajos previamente mostrados, se vuelve a mostrar que la tecnología digital tiene un gran potencial para mejorar la calidad de las mediciones en radioastronomía al ofrecer una forma eficaz de calibrar y compensar desequilibrios en sistemas de recepción de alta frecuencia.

Capítulo 3

Diseño e Implementación del Espectrómetro

En este capítulo se introduce el contexto y se explica el desarrollo del diseño e implementación del espectrómetro digital desarrollado para el radiotelescopio *Southern Millimeter Wave Telescope* (*SMWT*), o simplemente Mini. El espectrómetro se programó como un híbrido digital en la banda IF para realizar separación de banda lateral. El desarrollo abarcó desde el diseño inicial del espectrómetro hasta la implementación final en el radiotelescopio, incluyendo las modificaciones necesarias para adaptarse a sus limitaciones y mediciones de prueba.

3.1. Actualización del Back End del Mini

En el presente trabajo de título se trabajará con el Mini. En particular, actualizando su espectrómetro, para que tenga una mayor capacidad observacional. Dentro de los requisitos mínimos del espectrómetro a diseñar en el proyecto se encuentran:

- Aumentar el ancho de banda de cada banda lateral en la frecuencia IF a un valor cercano a los 2 GHz.
- Aumentar la cantidad de canales espectrales a un valor de al menos 8192, esto con el objetivo de tener un espectrómetro de alta resolución.
- Así como se aumentará la cantidad de canales espectrales, se espera poder capturar los 8192 canales con los que funcionará el nuevo espectrómetro y almacenarlos en un nuevo computador denominado *Data Server*.
- Los espectros que reciba el PIC están limitados por un número de 32 bits como máximo. Además que este componente del radiotelescopio solo puede recibir 512 canales espectrales a la vez.
- Debe estar sincronizado con la señal del *RESET* del RFG del sistema de control.

En la Figura 3.1, se tiene un diagrama general con la nueva estructura con la que se propone implementar el espectrómetro.

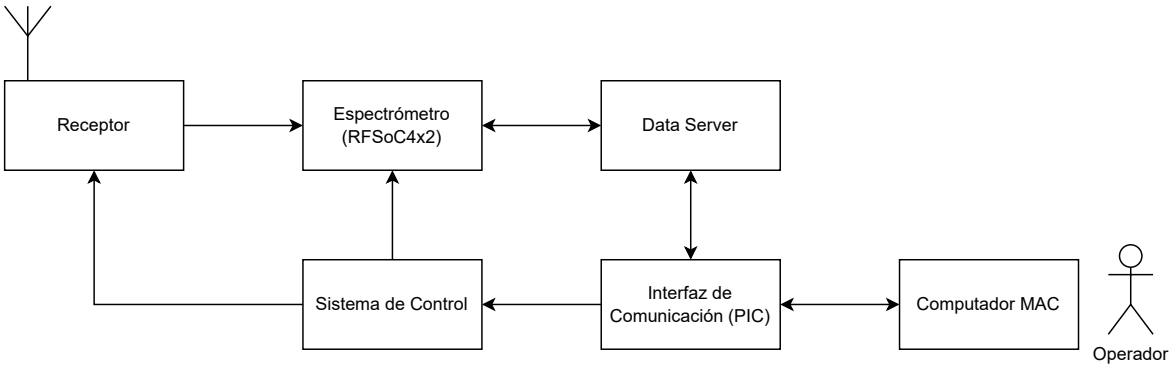


Figura 3.1: Diagrama general actualizado del radiotelescopio Mini

En esta nueva configuración, el sistema mantiene una arquitectura similar a la anterior, con la diferencia de que el espectrómetro basado en la placa ROACH ha sido reemplazado por una RFSoC 4x2. Además, se incorpora un *Data Server* con el objetivo de capturar una mayor cantidad de canales espectrales y facilitar su distribución a los distintos subsistemas.

El resto del sistema permanece sin cambios: el PIC continúa actuando como interfaz de comunicación entre el computador Mac del operador, el sistema de control del telescopio y, ahora, también el *Data Server*. Cabe destacar que la PIC está limitada a leer únicamente 512 canales espectrales de 32 bits. En esta configuración, el RFSoC genera el espectro completo y lo envía al *Data Server*, el cual se encarga de seleccionar los 512 canales requeridos y entregarlos a la PIC para su posterior análisis.

3.2. Diseño del Espectrómetro

3.2.1. Funcionamiento como Híbrido Digital

El funcionamiento del espectrómetro se basó completamente en la configuración mostrada de los receptores 2SB en el Marco Teórico, como el de la Figura 2.9.c, donde la separación de bandas se lleva cabo con el uso de dos híbridos de cuadratura. En particular, el espectrómetro implementado integra un híbrido digital como los presentados en el Estado del Arte para la banda IF, el cual, siguiendo el diagrama de la Figura 3.2, le aplica un desfase de 90º con las constantes C1 y C2 a las entradas I y Q y las suma resultando en las dos bandas laterales USB y LSB, por separado.

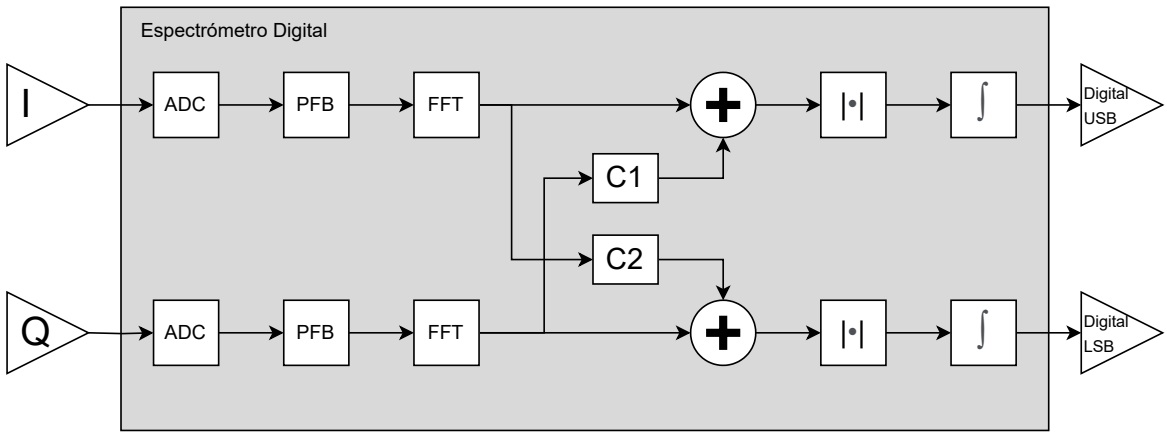


Figura 3.2: Diagrama del funcionamiento interno de la RFSoC.

A continuación, se explica la función de cada etapa:

1. **ADC *(Analog to Digital Converter)*:** es el conversor de una señal analógica a digital. Convierte una señal continua a discreta (en tiempo y amplitud) según una cierta tasa de muestreo. Como es información digital, esta se interpreta en números binarios o bits.

2. **PFB *(Polyphase Filter Bank)*:** es una técnica de procesamiento digital que mejora la precisión espectral en espectrómetros basados en FFT. Su principal ventaja es la reducción de efectos no deseados como la fuga espectral (spectral leakage) y la pérdida por ondulación (scalloping loss), comunes en una FFT directa. El PFB utiliza un conjunto de filtros aplicados antes de la FFT, los cuales suavizan los bordes de cada bloque de datos, mejorando así la separación entre canales de frecuencia. Esta arquitectura permite dividir una señal de banda ancha en múltiples sub-bandas más pequeñas con una respuesta más limpia y uniforme en frecuencia [18].

3. **FFT:** se usa para convertir la señal de dominio temporal en dominio frecuencial, permitiendo analizar su contenido espectral.

4. **Híbrido de Cuadratura:** es una configuración que toma dos señales entrantes, les aplica un desfase de 90º multiplicando por unas constantes complejas C_1 y C_2, y, finalmente, las suma. Juntando dos híbridos de cuadratura, como se mencionó en el Marco Teórico, es posible realizar separación de bandas laterales.

5. **Potencia:** se le extrae el valor en potencia de la señal calculando el módulo al cuadrado.

6. **Acumulación:** El valor en potencia de cada canal se integra, es decir, se suman los resultados de la potencia espectral en cada bin durante múltiples ciclos de FFT. Es decir, para cada bin de frecuencia, se acumulan los valores de potencia de diferentes instantes de tiempo.

7. **Lectura de datos:** Se obtienen los datos del espectro leyendo bloques de la FPGA.

Para efectos de este Trabajo de Título, el espectrómetro a implementar no incluirá un proceso de calibración, por lo que las constantes de calibración C_1 y C_2 se asumirán como

valores ideales. Específicamente, se utilizará el valor complejo $0 + 1j$, el cual representa una rotación exacta de 90º en el plano complejo. En sistemas calibrados, estas constantes se ajustan para corregir desbalances de amplitud y fase introducidos por componentes analógicos del sistema, como los mezcladores, amplificadores y el híbrido de cuadratura en el *Front End*. Dichos desbalances hacen que la separación entre las bandas laterales no sea perfecta, por lo que se requiere aplicar factores de corrección que no corresponden a una rotación de fase exacta. Sin embargo, para propósitos de implementación funcional, en este trabajo se considera una aproximación ideal sin corrección, utilizando directamente el valor $0 + 1j$.

3.2.1.1. Control de acumulaciones

Para el diseño del espectrómetro, uno de los parámetros clave a considerar es la cantidad de acumulaciones realizadas sobre el espectro calculado. En radioastronomía, las señales observadas suelen tener potencias muy bajas, inmersas en ruido térmico que domina la mayor parte del espectro. En un espectrómetro digital, este ruido se manifiesta como fluctuaciones aleatorias en la potencia espectral medida en cada canal de frecuencia.

La acumulación tiene como objetivo reducir esas fluctuaciones mediante integración temporal. En lugar de considerar una única Transformada Rápida de Fourier (FFT), cuyos resultados pueden variar significativamente de un instante a otro debido al carácter aleatorio del ruido, se suman múltiples espectros de potencia obtenidos en ciclos sucesivos. Dado que el ruido térmico tiene una distribución aleatoria, la acumulación suaviza sus variaciones, permitiendo obtener un valor promedio más estable de la potencia en cada canal. Este proceso no elimina el ruido, pero sí reduce su varianza, de acuerdo con la relación expresada en la ecuación del radiómetro [19], donde el error relativo disminuye con la raíz del tiempo de integración, como se puede notar en la Ecuación 3.1.

$$\sigma_T \approx \frac{T_s}{\sqrt{\Delta\nu \cdot \tau}}, \tag{3.1}$$

En esta expresión, σ_T representa la incertidumbre en la medición de temperatura (o potencia), T_s es el ruido total del sistema, $\Delta\nu$ es el ancho de banda efectivo del sistema, y τ corresponde al tiempo de integración, el cual es directamente proporcional al número de acumulaciones realizadas. Esta relación muestra que, al aumentar el número de acumulaciones, se mejora la estabilidad de la medición y, por ende, la capacidad de detectar señales débiles sobre el fondo de ruido.

Una mayor cantidad de acumulaciones mejora la relación señal/ruido (SNR), ya que permite distinguir con mayor claridad señales débiles superpuestas al fondo de ruido. No obstante, acumular durante intervalos demasiado largos puede incrementar la latencia del sistema, ya que se debe esperar a que se completen todas las acumulaciones antes de poder leer un nuevo espectro.

Considerando esto, el espectro entregado por el espectrómetro dependerá directamente del número de acumulaciones definidas. En esta implementación, el híbrido digital debe generar un espectro cada vez que se alcance el máximo de acumulaciones configurado. El funcionamiento del sistema de control de acumulaciones del espectrómetro se resume en el diagrama de flujo mostrado en la Figura 3.3.

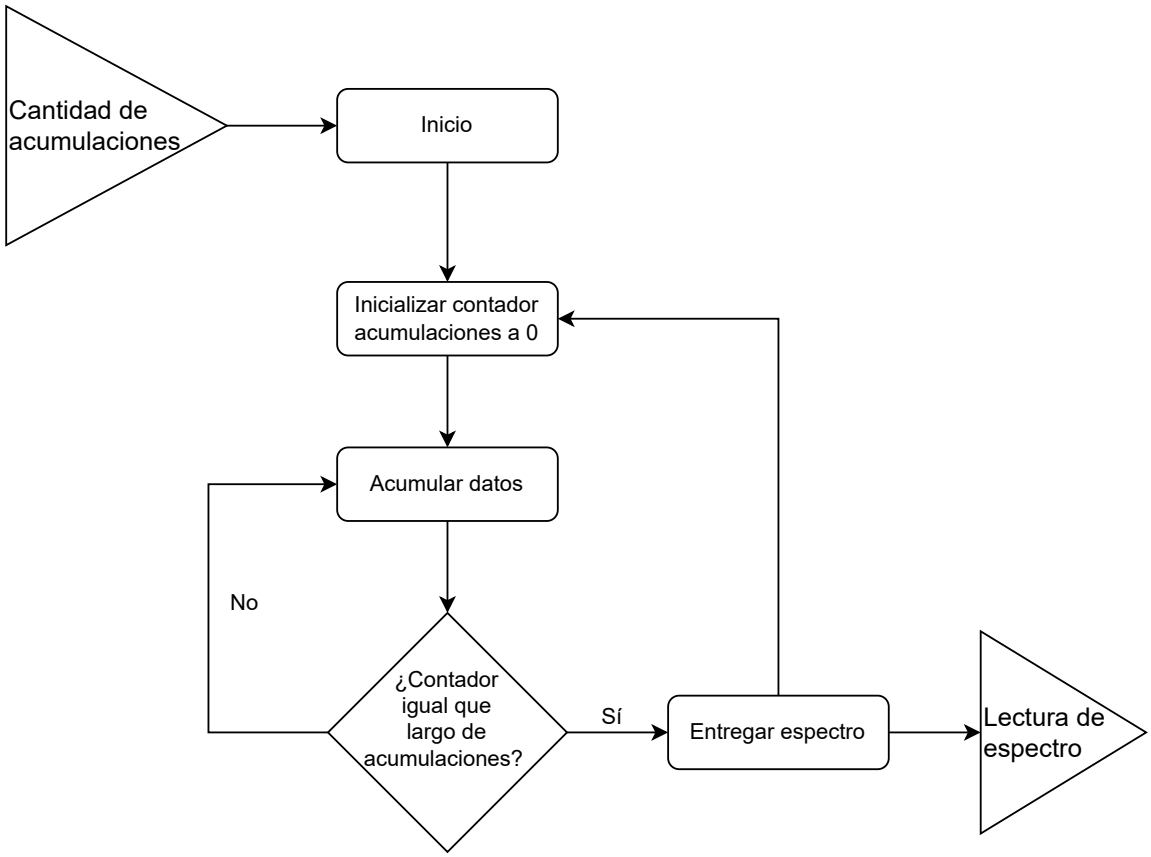


Figura 3.3: Diagrama de Flujo del Sistema de Control de Acumulaciones.

3.2.2. Modelos desarrollados

3.2.2.1. Primeras iteraciones: Modelos de 2048, 8192 y 16384 canales espectrales, y ancho de banda IF de 1.966 GHz

Para iniciar con el desarrollo del proyecto, se partió tomando como base los modelos del espectrómetro simple vistos en el tutorial 3 de CASPER. Este espectrómetro, como se mencionó previamente, utiliza la FFT para extraer la componente en frecuencia de la señal recibida.

Para ajustar el ancho de banda se utiliza el mismo bloque RFDC que se usa para programar los ADC que recibirán la señal IF provenientes del *Front End*. Si bien en el tutorial se vio una configuración para que el ancho de banda sea de 983.04 MHz, como se calculó en la Ecuación 2.3, es posible duplicar este ancho de banda cambiando el tipo de salida digital de los ADC. Cuando se tiene un tipo de muestreo digital "Real", el ancho de banda se encuentra entre 0 y $f_s/2$, la frecuencia de Nyquist, ya que toda la información de la señal está contenida en un solo lado del espectro, en este caso eliminando las frecuencias negativas. Por otro lado, cuando se usa el tipo de salida "I/Q", la señal se divide en dos componentes ortogonales eliminando ambigüedad entre frecuencias positivas y negativas, aumentando el rango de análisis entre $-f_s/2$ y $f_s/2$, es decir, un ancho de banda igual a la tasa de muestreo. Como la tasa de muestreo es de 3932.16 y se tiene una decimación de 2, el nuevo ancho de

banda es de 1966.08 MHz. En la Figura 3.4 se tiene la configuración que hay que seguir para cambiar el ancho de banda[9].

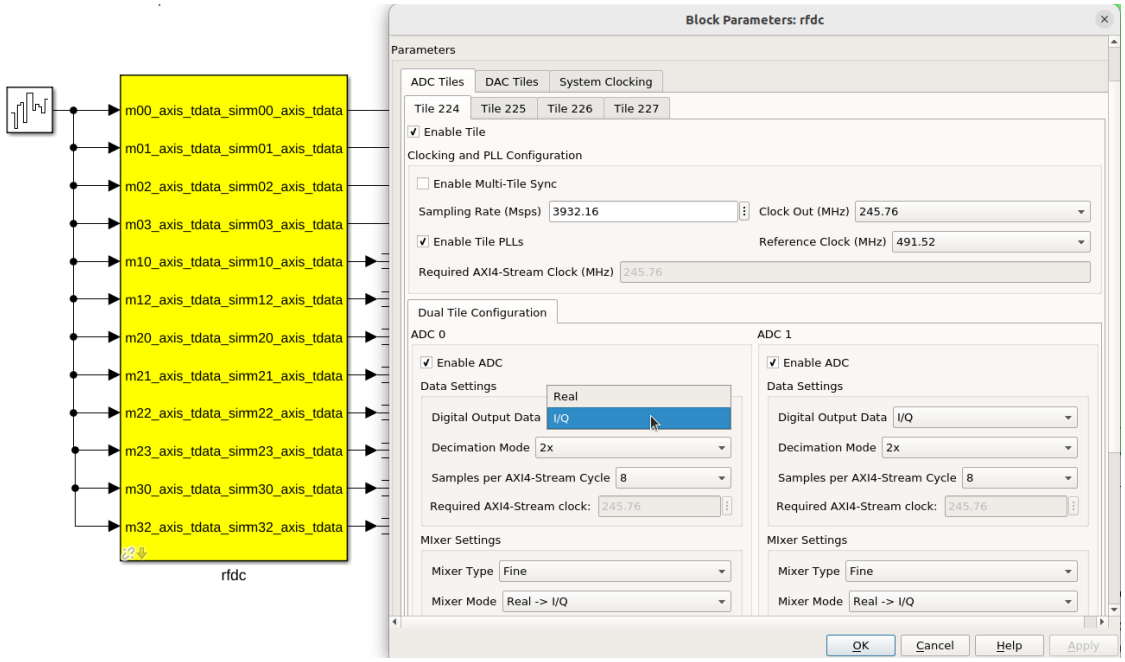


Figura 3.4: Configuración del bloque RFDC para aumentar el ancho de banda

Del bloque RFDC, salen en total 16 muestras de 16 bits (específicamente UFix16_0) generando un número de 256 bits sin signo UFix256_0. Luego, este número es dividido en un bus de 8 salidas paralelas que lo segmenta en 8 números binarios de 32 bits sin signo (UFix32_0). En cuanto a la asignación de canales espectrales, esto está determinado por el tamaño de la FFT que se configura en el modelo de Simulink en los bloques PFB, FFT y sync_gen. En la Figura 3.5 se tiene la distribución de los bloques implementados.

[9] Esta modificación también se puede aplicar al tutorial 2 de CASPER

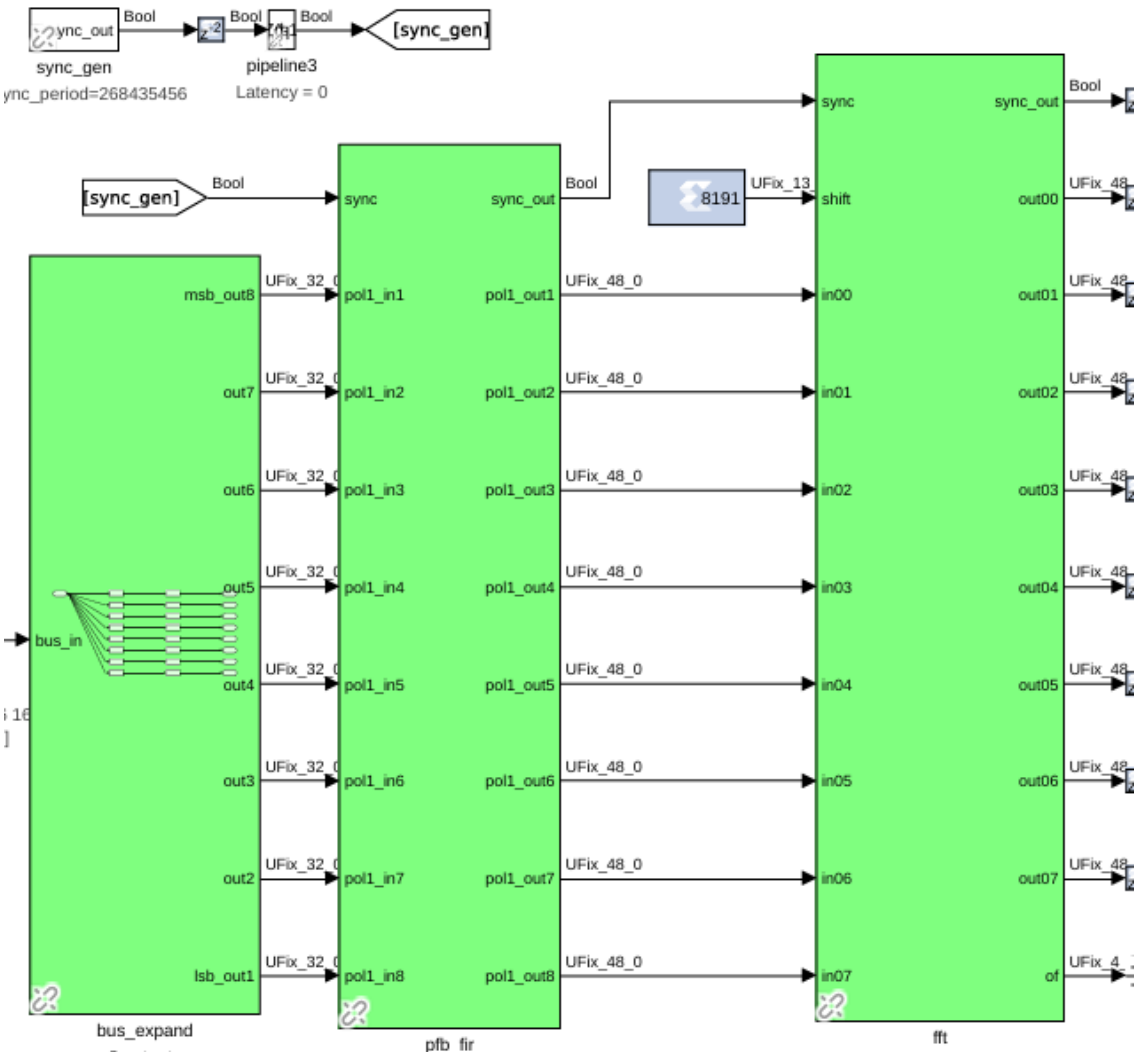


Figura 3.5: Bloques Bus, PFB, FFT y sync_gen del espectrómetro en Simulink.

El bloque sync_gen[10] genera una señal periódica de sincronización que permite el flujo constante de datos. Básicamente, es lo que permite que los bloques PFB y FFT funcionen continuamente.

Ya habiendo definido los parámetros de la FFT para el análisis en frecuencia del espéctro, se debe implementar el proceso de separación de banda lateral. Basándose en el diagrama de bloques de la Figura 3.2, falta definir las constantes de calibración. En esta implementación se usaron constantes de calibración ideales, por lo que C1 y C2 tienen valores $0 + 1j$. La multiplicación por estas constantes se llevó a cabo implementando un bloque multiplicador, el cual se muestra en la Figura 3.6.

[10] https://casper.berkeley.edu/wiki/Sync_gen

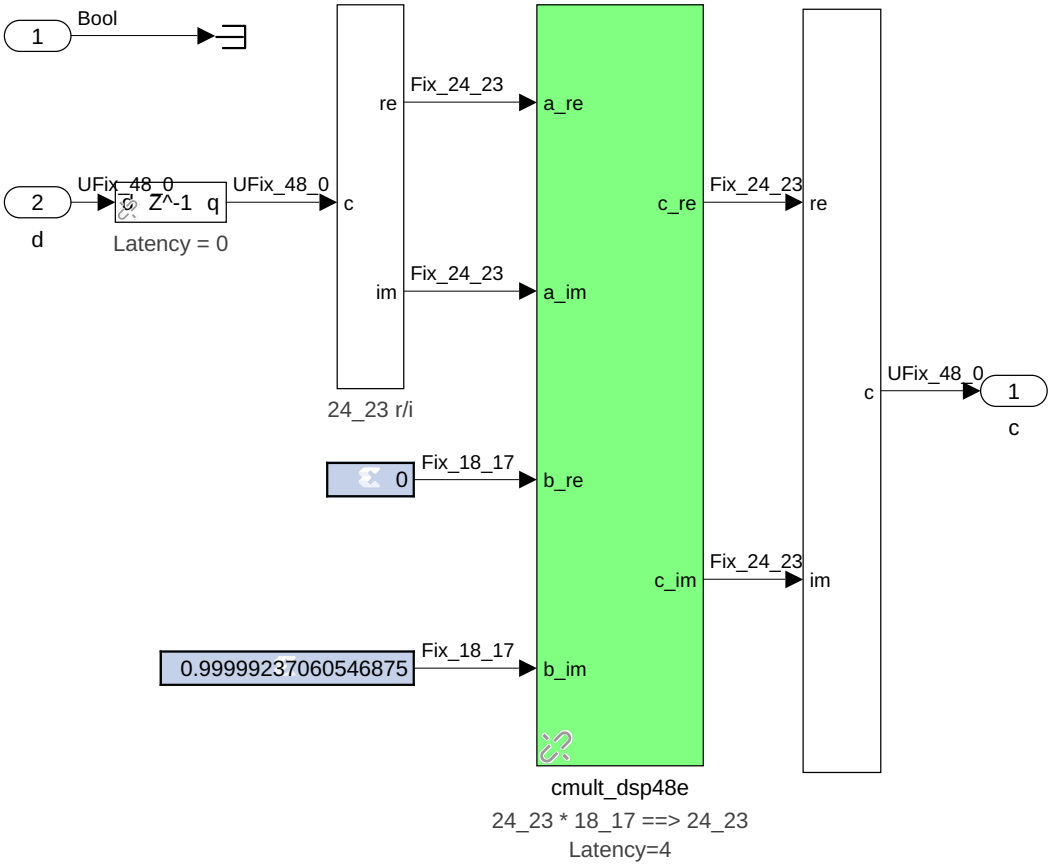


Figura 3.6: Bloque bram_mult implementado para la multiplicación de constantes de calibración.

Cada bloque recibe de entrada una de las salidas del bloque de la transformada rápida de Fourier FFT. Luego de la multiplicación, los números son sumados como se ve en la configuración de la Figura 3.2, donde posteriormente se les extrae el valor en potencia que llega a unos bloques BRAM para la posterior lectura del espectro. El espectrómetro depende de un parámetro denominado "largo de vector de acumulaciones", acc_len , el cual se le entrega a la FPGA al programarla con el modelo. En este caso, se refiere a los espéctros que llegan a los bloques SIMPLE_BRAM_VACC tras la separación de banda lateral. El bloque SIMPLE_BRAM_VACC depende de un sistema de control de las acumulaciones. Este sistema, se encuentra implementado en un bloque ACC_CNTRL, el cual se puede visualizar en la Figura 3.7.

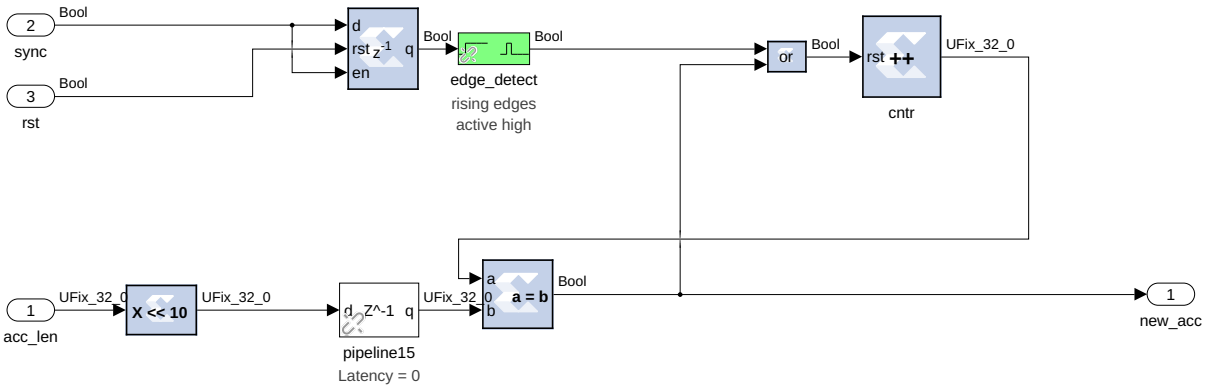


Figura 3.7: Bloque ACC_CNTRL para el control de acumulaciones

Finalmente, tras alcanzarse la cantidad de acumulaciones definida, los espectros pueden ser leídos consultando bloques BRAM de lectura. En particular para este modelo se usaron bloques BRAM de 64 bits y se les denominaron synth0_i y synth1_i, para las bandas USB y LSB, donde i=0,...,15. Esta configuración se usó para desarrollar modelos de tamaño de FFT de 2048, 8192 y 16384[11].

3.2.2.1.1. Trabajo adicional: Correlador

Como se mencionó previamente, el espectrómetro implementado no está calibrado, ya que se usaron constantes de calibración ideales, teniendo valores $0+1j$. Sin embargo, si se quisiera calibrar es necesario obtener la diferencia de fase de las señales que recibe el espectrómetro después del primer híbrido de cuadratura. Para obtener la diferencia de fase es necesario implementar un correlador en el espectrómetro.

El correlador funciona mediante el cálculo de la correlación entre ambas señales, lo cual se logra mediante la multiplicación de una señal con el conjugado complejo de la otra. Matemáticamente, si las señales de entrada $x(t)$ y $y(t)$ se representan como fasores, tenemos:

$$x(t) = Ae^{j(\omega t+\phi_1)}$$

y

$$y(t) = Ae^{j(\omega t+\phi_2)}$$

donde A es la amplitud, ω es la frecuencia angular, ϕ_1 y ϕ_2 son las fases iniciales de las señales $x(t)$ y $y(t)$, respectivamente.

La correlación entre estas dos señales se obtiene multiplicando $x(t)$ por el conjugado complejo de $y(t)$, lo que resulta en:

[11] Se diseñó un modelo adicional con una FFT más grande de 32768, pero no fue posible compilarlo debido a limitaciones de *hardware*.

$$R_{xy}(t) = x(t) \cdot y^*(t) = Ae^{j(\omega t+\phi_1)} \cdot Ae^{-j(\omega t+\phi_2)} = A^2 e^{j(\phi_1-\phi_2)}$$

Este valor de la correlación es un número complejo cuya fase $(\phi_1 - \phi_2)$ representa la diferencia de fase entre las señales de entrada.

En el modelo implementado, las salidas de los bloques de la FFT se multiplican utilizando el bloque CMULT de CASPER. Este bloque se puede configurar para realizar multiplicación conjugada, lo que permite obtener la correlación entre ambas señales. Posteriormente, el número resultante se separa en su parte real e imaginaria. En la Figura 3.8 se muestran los bloques utilizados para realizar la correlación.

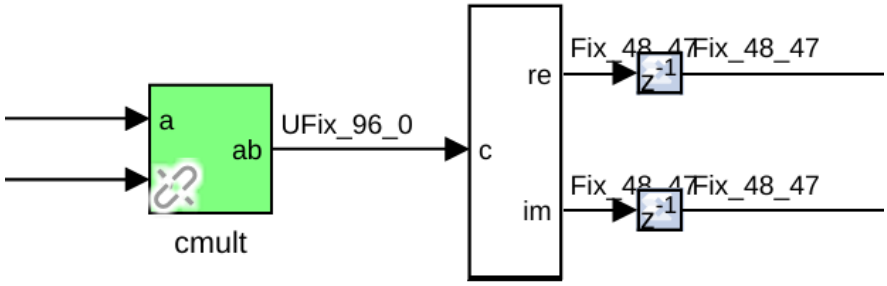


Figura 3.8: Correlación entre las señales de entrada.

Finalmente, de manera similar a la lectura de los espectros, se utilizaron bloques BRAM de 64 bits para leer las partes real e imaginaria, denominándolos ab_re_i y ab_im_i, respectivamente, donde $i = 0, \ldots, 8$.

Si bien esta parte del modelo es opcional para los objetivos del proyecto, se espera que se utilice en trabajos futuros para una implementación del espectrómetro calibrado.

3.2.2.2. Primera Re-Adaptación: Señal del *RESET*

Previamente se comentó que el *SMWT*, presenta un sistema que controla varios estados del radiotelescopio. Uno de ellos es la Señal del *RESET*, proveniente del generador de referencia RFG, que funciona según el modo de observación en el que opere el telescopio, como se puede apreciar en la Figura 2.7.

Esta señal hace que en cada canto de bajada, se reinicien las acumulaciones, para que el espectrómetro mida un nuevo espectro acumulado en cada período. Para que la RFSoC pueda recibir señales de entrada es posible programar los pines PMOD que posee. En la Figura 3.9 se muestra cómo configurar un puerto PMOD en Simulink con el entorno de CASPER. Específicamente, se muestra como hacer que el RFSoC detecte un canto de bajada.

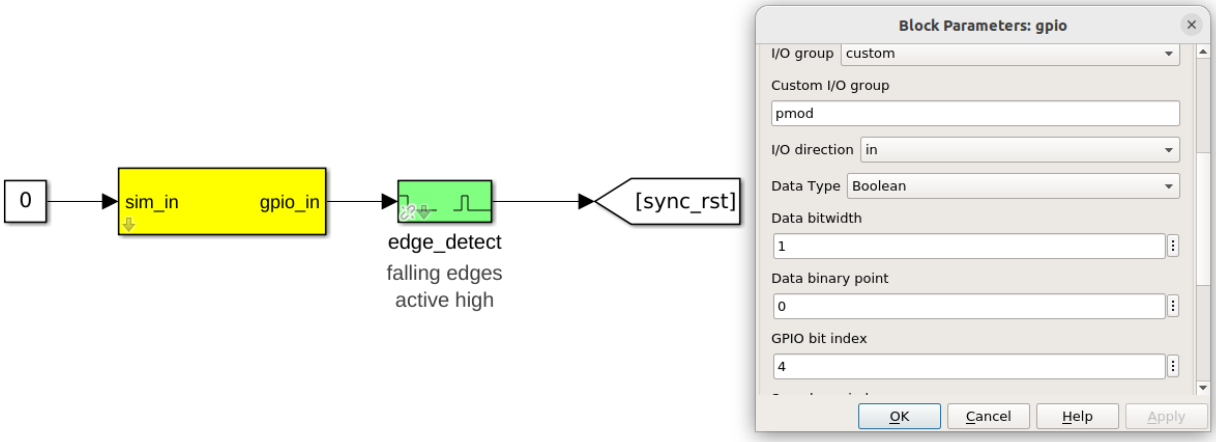
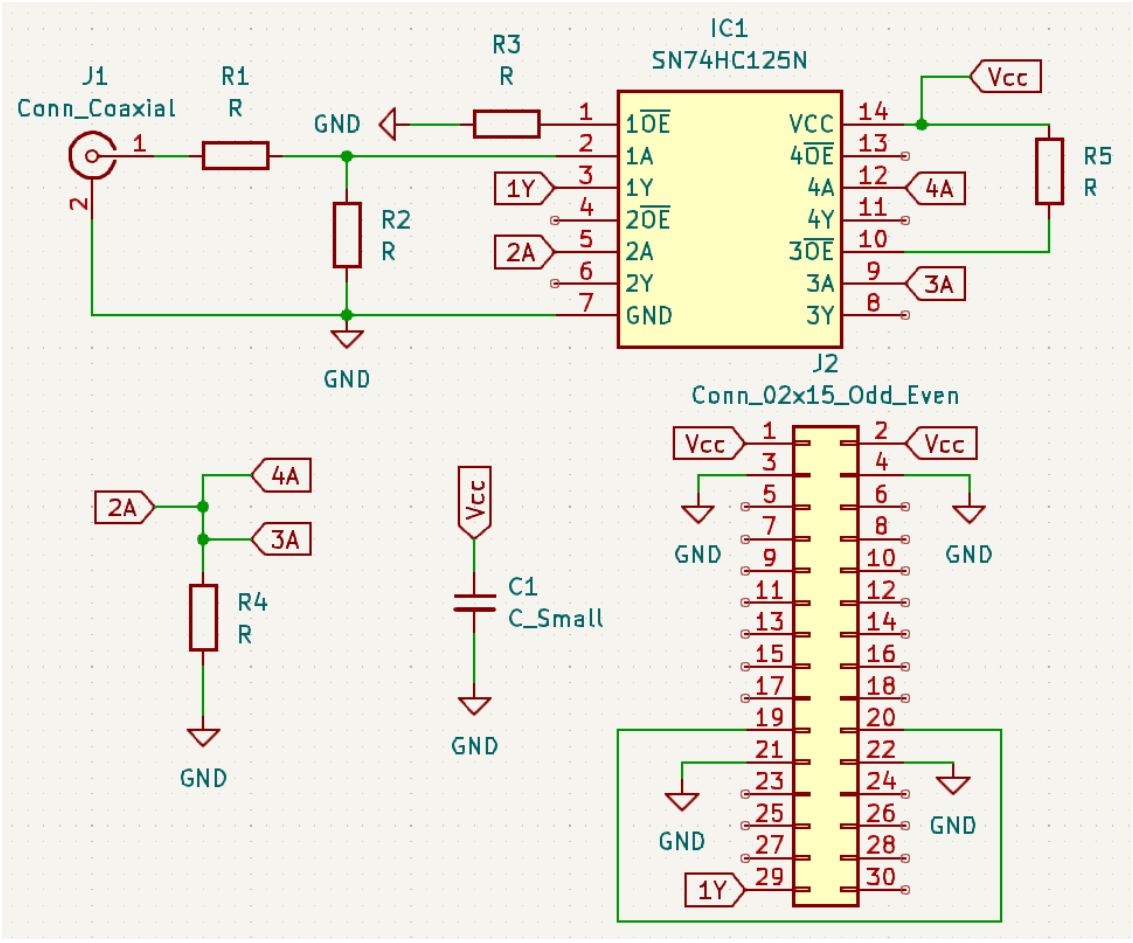


Figura 3.9: Configuración para programar un puerto PMOD para detectar un canto de bajada. En específico, se programó el PMOD con índice 4 en *GPIO bit index*.

