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

DETECCIÓN EN TIEMPO REAL DE RÁFAGAS DE RADIO RÁPIDAS, UTILIZANDO PROCESAMIENTO EN PARALELO EN UNA PLATAFORMA FPGA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

SERGIO VALERIANO SAAVEDRA TORRES

PROFESOR GUÍA:
RICARDO FINGER CAMUS

MIEMBROS DE LA COMISIÓN:
FAUSTO PATRICIO MENA MENA
WALTER MAX-MOERBECK ASTUDILLO

Se agradece el apoyo de CONICYT mediante sus fondos Basal PFB-06 y ALMA 31180005

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: SERGIO VALERIANO SAAVEDRA TORRES
FECHA: 2020
PROF. GUÍA: RICARDO FINGER CAMUS

DETECCIÓN EN TIEMPO REAL DE RÁFAGAS DE RADIO RÁPIDAS, UTILIZANDO PROCESAMIENTO EN PARALELO EN UNA PLATAFORMA FPGA

La radioastronomía es el campo que estudia las ondas de radiofrecuencia que emiten los cuerpos celestes. Dentro de la disciplina se encuentra el estudio de pulsos de corta duración, siendo uno de sus focos las Ráfagas de Radio Rápidas o *Fast Radio Bursts* (**FRBs**) ; señales de unas pocas milésimas de segundo hasta segundos de duración. Sus estimaciones de distancia, basadas en su medida de dispersión, sitúan las fuentes muy por fuera de nuestra galaxia, e implican eventos de gran energía dada la potencia recibida, no obstante, actualmente su origen es desconocido.

Históricamente los detectores de radiotelescopios no han estado adaptados para la detección de pulsos de tan corta duración. De hecho, el primer FRB fue detectado con datos históricos del radiotelescopio Parkes de 2007 y no fue hasta 2015 que se pudieron detectar en tiempo real, gracias al desarrollo de la electrónica digital y del uso de hardware especializado como GPUs (*Graphical Processing Units*) y FPGAs (*Field-Programable Gate Arrays*). Se estima que ocurren entre 1000 a 10.000 ráfagas de radio por día en todo el cielo, sin embargo, el catálogo oficial de las ocurrencias del fenómeno sólo posee algo más que una centena.

En este trabajo, se reporta el diseño, implementación y caracterización de un detector de ráfagas de radio rápidas en tiempo real en una plataforma FPGA (ROACH2), con capacidad para detectar múltiples medidas de dispersión en paralelo, con la novedad de incluir un bloque dedispersor parametrizado. Para ello, se ha desarrollado un banco de pruebas para generar FRBs sintéticas. Con ello, se desarrolló un dispositivo que analiza un ancho de banda de 540 MHz y es sensible a un amplio rango de medidas de dispersión de manera simultánea, el cual cubre la medida de las ráfagas reportadas a la fecha (de 100 a 2695 $\mathrm{pc\,cm^{-3}}$). Finalmente, y dada una detección, es posible disparar una descarga de los datos crudos para un análisis posterior.

¿A dónde van las nieblas, la borra del café,
los almanaques de otro tiempo?

Julio Cortázar

Agradecimientos

Es realmente extraño estar escribiendo estas palabras. El pregrado en la Universidad de Chile, en Beauchef, en Ingeniería Eléctrica no ha sido sencillo; fue un proceso que, por varios motivos, me ha costado sudor y lágrimas. Sinceramente, agradezco a toda la gente que estuvo en mi camino con su amistad, apoyo y ánimos, ustedes me han permitido llegar a esta instancia final.

A mis padres, que desde un comienzo se preocuparon que tuviera el privilegio de tener una buena educación, especialmente a mi madre, Patricia, quien desde un principio me apoyó a entrar a esta casa de estudios. A mi gr. hermano, Pablo. A mi abuela Isabel, que me quitó una vida de analfabetismo. A mis Abuelos, Nena por su cariño y Tata por inspirarme en las matemáticas. A mis tías, las que están y la que se recuerda.

A amigos de la vida; José, Bárbara y Carolina. A los trijarderos, especialmente a Andrés, Manuel, Gonzalo y Sebastián. A los des-estresantes y entretenidos Viernes con la gente de Salsalud. A mis amigos de eléctrica; los estudios y pizzas con Guillermo, las salidas a bailar con Laura, a Camila, Pí y Val. A ingratas e ingratos que han quedado en el camino.

A Fernanda y Valentina (¡y Mara!) sin su apoyo en momentos realmente difíciles no hubiera llegado a este punto. Me quedo con lo bueno.

Al grupo del laboratorio de ondas milimétricas de Cerro Calán, son un grupo humano increíble, colaborativo y con gran tema de conversación al almuerzo (¡me siento un nerd más! xD). Al team ROACH: Franco y Sebastián, sin su ayuda, ¡aún estaría compilando el espectrómetro!

A mi profesor guía, Ricardo Finger, por confiarme este proyecto y darme el apoyo para concretarlo desde el principio. A su curso Sistemas Digitales, que me ha enseñado más de lo que yo lo he impartido como ayudante y profesor auxiliar.

A mis correctores Patricio Mena y Walter M, por hacerme sufrir con sus observaciones que, finalmente, agradezco.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

La radioastronomía es el campo que estudia las ondas de radiofrecuencia que emiten los cuerpos celestes. Dicha emisión se percibe por medio de grandes antenas de radio conocidas como radiotelescopios, utilizando técnicas como interferometría y síntesis de apertura, así como radiotelescopios de plato único.

La existencia de pulsos de radio de corta duración fue predicha en los años setentas, atribuida a restos de supernovas en expansión, impactando el material adyacente [1]. Estas teorías motivaron búsquedas tempranas de este tipo de fenómenos en el telescopio de Arecibo con señales tan cortas como 16 ms. Si bien el ancho de banda y la resolución temporal era limitada, representó un primer esfuerzo en búsqueda de transientes extra galácticos de alta resolución temporal. A pesar de esto, no se detectaron pulsos astrofísicos de radio, pero se estableció un límite superior de la sensibilidad requerida para obtener señales de fuentes extragalácticas de radiofrecuencia [2].

Para entender cómo se descubrieron las ráfagas de radio rápidas o **FRBs** (por sus siglas en inglés) es importante mencionar los púlsares. Los cuales son un fenómeno de corta duración, pero de naturaleza periódica; consisten en estrellas de neutrones con rápida rotación que emiten pulsos de radio desde las líneas de campo magnético de sus polos.

El hecho de que los púlsares tengan un campo magnético gigantesco y estable los convierte en un acelerador de partículas natural que produce señales de radio con cierta coherencia, de un modo que aún no se conoce del todo [3]. A medida que la estrella de neutrones rota, los extremos de sus polos magnéticos lo hacen, permitiendo que el fenómeno se observe como pulsos periódicos de radiofrecuencia con un período de 0.1 a 1000 milisegundos.

Los pulsos de radio provenientes de púlsares experimentan un retraso temporal dependiente de la frecuencia, debido al paso en el medio interestelar ionizado o ISM (por sus siglas en inglés) en el que viajan. Dicho retraso, es cuantificado por una medida de dispersión o **DM** (por sus siglas en inglés) que es proporcional al número de electrones libres en la línea de visión. Esto resulta útil para medir el contenido de ionización del ISM y estimar la distancia de las fuentes.

Los primeros púlsares fueron encontrados, debido a sus singulares pulsos brillantes en el radio-observatorio Mullard en 1967 [4]. Dada la naturaleza periódica del fenómeno, dos años después, las búsquedas fueron optimizadas para utilizar los algoritmos de FFT (*Fast Fourier Transform*) y FFA (*Fast Folding Algorithm*). Este procedimiento resultó en el descubrimiento de una gran cantidad de púlsares en la galaxia y la caracterización de sus propiedades [5].

Las búsquedas de esta periodicidad han sido altamente satisfactorias, llevando el conteo de púlsares descubiertos a unas 2600 fuentes en 2018, debido a las mejoras en resolución temporal y en frecuencia.

Desde hace 50 años, cada nueva búsqueda de púlsares ha intentado expandir los parámetros de búsqueda: cobertura espacial, duración del pulso, DM, ciclo de trabajo, espectro y aceleración. Estas mejoras también contribuyeron, sorpresivamente, al descubrimiento de las FBRs.

En un intento de identificación de pulsos únicos en datos históricos del radiotelescopio Parkes, apuntando a la pequeña nube magallánica (SMC, por sus siglas en inglés), se identificó un pulso con características similares a un púlsar, sin embargo, su brillo ocasionó una saturación en el detector primario, lo que implica un máximo de densidad de flujo mayor a 30 Jy. Este pulso no sólo llamó la atención por su brillo, sino que su medida de dispersión estimaba ser ocho veces mayor a un pulso producido en la vía láctea, o en las inmediaciones de la SMC. Dado este descubrimiento, la 'ráfaga de Lorimer' sugirió la existencia de una nueva clase de pulsos extra galácticos de muy alto brillo [6], siendo ésta la primera FRBs documentada. El estudio de los FRBs generaron gran interés, pues dada su medida de dispersión, sólo un fenómeno de gran energía sería capaz de generar pulsos tan brillantes. La medida se corresponde con un retardo temporal de sus componentes de frecuencia debido al medio interestelar ionizado por el que viajan las ondas electromagnéticas [2].

A finales de 2018, la comunidad de investigadores no posee un consenso para la definición de un FRB. En la práctica, se identifica que una señal corresponde a una ráfaga rápida de radio si cumple ciertos criterios básicos, referentes a la duración del pulso, su brillo y su ancho de banda; siendo lo más relevante si su medida de dispersión efectivamente es mayor a la esperada para una fuente dentro de la galaxia [2].

A la fecha, la familia de FRBs conocidos consiste en más de 60 fuentes independientes detectadas en 10 telescopios y arreglos alrededor del mundo. Una recopilación de datos de ráfagas rápidas de radio se propuso en 2016, la cual puede ser consultada en línea [1] y que, actualmente, tiene un soporte para agregar datos en tiempo real [7].

Históricamente, la adquisición de datos en tiempo real de los radiotelescopios no ha permitido la identificación de los FRBs, debido a que el algoritmo que utilizan para procesar los datos, realizan un proceso de integración que filtra cualquier señal de corta duración. Últimamente, se han desarrollado nuevas técnicas para la detección de pulsos de corta duración, mediante el uso de unidades procesadoras gráficas o GPUs [8] (por sus siglas en inglés) y arreglos lógicos programables o FPGAs [9] (por sus siglas en inglés) en diferentes etapas del procesamiento.

Sin embargo, a la fecha de esta revisión (Agosto 2019), no existe una documentación sobre la implementación de la etapa de dedispersión (que corrige el efecto del medio estelar ionizado) con FPGAs. Es por esto, que la motivación de este trabajo es implementar y caracterizar un detector, con su correspondiente etapa de dedispersión, que aproveche el procesamiento paralelo y en tiempo real de una FPGA para la detección de FRBs para múltiples medidas

[1] FRB Catalogue - http://frbcat.org/

de dispersión.

Capítulo 1

Hipótesis y Objetivos

1.1. Hipótesis

Dada su alta capacidad de procesamiento paralelo y en tiempo real, un hardware tipo FPGA permitirá la detección de ráfagas rápidas de radio (FRBs, por sus siglas en inglés) para un amplio rango de medidas de dispersión.

1.2. Objetivo General

Desarrollar un detector de FRBs, con uso de procesamiento paralelo basado en FPGA.

1.3. Objetivos Específicos

- Desarrollar un banco de pruebas para generar FRBs sintéticos para la caracterización de los detectores
- Implementar un detector de FRBs que cubra un rango limitado de DM
- Caracterizar el detector implementado, usando el banco de pruebas antes mencionado
- Integrar varios detectores paralelos para cubrir el rango completo de DMs detectados a la fecha
- Integrar los subsistemas anteriores en un detector automático de FRBs listo para su despliegue en terreno

Capítulo 2

Marco Teórico

En este capítulo se presentan las consideraciones teórico-prácticas para el trabajo presentado. En primer lugar, se define la medida de dispersión o DM (por sus siglas en inglés), variable física clave de las ráfagas de radio rápidas y cómo ésta conlleva un retardo en las componentes de frecuencia de la señal. Se presenta el proceso de dedispersión de la etapa de detección el que supone revertir el efecto del medio de propagación y sus diferentes formas de implementación. Se presentan las ráfagas de radio rápidas identificadas a la fecha y los radiotelescopios implicados en su detección. Finalmente, se presenta el entorno de desarrollo (hardware y software) utilizado para la realización de este trabajo, destacándose el trabajo colaborativo de CASPER, desarrollador de la plataforma ROACH, versiones 1 y 2, utilizadas.

2.1. Medida de Dispersión (DM)

La medida de dispersión (DM, por sus siglas en inglés) es una de las métricas claves en la caracterización de las ráfagas rápidas de radio. Se define [2] como:

$$DM = \int_0^{\mathrm{d}} n_{\mathrm{e}} \, \mathrm{d}l \quad (\mathrm{pc\,cm^{-3}}) \tag{2.1}$$

donde n_{e} es la densidad de número de electrones del medio interestelar (ISM), dl es el diferencial de camino recorrido y d es el camino recorrido por el FRB.

En el contexto de astronomía de púlsares y FRBs, la medida de dispersión se representa en unidades de pársec por centímetro cúbico ($\mathrm{pc\,cm^{-3}}$) distancia[1] y densidad, respectivamente. Esto facilita el análisis numérico y la intuición al comparar distintos DMs [2]. A modo de ilustrar este hecho, las ráfagas rápidas de radio poseen un DM entre 100 ($\mathrm{pc\,cm^{-3}}$) y 2600 ($\mathrm{pc\,cm^{-3}}$), aproximadamente.

[1] 1 parsec $\approx 3{,}0857 \cdot 10^{16}$ metros

[2] Aún más, en caso de omitir la unidad, se asume que DM se encuentra en $\mathrm{pc\,cm^{-3}}$.

Es importante mencionar que, conocida una medida de dispersión, es posible estimar la distancia d a la que se encuentra la fuente, utilizando algún modelo para estimar el valor de n_e interestelar.

2.2. Tiempo de dispersión

Dada una onda de radio, de frecuencia ν, de ancho de banda infinitesimal, que se propaga por el medio interestelar desde una fuente a un destino. Su tiempo de propagación t puede ser modelado como [10]:

$$t = \frac{\mathrm{d}}{c} + \frac{\kappa\, DM}{\nu^2} \tag{2.2}$$

donde d es la distancia origen-destino, c es la velocidad de la luz en el vacío, κ es una constante y DM es la medida de dispersión del medio interestelar.

De la Ecuación 2.2 se desprende que las señales de menor frecuencia tienen un mayor tiempo de propagación que las de alta. Por tanto, dada una banda comprendida entre ν_h y ν_l un pulso temporal de gran ancho de banda se dispersará un tiempo Δt dado por:

$$\Delta t = t(\nu_l) - t(\nu_h) = \kappa(\nu_l^{-2} - \nu_h^{-2})DM \tag{2.3}$$

Dicho de otro modo Δt es el tiempo en que la onda de radio tardará en recorrer la ventana de frecuencia $[\nu_l, \nu_h]$.

A modo de ejemplo, en la Figura 2.1 se muestra la dispersión de una señal centrada en 1 GHz con un ancho de banda de 500 MHz y $\kappa = 4{,}15 \times 10^6$ ($\mathrm{MHz^2\ pc^{-1}\ cm^3\ ms}$). Los tiempos de llegada se han hecho coincidir con la frecuencia más baja de 750 MHz. En el gráfico se observa que para un $DM = 2000$ ($\mathrm{pc\ cm^{-3}}$) (curva en extremo izquierdo), el pulso de mayor frecuencia llegará ~ 9 segundos antes al receptor que su componente de baja frecuencia. En cambio, para un $DM = 100$ ($\mathrm{pc\ cm^{-3}}$) (extremo derecho) esto ocurrirá en aproximadamente 450 milisegundos.

Uno de los problemas en la detección de FRBs es que, por su naturaleza, dada una ventana de detección $[\nu_l, \nu_h]$, no se tiene conocimiento a priori de cuánto tiempo tardará en recorrer dicho espectro (i.e.: Δt) y, por lo tanto, se desconoce, también, su DM.

2.3. Dedispersión

Para un correcto análisis de los datos, se busca remover los efectos de propagación del medio interestelar (ISM), entre ellos, el tiempo de dispersión de una señal. Este proceso

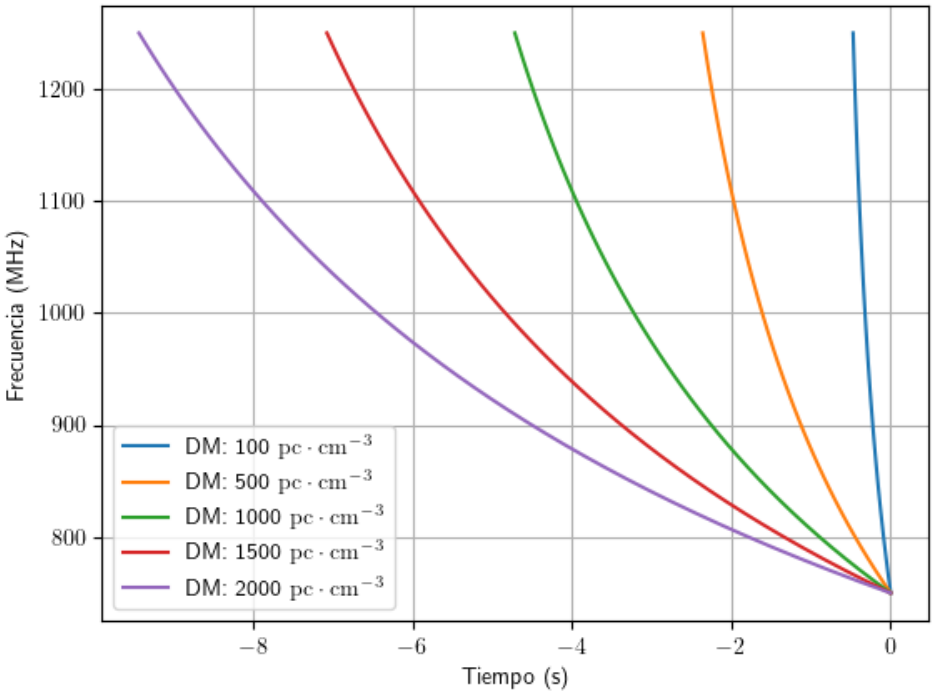

Figura 2.1: Dispersión de una señal centrada en 1 GHz con un ancho de banda de 500 MHz, para distintos DM entre 100 y 2000 (pc cm^{-3}). Se observa que para un $DM = 2000\,(\mathrm{pc\,cm^{-3}})$ (curva en extremo izquierdo), el pulso de mayor frecuencia llegará $\sim$ 9 segundos antes al receptor que su componente de baja frecuencia. En cambio, para un $DM = 100\,(\mathrm{pc\,cm^{-3}})$ (extremo derecho) esto ocurrirá en aproximadamente 450 milisegundos.

recibe el nombre de dedispersión. Dependiendo de la etapa en la que se realice el proceso puede ser coherente (o pre-detección) o incoherente (o de post-detección).

La dedispersión coherente [11] opera con señales de voltaje ($v(t)$), con información de amplitud y fase. Es requisito del procedimiento el muestreo a la frecuencia del ancho de banda total, lo que aumenta la cantidad de datos requeridos. Además, el proceso supone recuperar la señal de tiempo por medio de una deconvolución, asumiendo que el ISM funciona como una función de transferencia determinada.

La dedispersión incoherente [11] opera con señales de potencia (proporcionales a $v(t)^2$), canalizadas en un espectrómetro. Típicamente, después de la FFT, integradores promedian la potencia en cada canal espectral a intervalos de tiempo post-detección. Este tipo de dedispersión supone, luego, retardar temporalmente cada canal de frecuencia a fin de hacerlo coincidir con el comportamiento esperado de una medida de dispersión. Este procedimiento permite la minimización de supuestos y reduce la carga computacional asociada al problema, por tanto, es el enfoque adoptado para este trabajo.

La Figura 2.2 muestra una señal con DM = 2000 (pc cm^{-3}), centrada en 1000 MHz y con ancho de banda = 500 MHz. En ella se ilustra que si bien la señal puede tener una alta reso-

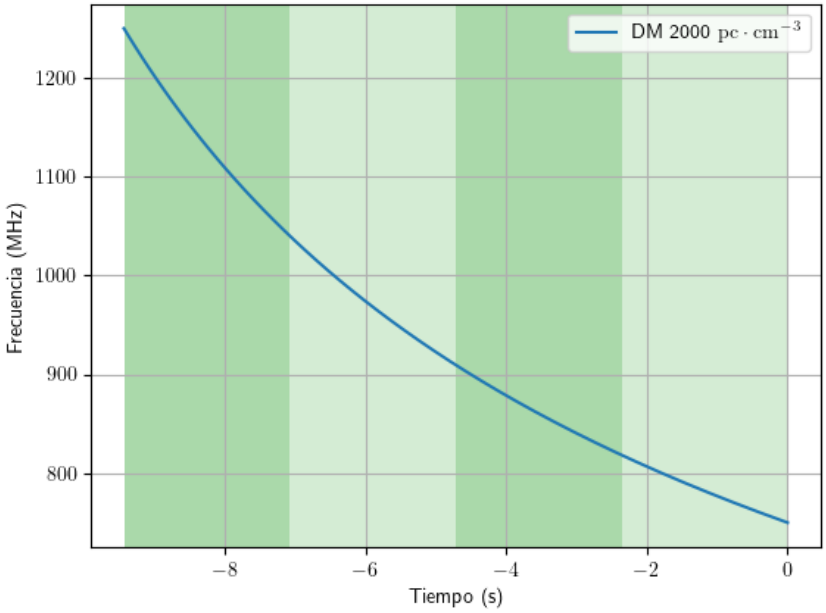

Figura 2.2: Dispersión de una señal con DM = 2000 (pc cm^{-3}), centrada en 1000 MHz y con ancho de banda = 500 MHz. En este caso, se han definido cuatro instantes de tiempo post-detección, sombreados alternadamente.

lución temporal, se puede definir un tiempo de integración, el cual reduce el ruido y cantidad de datos generados por el espectrómetro. Para simplificar la exposición, supondremos, en este caso, un tiempo de integración de 1.5 segundos, representado por las regiones sombreadas en la Figura 2.2.

2.4. Colección de ráfagas de radio rápidas a la fecha

A fin de dar un contexto actual sobre el estado de las búsquedas de FRBs se presenta a continuación el catálogo y los radiotelescopios implicados en las detecciones. En primer lugar, la colección actualizada de ráfagas de radio rápidas se encuentra en el sitio web *frbcat.org* [7]. En segundo lugar, en la Figura 2.3 se muestran los 110 FRBs verificados a inicio de 2020. Por otro lado, la Tabla 2.1 muestra la cantidad de detecciones por telescopio.

2.5. Entorno de Desarrollo

A continuación se presentan las principales herramientas de desarrollo (hardware y software) utilizadas para el desarrollo del detector. Se destaca la labor de CASPER quien desarrolla colaborativamente las plataformas FPGA utilizadas: ROACH 1 y ROACH 2. Información adicional se encuentra en la documentación de las mismas.

Tabla 2.1: Cantidad de detecciones de FRBs por telescopio (Revisión: Febrero 2020)

Telescopio	**Número de detecciones**
CHIME/FRB	30
ASKAP	28
Parkes	27
Pushchino	11
UTMOST	9
Arecibo	2
Apertif	1
DSA-10	1
GBT	1
Total	**110**

2.5.1. CASPER

CASPER (*Collaboration for Astronomy Signal Processing and Electronics Research*) [3] es un grupo colaborativo enfocado en crear soluciones electrónicas para el procesamiento de señales para la astronomía. Es responsable del diseño de varias placas de desarrollo basadas en FPGA (Figura 2.4). Lo anterior, con la filosofía de estandarizarlas para poder resolver problemas comunes de la comunidad astronómica mediante ellas, respetando un esquema de código abierto.