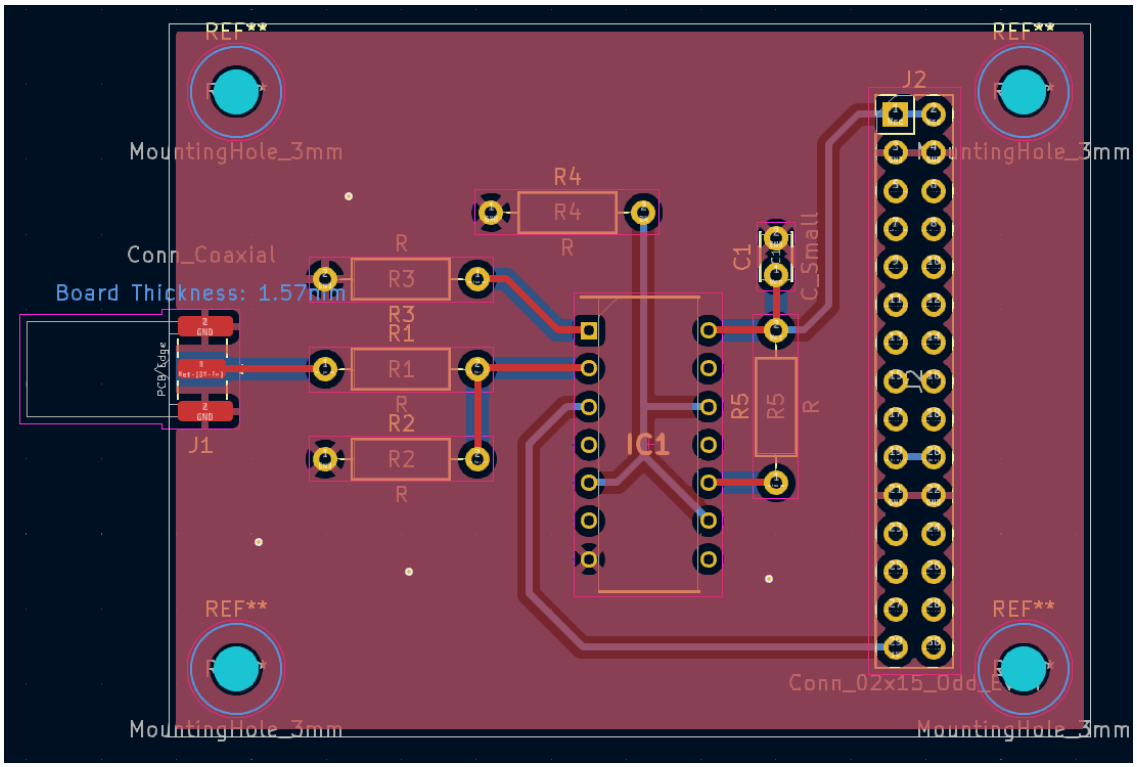
La señal de *RESET* es una señal cuadrada de 5V, mientras que el voltaje máximo que puede recibir los pines PMOD es de 3.3V. Para evitar dañar la RFSoC es necesario diseñar un circuito adecuado según estos requisitos. Para resolver este inconveniente, el circuito a implementar consta de un divisor de voltaje que reduce el voltaje de entrada a en un factor de 2/3, es decir reduce 5 V a 3.3 V. Adicionalmente, se agregó un buffer de 3 estados SN74HC125N[12]. Siguiendo el esquemático de la Figura 3.10.a, este componente lo que hace es replicar la señal de entrada 1A a la salida 1Y, impidiendo un posible consumo de corriente que pueda dañar el FPGA. En la Figura 3.10.b se tiene la placa diseñada en KiCad.

En la Figura 3.11.a y la Figura 3.11.b se muestra el diseño en 3D de la placa vista desde la parte frontal y trasera, respectivamente.

[12] https://www.mouser.cl/ProductDetail/Texas-Instruments/SN74HC125N?qs=gqbMQSs93zPnqY7zyK8v9w%3D%3D

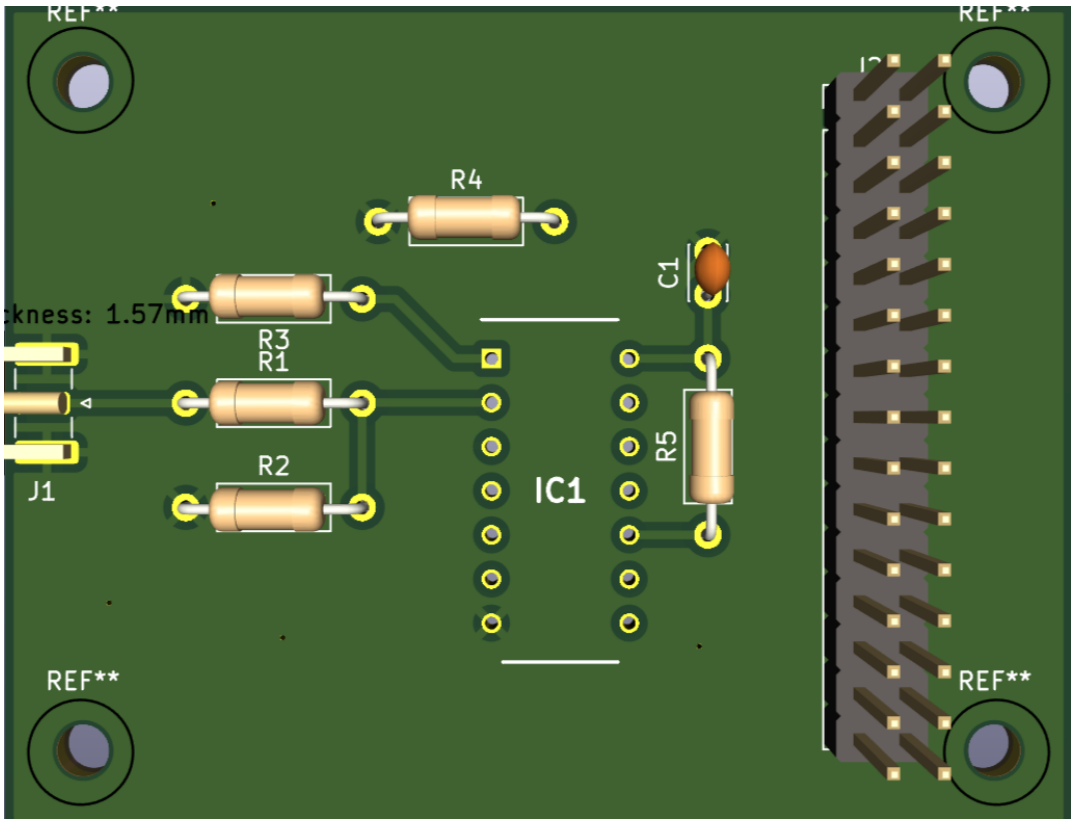


(a) Esquemático en KiCad de la placa

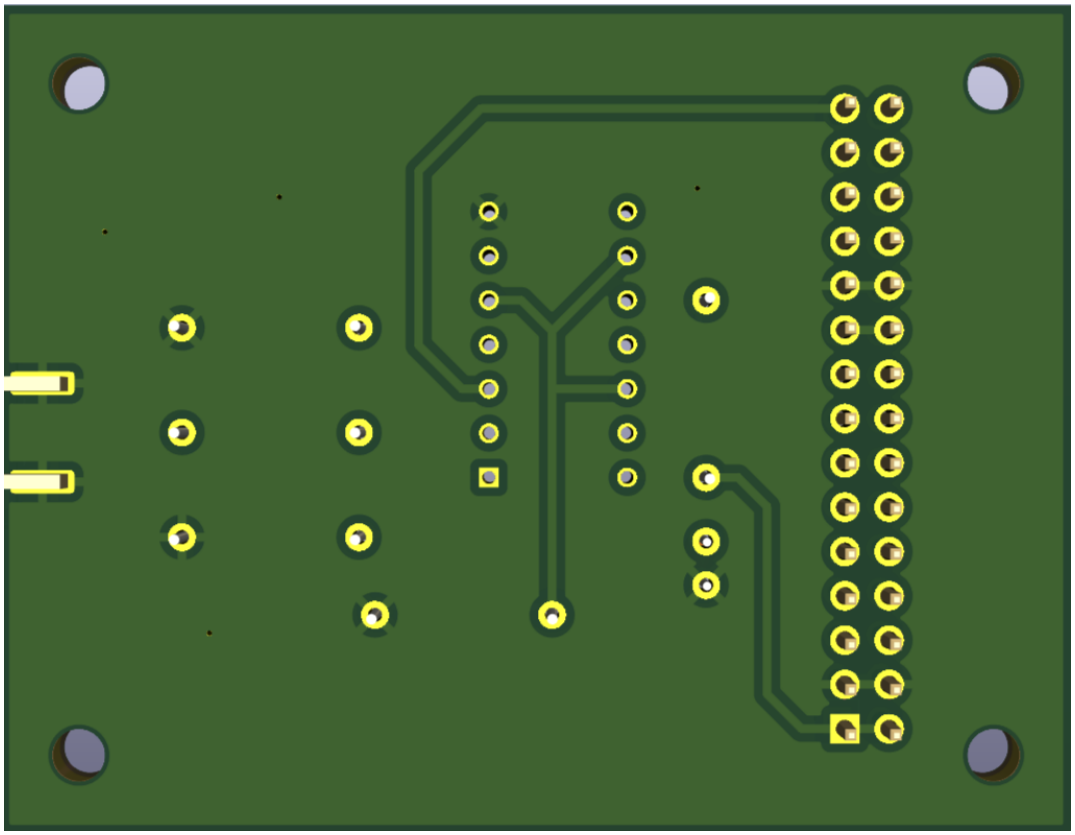


(b) Diseño en PCB de la placa

Figura 3.10: PCB diseñada en KiCad para implementar la señal del *RESET*



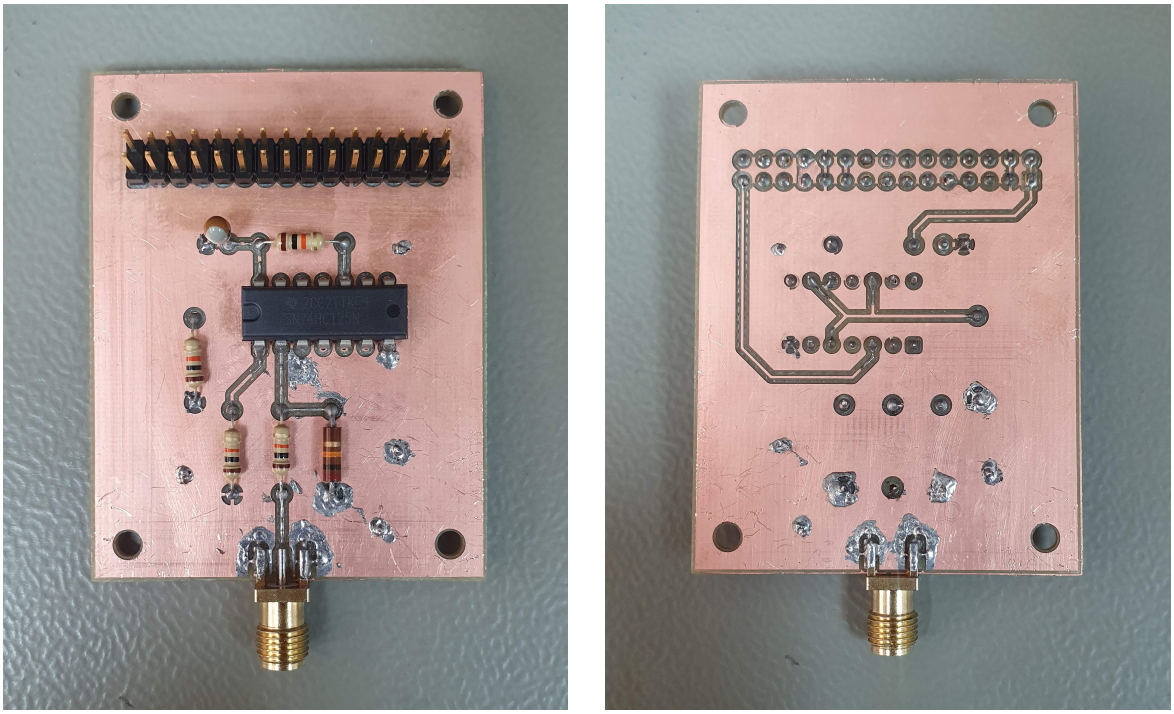
(a) Vista frontal de la placa



(b) Vista trasera de la placa

Figura 3.11: Modelo en 3D de la PCB diseñada con KiCad para la señal de *RESET*.

Finalmente, en la Figura 3.12 se tienen imágenes de la PCB resultante, con todos los componentes integrados en el circuito.



(a) Vista frontal de la placa

(b) Vista trasera de la placa

Figura 3.12: Fotos de la PCB diseñada para la señal de *RESET*.

Con la placa ya confeccionada, el modelo del espectrómetro tuvo que ser modificado para que el bloque ACC_CNTRL recibiera una señal que reinicie el conteo de acumulaciones. En la Figura 3.13, se tiene el diagrama de flujo del control de acumulaciones con el *RESET* integrado en el modelo. Para hacer esta modificación solo fue necesario cambiar la conexión de entrada del bloque ACC_CNTRL.

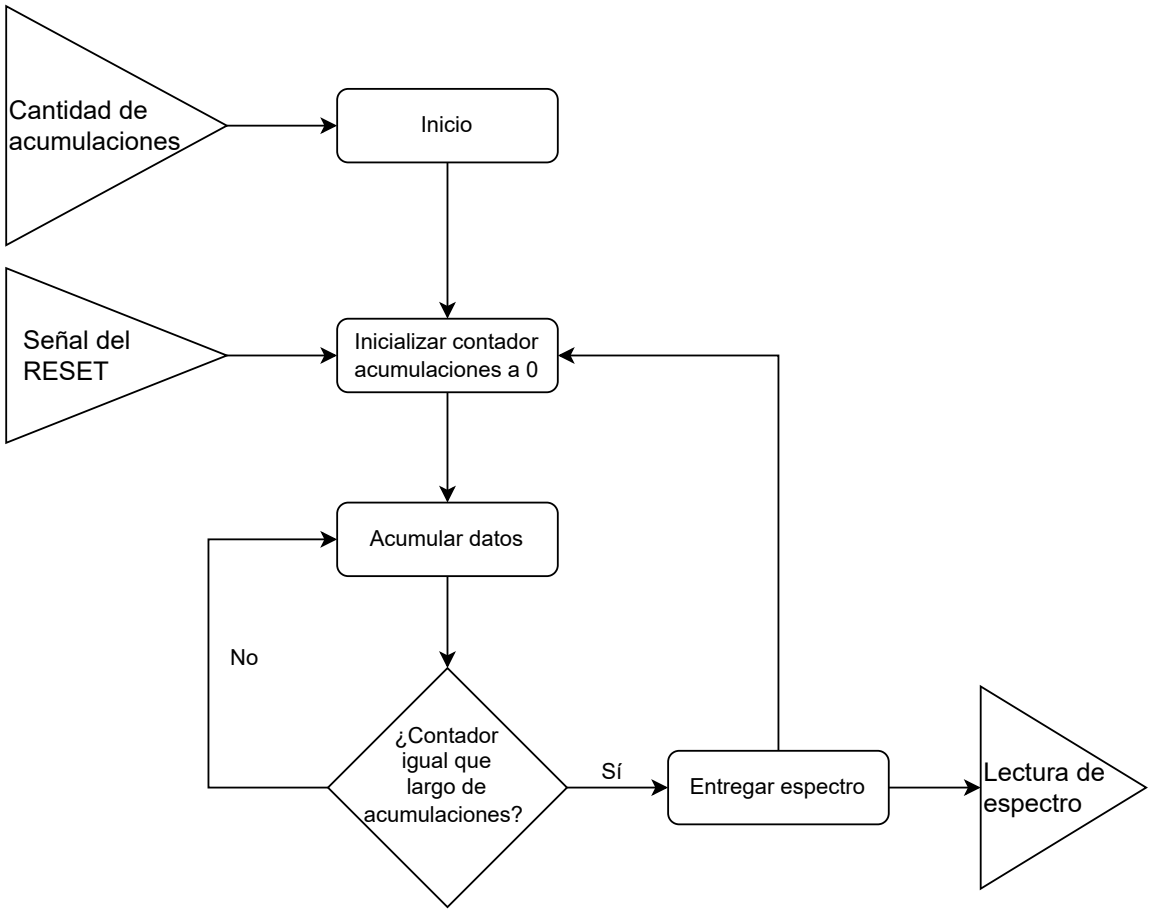


Figura 3.13: Diagrama de Flujo del Sistema de Control de Acumulaciones con *RESET* integrado.

Otra modificación del modelo que se tuvo que agregar fue la inclusión de un contador de espectros totales acumulados. Esto debido, a que el radiotelescopio Mini constantemente está consultando al espectrómetro si hay algún espectro listo para ser medido.

Se agregó un contador, denominado ACC_CYCLE, que entrega la cantidad de espectros acumulados entre cada canto de bajada en la señal *RESET*. Estos datos se leen en el bloque registrador ACC_PER_CYCLE.

También se agregó un contador, denominado ACC_CNTR, el cual está conectado a un bloque registrador ACC_CNT, que el Radiotelescopio constantemente va a estar consultando. En caso de que aumente en 1, se manda un espectro al computador. Este contador va a aumentar en 1 siempre y cuando el bloque ACC_PER_CYCLE entregue un valor distinto de 0, lo que quiere decir que hubo al menos un espectro acumulado entre dos cantos de bajada de la señal *RESET*. En la Figura 3.14, se tiene el sistema lógico del contador de espectros acumulados en el RFSoC, en Simulink.

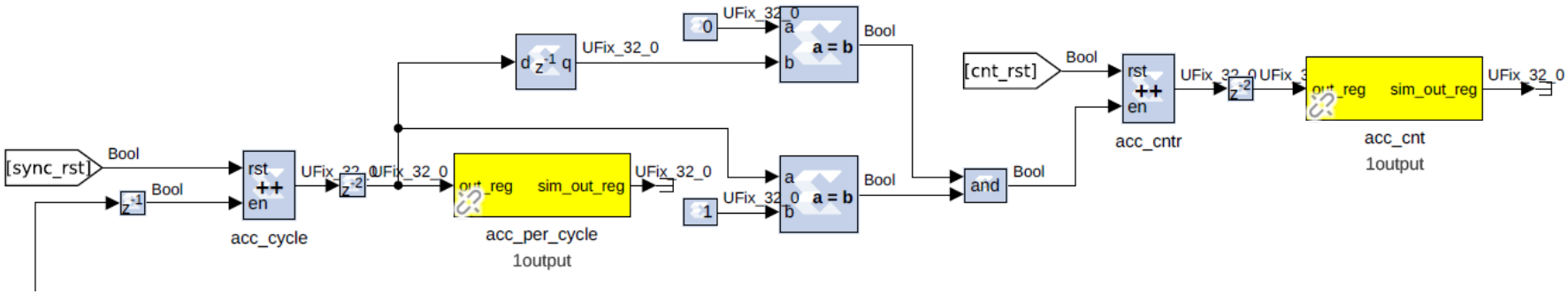


Figura 3.14: Sistema lógico del contador de espectros ACC_CNTR

3.2.2.3. Segunda Re-Adaptación: Re-Cuantización de bits

Otro aspecto importante a considerar en el diseño del espectrómetro es la limitación del microcontrolador PIC32, el cual solo puede procesar datos de hasta 32 bits. Esto implica que los espectros generados por la RFSoC, que originalmente pueden tener una resolución superior, deben ser reescalados o convertidos a un formato compatible de 32 bits. El problema de reducir la cantidad de bits disponibles es que se pierde precisión, especialmente en los valores de menor magnitud. Como consecuencia, los canales con potencias bajas pueden volverse indistinguibles del ruido de fondo o incluso aproximarse a cero, lo que degrada la fidelidad del espectro y dificulta la detección de señales débiles.

Para mitigar este problema, una estrategia consiste en aplicar una ganancia a los valores de potencia antes de realizar la reducción de bits. De este modo, se amplifican los niveles bajos y se preserva mejor la resolución relativa al convertir los datos a 32 bits. En la Figura 3.15 se muestra el bloque ROUND0, encargado de realizar el reescalamiento mediante multiplicación por una ganancia definida. Posteriormente, el bloque CAST lleva a cabo la reducción de la cantidad de bits. La ganancia utilizada es un parámetro que debe definirse en el momento de programar la RFSoC, y su elección implica un compromiso entre preservar la precisión de los valores pequeños y evitar la saturación de los canales con señales más intensas.

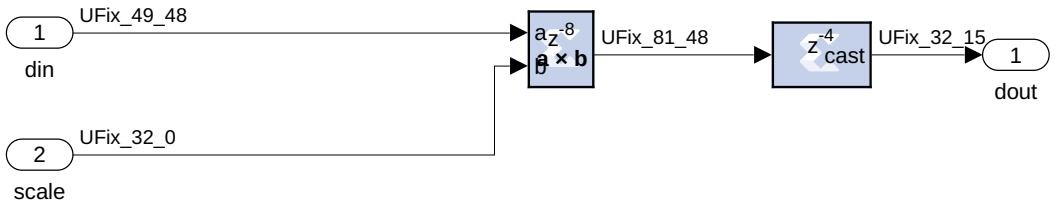


Figura 3.15: Re-Cuantización de bits con el bloque ROUND0

El resto de la arquitectura del espectrómetro permanece sin modificaciones respecto al modelo original. Una vez alcanzado el número de acumulaciones definido, los espectros son leídos desde los bloques BRAM de lectura, con la diferencia de que en esta versión los datos están almacenados en un formato de 32 bits. Esta adaptación permite compatibilizar el sistema con las restricciones del microcontrolador PIC32.

3.2.2.4. Tercera Re-Adaptación: Modelo de 8192 canales espectrales con ancho de banda IF de 1.966 GHz + Correlador + Señal de *RESET* + Re-Cuantización de bits

Esta última iteración consistió en juntar las adaptaciones previas en un mismo modelo de Simulink de 8192 canales espectrales. Este modelo se puede apreciar en la Figura D.1, en la sección de Anexos.

3.3. Mediciones en el laboratorio

3.3.1. Setup de laboratorio

Para simular el *Front End* en el laboratorio, se utilizó un *splitter* de 90°, el cual separa la señal de entrada en dos señales similares con un desfase de 90°. Este componente tiene un rango de operación de frecuencias entre 2000 y 4200 MHz[13]. A continuación, se emplearon dos *mixers* que combinan ambas señales con la señal de un oscilador local, generada a partir de un segundo *splitter* de 0°, cuyo rango de operación va de 1500 a 18000 MHz[14]. De esta forma se obtienen dos señales en la banda IF, correspondientes a las componentes en cuadratura I y Q, emulando el comportamiento de un híbrido de cuadratura en la banda RF. Estas señales fueron posteriormente filtradas y amplificadas. En la Figura 3.16 se muestra el *Front End* analógico utilizado durante las mediciones.

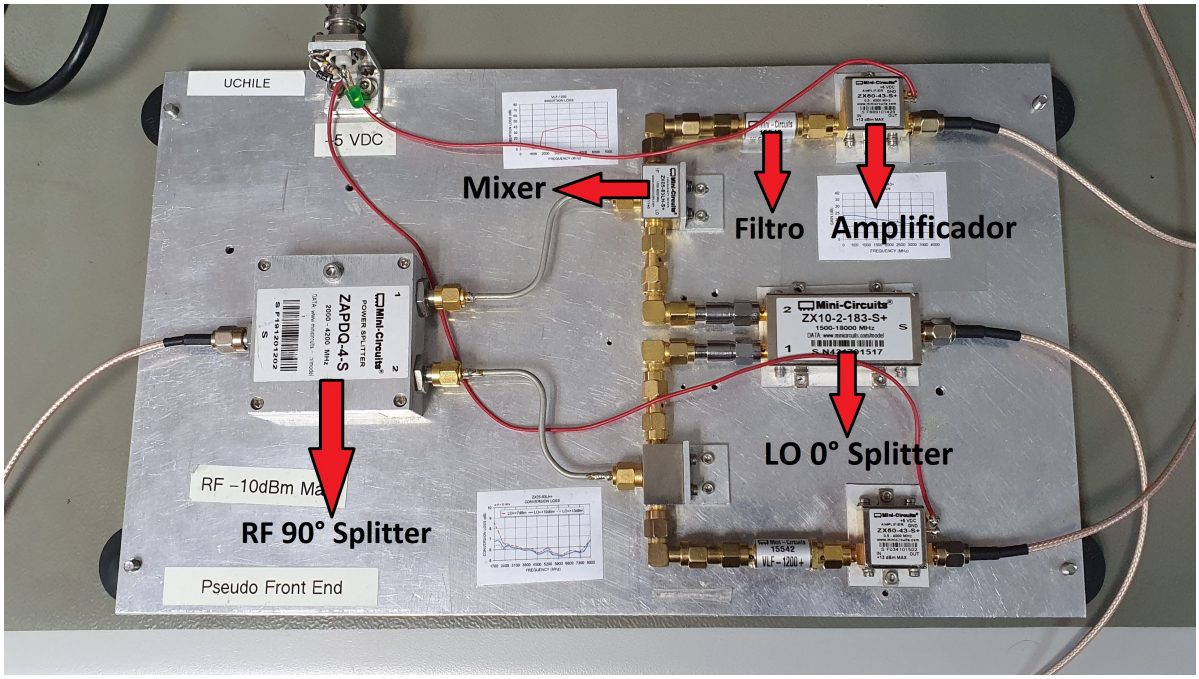


Figura 3.16: *Front End* analógico para frecuencias RF

Por otro lado, en la Figura 3.17 se muestra una imagen de una de los RFSoC 4x2 disponibles en el Laboratorio de Ondas Milimétricas (MWL), correspondiente a la parte *Back End* del proyecto.

[13] https://www.minicircuits.com/WebStore/dashboard.html?model=ZAPDQ-4-S&srsltid=AfmBOoo6J_8PjJ86_twsa_-lpy73AhYu_5IUOjXanWT7aRtz1vgZR-Fp

[14] https://www.minicircuits.com/WebStore/dashboard.html?model=ZX10-2-183-S%2B&srsltid=AfmBOopFUJcKWnxd6uMiJkPzD6zJ0GhoOfOCtd-zOvZz-GN_5_pteQJ1

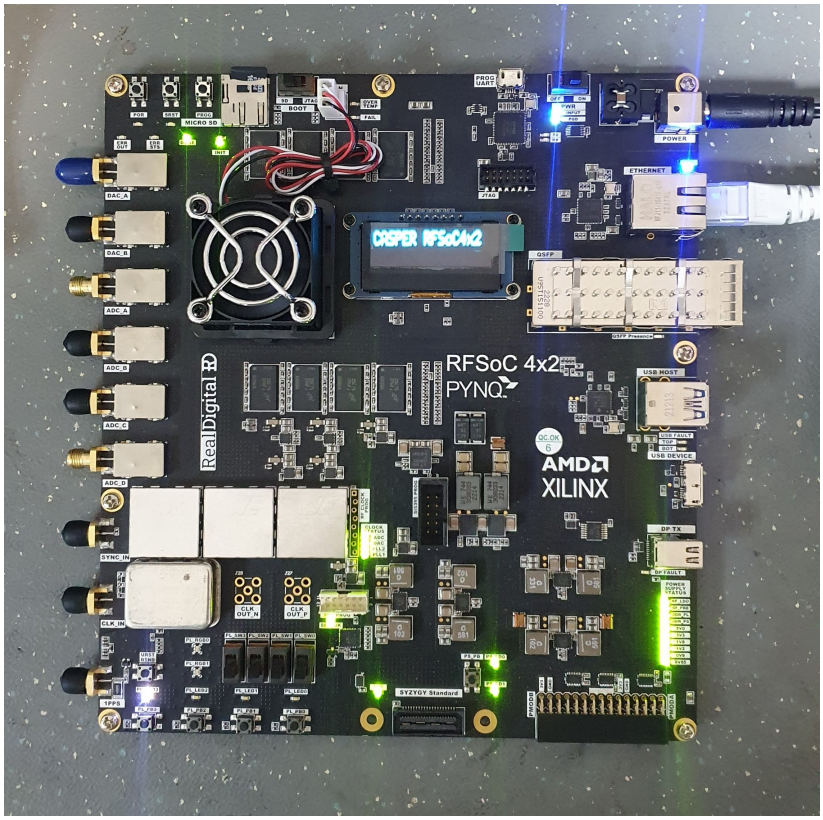


Figura 3.17: Imagen de una de las RFSoC4x2 disponibles en el *MWL*, en Cerro Calán.

Una vez identificadas las etapas de *Front End* y *Back End*, los instrumentos se conectan conforme al esquema mostrado en la Figura 3.18. Para simular la señal de entrada en RF y la señal del oscilador local (LO), se utilizaron dos generadores de señales: un Siglent SSG5085A[15] para la señal RF y un Rohde & Schwarz SMB100A[16] para la señal LO. Para alimentar los amplificadores del *Front End* se usó una fuente de voltaje MCP M10-QD305[17]. A continuación, las salidas de esta etapa se conectan a los ADC del RFSoC 4x2, el cual realiza el proceso de separación de bandas laterales. Además, el FPGA está conectado a la red del laboratorio mediante un cable Ethernet, lo que permite recuperar y visualizar los datos desde cualquier computador que esté conectado a la misma red.

Para la configuración experimental, se definieron los siguientes parámetros de operación. La señal de entrada en RF se generó con una potencia de -10 dBm, mientras que la señal del LO se configuró con una potencia de 15 dBm. Por otro lado, los amplificadores fueron alimentados con una fuente de voltaje ajustada a 5 V. En todas las mediciones realizadas se utilizó una frecuencia fija de LO de 3 GHz y un largo de acumulación de 2^{10} ciclos de FFT. Estas condiciones se mantuvieron constantes tanto para la visualización del espectro como para las pruebas de desempeño del espectrómetro.

[15] https://www.siglent.eu/product/8389211/siglent-ssg5085a-9-khz-20-ghz-microwave-signal-generator
[16] https://www.rohde-schwarz.com/products/test-and-measurement/analog-signal-generators/rs-smb100a-microwave-signal-generator_63493-9379.html
[17] https://multi-com.eu/,details,id_pr,16979,key,laboratory-power-supply-mcp-m10-qd305-2x30v-5a.html

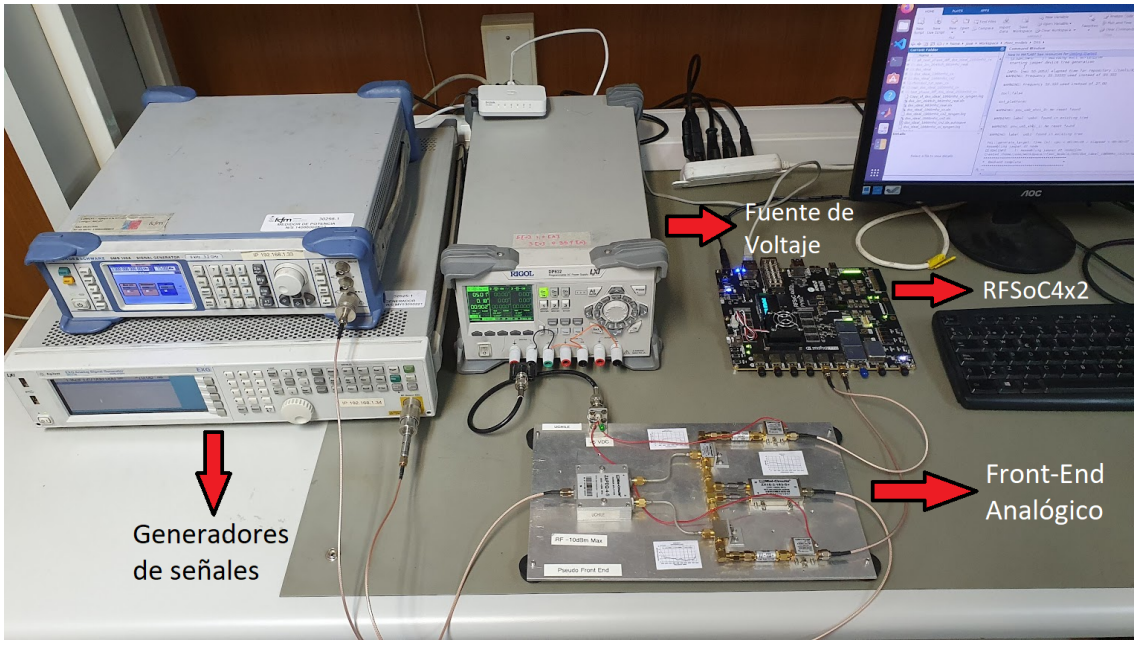


Figura 3.18: Setup de laboratorio

3.3.2. Visualización del espectro

Para visualizar el espectro, se diseñó un código en Python utilizando la librería `casperfpga` desarrollada por el grupo CASPER. Este programa se encarga de leer los bloques de memoria BRAM programados en el FPGA, los cuales contienen los datos del espectro codificados en bytes. Estos valores son convertidos a números enteros y luego graficados en tiempo real, mostrando el contenido espectral tanto de la banda lateral inferior como de la superior.

En este experimento, se generaron señales en RF con frecuencias de 2.5 GHz y 3.5 GHz, las cuales fueron inyectadas manualmente con el generador de señales al *Front End* y procesadas por el sistema completo. Estas mediciones se realizaron utilizando los tres modelos de espectrómetro con salida de 64 bits, correspondientes a 2048, 8192 y 16384 canales espectrales. Además, se incluyeron mediciones con un modelo adicional de 8192 canales con salida de 32 bits, sobre el cual se realizaron pruebas adicionales variando el valor de ganancia aplicado en la re-cuantización de los datos. Esto permitió comparar cómo afecta la ganancia al nivel de los tonos representados en el espectro, el cual fue graficado en decibeles.

El Código B.1 corresponde al código desarrollado. Al inicializarse, el programa configura el FPGA cargando el modelo del espectrómetro correspondiente, define el número de acumulaciones, y, en caso de utilizarse un espectrómetro con salida de 32 bits, también carga un valor de ganancia para la re-cuantización de los datos.

3.3.3. Medición de SRR

Para la medición de la razón de rechazo entre bandas laterales (SRR), se diseñó un código en Python que realiza un barrido en la frecuencia RF de entrada al *Front End*, desde $f_{\mathrm{LO}} - \mathrm{BW}$ hasta $f_{\mathrm{LO}} + \mathrm{BW}$, cubriendo así de forma automatizada tanto la banda LSB, como la USB. En cada punto del barrido, el sistema mide la potencia correspondiente en ambas bandas y calcula su diferencia en decibeles. Al finalizar el proceso, se genera un gráfico que

muestra la evolución del SRR en función de la frecuencia de entrada.

Esta prueba se llevó a cabo utilizando los tres modelos de espectrómetro de 64 bits (2048, 8192 y 16384 canales espectrales), así como un modelo de 32 bits con 8192 canales, el cual fue evaluado bajo distintas configuraciones de ganancia. En el Código B.2 se muestra el *script* encargado de ejecutar este procedimiento. Al igual que en el caso de la visualización del espectro, al inicializarse, el programa configura el RFSoC cargando el modelo correspondiente, define el número de acumulaciones y, si se trata de un espectrómetro de 32 bits, aplica la ganancia especificada para la re-cuantización.

3.3.4. Medición de diferencia de fases para las salidas I y Q

Para este experimento, la medición de la diferencia de fase entre las salidas I y Q se realizó mediante un procedimiento similar al utilizado para el cálculo del SRR. Se ejecutó un barrido en frecuencia que abarca todo el ancho de banda procesado, desde $f_{\mathrm{LO}} - \mathrm{BW}$ hasta $f_{\mathrm{LO}} + \mathrm{BW}$. En cada paso del barrido, el sistema lee los bloques de memoria BRAM y extrae el ángulo de la correlación compleja resultante entre las señales I y Q. Este ángulo representa la diferencia de fase entre ambas componentes, permitiendo evaluar si la relación de cuadratura ideal (90º de desfase) se mantiene en todo el rango de frecuencias.

El Código B.3, utilizado para esta medición, inicializa el modelo correspondiente cargándolo en el FPGA y asigna el largo de acumulaciones a utilizar. Esta prueba se realizó únicamente con los tres modelos de espectrómetro de 64 bits (2048, 8192 y 16384 canales), ya que no se diseñó ni implementó un bloque de correlación compatible con la arquitectura de 32 bits. Por esta razón, el experimento se ejecutó solamente tres veces, y no fue necesario realizar pruebas adicionales con diferentes ganancias ni aplicar re-cuantización de bits, a diferencia del caso del SRR.

3.3.5. Medición del desempeño de la PCB y señal de *RESET*

Uno de los aspectos críticos en el diseño del sistema de control es asegurar que la señal periódica de *RESET*, generada externamente por el sistema de referencia del radiotelescopio, llegue correctamente al RFSoC. Esta señal es esencial para reiniciar o sincronizar el ciclo de acumulaciones en el espectrómetro. Por ello, se realizaron pruebas de validación en laboratorio para comprobar que la señal de *RESET* transmitida a través de la PCB diseñada era efectivamente reconocida por el sistema digital del RFSoC en los tiempos esperados.

En la Figura 3.19 se tiene el setup con el que se midió el desempeño de la PCB diseñada. Se usó un generador de ondas RIGOL DG2102[18], y se programó para que generara señales cuadradas como las del generador de referencia en el Mini. El RFSoC aquí también está conectado a la red mediante un cable Ethernet, lo que le permite a un computador recolectar la información que le entregue el FPGA.

[18] https://www.rigolna.com/products/waveform-generators/dg2000/?srsltid=AfmBOoq97r1cXstuo0T1HdxMijs28aB78jmi9OjMalVChTyMy2-JRMNx

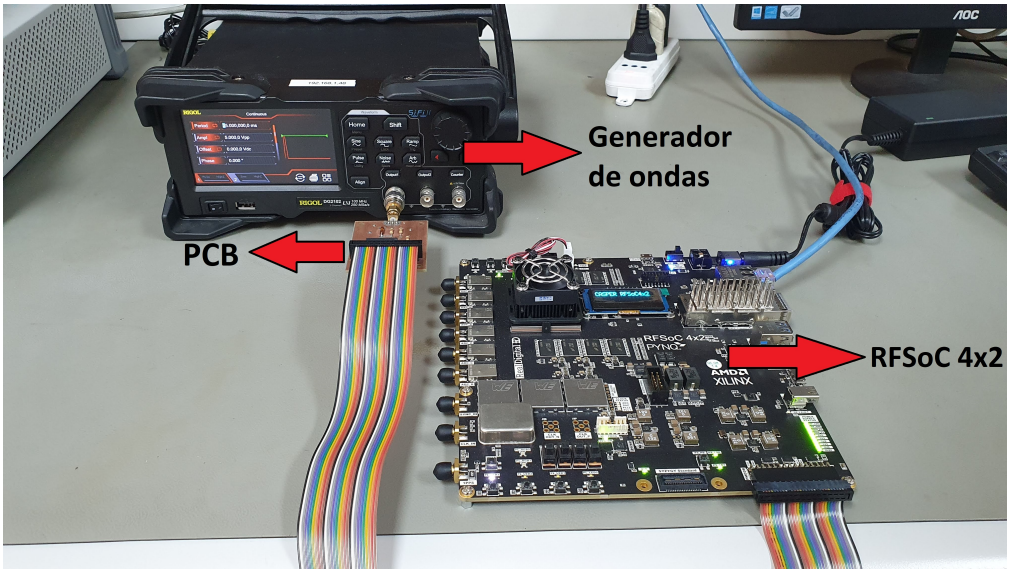


Figura 3.19: Setup de medición para evaluar el desempeño de la PCB

En estas mediciones, se evaluó el desempeño de la señal de *RESET*. Para esto se midió el tiempo que tardaba el RFSoC en mostrar un 0 en el contador *acc_per_cycle*, esto con el fin de medir la precisión con la que la FPGA podía medir un canto de bajada para medir un espectro. Estas mediciones se llevaron a cabo utilizando el script `test_spec_cnt.py`, presente en el Código B.4.

3.3.6. Evaluación de la velocidad de lectura de espectros

Con el objetivo de evaluar el desempeño temporal del sistema de adquisición de datos, se realizaron pruebas para medir los tiempos con los que se recuperan los espectros generados por el RFSoC desde un computador externo. Para ello, se implementaron y compararon dos métodos distintos de adquisición de datos.

El primer método empleó la librería `casperfpga`, desarrollada por el grupo CASPER, la cual se comunica con el FPGA a través del protocolo KATCP[19]. En este experimento, se midió la latencia con la que se leen los datos espectrales durante un intervalo de tiempo determinado. El código utilizado para este procedimiento se presenta en el Código B.5. Además, se extrajeron métricas estadísticas tales como el promedio, la desviación estándar, los valores máximos y mínimos, así como la cantidad y el momento de ocurrencia de fallos.

El segundo método consistió en una implementación personalizada de un sistema cliente-servidor basado en sockets. El servidor fue programado en C++ y ejecutado directamente en el RFSoC; su código se presenta en el Código B.6. El cliente, también desarrollado en C++, solicita los espectros al servidor mediante un socket, como se muestra en el Código B.7. Finalmente, se diseñó una interfaz en Python para conectarse con el servidor y recibir los datos espectrales, cuyo código se encuentra en el Código B.8. Al igual que en el método anterior, se registró la latencia durante un periodo sostenido de operación y se analizaron métricas estadísticas para comparar el desempeño de ambas implementaciones.

[19] https://pythonhosted.org/katcp/

Ambas metodologías se evaluaron utilizando el modelo final del espectrómetro digital, correspondiente a 8192 canales espectrales con salida de 32 bits. Durante un periodo continuo de una hora, se registró el tiempo necesario para adquirir espectros completos mediante ambos métodos, variando la cantidad de canales leídos en cada medición. Específicamente, se realizaron mediciones para distintas cantidades de espectros: 512, 1024, 2048, 4096 y finalmente los 8192 canales del espectro completo, con el objetivo de observar cómo escala el tiempo de lectura en función del volumen de datos transferidos.

Esta evaluación permite comparar directamente la eficiencia de lectura entre una solución basada en librerías existentes y una implementación optimizada mediante comunicación directa, entregando información clave para decidir qué enfoque es más adecuado para una futura operación en tiempo real o para implementaciones más exigentes en términos de latencia.

3.4. Mediciones en el Radiotelescopio

En la Figura 3.20 se muestra el esquema de conexión utilizado para poner a prueba el espectrómetro digital en el radiotelescopio Mini. Esta configuración es similar a la empleada durante las pruebas en laboratorio: los convertidores analógico-digitales (ADC) de la RFSoC se conectan directamente a la cadena IF que entrega el *front end* del telescopio, mientras que la señal de control *RESET* proveniente del generador de referencia es conectada al pin PMOD programado de la RFSoC, utilizando la placa PCB diseñada como interfaz.

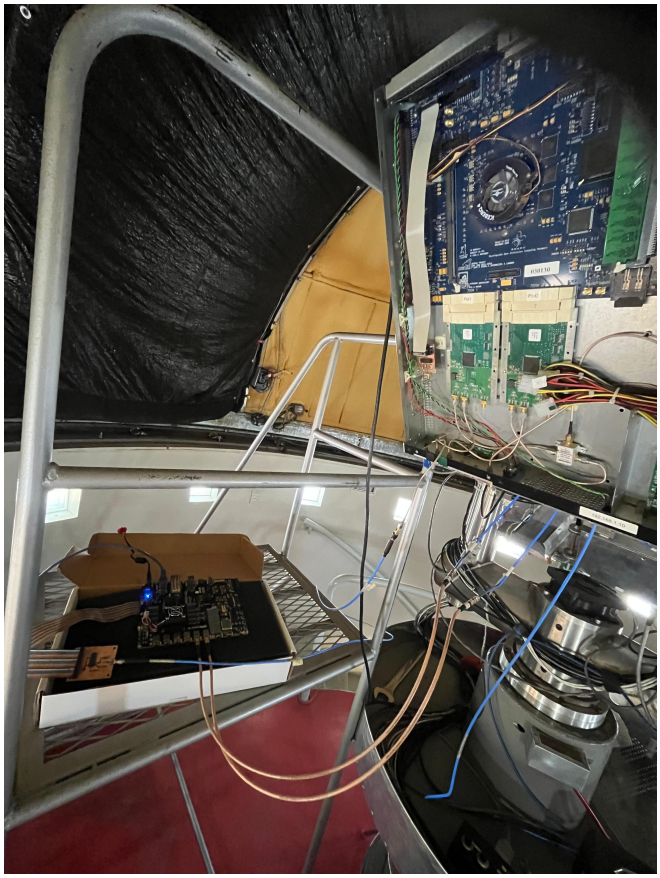


Figura 3.20: RFSoC 4x2 conectado al radiotelescopio Mini

Las pruebas realizadas en el telescopio se centraron principalmente en verificar la operación del nuevo espectrómetro digital implementado en el RFSoC, así como su compatibilidad con el sistema de control y los parámetros adecuados para su operación. Las pruebas fueron las siguientes:

1. **Evaluación del largo de acumulaciones según la señal *RESET*:** Esta prueba tuvo como objetivo analizar cuántos espectros eran generados completamente entre cada flanco de bajada de la señal *RESET*, utilizada como señal de control del radiotelescopio. Se realizaron mediciones para distintos valores del largo de acumulación, con el fin de determinar configuraciones que permitieran obtener una cantidad de espectros adecuada por cada ciclo de la señal. Esta información resulta clave para ajustar el espectrómetro a las condiciones operativas del telescopio y maximizar su utilidad científica. Estas mediciones se llevaron a cabo utilizando nuevamente el *script* `test_spec_cnt.py`, presente en el Código B.4.

2. **Comunicación del nuevo espectrómetro con el sistema completo del radiotelescopio y visualización de espectros:** Para esta parte del proyecto se diseñó el código que permite la comunicación del RFSoC 4x2 con el sistema completo del radiotelescopio. Esta comunicación sigue la estructura mostrada en la Figura 3.1. Luego, se comparó el espectro visualizado por el nuevo espectrómetro en el RFSoC con el que recibe actualmente el microcontrolador PIC desde el sistema anterior basado en ROACH, validando que ambos muestren estructuras espectrales consistentes.

3. **Prueba de estabilidad en modo calibración y modo de observación:** Dado que el sistema de operación del radiotelescopio SMWT es muy sensible, debido a que el sistema se detiene si es que hay un fallo en la sincronización o los datos adquiridos tardan más de lo esperado, se realizaron mediciones continuas tanto en modo calibración, como observación durante un minuto, verificando la recepción de datos en el PIC desde el *Data Server*.

Capítulo 4

Resultados y Discusión

4.1. Separación de bandas laterales y funcionamiento del híbrido digital

A continuación, se presentan los resultados del funcionamiento del híbrido. Para esto se visualizó el espectro en cada banda lateral IF, se midió el SRR en cada modelo y se midió la diferencia de fase en las señales I y Q.

4.1.1. Espectros en LSB y USB

4.1.1.1. Modelos de 64 bits

Para todos los modelos se utilizó un largo de acumulación de 2^{10}. A continuación, en las Figuras 4.1 y 4.2, se presentan los resultados obtenidos para el modelo de 64 bits con 8192 canales espectrales, utilizando señales de entrada en RF de 2500 MHz y 3500 MHz, respectivamente. En ambas figuras se puede observar la respuesta espectral en las bandas laterales LSB y USB.

Los resultados correspondientes a los modelos de 2048 y 16384 canales espectrales, bajo las mismas condiciones de prueba (frecuencias de entrada de 2500 MHz y 3500 MHz), se incluyen en el Anexo.C.1, ya que presentan un comportamiento espectral similar.

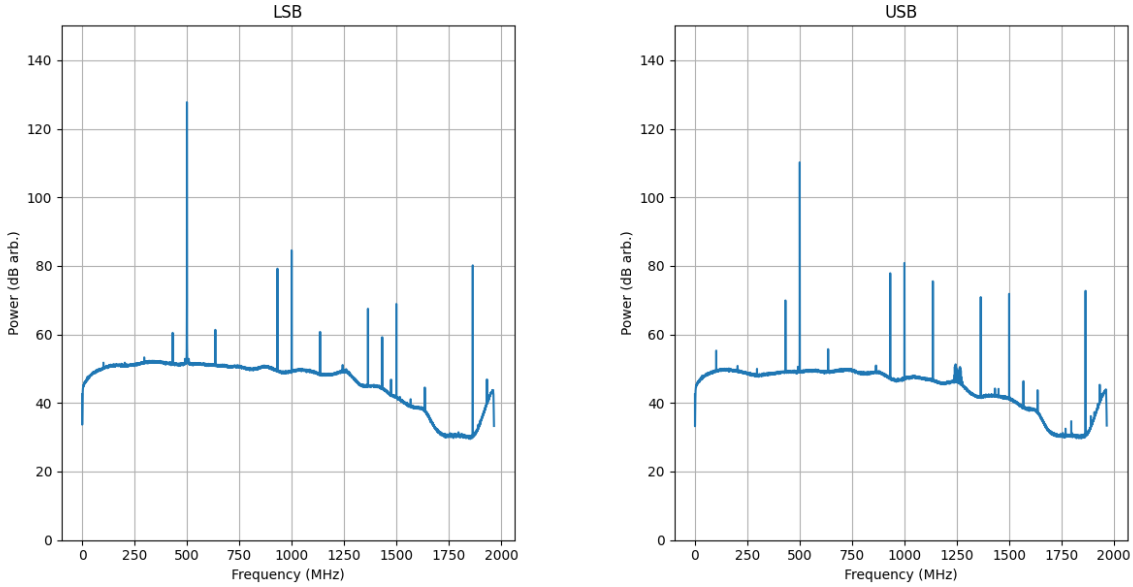


Figura 4.1: Visualización del espectro de 64 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz. Se observan las bandas LSB y USB.

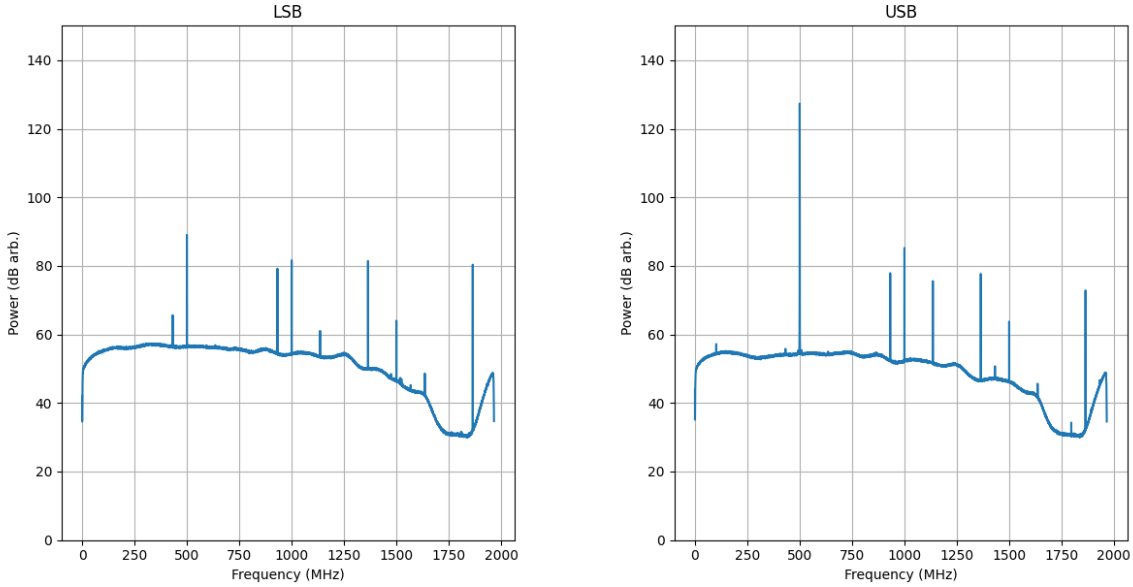


Figura 4.2: Visualización del espectro de 64 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz. Se observan las bandas LSB y USB.

En ambos gráficos se puede apreciar el correcto funcionamiento del híbrido digital encargado de la separación de bandas laterales. Por ejemplo, con una frecuencia de entrada de 2500 MHz —la cual se encuentra en la banda lateral inferior respecto al LO de 3 GHz— se observa un tono predominante en el espectro LSB, centrado alrededor de 500 MHz, con una diferencia de potencia cercana a 20 dB respecto al mismo tono en la banda USB. De forma análoga, para una frecuencia de entrada de 3500 MHz —que pertenece a la banda lateral

superior— el tono principal aparece claramente en la USB, con una diferencia de más de 30 dB en comparación con el LSB.

Los otros tonos presentes en los espectros podrían corresponder a interferencias externas, ruido de los ADC del FPGA o armónicos generados por los generadores de señal utilizados en el experimento. No obstante, las figuras permiten validar que el sistema implementado logra distinguir correctamente entre ambas bandas laterales, reflejando un desempeño adecuado del híbrido digital en las distintas configuraciones espectrales.

Finalmente, cabe resaltar que el *splitter* usado del laboratorio para la frecuencia RF tiene un rango de operación óptimo entre 2000 y 4200 MHz, lo que justifica por qué el piso de ruido empieza a decaer a partir de los 1000 MHz.

4.1.1.2. Modelo de 32 bits

A continuación, usando un largo de acumulación de 2^{10}, se muestran los resultados para un modelo de 32 bits con 8192 canales espectrales, utilizando señales de entrada en RF de 2500 MHz y 3500 MHz.

En las Figuras 4.3 y 4.4 se muestran los espectros usando una ganancia de valor 1. En ambas figuras se puede observar la respuesta espectral en las bandas laterales LSB y USB.

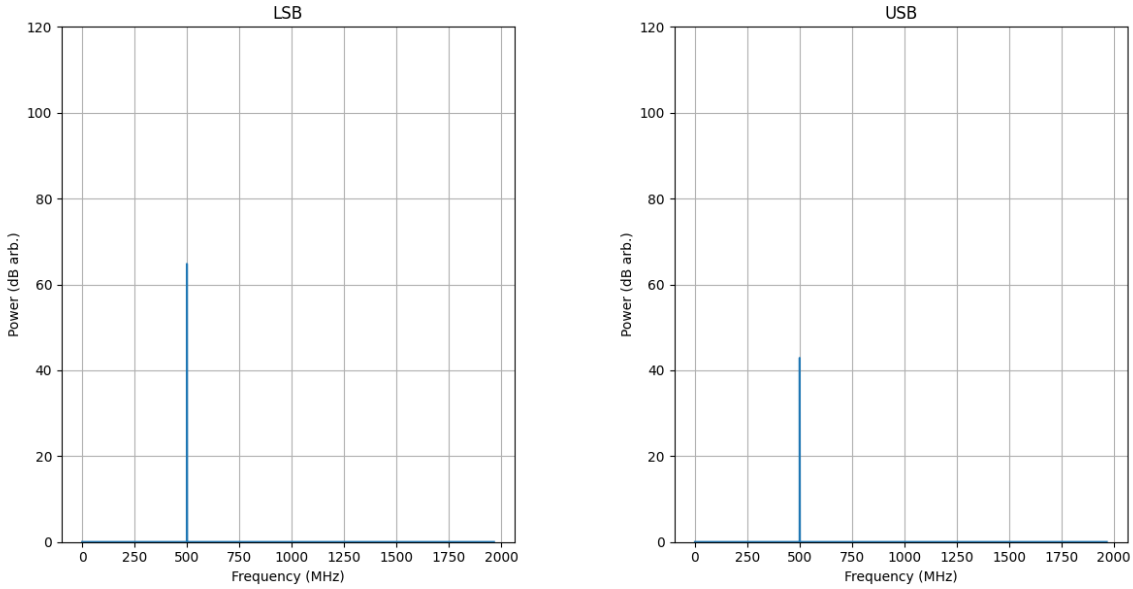


Figura 4.3: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 1. Se observan las bandas LSB y USB.

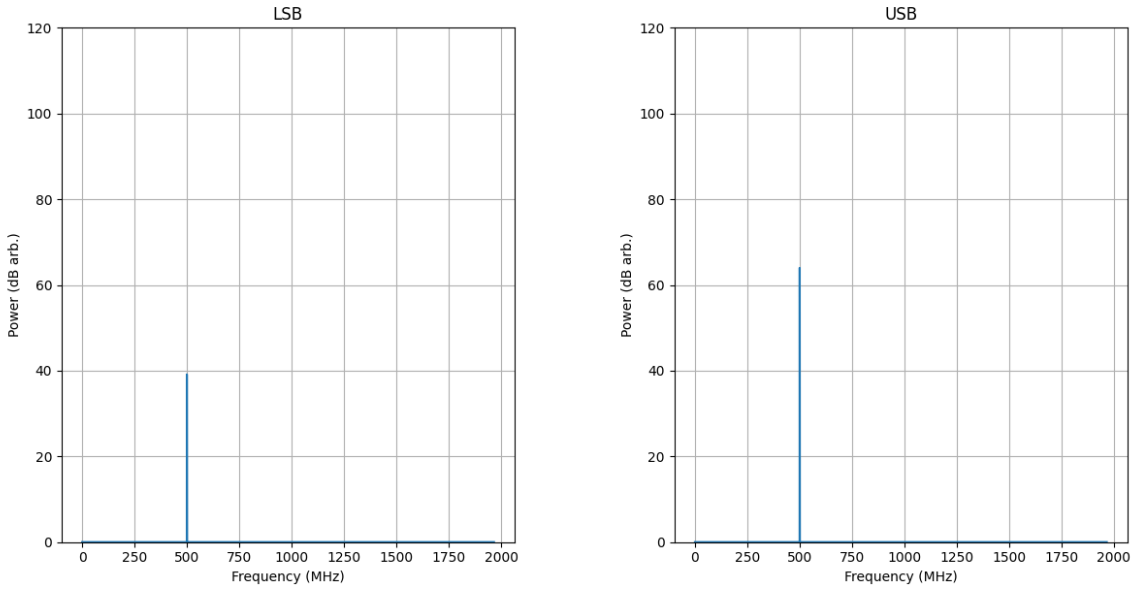


Figura 4.4: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 1. Se observan las bandas LSB y USB.

En las Figuras 4.5 y 4.6 se muestran los espectros usando una ganancia de valor 2^{10}.

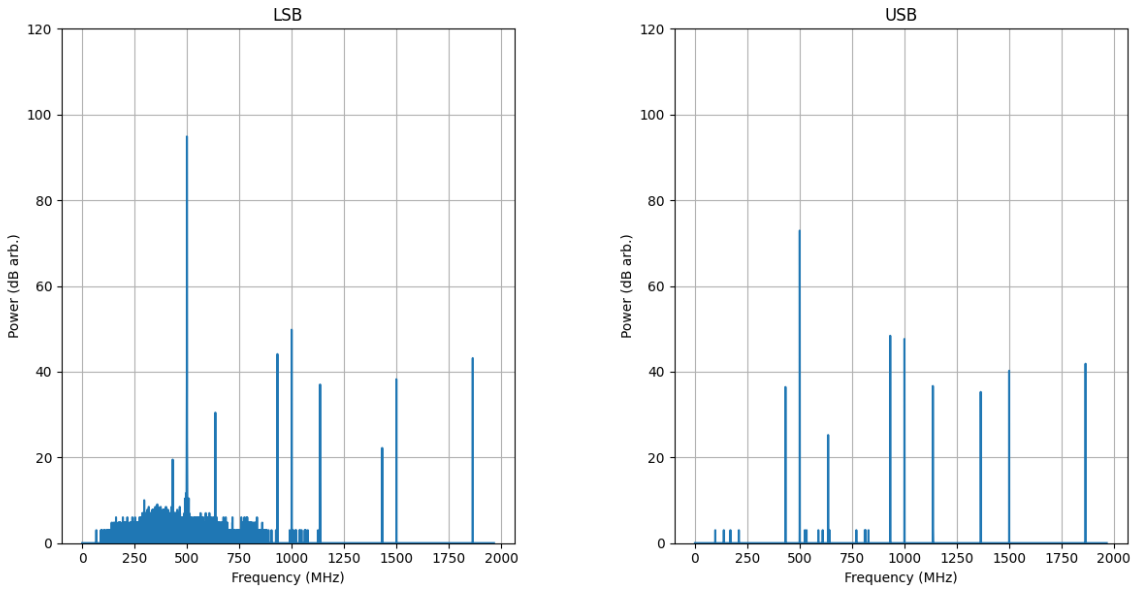


Figura 4.5: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 2^{10}. Se observan las bandas LSB y USB.

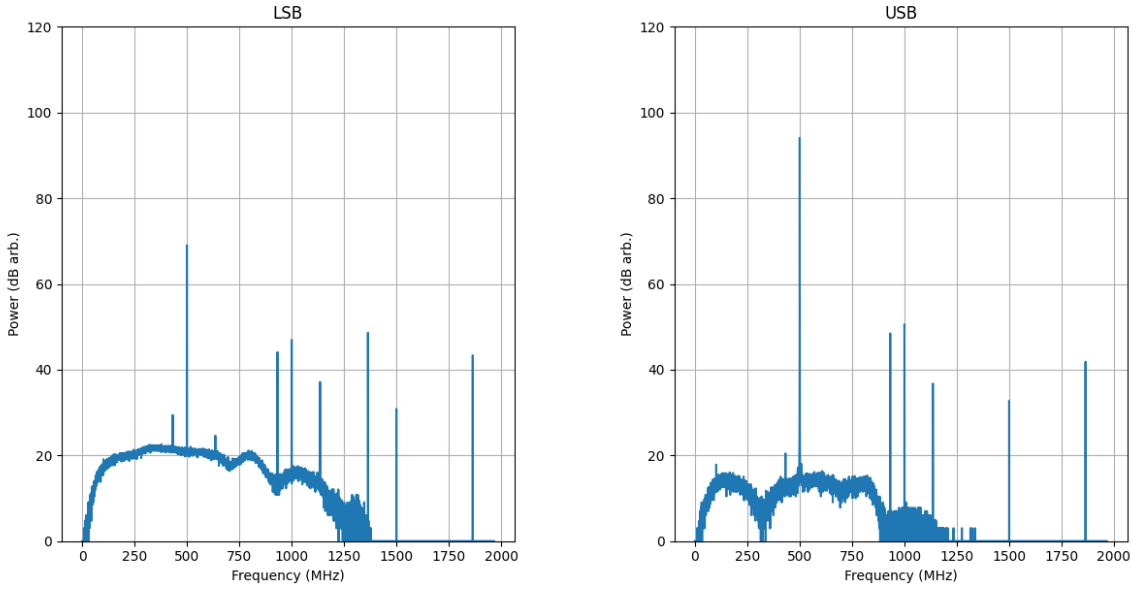


Figura 4.6: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 2^{10}. Se observan las bandas LSB y USB.

En las Figuras 4.7 y 4.8 se muestran los espectros usando una ganancia de valor 2^{15}.

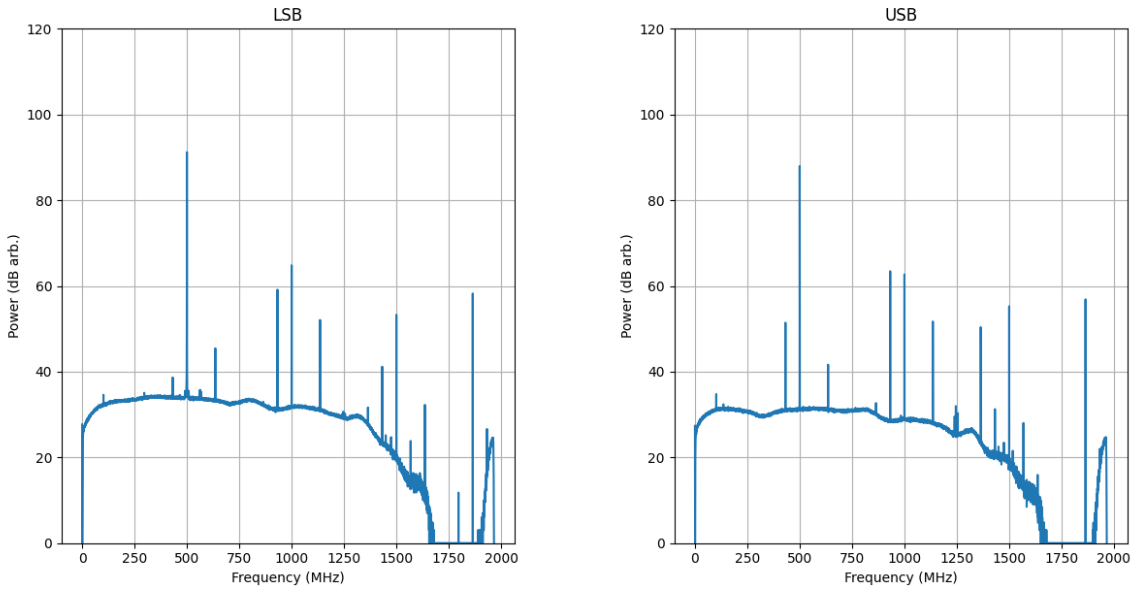


Figura 4.7: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 2^{15}. Se observan las bandas LSB y USB.

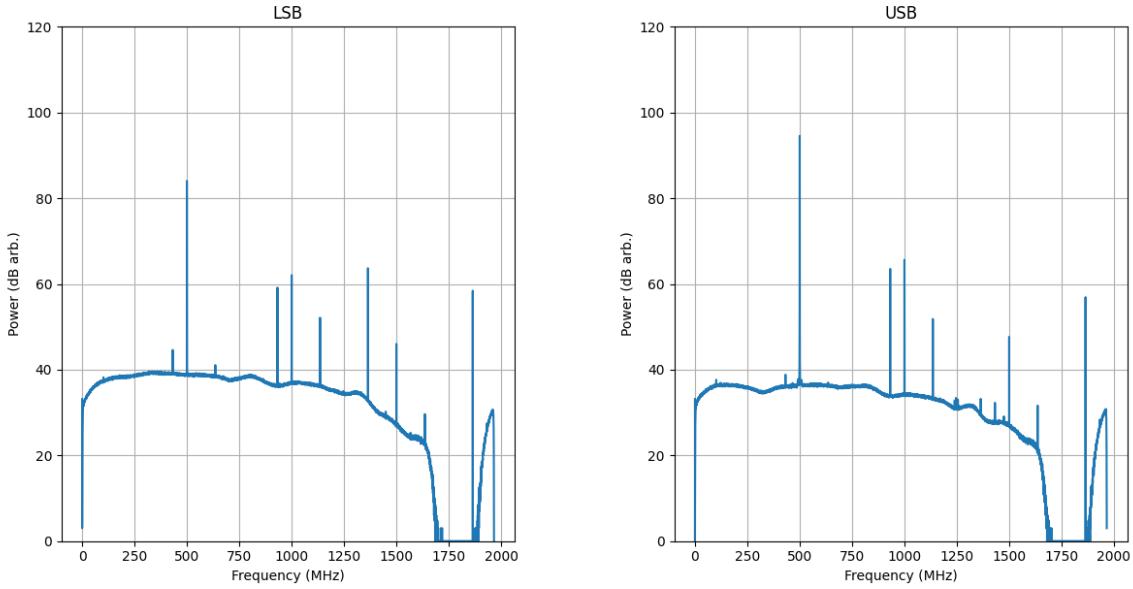


Figura 4.8: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 2^{15}. Se observan las bandas LSB y USB.

En las Figuras 4.9 y 4.10 se muestran los espectros usando una ganancia de valor 2^{20}.

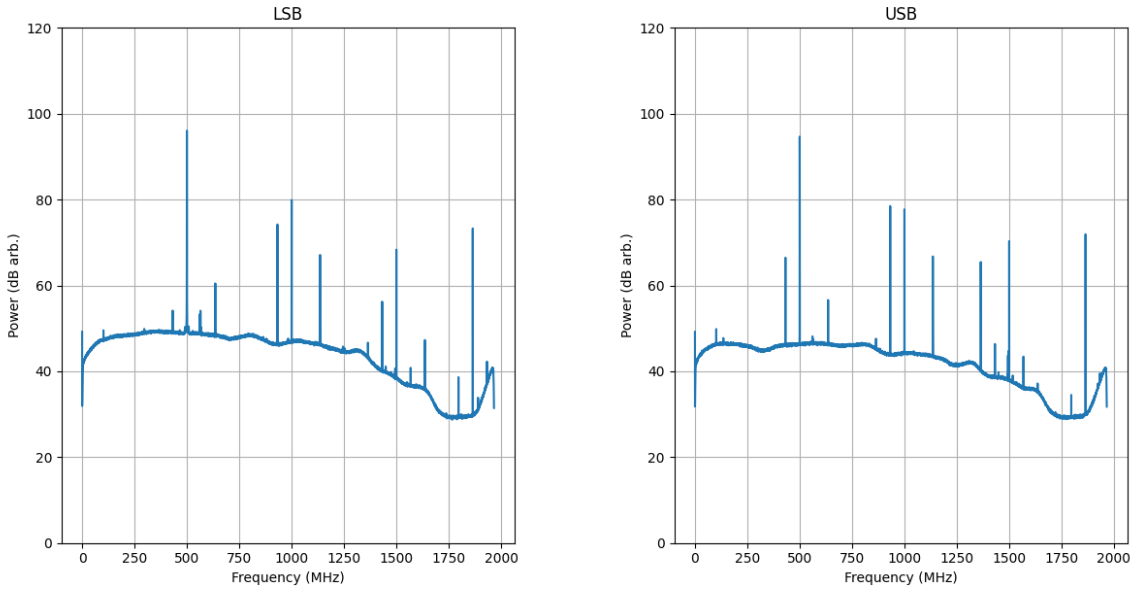


Figura 4.9: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 2500 MHz y ganancia 2^{20}. Se observan las bandas LSB y USB.

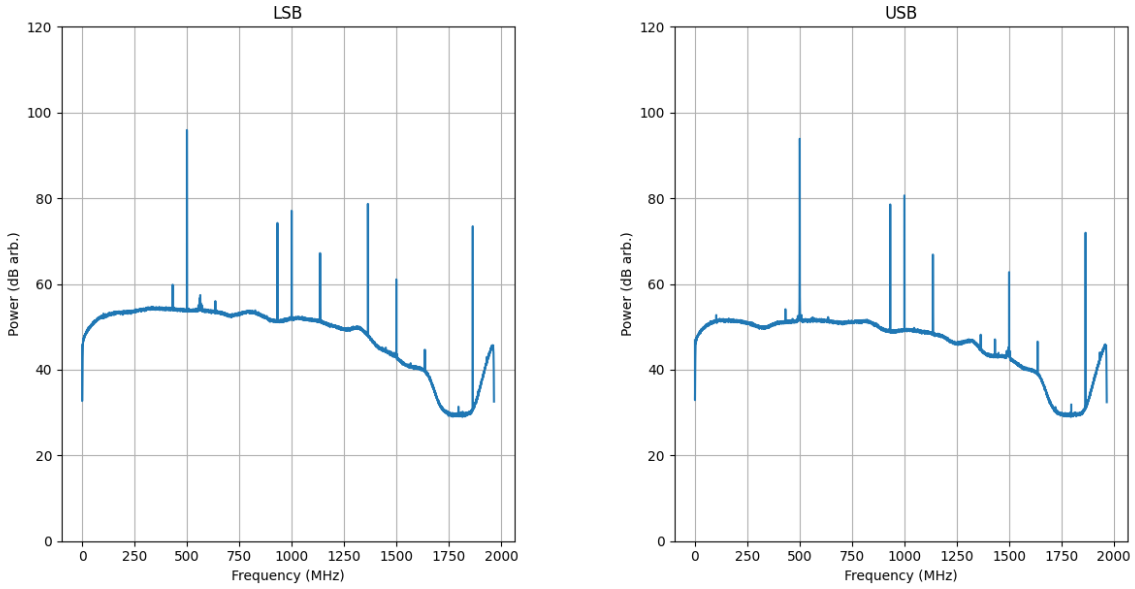


Figura 4.10: Visualización del espectro de 32 bits para 8192 canales, con una frecuencia de entrada RF de 3500 MHz y ganancia 2^{20}. Se observan las bandas LSB y USB.

A partir de estos experimentos se concluye que la re-cuantización de bits es una herramienta efectiva para ajustar el rango dinámico de los espectros generados en un modelo de

32 bits. Al aumentar progresivamente la ganancia, los tonos en las bandas IF se vuelven más visibles, y el piso de ruido del espectro comienza a definirse con mayor claridad. Este ajuste permite que los tonos de interés sobresalgan sobre el ruido, facilitando su identificación. La separación entre bandas laterales se mantiene funcional incluso con ganancias elevadas, como 2^{10}, donde se observa una diferencia de potencia superior a 20 dB entre las bandas LSB y USB.

No obstante, a partir de una ganancia de 2^{15}, el sistema comienza a mostrar signos de saturación. En estos casos, los valores de potencia alcanzan el límite máximo que puede representar el formato de 32 bits, lo cual impide observar adecuadamente la relación esperada entre los tonos de las bandas laterales. Por ejemplo, en la Figura 4.7, correspondiente a una señal de entrada de 2500 MHz, se esperaría un tono claramente dominante en LSB (500 MHz), pero este no se distingue del tono en USB debido a la saturación. Este mismo comportamiento se repite con una ganancia de 2^{20}, confirmando que la información de potencia se ve truncada por limitaciones de como está programado el sistema.

En resumen, para asegurar el correcto funcionamiento del espectrómetro implementado y evitar efectos de saturación, se recomienda utilizar una ganancia moderada, idealmente cercana a 2^{10}. Este valor permite conservar la diferenciación entre tonos y visualizar adecuadamente el piso de ruido sin comprometer la precisión de los datos por sobrecarga numérica.

4.1.2. Medición de SRR

4.1.2.1. Modelos de 64 bits

Para la medición del SRR en los tres modelos de 64 bits se usó un largo de acumulaciones de 2^{10}. En la Figura 4.11, se tiene el SRR del espectrómetro de 8192 canales espectrales. El barrido de frecuencias se hizo abordando ambas bandas laterales, LSB y USB, desde 1033.92 MHz hasta 4966.08 MHz.