Las placas desarrolladas por CASPER vienen acompañadas de una serie de librerías con elementos parametrizados, también de codigo abierto, diseñados para resolver los problemas generales de la radioastronomía.

2.5.2. Plataforma FPGA: ROACH 1 y ROACH 2

ROACH 1 [4] es una plataforma de desarrollo basada en FPGA modelo Virtex 5 de Xilinx. Por otro lado, ROACH 2 [5] es la segunda generación de la placa basada en Virtex 6.

De la Figura 2.4 notamos la diferencia en recursos lógicos entre la primera y segunda generación. Sin embargo, tienen en común el uso de un PPC (Power PC) procesador independiente para funciones de control. Además, ambas utilizan la herramienta Xilinx ISE para compilar sus modelos. En efecto, CASPER ha agregado una serie de chips y periféricos para agregar funcionalidades necesarias en astronomía.

[3] https://casper.berkeley.edu/
[4] https://casper.ssl.berkeley.edu/wiki/ROACH
[5] https://casper.ssl.berkeley.edu/wiki/ROACH2

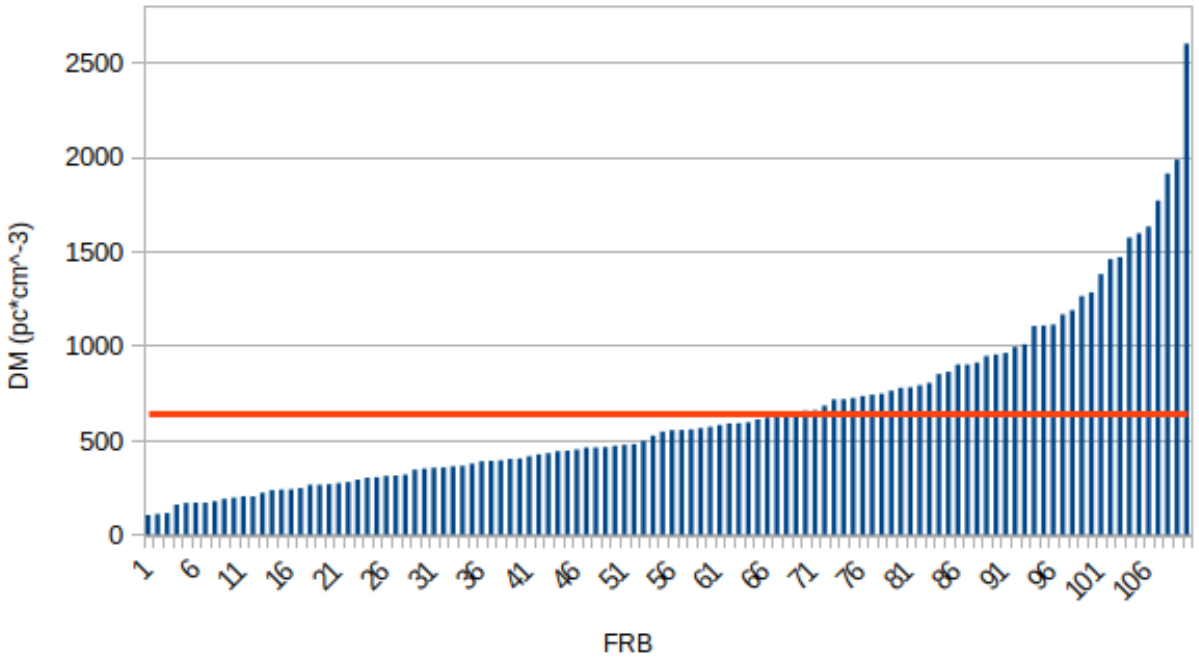

Figura 2.3: FRBs reportados y verificados a Febrero de 2020, ordenados por su medida de dispersión. La línea roja representa el valor medio de 647 ($\mathrm{pc\,cm^{-3}}$) (*Fuente: frbcat.org)*

2.5.3. Descripción de Hardware

Si bien existen los lenguajes de descripción de hardware (o HDLs) para construir y simular circuitos digitales. El principal método de creación de modelos de CASPER, es una interfaz gráfica basada en *Simulink*. *Simulink* es un entorno de programación visual que funciona sobre el entorno de programación *MATLAB* [6].

Si bien Xilinx, desarrollador de las FPGA "Virtex", provee una librería base para *Simulink* sus funciones son bloques lógicos básicos o "primitivos". CASPER ha tomado dichos bloques y combinándolos convenientemente ha creado sus "bloques amarillos": una colección de funciones de alto nivel donde destaca la transformada rápida de Fourier o FFT (por sus siglas en inglés), parametrizada para el usuario.

Por otro lado, el programador puede tomar cualquiera de los bloques disponibles, ya sea de CASPER o Xilinx, y crear nuevos bloques personalizados con nuevas funcionalidades adaptadas a sus requerimientos.

2.5.4. Compilador

Xilinx ISE es el software que crea y transfiere los circuitos diseñados al hardware FPGA, en este sentido funciona como compilador en segundo plano de la interfaz gráfica simulink.

[6]https://es.wikipedia.org/wiki/Simulink

CASPER Processing Platform Comparison Matrix						
Platform	**iBOB**	**ROACH**	**ROACH-2**	**SKARAB**	**SNAP**	**SNAP-2**
Status	***OBSOLETE***	Deprecated	Current	Current	Current	Current
FPGA	Virtex-II Pro XC2VP50	Virtex-5 V5SX95T	Virtex-6 XC6VSX475T	Virtex-7 XC7VX690T	Kintex-7 XC7K160T	Kintex Ultrascale XCKU115
Auxiliary processor	PPC	PPC	PPC	microblaze	Raspberry Pi	Zynq XC7Z010 ARM
Logic Cells	53k	94k	476k	693k	162k	1160k
DSP Slices	232	640	2016	3600	600	5520
BRAM	4.2 Mb	8.8 Mb	38 Mb	53 Mb	11 Mb	76 Mb
SDRAM	-	-	-	256Mb*	-	-
SRAM	2x18 Mb	2x36 Mb	4x144Mb	-	-	4x36 Mb
SRAM BW	9 Gbps	43 Gbps	200 Gbps	*HMC*	-	40 Gbps
DDR	-	1x8 Gb	1x16 Gb	*replaces*	-	1x 1GB DDR3
DDR BW	-	38 Gbps	50 Gbps	*SRAM and DDR*	-	8 bit interface
HMC Memory	-	-	-	*Up to 4 modules** @ 4GB ea*	-	-
HMC BW	-	-	-	*Ea module: 64Gbps R + W (128Gbps) (2 Links each @ 32Gbps R + W)*	-	-
High-speed Ethernet	2x10G	4x10G	8X10G	Up to 16x40G***	2x10G	4x40G, 16x10G
Low-speed Ethernet	1x10 Mbps	1x100 Mbps	2*1 Gbps	1x1 Gbps	1x1G on R-PI	2x 1G
Expansion Bus	2*ZDOK	2*ZDOK	2*ZDOK	4x Mega array 16x10 Gbps	1x ZDOK	2x HPC FMC, 1 ZD+
ADCs	-	-	-	-	3x HMCAD1511 (12 inputs)	-
Xilinx Tools Required	ISE	ISE	ISE	Vivado	Vivado	Vivado
				* Currently used for dynamic FPGA reconfiguration ** Toolflow supports 3 x 32 Gb and up to 3 modules *** Toolflow supports 1 x 40GbE		

Figura 2.4: Placas de desarrollo basadas en FPGA de CASPER (Fuente: CASPER). Se observa que el hardware ROACH 1 se encuentra sin soporte, sin embargo, posee recursos comparables a la placa SNAP, aún en uso. ROACH 2, se posiciona en gama media, siendo la única placa que no utiliza el sistema Vivado de Xilinx.

2.5.5. Comunicación

La comunicación con el hardware se lleva a cabo en alto nivel por la librería de Python "Corr" [7]. Esta librería es utilizada como un entorno de control de propósito general para ROACH 1 y ROACH 2.

2.6. Resumen

La medida de dispersión (DM, por sus siglas en inglés) es la principal variable física de un FRB. Se produce porque el espacio no es perfectamente vacío y posee una densidad de electrones (n_e). Que un FRB tenga dispersión produce que el tiempo de arribo de las frecuencias altas y bajas a un receptor de FRBs sean más dispares a mayor DM.

Para detectar FRBs es necesario corregir la diferencia temporal producida por el ISM, con esto se recupera la señal original en su fuente. El proceso para lograrlo se denoomina dedispersión. En este trabajo se utilizará la dedispersión incoherente (o post-detección) que supone trabajar con señales de potencia, integrando en ciertos instantes de acumulación y corrigiendo en ventanas de tiempo-frecuencia.

Los FRBs detectados a la fecha pueden ser consultados en una plataforma online. Nueve

[7] https://casper.ssl.berkeley.edu/wiki/Corr

radiotelescopios han reportado los ciento diez fenómenos identificados.

El entorno de hardware y software utilizado en este trabajo se basa en la colaboración de CASPER. El grupo ha desarrollado el hardware ROACH 1 y ROACH 2 utilizado. El software utilizado es *Simulink*. El grupo colaborativo ha dispuesto asimismo, bloques funcionales para el software que implementan funciones comunes dentro de la radioastronomía como la transformada rápida de Fourier (FFT).

Capítulo 3

Metodología

En primer lugar, es importante saber utilizar el hardware y software que se utilizará replicando tutoriales básicos de CASPER y del laboratorio de ondas milimétricas de la Universidad de Chile. En segundo lugar, se presenta cómo se realizará la emulación de un FRB y el circuito analógico, banco de pruebas, encargado de ello. En tercer lugar, se presenta la metodología de diseño que expone conceptualmente los bloques funcionales del detector que más tarde serán implementados en ROACH 1 y ROACH 2, poniendo especial énfasis en el bloque dedispersor que será diseñado desde un bajo nivel y parametrizado; una poderosa idea para el diseño de sistemas digitales presentada más adelante. Asimismo, se exponen las métricas relevantes a caracterizar del detector una vez implementado. Finalmente, se presenta cuál será la estrategia para implementar el paralelismo, esto es, tener la capacidad de dedispersar más de un DM simultáneamente.

3.1. Primeros pasos en la plataforma ROACH

Los primeros pasos de aprendizaje en la plataforma ROACH consisten en replicar los tutoriales básicos de CASPER[1] y del grupo del laboratorio de ondas milimétricas[2]. Ellos abarcan las funciones básicas esenciales que puede ejecutar la plataforma de desarrollo, y otras que se utilizarán más adelante en el detector, tanto para su versión definitiva como en corrección de errores.

3.1.1. Tutorial 1: Funciones básicas

En el primer tutorial [3], se busca familiarizar el entorno de desarrollo con el usuario. Contempla la realización del "Hola Mundo" del hardware que es programar el prender un LED. Puede sonar sencillo, sin embargo, esto implica una correcta configuración del hardware

[1] `https://casper-tutorials.readthedocs.io/`

[2] `https://sites.google.com/site/calandigital`

[3] `https://casper.ssl.berkeley.edu/wiki/Introduction_to_Simulink_ROACH2`

y sofware y una correcta comunicación entre ROACH y un ordenador. Se prueban lecturas y escrituras de registros con la librería Python: "Corr".

3.1.2. Tutorial 2: Snapshot

En el segundo tutorial [4] se implementa un modelo con el bloque "Snapshot" el cual sirve para ver los datos inmediatamente a la salida del conversor análogo digital o ADC (por sus siglas en inglés).

3.1.3. Tutorial 3: Espectrómetro

En el tercer tutorial [5] se implementa un espectrómetro digital. Se utiliza el bloque de FFT de CASPER y sus correspondientes señales de sincronización y validez.

3.2. Emulación de un FRB

Para emular un FRB, se considera su característica distintiva, esto es, tener una medida de dispersión que implica que su fuente de emisión es extragaláctica. Lo anterior puede ser interpretado como un pulso moviéndose desde una alta a una baja frecuencia.

Asimismo, se asume que la señal a imitar corresponde a la recibida por el *backend* de un telescopio, por lo tanto, ya ocurrió una etapa de *downconversion* de la señal original para poder estar dentro del rango del conversor análogo digital del hardware de procesamiento.

En síntesis, un FRB será modelado como lo expuesto en la Figura 3.1, un pulso gaussiano que se desplaza desde las componentes de alta a baja frecuencia. El área sombreada representa el ancho de banda del conversor análogo digital de ROACH. El circuito para construir esta señal se presenta en la sección 3.3, a continuación.

3.3. Banco de Pruebas: Creación de un FRB sintético

Para poder probar el detector, es necesaria la construcción de un banco de pruebas para emular el comportamiento de una ráfaga rápida de radio. En la Figura 3.2 se muestra la configuración propuesta para tal efecto.

Un elemento esencial del montaje es el generador controlado por voltage. En su salida, el VCO entrega una onda sinusoidal a una frecuencia dependiente de su tensión de sintonización

[4] `https://sites.google.com/site/calandigital/tutorials/snapshot-tutorial`
[5] `https://casper.ssl.berkeley.edu/wiki/Wideband_Spectrometer`

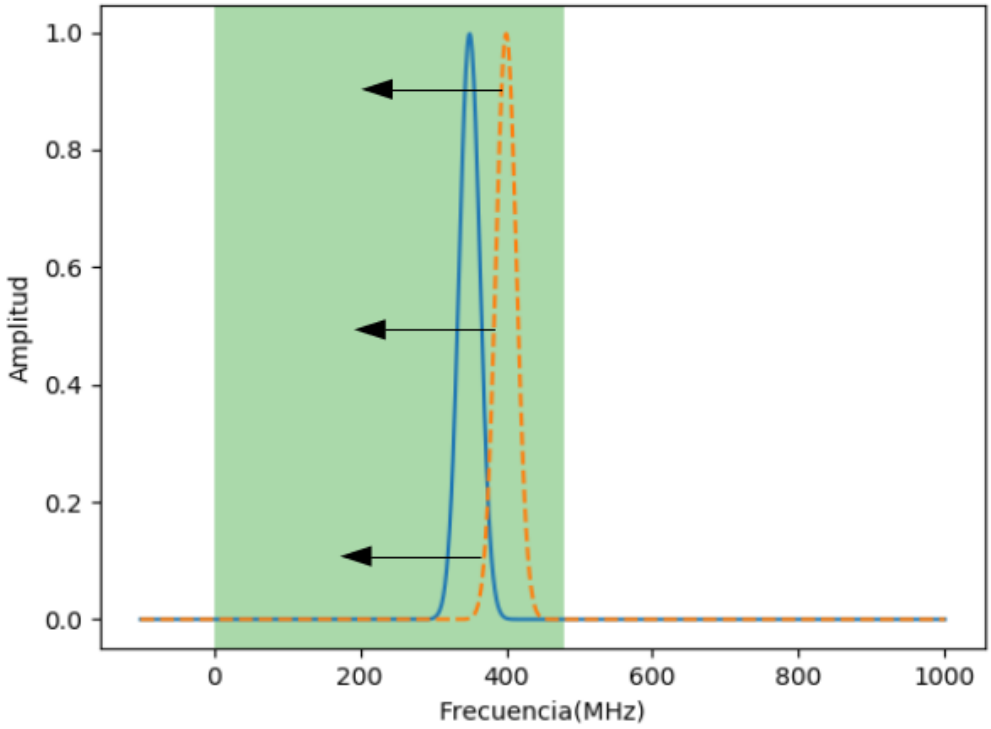

Figura 3.1: Modelamiento de un FRB en banda base. Se asume que la ráfaga es una señal Gaussiana que se mueve a través del espectro. El área sombreada representa el ancho de banda del conversor análogo digital (ADC) disponible.

(V_{tune}). Luego, si se entrega un valor de V_{tune} variable, se podrá tener una onda variable en frecuencia en el tiempo, tal como un FRB dispersado.

La siguiente etapa está compuesta por un mezclador, donde ocurren dos efectos a destacar. El primero, es el traslado en frecuencia de la señal a una banda adecuada para la digitalización en ROACH (downconversion). El segundo, es que se modula una onda proveniente de un generador de señales el cual genera una señal de espectro ancho de forma Gaussiana, la cual es modulada por la señal proveniente del VCO, esto supone el último ingrediente para obtener un FRB sintético. Finalmente, se añade una fuente de ruido blanco al sistema, simulando el ruido presente en toda observación astronómica.

3.4. Diseño del Detector

Para el diseño del detector se parte por una arquitectura propuesta por comunicación interna, que se basa en la dedispersión incoherente, buscando minimizar la carga computacional a fin de implementar varios detectores en paralelo en una etapa posterior.

El detector recibe la señal del FRB emulado, la digitaliza, la procesa (incluyendo el proceso

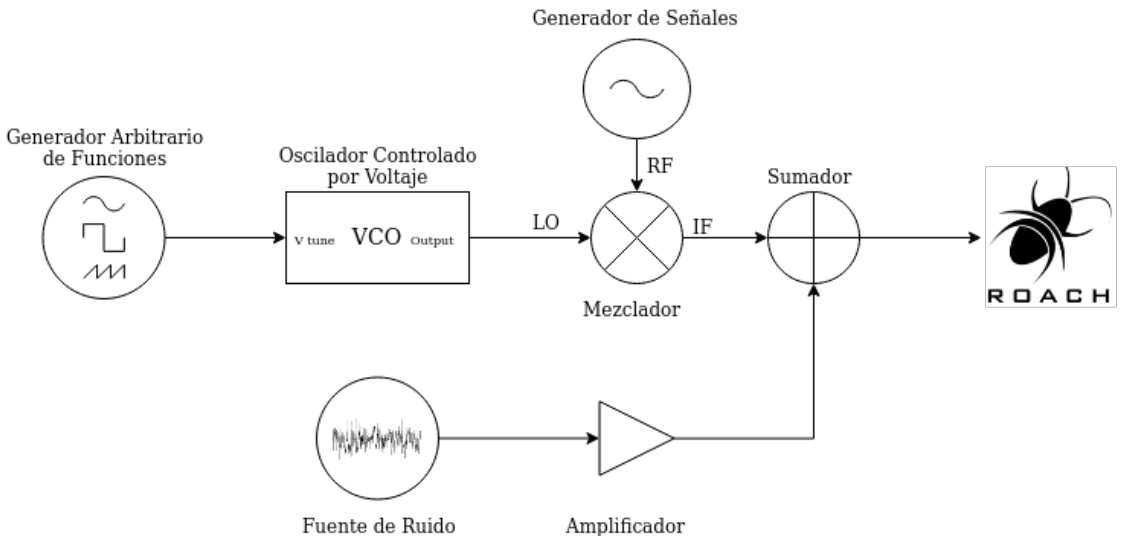

Figura 3.2: Diagrama propuesto del banco de pruebas. La primera rama consiste en un generador arbitrario de funciones (AWG, por sus siglas en inglés) que controla un oscilador controlado por voltaje (VCO, por sus siglas en inglés) esto entra a un mezclador junto a la señal de un generador de señales. La segunda rama es una fuente de ruido con un amplificador. Ambas señales se suman para luego entrar a la digitalización de ROACH.

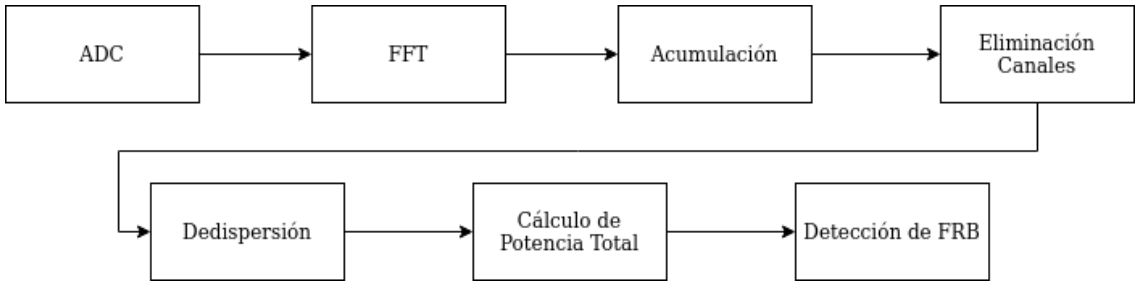

Figura 3.3: Diagrama de funcionamiento de la FPGA, inicia con el ADC y termina con la Detección de FRB. El detalle de cada bloque funcional se puede leer en el texto.

de dedispersión) y finalmente comunica si existe una detección válida. Lo anterior es válido para una medida de dispersión determinada.

3.4.1. Diagrama General

En la Figura 3.3 se muestra un esquema general de los procesos ejecutados por el hardware de la FPGA para hacer la función del detector propuesto. Brevemente, la función de cada etapa se describe a continuación.

- ADC (*Analog to Digital Converter*): es el conversor análogo a digital. Convierte una señal continua a discreta (en tiempo y amplitud) con una cierta tasa de muestreo. De aquí en adelante la información se interpreta en números binarios o *bits*.
- FFT: Algoritmo de la transformada rápida de Fourier. Transforma datos en el dominio

temporal al dominio de la frecuencia. En este caso, se considera la potencia de la señal.

- Acumulación: El valor de la potencia en cada canal se integra en esta etapa, esto es, se toma la suma entre la contribución actual para un canal, saliendo de la FFT, y los valores anteriores.
- Eliminacion de Canales: En esta etapa se eliminan ciertas componentes de frecuencia atribuibles a fuentes conocidas de ruido, en canales definidos, ya sea RFI (*Radio Frecuency Interference*) o atribuible a los instrumentos.
- Dedispersión: En esta etapa se reordenan las muestras de frecuencia de cada canal espectral a distintos tiempos, a fin de remover la dispersión de los datos que ingresan al detector.
- Cálculo de potencia total: Los espectros dedispersados de la etapa anterior son sumados en un valor correspondiente al flujo total de potencia.
- Detección de FRB: Se compara la potencia del flujo total de la entrada con cierto valor umbral. Si es superado, se considera una detección válida.

3.4.2. Principio de funcionamiento del dedispersor

Antes de presentar su implementación en *Simulink*, es importante dar a entender el principio de funcionamiento del dedispersor. Para simplificar el análisis, se asume que existe un único puerto de entrada que recibe una única muestra en cada ciclo de integración. Se conoce a priori la cantidad de canales espectrales que posee la FFT, suponiendo que son cuatro canales: A,B,C y D. Luego, los datos de entrada DD_{input} pueden ser convenientemente escritos como:

$$DD_{input} = A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2, ... \tag{3.1}$$

Los índices representan los canales antes definidos y los subíndices representan la ventana de tiempo en que se capturó la totalidad del espectro. Así, por ejemplo, si queremos representar la totalidad del espectro en un tiempo de integración $t = 0$, necesitamos la información A_0, B_0, C_0 y D_0.

Por simplicidad, supongamos que el DM de la Figura 2.2 produce una dispersión lineal en el tiempo. Luego, manteniendo el tiempo de integración y considerando cuatro canales, se tiene la curva de la Figura 3.4. Notamos que para dedispersar la señal, la potencia promedio de cada canal debe ser llevada al mismo instante temporal (representado con flechas celestes), esto implica recordar dicha potencia. El canal de 1100 a 1200 MHz, debe recordarse, por lo menos, tres tiempos de integración. El canal siguiente, de 1000 a 1100 MHz, debe recordarse dos. El tercero, de 900 a 1000 MHz, uno. El último canal es el tiempo real y base para la dedispersión.