Los resultados correspondientes a los modelos de 2048 y 16384 canales espectrales, nuevamente, se incluyen en el Anexo.C.2. En la Tabla 4.1 se resumen las estadísticas del SRR para distintas resoluciones espectrales.

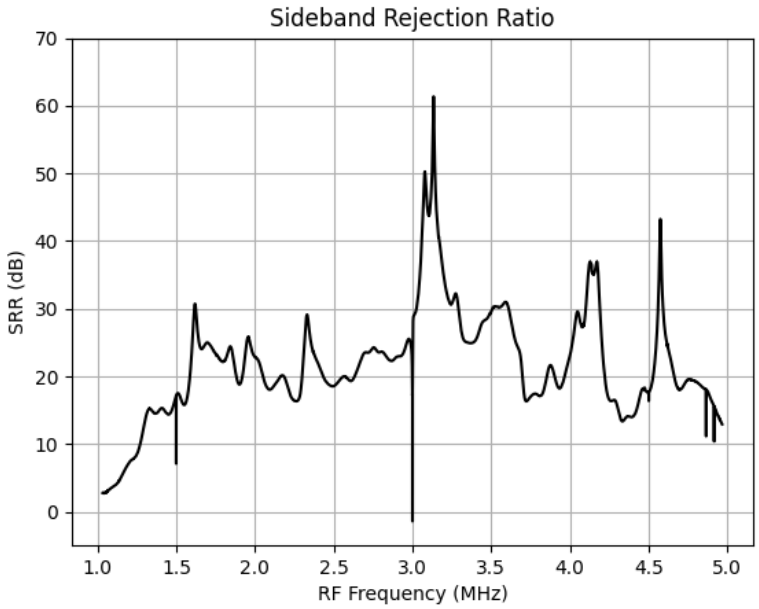


Figura 4.11: SRR para modelo de 64 bits y 8192 canales espectrales.

Tabla 4.1: Estadísticas del SRR para modelos de 64 bits con distintas resoluciones espectrales.

Canales	**Average (dB)**	**Std Dev (dB)**	**Max (dB)**	**Min (dB)**
2048	21.76	7.93	59.42	2.69
8192	21.75	7.97	61.36	2.72
16384	22.76	8.82	54.56	2.37

De estos experimentos, se puede concluir nuevamente que el híbrido digital hace la separación de bandas de forma adecuada, para distintas resoluciones espectrales, incluso sin estar calibrado. Esto se respalda con el promedio de los tres modelos puestos a prueba, siendo los tres superiores a 20 dB.

Nuevamente, se puede notar la limitación del rango de frecuencia del *splitter* de la señal de entrada en RF, ya que en las frecuencias inferiores a 2000 MHz y superiores a 4200 MHz el SRR tiende a disminuir.

4.1.2.2. Modelo de 32 bits

Como se diseñó un único modelo de 32 bits, cuya FFT es de tamaño 8192, se calculó el SRR para distintas ganancias y cantidades de acumulaciones, evaluando su comportamiento a lo largo de toda la banda LSB y USB.

En las siguientes figuras se muestran los resultados obtenidos para diversas configuraciones de ganancia (desde 2^0 hasta 2^{11}) y largos de acumulación (2^8, 2^9 y 2^{10}). Estos parámetros se ajustaron con el fin de observar cómo influyen en el SRR, tanto en términos de estabilidad como de supresión de la banda opuesta.

Los gráficos permiten visualizar qué tan eficiente es la separación entre bandas laterales bajo distintas condiciones de cuantización y largo de acumulaciones. A continuación, se presentan algunos de los casos más representativos:

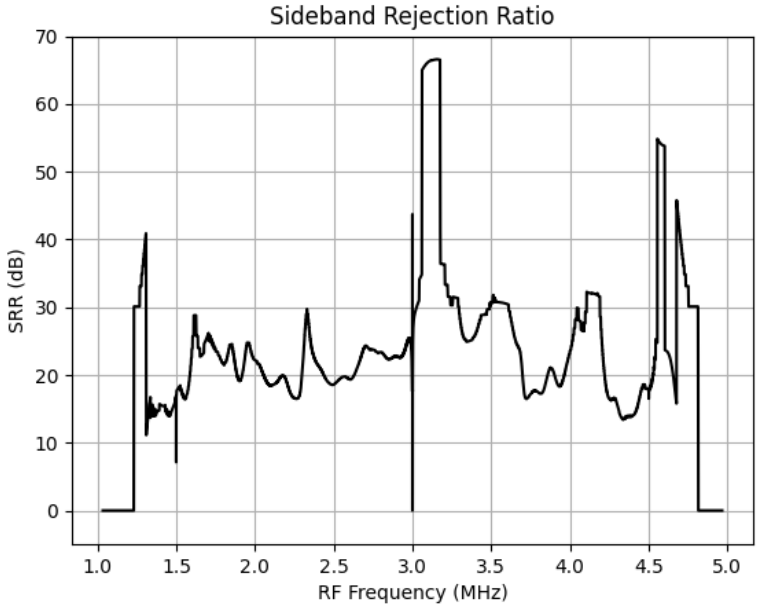


Figura 4.12: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 1 y cantidad de acumulaciones 2^{10}.

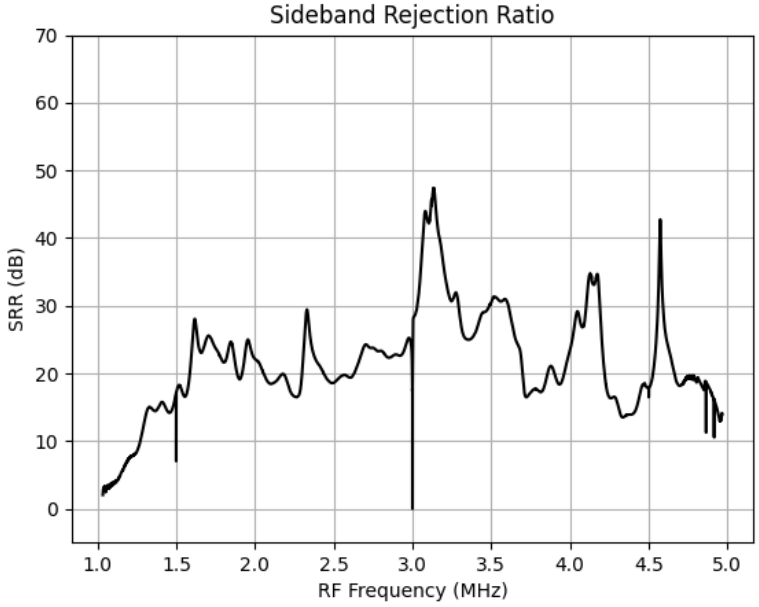


Figura 4.13: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^9 y cantidad de acumulaciones 2^{10}.

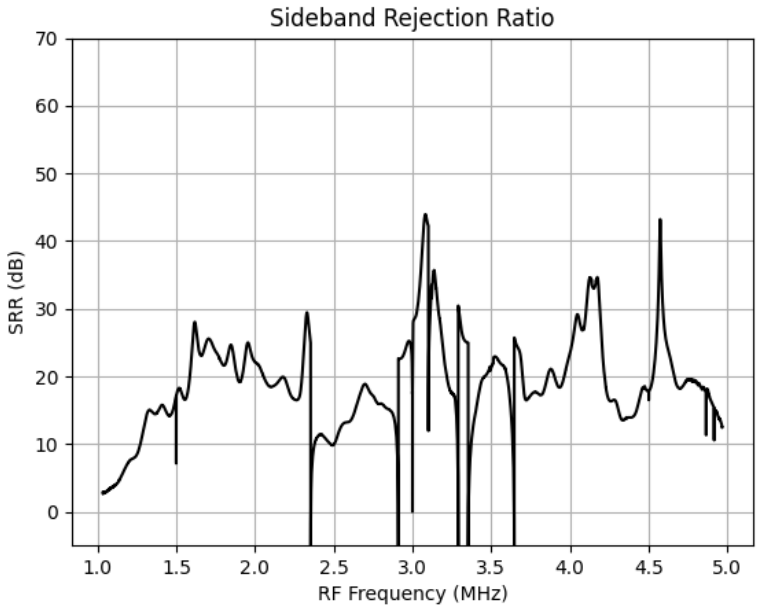


Figura 4.14: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{10} y cantidad de acumulaciones 2^{10}.

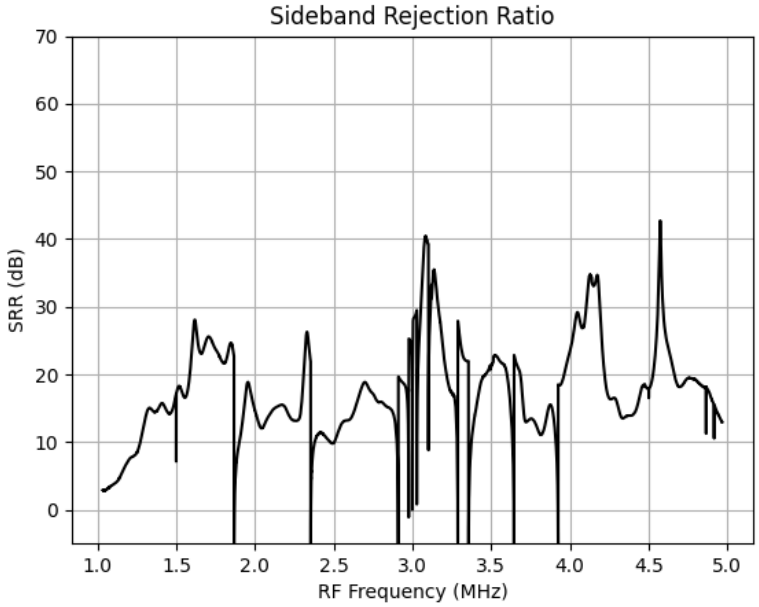


Figura 4.15: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{11} y cantidad de acumulaciones 2^{10}.

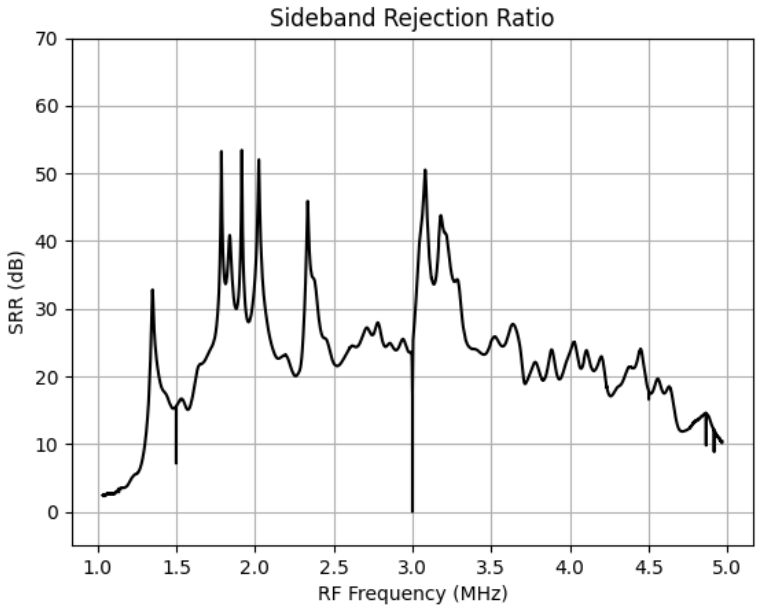


Figura 4.16: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{10} y cantidad de acumulaciones 2^9.

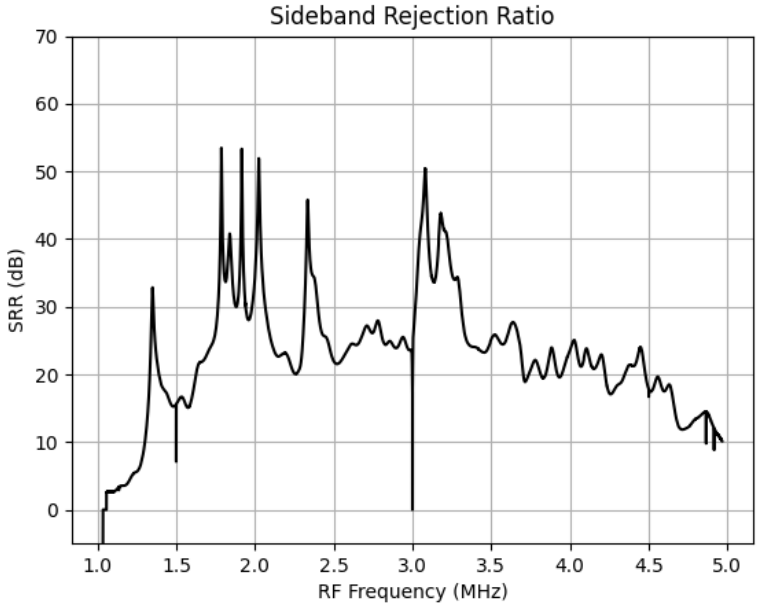


Figura 4.17: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{11} y cantidad de acumulaciones 2^8.

En el Anexo.C.3, se tienen más gráficos de SRR con otros parámetros de ganancia y largos de acumulación. En la Tabla 4.2 se tiene un resumen de las estadísticas de estas mediciones.

Tabla 4.2: Estadísticas del espectro medido con el modelo de 8192 canales, 32 bits, para distintas ganancias y largos de acumulación.

Gain	**Len**	**Average (dB)**	**Std Dev (dB)**	**Max (dB)**	**Min (dB)**
2^0	2^{10}	22.58	11.97	66.60	0.00
2^6	2^{10}	22.89	8.36	47.45	0.00
2^7	2^{10}	21.65	7.64	47.49	0.00
2^8	2^{10}	21.62	7.53	47.41	1.84
2^9	2^{10}	21.62	7.53	47.40	2.03
2^{10}	2^{10}	19.01	6.99	43.92	-14.17
2^{10}	2^9	22.78	8.83	53.43	2.31
2^{11}	2^{10}	17.66	6.99	42.72	-33.04
2^{11}	2^9	20.69	9.60	53.16	-19.96
2^{11}	2^8	22.76	8.87	53.47	-10.39
2^{12}	2^{10}	15.94	7.02	42.80	-22.00
2^{12}	2^7	22.78	8.83	53.30	2.31
2^{13}	2^{10}	13.03	7.56	42.77	-39.55
2^{13}	2^6	22.78	8.83	55.75	2.32
2^{20}	2^{10}	3.16	7.07	37.45	-40.86

Respecto al SRR del modelo de 32 bits, a medida que se incrementa la ganancia, se observa una mejora en la curva obtenida, lo que refleja el correcto funcionamiento del híbrido digital del espectrómetro. No obstante, también se evidencia saturación para ganancias elevadas. Por ejemplo, en la Figura 4.14, correspondiente a una ganancia de 2^{10}-valor que previamente se había identificado como adecuado-se aprecia una degradación del SRR en ciertos rangos de frecuencia. Este efecto se debe al largo de acumulaciones utilizado, que provoca saturación del espectro en frecuencias específicas. Al reducir la cantidad de acumulaciones a 2^9, se recupera una curva suave del SRR (Figura 4.16), manteniéndose un promedio superior a 20 dB.

Sin embargo, cabe destacar que esta saturación está asociada a la potencia de los tonos inyectados durante las pruebas. En una implementación real, es posible operar con ganancias cercanas o incluso superiores a 2^{10}, dado que las señales provenientes de fuentes radioastronómicas poseen niveles de potencia considerablemente más bajos, lo que evita alcanzar los límites de saturación del sistema.

4.1.3. Medición de la diferencia de fase en las señales I y Q

La medición de la diferencia de fase entre las señales I y Q se realizó utilizando los bloques del correlador implementado. La lectura se tiene en BRAM de 64 bits. Dado que solo se desarrollaron correladores con esta arquitectura, el análisis se enfocó en tres resoluciones espectrales distintas: 2048, 8192 y 16384 canales espectrales.

A continuación, en la Figura 4.18, se muestra el resultado de la medición para el modelo con 8192 canales. En el gráfico se observa la diferencia de fase a lo largo de toda la banda procesada.

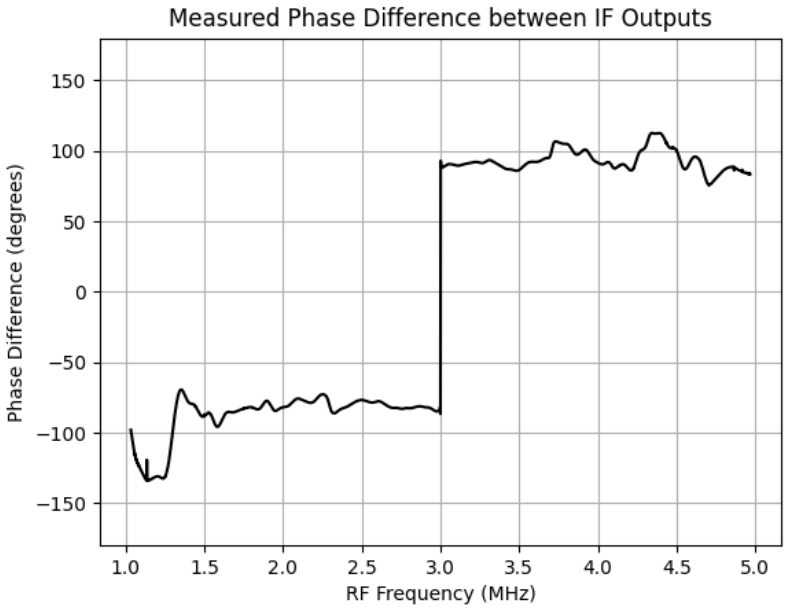


Figura 4.18: Diferencia de fase entre las señales I y Q para el modelo de 8192 canales espectrales.

Los resultados correspondientes a los modelos de 2048 y 16384 canales espectrales, que presentan un comportamiento similar, se incluyen en el Anexo.C.4. En la Tabla 4.3, se tienen los datos estadísticos de la medición de diferencia de fases.

Tabla 4.3: Estadísticas de la diferencia de fase entre señales I y Q en las bandas LSB y USB para modelos de 64 bits.

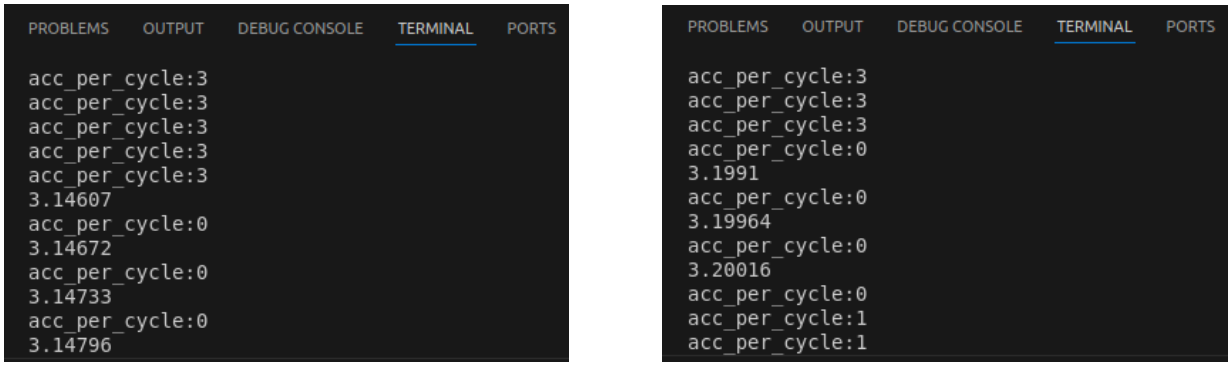
Canales	**Banda**	**Promedio (º)**	**Desv. Est. (º)**	**Máx (º)**	**Mín (º)**
2048	LSB	-87.42	16.09	-66.39	-134.08
2048	USB	92.49	13.02	112.62	-81.44
8192	LSB	-87.40	16.05	-69.41	-133.93
8192	USB	93.11	7.69	112.65	75.94
16384	LSB	-87.08	15.81	-68.67	-133.50
16384	USB	93.35	7.66	113.05	76.36

Respecto a estos resultados, la diferencia de fase idealmente debe mantenerse cercana a 90º, por lo que se puede concluir que las señales I y Q están efectivamente en cuadratura en todo el rango de frecuencias, cuyo promedio es cercano a -90º en la banda LSB y +90º en la banda USB. Desviaciones significativas podrían atribuirse a desbalances en la cadena analógica del *front end*, como así también se está trabajando con ciertas frecuencias fuera del rango óptimo para algunos componentes.

4.2. Desempeño de la señal de *RESET* en el laboratorio

4.2.1. Verificación del periodo de la señal

El objetivo de esta prueba fue verificar que los flancos de bajada de la señal de *RESET* fueran detectados correctamente por la lógica implementada en el RFSoC. Para ello, se utilizó un generador de ondas configurado para emitir una señal periódica con distintos periodos de referencia. El bloque `acc_per_cycle`, implementado dentro del espectrómetro, genera una marca (valor cero) cada vez que se detecta un flanco descendente de la señal de *RESET*. Midiendo el tiempo entre dos marcas consecutivas, es posible verificar si la señal se detecta con la periodicidad configurada.



(a) Primer reinicio

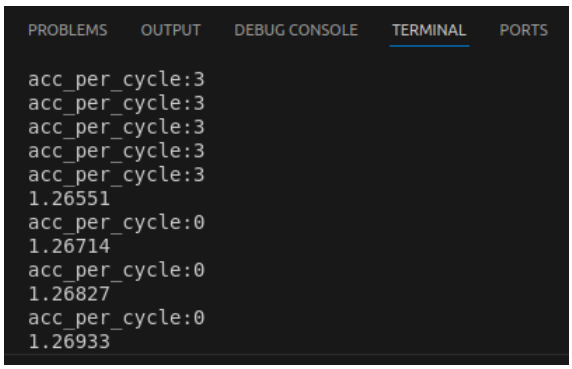
(b) Segundo reinicio

Figura 4.19: Medición del tiempo entre flancos de bajada del *RESET* para un periodo de 53 ms.

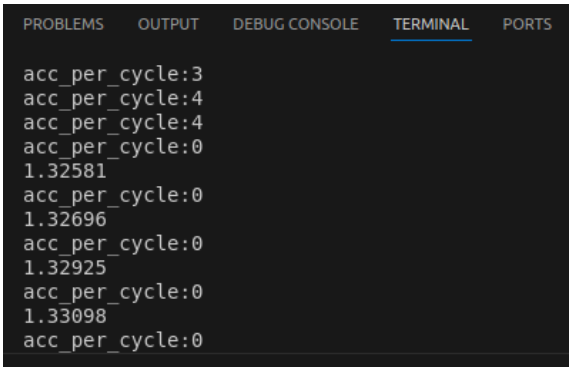
En la Figura 4.19 se muestra el resultado de una prueba en la que el generador de señales fue configurado con un periodo de 53 ms. La diferencia temporal entre los dos eventos de reinicio fue:

$$\Delta t = 3.1991 \text{ s} - 3.14607 \text{ s} = 0.05303 \text{ s} = 53.03 \text{ ms} \tag{4.1}$$

Este resultado demuestra que la señal de *RESET* fue correctamente recibida por la RFSoC y reconocida con la periodicidad esperada.



(a) Primer reinicio



(b) Segundo reinicio

Figura 4.20: Medición del tiempo entre flancos de bajada del *RESET* para un periodo de 60 ms.

De forma similar, en la Figura 4.20 se repitió la prueba con un periodo configurado de 60 ms. El resultado fue:

$$\Delta t = 1.32581 \text{ s} - 1.26551 \text{ s} = 0.0603 \text{ s} = 60.3 \text{ ms} \tag{4.2}$$

Al igual que en el caso anterior, el resultado confirma que la señal periódica fue correctamente interpretada por el sistema. En conclusión, se verifica que la PCB es capaz de transmitir la señal de *RESET* de forma estable y que la lógica en el RFSoC la detecta de manera confiable dentro del periodo especificado. Esto valida la funcionalidad del canal de sincronización en condiciones de laboratorio y garantiza su utilidad en condiciones reales de observación.

4.3. Mediciones de velocidad de lectura de espectros

4.3.1. Protocolo KATCP y lectura de datos con librería de CASPER en Python

El primer método consistió en utilizar la librería `casperfpga`, desarrollada por el grupo CASPER, la cual se comunica con el FPGA mediante el protocolo KATCP. Las mediciones de tiempo se realizaron leyendo únicamente una de las bandas (USB), comenzando con bloques de 512 canales y aumentando progresivamente hasta alcanzar los 8192 canales disponibles en el modelo. Para cada configuración, se efectuó una medición continua durante un periodo de una hora, registrando el tiempo requerido para adquirir cada espectro. En el Código B.5 se tiene la implementación donde se miden los tiempos de latencia con este método.

Los resultados de la lectura de datos se analizaron en función del tiempo de ejecución. Cabe destacar que los gráficos generados están delimitados por un umbral de 25 ms, ya que este representa el tiempo máximo permitido por el *Data Server* para adquirir un espectro sin generar conflictos en el modo de calibración del telescopio. Este límite está definido por el comportamiento de la señal *HOLD*, explicada en la Subsubsección 2.1.6.2 del Marco Teórico.

Los gráficos con la evolución temporal de estas mediciones se incluyen en el Anexo.C.5, donde se puede observar el comportamiento de la latencia para cada cantidad de canales

evaluada.

4.3.2. Servidor personalizado del RFSoC con C++, Socket programado en C++ y lectura de datos usando Python

El segundo método consistió en una implementación personalizada de cliente-servidor. El servidor fue programado en C++ y ejecutado directamente dentro del RFSoC, mientras que el cliente, desarrollado también en C++, se conectó mediante sockets TCP para solicitar los datos espectrales. El servidor y el cliente se programaron con los Códigos B.6 y B.7. Finalmente, se desarrolló una interfaz de lectura en Python que recibe los espectros que lee el socket y mide el tiempo que tarda en realizar la lectura de los espectros. Esta implementación está presente en el Código B.8.

Con este enfoque, se realizaron las mismas pruebas que con el primer método, comenzando con la lectura de 512 canales USB y aumentando hasta 2048. Al pedir más espectros la comunicación tiende a fallar debido a la cantidad de canales solicitados y el tamaño de los datos, por lo que las mediciones se detienen, como se pueden ver en las Figuras C.21 y C.30. Adicionalmente, con este método se hicieron más pruebas de lectura donde se leyeron ambas bandas laterales (USB y LSB).

Al igual que el método de lectura anterior, los gráficos con el tiempo de las mediciones se incluyen en el Anexo.C.6, donde se puede observar el comportamiento de la latencia para cada cantidad de canales evaluada.

4.3.3. Resumen con los tiempos de lectura

A continuación, en la Tabla 4.4, se presentan los tiempos de lectura obtenidos para ambos métodos:

Tabla 4.4: Tiempos de lectura de espectros desde la RFSoC utilizando distintos métodos, número de canales y cantidad de bandas leídas.

Método	**Configuración**	**Promedio (ms)**	**Desv. Est. (ms)**	**Máx (ms)**	**Mín (ms)**
Python + KATCP	512 canales, 1 banda	16.35	2.74	223.53	10.99
Python + KATCP	1024 canales, 1 banda	17.03	2.99	225.94	11.33
Python + KATCP	2048 canales, 1 banda	19.43	2.32	226.93	12.30
Python + KATCP	4096 canales, 1 banda	24.73	3.32	235.58	14.25
Python + KATCP	8192 canales, 1 banda	32.64	41.70	10556.30	19.22
Python + C++ Server	512 canales, 1 banda	10.68	0.55	67.22	9.61
Python + C++ Server	1024 canales, 1 banda	10.91	1.34	216.52	9.78
Python + C++ Server	2048 canales, 1 banda	11.39	1.24	218.80	10.16
Python + C++ Server	512 canales, 2 bandas	20.95	1.61	231.40	19.29
Python + C++ Server	1024 canales, 2 bandas	17.03	2.99	225.94	11.33
Python + C++ Server	2048 canales, 2 bandas	22.39	1.76	232.17	20.39

4.3.4. Resumen de lecturas fallidas (> 25 ms)

A continuación, en la Tabla 4.5, se presenta un resumen de las lecturas que superaron el umbral de 25 ms. Este umbral corresponde al límite de latencia máxima permitido por el *Data Server* en el modo de calibración. Se indican tanto la cantidad total de fallos como el porcentaje que representan respecto al total de muestras, así como el tiempo en que ocurrió el primer fallo.

Tabla 4.5: Frecuencia de lecturas fallidas (> 25 ms) para distintas configuraciones. Se incluye el tiempo en que se detectó el primer fallo.

Método	**Configuración**	**Fallos (>25 ms)**	**% Fallos**	**1er fallo (min)**
Python + KATCP	512 canales, 1 banda	14 / 98483	0.014 %	2.999
Python + KATCP	1024 canales, 1 banda	47 / 96659	0.049 %	1.517
Python + KATCP	2048 canales, 1 banda	1064 / 90866	1.171 %	0.003
Python + KATCP	4096 canales, 1 banda	29862 / 80136	37.264 %	0.001
Python + KATCP	8192 canales, 1 banda	38620 / 68097	56.713 %	0.001
Python + C++ Server	512 canales, 1 banda	1 / 116463	0.0009 %	16.619
Python + C++ Server	1024 canales, 1 banda	5 / 115629	0.0043 %	9.025
Python + C++ Server	2048 canales, 1 banda	3 / 113864	0.0026 %	7.503
Python + C++ Server	512 canales, 2 bandas	11 / 87358	0.0126 %	0.690
Python + C++ Server	1024 canales, 2 bandas	13 / 86386	0.0150 %	2.897
Python + C++ Server	2048 canales, 2 bandas	109 / 84393	0.1292 %	3.007

4.3.5. Conclusión de los resultados de velocidad de lectura

A partir del análisis de desempeño temporal realizado para ambos métodos de adquisición de espectros, se concluye que la implementación personalizada del sistema cliente-servidor en C++ ofrece resultados superiores en términos de latencia y estabilidad. Como se observa en la Tabla 4.4, este método presenta tiempos promedio de lectura considerablemente más bajos que el enfoque basado en la librería `casperfpga` usando el protocolo KATCP. Además, según la Tabla 4.5, el porcentaje de fallos -definidos como lecturas que exceden los 25 ms de latencia- es significativamente menor en el sistema personalizado, incluso para configuraciones de hasta 2048 canales por banda.

Estas diferencias son especialmente relevantes en contextos de operación sensibles como el modo de calibración del radiotelescopio, donde los márgenes de tiempo son reducidos y los fallos en la lectura pueden detener la adquisición de datos. En contraste, el modo de observación resulta más tolerante a fallos esporádicos, gracias a sus ciclos de integración más largos.

Por lo tanto, se ha decidido que la implementación definitiva en el radiotelescopio utilizará el método basado en el servidor C++ con interfaz en Python. Esta elección garantiza una adquisición de espectros más confiable y eficiente dentro de los límites de tiempo exigidos por el sistema de control del telescopio, maximizando la compatibilidad y estabilidad operativa.

4.4. Mediciones en el radiotelescopio

4.4.1. Evaluación del largo de acumulaciones según la señal *RESET*

Durante la evaluación del sistema de control de acumulaciones, se registró la cantidad máxima de acumulaciones que podían completarse entre dos pulsos consecutivos de la señal *RESET*, tanto en modo *SPLOBS* (observación) como en modo *CAL* (calibración). Para ello, se configuró el Generador de Referencia para emitir pulsos periódicos con un periodo de 53 ms en modo observación y de 60 ms en modo calibración. En cada caso, se probaron distintos largos de acumulación, incrementando el valor en potencias de 2, y se verificó cuántas acumulaciones completas se lograban ejecutar antes de la llegada del siguiente pulso de reinicio.

En la Tabla 4.6 se resumen los resultados obtenidos. Para cada largo de acumulación, se indica el número máximo de acumulaciones exitosas completadas dentro del intervalo temporal disponible. Cabe señalar que, en el caso particular de 2^{15}, se observó un comportamiento inestable: si bien en algunos ciclos se logra completar una acumulación, en muchos otros no se alcanza a finalizar ninguna antes del siguiente pulso de *RESET*, resultando en una lectura nula. Por esta razón, dicho valor se marca con un asterisco para resaltar su comportamiento inconsistente.

Tabla 4.6: Conteo máximo de acumulaciones completadas entre pulsos de *RESET* para distintos valores de acumulaciones, en modos observación y calibración.

Largo de Acumulaciones	**Observación (53 ms)**	**Calibración (60 ms)**
2^9	24	28
2^{10}	12	14
2^{11}	6	7
2^{12}	3	4
2^{13}	2	2
2^{14}	1	1
2^{15}	1*	1*

*En estos casos, el sistema no logra siempre obtener un espectro acumulado completo; en algunos ciclos se obtienen cero.

En este experimento, cuando se obtienen múltiples espectros completamente acumulados dentro de un mismo ciclo de *RESET*, el sistema descarta todos excepto uno. Se pudo verificar que, a medida que se incrementa el largo de acumulaciones por espectro, disminuye la cantidad de espectros válidos obtenidos por ciclo, hasta llegar a un punto en que no se completa ninguno, como ocurre con el valor 2^{15}.

Considerando estos resultados, se optó por utilizar un largo de acumulación de 2^{12} en la implementación final, ya que en el peor de los casos permite completar hasta cuatro espectros por ciclo, minimizando la pérdida de datos. En cambio, no se consideraron los valores 2^{13} ni 2^{14} para evitar una posible saturación en los espectros que recibe el PIC con un posible *overflow* en los datos.

4.4.2. Comunicación del nuevo espectrómetro con el sistema completo del radiotelescopio y visualización de espectros

Para verificar la comunicación entre PIC y RFSoC, primero se realizó un test de observación y otro de calibración usando la implementación actual en ROACH. Este último recibe los espectros de las frecuencias IF que se obtienen del *front end* del Mini. Los resultados que se muestran en las Figuras 4.21 y 4.22 son obtenidos del computador MAC que se comunica con el PIC.

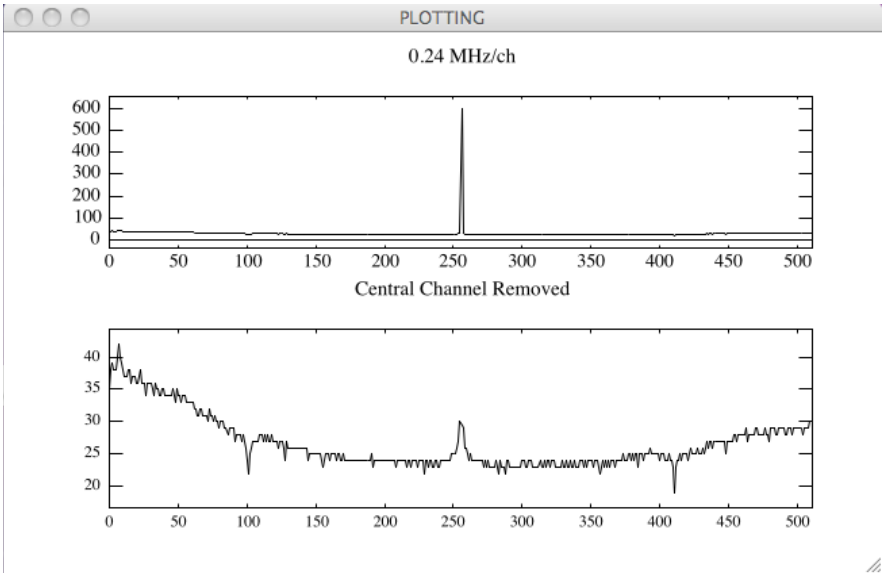


Figura 4.21: Espectro visualizado en el modo observación usando ROACH

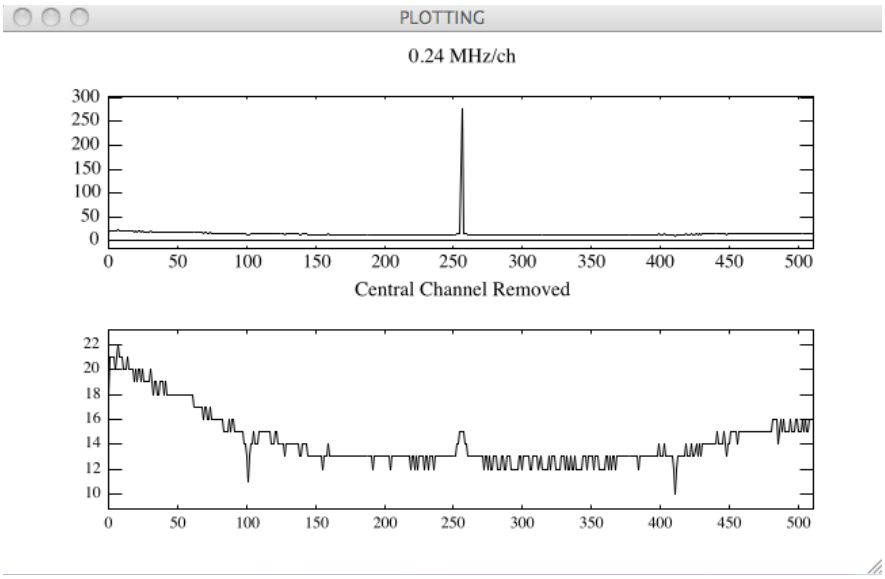


Figura 4.22: Espectro visualizado en el modo calibración usando ROACH

En estos gráficos se visualizan 512 canales espectrales cuyo espectro inicial corresponde al 768. La potencia de los tonos está en unidades de potencias arbitrarias, es decir, no están en decibeles. Considerando que el sistema en ROACH tiene un ancho de banda de 500 MHz, distribuido en un total de 2048 canales espectrales, estos gráficos muestran el espectro entre una banda entre 187.5 y 312.5 MHz en IF. Cabe resaltar que para cada modo se usa una ganancia y largo de acumulaciones distinto, lo que explica la diferencia en el eje vertical de los gráficos. Adicionalmente, estas figuras presentan una elevación en el canal central debido a que corresponde al número 1024, esto es producido por el ruido de los ADC del FPGA.

Posteriormente, usando un modelo de 64 bits, 8192 canales espectrales y un largo de acumulaciones de 2^{12}, con el Código B.1 se visualizó la IF con RFSoC, obteniéndose los

espectros de la Figura 4.23:

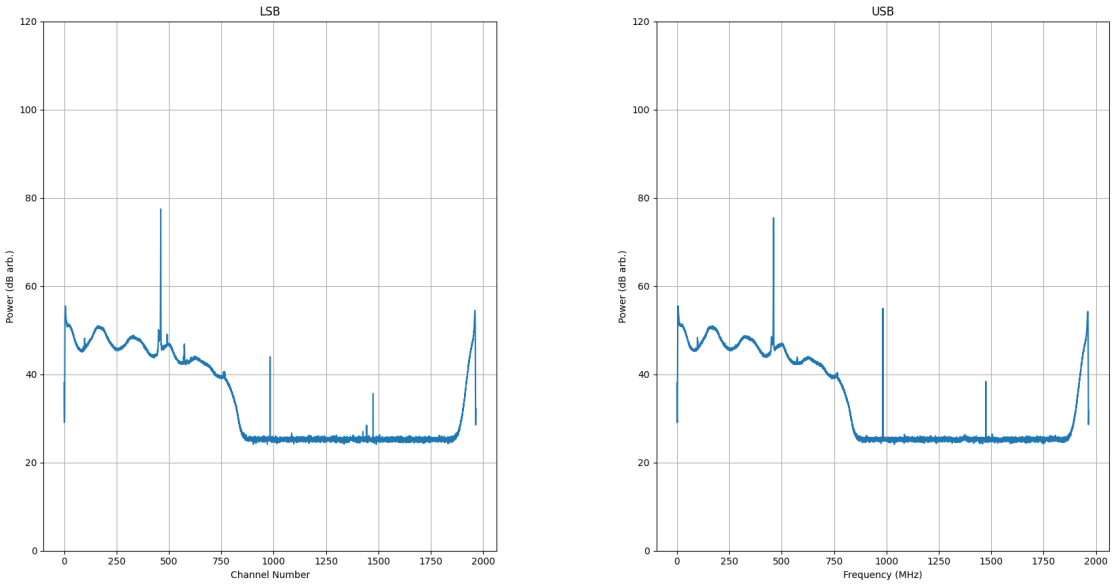


Figura 4.23: Espectro de las bandas LSB y USB con modelo de 64 bits en RFSoC.

Debido al diseño del *front end* del Mini y los filtros que tiene, de la IF solo predominan los espectros con frecuencias menores a 800 MHz.

Luego, usando el modelo final 32 bits compatible con la señal del *RESET*, 8192 canales y un largo de acumulaciones de 2^{12}, se visualizó nuevamente la IF con RFSoC. Usando ganancias de 2^{12} y 2^{13}, se obtienen los espectros de las Figuras 4.24 y 4.24, respectivamente.

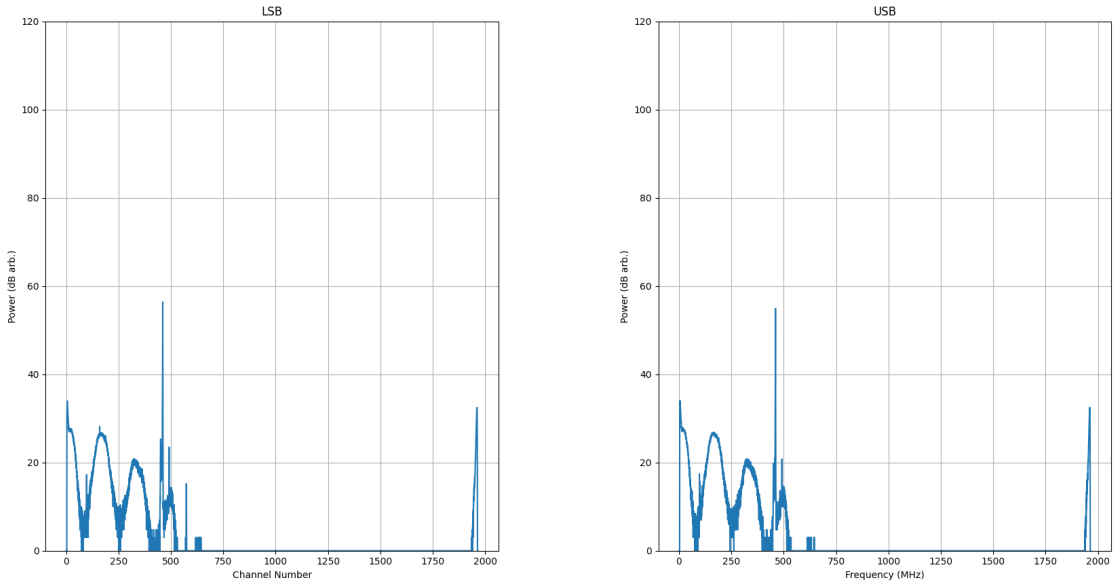


Figura 4.24: Espectro de las bandas LSB y USB con modelo de 32 bits en RFSoC. Se usó una ganancia de 2^{12}.

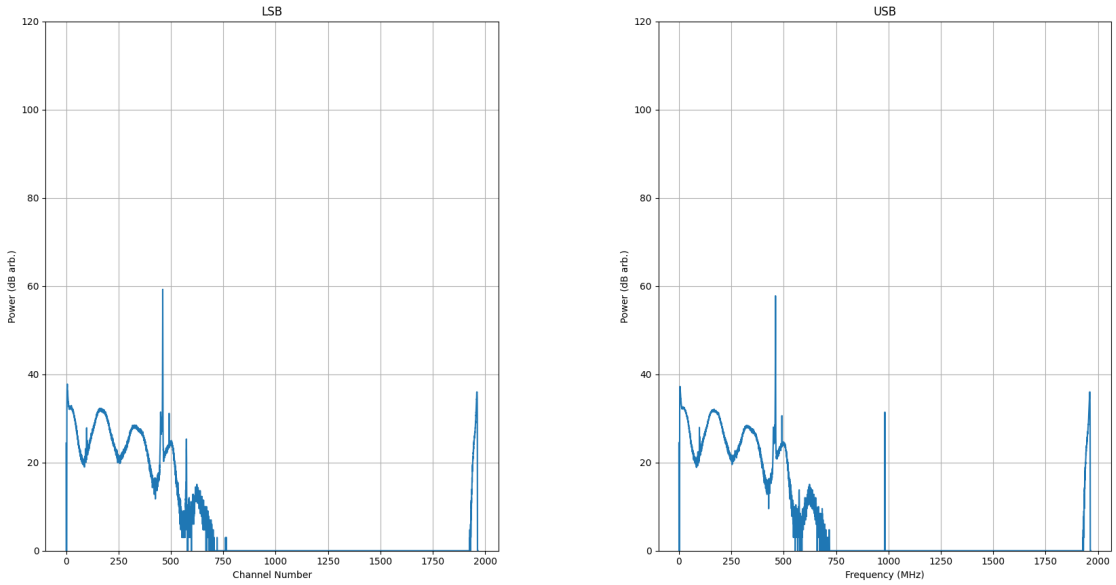


Figura 4.25: Espectro de las bandas LSB y USB con modelo de 32 bits en RFSoC. Se usó una ganancia de 2^{13}.

A partir de los gráficos se usó una ganancia de 2^{12} ya que es suficiente para que se vea el piso del ruido del espectro. Se descartó la ganancia 2^{13} para evitar un posible *overflow*. Cabe resaltar que, si bien el microcontrolador PIC32 funciona con datos de hasta 32 bits, actualmente está programado para que reciba números binarios con complemento de 2, por lo que su rango numérico en decimal va desde -2^{16} hasta $2^{16} - 1$.

Leyendo 512 canales, a partir del canal 768, en este caso sin decibeles, se graficaron los espectros de la banda USB tanto en observación, como en calibración, obteniéndose las Figuras 4.26 y 4.27, respectivamente.

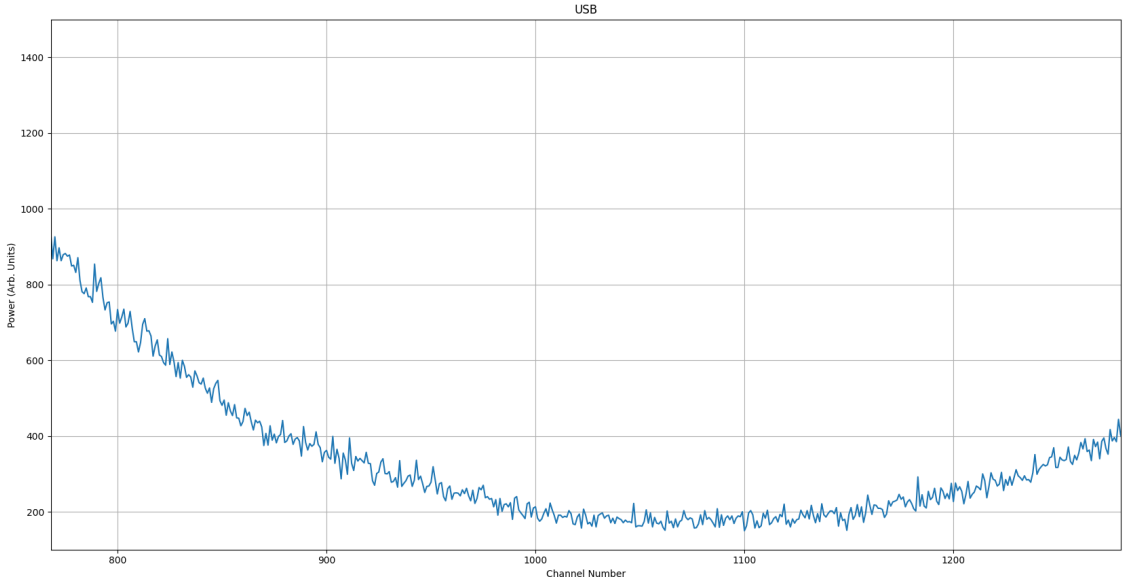


Figura 4.26: Espectro de la banda USB en modo observación. Se presentan 512 canales espectrales, tomando como punto de partida el canal 768.

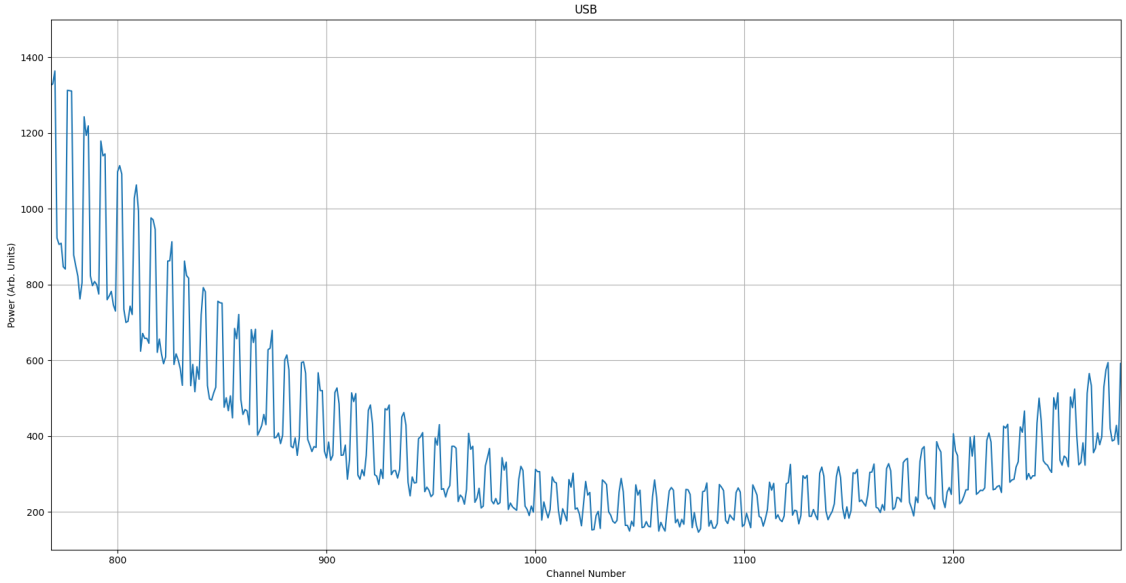


Figura 4.27: Espectro de la banda USB en modo calibración. Se presentan 512 canales espectrales, tomando como punto de partida el canal 768.

Finalmente, utilizando el servidor ejecutado en el RFSoC —presentado en el Código B.6— se estableció comunicación desde el *Data Server* mediante el sistema descrito en el Código B.9, el cual se utilizó como interfaz para conectarse servidor al interno del RFSoC. Posteriormente, desde el computador MAC se solicitaron espectros de la banda USB, tanto en modo observación como en modo calibración, utilizando el PIC como intermediario.

La solicitud de espectros desde el MAC se realiza mediante comandos previamente implementados en el sistema del radiotelescopio. En particular, se utilizaron los comandos EXEC SPTST para el modo de observación y EXEC CALTST para el modo de calibración. Estos comandos ajustan el sistema de control del radiotelescopio según las condiciones descritas en la Subsubsección 2.1.6.2, y durante un minuto solicitan espectros de forma continua, esperando que no se produzca ninguna falla en la transmisión o adquisición. En caso de una falla la prueba se detiene.

Durante la ejecución de estas pruebas, también es posible solicitar una captura del espectro en tiempo real mediante el comando DCH desde el computador MAC. Para cada modo se obtuvieron los espectros mostrados en las Figuras 4.28 y 4.29. En ambas figuras se visualiza la misma curva dos veces, ya que el PIC estaba configurado para leer únicamente una banda —en este caso, la USB— y duplicaba la salida para compatibilidad con los sistemas existentes.

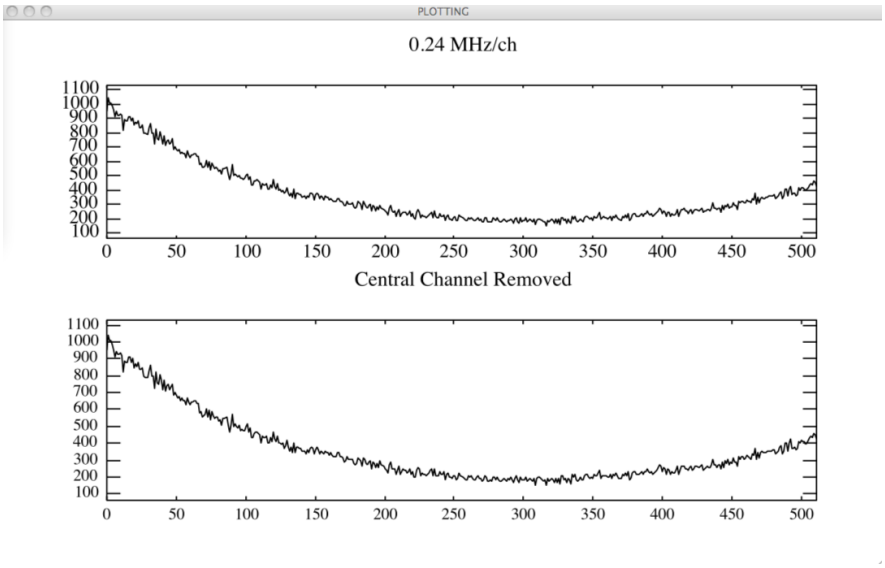


Figura 4.28: Espectro visualizado desde el computador MAC en el modo observación usando RFSoC.

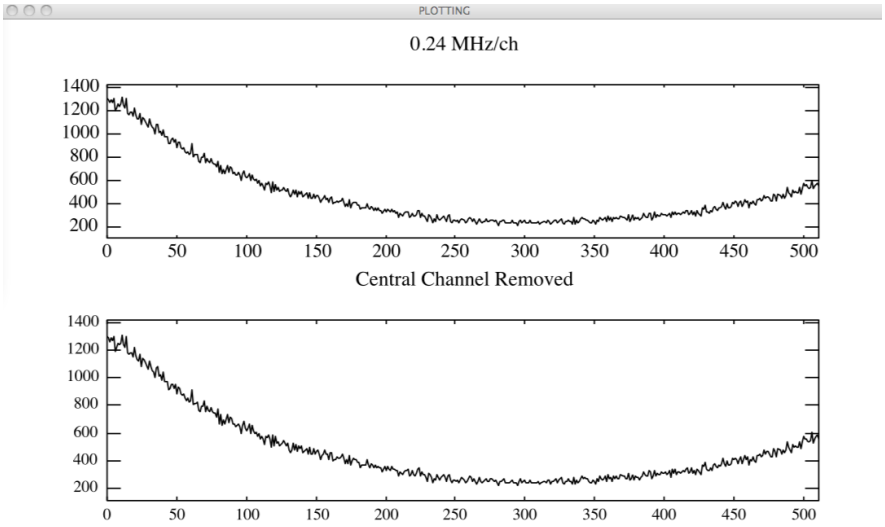


Figura 4.29: Espectro visualizado desde el computador MAC en el modo calibración usando RFSoC.

Comparando la Figura 4.28 con la Figura 4.21, y la Figura 4.29 con la Figura 4.22, se observa que las curvas obtenidas presentan formas similares, lo que indica que el sistema implementado en la RFSoC es capaz de comunicarse correctamente y replicar el comportamiento esperado. No obstante, existen diferencias en los órdenes de magnitud entre ambas plataformas, lo cual puede atribuirse a variaciones en los parámetros de operación, tales como el largo de acumulaciones o el valor de ganancia, así como a posibles diferencias en las características de los dispositivos FPGA utilizados durante las mediciones.

Sin embargo, estos resultados son la primera luz de una implementación del espectrómetro diseñado con el nuevo dispositivo RFSoC 4x2 al haber comunicación con el sistema completo del Radiotelescopio Mini.

4.4.3. Prueba de estabilidad en modo calibración y modo de observación

Finalmente, con el objetivo de verificar la estabilidad operativa del sistema en condiciones similares a una sesión real de observación astronómica, se realizaron múltiples pruebas ejecutando los comandos EXEC SPTST (modo observación) y EXEC CALTST (modo calibración) desde el computador MAC. Como se mencionó anteriormente, estos comandos solicitan una lectura continua de espectros durante aproximadamente un minuto, por lo que se ejecutaron diez veces por configuración.

En la Figura 4.30, se tiene una imagen de la interfaz en el MAC donde se monitorean características de operación del Mini. En esta imagen se tiene encerrado en un recuadro en rojo el tiempo transcurrido en una medición de prueba.

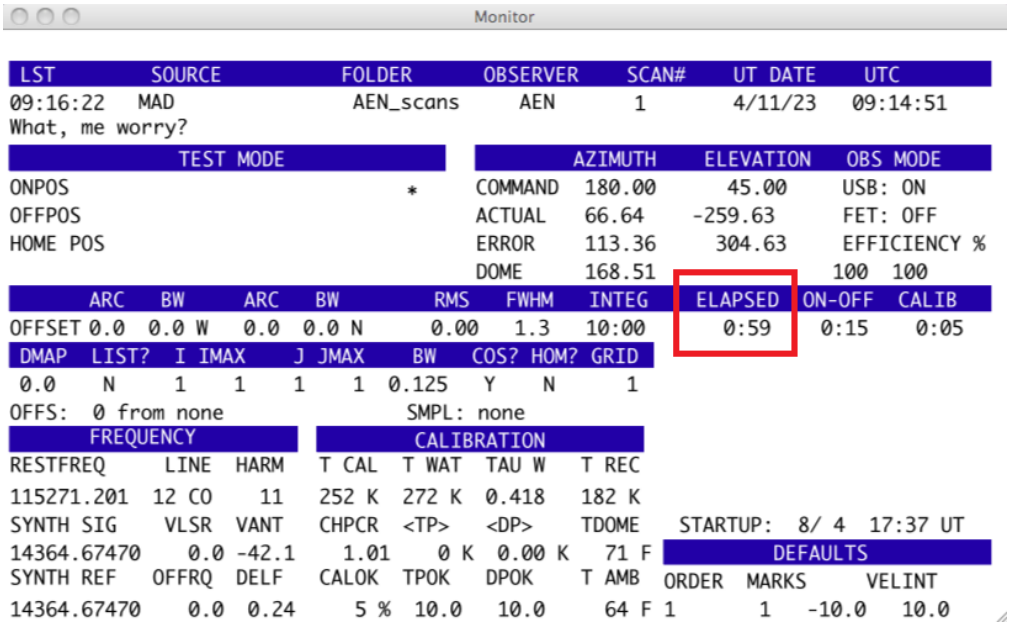


Figura 4.30: interfaz del MAC donde se monitorean características de operación del Radiotelescopio Mini.

Estas pruebas se llevaron a cabo utilizando el sistema cliente-servidor programado en C++ del RFSoC, cuya lectura se efectuó mediante una interfaz en Python. El Código B.9 se utilizó como interfaz en esta ocasión, y se evaluó el desempeño del sistema variando el número de canales leídos (512 a 2048), considerando tanto una como dos bandas. De estas lecturas, se enviaron 512 canales al PIC.

En la Tabla 4.7, se tiene un sumario con las pruebas llevadas a cabo.

Tabla 4.7: Resumen de pruebas de estabilidad del sistema en distintas configuraciones de lectura de espectros.

Modo	**Bandas**	**Canales**	**Resultado**	**Comentarios**
Calibración	1	512	8/10	Fallos en los segundos 0:00 y 0:45
Calibración	1	1024	8/10	Fallos en los segundos 0:50 y 0:34
Calibración	1	2048	9/10	Fallo en el segundo 0:25
Observación	1	512	10/10	Sin fallos
Observación	1	1024	10/10	Sin fallos
Observación	1	2048	10/10	Sin fallos
Calibración	2	512	0/10	Fallos en todos los intentos
Calibración	2	1024	0/10	Fallos en todos los intentos
Calibración	2	2048	0/10	Fallos en todos los intentos
Observación	2	512	10/10	Sin fallos
Observación	2	1024	10/10	Sin fallos
Observación	2	2048	10/10	Sin fallos

4.4.3.1. Lectura de una banda

En los experimentos con el modo calibración, leyendo una banda, el sistema fue generalmente estable para 512, 1024 y 2048 canales, aunque presentó fallos puntuales en algunos intentos. Por ejemplo, con 512 canales, se observaron fallos en los segundos 0:00 y 0:45 de dos de las diez pruebas (8/10); con 1024 y 2048 canales también se registraron fallos aislados.

Por otro lado, para el modo observación se pudo comprobar que el sistema fue completamente estable para 512, 1024 y 2048 canales, superando las diez pruebas sin errores en cada configuración.

4.4.3.2. Lectura de dos bandas

A diferencia del caso anterior, con el sistema en modo calibración, esta configuración resultó inestable incluso para 512 canales. En los tres tamaños evaluados (512, 1024 y 2048 canales), el sistema falló en los diez intentos, sin lograr una sola prueba exitosa (0/10). Esto sugiere que la doble carga de datos por ciclo supera los márgenes temporales del modo calibración.

Sin embargo, para el modo observación leyendo dos bandas, aquí el sistema nuevamente mostró un comportamiento robusto hasta los 2048 canales por banda. Las diez pruebas se completaron exitosamente para 512, 1024 y 2048 canales (10/10 en todos los casos).

4.4.3.3. Conclusión de los resultados de pruebas de estabilidad

Estos resultados indican que el sistema es estable y funcional hasta un máximo de 2048 canales por banda, ya sea en modo de observación o calibración (siempre que sea una sola banda). Las configuraciones con dos bandas simultáneas son viables únicamente en modo observación.

Para el modo calibración, una posible solución a esto es ajustar los tiempos de medición en la solicitud espectros en el Código B.9, ya que en este código se presentan unos *delays* con la función TIME.SLEEP(). Estos *delays* están implementados para que el sistema de comunicación no se sature y pueda solicitar los espectros al servidor sin problemas. Por lo tanto, un trabajo a futuro sería realizar nuevamente las pruebas de estabilidad en el Mini escogiendo un *delay* que mejore la velocidad con la que se piden espectros, pero que tampoco comprometa el funcionamiento de la comunicación con el servidor del FPGA.

Capítulo 5

Conclusiones

Este trabajo logró desarrollar e implementar un espectrómetro digital funcional para el Radiotelescopio Mini, el cual permite la separación de bandas laterales mediante un híbrido digital implementado en un FPGA RFSoC 4x2. El sistema diseñado fue validado tanto en laboratorio como en pruebas en el radiotelescopio, y logró cumplir con los objetivos propuestos en cuanto a resolución espectral, lectura de datos y compatibilidad con el sistema de control del telescopio.

Las pruebas realizadas en laboratorio demostraron que el sistema diseñado es capaz de separar con éxito las bandas laterales, alcanzando valores de SRR superiores a 20 dB con las configuraciones adecuadas de ganancia y acumulación, incluso sin aplicar calibración. Esto confirma la eficacia del híbrido digital implementado y su potencial para mejorar la calidad de las observaciones radioastronómicas.

El espectrómetro diseñado es compatible con las señales de control *HOLD* y *RESET*, logro que permitió operar en modos de observación y calibración de forma sincronizada. Asimismo, se logró establecer comunicación entre el espectrómetro y el sistema de control del telescopio, permitiendo la adquisición de los datos espectrales. Sin embargo, esta comunicación no es perfecta ya que presenta cierto índices de falla, pero que a lo largo del proyecto fueron mejorando al explorar distintos métodos de adquisición de datos, por ejemplo usar C++ en lugar de Python para la solicitud de datos espectrales.

El desarrollo de este proyecto demuestra la viabilidad de los espectrómetros digitales en radioastronomía y destaca la importancia de la integración de tecnologías modernas en la mejora de las capacidades de observación del universo.

Desde una perspectiva formativa, este proyecto representó una instancia fundamental de desarrollo profesional. La experiencia permitió aplicar conocimientos relacionados con procesamiento digital de señales, diseño en FPGA y protocolos de comunicación, enfrentando problemas reales en un entorno multidisciplinario. Además, se adquirieron competencias clave en validación experimental, integración de sistemas embebidos y trabajo colaborativo con un equipo científico. La implementación de este sistema no solo aporta soluciones técnicas concretas, sino que también fortalece mi preparación como ingeniero con capacidades en instrumentación astronómica y electrónica de alta frecuencia.

A nivel institucional, el trabajo desarrollado aporta directamente al proyecto QUIMAL que financió esta iniciativa, cumpliendo con sus objetivos de modernizar la infraestructura instrumental del Departamento de Astronomía. La incorporación del nuevo espectrómetro eleva significativamente las capacidades de observación del Southern Mini, permitiendo futuras investigaciones en radioastronomía de forma más precisa y eficiente. Asimismo, este desarrollo fortalece la docencia práctica al ofrecer a estudiantes y académicos una herramienta moderna para la enseñanza de instrumentación astronómica.

5.1. Trabajo Futuro

Tras la finalización del trabajo relacionado a este proyecto, se puede concluir que se cumplieron los objetivos planteados. Aún así, se plantean las siguientes tareas que se pueden realizar para mejorar la implementación:

- Implementar un espectrómetro calibrado utilizando el correlador diseñado en el proyecto para mejorar la precisión en la separación de bandas laterales.
- Aumentar el número de canales espectrales del espectrómetro implementado en el radiotelescopio para obtener resoluciones más precisas.
- Mejorar la adquisición de los datos para que pueda hacer una lectura completa de los espectros en la IF.
- Se podría evaluar el desempeño del sistema en un rango más amplio de frecuencias y explorar su aplicación en otros radiotelescopios.

Bibliografía

[1] Bronfman, L., Cohen, R. S., Alvarez, H., May, J., y Thaddeus, P., "A CO Survey of the Southern Milky Way: The Mean Radial Distribution of Molecular Clouds within the Solar Circle", , vol. 324, p. 248, 1988, doi:10.1086/165892.

[2] Bronfman, L., Alvarez, H., Cohen, R. S., y Thaddeus, P., "A Deep CO Survey of Molecular Clouds in the Southern Milky Way", , vol. 71, p. 481, 1989, doi:10.1086/191384.

[3] DAS, "Radio Telescopio MINI", 2022, https://ingenieria.uchile.cl/investigacion/laboratorios/departamento-de-astronomia/radio-telescopio-mini.

[4] Aldami, H. A. N. y Karim, H., GPR DATA SIMULATION FOR ENGINEERING INVESTIGATIONS IN SHALLOW REGIONS. Tesis PhD, 01 2011.

[5] NRAO, "Chapter 1: Introduction", 2018, https://www.cv.nrao.edu/~sransom/web/Ch1.html.

[6] Cassanelli, T., "Chapter I: Introduction to radio astronomy", 2024. Apuntes de clase, curso EL6053: *Radioastronomia: aplicaciones, herramientas e impacto*, Universidad de Chile.

[7] NRAO, "Chapter 7: Spectral Lines", 2018, https://www.cv.nrao.edu/~sransom/web/Ch7.html.

[8] Price, D. C., "Spectrometers and polyphase filterbanks in radio astronomy", 2018, https://arxiv.org/abs/1607.03579.

[9] Max-Moerbeck, W. K., "Implementación de un oscilador Gunn en un Receptor a 115GHz, para fines radiastronómicos". Memoria para optar al título de Ingeniero Civil Electricista, Universidad de Chile, 2005.

[10] Prieto, M., "Generador de referencia (RFG)". Reporte técnico interno del MWL, 2024.

[11] Finger, R., Design and construction of a digital side-band separating spectrometer for the 1.2-meter Southern Radio Telescope. Tesis PhD, Universidad de Chile, 2013.

[12] Cubillos, D., Desarrollo de un receptor en configuración 2SB para banda W extendida (67-116 GHz) con aplicaciones en el telescopio". Tesis PhD, Universidad de Chile, 2023.

[13] Pozar, D. M., Microwave Engineering. Wiley: John Wiley & Sons, 4th ed., 2012.

[14] Curotto, F., Finger, R., Kojima, T., Uemizu, K., González, Á., Uzawa, Y., y Bronfman, L., "Digital calibration test results for Atacama Large Millimeter/submillimeter Array band 7+8 sideband separating receiver", 2022, http://www.das.uchile.cl/lab_mwl/publicaciones/Articles/JATIS-21143G_online.pdf.

[15] NAND-LAND, "Lesson 1: What is an FPGA?", 2022, https://nandland.com/lesson-1-what-is-an-fpga/.

[16] Real Digital, “RFSoC 4x2 Reference Manual”., https://www.realdigital.org/hardware/rfsoc-4x2. Revision: A5.

[17] Rodríguez, R., Finger, R., Mena, F. P., Reyes, N., Michael, E., y Bronfman, L., “A Sideband-separating Receiver with a Calibrated Digital IF-Hybrid Spectrometer for the Millimeter Band”, 2014, http://www.das.uchile.cl/lab_mwl/publicaciones/Articles/paper_rafa.pdf.

[18] CASPER, “The Polyphase Filter Bank Technique”, 2016, https://casper.berkeley.edu/wiki/The_Polyphase_Filter_Bank_Technique.

[19] NRAO, “Chapter 3: Radio Telescopes and Radiometers”, 2018, https://www.cv.nrao.edu/~sransom/web/Ch3.html.

Anexos

Anexo A. Desarrollo Matemático de Separación de Bandas Laterales para la configuración 2SB

Un mezclador convencional genera dos frecuencias resultantes:

$$f_{IF} = |f_{RF} - f_{LO}| \tag{A.1}$$

Sin embargo, existen dos posibles señales de entrada en el puerto RF que pueden generar la misma frecuencia intermedia (IF):

$$f_{RF} = f_{LO} + f_{IF} \quad \text{(USB - Upper Sideband)} \tag{A.2}$$

$$f_{RF} = f_{LO} - f_{IF} \quad \text{(LSB - Lower Sideband)} \tag{A.3}$$

La configuración que se presenta en la Figura A.1, permite aislar estas bandas mediante el uso de dos híbridos de 90º y dos mezcladores balanceados.

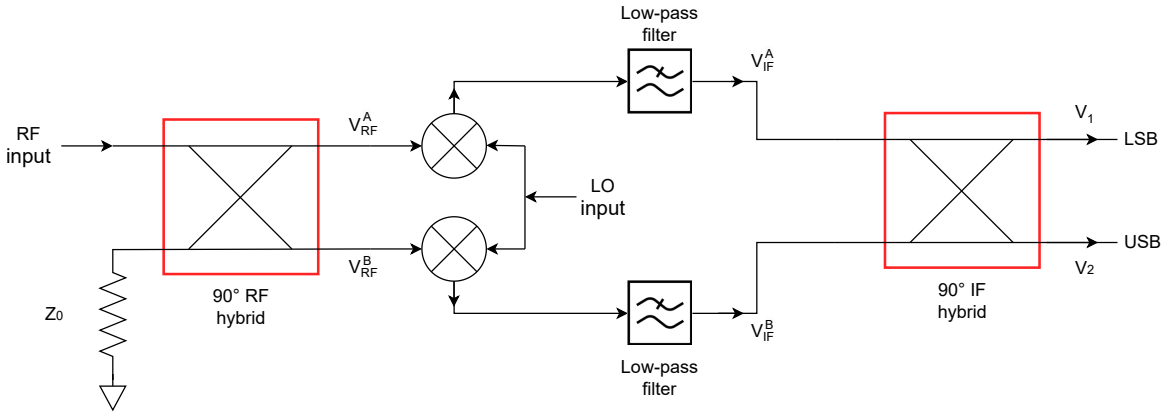


Figura A.1: Circuito de la configuración 2SB

Sea la señal de entrada RF en el dominio del tiempo:

$$v_{RF}(t) = V_U \cos(\omega_{LO} + \omega_{IF})t + V_L \cos(\omega_{LO} - \omega_{IF})t \tag{A.4}$$

Las señales de RF entrantes a los mezcladores, después del híbrido de 90º son:

$$v_{RF}^{A}(t) = \frac{1}{\sqrt{2}}[V_U \cos(\omega_{LO}t + \omega_{IF}t - 90^{\text{o}}) + V_L \cos(\omega_{LO}t - \omega_{IF}t - 90^{\text{o}})] \tag{A.5}$$

$$= \frac{1}{\sqrt{2}}[V_U \sin(\omega_{LO} + \omega_{IF})t + V_L \sin(\omega_{LO} - \omega_{IF})t], \tag{A.6}$$

$$v_{RF}^{B}(t) = \frac{1}{\sqrt{2}}[V_U \cos(\omega_{LO}t + \omega_{IF}t - 180^{\text{o}}) + V_L \cos(\omega_{LO}t - \omega_{IF}t - 180^{\text{o}})] \tag{A.7}$$

$$= -\frac{1}{\sqrt{2}}[V_U \cos(\omega_{LO} + \omega_{IF})t + V_L \cos(\omega_{LO} - \omega_{IF})t] \tag{A.8}$$

Después de la mezcla con la señal del oscilador local (LO) y filtrado pasa-bajo, las señales en la frecuencia intermedia (IF) son:

$$v_{IF}^{A}(t) = \frac{KV_{LO}}{2\sqrt{2}}(V_U - V_L)\sin(\omega_{IF}t), \tag{A.9}$$

$$v_{IF}^{B}(t) = -\frac{KV_{LO}}{2\sqrt{2}}(V_U + V_L)\cos(\omega_{IF}t), \tag{A.10}$$

donde, V_{LO} es la amplitud del voltaje del LO, y K corresponde a una constante relacionada a la ganancia del mezclador. Reescribiendo estas señales en su forma fasorial se tiene:

$$V_{IF}^{A} = -j\frac{KV_{LO}}{2\sqrt{2}}(V_U - V_L), \tag{A.11}$$

$$V_{IF}^{B} = -\frac{KV_{LO}}{2\sqrt{2}}(V_U + V_L), \tag{A.12}$$

Combinando estas señales en el híbrido IF, se obtienen las siguientes salidas:

$$V_1 = -j\frac{V_{IF}^{A}}{\sqrt{2}} - \frac{V_{IF}^{B}}{\sqrt{2}} = \frac{KV_{LO}V_L}{2} \quad \text{(LSB)}, \tag{A.13}$$

$$V_2 = -\frac{V_{IF}^{A}}{\sqrt{2}} - j\frac{V_{IF}^{B}}{\sqrt{2}} = -j\frac{KV_{LO}V_U}{2} \quad \text{(USB)}, \tag{A.14}$$

las cuales, en el dominio del tiempo, pueden ser representadas como

$$v_1(t) = \frac{KV_{LO}V_L}{2}\cos(\omega_{IF}t) \quad \text{(LSB)}, \tag{A.15}$$

$$v_2(t) = \frac{KV_{LO}V_U}{2}\sin(\omega_{IF}t) \quad \text{(USB)}, \tag{A.16}$$

donde se observa que hay una separación clara entre las dos bandas laterales, debido a un desfase de 90º.

Anexo B. Códigos desarrollados

A continuación, se muestran los diferentes códigos que se desarrollaron para el proyecto.