A continuación, correspondemos los canales A, B, C y D, con 1100 a 1200, 1000 a 1100, 900 a 1000 y 800 a 900 MHz, respectivamente. Luego, consideramos el esquema de la Figura 3.5, que introduce el modelo de pilas que ilustra el principio de funcionamiento del dedispersor. En la entrada serial se pregunta paralelamente cuál es el canal actual (flechas azules), si se

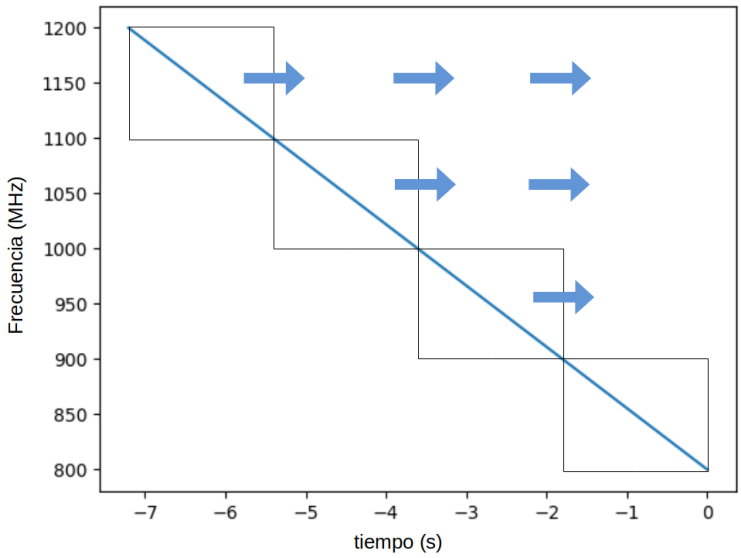

Figura 3.4: Linealización de la señal dispersada de la Figura 2.2. Se ha conservado el tiempo de integración post-detección definiendo, además, cuatro canales espectrales. Notamos que para dedispersar la señal la potencia promedio del canal de 1100 a 1200 (MHz), debe recordarse dicho valor, por lo menos, tres tiempos de integración. En el canal siguiente de 1000 a 1100 (MHz), debe recordarse dos. En el tercero de 900 a 1000 (MHz), uno.

corresponde se le asigna el primer valor en la pila "Empujando" a los anteriores hacia arriba. Los valores en la casilla superior de cada pila son seleccionados (en secuencia de 1 a 4, flechas naranjas) y reinterpretados como los nuevos canales, esta vez, dedispersados.

La Figura 3.6 muestra la secuencia de llenado del modelo de pilas de la Figura 3.5. La figura se lee de izquierda a derecha, arriba a abajo. Cada cuadro representa la transición de recibir cuatro datos seriales y se muestra la salida correspondiente al mismo período. Los valores no definidos se han marcado con la letra "x", típicamente estos valores son nulos al inicializar el programa. Notamos que la cuarta transición de la Figura 3.6 corresponde a interpretar en la salida: $A'_3 = A_0$, $B'_3 = B_1$, $C'_3 = C_2$ y $D'_3 = D_3$. Si consideramos que la primera ventana temporal de la Figura 3.4 se corresponde a $t = 0$, hemos dedispersado tal como se buscaba.

La salida serial (válida) del dedispersor DD_{output} será, entonces:

$$DD_{output} = S_3, S_4, S_5, ... = A'_3, B'_3, C'_3, D'_3, A'_4, B'_4, C'_4, D'_4, A'_5, B'_5, C'_5, D'_5, ... \tag{3.2}$$

En la Figura 3.7 se resume la explicación anterior. La entrada de los datos se corresponde a las columnas de la figura. La salida del dedispersor corresponde a los óvalos en color azul desde S_3 en adelante. En este sentido, la dedispersión ocurre de manera continua, espectro a espectro y en tiempo real. Esto último es relevante pues no se conoce a priori en qué instante ocurrirá el FRB.

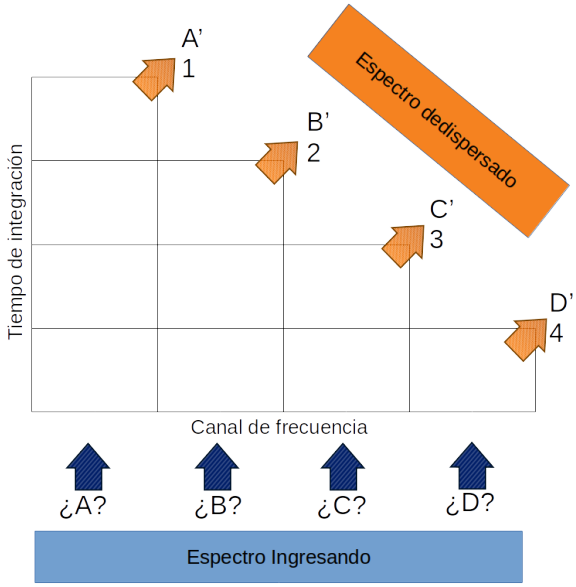

Figura 3.5: Modelo de pilas para ilustrar el principio de funcionamiento del dedispersor. En la entrada serial se pregunta paralelamente cuál es el canal actual (flechas azules), si se corresponde se le asigna el primer valor en la pila "Empujando" a los anteriores hacia arriba. Los valores en la casilla superior de cada pila son seleccionados (en secuencia de 1 a 4, flechas naranjas) y reinterpretados como los nuevos canales, esta vez, dedispersados.

3.4.3. Parametrización del Detector

Uno de los aspectos esenciales del detector a construir es la posibilidad de su adaptación a cualquier medida de dispersión y a un número arbitrario de canales de frecuencia que reciba de la FFT. Para esto, se recurre a la parametrización.

La parametrización de un bloque o sistema, consiste en "enmascarar" su funcionamiento interno al usuario/programador a quién solo le interesa su relación entrada/salida, regida por ciertos parámetros predefinidos. Esto permite crear niveles sucesivos de abstracción y es una de las herramientas más potentes de los sistemas digitales. Por ejemplo, en la Figura 3.8 se muestra la abstracción de un multiplexor al lado derecho, donde basta fijar el parámetro S a 0 ó 1, para conocer su salida A ó B. El usuario no necesita conocer el funcionamiento interno, mostrado a la izquierda, para utilizar el bloque, aún más, el bloque puede ser usado incluso como bloque funcional de otros bloques a mayor escala.

El proceso en la plataforma MATLAB/Simulink sigue la misma lógica de abstracción. Los parámetros de cierto bloque pueden ser modificados dentro de la "máscara"del mismo. Por otro lado, para poder establecer la funcionalidad interna existen dos procesos:

- La guía de "Simulink Mask Scripting"de CASPER [12].
- Herramienta oficial xBlock de Xilinx [13].

Cada herramienta tiene sus pro y contras. La primera permite manejar en mayor detalle

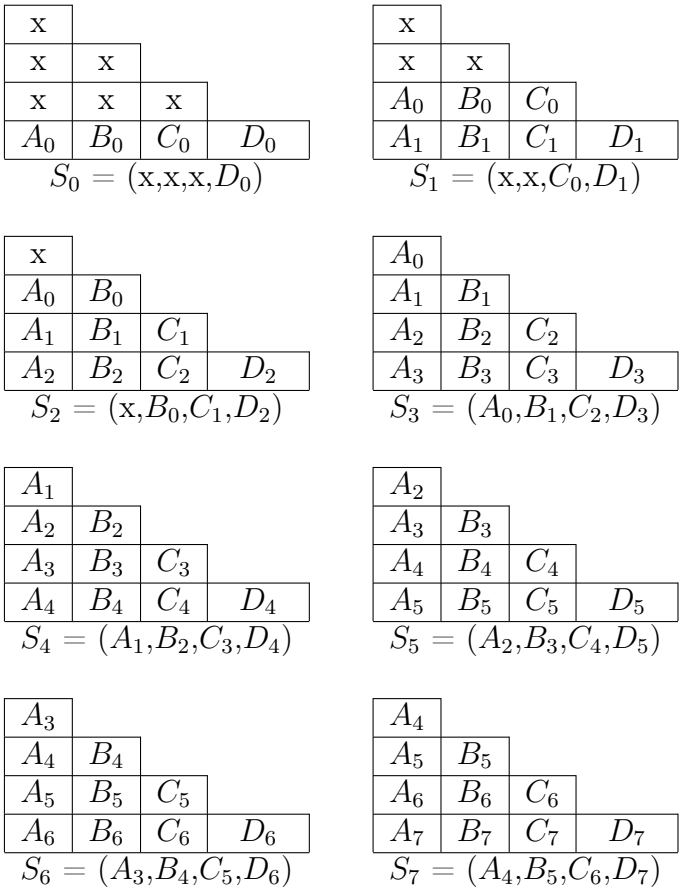

Figura 3.6: Secuencia de llenado del modelo de pilas de la Figura 3.5. La figura se lee de izquierda a derecha, arriba a abajo. Cada conjunto representa la transición de recibir cuatro datos seriales y se muestra la salida S_{i} correspondiente al mismo período. Los valores no definidos se han marcado con la letra "x".

los componentes y su ubicación, empero es más engorrosa. La segunda es más sencilla de implementar, pero es poco documentada y no permite implementar subniveles. Ambas son utilizadas para la creación de los modelos. Finalmente, es conveniente organizar estos nuevos modelos parametrizados en una librería. Tal como lo hace CASPER para sus bloques personalizados.

3.5. Caracterización del Detector

Una vez implementado, es importante conocer métricas que indiquen qué tan efectivo es el detector frente a variaciones en parámetros de la detección. Nuestro análisis se centra en dos figuras de mérito.

- Discriminación en la medida de dispersión
- Sensibilidad a señales de baja relación señal a ruido (SNR, por sus siglas en inglés)

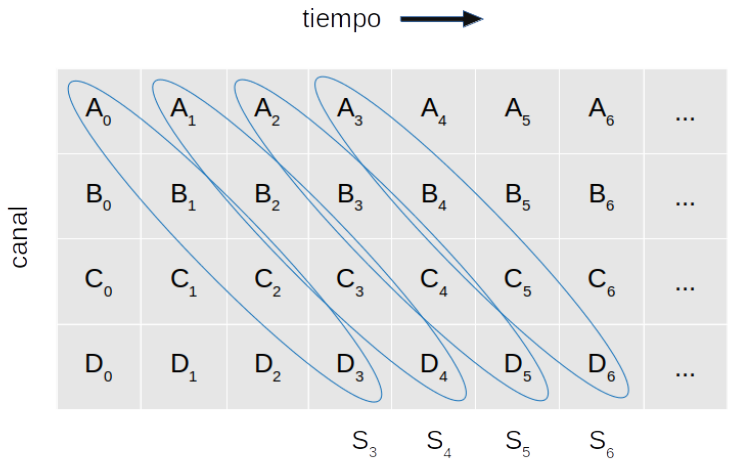

Figura 3.7: Ilustración de la interpretación de los datos de entrada y salida del dedispersor. La entrada de los datos se corresponde a las columnas de la figura. La salida del dedispersor corresponde a los óvalos en color azul desde S_3 en adelante. En este sentido, la dedispersión ocurre de manera continua, espectro a espectro.

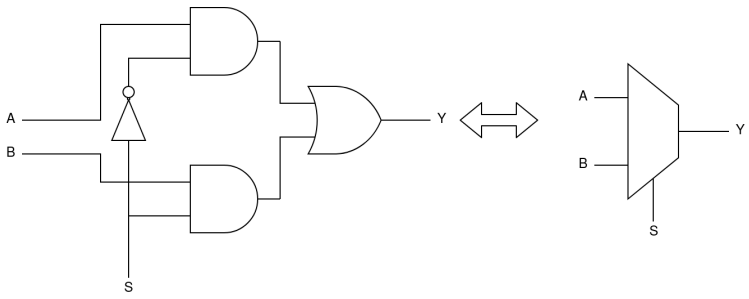

Figura 3.8: Abstracción de un multiplexor 2-a-1. A la izquierda se muestra la implementación a nivel de compuertas lógicas. A la derecha se muestra su abstracción, bastando configurar S a 0 ó 1 para obtener A o B, respectivamente. El usuario puede ocupar el bloque sin necesidad de conocer el funcionamiento interno.

El primero se refiere a que, dada una cierta medida de dispersión a detectar, cuánto se puede variar este parámetro hasta que ya no sea una detección válida. Para el segundo, se trata de averiguar qué tan efectivo resulta, en términos de aumento de SNR, el método de dedispersión frente a una señal inmediatamente a la salida del ADC.

3.6. Implementación del Paralelismo

El concepto para la implementación del paralelismo se muestra en la Figura 3.9, donde se observa que la propagación de datos tiene en común el paso por el conversor análogo digital y la transformada rápida de Fourier (FFT). Como cada medida de dispersión toma distintos tiempos para atravesar todos los canales espectrales, se debe adaptar el tiempo de integración de cada dedispersor, ya que en la plataforma ROACH resulta más sencillo conservar fija la

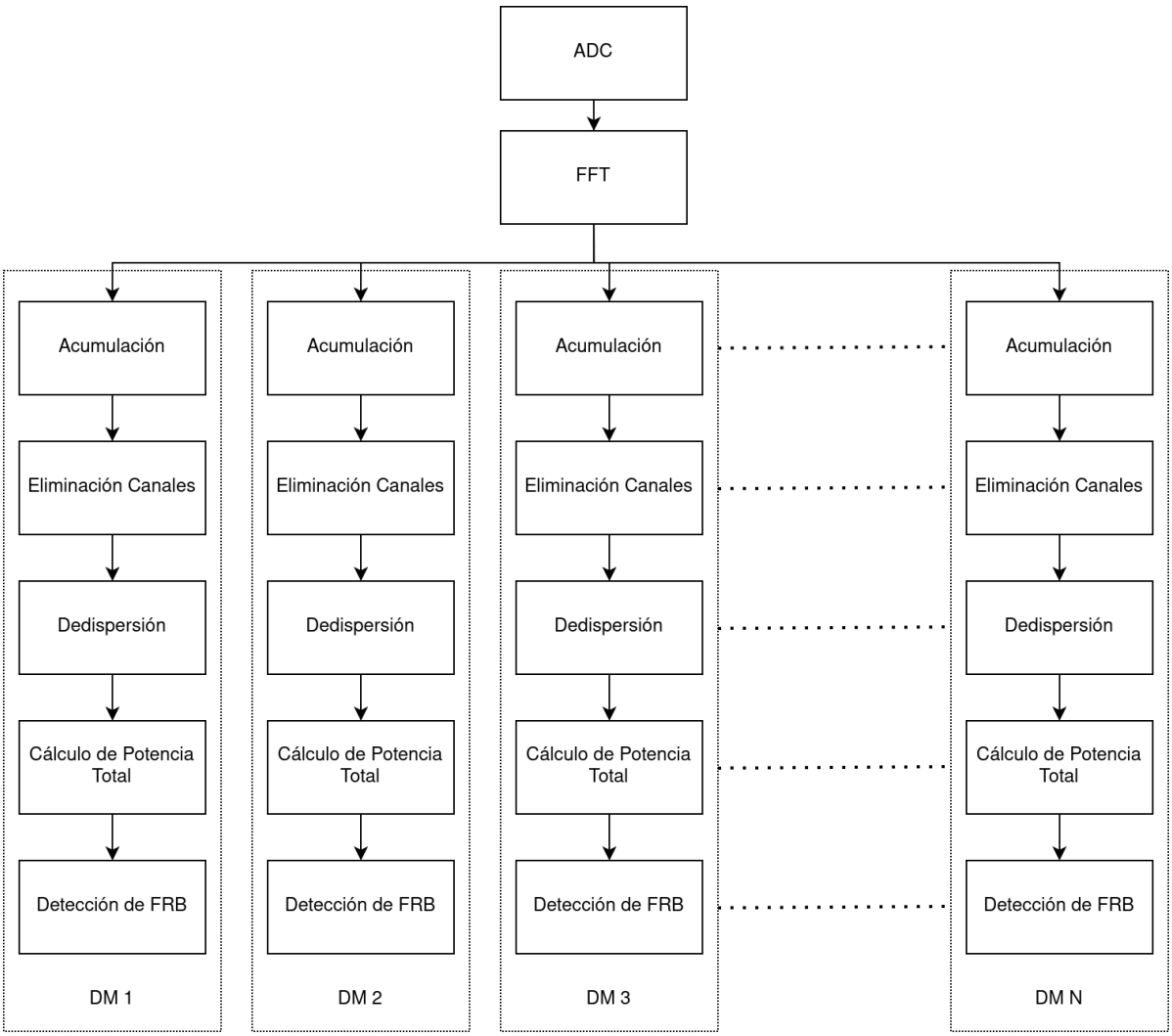

Figura 3.9: Diagrama de bloques de la implementación del paralelismo. El flujo de datos se separa en la etapa de acumulación al definir un tiempo de integración determinado. Cada detector paralelo detecta una medida de dispersión determinada.

cantidad de canales de la FFT y variar el tiempo de acumulación, mediante un registro escrito externamente. Por lo tanto, para un bloque de dedispersión asociado a cada DM, existirá un tiempo de acumulación dado para esa medida en específico y un camino hasta su detección único.

3.7. Resumen

Se ha presentado la metodología para llevar a cabo la construcción de un detector basado en la dedispersión incoherente y el procesamiento paralelo de distintas medidas de dispersión.

Antes que todo, es esencial familiarizarse con el entorno de desarrollo de ROACH 1 y ROACH 2 con pruebas básicas: prender un LED, observar la salida del ADC en el ordenador

y replicar un espectrómetro.

Para simular un FRB se supone que es un pulso gaussiano en banda base que se traslada en frecuencia desde las mayores a menores. Esto se implementa con un banco de pruebas que por un lado regula la velocidad de paso por el espectro y por otro toma una forma de pulso gaussiano. Finalmente, se adiciona una fuente de ruido simulando el intrínseco de toda observación astronómica.

Se diseña un detector para un único DM, basado en la dedispersión incoherente. Supone digitalizar la señal de entrada, procesar su espectro para luego dar paso a la dedispersión; proceso que corrige el retardo temporal supuesto para la medida de dispersión propuesta. Finalmente, si se ha acertado a la medida de dispersión se tendrá un umbral de potencia superior al del ruido lo que indicará una detección válida de un FRB.

Finalmente, se busca no sólo detectar un único DM sino la mayor cantidad posible dados los recursos lógicos. Para esto es necesario dedicar recursos exclusivos desde la etapa de dedispersión de la señal en adelante, pues los tiempos de integración son diferentes para cada DM.

Capítulo 4

Desarrollo

En este capítulo se presenta la implementación de lo expuesto en la metodología, a excepción de los tutoriales iniciales (que se encuentran adecuadamente documentados en sus propias fuentes). Se presenta la implementación del banco de pruebas, sus componentes, puntos de operación y desafíos encontrados. Se exponen las implementaciones en *Simulink* de las etapas de acumulación en adelante, destacándose la implementación del dedispersor el cual ha sido construido con niveles sucesivos de parametrización y abstracción. Se presenta "Honest Library"; librería personalizada que contiene el desarrollo realizado. Finalmente, se detallan las diferentes versiones construidas del detector.

4.1. Implementación del Banco de Pruebas

El esquema del banco de pruebas construído en laboratorio se presenta en la Figura 4.1. Se han agregado fuentes de voltaje, filtros, atenuadores y las ganancias de cada etapa, complementando al presentado conceptualmente en la Figura 3.2. El fundamento de las modificaciones se discute en la Sección 4.1.2. Los componentes utilizados son del estándar de conexión SMA de 50 Ω.

4.1.1. Componentes Principales

Oscilador controlado por voltaje (VCO)

Se utiliza el oscilador controlado por voltaje fabricado por **minicircuits** modelo ZX95-3360R-S+ (Figura 4.2). En la Tabla 4.1 se muestran sus principales características.

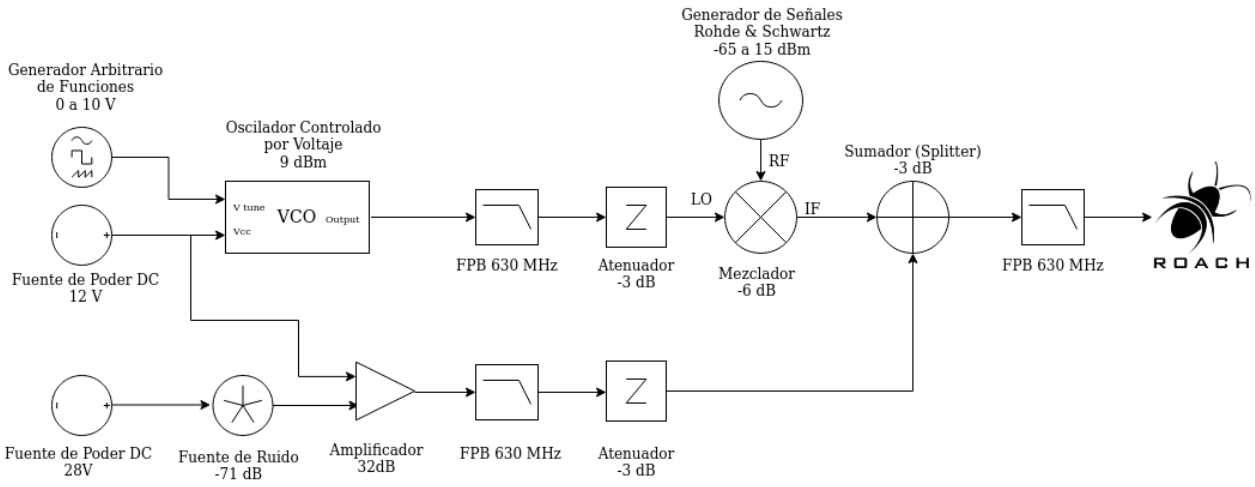

Figura 4.1: Banco de pruebas de laboratorio. El circuito se alimenta por dos fuentes de voltaje de 12V y 28V. Se han agregado filtros y atenuadores en relación al diagrama de la Figura 3.2, además, se ha resaltado la ganancia de cada etapa.

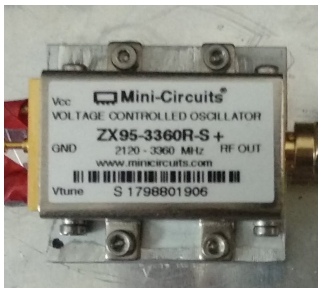

Figura 4.2: VCO modelo ZX95-3360R-S+ desarrollado por minicircuits

Mezclador (Mixer)

El mezclador utilizado es fabricado por **minicircuits** modelo ZX05-C42-S+. En la Tabla 4.2 se muestran sus principales características.

Generador Arbitrario de Funciones (AWG)

El generador arbitrario de funciones es el **KKMOON** modelo FY6800. Sus principales características son:

- Distintas formas de onda, entre ellas la rampa que será utilizada para generar un FRB
- Posibilidad de crear ondas personalizadas a través de un software y ondas en formato csv (sujeto a la resolución del instrumento)
- Resolución de frecuencia 1 μHz y de 1 mV de amplitud
- En función de formas de onda, oscila entre valores de 0 a 10 V
- Posibilidad de generar ráfagas de alguna onda con un disparador manual o automático y una repetición arbitraria.

Tabla 4.1: Especificaciones principales del VCO

Impedancia	50	Ohm
Banda de Operación	2120 a 3360	MHz
Tensión de Alimentación	12	V
Tensión de Sintonización	0.5 a 18	V
Potencia de Salida	9	dBm

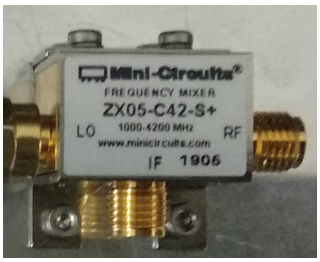

Figura 4.3: Mixer modelo ZX05-C42-S+ desarrollado por minicircuits

Recordando, la Ecuación 2.3 que determina el tiempo de dedispersión Δt. Notamos que la frecuencia del generador arbitrario es el recíproco de este tiempo, luego:

$$f_{AWG} = \frac{1}{\Delta t(s)} \quad [\text{Hz}] \tag{4.1}$$

Fuente de Ruido

La fuente de ruido utilizada es fabricada por Agilent modelo 346B (Figura 4.5), con un espectro de ruido desde 10 MHz a 18 GHz. Calculamos su ganancia como sigue [14]:

$$P_{Noise} = N_0 + 10 \cdot log(BW\ [\text{Hz}]) \quad [\text{dBm}] \tag{4.2}$$

donde N_0,

$$N_0 = -174 + ENR \quad [\text{dBm}] \tag{4.3}$$

Donde el ENR es la razón de exceso de ruido, que asumimos en 15 dBm. Reemplazando se tiene lo siguiente:

$$P_{Noise} = -159 + 10 \cdot log(BW\ [\text{Hz}]) \quad [\text{dBm}] \tag{4.4}$$

Tabla 4.2: Especificaciones principales del Mixer

Banda de Operación	1000 a 4200	MHz
Potencia Máx. RF	17	dBm
Potencia LO	7	dBm
Pérdidas por conversión	6.1	dBm

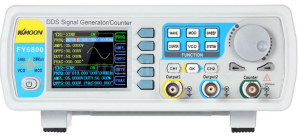

Figura 4.4: AWG KKMOON FY6800

Amplificador

Como amplificador se utiliza uno genérico (Figura 4.6) cuyas especificaciones se presentan en la Tabla 4.3.

4.1.2. Consideraciones del montaje

Al momento de implementar el circuito conceptual presentado en la Figura 3.2, se realizan modificaciones que conducen al circuito práctico de la Figura 4.1, por las razones listadas a continuación.

- Fuente de ruido: se agrega un filtro pasa bajos (FPB) de 630 MHz, lo que además define la fórmula para determinar la ganancia de la fuente, reemplazando su valor de corte en BW [Hz] en la Ecuación 4.4.
- Amplificador: debido a la alta ganancia del componente, la interferencia de radio frecuencia (RFI) de los dispositivos adyacentes (WiFi, Red Celular) toma relevancia en la señal de salida. Para solucionar esto se agrega un atenuador de 3 dB y se confina la cadena de ruido dentro de una caja metálica.
- VCO - Mixer: se agrega un atenuador de 3 dB en la salida del VCO de modo que la entrada LO del Mixer sea de 6 dB. Asimismo, se agrega un FPB de 630 MHz para evitar armónicos.
- Sumador (Splitter): Se agrega un FPB a la salida por precaución para reforzar la ausencia de armónicos.
- Generador de Señales: La potencia del generador "Rohde & Schwartz" se varía y opera según los rangos del ADC de ROACH. Se alcanza el piso de ruido con una potencia $\sim$ -65 dB. La saturación se alcanza con una potencia de 15 dBm.
- Generador de Señales: Se obtiene una señal pseudo gaussiana en el generador "Rohde & Schwartz" con una modulación de fase senoidal, con una desviación de 76 radianes a 20 kHz.

Figura 4.5: Fuente de Ruido Agilent 346-B

Tabla 4.3: Especificaciones del Amplificador

Voltaje de Operación	12	V
Rango de Frecuencia	1 - 2000	MHz
Amplificación	32	dB
Corriente de Operación	35	mA

4.2. Implementación del Detector en *Simulink*

En la presente sección se detalla la implementación en *Simulink* de cada diagrama de bloques de la Figura 3.3 desde el bloque acumulador en adelante. El bloque ADC y FFT se encuentra debidamente documentado en los tutoriales de CASPER. Se da especial énfasis al bloque dedispersor el cual ha sido implementado utilizando abstracciones y parametrizaciones sucesivas desde bloques primitivos.

4.2.1. Acumulación

El bloque de acumulación se encuentra inmediatamente después de la FFT y, tal como sugiere su nombre, acumula la potencia espectral durante un tiempo de integración determinado. Sin embargo, el hardware FPGA define el paso del tiempo de manera discreta, por medio de una señal de reloj, por lo que el concepto de tiempo de integración se reemplaza a uno de número de acumulaciones.

En primer lugar, se define el tiempo del pulso dedispersado t_{d} como el tiempo de integración requerido para capturar un único espectro (con todos sus canales de frecuencia). Recordando que el FRB atraviesa todos los canales espectrales, se relaciona a Δt por la expresión.

$$t_{\mathrm{d}} = \frac{\Delta t}{FFT_{chans} \times \alpha} \tag{4.5}$$

donde FFT_{chans} es la cantidad de canales de la FFT y α es un factor igual al cociente entre la cantidad de canales no mitigados y los canales totales. Luego, la cantidad de acumulación espectral n_{acc} estará dada por:

$$n_{acc} = t_{\mathrm{d}} \times clk_{fpga} \times par_{chn} \tag{4.6}$$

donde clk_{fpga} es la velocidad de reloj de la FPGA y par_{chn} es la cantidad de hilos o canales

Figura 4.6: Amplificador Genérico de 32 dB

Figura 4.7: Registro de número de acumulaciones (bloque amarillo). El número se calcula en una rutina de Python y se carga al registro al iniciar la ejecución del hardware. A continuación se agrega un retardo por estabilidad y se define un nodo con el nombre de la señal.

siendo procesados en paralelo. Este número se calcula en una rutina de Python y es escrito en un registro en el modelo compilado en la FPGA (Figura 4.7).

Esta señal entra a un bloque de tres entradas mostrado en la Figura 4.8. Este bloque se encarga de informar cuándo comienza un nuevo intervalo de acumulación. Recibe la información del número de acumulaciones y una señal de sincronización proveniente de la FFT. Además, el bloque tiene una señal de reinicio (rst) general.

4.2.2. Eliminación de Canales

Luego de la acumulación, el bloque de eliminación de canales se implementa en *Simulink* con lo mostrado en la Figura 4.9. Los datos que salen de los bloques de memoria (Figura 4.10) se dirigen a multiplexores. Se presentan dos casos: con eliminación (salida del bloque de memoria superior) y sin eliminación (salida del bloque de memoria inferior). Para el caso sin eliminación basta que la señal de validación "valid" esté encendida para tener un valor no nulo a la salida "dataout2".

En el caso de eliminación de canales, se ha añadido una compuerta AND de tres entradas al selector del multiplexor, equivalente a que tres condiciones deben cumplirse simultáneamente para su activación, ellas son:

- La señal debe ser válida
- El número a la salida del contador es distinto a la constante con valor 0
- El número a la salida del contador es distinto a la constante con valor 4

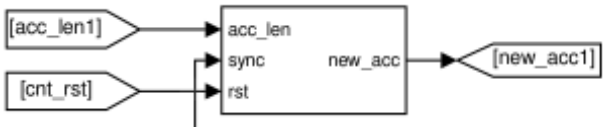

Figura 4.8: Bloque generador de señal de sincronización. Recibe la información del número de acumulaciones y una señal de sincronización proveniente de la FFT. Esto permite informar en la salida cuándo comienza un nuevo intervalo de acumulación. Por último el bloque tiene una señal de reinicio (rst).

El contador es activado por la señal de validación y reiniciado por la señal que indica una nueva acumulación. El valor B máximo que toma el contador esta dado por la expresión

$$B = \frac{\#CanalesFFT}{\#CanalesParalelos} - 1 \tag{4.7}$$

donde la fracción representa el cociente entre la cantidad de canales de la FFT y el número de canales en paralelo que se han implementado. A modo de ejemplo, en la Figura 4.9 se tienen dos canales paralelos (uno con y otro sin mitigación), suponiendo que la FFT tiene 64 canales, el contador produce 32 valores diferentes que son los números binarios de 0 a 31 (implicando un contador de 5 bits).

Ahora bien, la constante A con la que se compara el contador se calcula con la expresión

$$A = \lfloor \frac{CanalFFT_{eliminado}}{\#CanalesParalelos} \rfloor \tag{4.8}$$

Que corresponde a la función piso del cociente entre el canal eliminado de la FFT y el número de canales paralelos. Por otro lado, el canal C donde se debe conectar la compuerta AND de condición o condiciones (en el caso de más de un canal mitigado) esta dado por

$$C = (CanalFFT_{eliminado}) \ mod \ (\#CanalesParalelos) \tag{4.9}$$

donde mod es la operación modulo, correspondiente al resto de la división entera. En nuestro ejemplo, se ha eliminado el canal 0, ya que $\lfloor \frac{0}{2} \rfloor = 0$ (constante igual 0) y el canal 8, ya que $\lfloor \frac{8}{2} \rfloor = 4$, ambos conectados al canal 0, pues $0 = 0 \, mod \, 8 = 8 \, mod \, 4$

En el modelo de ROACH 2 se tienen 64 canales en la FFT, de los cuales se eliminan los canales 0 y 32. El primero corresponde a la componente DC de la señal y el segundo corresponde a un artefacto de calibración del ADC. La implementación tiene 8 canales paralelos, luego, las constantes son $\lfloor \frac{0}{8} \rfloor = 0$ y $\lfloor \frac{32}{8} \rfloor = 4$, ambos se conectan al canal 0, pues la división es exacta.

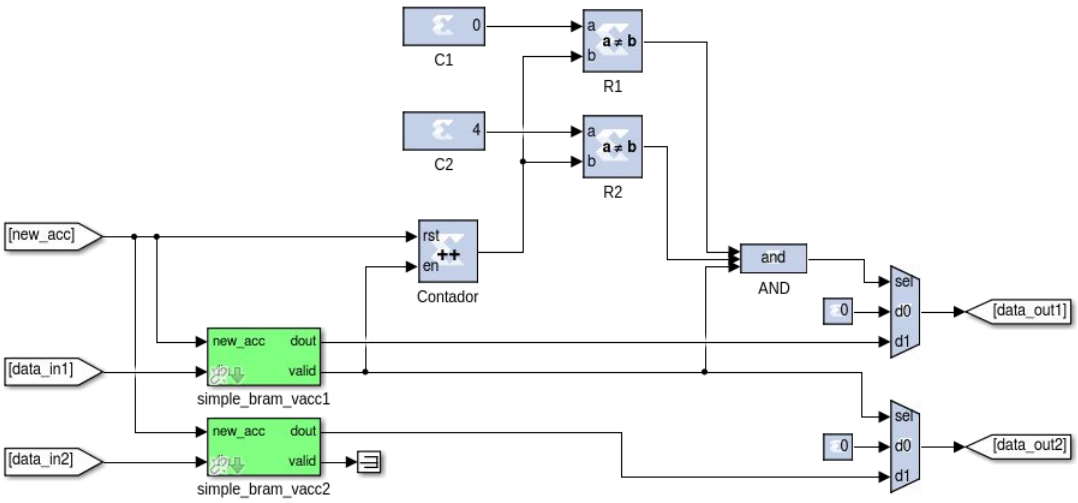

Figura 4.9: Etapa de eliminación de canales. Los datos que salen de los bloques de memoria (Figura 4.10) se dirigen a multiplexores. Se presentan dos casos: con eliminación (bloque superior) y sin eliminación (bloque inferior). Para el caso sin eliminación basta que la señal de validación "valid" esté encendida para tener un valor no nulo a la salida "dataout2". Paralelamente, se ha añadido un contador activado por la señal de validación y reiniciado por la señal de nueva acumulación. Para el caso con eliminación de canales, se han añadido tres condiciones al selector del multiplexor que deben cumplirse simultáneamente (AND): debe ser válido, la cuenta no puede ser 0 y no puede ser 4.

4.2.3. Dedispersor

En esta sección se presenta la implementación en *Simulink* del bloque dedispersor, con una exposición desde el más alto al más bajo nivel, comenzando con el dedispersor paralelo (Figura 4.12); bloque que recibe parámetros para implementar la dedispersión completa. Los parámetros son vector de dedispersión, cantidad de canales paralelos y número de bits del contador interno. La exposición se adentra hasta llegar al denominado "bloque detector" (Figura 4.18) el cual está construido únicamente con bloques de bajo nivel del fabricante de la FPGA, Xilinx.

Dedispersor Paralelo

Ya hemos explicado en la sección 3.4.2 el principio de funcionamiento del bloque dedispersor. Notamos que la cantidad de retardos de cada canal queda completamente definida por un vector, que llamaremos de dedispersión, de la forma

$$\begin{array}{cccccc} & ch_0 & ch_1 & ch_2 & ... & ch_n \\ \mathrm{v} \;=\; & [r_0, & r_1, & r_2, & ..., & r_n] \end{array} \tag{4.10}$$

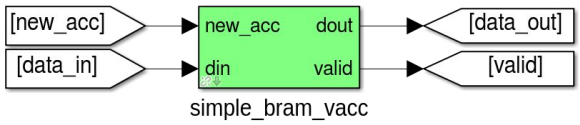

Figura 4.10: Bloque de memoria acumulador. Este bloque conoce la longitud del espectro y acumula los valores de cada canal a medida que entran por "din". Cuando comienza una nueva acumulación ("newacc") los datos acumulados son liberados ("dataout"). Mientras los datos de salida son válidos la señal "valid" se mantiene activa.

donde r_{i} es la cantidad de retardos o la cantidad de veces que se debe recordar el canal i (ch_{i}) para dedispersar la señal.

Para generar el bloque, además del vector de dedispersión, necesitamos especificar cuántos canales paralelos serán necesarios y cuántos bits tendrá un contador interno [1], dado por la expresión

$$bits_{counter} = \log_2(\frac{\#CanalesFFT}{\#CanalesParalelos}) \tag{4.11}$$

que corresponde al logaritmo en base 2 del cociente entre la cantidad de canales de la FFT y el número de canales en paralelo del modelo. Luego, en la Figura 4.11 se muestra la interfaz de usuario del bloque dedispersor paralelo. Su código de inicialización puede ser consultado en los anexos (Código A.3).

Al ingresar los parámetros, se ejecuta en segundo plano una función que genera los bloques correspondientes de la implementación. La generación de bloques internos depende de los valores ingresados y, es importante recalcar, que esto conducirá, luego de compilar el modelo, a distintos circuitos dentro de la FPGA. Por construcción de la función generadora, cada canal paralelo del dedispersor puede manejar hasta 1024 retardos en sus canales.

En la Figura 4.12 se muestra una posible generación del bloque dedispersor paralelo. En este caso, la parametrización ha generado cuatro entradas paralelas con sus correspondientes salidas. El bloque recibe, además, la señal de validez de los acumuladores.

Dedispersor de 1 canal

A continuación describiremos el dedispersor de 1 canal; en primer lugar, como el bloque que conforma el dedispersor paralelo y, en segundo lugar, el bloque que implementa el modelo de pilas presentado en la Sección 3.4.2, por medio de sus bloques internos.

Antes que todo, nos interesa discutir el formato de salida de cada uno de los canales

[1] Este parámetro (usado en principio para corrección de errores) resulta ser recurrente y puede ser eliminado en versiones futuras del dedispersor paralelo

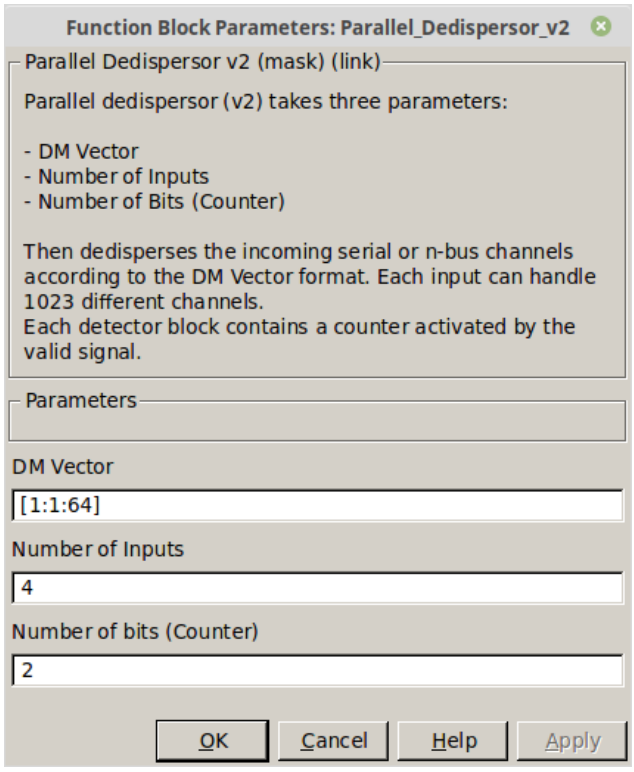

Figura 4.11: Interfaz de usuario del bloque dedispersor paralelo. Se requiere como parámetros el vector de dedispersión, la cantidad de entradas y el número de bits del contador interno, respectivamente.

espectrales provenientes de la FFT a la salida del bloque de mitigación de canales, más aún; el caso en que se tiene más de un canal paralelo, por ejemplo, en la Figura 4.9 se tienen dos. En la Figura 4.13 se presenta conceptualmente el formato de salida de los datos con cuatro (4) canales paralelos; para cada ciclo de reloj t_{i} se tiene, simultáneamente, la información de cuatro canales espectrales, uno por cada camino paralelo. Así, en el primer camino asociado a $data_1$ recibiremos los canales espectrales 0, 4, 8, 12, etc. (hasta agotar la cantidad total de ellos).

Se reinterpreta el vector de dedispersión definido en la Ecuación 4.10, que definía el retardo de cada canal espectral, al retardo que debe asociarse a cada camino paralelo (denotado con $data_{\mathrm{i}}$) como

$$v_{data1} = [r_0, r_4, r_8, r_{12}, ...] \tag{4.12}$$

$$v_{data2} = [r_1, r_5, r_9, r_{13}, ...] \tag{4.13}$$

$$v_{data3} = [r_2, r_6, r_{10}, r_{14}, ...] \tag{4.14}$$

$$v_{data4} = [r_3, r_7, r_{11}, r_{15}, ...] \tag{4.15}$$

cada uno de estos vectores de retardos es el parámetro de un bloque dedispersor con una entrada y salida seriales (una única entrada y salida), ilustrados en la Figura 4.14; en efecto, hemos "desenmascarado" el dedispersor paralelo. Con la información del vector de

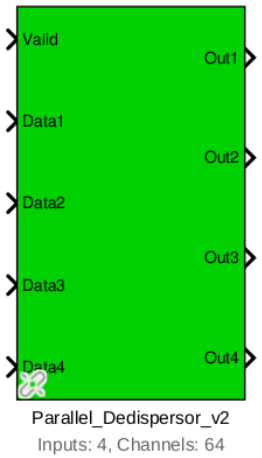

Figura 4.12: Dedispersor Paralelo. La parametrización ha generado, en este caso, cuatro entradas paralelas con sus correspondientes salidas. El bloque recibe, además, la señal de validez de los acumuladores.

	t_0	t_1	t_2	t_3	...
$data_1$	0	4	8	12	...
$data_2$	1	5	9	13	...
$data_3$	2	6	10	14	...
$data_4$	3	7	11	15	...

Figura 4.13: Orden de datos en salida de los acumuladores con cuatro canales en paralelo (denotados data_{i}). Se observa que los primeros cuatro canales salen en el mismo tiempo de acumulación t_0. Lo mismo se repite para los canales siguientes.

dedispersión y el número de canales paralelos, es posible descomponer vectores de retardos para cada canal que se rellenan automáticamente en la máscara mostrada en la Figura 4.15 de los bloques dedispersores.

estos bloques de retardos se corresponden con los mencionados en el principio de funcionamiento del dedispersor. En efecto, cada retardo r_{i}, corresponde a la altura de cada pila asociada a cada canal, el largo del vector corresponde al largo de dicho conjunto de pilas, se tiene una entrada serial y una salida serial. El bloque dedispersor es el que implementa esto.

Antes de entrar en la implementación del bloque dedispersor de 1 canal, resulta útil recordar el modelo de pilas de la Seccion 3.4.2. La cantidad r_{i} de cada vector de retardo, es, de hecho, la altura de cada pila asociada a un canal determinado y, el largo del vector, se corresponde con la cantidad de canales (y el número de pilas verticales). Sin embargo, hay un elemento aún ausente del modelo; el selector que toma el dato superior de cada pila y lo coloca en la salida serial; se explica, a continuación, que es posible su implementación con un contador y un multiplexor, debidamente conectados.

Dentro del bloque, en la Figura 4.16, encontramos que se tienen los datos entrando a cada bloque detector básico (que representa una pila). La señal de validación será la encargada de indicar cuál es el canal actual y si corresponde guardarla en dicho bloque. A la salida de

cada detector básico se conecta un multiplexor, cuyo selector es manejado por un contador; su efecto es seleccionar secuencialmente, uno a uno, el valor fuera de los bloques básicos, obteniéndose una salida serial.

Dentro de la implementación del bloque dedispersor de 1 canal, se encuentran dos bloques de bajo nivel, identificados por su color azul y marca de agua del logo de Xilinx, estos son, el contador y el multiplexor (mux). Éste último al ser un componente de bajo nivel posee un límite de 32 señales de entrada, o, alternativamente, hasta 5-bits para la señal de selección, lo que limitaría el largo máximo del vector de retardos a 32 para un canal. Para evitarlo, la función inicializadora del bloque implementa una doble etapa de multiplexores de ser necesaria, con un soporte de hasta 1024 canales espectrales por cada canal paralelo, con un contador de hasta 10 bits. La rutina que inicializa el bloque puede ser consultada en los apéndices (Código A.2).

Bloque detector básico

La última capa del dedispersor paralelo es el bloque detector, que emula una única pila. Sus parámetros son el canal sobre el que se hace el retardo, la latencia o la cantidad de ciclos que debe ser recordado un dato y el número de bits del contador interno que corresponde a

$$bits_{counter} = \log_2(len_v) \tag{4.16}$$

donde len_v corresponde al largo del vector de retardos del bloque dedispersor. Esto asegura que la selección secuencial de los canales esté sincronizada. Al igual que los casos anteriores, su máscara (Figura 4.17) se rellena automáticamente al inicializar un bloque dedispersor con un vector de retardos. Su código de inicialización se encuetra anexo (Código A.1).

La implementación a nivel de circuito del bloque detector se muestra en la Figura 4.18. Aquí se tienen dos señales de entrada *Data* y *Valid*. La primera señal, es el dato de entrada del bloque dedispersor que ha entrado a cada bloque detector en paralelo. La segunda es la señal de validez de los datos. El bloque se compone sólo de bloques de bajo nivel, en efecto, se ha alcanzado el nivel más elemental de representación posible.

La lógica de funcionamiento es la siguiente. Cuando el dato es válido, un contador lleva registro del canal actual que se está recibiendo. Si este canal coincide con el pre-configurado para el bloque, entonces se guarda el dato en "Delay". Éste funciona precisamente como una pila de datos o una memoria FIFO (*first in, first out*). La llegada de un nuevo dato "expulsa" el dato anteriormente guardado, en este caso, por el puerto de salida "Out1" del bloque.

4.2.4. Cálculo de Potencia Total

Esta etapa recibe en cada instante de tiempo porciones del espectro dedispersado. En primer lugar, se implementa una cadena de sumadores que sintetiza la potencia total. Este

número se vuelve a reinterpretar en el estándar de 64 bits descartando los bits menos significativos. La implementación se muestra en la Figura 4.19. En segundo lugar, la potencia va a un acumulador que suma el valor de potencia actual con el anterior, hasta barrer todo el espectro, que es lo mismo que la señal de validez cambie a 0.

Cada salida válida del acumulador se guarda en una memoria ("ACC1" en la Figura 4.19). Esta señal será de gran interés en el análisis final, donde cada punto resume el valor de la potencia total de un espectro dedispersado. Dada su importancia, la llamaremos **señal de potencia total dedispersada**.

4.2.5. Detección de FRB

La detección de FRB se implementa comparando la potencia acumulada del espectro dedispersado con una tolerancia guardada en un registro "thetha". Si esto ocurre se levanta una alerta (se escribe el registro "frb detector1") que se desactiva al reiniciar el proceso. La implementación se encuentra en la Figura 4.20.

4.2.6. Honest Library

Los modelos de bloques dedispersores parametrizados mencionados en la Sección 4.2.3 son agrupados en una librería personalizada: "Honest Library" (Figura 4.21), a fin de que pueda ser usado en otros modelos y puedan ser implementadas mejoras de manera eficiente.

4.2.7. Versiones Detector

En el Anexo C se encuentra un detalle de las distintas versiones del detector tanto en la plataforma ROACH 1, como en ROACH 2.

4.3. Conclusiones

Se ha implementado un banco de pruebas para emular el comportamiento de un FRB en banda base, con un ruido de observación asociado. El montaje permite emular una cierta medida de dispersión, cambiando la frecuencia de operación del generador arbitrario de funciones (f_{AWG}) , y alterar la intensidad de la señal gaussiana, alterando la potencia de salida en dBm del generador. La construcción del banco supuso agregar atenuadores y filtros para ajustar las potencias de entrada salida de los componentes y eliminar posibles componentes armónicos, respectivamente. Con todo lo anterior, se tiene una señal lista para ser digitalizada por el conversor análogo digital (ADC) de ROACH.