Código B.1: Código que muestra en tiempo real los espectros de ambas bandas laterales: anim_dss_spectrum_1966mhz.py

```
import sys, time, struct
import numpy as np
from numpy import fft
import matplotlib.pyplot as plt
import matplotlib.animation as anim
import casperfpga
import argparse

def get_vacc_data_power(fpga, n_outputs, nfft, n_bits):
  """Get the raw data from fpga digital sideband separation spectrometer"""

  bins_out = nfft//n_outputs    # Number of bins for each output

  if n_bits == 64:    # Shared BRAMs data width of 8 bytes (64 bits)

    data_width = 8    # 8 bytes (64 bits)

    data_type = 'Q'   # Format character 'unsigned long long'

  else:             # Shared BRAMs data width of 4 bytes (32 bits)

    data_width = 4    # 4 bytes (32 bits)

    data_type = 'L'   # Format character 'unsigned long'


  add_width = bins_out    # Number of "Data Width" words of the implemented BRAM
                    # Must be set to store at least the number of output bins of each bram

  raw1 = np.zeros((n_outputs, bins_out))
  raw2 = np.zeros((n_outputs, bins_out))

  t = time.time()
  for i in range(n_outputs):    # Extract data from BRAMs blocks for each output

    raw1[i,:] = struct.unpack(f'>{bins_out}{data_type}',
    fpga.read(f'synth0_{i}', add_width * data_width, 0))
    # print(fpga.read(f'synth0_{i}', 256, 0))
    raw2[i,:] = struct.unpack(f'>{bins_out}{data_type}',
    fpga.read(f'synth1_{i}', add_width * data_width, 0))

  interleave_i = raw1.T.ravel().astype(np.float64)
  interleave_q = raw2.T.ravel().astype(np.float64)

  return interleave_i, interleave_q

def plot_spectrum(fpga, Nfft, n_bits):

    # fig, (ax1, ax2, ax3) = plt.subplots(3, 1)
```

```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.grid()
ax2.grid()
# ax3.grid()

print(Nfft)
fs = 3932.16 / 2
n_outputs = 8

faxis = np.linspace(0,fs, Nfft ,endpoint=False)

spectrum1, spectrum2 = get_vacc_data_power(fpga, n_outputs=n_outputs, nfft=Nfft,
 ↪ n_bits=n_bits)

line1, = ax1.plot(faxis, 10 * np.log10(fft.fftshift(spectrum2+1)), '-')
ax1.set_xlabel('Frequency (MHz)')
ax1.set_ylabel('Power (dB arb.)')
ax1.set_title('LSB')

# ax1.axvline(407.04, color = "red")
ax1.set_ylim([0, 120])

line2, = ax2.plot(faxis, 10 * np.log10(fft.fftshift(spectrum1+1)), '-')
ax2.set_xlabel('Frequency (MHz)')
ax2.set_ylabel('Power (dB arb.)')
ax2.set_title('USB')

# ax2.axvline(407.04, color = "red")
ax2.set_ylim([0, 120])

# line3, = ax3.plot(faxis, np.abs(10 * np.log10(fft.fftshift(spectrum1+1)/fft.fftshift(
 ↪ spectrum2+1))), '-')
# ax3.set_xlabel('Frequency (MHz)')
# ax3.set_ylabel('SRR (dB)')
# ax3.set_title('Sideband Rejection Ratio')

# # ax3.axvline((3-1.10712)*1000, color = "red")
# ax3.set_ylim([-10, 100])

def update(frame, *fargs):

    spectrum1, spectrum2 = get_vacc_data_power(fpga, n_outputs=n_outputs, nfft=
 ↪ Nfft, n_bits=n_bits)

    line1.set_ydata(10 * np.log10(fft.fftshift(spectrum2+1)))
    line2.set_ydata(10 * np.log10(fft.fftshift(spectrum1+1)))
    # print(spectrum2+1)
    # line3.set_ydata(np.abs(10 * np.log10(fft.fftshift(spectrum1+1)/fft.fftshift(spectrum2
 ↪ +1))))

v = anim.FuncAnimation(fig, update, frames=1, repeat=True, fargs=None, interval=10)
plt.tight_layout()
```

```
    plt.show()


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description='Shows real time sidebands spectrums and SRR with given options',
        usage='python anim_dss_spectrum_1966mhz.py <HOSTNAME_or_IP> <Nfft Size> <
     ↪ Data Output Width> [options]'
    )

    parser.add_argument('hostname', type=str, help='Hostname or IP for the Casper
     ↪ platform')
    parser.add_argument('nfft', type=int, help='Operation mode: Nfft Size')
    parser.add_argument('data_output_width', type=int, help='BRAMs data output width')

    parser.add_argument('-l', '--acc_len', type=int, default=2**10,
                        help='Set the number of vectors to accumulate between dumps. Default is
     ↪ 2*(2^28)/2048')
    parser.add_argument('-s', '--skip', action='store_true',
                        help='Skip programming and begin to plot data')
    parser.add_argument('-b', '--fpgfile', type=str, default='',
                        help='Specify the FPG file to load')

    args = parser.parse_args()

    hostname = args.hostname
    Nfft = args.nfft
    n_bits = args.data_output_width

    # Use your .fpg file
    bitstream = args.fpgfile if args.fpgfile else '/home/jose/Workspace/RFSoC-MINI-2SB-
     ↪ Receiver/8192ch_32bits_reset/dss_ideal_8192ch_32bits_reset_1966mhz_cx/outputs
     ↪ /dss_ideal_8192ch_32bits_reset_1966mhz_cx_2025-04-07_1901.fpg'

    print(f'Connecting to {hostname}...')
    fpga = casperfpga.CasperFpga(hostname)
    time.sleep(0.2)

    if not args.skip:
        print(f'Programming FPGA with {bitstream}...')
        fpga.upload_to_ram_and_program(bitstream)
        print('Done')
    else:
        fpga.get_system_information()
        print('Skip programming FPGA...')

    print('Initializing RFDC block...')
    fpga.adcs['rfdc'].init()
    c = fpga.adcs['rfdc'].show_clk_files()
    fpga.adcs['rfdc'].progpll('lmk', c[1])
    fpga.adcs['rfdc'].progpll('lmx', c[0])
    time.sleep(1)
```

```
145
146     print('Configuring accumulation period...')
147     fpga.write_int('acc_len', args.acc_len)
148
149     if n_bits == 32:
150         fpga.write_int('gain', 2**10)
151     time.sleep(1)
152     print('Done')
153
154     print('Resetting counters...')
155     fpga.write_int('cnt_rst', 1)
156     fpga.write_int('cnt_rst', 0)
157     time.sleep(1)
158     print('Done')
159     try:
160         plot_spectrum(fpga, Nfft, n_bits)
161     except KeyboardInterrupt:
162         sys.exit()
```

Código B.2: Barrido de frecuencias para el cálculo de SRR en todo el ancho de banda: sweep_srr_plot_1966mhz.py

```
1  import sys, time, struct
2  import numpy as np
3  from numpy import fft
4  import matplotlib.pyplot as plt
5  import casperfpga
6  import pyvisa
7  import time
8  import argparse
9  import csv
10
11 def get_vacc_data_power(fpga, n_outputs, nfft, n_bits):
12   """Get the raw data from fpga digital sideband separation spectrometer"""
13
14   bins_out = nfft//n_outputs    # Number of bins for each output
15
16   if n_bits == 64:    # Shared BRAMs data width of 8 bytes (64 bits)
17
18     data_width = 8    # 8 bytes (64 bits)
19
20     data_type = 'Q'   # Format character 'unsigned long long'
21
22   else:               # Shared BRAMs data width of 4 bytes (32 bits)
23
24     data_width = 4    # 4 bytes (32 bits)
25
26     data_type = 'L'   # Format character 'unsigned long'
27
28
29   add_width = bins_out    # Number of "Data Width" words of the implemented BRAM
```

```
                    # Must be set to store at least the number of output bins of each bram

  raw1 = np.zeros((n_outputs, bins_out))
  raw2 = np.zeros((n_outputs, bins_out))

  for i in range(n_outputs):    # Extract data from BRAMs blocks for each output
    # time.sleep(0.001)
    raw1[i,:] = struct.unpack(f'>{bins_out}{data_type}',
    fpga.read(f'synth0_{i}', add_width * data_width, 0))
    # time.sleep(0.001)
    raw2[i,:] = struct.unpack(f'>{bins_out}{data_type}',
    fpga.read(f'synth1_{i}', add_width * data_width, 0))

  interleave_i = raw1.T.ravel().astype(np.float64)
  interleave_q = raw2.T.ravel().astype(np.float64)

  return interleave_i, interleave_q

def sweep_SRR(fpga, instrument, Nfft, n_bits, bin_step, output_file='srr_data.csv'):
    '''Sweeps frequencies and plots SRR with given options, and saves data to CSV.'''

    fs = 3932.16 / 2  # Bandwidth
    LO = 3000  # Local Oscillator
    SRR = []  # Sideband Rejection Ratio
    freq_data = []

    try:
        n_outputs = 8
        if_freqs = np.linspace(0, fs, Nfft, endpoint=False)
        faxis_LSB = LO - if_freqs
        faxis_USB = LO + if_freqs

        for i in range(0, Nfft, bin_step):
            instrument.write(f'FREQ {faxis_LSB[-i-1]}e6')
            time.sleep(0.05)

            spectrum1, spectrum2 = get_vacc_data_power(fpga, n_outputs=n_outputs, nfft=
    ↪ Nfft, n_bits=n_bits)
            # time.sleep(0.1)
            diff = 10 * (np.log10(fft.fftshift(spectrum2 + 1)[-i-1] / fft.fftshift(spectrum1 + 1)[-i
    ↪ -1]))

            print(faxis_LSB[-i-1] / 1000, diff)
            freq_data.append(faxis_LSB[-i-1] / 1000)
            SRR.append(diff)

        for i in range(0, Nfft, bin_step):
            instrument.write(f'FREQ {faxis_USB[i]}e6')
            time.sleep(0.05)

            spectrum1, spectrum2 = get_vacc_data_power(fpga, n_outputs=n_outputs, nfft=
    ↪ Nfft, n_bits=n_bits)
```

```
            # time.sleep(0.1)
            diff = 10 * (np.log10(fft.fftshift(spectrum1 + 1)[i] / fft.fftshift(spectrum2 + 1)[i]))

            print(faxis_USB[i] / 1000, diff)
            freq_data.append(faxis_USB[i] / 1000)
            SRR.append(diff)

    except KeyboardInterrupt:
        print("User interruption. Plotting data...")

    # Save CSV
    with open(output_file, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(["Frequency (MHz)", "SRR (dB)"])
        csv_writer.writerows(zip(freq_data, SRR))

    faxis = np.concatenate((faxis_LSB[::-1][::bin_step], faxis_USB[::bin_step]))[:len(SRR)]

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.grid()
    ax.plot(faxis, SRR, '-', color="black")

    plt.xlabel('RF Frequency (MHz)')
    plt.ylabel('SRR (dB)')
    plt.title('Sideband Rejection Ratio')
    ax.set_ylim(-10, 70)
    plt.show()


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description='Sweeps frequencies and plots SRR with given options',
        usage='python sweep_srr_plot_1966mhz.py <HOSTNAME_or_IP> <Nfft Size> <RF
    ↪ Instrument IP address> <Data Output Width> [options]'
    )

    parser.add_argument('hostname', type=str, help='Hostname or IP for the Casper
    ↪ platform')
    parser.add_argument('nfft', type=int, help='Operation mode: Nfft Size')
    parser.add_argument('rf_instrument', type=str, help='RF instrument IP address')
    parser.add_argument('data_output_width', type=int, help='BRAMs data output width')

    parser.add_argument('-l', '--acc_len', type=int, default=2**10,
                        help='Set the number of vectors to accumulate between dumps')
    parser.add_argument('-s', '--skip', action='store_true',
                        help='Skip programming and begin to plot data')
    parser.add_argument('-b', '--fpgfile', type=str, default='',
                        help='Specify the FPG file to load')

    args = parser.parse_args()
```

```
    hostname = args.hostname
    Nfft = args.nfft
    rf_instrument = args.rf_instrument
    n_bits = args.data_output_width

    # Use your .fpg file
    bitstream = args.fpgfile if args.fpgfile else '/home/jose/Workspace/RFSoC-MINI-2SB-
     ↪ Receiver/16384ch_64bits/dss_ideal_16384ch_1966mhz_cx/outputs/
     ↪ dss_ideal_16384ch_1966mhz_cx_2025-04-07_1722.fpg'

    print(f'Connecting to {hostname}...')
    fpga = casperfpga.CasperFpga(hostname)
    time.sleep(0.2)

    if not args.skip:
        print(f'Programming FPGA with {bitstream}...')
        fpga.upload_to_ram_and_program(bitstream)
        print('Done')
    else:
        fpga.get_system_information()
        print('Skip programming FPGA...')

    print('Initializing RFDC block...')
    fpga.adcs['rfdc'].init()
    c = fpga.adcs['rfdc'].show_clk_files()
    fpga.adcs['rfdc'].progpll('lmk', c[1])
    fpga.adcs['rfdc'].progpll('lmx', c[0])
    time.sleep(1)
    print('Done')

    print('Configuring accumulation period...')
    fpga.write_int('acc_len', args.acc_len)

    if n_bits == 32:
       fpga.write_int('gain', 2**10)
    time.sleep(1)
    print('Done')

    print('Resetting counters...')
    fpga.write_int('cnt_rst', 1)
    fpga.write_int('cnt_rst', 0)
    time.sleep(1)
    print('Done')

    print('Connecting to instruments...')
    rm = pyvisa.ResourceManager('@py')
    instrument = rm.open_resource(f'TCPIP0::{rf_instrument}::INSTR')
    time.sleep(1)
    print('Done')

    try:
        sweep_SRR(fpga, instrument, Nfft, n_bits, 4)
```

```
    except KeyboardInterrupt:
        sys.exit()
```

Código B.3: Barrido de frecuencias para el cálculo de la diferencia de fase en todo el ancho de banda para las señales I y Q en la IF: *sweep_ph_plot_1966mhz.py*

```
import sys, time, struct
import numpy as np
from numpy import fft
import matplotlib.pyplot as plt
import casperfpga
import pyvisa
import argparse
import csv

def get_vacc_data_re_im(fpga, n_outputs, nfft, n_bits):
  """Get the raw data from fpga digital correlator"""

  bins_out = nfft//n_outputs    # Number of bins for each output

  if n_bits == 64:    # Shared BRAMs data width of 8 bytes (64 bits)

    data_width = 8    # 8 bytes (64 bits)

    data_type = 'q'   # Format character 'long long'

  else:             # Shared BRAMs data width of 4 bytes (32 bits)

    data_width = 4    # 4 bytes (32 bits)

    data_type = 'l'   # Format character 'long'


  add_width = bins_out    # Number of "Data Width" words of the implemented BRAM
                    # Must be set to store at least the number of output bins of each bram

  raw1 = np.zeros((n_outputs, bins_out))
  raw2 = np.zeros((n_outputs, bins_out))

  for i in range(n_outputs):    # Extract data from BRAMs blocks for each output

    raw1[i,:] = struct.unpack(f'>{bins_out}{data_type}',
    fpga.read(f'ab_re{i}', add_width * data_width, 0))
    raw2[i,:] = struct.unpack(f'>{bins_out}{data_type}',
    fpga.read(f'ab_im{i}', add_width * data_width, 0))

  re = raw1.T.ravel().astype(np.float64)
  im = raw2.T.ravel().astype(np.float64)

  return re, im
```

```
45
46 def plot_phase_diff(fpga, instrument, Nfft, n_bits, bin_step):
47     '''Sweeps frequencies and plots phase difference with given options'''
48
49     fs = 3932.16/2    # Bandwidth
50     LO = 3000   # Local Oscillator
51     phase = []    # Phase Difference
52     freq_data = []
53
54     try:
55         n_outputs = 8
56         if_freqs = np.linspace(0, fs, Nfft, endpoint=False)
57         faxis_LSB = LO - if_freqs
58         faxis_USB = LO + if_freqs
59
60         for i in range(0, Nfft, bin_step):
61             instrument.write(f'FREQ {faxis_LSB[-i-1]}e6')
62             time.sleep(0.1)
63
64             re, im = get_vacc_data_re_im(fpga, n_outputs=n_outputs, nfft=Nfft, n_bits=
    ↪ n_bits)
65             comp = fft.fftshift(re + 1j*im)
66             angle = np.angle(comp[-1-i], deg=True)
67
68             freq_data.append(faxis_LSB[-i-1] / 1000)
69             phase.append(angle)
70             print(faxis_LSB[-i-1]/1000, angle)
71
72         for i in range(0, Nfft, bin_step):
73             instrument.write(f'FREQ {faxis_USB[i]}e6')
74             time.sleep(0.1)
75
76             re, im = get_vacc_data_re_im(fpga, n_outputs=n_outputs, nfft=Nfft, n_bits=
    ↪ n_bits)
77             comp = fft.fftshift(re + 1j*im)
78             angle = np.angle(comp[i], deg=True)
79
80             freq_data.append(faxis_USB[i] / 1000)
81             phase.append(angle)
82             print(faxis_USB[i]/1000, angle)
83
84     except KeyboardInterrupt:
85         print("User interruption. Plotting data...")
86
87     # Save CSV
88     with open('phase_diff_data.csv', 'w', newline='') as csvfile:
89         csv_writer = csv.writer(csvfile)
90         csv_writer.writerow(["Frequency (MHz)", "Phase Difference (degrees)"])
91         csv_writer.writerows(zip(freq_data, phase))
92
93     faxis = np.concatenate((faxis_LSB[::-1][::bin_step], faxis_USB[::bin_step]))[:len(phase)]
94
```

```
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.grid()
    ax.plot(faxis, phase, '-', color="black")

    ax.set_xlabel('RF Frequency (MHz)')
    ax.set_ylabel('Phase Difference in Degrees')
    ax.set_title('Measured Phase Difference between IF Outputs')
    ax.set_ylim(-180, 180)

    plt.show()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description='Sweeps frequencies and plots phase difference with given options',
        usage='python sweep_ph_plot_1966mhz.py <HOSTNAME_or_IP> <Nfft Size> <RF instrument IP address> <Data Output Width> [options]'
    )

    parser.add_argument('hostname', type=str, help='Hostname or IP for the Casper platform')
    parser.add_argument('nfft', type=int, help='Operation mode: Nfft Size')
    parser.add_argument('rf_instrument', type=str, help='RF instrument IP address')
    parser.add_argument('data_output_width', type=int, help='BRAMs data output width')

    parser.add_argument('-l', '--acc_len', type=int, default=2**7,
                        help='Set the number of vectors to accumulate between dumps. Default is 2*(2^28)/2048')
    parser.add_argument('-s', '--skip', action='store_true',
                        help='Skip programming and begin to plot data')
    parser.add_argument('-b', '--fpgfile', type=str, default='',
                        help='Specify the FPG file to load')

    args = parser.parse_args()

    hostname = args.hostname
    Nfft = args.nfft
    rf_instrument = args.rf_instrument
    n_bits = args.data_output_width

    # Use your .fpg file
    bitstream = args.fpgfile if args.fpgfile else '/home/jose/Workspace/rfsoc_models/DSS/dss_ideal_16384ch_1966mhz_cx/outputs/dss_ideal_16384ch_1966mhz_cx_2024-10-24_1244.fpg'

    print(f'Connecting to {hostname}...')
    fpga = casperfpga.CasperFpga(hostname)
    time.sleep(1)

    if not args.skip:
        print(f'Programming FPGA with {bitstream}...')
        fpga.upload_to_ram_and_program(bitstream)
```

```
        print('Done')
    else:
        fpga.get_system_information()
        print('Skip programming FPGA...')

    print('Initializing RFDC block...')
    fpga.adcs['rfdc'].init()
    c = fpga.adcs['rfdc'].show_clk_files()
    fpga.adcs['rfdc'].progpll('lmk', c[1])
    fpga.adcs['rfdc'].progpll('lmx', c[0])
    time.sleep(1)

    print('Configuring accumulation period...')
    fpga.write_int('acc_len', args.acc_len)

    if n_bits == 32:
       fpga.write_int('gain', 2**10)
    time.sleep(1)
    print('Done')

    print('Resetting counters...')
    fpga.write_int('cnt_rst', 1)
    fpga.write_int('cnt_rst', 0)
    time.sleep(1)
    print('Done')

    print('Connecting to instruments...')
    rm = pyvisa.ResourceManager('@py')
    instrument = rm.open_resource(f'TCPIP0::{rf_instrument}::INSTR')
    time.sleep(1)
    print('Done')

    try:
        plot_phase_diff(fpga, instrument, Nfft, n_bits, 1)
    except KeyboardInterrupt:
        sys.exit()
```

Código B.4: Test para evaluar el desempeño de la PCB con la señal del RESET: test_spec_cnt.py

```
import casperfpga
import time

'''The next code is intended to test the Reset signal on the RFSoC board.
This signal prepares the spectrometer for every cycle of integration and resets
the counter of accumulation in every falling edge.'''

fpga = casperfpga.CasperFpga('10.17.90.40')     # Use your RFSoC IP address
print('Done')

fpga.upload_to_ram_and_program('8192ch_32bits_reset/
```

```
    ↪ dss_ideal_8192ch_32bits_reset_1966mhz_cx.fpg')

print('Configuring accumulation period...')
fpga.write_int('acc_len', 2**13)
time.sleep(1)
print('Done')

print('Resetting counters...')
time.sleep(1)
fpga.write_int('cnt_rst', 1)
time.sleep(1)
fpga.write_int('cnt_rst', 0)
time.sleep(1)
print('Done')

"""The next while cycle print the time when the software register block 'acc_per_cycle'
reads 0, which means that the accumulation counter on the spectrometer was reset."""
t = time.time()
while True:
    # time.sleep(1)
    # print(f"sync_cnt:{fpga.read_uint('sync_cnt')}")
    print(f"acc_per_cycle:{fpga.read_uint('acc_per_cycle')}")
    # print(f"acc_cnt:{fpga.read_uint('acc_cnt')}")
    if fpga.read_uint('acc_per_cycle') == 0:
        print(round(time.time()-t, 5))
```

Código B.5: Medición del tiempo de lecturas de espectros con Python usando el protocolo KATCP de la librería casperfpga de CASPER: test_time_experiment.py

```
import sys, time, struct, csv
import numpy as np
import matplotlib.pyplot as plt
import casperfpga
from datetime import datetime

# Configuración inicial
hostname = '10.17.90.187'
bitstream = '/home/jose/Workspace/RFSoC-MINI-2SB-Receiver/8192ch_32bits_reset/
    ↪ dss_ideal_8192ch_32bits_reset_1966mhz_cx/outputs/
    ↪ dss_ideal_8192ch_32bits_reset_1966mhz_cx_2025-04-07_1901.fpg'
n_bits = 32

print(f'Connecting to {hostname}...')
fpga = casperfpga.CasperFpga(hostname)
time.sleep(0.2)

fpga.upload_to_ram_and_program(bitstream)

print('Initializing RFDC block...')
fpga.adcs['rfdc'].init()
```

```
c = fpga.adcs['rfdc'].show_clk_files()
fpga.adcs['rfdc'].progpll('lmk', c[1])
fpga.adcs['rfdc'].progpll('lmx', c[0])
time.sleep(1)

print('Configuring accumulation period...')
fpga.write_int('acc_len', 2**14)
fpga.write_int('gain', 2**20)
time.sleep(1)
print('Done')

print('Resetting counters...')
fpga.write_int('cnt_rst', 1)
fpga.write_int('cnt_rst', 0)
time.sleep(1)
print('Done')

# Parámetros de medición
duration = 60 * 60  # 60 minutos en segundos

n_channels = [8192]

for j in range(len(n_channels)):

    start_time = time.time()
    time_stamps = []
    measure_times = []

    # Nombre del archivo CSV con timestamp
    csv_filename = f'python_katcp_{n_channels[j]}ch_time_differences_{datetime.now().
    ↪ strftime("%Y%m%d_%H%M%S")}.csv'

    # Escribir encabezado del archivo CSV
    with open(csv_filename, 'w', newline='') as file:
        csv_writer = csv.writer(file)
        csv_writer.writerow(["Iteration", "Measurement Time (us)", "Elapsed Time (ns)"])

    print("Iniciando medición continua...")
    iteration = 0

    while time.time() - start_time < duration:
        iter_start = time.time()

        for i in range(8):

            struct.unpack(f'>{n_channels[j] // 8}L',
            fpga.read(f'synth0_{i}', n_channels[j] // 8 * 4, 0))

            time.sleep(0.001)

        iter_end = time.time()

```

```
        time.sleep(0.020)

        elapsed_time = iter_end - start_time  # Tiempo transcurrido en segundos
        measurement_time = (iter_end - iter_start) * 1e3  # Tiempo que tarda la medición en segundos

        time_stamps.append(elapsed_time)
        measure_times.append(measurement_time)

        # Guardar datos en el archivo CSV
        with open(csv_filename, 'a', newline='') as file:
            csv_writer = csv.writer(file)
            csv_writer.writerow([iteration, measurement_time, elapsed_time])

        iteration += 1

    print(f"Medición finalizada. Datos guardados en {csv_filename}")

# Leer datos del CSV y graficar
iterations = []
time_differences = []
elapsed_time = []

with open(csv_filename, 'r') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # Saltar encabezado
    for row in csv_reader:
        iterations.append(int(row[0]))
        time_differences.append(float(row[1]))  # Tiempo de medición en milisegundos
        elapsed_time.append(float(row[2]) / 60)  # Tiempo transcurrido en minutos

# Graficar resultados
plt.figure(figsize=(10, 6))
plt.plot(elapsed_time, time_differences, label='Measurement Latency (ms)', color='b')
plt.xlabel('Elapsed Time (min)')
plt.ylabel('Measurement Latency (ms)')
plt.title('Measurement Latency Over Time')
plt.grid(True)
plt.ylim(0, 240)
plt.axhline(y=25, color='r', linestyle='--', label='25 ms')
plt.legend()
plt.tight_layout()
plt.show()
```

Código B.6: Servidor en C++ que lee las BRAM del RFSoC 4x2 y la entrega a un cliente que las solicita por un Socket: `rfsoc_server.cpp`

```
#include <iostream>
#include <iomanip>
#include <fcntl.h>
#include <unistd.h>
```

```cpp
#include <sys/mman.h>
#include <cstdint>
#include <sys/socket.h>
#include <netinet/in.h>
#include <cstring>
#include <unordered_map>
#include <mutex>

#define BRAM_SIZE 4096
#define ACC_CNT_SIZE 4
#define PORT 12345

std::unordered_map<std::string, uintptr_t> bram_addresses = {
    {"synth0_0", 0xA0160000}, {"synth0_1", 0xA0161000}, {"synth0_2", 0xA0162000}, {"synth0_3", 0xA0163000},
    {"synth0_4", 0xA0164000}, {"synth0_5", 0xA0165000}, {"synth0_6", 0xA0166000}, {"synth0_7", 0xA0167000},
    {"synth1_0", 0xA0168000}, {"synth1_1", 0xA0169000}, {"synth1_2", 0xA016A000}, {"synth1_3", 0xA016B000},
    {"synth1_4", 0xA016C000}, {"synth1_5", 0xA016D000}, {"synth1_6", 0xA016E000}, {"synth1_7", 0xA016F000},
    {"acc_cnt", 0xA0170000}
};

std::unordered_map<std::string, void*> mapped_brams;
std::mutex bram_mutex;
int fd = -1;

bool init_bram() {
    fd = open("/dev/mem", O_RDWR | O_SYNC);
    if (fd < 0) {
        std::cerr << "Error al abrir /dev/mem" << std::endl;
        return false;
    }

    for (const auto& pair : bram_addresses) {
        size_t size = (pair.first == "acc_cnt") ? ACC_CNT_SIZE : BRAM_SIZE;
        void* ptr = mmap(nullptr, size, PROT_READ, MAP_SHARED, fd, pair.second);
        if (ptr == MAP_FAILED) {
            std::cerr << "Error al mapear BRAM: " << pair.first << std::endl;
            return false;
        }
        mapped_brams[pair.first] = ptr;
    }
    return true;
}

void cleanup_bram() {
    for (const auto& pair : mapped_brams) {
        size_t size = (pair.first == "acc_cnt") ? ACC_CNT_SIZE : BRAM_SIZE;
        munmap(pair.second, size);
    }
```

```
53      if (fd >= 0) close(fd);
54  }
55
56  void send_bram_data(int client_fd, const std::string& bram_name, size_t offset, size_t
        ↪ length) {
57      std::lock_guard<std::mutex> lock(bram_mutex);
58
59      auto it = mapped_brams.find(bram_name);
60      if (it == mapped_brams.end()) {
61          std::cerr << "BRAM no encontrada: " << bram_name << std::endl;
62          send(client_fd, "ERROR", 5, 0);
63          return;
64      }
65
66      size_t max_size = (bram_name == "acc_cnt") ? ACC_CNT_SIZE : BRAM_SIZE;
67      if (offset >= max_size) {
68          send(client_fd, "ERROR", 5, 0);
69          return;
70      }
71      if (offset + length > max_size) {
72          length = max_size - offset;
73      }
74
75      send(client_fd, static_cast<uint8_t*>(it->second) + offset, length, 0);
76  }
77
78  int main() {
79      if (!init_bram()) return -1;
80
81      int server_fd = socket(AF_INET, SOCK_STREAM, 0);
82      if (server_fd < 0) {
83          std::cerr << "Error al crear el socket" << std::endl;
84          return -1;
85      }
86
87      sockaddr_in server_addr{};
88      server_addr.sin_family = AF_INET;
89      server_addr.sin_addr.s_addr = INADDR_ANY;
90      server_addr.sin_port = htons(PORT);
91
92      if (bind(server_fd, (sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
93          std::cerr << "Error al enlazar el socket" << std::endl;
94          return -1;
95      }
96
97      if (listen(server_fd, 1) < 0) {
98          std::cerr << "Error al escuchar conexiones" << std::endl;
99          return -1;
100     }
101
102     std::cout << "Servidor esperando una conexion en el puerto " << PORT << "..." << std::
        ↪ endl;
```

```
103
104     sockaddr_in client_addr;
105     socklen_t client_len = sizeof(client_addr);
106     int client_fd = accept(server_fd, (sockaddr*)&client_addr, &client_len);
107     if (client_fd < 0) {
108         std::cerr << "Error al aceptar conexion" << std::endl;
109         return -1;
110     }
111
112     std::cout << "Cliente conectado." << std::endl;
113
114     while (true) {
115         char buffer[128] = {0};
116         ssize_t bytes = recv(client_fd, buffer, sizeof(buffer) - 1, 0);
117         if (bytes <= 0) break;
118
119         std::string request(buffer);
120         std::istringstream iss(request);
121         std::string bram_name;
122         size_t offset = 0, length = 0;
123
124         if (!(iss >> bram_name >> offset >> length)) {
125             std::cerr << "Formato de solicitud invalido" << std::endl;
126             send(client_fd, "ERROR", 5, 0);
127             continue;
128         }
129
130         send_bram_data(client_fd, bram_name, offset, length);
131     }
132
133     std::cout << "Cliente desconectado." << std::endl;
134     cleanup_bram();
135     close(client_fd);
136     close(server_fd);
137     return 0;
138 }
```

Código B.7: Socket programado en C++ para solicitar espectros al servidor del RFSoC 4x2: cpp_socket.cpp

```
1  #include <pybind11/pybind11.h>
2  #include <pybind11/stl.h>
3  #include <pybind11/stl_bind.h>
4  #include <iostream>
5  #include <string>
6  #include <vector>
7  #include <sys/socket.h>
8  #include <arpa/inet.h>
9  #include <unistd.h>
10
11 class CPPSocket {
```

```
public:
    CPPSocket(const std::string& host, int port) {
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0) {
            throw std::runtime_error("Failed to create socket");
        }

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(port);
        if (inet_pton(AF_INET, host.c_str(), &server_addr.sin_addr) <= 0) {
            throw std::runtime_error("Invalid address/ Address not supported");
        }

        if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
            throw std::runtime_error("Connection failed");
        }
    }

    std::vector<uint8_t> send_request(const std::string& request) {
        // std::string modified_request = request + "\n"; // Create a modifiable copy
        if (send(sockfd, request.c_str(), request.length(), 0) < 0) {
            throw std::runtime_error("Failed to send request");
        }

        std::vector<uint8_t> buffer(20000); // Allocate buffer for receiving raw data
        int valread = recv(sockfd, buffer.data(), buffer.size(), 0);
        if (valread < 0) {
            throw std::runtime_error("Failed to receive response");
        }

        buffer.resize(valread); // Trim to actual received size
        return buffer;

    }
    //     char buffer[4096] = {0};
    //     int valread = recv(sockfd, buffer, 4096, 0);
    //     if (valread < 0) {
    //         throw std::runtime_error("Failed to receive response");
    //     }
    //     return std::string(buffer, valread);
    // }

    ~CPPSocket() {
        close(sockfd);
    }

private:
    int sockfd;
    struct sockaddr_in server_addr;
};

PYBIND11_MODULE(cpp_socket, m) {
```

```
    pybind11::class_<CPPSocket>(m, "CPPSocket")
        .def(pybind11::init<const std::string&, int>())
        .def("send_request", [](CPPSocket& self, const std::string& request) {
            std::vector<uint8_t> data = self.send_request(request);
            return pybind11::bytes(reinterpret_cast<const char*>(data.data()), data.size());
        }, pybind11::return_value_policy::move);
}
```

Código B.8: Medición del tiempo de lecturas de espectros con Python usando un cliente personalizado y un socket programado en C++: cpp_interface.py

```
import cpp_socket
import time
import struct
from numpy import fft
import csv
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import re

# Parámetros de medición
duration = 60 * 60  # 60 minutos en segundos

nchan = [8192]

client = cpp_socket.CPPSocket("10.17.90.187", 12345)

for j in range(1):

    # Nombre del archivo CSV con timestamp
    csv_filename = f'python_{nchan[j]}ch_times_{datetime.now().strftime("%Y%m%d_%
    ↪ H%M%S")}.csv'

    # Escribir encabezado del archivo CSV
    with open(csv_filename, 'w', newline='') as file:
        csv_writer = csv.writer(file)
        csv_writer.writerow(["Iteration", "Measurement Time (ms)", "Elapsed Time (min)"])

    print("Iniciando medición continua...")
    iteration = 0

    n_bytes = nchan[j] // 8 * 4

    start_time = time.time()
    time_stamps = []
    measure_times = []

    while time.time() - start_time < duration:

```

```
39          iter_start = time.time()
40
41          raw1 = np.zeros((8, nchan[j]//8))
42          raw2 = np.zeros((8, nchan[j]//8))
43
44          for i in range(8):    # Extract data from BRAMs blocks for each output
45
46              response1 = client.send_request(f"synth0_{i} 0 {n_bytes}")
47              # print(response1)
48              # print(len(response1))
49              raw1[i,:] = struct.unpack(f'<{nchan[j] // 8}L', response1)
50              # #print(raw1[i,:])
51              time.sleep(0.001)
52
53              response2 = client.send_request(f"synth1_{i} 0 {n_bytes}")
54              # # # print(len(msg_translated2))
55              raw2[i,:] = struct.unpack(f'<{nchan[j] // 8}L', response2)
56              time.sleep(0.001)
57
58          interleave_i = raw1.T.ravel().astype(np.float64)
59          interleave_q = raw2.T.ravel().astype(np.float64)
60
61          interleave_shift_i = fft.fftshift(interleave_i)
62          interleave_shift_q = fft.fftshift(interleave_q)
63
64          # print(interleave_shift_i)
65          # print(interleave_shift_q)
66
67
68          #print(len(msg_translated))
69          #print(type(msg_translated))
70
71          iter_end = time.time()
72
73          time.sleep(0.02)
74
75          elapsed_time = iter_end - start_time  # Tiempo transcurrido
76          measurement_time = (iter_end - iter_start) * 1e3  # Tiempo que tarda la medición
    ↪ en milisegundos
77          #print(f"Elapsed time {measurement_time}")
78          # Guardar datos en el archivo CSV
79          with open(csv_filename, 'a', newline='') as file:
80              csv_writer = csv.writer(file)
81              csv_writer.writerow([iteration, measurement_time, elapsed_time])
82
83          iteration += 1
84          # time.sleep(0.1)  # Pequeña pausa para evitar sobrecarga
85
86      print(f"Medición finalizada. Datos guardados en {csv_filename}")
87
88
89
```

```
# Leer datos del CSV y graficar
iterations = []
time_differences = []
elapsed_time = []

with open(csv_filename, 'r') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # Saltar encabezado
    for row in csv_reader:
        iterations.append(int(row[0]))
        time_differences.append(float(row[1]))  # Tiempo de medición en milisegundos
        elapsed_time.append(float(row[2])/60)  # Tiempo transcurrido en minutos

# Graficar resultados
plt.figure(figsize=(10, 6))
plt.plot(elapsed_time, time_differences, label='Measurement Latency (ms)', color='b')
plt.xlabel('Elapsed Time (min)')
plt.ylabel('Measurement Latency (ms)')
plt.title('Measurement Latency Over Time')
# Optionally, add a grid
plt.grid(True)
plt.ylim(0, 240)
plt.axhline(y=25, color='r', linestyle='--', label='25 ms')
# Show the plot
plt.legend()
plt.tight_layout()
plt.show()
```

Código B.9: Comunicación implementada en el Radiotelescopio Mini: `rf-soc_server.py`. (Este código fue desarrollado en conjunto con el operador del Mini)

```
#!/usr/bin/env python3

# SOCKET SERVER

import socket
import struct
import time, datetime

import threading
import queue
import select
import numpy as np
import casperfpga
from numpy import fft
import csv

import os
import cpp_socket