Se ha explicado la implementación de los bloques que conforman el detector desde la

acumulación hasta la detección de un FRB. La etapa de acumulación, define el tiempo de pulso dedispersado t_{d}, para tomar una muestra de los canales espectrales que sea relevante dada la medida de dispersión, dicho tiempo continuo se traduce en un parámetro discreto llamado "número de acumulaciones", escrito en un registro. Se ha discutido la implementación del bloque de eliminación de canales para el caso de varios hilos paralelos y el porqué de la interconexión de los componentes basado en el número de canal eliminado.

Se ha implementado un dedispersor paralelo completamente parametrizado por el vector de dedispersión y conformado por dedispersores de 1 canal que recibe la fragmentación del parámetro en forma de vectores de retardos. Se ha hecho la analogía de cómo el dedispersor de 1 canal implementa el modelo de pilas presentado en la Sección 3.4.2, en efecto, cada pila es un bloque detector básico y el selector de canales es un conjunto multiplexor-contador que recorre secuencialmente cada pila.

A la salida del dedispersor se calcula la suma las contribuciones de todos los canales espectrales dedispersados para un tiempo de acumulación determinado, generando una nueva señal llamada **señal de potencia total dedispersada**, la cual condensa punto a punto el historial de la detección. Finalmente, el bloque de detección de FRB se implementa comparando con un valor umbral almacenado en un registro que, de ser superado, dispara una alerta de detección.

Este trabajo presenta una "versión final" del detector implementado. Las sucesivas iteraciones que han conducido a esta versión pueden ser consultadas en los anexos así como el detalle del desarrollo en ROACH 1 y ROACH 2.

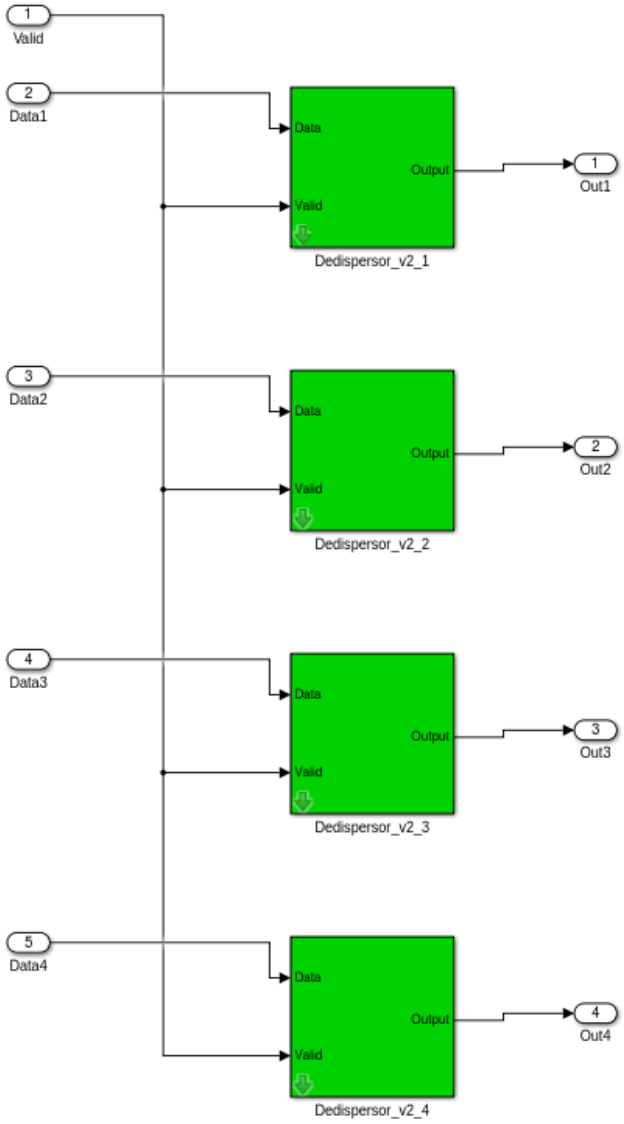

Figura 4.14: Bloques dedispersores para 4 canales en paralelo. Cada uno recibe la señal de validez y datos de entrada de forma serial.

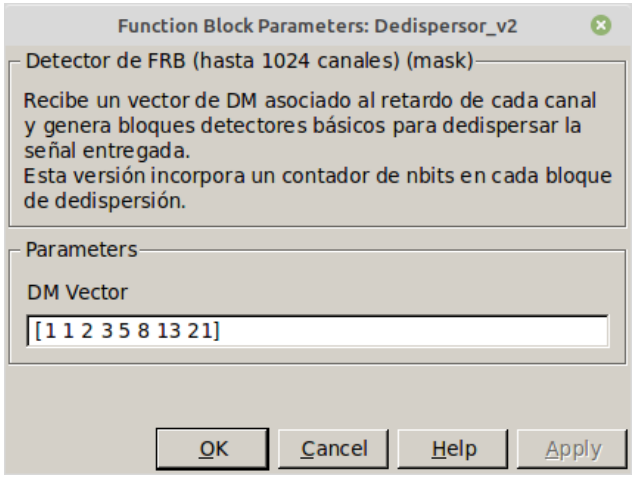

Figura 4.15: Interfaz del bloque dedispersor. Sólo requiere del parámetro del vector de dedispersión, cuya longitud puede ser de hasta 1024 valores.

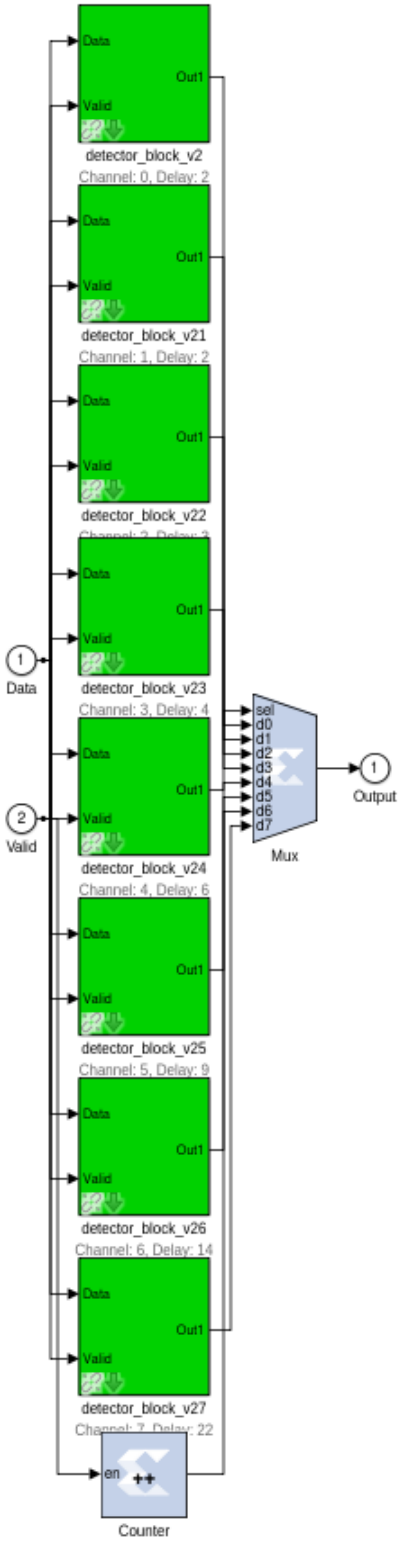

Figura 4.16: Implementación del Bloque Dedispersor. Cada retardo del vector se corresponde con un bloque detector (en este caso 8). Los datos van paralelamente a cada uno de ellos, similar al modelo de pilas. Finalmente, un multiplexor controlador por un contador hace de selector uno a uno de la salida serial.

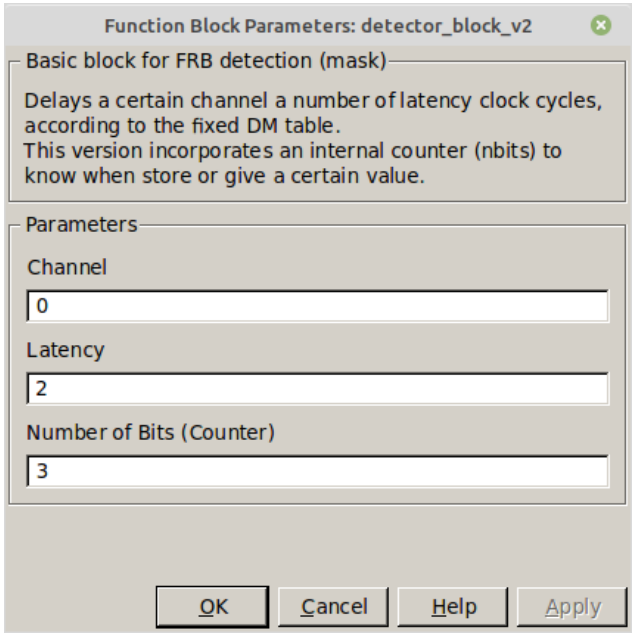

Figura 4.17: Interfaz del bloque detector. Sus parámetros son el canal sobre el cuál se hará el retardo, la latencia o cantidad de retardo y el número de bits del contador interno.

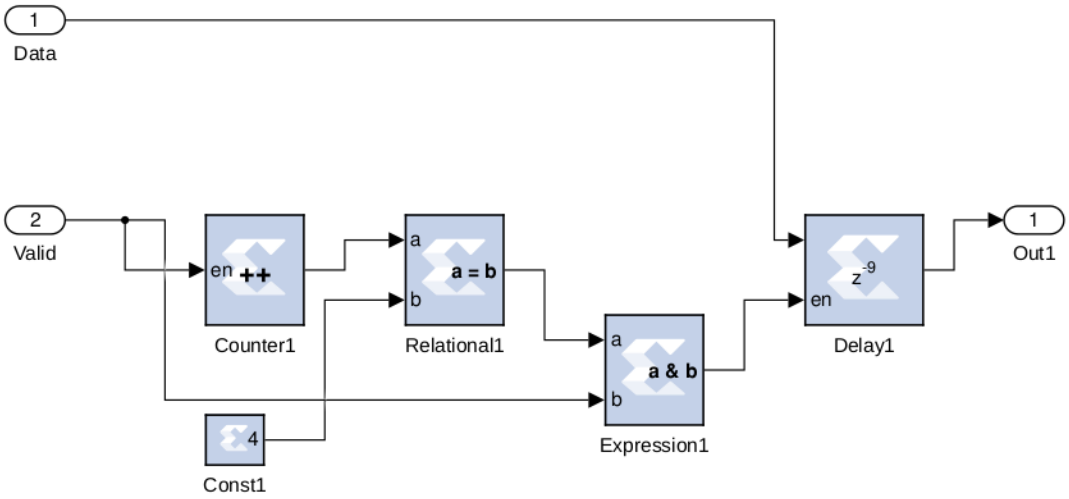

Figura 4.18: Implementación del Bloque Detector. La señal de validación se vale de un contador (Counter1) para identificar cuándo el canal preconfigurado corresponde al actual (Const1). En ese momento "empuja" un valor en la memoria (Delay1). En este caso, el cuarto canal se retarda nueve veces.

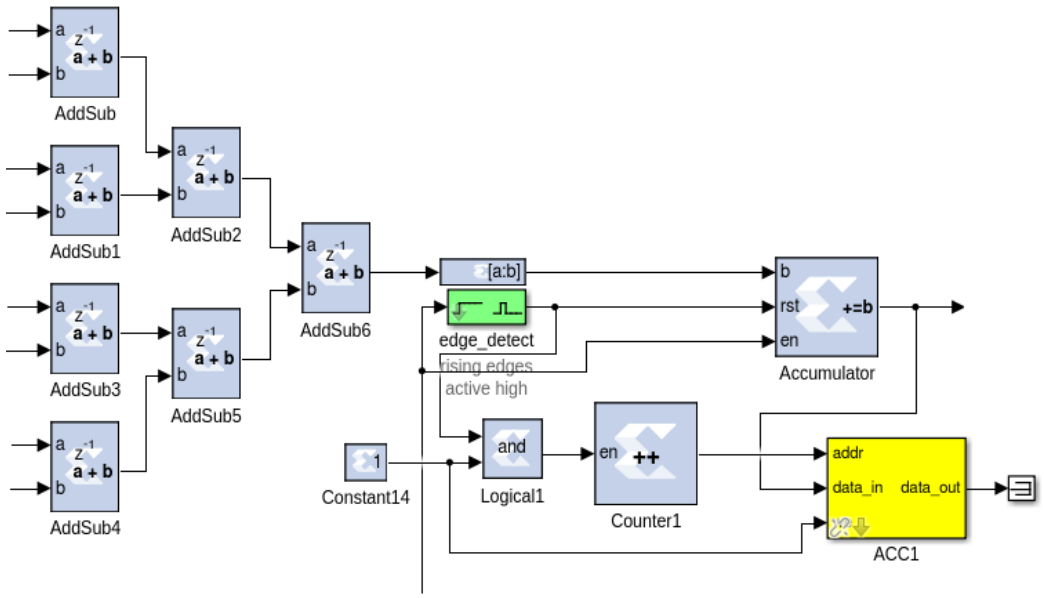

Figura 4.19: Implementación de la suma espectral y la acumulación. Los datos del dedispersor se suman en una etapa de sumadores (AddSub) sucesivos. Los datos son reinterpretados en 64 bits eliminando los bits menos significativos. Finalmente, son acumulados mientras el espectro sea válido. Cada salida válida del acumulador se guarda en una memoria (ACC1).

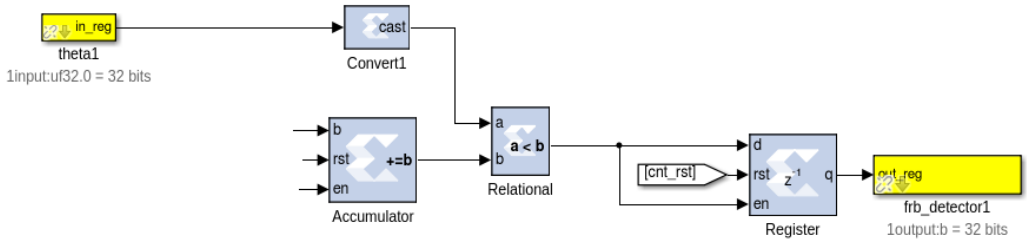

Figura 4.20: Etapa de detección de FRB. La señal de acumulación actual se compara con un número "theta" guardado en un registro de 32 bits extendido (cast) a 64 bits. Si la primera es mayor, se guarda en un registro y se enciende la advertencia de detección (frb detector1 = 1). La advertencia puede ser reiniciada con la señal de reinicio general.

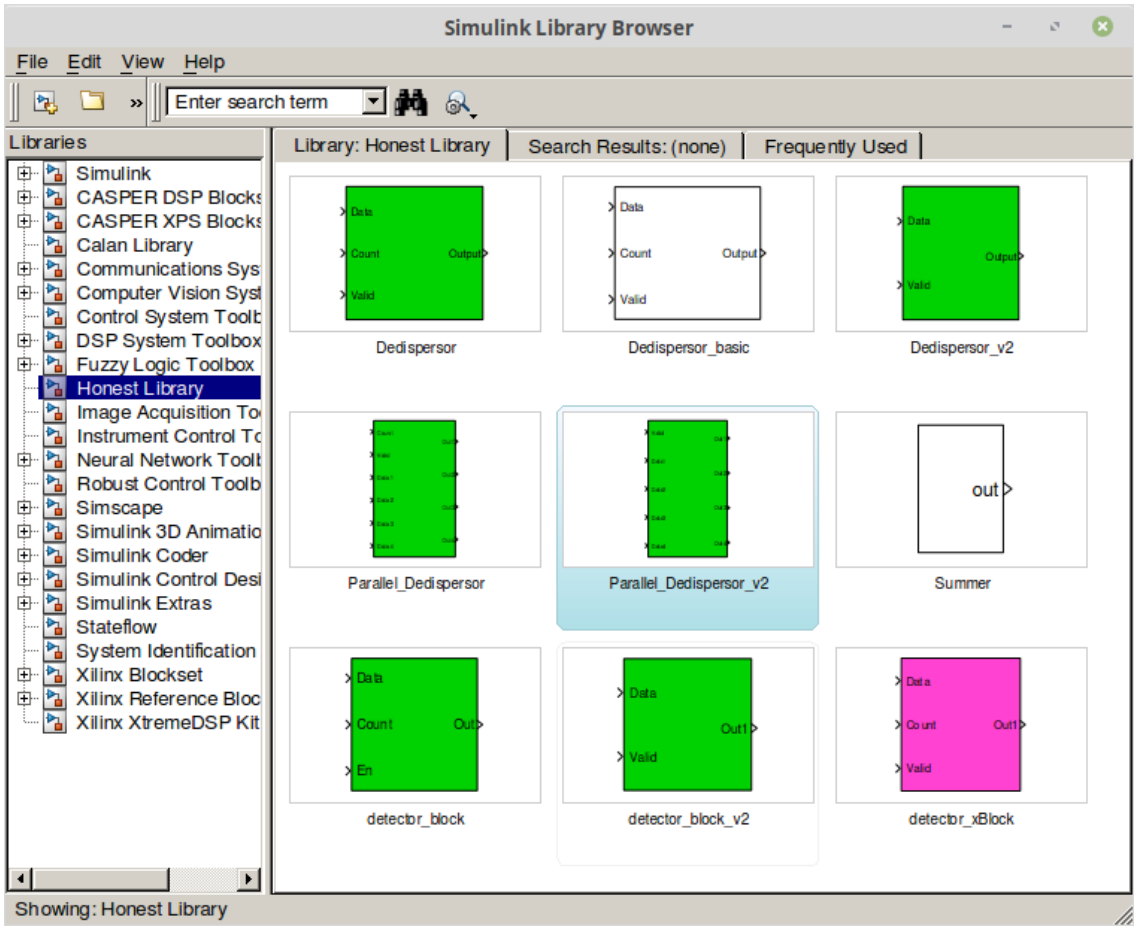

Figura 4.21: Honest Library. Librería que agrupa el bloque dedispersor paralelo y los bloques individuales que lo conforman.

Capítulo 5

Resultados y Análisis

En primer lugar, preliminarmente se mencionan parámetros de operación del conjunto del banco de pruebas y el detector final, ajustados en la etapa de obtención de resultados. Luego se presentan pruebas básicas de desempeño de los componentes del banco de pruebas. Se caracteriza en el detector el decaimiento de la señal de potencia total dedispersada en función de la variación del DM de entrada y la relación señal ruido de la entrada versus la de la señal de potencia total dedispersada. Se realiza un análisis sobre qué medidas de dispersiones deberían ser detectadas y cuántos detectores pueden ser implementados en paralelo dado los recursos lógicos de la FPGA. Finalmente, se presentan los resultados del modelo final, con 10 detectores en paralelo y se analiza su respuesta en función de la señal de potencia total dedispersada; señal que toma relevancia en el análisis y monitoreo de resultados.

5.1. Parámetros de Operación

Preliminarmente, a la obtención de resultados se fijan los parámetros de operación presentados en esta sección.

El generador "Rohde & Schwartz" se configura a una frecuencia de 2850 MHz, variando su potencia de salida desde los -65 dBm aproximadamente (piso de ruido) hasta los 15 dBm, donde se produce una saturación del ADC de ROACH. Para obtener una señal aproximadamente Gaussiana se aplica una modulación de fase senoidal con una desviación de 76 rad a 20 kHz. Sin embargo, su forma es demasiado angosta incluso maximizando los parámetros, en el sentido que no se observan diferencias significativas en la detección de FRB con un tono puro (es decir, sin la modulación de fase).

Las tensiones de operación del generador arbitrario de funciones se muestran en la Tabla 5.1, donde la amplitud es mayor en la plataforma ROACH 2. En efecto, el ancho de banda de la primera es de 480 MHz y el de la segunda es 540 MHz, definido por las velocidades de reloj de cada una. Estos rangos de operación son tales que la señal atraviesa todo el espectro visible por el hardware.

Tabla 5.1: Tensiones de operación del generador arbitrario de funciones para ROACH 1 y ROACH 2

	Plataforma	
	ROACH	ROACH 2
V_{low} (V)	3.59	3.10
V_{high} (V)	8.11	8.08
Amplitude (V)	4.52	4.98
Offset (V)	5.85	5.59

El modelo sometido a pruebas se compila en ROACH 2 con una FFT de 64 canales y 8 canales paralelos. Se elimina el canal 0 y el canal 32 en la etapa de mitigación de canales; el primero correspondiente a la componente de corriente continua y el segundo es el ruido de calibración del conversor análogo digital (ADC). Ya cargado en el hardware, una rutina de Python escribe los registros de las cantidades de acumulación para cada rama de DM a detectar. Se considera para el cálculo una frecuencia central de 1000 a 1100 MHz, considerando que la promedio de los radiotelescopios del catálogo de FRBs es, en promedio, 1069 MHz. La rutina permite, asimismo, leer la señal de potencia total dedispersada y el registro de detección de FRB.

5.2. Banco de Pruebas

Para todos los componentes del banco de pruebas se realizaron pruebas básicas de su correcto funcionamiento. A continuación se presentan los resultados relevantes a tener en cuenta para la operación y el análisis.

5.2.1. Oscilador controlado por voltaje (VCO)

En el caso del VCO se midió su amplitud en frecuencia dada una cierta tensión de entrada. El resultado de dicho experimento se muestra en la Figura 5.1. Se observa que la respuesta se comporta bien en tensiones inferiores a los 14 V, luego el incremento de tensión a incremento de frecuencia de oscilación no es del todo lineal.

5.2.2. Generador arbitrario de funciones (AWG)

Se observó en un osciloscopio la forma de onda del AWG para generar un FRB simplificado lineal (Figura 5.2). En la figura se observa una componente de alta frecuencia al inicio de la señal para luego dar paso a la rampa, cuyo final también presenta componentes de alta frecuencia debido al cambio abrupto del nivel de la misma. Se considera esto para posibles ruidos armónicos y es una de las razones que se han añadido los filtros en el montaje.

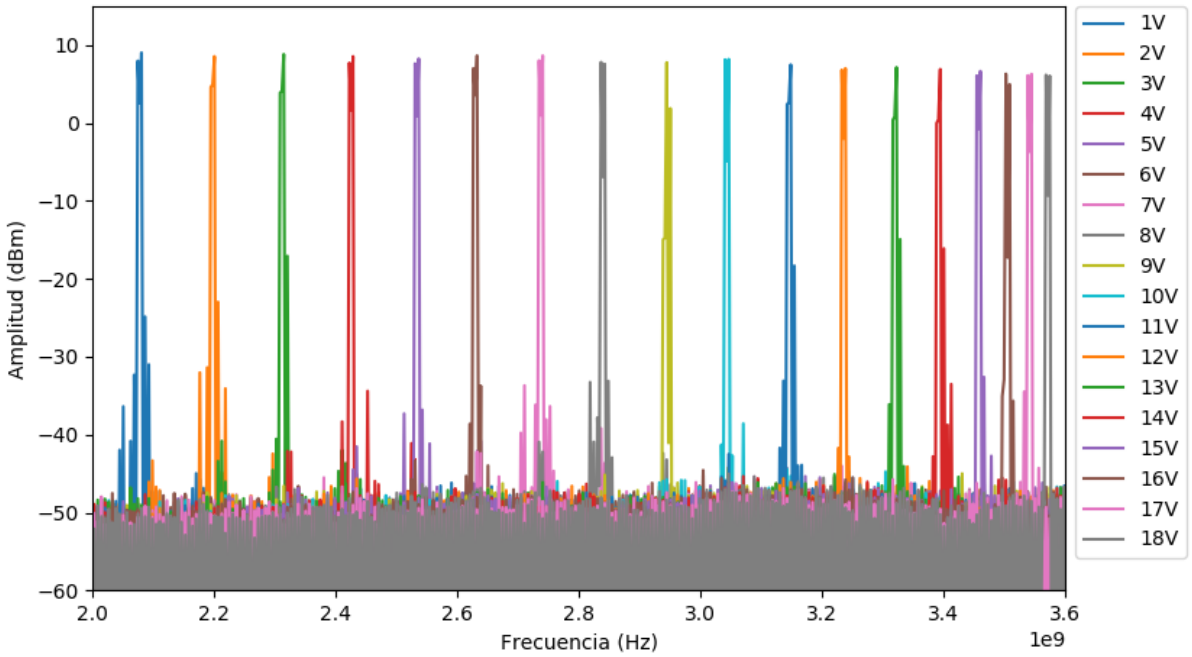

Figura 5.1: Respuesta del VCO en el dominio de la frecuencia. Se observa una relación aproximadamente lineal hasta los 14 V, aproximadamente. En efecto, el interespaciado entre máximos es similar. Con tensiones mayores la relación pierde linealidad.

5.3. Detector

5.3.1. Caracterización: Medida de Dispersión

Recordamos que el detector implementado sigue el diagrama de bloques de la Figura 3.9, donde cada rama paralela está diseñada para detectar una medida de dispersión central determinada. Nos interesa caracterizar la tolerancia que tiene el sistema para detectar si el DM no es exactamente el predefinido a detectar. Por lo tanto, es de interés caracterizar cómo es que varía la amplitud de la señal de potencia total dedispersada frente a variaciones en la frecuencia del generador arbitrario de funciones (f_{AWG}), lo que es equivalente por medio de las Ecuaciones 4.1 y 2.3, a realizar una variación de la medida de dispersión.

En una primera aproximación, se comenzó con un detector centrado en un $DM \sim 78{,}88(\text{pc cm}^{-3})$, esto asociado a una frecuencia del AWG, $f_{AWG} = 3{,}325$ Hz, que se incrementó cada 0.2 Hz. Los resultados se muestran en la Figura 5.3, se destancan dos efectos; primero, el decaimiento progresivo del máximo de la señal y, segundo, que es posible observar que la potencia total se reparte en los instantes de tiempo adyacentes.

Ahora bien, para tener un análisis más generalizado, se toman máximos sucesivos de la señal de potencia total dedispersada repitiendo la señal de la Figura 5.2. De 200 máximos se toma el promedio y se obtiene su magnitud. A la frecuencia del generador arbitrario de

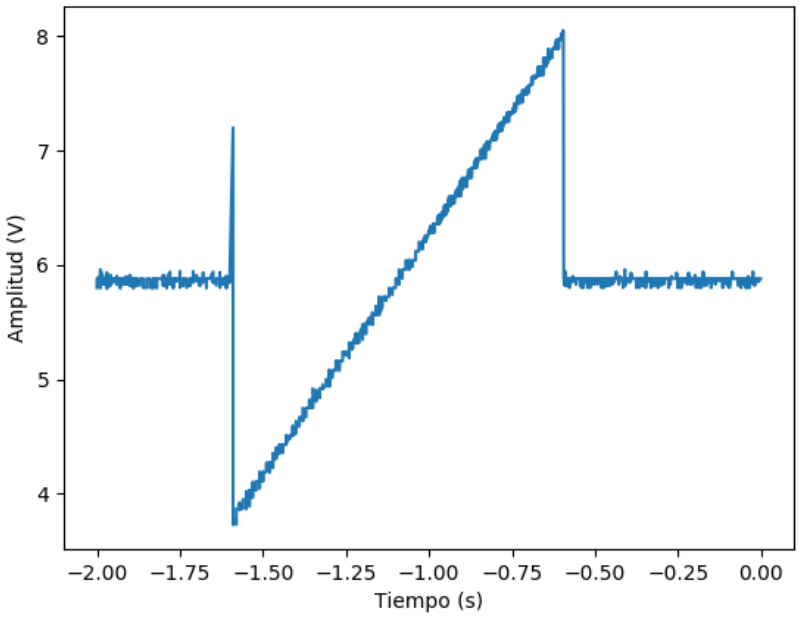

Figura 5.2: Señal rampa generada por el AWG. Se observan cambios bruscos al inicio y final de la rampa. Además tiene un ruido numérico de base. La señal tiene una frecuencia de 1 Hz, usando las ecuaciones 4.1 y 2.3 se asocia a un $DM \sim 262(\mathrm{pc\,cm^{-3}})$

cada caso se asocia su medida de dispersión y finalmente se normaliza. Finalmente, se obtiene el resultado de la Figura 5.4. Estos resultados se han obtenido con la ayuda de las rutinas utilitarias de Python disponibles en los Anexos.

Se observa que el peak de la Figura 5.4 corresponde al valor del DM central (100 %), esto es, el que se ha configurado para detectar. Luego, se incrementa o decrementa dicho valor, observándose que la atenuación de la señal de potencia es asimétrica. Esto se atribuye a que, a bajas velocidades de paso del pulso por el espectro (altos DM) la potencia se acumula en un único canal de frecuencia que es equivalente a que la potencia se distribuye en la señal de potencia total dedispersada. A altas velocidades de paso del pulso por el espectro (bajos DM) la potencia no alcanza a acumularse, de hecho, ocurre un efecto de un integrador con un pulso no periódico.

5.3.2. Caracterización: Relación Señal a Ruido

En este punto nos interesa saber qué tan eficiente es el detector en términos de cuanta relación señal a ruido se gana en observar la señal de potencia total dedispersada en relación a observar el espectro en tiempo real. En primer lugar, se obtiene la relación señal a ruido entre la potencia de entrada del generador “Rohde & Schwartz” y el espectro en tiempo real. La curva se ajusta y se obtiene la siguiente relación

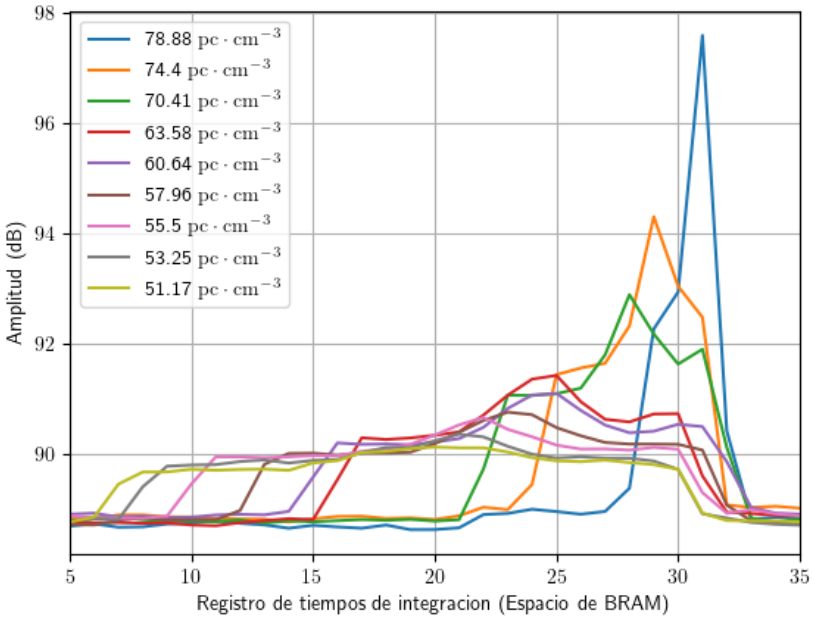

Figura 5.3: Respuesta del la señal de potencia total dedispersada frente a variaciones de frecuencia del AWG. La señal del extremo derecho corresponde al DM central de 78,88(pc cm^{-3}). La frecuencia del AWG se incrementa 0.2 Hz y se obtienen sucesivamente las señales a la izquierda de la original con menor amplitud, asociadas a menores DM.

$$SNR_{spec}[\text{dB}] = 0{,}9565 \times P_{R\&S}[\text{dBm}] + 51{,}53 \tag{5.1}$$

luego, se obtiene la relación señal a ruido entre la potencia de salida del generador "Rohde & Schwartz" y la señal de potencia total dedispersada. La curva se ajusta y se obtiene la siguiente relación

$$SNR_{acc}[\text{dB}] = 0{,}8255 \times P_{R\&S}[\text{dBm}] + 64{,}44 \tag{5.2}$$

,despejando $P_{R\&S}$ en ambas ecuaciones se obtiene una relación entre la relación señal a ruido del espectro en tiempo real y de la señal de potencia total dedispersada.

$$SNR_{acc} = 0{,}863042 \times SNR_{spec} + 19{,}9674 \tag{5.3}$$

Ahora bien, si consideramos que nuestro espectro es una señal de potencia P_{Spec} al sumar todos los canales espectrales en realidad estamos tomando 64 veces dicha potencia. Además, en la Figura 4.19 se observa que existe una transformación del valor de la suma en el bloque entre el último sumador y el acumulador. En efecto, se vuelve a reinterpretar el valor a 64 bits descartando los valores menos significativos. En ROACH 1 al tener 4 canales se tienen 2 etapas de sumas lo que supone eliminar 2 bits menos significativos. Esta eliminación de bits

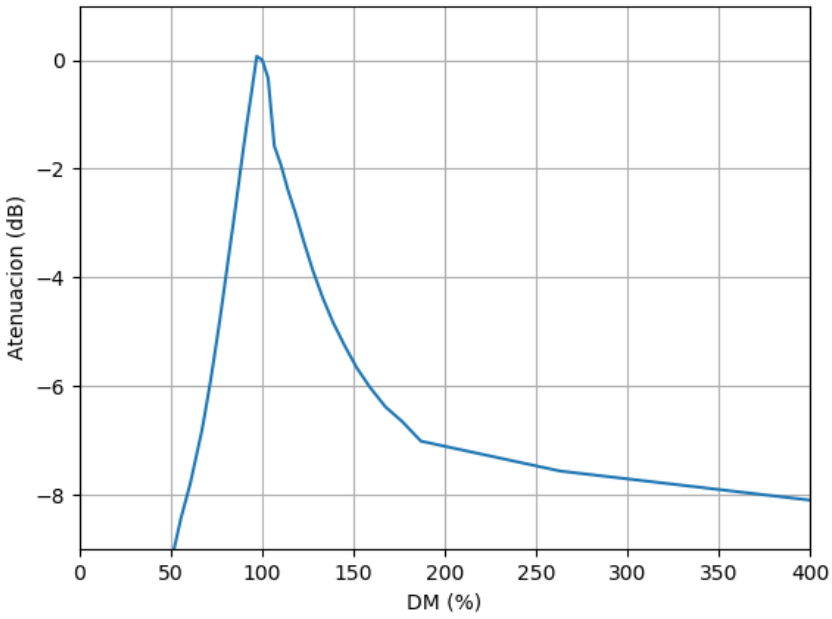

Figura 5.4: Atenuación de la señal de potencia total dedispersada frente a variaciones porcentuales de la medida de dispersión normalizada (DM). Se observa que dicha atenuación no es simétrica. A menores DM del unitario la atenuación es aproximadamente lineal, mientras que a mayores DM el decaimiento es exponencial.

es equivalente a una ponderación de nuestro número por 2^n donde n es la cantidad de bits eliminados. Luego, teóricamente se tendrá que

$$P_{total} = P_{Spec} \times 64 \times 4 \tag{5.4}$$

lo que expresado en decibeles es,

$$P_{total}[\text{dB}] = P_{Spec}[\text{dB}] + 10\log(64 \times 4) \tag{5.5}$$

La comparación de ambos escenarios se muestra en la Figura 5.5, donde se desprende que, en primer lugar, existe en efecto una ganancia. Dicho de otro modo, un FRB que se encuentra en el ruido (SNR=0) imperceptible en la señal de entrada, será visto como una señal con SNR = 20 dB en la potencia total dedispersada. Experimentalmente, esta ganancia se ve atenuada a mayor potencia del generador y se asemeja al caso teórico cuando disminuye.

Naturalmente, se observa que el resultado experimental y teórico no coinciden. Esto se atribuye a que a altos valores de potencia de la señal de entrada, la potencia total dedispersada comienza a mostrar un mayor rizado en su piso de ruido, impactando en una peor SNR.

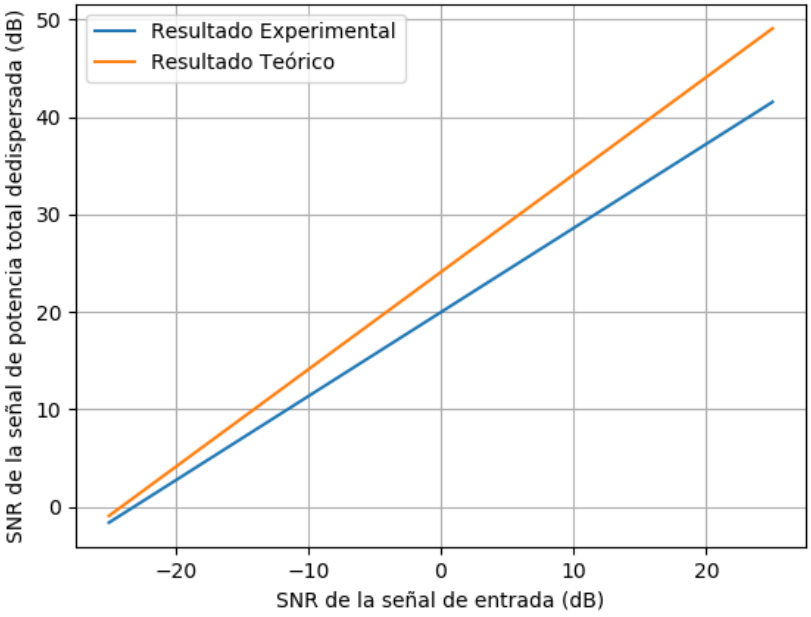

Figura 5.5: Relación señal a ruido (SNR) del espectro en tiempo real versus la relación señal a ruido de la potencia total dedispersada. En azul se muestra el resultado experimental y en naranja el teórico. A menor SNR del espectro los modelos se asemejan. De cualquier forma, el proceso de dedispersión implica obtener un aumento de 20 dB en relación señal a ruido.

5.3.3. Análisis de Paralelismo: Cantidad de detectores paralelos necesarios

Se ha evidenciado en la Sección 5.3.1 cómo se atenúa la medida de dispersión para un único detector implementado, considerándose aceptable que un detector fijo observe hasta a -3 dB de atenuación en relación a su DM central. Ahora es de interés saber cuántos detectores deben ser colocados en paralelo para cubrir el rango completo de medidas de dispersión detectadas a la fecha, según el catálogo en línea. Para ello, cosideramos la atenuación para un DM normalizado de la Figura 5.4, aplicando esto a valores dentro del rango de DM detectados de 100 a 2500 (pc cm^{-3}) (inclusive) se obtiene lo mostrado en la Figura 5.6. Notamos que a mayor medida de dispersión central del dedispersor construido, mayor es su rango de detección.

El hecho que el rango de DM incremente con la medida de DM central supone un resultado de gran importancia. En efecto, se demuestra que se puede capturar la totalidad del rango de DM detectados a la fecha con una cantidad limitada de detectores en paralelo. En efecto, en la Tabla 5.2 se han calculado 10 detectores en paralelo configurados con la columna $DM_{central}$ para cubrir todo el rango de DM desde 100 hasta 2695 (pc cm^{-3}).

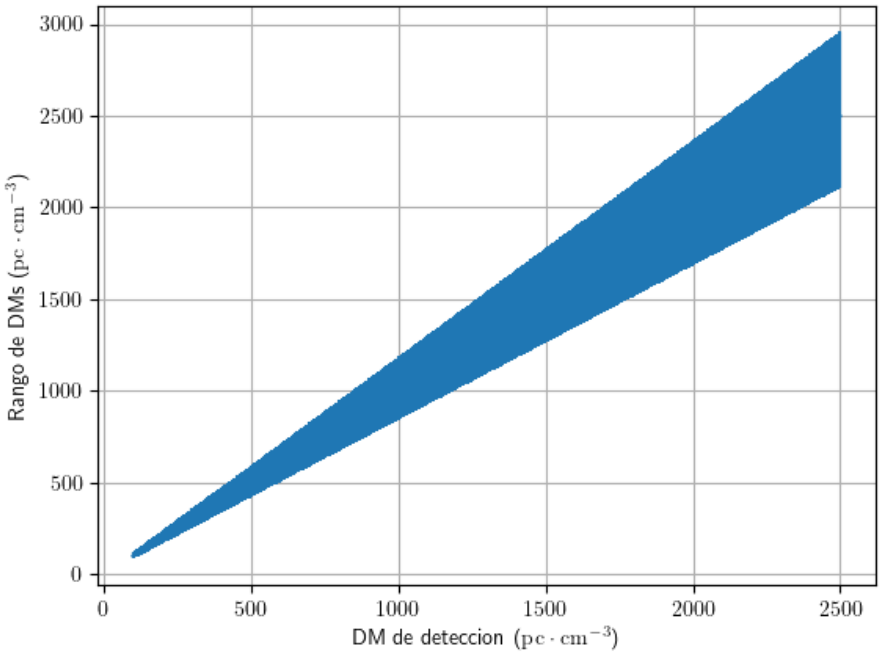

Figura 5.6: Sensibilidad de la medida de dispersión según el valor central de DM. Se observa que a mayor medida de dispersión de la detección, mayor será la sensibilidad del instrumento. Se ha considerado sensible si la atenuación no sobrepasa los 3 dB.

5.3.4. Implementación del Paralelismo

En la plataforma ROACH 1, dado sus recursos lógicos, sólo es posible compilar e implementar un modelo de dedispersor con una medida de dispersión fija y una FFT de 4 canales paralelos. Para implementar paralelismo de detectores es necesario migrar el modelo a su sucesora, ROACH 2.

En la plataforma ROACH 2, se compila e implementa un modelo con una FFT de 8 canales paralelos en su salida y permite implementar múltiples detectores paralelos, limitado a la cantidad de tablas de verdad (LUTs, por sus siglas en inglés) disponibles en el chip, observado en los informes de compilación disponible en el Anexo D.

Con los informes de uso de recursos de uno, dos, siete y, finalmente, los diez detectores en paralelo se construye el gráfico de la Figura 5.7, que da cuenta del uso porcentual de LUTs en el chip. Se observa que es necesario un mínimo del 12 % para compilar un detector en paralelo, con un máximo teórico de 12 detectores.

Es importante mencionar que se decide implementar sólo 10 detectores en paralelo, en primer lugar, porque se cubren todos los rangos de DM detectados a la fecha con ellos. En segundo lugar, se evita como regla general, utilizar la totalidad de los recursos, pues el compilador del modelo, encargado de posicionar cada bloque en el chip, no logra encontrar un circuito posible y deben aplicarse estrategias de posicionamiento más detalladas que escapan

Tabla 5.2: Medidas de dispersión centrales para cubrir todo el rango de DM a la fecha. A cada $DM_{central}$ se le asocia su límite inferior y superior, donde la señal tiene una atenuación de 3dB.

$DM_{inferior}$(pc cm^{-3})	$DM_{central}$(pc cm^{-3})	$DM_{superior}$(pc cm^{-3})
100.00	118.60	140.37
140.00	166.05	196.51
190.00	225.35	266.70
260.00	308.37	364.95
360.00	426.98	505.32
500.00	593.02	701.83
700.00	830.23	982.57
980.00	1162.33	1375.60
1370.00	1624.88	1923.03
1920.00	2277.21	2695.05

del enfoque de este trabajo.

5.4. Modelo Final

5.4.1. Implementación del modelo

La implementación del modelo final diseñado en *Simulink*, para ser implementado en ROACH 2, se encuentra en el Anexo E. La versión contempla 10 detectores en paralelo (detección de 10 DM distintos), con una FFT y bloque dedispersor paralelo de 8 canales de entrada y salida simultáneos. Cada uno de los 10 detectores maneja una cantidad de acumulaciones distinta escrita en un registro (Figura 4.7) y una señal de potencia total dedispersada (bloques "ACC" del modelo (Figura 4.19)) asociada a la detección. Notamos que la arquitectura sigue el diagrama de bloques de la Figura 3.9, presentado en la metodología.

5.4.2. Configuración del modelo

El modelo se configura por medio de un código Python el cual puede ser consultado en el Código B.7. En él se calcula la frecuencia central de operación y las medidas de dispersión para cada detector paralelo. El programa se encarga de enviar a ROACH 2 el número de acumulación de cada DM (por medio de la librería "Corr").

El siguiente paso es una rutina una inicialización, donde se obtienen muestras de las señales de potencia total dedispersada y se determinan los umbrales de detección de un FRB, como un múltiplo de un número de desviaciones de la señal de ruido base. Para pruebas de laboratorio, se sugiere el uso de 5 desviaciones, teniendo en mente que para su implementación en terreno se utilizarán 3.

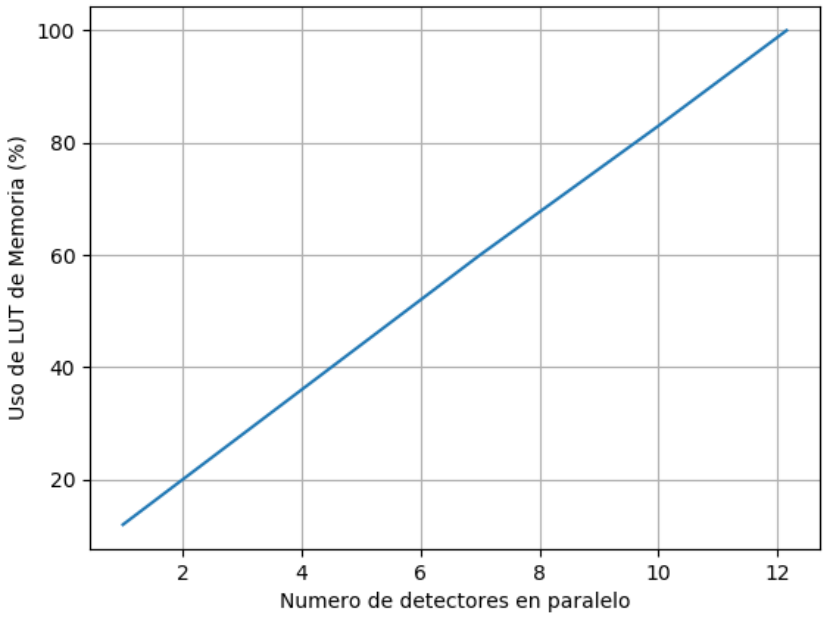

Figura 5.7: Uso porcentual de LUTs de memoria según la cantidad de detectores en paralelo para ROACH 2. Se observa que basalmente se utiliza cerca del 12 % de este recurso y luego crece proporcional a los detectores paralelos. La máxima capacidad teórica son 12 detectores en paralelo.

Finalmente, se obtiene continuamente la señal de potencia total dedispersada de 10 canales en paralelo. Un ejemplo de una detección realizada con el banco de pruebas se observa en la Figura 5.8. En ella el DM = 128 (pc cm^{-3}) presenta una detección, las señales adyacentes presentan cierto grado de perturbaciones. Cada señal tiene 512 puntos (por la capacidad de la memoria) correspondientes cada uno al tiempo de integración (o número de acumulaciones) del espectro, en primer lugar, acumulado y, luego, dedispersado. Como las medidas de dispersión más pequeñas requieren una menor cantidad de acumulación, la tasa de refresco de su gráfico será mayor que para medidas de dispersión mayores.

5.4.3. La señal de potencia total dedispersada

Es importante mencionar la importancia que toma la señal de potencia total dedispersada en nuestro análisis y resultado. Un único punto de la señal representa la suma de todo un espectro, condensándose en un único valor lo ocurrido en todo el tiempo de integración para nuestro análisis de detección. Asimismo, podemos monitorear esta señal en tiempo real, pues la tasa de refresco del gráfico de la señal de potencia total dedispersada (código python y comunicación con roach 2) es mayor al tiempo en que se refresca la memoria completamente en el chip.