```

```
# Define IP and port to use

# REMEMBER TO SET COMPUTER's IP TO 192.168.1.14

HOST_PIC = "192.168.7.119" # IP
PORT_PIC = 1234 # Port to listen on

HOST_RFSOC = "192.168.7.187"

# FPGA Model Parameters
ACC_LEN_SPLOBS = 2**12
ACC_LEN_CAL = 2**12
GAIN = 2**12

# Define the numbers of telescope states per package, and the packet length
statesPerPackage = 20
packetLength = 70

# Queue for sending
RFSoC_requests_queue = queue.Queue()
send_to_PIC_queue = queue.Queue()
spectra_write_to_disk_queue = queue.Queue()

# Read and change to target path for storing the data
with open('datapath.txt') as f:
    lines = f.readlines()

target_path = lines[0][:lines[0].index('\n')]
os.chdir(target_path)

# checking if the data directory
# exists or not
if not os.path.exists("bin_spectra_and_states"):

    # if the directory is not present
    # then create it.
    os.makedirs("bin_spectra_and_states")

def program_fpga(fpga):
  """
  Function to program the fpga
  """
  a = 0
  # Program fpga and stuff
def request_acc_cnt(client):
  acc_recv = client.send_request('acc_cnt 0 4')
  #print(f"acc_recv: {acc_recv}")
  acc_int = struct.unpack('<1L', acc_recv)[0]
  #print(f"acc_int: {acc_int}")
  acc_cnt_hex = hex(acc_int)
  # 4 Bytes!!! Will this be a problem? Maybe PIC reads only 2 bytes

```

```
    acc_bytes = str.encode(acc_cnt_hex)
    #response = acc_cnt.to_bytes(2, byteorder = "big")
    return acc_bytes

def request_512_channels_1band(client, first_chan:int):
    """Get the raw data in bytes from fpga digital spectrometer"""
    nfft = 8192
    n_outputs = 8
    # bins_out = nfft//n_outputs    # Number of bins for each output

    N_CHANNELS = 512

    bins_out = N_CHANNELS // 8
    data_width = 4    # Data output width of 4 bytes (32 bits)

    add_width = bins_out    # Number of "Data Width" words of the implemented BRAM
                      # Must be set to store at least the number of output bins of each bram

    raw1 = np.zeros((n_outputs, bins_out))
    raw2 = np.zeros((n_outputs, bins_out))


    for i in range(n_outputs):    # Extract data from BRAMs blocks for each output
        data_in_bytes_USB = client.send_request(f"synth0_{i} {1024*4//2+(first_chan)
        ↪ //8*4} {add_width * data_width}")
        # print(len(data_in_bytes_USB))
        raw1[i,:] = struct.unpack(f'<{bins_out}L', data_in_bytes_USB)
        time.sleep(0.0005)

        # data_in_bytes_LSB = client.send_request(f"synth1_{i} {1024*4//2+(first_chan)
        ↪ //8*4} {add_width * data_width}")
        # print(len(data_in_bytes_LSB))
        # raw2[i,:] = struct.unpack(f'<{bins_out}L', data_in_bytes_LSB)
        # time.sleep(0.0005)

        #fpga.read(f'synth1_{i}', add_width * data_width, 1024*4//2+768//8*4))

    interleave_i = raw1.T.ravel().astype(np.uint32)
    interleave_q = raw2.T.ravel().astype(np.uint32)

    # print(len(interleave_i))
    return interleave_i

def request_512_channels_and_save(fpga):
    """Get the raw data in bytes from fpga digital spectrometer"""
    n_outputs = 8
    nfft = 8192
    N_CHANNELS = 512 # Number of channels to read
    FIRST_CHANNEL = 768 # the first channel to read, copied from Roach communication
    #q, i = get_vacc_data_power(fpga, n_outputs, nfft)

    spectra_write_to_disk_queue.put([q, i])
```

```
122
123     spectrum_for_pic = q[FIRST_CHANNEL:FIRST_CHANNEL+N_CHANNELS]
124     return spectrum_for_pic
125
126     # Pack the first 512 values
127
128 def set_cal_mode(fpga):
129     #fpga.write_int('acc_len', ACC_LEN_CAL)
130     acc_len = fpga.read_uint('acc_len')
131     print("FPGA acc len set to " + str(acc_len))
132
133 def set_splobs_mode(fpga):
134     #fpga.write_int('acc_len', ACC_LEN_SPLOBS)
135     acc_len = fpga.read_uint('acc_len')
136     print("FPGA acc len set to " + str(acc_len))
137
138 def receive_from_PIC(PIC_socket):
139     """
140     Receives the data from the PIC socket and if it is the telescope status,
141     saves it into a binary file.
142     PIC_socket: PIC socket
143     """
144
145     # Initializes filename of the csv to save, and the folder of states
146     filename = "bin_spectra_and_states/"+str(datetime.datetime.now())+"_STATE"
147     filename = filename.replace(" ", "_")
148
149     with open(filename, 'ab') as f:
150       while True:
151         try:
152           while True:
153             rlist, _, _ = select.select([PIC_socket], [], [])
154             #if not sendingToPIC.is_set():
155             #receivingFromPIC.set()
156             #print("Receiving from PIC:")
157             data_from_PIC = PIC_socket.recv(2048)
158
159             while True:
160               statusIndex = data_from_PIC.find(b'ST')
161               if statusIndex != -1:
162                 # Data containing status from the telescope, saving to server
163
164                 f.write(data_from_PIC[statusIndex:statusIndex+packetLength])
165                 #print(dataFromPIC[statusIndex:statusIndex+packetLength])
166                 #print("Saving telescope status at:")
167                 #print(time.time())
168                 data_from_PIC = data_from_PIC[:statusIndex] + data_from_PIC[
    ↪ statusIndex+packetLength:]
169               else:
170                 break
171
172           if len(data_from_PIC) != 0:
```

```
173                 # Data going to RFSoC
174                 #print("PIC: ",data_from_PIC)
175                 RFSoC_requests_queue.put(data_from_PIC)
176
177
178                 #send_to_RFSOC_queue.put(dataFromPIC)
179
180               #print("Received from PIC and added to queue:")
181               #print(dataFromPIC)
182               #if isTelescopeStatus(dataFromPIC):
183               #  f.write(dataFromPIC[:])
184               #  print("Saving telescope status at:")
185               #  print(time.time())
186               #else:
187               #  PIC_queue.put(dataFromPIC)
188               #receivingFromPIC.clear()
189           finally:
190             f.close()
191
192
193
194 def process_RFSoC_request(fpga, client):
195   """
196   Receives the data from the ROACH socket
197   ROACH_socket:  Roach socket
198   """
199   # Spectrum 512 ch buffer
200   spectrum_buffer_512 = np.zeros(512).astype(np.uint32)
201   last_acc = request_acc_cnt(client)
202   counter = 0
203   while True:
204     request = RFSoC_requests_queue.get()
205     if request[:17] == b"?wordread acc_cnt":
206       counter += 1
207       cnt_in_bytes = request_acc_cnt(client)
208       response = b"!wordread ok "+ cnt_in_bytes+ b"\n"
209       if cnt_in_bytes != last_acc:
210         counter = 0
211         #spectrum_buffer_512 = request_512_channels_and_save(fpga)
212         #t1 = time.time()
213         #spectrum_buffer_512 = request_512_channels(fpga)
214         spectrum_buffer_512 = request_512_channels_1band(client, 768)[:512]
215         #print(time.time()-t1)
216         last_acc = cnt_in_bytes
217       send_to_PIC_queue.put(response)
218       #cnt_in_bytes = request_acc_cnt(fpga)
219       #response_in_bytearray = bytearray(b"!read ok ")
220       #response_in_bytearray.append(cnt_in_bytes)
221       #bytes_result = bytes([byte for byte in response_in_bytearray])
222       #send_to_PIC_queue.put(bytes_result)
223
224     elif request[:11] == b"?read bram0":
```

```
225         spectrum = spectrum_buffer_512
226         spec_in_bytes = struct.pack(f">{len(spectrum)}L", *spectrum)
227         response = b"!read ok " + spec_in_bytes + b"\n"
228         send_to_PIC_queue.put(response)
229 
230         #spec_in_bytes = request_512_channels(fpga)
231         #response_in_bytearray = bytearray(b"!read ok ")
232         #response_in_bytearray.append(spec_in_bytes)
233         #response_in_bytearray.append(b'\n')
234         #bytes_result = bytes([byte for byte in response_in_bytearray])
235         #send_to_PIC_queue.put(bytes_result)
236         #spectra_write_to_disk_queue.put(spec_in_bytes)
237         #save_full_spectrum(fpga)
238 
239     elif request[:8] == b"?progdev":
240         response = b"!progdev ok 525\n"
241         send_to_PIC_queue.put(response)
242         #response_in_bytearray = bytearray(b"!progdev ok 525\n")
243         #bytes_result = bytes([byte for byte in response_in_bytearray])
244         #send_to_PIC_queue.put(bytes_result)
245 
246     elif request[:25] == b"?wordwrite integ_mode 0 0":
247         set_cal_mode(fpga)
248         response = b"!wordwrite ok\n"
249         send_to_PIC_queue.put(response)
250 
251     elif request[:25] == b"?wordwrite integ_mode 0 1":
252         set_splobs_mode(fpga)
253         response = b"!wordwrite ok\n"
254         send_to_PIC_queue.put(response)
255 
256     elif request[:10] == b"?wordwrite":
257         response = b"!wordwrite ok\n"
258         send_to_PIC_queue.put(response)
259 
260         #response_in_bytearray = bytearray(b"!wordwrite ok\n")
261         #bytes_result = bytes([byte for byte in response_in_bytearray])
262         #send_to_PIC_queue.put(bytes_result)
263 
264     elif request[:6] == b"?write":
265         response = b"!write ok\n"
266         send_to_PIC_queue.put(response)
267 
268         #response_in_bytearray = bytearray(b"!write ok\n")
269         #bytes_result = bytes([byte for byte in response_in_bytearray])
270         #send_to_PIC_queue.put(bytes_result)
271 
272 
273 def save_spectra_to_hdd():
274     """
275     Saves spectra from queue to hard drive
276     """
```

```
277     # Initializes filename of the csv to save, and the folder of states
278   filename = "bin_spectra_and_states/"+str(datetime.datetime.now()) +"_ROACHDATA.
        ↪ csv"
279   filename = filename.replace(" ", "_")
280
281   with open(filename, 'a') as f:
282     try:
283       while True:
284         #if not sendingToROACH.is_set():
285         #receivingFromROACH.set()
286         #print("Receiving from ROACH:")
287         spectrum = spectra_write_to_disk_queue.get()
288         #print("Received from ROACH and added to queue:")
289         #print(dataFromROACH)
290
291         #f.write('Tmst')
292         actualTime = time.time()
293         #print(actualTime)
294         f.write(str(actualTime))
295         f.write("\n")
296         csv.writer(f,delimiter=",").writerows(spectrum)
297         #receivingFromROACH.clear()
298     finally:
299       f.close()
300
301 def sending_data(PIC_socket:socket):
302   """
303   Sends the data to the PIC socket
304   """
305   while True:
306     _, wlist, _ = select.select([], [PIC_socket], [])
307     for readysocket in wlist:
308       if readysocket is PIC_socket and not send_to_PIC_queue.empty(): # and not
        ↪ receivingFromPIC.is_set():
309         #sendingToPIC.set()
310         data_to_PIC = send_to_PIC_queue.get()
311         #print("DATASERVER: ",data_to_PIC)
312         #print("Sending to PIC:")
313         #print(dataToPIC)
314         #print(len(dataToPIC))
315         PIC_socket.send(data_to_PIC)
316         #now = datetime.datetime.now()
317         #print(now.time())
318         #sendingToPIC.clear()
319
320
321 """
322 MAIN PROGRAM
323 """
324 """
325 ROACH_s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
326 print(f"Waiting for connection to {HOST_RF} Port {PORT_ROACH}")
```

```
ROACH_s.connect((HOST_ROACH, PORT_ROACH))
print(f"Connected to {HOST_ROACH}")

CONNECT TO RFSOC HERE!!!!!!!
"""
client = cpp_socket.CPPSocket(HOST_RFSOC, 12345)

hostname = HOST_RFSOC   # IP address RFSoC
Nfft = 8192    # FFT Size

# FPGA .fpg file and .dtbo must be in the same folder
bitstream = '/home/mini-dataserver/miniQuimal/src/dataserver_as_interface/models/
    ↪ dss_ideal_8192ch_32bits_reset_1966mhz_cx_2025-04-07_1901.fpg'

print(f'Connecting to {hostname}...')
fpga = casperfpga.CasperFpga(hostname)
time.sleep(0.2)

print(f'Programming FPGA with {bitstream}...')
fpga.upload_to_ram_and_program(bitstream)
time.sleep(5)
print('Done')

print('Initializing RFDC block...')
fpga.adcs['rfdc'].init()
c = fpga.adcs['rfdc'].show_clk_files()
fpga.adcs['rfdc'].progpll('lmk', c[1])
fpga.adcs['rfdc'].progpll('lmx', c[0])
time.sleep(1)

print('Configuring accumulation period...')
fpga.write_int('acc_len', ACC_LEN_SPLOBS)
fpga.write_int('gain', GAIN)
time.sleep(1)
print('Done')

print('Resetting counters...')
fpga.write_int('cnt_rst', 1)
fpga.write_int('cnt_rst', 0)
time.sleep(1)
print('Done')


#fpga = casperfpga.CasperFpga(HOST_RFSOC)

PIC_s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
PIC_s.bind((HOST_PIC, PORT_PIC))
PIC_s.listen()
print(f"Waiting for connection in {HOST_PIC} Port {PORT_PIC}")
conn, addr = PIC_s.accept()
print(f"Connected to {addr}")

```

```
378
379 recv_from_PIC_thread = threading.Thread(target=receive_from_PIC,args=([conn]))
380 process_RFSoC_request_thread = threading.Thread(target=process_RFSoC_request, args
    ↪ =([fpga, client]))
381 #process_RFSoC_request_thread = threading.Thread(target=process_RFSoC_request)
382 sending_data_thread = threading.Thread(target=sending_data, args=([conn]))
383 saving_spectra_thread = threading.Thread(target=save_spectra_to_hdd)
384 #fill_spectrum_thread = threading.Thread(target=fill_spectrum_buffer, args=([fpga]))
385
386 recv_from_PIC_thread.start()
387 process_RFSoC_request_thread.start()
388 sending_data_thread.start()
389 saving_spectra_thread.start()
390 #fill_spectrum_thread.start()
391
392 #except:
393 #  conn.close()
394 #  PIC_s.close()
395 #  print("Program interrupted\n")
```

Anexo C. Resultados adicionales

C.1. Espectros adicionales de 64 bits

En las Figuras C.1 y C.2, se presentan los resultados obtenidos para el modelo de 64 bits con 2048 canales espectrales, utilizando señales de entrada en RF de 2500 MHz y 3500 MHz, respectivamente.

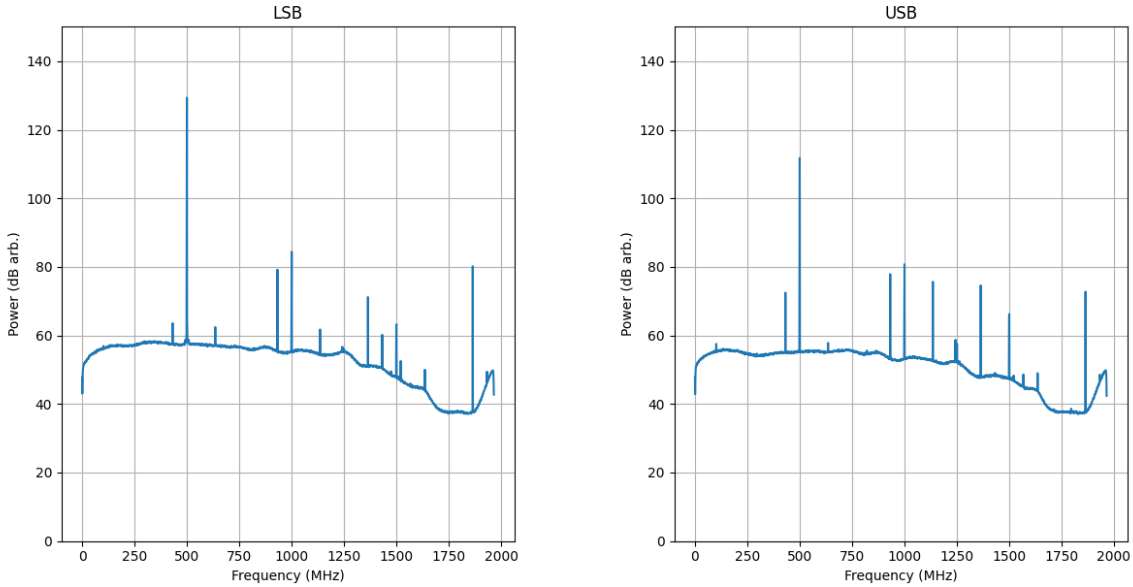


Figura C.1: Visualización del espectro de 64 bits para 2048 canales, con una frecuencia de entrada RF de 2500 MHz. Se observan las bandas LSB y USB.

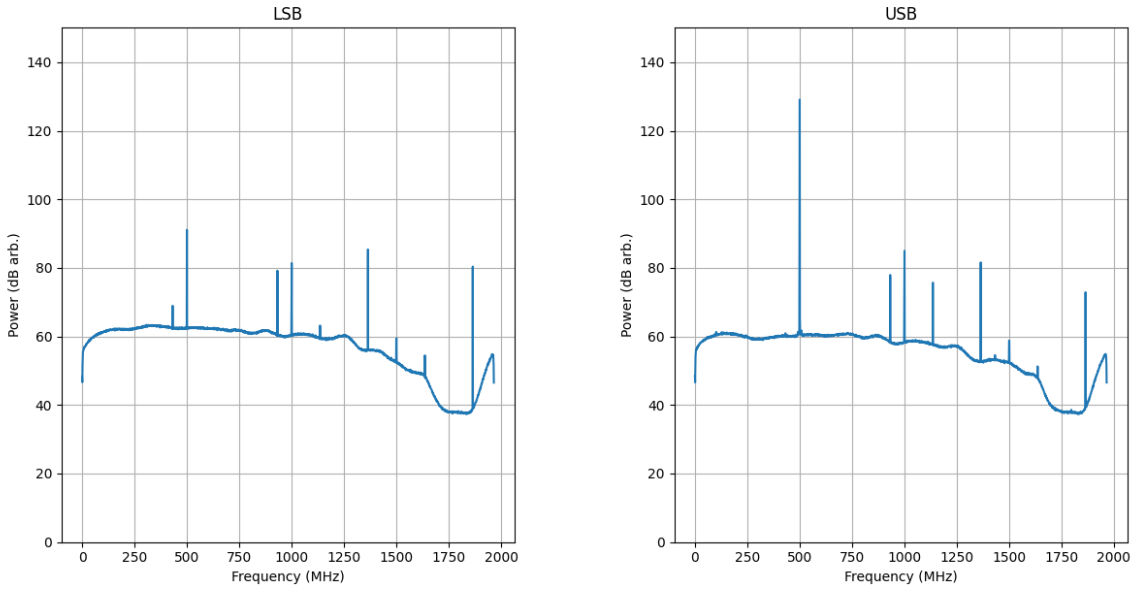


Figura C.2: Visualización del espectro de 64 bits para 2048 canales, con una frecuencia de entrada RF de 3500 MHz. Se observan las bandas LSB y USB.

En las Figuras C.3 y C.4, se presentan los resultados obtenidos para el modelo de 64 bits con 16384 canales espectrales, utilizando señales de entrada en RF de 2500 MHz y 3500 MHz, respectivamente.

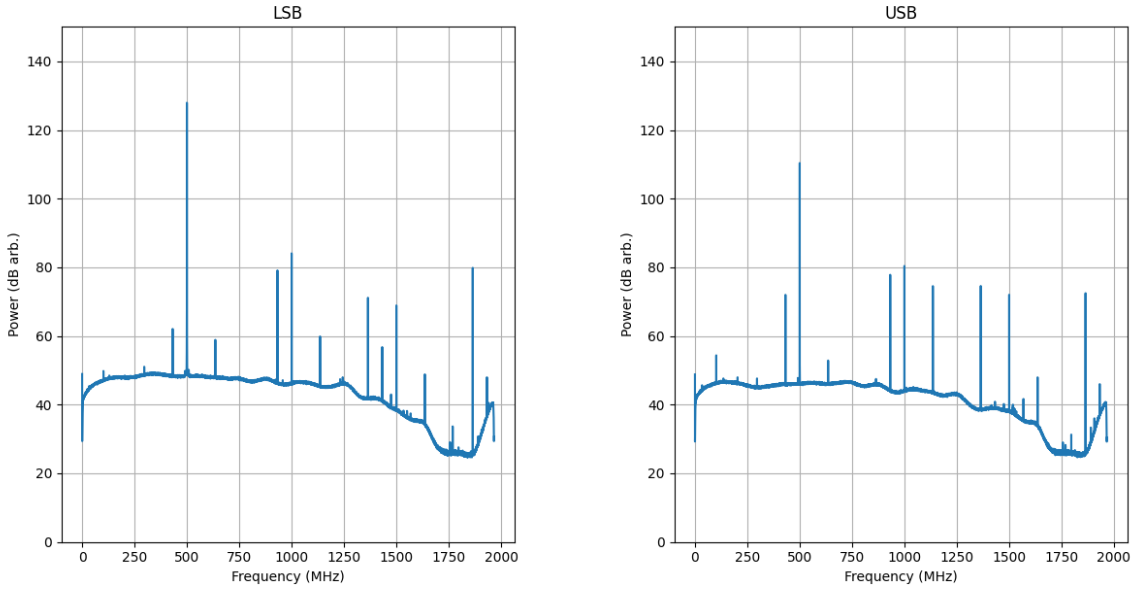


Figura C.3: Visualización del espectro de 64 bits para 16384 canales, con una frecuencia de entrada RF de 2500 MHz. Se observan las bandas LSB y USB.

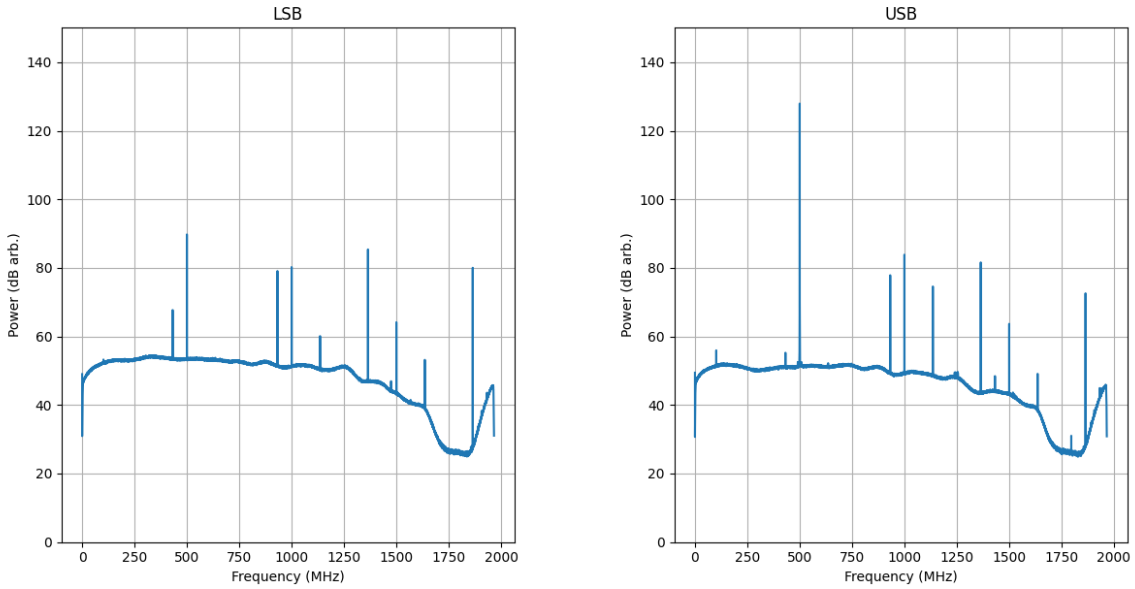


Figura C.4: Visualización del espectro de 64 bits para 16384 canales, con una frecuencia de entrada RF de 3500 MHz. Se observan las bandas LSB y USB.

C.2. SRR adicionales de modelos de 64 bits

En las Figuras C.5 y C.6, se presentan los gráficos obtenidos de SRR para los modelos de 64 bits con 2048 y 16384 canales espectrales, respectivamente.

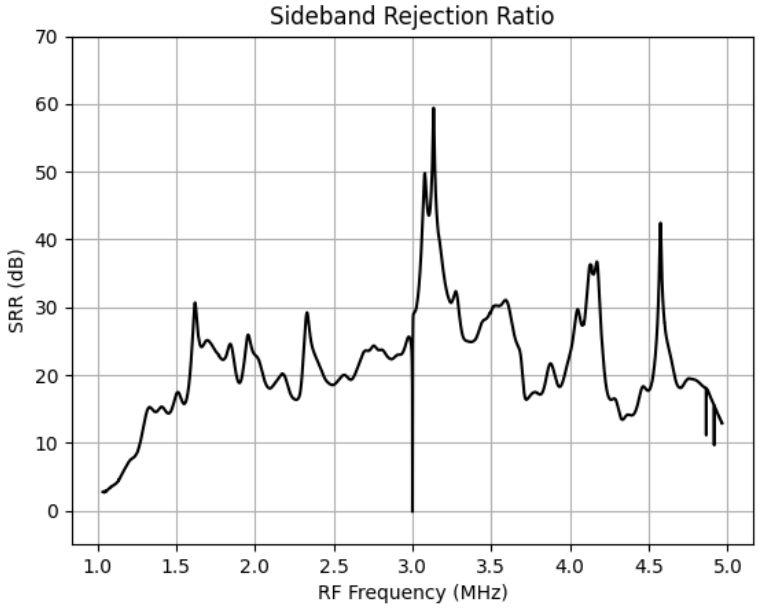


Figura C.5: SRR para modelo de 64 bits y 2048 canales espectrales.

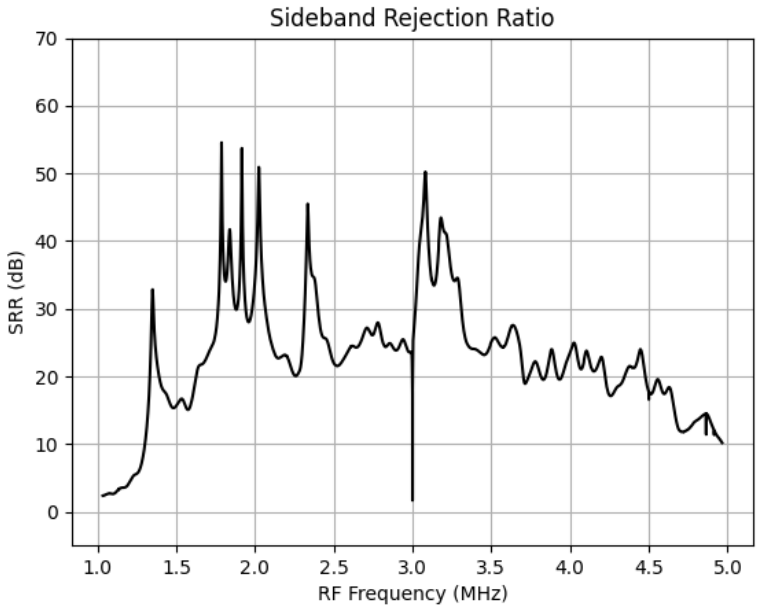


Figura C.6: SRR para modelo de 64 bits y 16384 canales espectrales.

C.3. SRR adicionales de modelos de 32 bits

En las siguientes figuras, se presentan más resultados de los gráficos obtenidos de SRR para el modelo de 32 bits con 8192 canales espectrales, para distintos valores de ganancia y largos de acumulaciones.

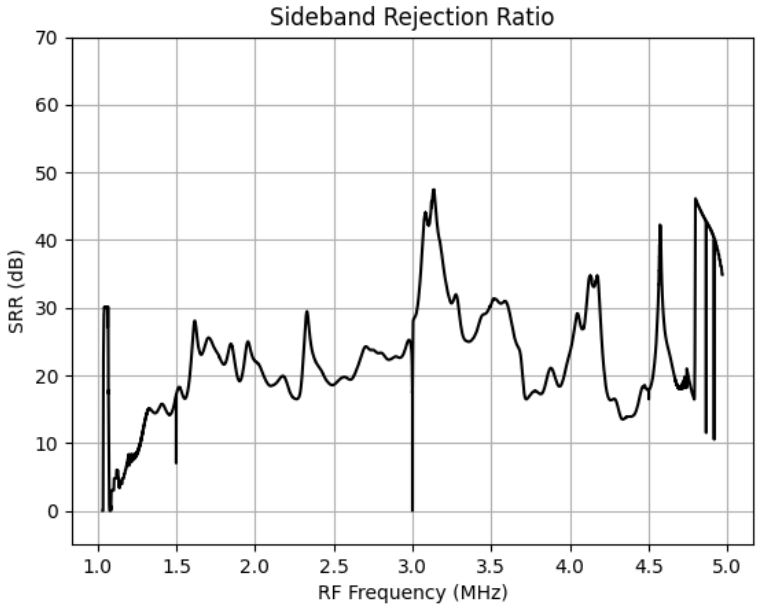


Figura C.7: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^6 y cantidad de acumulaciones 2^{10}.

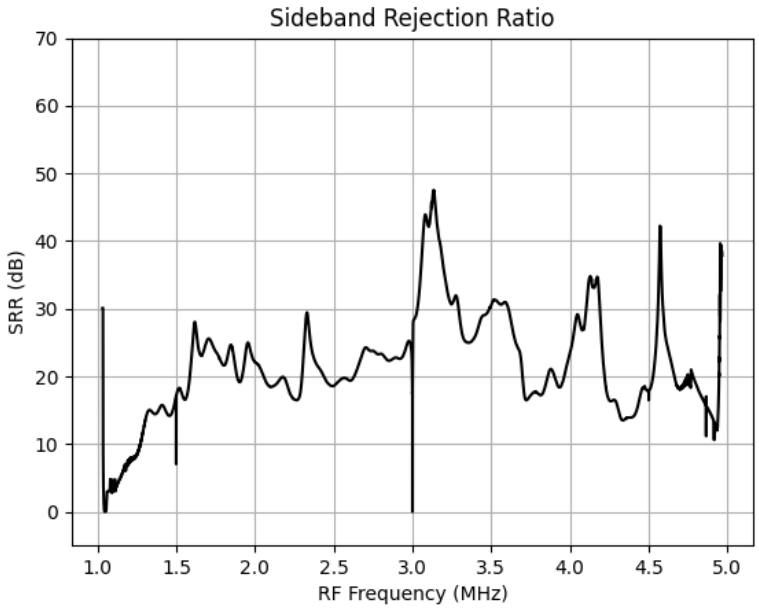


Figura C.8: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^7 y cantidad de acumulaciones 2^{10}.

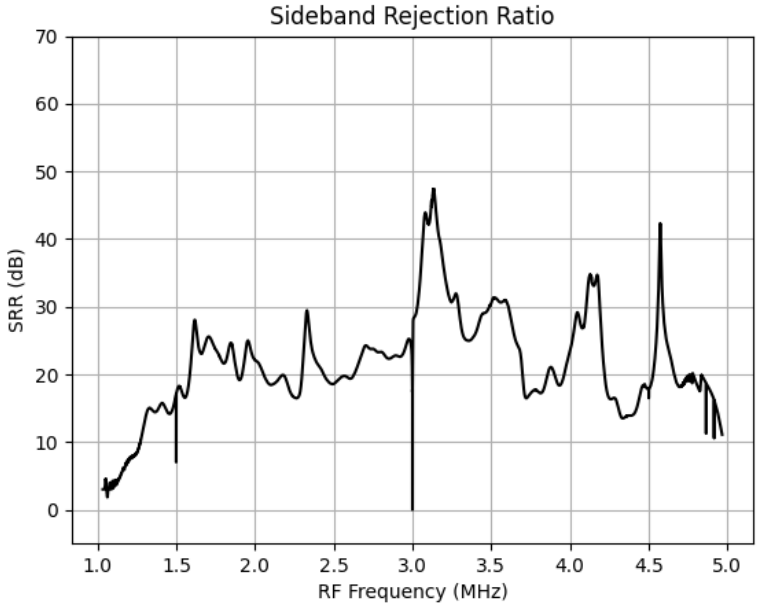


Figura C.9: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^8 y cantidad de acumulaciones 2^{10}.

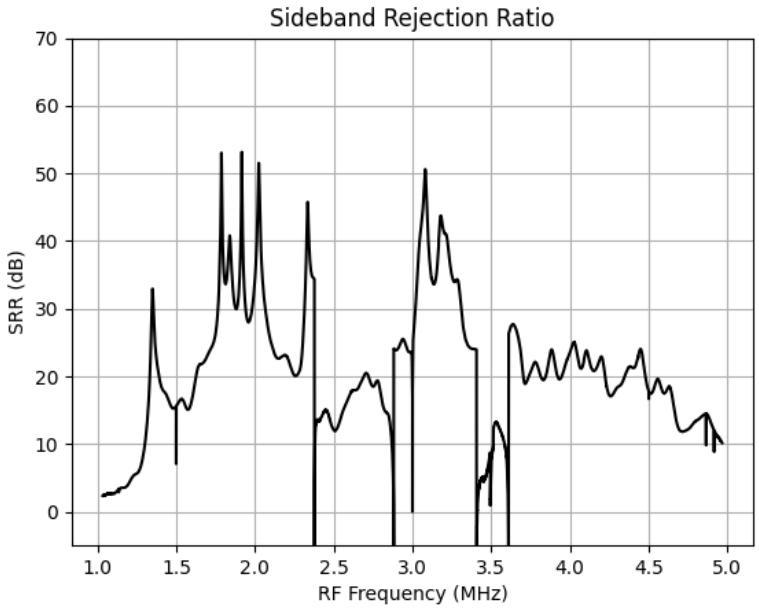


Figura C.10: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{11} y cantidad de acumulaciones 2^{9}.

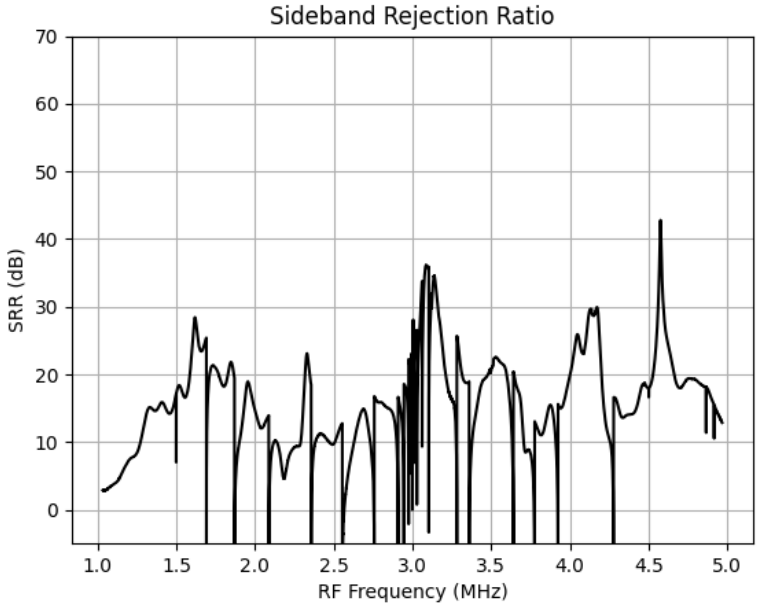


Figura C.11: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{12} y cantidad de acumulaciones 2^{10}.

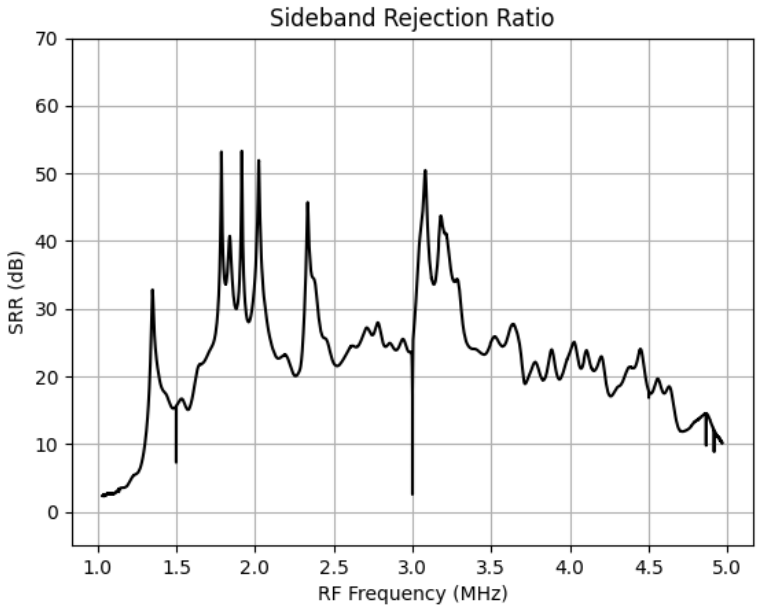


Figura C.12: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{12} y cantidad de acumulaciones 2^{7}.

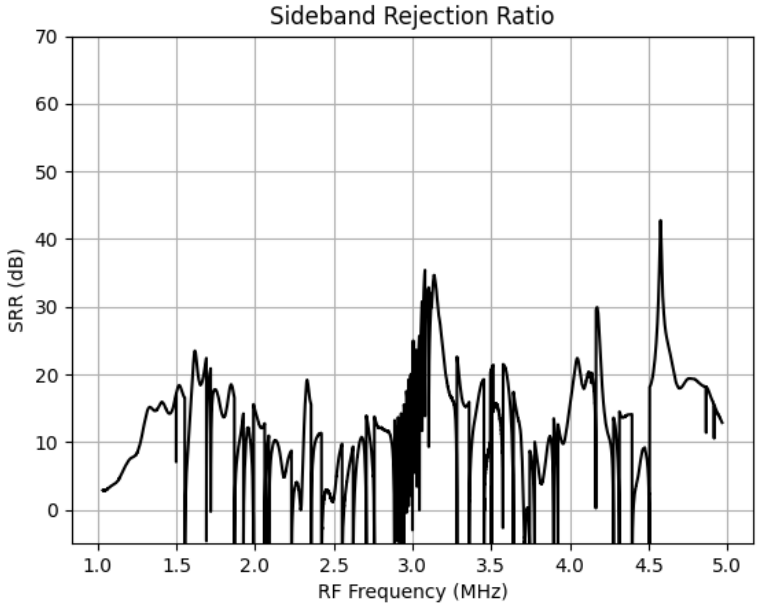


Figura C.13: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{13} y cantidad de acumulaciones 2^{10}.