Otro factor relevante, es que un tono en el espectro puede aparecer, por ejemplo, por efecto de radio interferencias. Por construcción del dedispersor, se observa en la señal de potencia

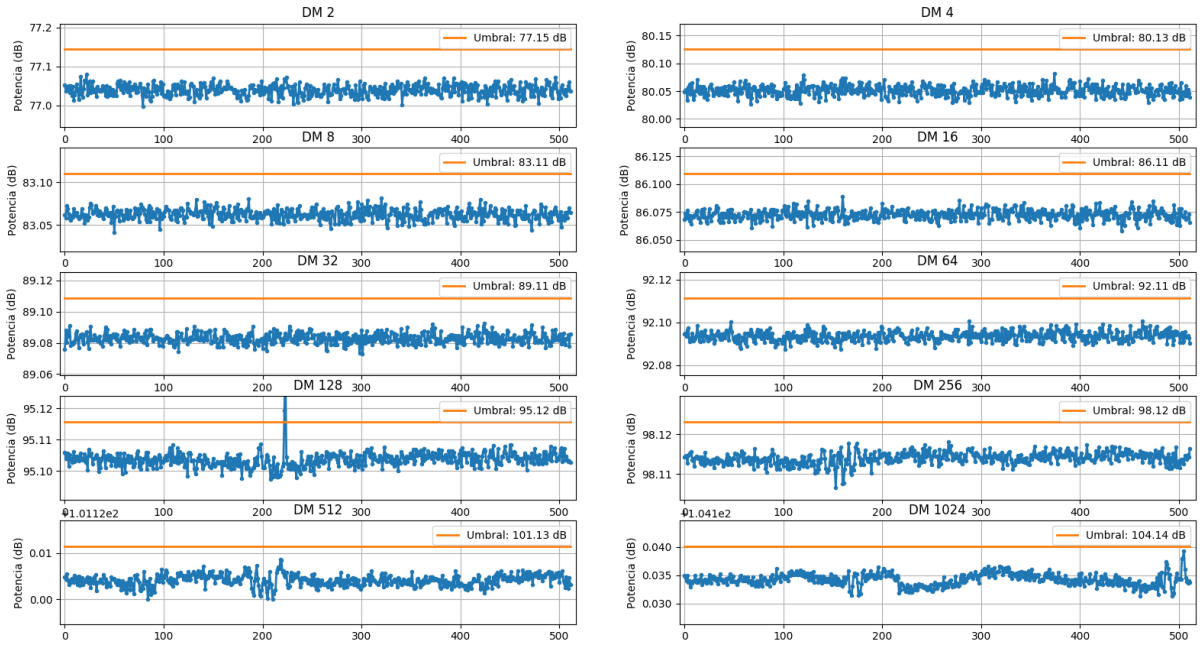

Figura 5.8: Señales de potencia total dedispersada del 10 detectores simultáneos. Se detecta una señal en el DM = 128. Se observan, asimismo, perturbaciones en los DM adyacentes mayores (más lentos) las perturbaciones en DM menores (más rápidos) no alcanzan la escala temporal de la captura. Al eje de las abscisas para el DM 512 y 1024 se le han restado los valores 1.0112e2 y 1.041e2, respectivamente.

total dedispersada como un cambio (aumento) en el piso de ruido de la misma. Si la señal tuviera una potencia considerable podría activar la detección, sin embargo, la información histórica de la señal de potencia total dedispersada bastaría para descartar este falso positivo.

Como cada punto de la señal se basa entre tiempos sucesivos de señal de validez, los menores DM tienen una tasa de refresco (en memoria "ACC") mayor. Por otro lado, los DM mayores pueden tener una tasa de refresco de minutos, inclusive. Es por este factor que se observa en la Figura 5.8 una detección y un rizado en los DM superiores, pero no en los inferiores. En efecto, la velocidad de captura de la imagen es más lenta que la tasa de refresco de las memorias de los DM inferiores.

5.5. Conclusiones

En el banco de pruebas, se han realizado pruebas a los componentes, donde se han identificado regiones de operación óptimas para el VCO y, por otro lado, se observan componentes de alta frecuencia al inicio de la señal de rampa del AWG.

Para el detector se ha caracterizado la disminución del máximo de la señal de potencia total dedispersada para variaciones en el DM central de detección, donde la relación es asimétrica por la forma en que se realiza la acumulación de potencia. Por otro lado, se ha comprobado la

utilidad de la técnica de dedispersión, permitiendo que la relación señal a ruido de la potencia total dedispersada sea hasta 20 dB más alta que la relación señal ruido de la entrada, lo que claramente facilita el proceso de detección.

Para implementar el paralelismo del detector se ha calculado que son necesarios 10 detectores en paralelo para detectar el rango de DM de todos los FRB detectados a la fecha (100 a 2695 ($\mathrm{pc\,cm^{-3}}$)), gracias a que, por construcción, el rango de DM detectable incrementa con la medida de dispersión central del detector. Por su cantidad de recursos, ROACH 1, sólo admite 1 detector, sin embargo, es posible implementar la totalidad de detectores calculados en ROACH 2, dejando cerca de un 20 % de los recursos lógicos (LUTs) libres, utilizando una FFT de 64 canales, un dedispersor paralelo de 8 entradas y salidas y la posibilidad de monitorear cada una de las diez señales de potencia total dedispersada asociada a cada detector, mediante una rutina de Python en un ordenador de escritorio, por la cual también es posible configurar el número de acumulaciones sobre cada detector.

Finalmente, se reconoce la importancia de la señal de potencia total dedispersada, la cual puede ser monitoreada en tiempo real pues su tasa de refresco es menor que el tiempo de lectura de la rutina Python. La señal puede dar cuenta de una detección válida o detectar cuando ocurren casos de radiointerferencias, pues ocurre un cambio en el nivel base de la misma.

Conclusión

Se ha implementado un banco de pruebas para emular el comportamiento de un FRB en banda base, con un ruido de observación asociado. El montaje permite emular una cierta medida de dispersión, cambiando la frecuencia de operación del generador arbitrario de funciones (f_{AWG}) , y alterar la intensidad de la señal gaussiana, alterando la potencia de salida en dBm del generador. La construcción del banco supuso agregar atenuadores y filtros para ajustar las potencias de entrada salida de los componentes y eliminar posibles componentes armónicos, respectivamente. Con todo lo anterior, se tiene una señal lista para ser digitalizada por el conversor análogo digital (ADC) de ROACH.

Se ha diseñado una implementación de las funcionalidades de acumulación, eliminación de canales, dedispersión y la detección de un FRB. La etapa de acumulación, define el parámetro número de acumulaciones; que define el tiempo de integración para una medida de dispersión determinada. Se ha implementado el bloque de eliminación de canales para el caso de varios hilos paralelos y el porqué de la interconexión de los componentes basado en el número de canal eliminado. El núcleo del trabajo de diseño en *Simulink* es la implementación del bloque dedispersor paralelo parametrizado por el "vector de dedispersión", conformado por dedispersores de un canal que reciben cada uno la fragmentación del parámetro principal en forma de vectores de retardos de hasta 1024 valores cada uno. Se ha hecho la analogía de cómo el dedispersor de 1 canal implementa el modelo de pilas presentado en la Sección 3.4.2, en efecto, cada pila es un bloque detector básico y el selector de canales es un conjunto multiplexor-contador que recorre secuencialmente cada pila. Finalmente, se calcula la suma de las contribuciones de todos los canales espectrales dedispersados para un tiempo de acumulación determinado, generando una nueva señal llamada **señal de potencia total dedispersada**. Finalmente, el bloque de detección de FRB se implementa comparando con un valor umbralo que de ser superado, dispara una alerta de detección.

Para el conjunto banco de pruebas y detector, se ha caracterizado la disminución del máximo de la señal de potencia total dedispersada para variaciones en el DM central de detección, donde la relación es asimétrica por la forma en que se realiza la acumulación de potencia. Por otro lado, se ha comprobado la utilidad de la técnica de dedispersión, permitiendo que la relación señal a ruido de la potencia total dedispersada sea hasta 20 dB más alta que la relación señal ruido de la entrada. Se ha calculado que son necesarios 10 detectores en paralelo para detectar el rango de DM de todos los FRB detectados a la fecha (100 a 2695 ($\mathrm{pc\,cm^{-3}}$)).

Los diez detectores en paralelo se implementan en ROACH 2, dejando cerca de un 20 % de

los recursos lógicos (LUTs) libres para desarrollos futuros. Se utiliza una FFT de 64 canales, un dedispersor paralelo de 8 entradas y salidas.

Se reconoce la importancia de la señal de potencia total dedispersada, las cuales (10 de ellas) pueden ser monitoreadas en tiempo real pues su tasa de refresco es menor que el tiempo de lectura de la rutina Python. La señal otorga información que puede dar cuenta de una detección válida o de un falso positivo.

Por los párrafos anteriores, se concluye que la hipótesis es correcta, en efecto, es posible implementar una detección de ráfagas de radio rápidas en una plataforma FPGA para un amplio rango de medidas de dispersión, utilizando su alta capacidad de procesamiento paralelo y en tiempo real.

Capítulo 6

Trabajo Futuro

Dadas la conclusiones de esta memoria, se presentan nuevos objetivos y oportunidades abiertas a trabajo futuro.

6.1. Correcciones al modelo

En el dedispersor paralelo, se sugiere remover el parámetro de bits del contador, el cual fue utilizado para corroborar errores, durante el diseño. Otro factor relevante a simplificar es mover el bloque de eliminación de canales antes de la etapa de acumulación, esto permitirá implementarla una vez y no para cada rama de detección, otorgando la misma funcionalidad.

6.2. Recuperación de datos

Es importante mencionar, que la implementación presentada es un **detector**. Esto, en el sentido de que su finalidad es identificar si es que ha ocurrido o no una ráfaga de radio rápida, aislando su principal característica: la medida de dispersión. Sin embargo, para una investigación *offline* (no en tiempo real) es de interés disponer de los datos originales del fenómeno, esto es, de los datos crudos que fueron digitalizados a la salida del ADC. Por tanto, se sugiere la implementación de una memoria circular (con el recurso no utilizado) dentro de la FPGA que descargue a un medio de almacenamiento externo luego de que la señal de detección fue activada. La tecnología puede ser un disco de almacenamiento masivo o la transferencia de datos por el puerto 10 GbE de ROACH 2 a un ordenador compatible.

6.3. Monitoreo de la señal de potencia total dedispersada

Hemos mencionado la importancia de la señal de potencia total dedispersada en nuestro análisis, permitiendo identificar incluso falsos positivos en la detección. Además, su tiempo de lectura es comparable a la escritura de los datos en un disco duro. Por lo tanto, y valiéndose de una conexión de internet de alta velocidad desde el radiotelescopio, estos datos (de ,al menos, 10 medidas de dispersión centrales) pueden ser visualizados en tiempo real en cualquier navegador del mundo.

6.4. Adaptación a Radiotelescopio

Para integrar el detector al radiotelescopio es importante considerar la frecuencia central de operación del mismo y su ancho de banda. Esto definirá, en última instancia, qué región de las ráfagas se estará observando i.e. regiones más rápidas o lentas a través del espectro. Esto no se encuentra desacoplado de otros problemas, en efecto, es importante considerar que a menor frecuencia de operación la cantidad de memoria (circular o no) requerida para capturar la totalidad del fenómeno es mayor, sin embargo, a altas frecuencias la señal de potencia total dedispersada podría no ser monitoreable en tiempo real, debido a su alta velocidad de paso por el espectro.

6.5. Efectos de propagación de las FRB

Para la emulación de un FRB se ha considerado a la medida de dispersión como el parámetro central a analizar (siendo también el común acuerdo de la comunidad científica). Sin embargo, existen otros efectos en la propagación de los FRB, como el efecto de *Scincillation* y *Scattering*, alargando la forma del pulso original en frecuencia y/o tiempo. Por su construcción se prevé que estos efectos redistribuirán la potencia total de la señal de potencia total dedispersada de forma análoga al variar el DM central de un detector. Esto podría ser evaluado con un nuevo banco de pruebas, más especializado, que considere éstos y otros efectos.

6.6. Rendimiento

Otro punto de interés (que escapa al alcance de este trabajo) es comparar el rendimiento del detector/dedispersor implementado en FPGA frente a dedispersores implementados en GPUs o sistemas afines. Se deberán idear métricas comunes que traduzcan el desempeño de una tecnología para ser comparable con la otra, dejando todo lo demás constante.

Bibliografía

[1] S. Colgate, "Electromagnetic pulse from supernovae," *The Astrophysical Journal*, vol. 198, pp. 439–445, 1975.

[2] E. Petroff, J. Hessels, and D. Lorimer, "Fast radio bursts," *The Astronomy and Astrophysics Review*, vol. 27, no. 1, p. 4, 2019.

[3] D. Melrose, "Coherent emission mechanisms in astrophysical plasmas," *Reviews of Modern Plasma Physics*, vol. 1, no. 1, p. 5, 2017.

[4] A. Hewish, S. J. Bell, J. D. Pilkington, P. F. Scott, and R. A. Collins, "Observation of a rapidly pulsating radio source," *Nature*, vol. 217, no. 5130, p. 709, 1968.

[5] W. Burns and B. Clark, "Pulsar search techniques," *Astronomy and Astrophysics*, vol. 2, pp. 280–287, 1969.

[6] D. Lorimer, M. Bailes, M. McLaughlin, D. Narkevic, and F. Crawford, "A bright millisecond radio burst of extragalactic origin," *Science*, vol. 318, no. 5851, pp. 777–780, 2007.

[7] E. Petroff, E. Barr, A. Jameson, E. Keane, M. Bailes, M. Kramer, V. Morello, D. Tabbara, and W. van Straten, "Frbcat: the fast radio burst catalogue," *arXiv preprint arXiv:1601.03547*, 2016.

[8] W. Jun, C. Mao-zheng, P. Xin, and W. Zhi-qiao, "Research and test of real-time search algorithm of fast radio bursts based on gpu acceleration," *Chinese Astronomy and Astrophysics*, vol. 42, no. 2, pp. 313–324, 2018.

[9] M. Bailes, A. Jameson, C. Flynn, T. Bateman, E. D. Barr, S. Bhandari, J. D. Bunton, M. Caleb, D. Campbell-Wilson, W. Farah, and et al., "The utmost: A hybrid digital signal processor transforms the molonglo observatory synthesis telescope," *Publications of the Astronomical Society of Australia*, vol. 34, 2017.

[10] N. Clarke, L. D'Addario, R. Navarro, T. Cheng, and J. Trinh, "An architecture for incoherent dedispersion," tech. rep., CRAFT Memo 6, 2011.

[11] J. Cordes and M. A. McLaughlin, "Searches for fast radio transients," *The Astrophysical Journal*, vol. 596, no. 2, p. 1142, 2003.

[12] CASPER, *Tutorials*. https://casper.ssl.berkeley.edu/wiki/Tutorials.

[13] Xilinx, "System generator for dsp *Reference Guide*," tech. rep., Xilinx inc., 2009.

[14] Noisewave, *Application Note - Noise*. http://www.noisewave.com/faq.pdf.

Apéndice A

Códigos Matlab

A.1. Detector

Código A.1: Posiciona y conecta un bloque básico detector (Detector block v2) dado un canal - un retardo y bits de contador

```
1
2
3  function detector_block_init_v2(blk,varargin)
4
5  if same_state(blk,varargin(:)), return, end %if it has the same state that
       already has get out
6
7  % disable link to the library so you can edit freely this block when you
8  % use it in a model:
9  munge_block(blk,varargin(:));
10
11 %Get parameters into variables
12 chn = get_var('chn',varargin{:}); % get variable from the varargin given
       to the function
13 lat = get_var('lat',varargin{:}); % pay attention to brackets
14 nbits = get_var('nbits',varargin{:});
15
16 %Erase every line in block
17 delete_lines(blk);
18
19 % Initialize blocks (by reusing them from library)
20
21 %Inputs
22 in_x = 30; % x block dimension
23 in_y = 14; % y block dimension
24 reuse_block(blk,'Data','built-in/inport','Port','1','Position',
       get_position(100,100,in_x,in_y));
25 reuse_block(blk,'Valid','built-in/inport','Port','2','Position',
       get_position(100,200,in_x,in_y));
26
27 %Constants
28 c_x = 30; % x block dimension
```

```
c_y = 26; % y block dimension
reuse_block(blk,'Const1','xbsIndex_r4/Constant','const',num2str(chn),'
    n_bits','16','bin_pt','0','Position',get_position(200,300,c_x,c_y));

%Counter
cnt_x = 50;
cnt_y = 56;
reuse_block(blk,'Counter1','xbsIndex_r4/Counter','n_bits',num2str(nbits),'
    Position',get_position(200,200,cnt_x,cnt_y));

%Relational
r_x = 50;
r_y = 56;
reuse_block(blk,'Relational1','xbsIndex_r4/Relational','latency','0','
    Position',get_position(300,200,r_x,r_y));

%Expression
e_x = 50;
e_y = 56;
reuse_block(blk,'Expression1','xbsIndex_r4/Expression','expression','a & b
    ','Position',get_position(400,250,e_x,e_y));

%Delay
d_x = 60;
d_y = 56;
reuse_block(blk,'Delay1','xbsIndex_r4/Delay','latency',num2str(lat),'
    Position',get_position(500,200,d_x,d_y));

%Output
o_x = 30;
o_y = 14;
reuse_block(blk,'Out1','built-in/outport','Port','1','Position',
    get_position(600,200,in_x,in_y));

% Connect Blocks

%Counter Inputs
add_line(blk,'Valid/1','Counter1/1','autorouting','on')

%Relational Inputs
add_line(blk,'Counter1/1','Relational1/1','autorouting','on')
add_line(blk,'Const1/1','Relational1/2','autorouting','on')

%Expression Inputs
add_line(blk,'Relational1/1','Expression1/1','autorouting','on')
add_line(blk,'Valid/1','Expression1/2','autorouting','on')

%Delay Inputs
add_line(blk,'Data/1','Delay1/1','autorouting','on')
add_line(blk,'Expression1/1','Delay1/2','autorouting','on')

%Output Inputs
add_line(blk,'Delay1/1','Out1/1','autorouting','on')

%Final Setup

```

```
% use this to show on model the parameters of your block
set_param(blk, 'AttributesFormatString',['Channel: ',num2str(chn),', Delay
    : ',num2str(lat)])

% Finds any block that it's not wired and removes it
clean_blocks(blk);

% Save state and parameters of block
save_state(blk,varargin{:});

end
```

Código A.2: Posiciona y conecta un bloque detector (Dedispersor v2) dado un vector de DM

```


function honest_detector_v2(DM_Vector)

%DM_Vector = [0:1:15]                          %Example DM Vector (for
    debugging)
%nbits = 8                                     %Example nbits (for debugging
    )

n_channels = length(DM_Vector);               % Number of channels and basic
     blocks
nbits = log2(n_channels);                     % Number of bits of counter

[a,b] = xInport('Data','Valid');              % In ports of subsystem
out = xOutport('Output');                     % Out ports of subsystem


%Create names for detector blocks
det_name = {};
for i = 1:n_channels
    det_name{end+1} = strcat('detector',num2str(i));  %det_name{end+1} = '
    detector<number>'
end

% Create n_channels signals to connect at detector blocks outputs with
% multiplexer block
mux_signals = {};
for i = 1:n_channels
    mux_signals{end+1} = xSignal;
end

% Create n_channels detector blocks with names det_names, using xBlock.
% Each detector block it's connected to a 'mux signal' output.
for i = 1:n_channels
    det_name(i) = xBlock('honestlibrary/detector_block_v2',struct('chn',i
    -1,'lat',DM_Vector(i)+1,'nbits',nbits),{a,b},{mux_signals{i}});
end

% Create counter to map muxes
sel = xSignal;                                                          %
    Create selection signal from counter to mux
cnt1 = xBlock('Counter',struct('n_bits',nbits,'en','on'),{b},{sel});    %
```

```
    Create counter for mux selection
38
39 % Separate cases:
40 % 1.- Single multiplexer
41 % 2.- Two stage multiplexer
42 % This comes from the fact that Xilinx's mux max number of inputs it's 32
43
44
45 % Case 1: Single mux
46 if n_channels <= 32
47     mux_out = xSignal;                  % Output signal of multiplexer
48     mux_input = [{sel},mux_signals];  % Add 'sel' signal to the first
    element of mux inputs
49     mux1 = xBlock('Mux',struct('inputs',n_channels),mux_input,{mux_out});
       % Create Multiplexer with #(n_channels) inputs
50     out.bind(mux_out);
       % Bind mux_out with out port of subsystem
51 end
52
53
54
55 % Case 2: Two stage multiplexation
56 if (n_channels <= 1024) && (n_channels > 32)
57     number_muxes = ceil(n_channels/32);          % Number of muxes to
    create equal to the number of inputs of selector mux
58     bits_selector = ceil(log2(n_channels))-5;   % Number of bits required
    from the counter signal to the selector mux from MSB to LSB
59
60     common_mux = xSignal;    % Signal to common muxs
61     stage_mux = xSignal;     % Signal to stage mux which selects a single
    mux for output
62
63     slice_common = xBlock('Slice',struct('nbits',5,'mode','Lower Bit
    Location + Width'),{sel},{common_mux});                % Select bits of
    'Count' common muxes
64     slice_stage = xBlock('Slice',struct('nbits',bits_selector,'mode','
    Upper Bit Location + Width'),{sel},{stage_mux});      % Select bits of '
    Count' stage mux
65
66     % Create number_muxes signals for muxes' outputs and names/tags
67     mux_out = {};
68     mux_name = {};
69     for i = 1:number_muxes
70         mux_out{end+1} = xSignal;
71         mux_name{end+1} = strcat('Multiplexer',num2str(i));
72     end
73
74     % Create muxes of first stage
75     for i = 1:number_muxes
76         mux_name(i) = xBlock('Mux' , struct('inputs','32') ,  [{common_mux
    }, mux_signals(1+32*(i-1) : 32+32*(i-1))] , mux_out(i) );
77     end
78
79     % Create mux of second stage
80
81     mux_stage = xBlock('Mux', struct('inputs',number_muxes),[{stage_mux},
```

```
    mux_out],{out});
82
83
84
85 end
```

Código A.3: Posiciona y conecta un bloque detector en paralelo (Parallel Dedispersor v2) dado un vector de DM y el número de entradas

```
1
2
3 function honest_parallel_detector_init_v2(blk,varargin)
4
5 if same_state(blk,varargin(:)), return, end % if it has the same state
    that already has, just get out
6
7 % disable link to the library so you can edit freely this block when you
8 % use it in a model:
9 munge_block(blk,varargin(:));
10
11 %Get parameters into variables
12 n_inputs = get_var('n_inputs',varargin{:});     % get variable from the
    varargin given to the function
13 DM_Vector = get_var('DM_Vector',varargin{:});   % pay attention to
    brackets
14
15 %Examples (for debugging)
16 % n_inputs = 4;
17 % DM_Vector = [1:1:64];
18
19 n_channels = length(DM_Vector); % Number of total channels from the FFT
20
21 channels_input = n_channels/n_inputs; % Number of channels per input
22
23 DM_Matrix = zeros(n_inputs,channels_input); % Matrix containing
    dedispersion vectors
24
25 % We take advantage of Matlab's native definition of matrix single-indexes
26 % Then we get a row adjusted vector for every stream of FFT
27 for i=1:n_channels
28     DM_Matrix(i) = DM_Vector(i);
29 end
30
31 %Erase every line in block
32 delete_lines(blk);
33
34 % Initialize blocks (by reusing them from library)
35
36 %Inputs
37 in_x = 30; % x block dimension
38 in_y = 14; % y block dimension
39 reuse_block(blk,'Valid','built-in/inport','Port','1','Position',
    get_position(0,0,in_x,in_y));
40 for i=1:n_inputs
41     port_num = num2str(1+i);
42     port_name = strcat('Data',num2str(i));
43     reuse_block(blk,port_name,'built-in/inport','Port',port_num,'Position'
```

```
    ,get_position(0,200+200*(i-1),in_x,in_y));
end

%Dedispersor Blocks
d_x = 115;
d_y = 112;
for i=1:n_inputs
    dd_name = strcat('Dedispersor_v2_',num2str(i));
    reuse_block(blk,dd_name,'HonestLibrary/Dedispersor_v2','DM_Vector',
    mat2str(DM_Matrix(i,:)),'Position',get_position(200,200*i,d_x,d_y));
end

%Outputs
o_x = 30;
o_y = 14;
for i=1:n_inputs
    port_num = num2str(i);
    port_name = strcat('Out',num2str(i));
    reuse_block(blk,port_name,'built-in/outport','Port',port_num,'Position
    ',get_position(400,250+200*(i-1),o_x,o_y));
end


%Connect Blocks

%Valid input to Dedispersor blocks
for i=1:n_inputs
   dd_name = strcat('Dedispersor_v2_',num2str(i),'/2');
   add_line(blk,'Valid/1',dd_name,'autorouting','on');
end

%Input ports to Dedispersor blocks
for i=1:n_inputs
    in_name = strcat('Data',num2str(i),'/1');
    dd_name = strcat('Dedispersor_v2_',num2str(i),'/1');
    add_line(blk,in_name,dd_name,'autorouting','on');
end

%Output ports to Dedispersor blocks
for i=1:n_inputs
    dd_name = strcat('Dedispersor_v2_',num2str(i),'/1');
    out_name = strcat('Out',num2str(i),'/1');
    add_line(blk,dd_name,out_name,'autorouting','on');
end


%Final Setup

% use this to show on model the parameters of your block
set_param(blk, 'AttributesFormatString',['Inputs: ',num2str(n_inputs),',
    Channels: ',num2str(n_channels)])

% Finds any block that it's not wired and removes it
clean_blocks(blk);

% Save state and parameters of block
```

```
96 save_state(blk,varargin{:});
```