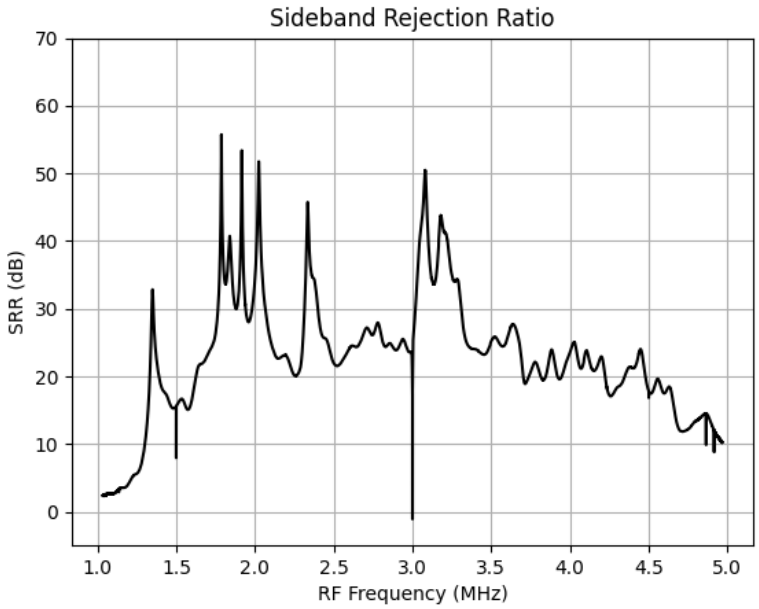


Figura C.14: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{13} y cantidad de acumulaciones 2^{6}.

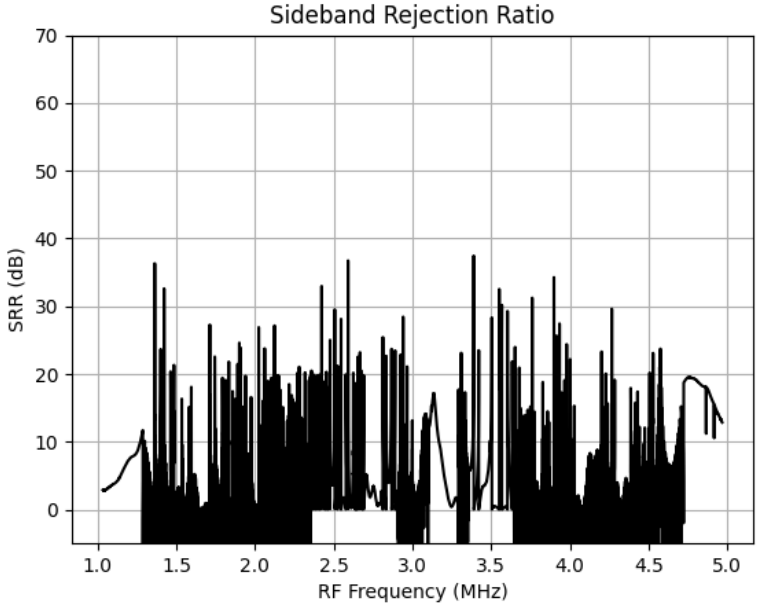


Figura C.15: SRR para el modelo de 32 bits y 8192 canales espectrales. Ganancia 2^{20} y cantidad de acumulaciones 2^{10}.

C.4. Mediciones adicionales de diferencia de fase

En las siguientes figuras, se presentan más resultados de los gráficos obtenidos de la diferencia de fase en las señales I y Q, con 2048 y 16384 canales espectrales y largo de acumulación de 2^{10}.

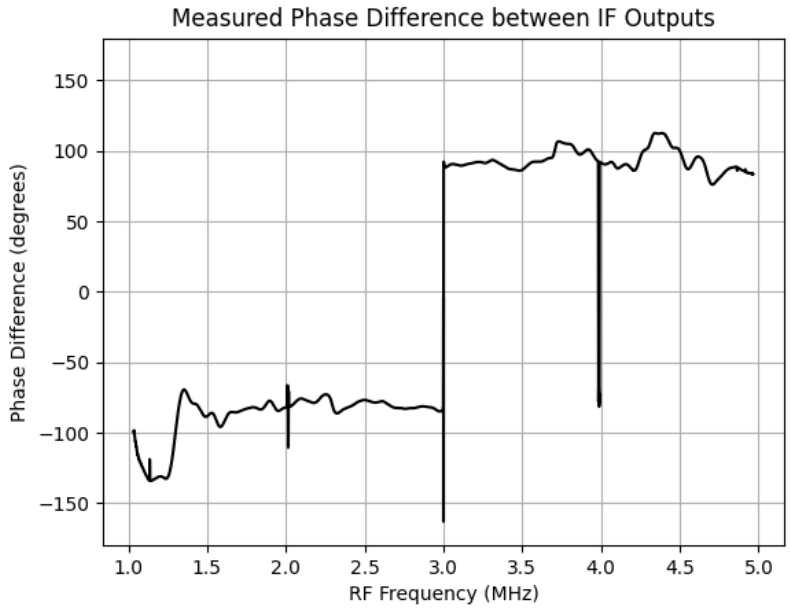


Figura C.16: Diferencia de fase entre las señales I y Q para el modelo de 2048 canales espectrales.

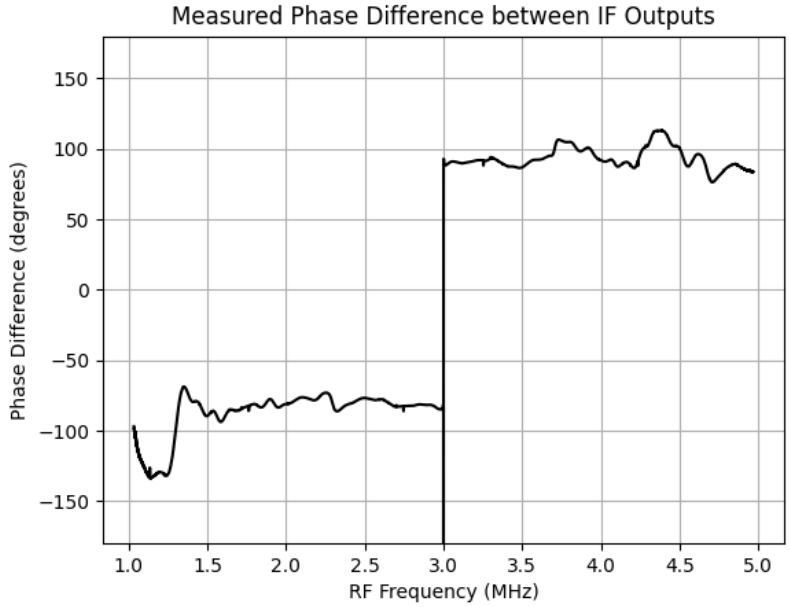


Figura C.17: Diferencia de fase entre las señales I y Q para el modelo de 16384 canales espectrales.

C.5. Gráficos con la latencia en lectura de datos con KATCP y librería de CASPER en Python

A continuación, se muestran los gráficos con la latencia en la toma de espectros del RFSoC usando el protocolo KATCP y la librería `casperfpga` de CASPER en Python:

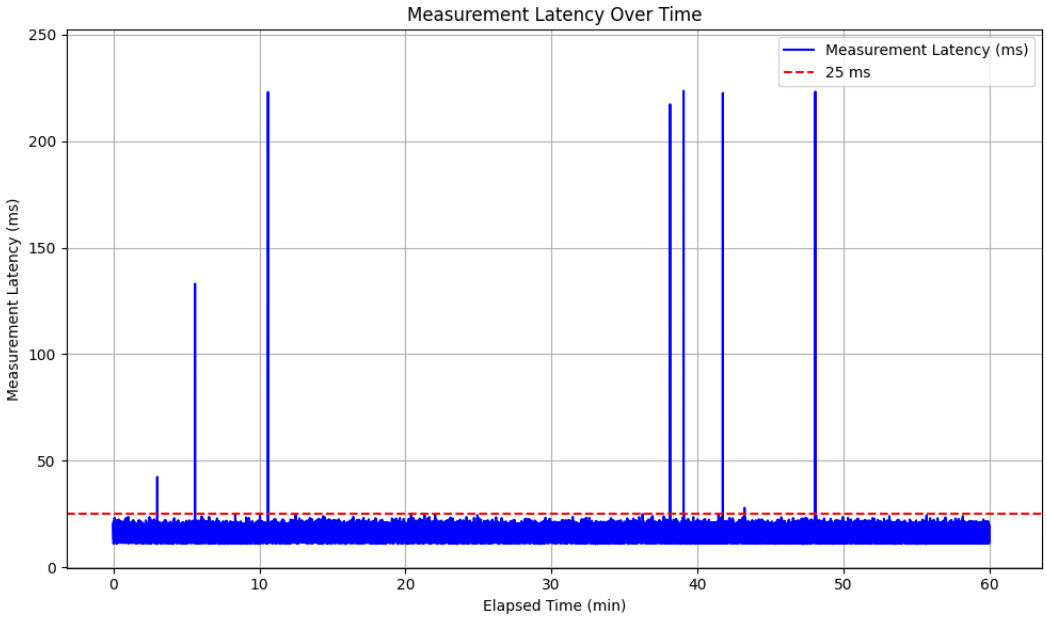


Figura C.18: Latencia de lectura de una banda de 512 canales espectrales usando el servidor de KATCP y `casperfpga`.

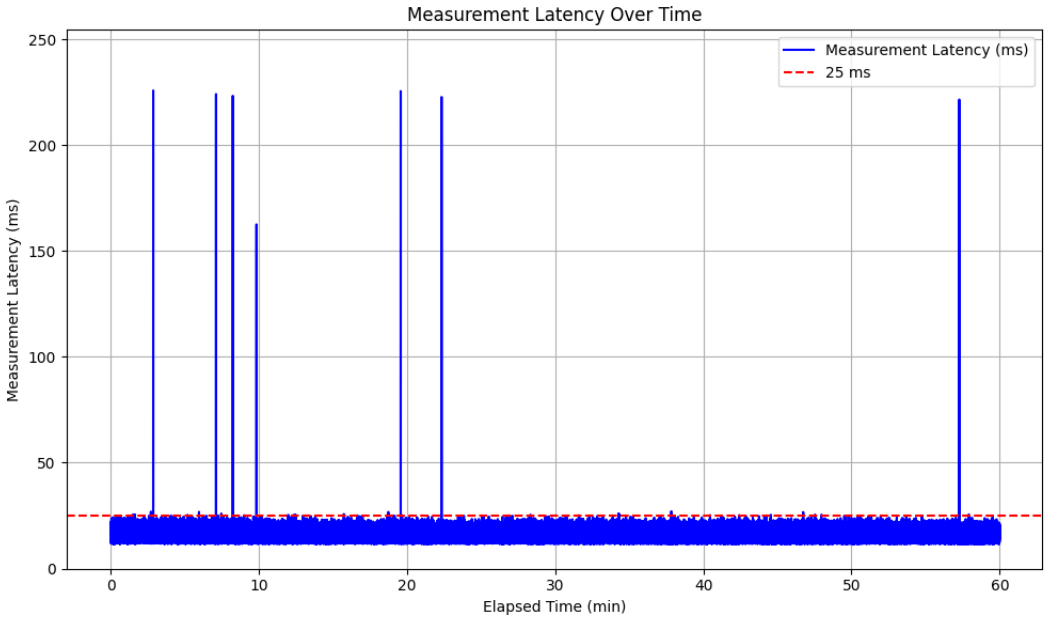


Figura C.19: Latencia de lectura de una banda de 1024 canales espectrales usando el servidor de KATCP y `casperfpga`.

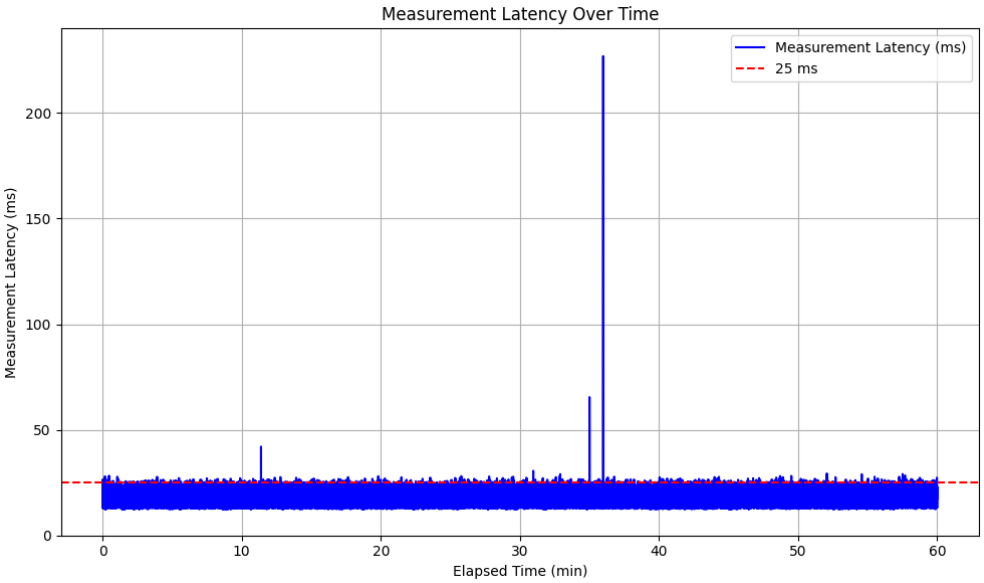


Figura C.20: Latencia de lectura de una banda de 2048 canales espectrales usando el servidor de KATCP y `casperfpga`.

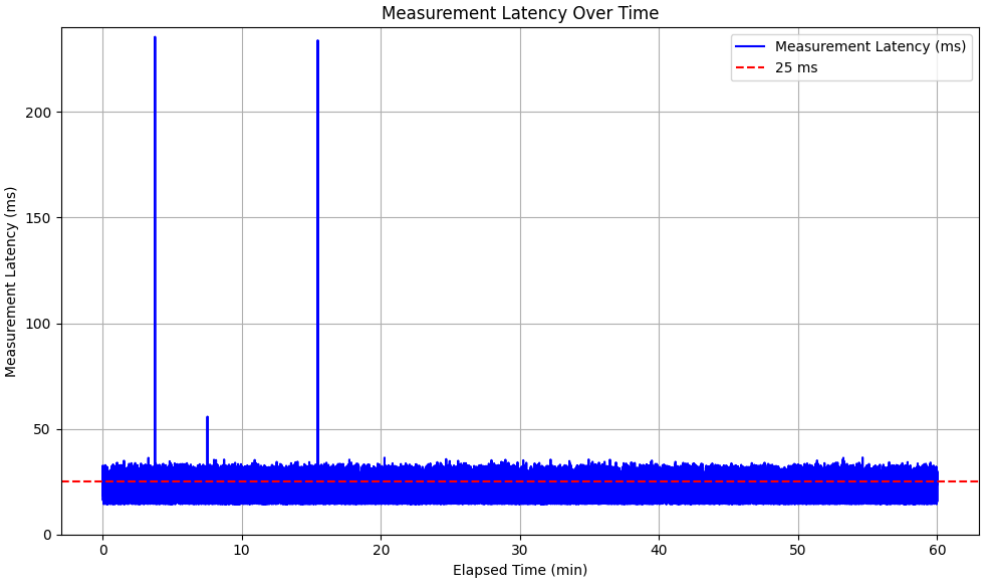


Figura C.21: Latencia de lectura de una banda de 4096 canales espectrales usando el servidor de KATCP y `casperfpga`.

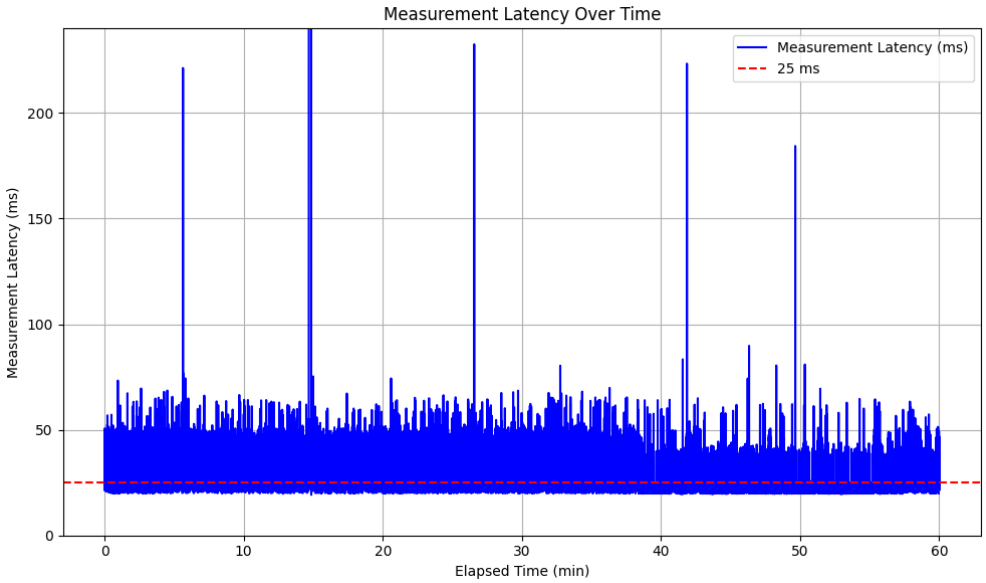


Figura C.22: Latencia de lectura de una banda de 8192 canales espectrales usando el servidor de KATCP y `casperfpga`.

C.6. Gráficos con la latencia en lectura de datos con servidor y clientes personalizados en C++ e interfaz en Python

A continuación se muestran los gráficos con la latencia en la toma de espectros del RFSoC usando el sistema cliente-servidor con C++ y lectura de datos con Python.

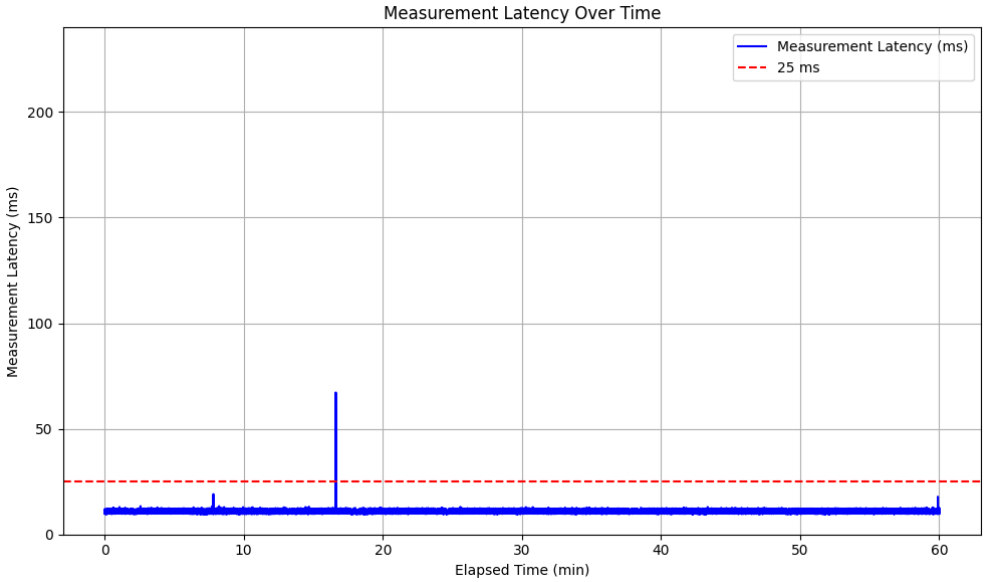


Figura C.23: Latencia de lectura de una banda de 512 canales espectrales usando servidor en C++ y Sockets de Python.

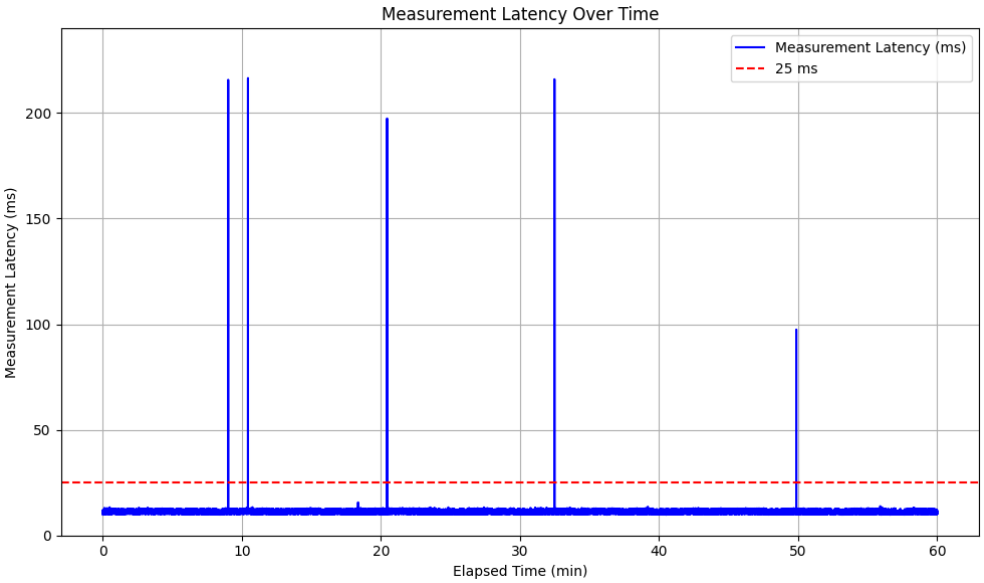


Figura C.24: Latencia de lectura de una banda de 1024 canales espectrales usando servidor en C++ y Sockets de Python.

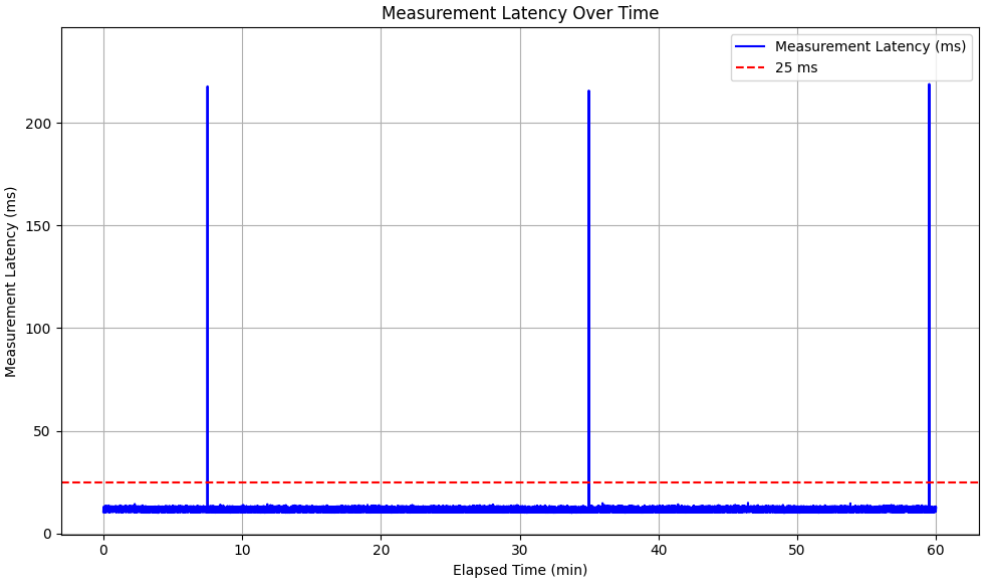


Figura C.25: Latencia de lectura de una banda de 2048 canales espectrales usando servidor en C++ y Sockets de Python.

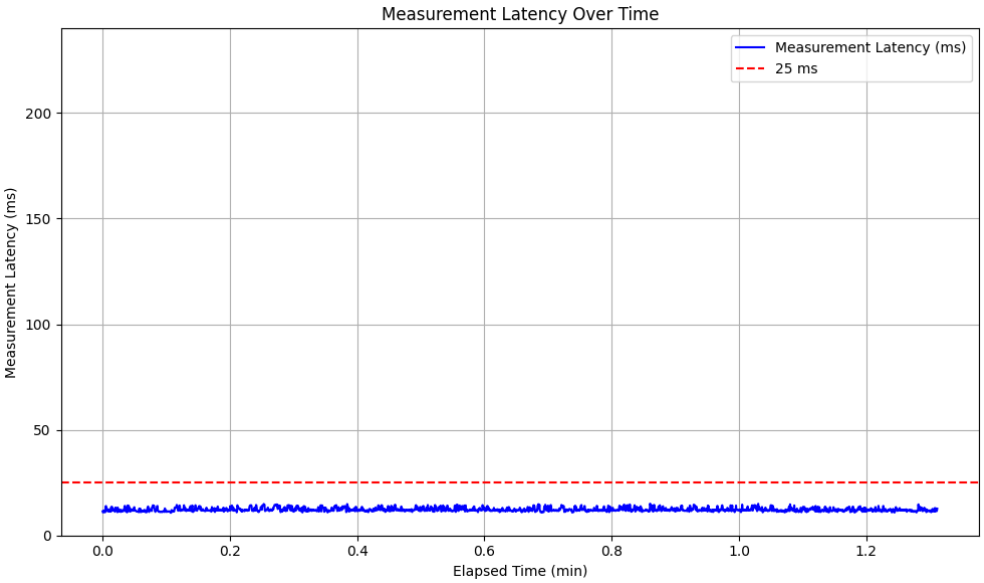


Figura C.26: Latencia de lectura de una banda de 4096 canales espectrales usando servidor en C++ y Sockets de Python.

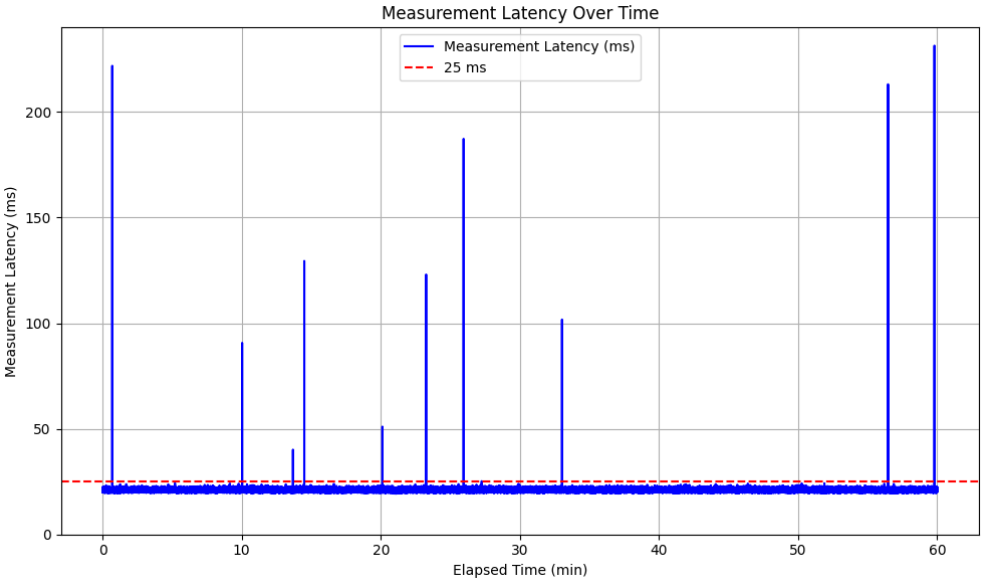


Figura C.27: Latencia de lectura de dos bandas de 512 canales espectrales usando servidor en C++ y Sockets de Python.

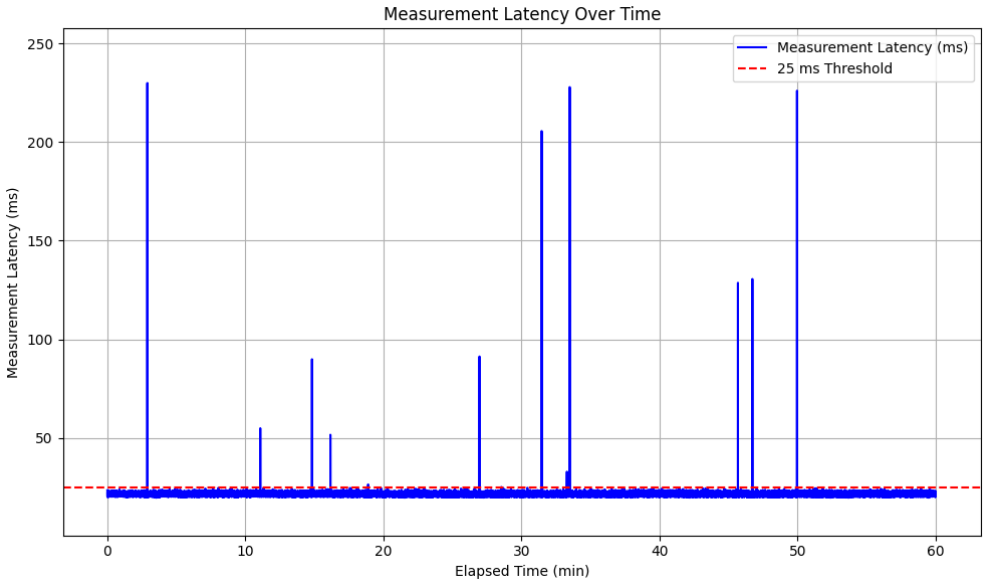


Figura C.28: Latencia de lectura de dos bandas de 1024 canales espectrales usando servidor en C++ y Sockets de Python.

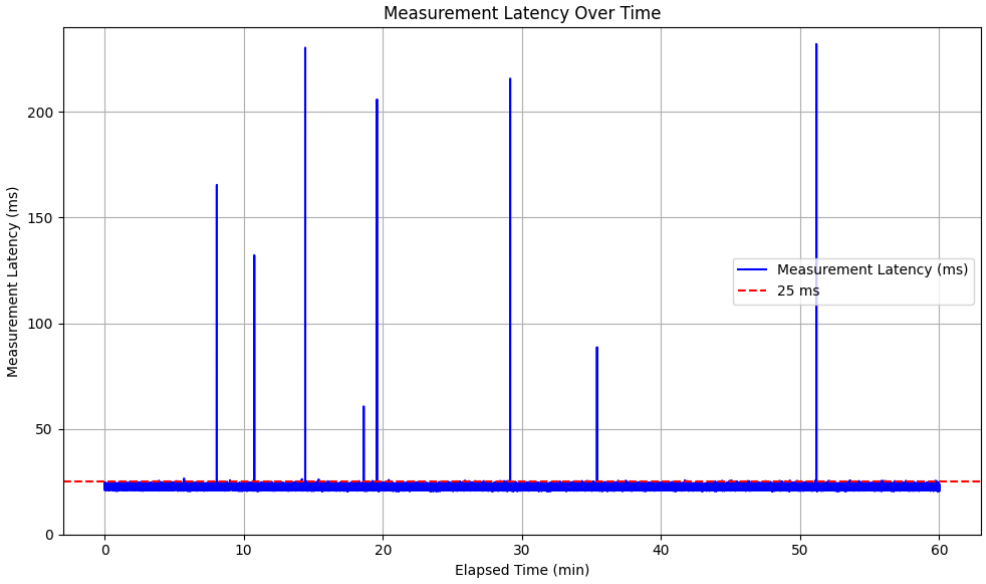


Figura C.29: Latencia de lectura de dos bandas de 2048 canales espectrales usando servidor en C++ y Sockets de Python.

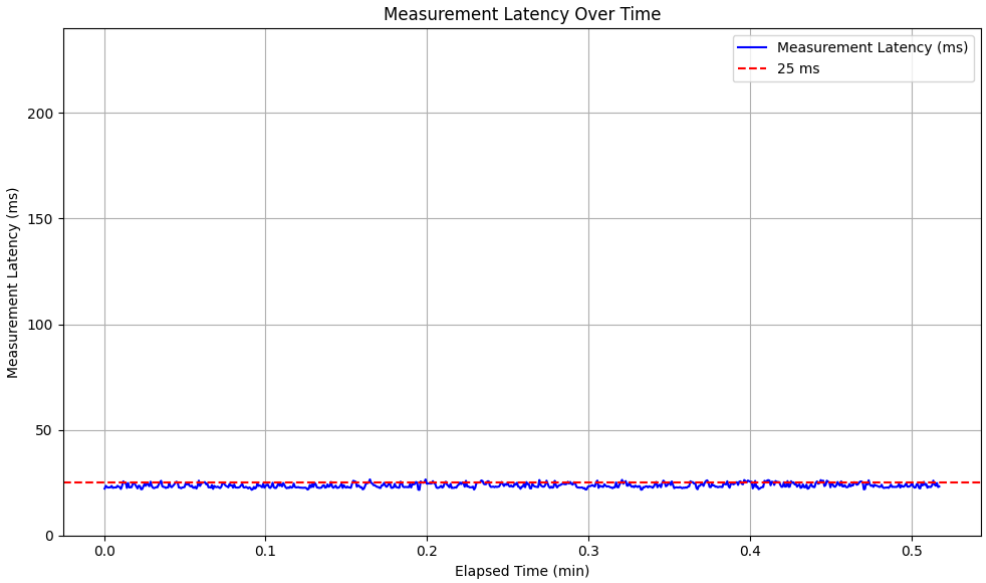


Figura C.30: Latencia de lectura de dos bandas de 4096 canales espectrales usando servidor en C++ y Sockets de Python.

Anexo D. Modelo completo de Simulink

Ver siguiente página

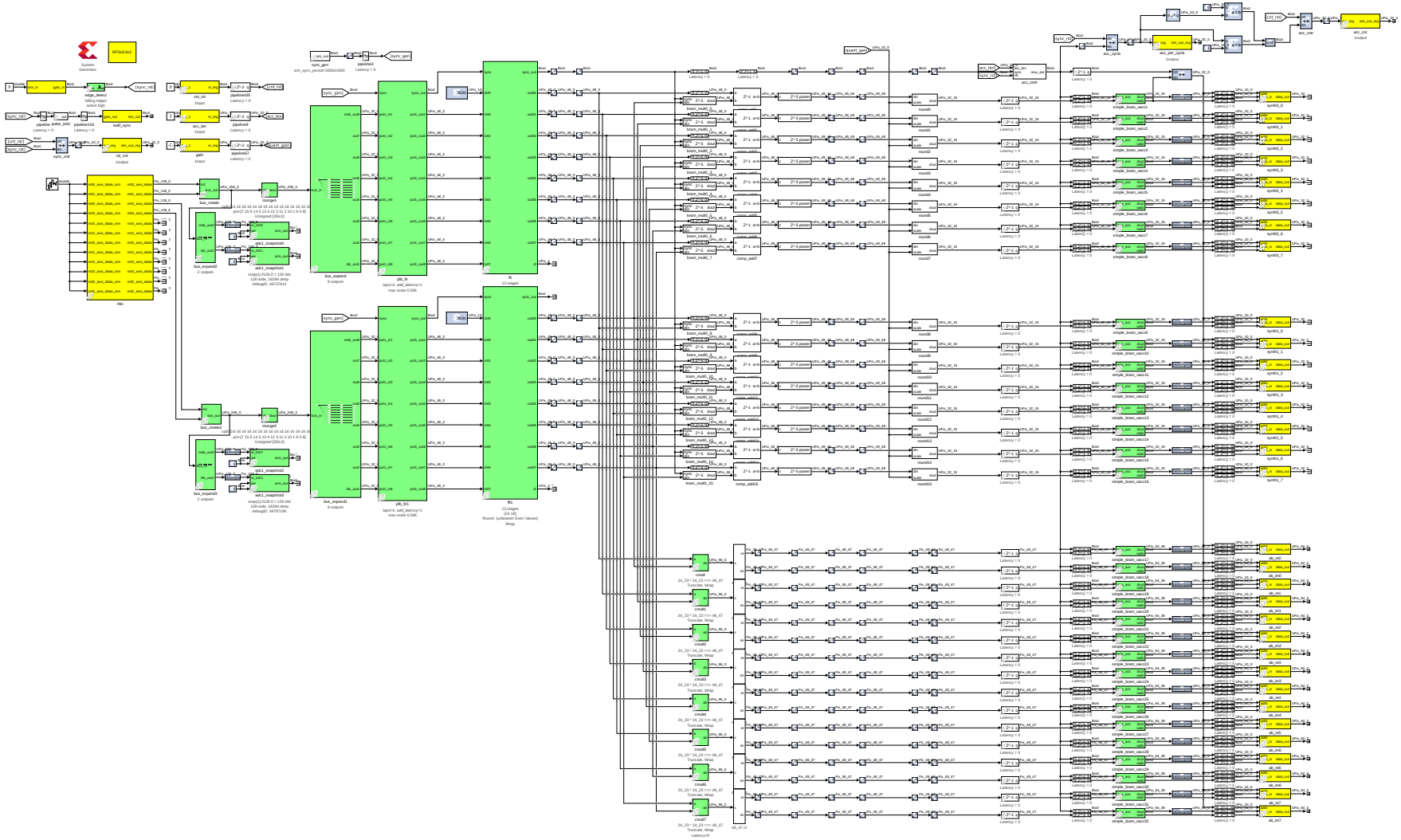


Figura D.1: Modelo del Espectrómetro Digital con Separación de Banda Lateral diseñado en Simulink para 8192 canales espectrales. Incluye bloques con correlación para medir la diferencia de fase, señal de control RESET y Re-Cuantización de bits para obtener espectros de 32 bits.