A.2. Funciones auxiliares

Código A.4: Transforma formato de posición de objeto

```
1
2 function pos = get_position(x_coor,y_coor, x_min, y_min)
3
4 pos = [x_coor y_coor x_coor+x_min y_coor+y_min];
```

Código A.5: Redistribuye el vector de DM en una cantidad n de canales

```
1
2
3 function DM_matrix = dedispersion_vectors(n_inputs,DM_vector)
4
5 n_channels = length(DM_vector); % Number of total channels from the FFT
6
7 channels_input = n_channels/n_inputs; % Number of channels per input of
    total DM
8
9 DM_matrix = zeros(n_inputs,channels_input); % Matrix containing
    dedispersion vectors
10
11 % We take advantage of Matlab's native definition of matrix single-indexes
12 for i=1:n_channels
13     DM_matrix(i) = DM_vector(i);
14 end
```

Apéndice B

Códigos Python

B.1. ROACH 1

Código B.1: Grafica espectros en tiempo real, de FRB y Dedispersado

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import corr
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

global line4

"""
Parallel Detector v2.1 - ROACH 1

Obtiene y grafica los datos de:
  - Espectro en tiempo real (FFT)
  - Espectro dedispersado ie. salida del dedispersor
  - Espectro FRB ie. espectro fijo cuando se detecta un FRB

sergio@mwl-calan

Registers List (from detector_parallel_v2.slx):

From PC:
  - acc_len
  - cnt_rst
  - th

From FPGA:
  > Current Spectrum
    - Spec1
    - Spec2
    - Spec3
```

```
    - Spec4
  > Dedispersed Spectrum
    - Dedisp1
    - Dedisp2
    - Dedisp3
    - Dedisp4
  > Detected FRB
    - FRB1
    - FRB2
    - FRB3
    - FRB4
  > Accumulated Spectrum
    - ACC

  - sync_cnt
  - acc_cnt
"""

t_d        = 5E-3              # Dedispersed pulse duration
f_fpga     = 120E6               # Clock freq of fpga
fft_chans  = 64                # Real fft channels
par_chans  = 4                 # Number of parallel streams of fft
number_acc = t_d*f_fpga*par_chans/fft_chans  # Number of Accumulations
factor = 62/64.0                       # Relative number of channels to the
   total spectrum of simulated FRB
t_dd     = t_d*fft_chans*factor        # Dispersed pulse duration
frec_awg = 1/(t_dd)
print('Tiempo pulso dedispersado(ms): ' + str(t_d*1000))
print('Numero de acumulaciones: ' + str(number_acc))
print('Tiempo sweep FRB (ms): ' + str(t_dd*1000))
print('Frecuencia AWG: ' + str(frec_awg))

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.11')
time.sleep(1)

# Run just once to program the fpga
fpga.progdev('detector_parallel_v2.5.bof')
time.sleep(1)
print('Roach programmed, you can comment this section now')

# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

# get_data: Return a data vector concatenating data from 4-rams registers
def get_data(reg_name):
   ram0_data = read_ram(reg_name+str(1),2**9*8)
   ram1_data = read_ram(reg_name+str(2),2**9*8)
   ram2_data = read_ram(reg_name+str(3),2**9*8)
   ram3_data = read_ram(reg_name+str(4),2**9*8)

   data_vector = []

```

```
    for i in range(len(ram0_data)):
        data_vector.append(ram0_data[i])
        data_vector.append(ram1_data[i])
        data_vector.append(ram2_data[i])
        data_vector.append(ram3_data[i])

    return data_vector

# init: init function for animation
def init():
  line4[0].set_data([],[])
  line4[1].set_data([],[])
  line4[2].set_data([],[])
  return line4

# Create figure
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax2 = fig.add_subplot(140,140,1)
ax3 = fig.add_subplot(140,140,2)

# Axis limit
ax1.set_xlim(-20,500)
ax1.set_ylim(60,120)
ax2.set_xlim(-20,500)
ax2.set_ylim(60,120)
ax3.set_xlim(-20,500)
ax3.set_ylim(60,120)

# Create arrays. Will contain data to be plotted
line1, = ax1.plot([],[],lw=2, marker = '.', linestyle='-') # Spec
line2, = ax2.plot([],[],lw=2, marker = '.', linestyle='-') # Dedisp
line3, = ax3.plot([],[],lw=2, marker = '.', linestyle='-') # FRB
line4 = [line1, line2, line3] # line4 for historical reasons haha

# Write registers
fpga.write_int('acc_len',number_acc)
fpga.write_int('cnt_rst',1)
fpga.write_int('cnt_rst',0)

# Create frequency array
freq = np.linspace(0,480,64,endpoint=False)

# Set detector limit
fpga.write_int('theta',2^32-1) # Max value 2^32-1
# ~ fpga.write_int('theta',100)

def animate(i):
  frb_detect = fpga.read_int('frb_detector')
  # Obtain data from registers
  spec = np.array(get_data('Spec'))
  ddisp = np.array(get_data('Dedisp'))
  frb = np.array(get_data('FRB'))
  # Take Only 64 Valid Values
  spec = spec[0:64]
  ddisp = ddisp[1:64]
```

```
  frb = frb[1:64]
  # Calculate data in dB
  spec_db = 10*np.log10(spec+1)
  ddisp_db = 10*np.log10(ddisp+1)
  frb_db = 10*np.log10(frb+1)
  # ~ print(max(spec_db[2:]))
  #Asign data to arrays
  line4[0].set_data(freq,spec_db)
  #line4[3].set_data(freq[1:],len(freq)*[100])
  line4[1].set_data(freq[1:],ddisp_db)
  line4[2].set_data(freq[1:],frb_db)
  return line4

anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
    =init,repeat=True)

fig.canvas.set_window_title('Parallel Detector v2.0')

fig.suptitle('Espectros para deteccion de FRB')

ax1.title.set_text('Espectro en tiempo real')
ax2.title.set_text('Espectro dedispersado')
ax3.title.set_text('Espectro FRB detectado')

ax1.set_ylabel('Potencia (dB)')
ax2.set_ylabel('Potencia (dB)')
ax3.set_ylabel('Potencia (dB)')

ax3.set_xlabel('Frecuencia (Mhz)')

plt.show()
```

Código B.2: Grafica la señal de acumulación en tiempo real

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  tutorial2_osc_v3.py
#
#  sergio@mwl-roach

import corr
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

global line4
global acc_lim

"""
Parallel Detector ACC v2.1 - ROACH 1

Entrega la se al del espectro acumulado de la memoria 'ACC' en tiempo
    real.


```

```
Registers List (from detector_parallel_v2.slx):

From PC:
  - acc_len
  - cnt_rst
  - th

From FPGA:
  > Current Spectrum
    - Spec1
    - Spec2
    - Spec3
    - Spec4
  > Dedispersed Spectrum
    - Dedisp1
    - Dedisp2
    - Dedisp3
    - Dedisp4
  > Detected FRB
    - FRB1
    - FRB2
    - FRB3
    - FRB4
  > Accumulated Spectrum
    - ACC

  - sync_cnt
  - acc_cnt
"""

t_d        = 5E-3              # Dedispersed pulse duration
f_fpga     = 120E6               # Clock freq of fpga
fft_chans  = 64                # Real fft channels
par_chans  = 4                 # Number of parallel streams of fft
number_acc = t_d*f_fpga*par_chans/fft_chans  # Number of Accumulations
factor = 62/64.0                 # Relative number of channels to the
    total spectrum of simulated FRB
t_dd     = t_d*fft_chans*factor        # Dispersed pulse duration
frec_awg = 1/(t_dd)              # Frequency of the arbitrary function
    generator (awg)

print('Tiempo pulso dedispersado(ms): ' + str(t_d*1000))
print('Numero de acumulaciones: ' + str(number_acc))
print('Tiempo sweep FRB (ms): ' + str(t_dd*1000))
print('Frecuencia AWG (Hz): ' + str(frec_awg))

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.11')
time.sleep(1)


# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data
```

```
77
78 # get_data: Return a data vector concatenating data from 4-rams registers
79 def get_data(reg_name):
80     ram0_data = read_ram(reg_name+str(1),2**9*8)
81     ram1_data = read_ram(reg_name+str(2),2**9*8)
82     ram2_data = read_ram(reg_name+str(3),2**9*8)
83     ram3_data = read_ram(reg_name+str(4),2**9*8)
84
85     data_vector = []
86
87     for i in range(len(ram0_data)):
88         data_vector.append(ram0_data[i])
89         data_vector.append(ram1_data[i])
90         data_vector.append(ram2_data[i])
91         data_vector.append(ram3_data[i])
92
93     return data_vector
94
95 # init: init function for animation
96 def init():
97   line4[0].set_data([],[])
98   line4[1].set_data([],[])
99   return line4
100
101 # Write registers
102 fpga.write_int('acc_len',number_acc)
103 fpga.write_int('cnt_rst',1)
104 fpga.write_int('cnt_rst',0)
105 time.sleep(1)
106
107 # Get some samples from ACC register to get avg and std
108 samp = 20      #Number of samples
109 acc_array = [1]*samp
110 for i in range(samp):
111   acc = read_ram('ACC',2**9*8)
112   acc_array[i] = acc
113
114 acc_avg = np.mean(acc_array)
115 acc_std = np.std(acc_array)
116 acc_lim = acc_avg + 5*acc_std
117
118 # ~ acc_lim = 10**(90.0/10) # Fix acc_lim to 90 dB
119
120 print('Average acc (dB): '+str(10*np.log10(acc_avg)))
121 print('std acc (dB): '+str(10*np.log10(acc_std)))
122 print('Acc lim (dB): '+str(10*np.log10(acc_lim)))
123 print('Acc lim : '+str(acc_lim))
124
125 # Create figure
126 fig = plt.figure()
127 ax1 = fig.add_subplot(1,1,1)
128
129 # Axis limit
130 ax1.set_xlim(-5,517)
131 ax1.set_ylim(10*np.log10(min(acc)*0.97),10*np.log10(acc_lim*1.08))
132 # ~ ax1.set_ylim(min(acc)*0.97,acc_lim*1.08)
```

```

# Create arrays. Will contain data to be plotted
line1, = ax1.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC
line2, = ax1.plot([],[],lw=2, marker = None, linestyle='-', label= "Umbral
    : "+str(round(10*np.log10(acc_lim),2))+" dB" ) # Threshold
line4 = [line1, line2] # 'line4' for historical reasons haha

# Create frequency array
freq = np.linspace(0,480,64,endpoint=False)
acc_x = np.linspace(0,511,512,endpoint=True)

# Set detection limit
# ~ fpga.write_int('theta',int(acc_lim))
fpga.write_int('theta',2^32-1)


time.sleep(2)

def animate(i):
  frb_detect = fpga.read_int('frb_detector')
  # Obtain data from registers
  acc = read_ram('ACC',2**9*8)
  # Data in dB
  acc_db = 10*np.log10(acc)
  # ~ if(max(acc)>acc_lim):
    # ~ print('Detected!')
  #Asign data to arrays
  line4[0].set_data(acc_x,acc_db)
  line4[1].set_data(acc_x,len(acc_x)*[10*np.log10(acc_lim)])
  return line4

anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
    =init,repeat=True)

fig.canvas.set_window_title('Parallel Detector v2.0')

fig.suptitle('Espectros para deteccion de FRB')

ax1.title.set_text('Potencia Acumulada')

ax1.set_ylabel('Potencia (dB)')
ax1.set_xlabel('Muestras')
plt.legend()
plt.show()
```

Código B.3: Utilidades para medición: valores RMS y capturas en formato csv

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  tutorial2_osc_v3.py
#
#  sergio@mwl-roach

import corr
import time
```

```
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from scipy.signal import find_peaks

"""
Parallel Detector RMS v2.1 - ROACH 1

Permite guardar muestras en csv del espectro en tiempo real y de acc
Obtiene el valor RMS de la se al con ayuda de la librer a scipy.signal

Registers List (from detector_parallel_v2.slx):

From PC:
  - acc_len
  - cnt_rst
  - th

From FPGA:
  > Current Spectrum
    - Spec1
    - Spec2
    - Spec3
    - Spec4
  > Dedispersed Spectrum
    - Dedisp1
    - Dedisp2
    - Dedisp3
    - Dedisp4
  > Detected FRB
    - FRB1
    - FRB2
    - FRB3
    - FRB4
  > Accumulated Spectrum
    - ACC

  - sync_cnt
  - acc_cnt
"""

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.11')
time.sleep(1)

# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

# get_data: Return a data vector concatenating data from 4-rams registers
def get_data(reg_name):
   ram0_data = read_ram(reg_name+str(1),2**9*8)
   ram1_data = read_ram(reg_name+str(2),2**9*8)
   ram2_data = read_ram(reg_name+str(3),2**9*8)
```

```
    ram3_data = read_ram(reg_name+str(4),2**9*8)

    data_vector = []

    for i in range(len(ram0_data)):
        data_vector.append(ram0_data[i])
        data_vector.append(ram1_data[i])
        data_vector.append(ram2_data[i])
        data_vector.append(ram3_data[i])

    return data_vector

# Read spectrum
spec = get_data('Spec')
spec = spec[:64]
freq = np.linspace(0,480,64,endpoint=False)
freq = freq[:]

# Read ACC ram
acc = read_ram('ACC',2**9*8)
samp = np.linspace(0,len(acc),len(acc),endpoint=False)

while True:

  print("Para guardar acc presione 1")
  print("Para guardar espectro presione 2")
  print("Para calcular SNR de acc presione 8")
  print("Para obtener una nueva muestra presione 9")
  num = input()

  if num == 9:

    # Read esp_disp
    spec = get_data('Spec')
    spec = spec[:64]
    freq = np.linspace(0,480,64,endpoint=False)
    freq = freq[:]

    # Read ACC ram
    acc = np.array(read_ram('ACC',2**9*8))
    samp = np.linspace(0,len(acc),len(acc),endpoint=False)

    print("Muestra finalizada")

  if num == 1:

    # Create array to be saved
    data = np.transpose([samp,acc])

    # Get current date and create file name
    date = datetime.now()
    file_name = "accspec_"+str(date.year)+str(date.month)+str(date.day)+"_
   "+str(date.hour)+str(date.minute)+str(date.second)+".csv"

    # Export data to csv
    np.savetxt(file_name, data, delimiter=",")
```

```
121
122     # Import same csv data
123     x, y = np.loadtxt(file_name, delimiter=',', usecols=(0, 1), unpack=
    True)
124
125     # Convert 'y' data to dB
126     y = 10*np.log10(y)
127
128     # Calculate RMS, AVG and STD
129     y_rms = np.sqrt(np.mean(y**2))
130     y_avg = np.mean(y)
131     y_std = np.std(y)
132
133     # Draw Figure
134     fig = plt.figure()
135     ax1 = fig.add_subplot(1,1,1)
136     ax1.plot(x, y, marker= '.')
137     ax1.plot(x, len(x)*[y_avg], label="Valor Promedio: "+str(round(y_avg
    ,2)), color = "blue")
138     ax1.plot(x, len(x)*[y_avg + 3*y_std], label="Limite deteccion: " + str
    (round(y_avg + 3*y_std,2)), color = "red")
139     ax1.set(xlabel='Slot Memoria', ylabel='Amplitud (dB)', title='Espectro
     Acumulado')
140     ax1.grid()
141     plt.legend()
142     plt.ion()
143     plt.show()
144
145   if num == 2:
146
147     # Create array to be saved
148     data = np.transpose([freq,spec])
149
150     # Get current date and create file name
151     date = datetime.now()
152     file_name = "spec_"+str(date.year)+str(date.month)+str(date.day)+"_"+
    str(date.hour)+str(date.minute)+str(date.second)+".csv"
153
154     # Export data to csv
155     np.savetxt(file_name, data, delimiter=",")
156
157     # Import same csv data
158     x, y = np.loadtxt(file_name, delimiter=',', usecols=(0, 1), unpack=
    True)
159
160     # Convert 'y' data to dB
161     y = 10*np.log10(y)
162
163     # Just select the values relevant to noise calculation
164     # ~ x1 = np.concatenate((x[:32],x[33:]))
165     # ~ y1 = np.concatenate((y[:32],y[33:]))
166
167     x1 = x
168     y1 = y
169
170     # Complement to those values
```

```
    x0 = x[30:35]
    y0 = y[30:35]

    # Calculate RMS, AVG and STD
    y_rms = np.sqrt(np.mean(y1**2))
    y_avg = np.mean(y1)
    y_std = np.std(y1)


    # Draw Figure
    fig = plt.figure()
    ax1 = fig.add_subplot(1,1,1)
    # ~ ax1.plot(x0,y0, marker= '.', color = "orange")
    ax1.plot(x1, y1, marker= '.')
    ax1.plot(x, len(x)*[y_avg], label="Valor Promedio: "+str(round(y_avg
    ,2)) , color = "blue")
    ax1.plot(x, len(x)*[y_avg + 3*y_std], label="Limite deteccion: " + str
    (round(y_avg + 3*y_std,2)), color = "red")
    ax1.set(xlabel='Frecuencia (MHz)', ylabel='Amplitud (dB)', title='
    Espectro en tiempo real')
    ax1.grid()
    plt.legend()
    plt.ion()
    plt.show()

  if num == 7:

    x = np.array(freq)
    y = np.array(spec)

    x = np.delete(x,0)
    y = np.delete(y,0)

    x = np.delete(x,31)
    y = np.delete(y,31)

    # Calculate RMS, AVG and STD of noise
    y_rms = np.sqrt(np.mean(y**2))
    y_avg = np.mean(y)
    y_std = np.std(y)

    print("El valor RMS del ruido es: "+str(y_rms))
    print("El ruido tiene una desviacion de: "+str(y_std))

    # Draw Figure
    fig = plt.figure()
    ax1 = fig.add_subplot(1,1,1)
    ax1.plot(x,y,marker='.')
    ax1.plot(x,len(x)*[y_rms], label="Valor RMS: "+str(round(y_rms,2))+'
    lineal')
    ax1.set(xlabel='Frecuencia (MHz)', ylabel='Amplitud', title='Espectro
    Original')
    ax1.grid()
    plt.legend()
    plt.show()

```

```
222
223  if num == 8:
224    # Create array to be saved
225    data = np.transpose([samp,acc])
226
227    # Get current date and create file name
228    date = datetime.now()
229    file_name = "accspec_"+str(date.year)+str(date.month)+str(date.day)+"_
     "+str(date.hour)+str(date.minute)+str(date.second)+".csv"
230
231    # Export data to csv
232    np.savetxt(file_name, data, delimiter=",")
233
234    # Load saved data
235    x, y = np.loadtxt(file_name, delimiter=',', usecols=(0, 1), unpack=
     True)
236
237    # Convert 'y' data to dB
238    # ~ y = 10*np.log10(64*y)
239
240    y_rms = np.sqrt(np.mean(y**2))
241    y_lim = y_rms*1.0035
242
243    peaks, _ = find_peaks(y, height = y_lim)
244
245    # Calculate AVG of peaks
246    y_peak_avg = np.mean(y[peaks])
247    num_peaks = len(peaks)
248
249    # Extend peaks indexes to +/-2
250    for i in peaks:
251      peaks = np.append(peaks,i+2)
252      peaks = np.append(peaks,i+1)
253      peaks = np.append(peaks,i-1)
254      peaks = np.append(peaks,i-2)
255
256    # Noise values
257    y_n = np.delete(y,peaks)
258
259    # Calculate RMS, AVG and STD of noise
260    y_n_rms = np.sqrt(np.mean(y_n**2))
261    y_n_avg = np.mean(y_n)
262    y_n_std = np.std(y_n)
263
264    print("Se han detectado "+str(num_peaks)+" peaks")
265    print("El valor promedio de los peaks es: "+str(y_peak_avg))
266    print("El valor RMS del ruido es: "+str(y_n_rms))
267    print("El ruido tiene una desviacion de: "+str(y_n_std))
268    print("La relacion senal a ruido (SNR) es: "+ str((y_peak_avg-y_n_rms)
     /y_n_std))
269
270    print('y_lim se encuentra un '+str(round((y_lim/y_n_rms-1)*100,2)) +'%
     sobre el valor RMS del ruido')
271
272    # Draw Figure
273    fig = plt.figure()
```

```
    ax1 = fig.add_subplot(1,1,1)
    ax1.plot(y_n, marker= '.')
    ax1.plot(y)
    ax1.plot(x, len(x)*[y_n_rms], label="Valor RMS: "+str(round(y_n_rms,2)
    ), color = "blue")
    ax1.plot(x, len(x)*[y_lim], label="Limite Peaks" + str(round(y_lim,2))
    , color = "red")
    ax1.set(xlabel='Slot Memoria', ylabel='Amplitud', title='Espectro
    Acumulado')
    ax1.grid()
    plt.legend()
    plt.show()
```

Código B.4: Utilidad para máximos de la señal de acumulación

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  tutorial2_osc_v3.py
#
#  sergio@mwl-roach

import corr
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

global line4
global acc_lim

"""
TrackerMAX_ACC - ROACH 1

1 Toma los datos en tiempo real de spec y de acc, mostrando los valores
    m ximos en el tiempo
2 Toma una muestra de x valores de spec y devuelve el valor promedio

"""

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.11')
time.sleep(1)

# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

# get_data: Return a data vector concatenating data from 4-rams registers
def get_data(reg_name):
   ram0_data = read_ram(reg_name+str(1),2**9*8)
   ram1_data = read_ram(reg_name+str(2),2**9*8)
   ram2_data = read_ram(reg_name+str(3),2**9*8)
   ram3_data = read_ram(reg_name+str(4),2**9*8)
   data_vector = []
```

```
    for i in range(len(ram0_data)):
        data_vector.append(ram0_data[i])
        data_vector.append(ram1_data[i])
        data_vector.append(ram2_data[i])
        data_vector.append(ram3_data[i])
    return data_vector

acc = read_ram('ACC',2**9*8)

spec = get_data('Spec')
spec = spec[:64]

i = 0
max_acc = 0
max_spec = 0
mean_spec = []
sel = 0

print('Para capturar ACC marque 1')
print('Para capturar Spec marque 2')
print('Para capturar el promedio de MAX(Spec) marque 3')

sel = input("Seleccion: ")

if sel == 1:
  while True:
    acc = read_ram('ACC',2**9*8)
    acc_max = max(acc)
    if(acc_max > max_acc):
      max_acc = acc_max
      print('Nuevo maximo '+str(i)+': '+str(max_acc)+' (dB): '+str(10*np.log10(max_acc)))
      i+=1
    time.sleep(0.001)

if sel == 2:
  while True:
    spec = get_data('Spec')
    spec = spec[:64]
    spec_max = max(spec)
    if(spec_max > max_spec):
      max_spec = spec_max
      print('Nuevo maximo '+str(i)+': '+str(max_spec)+'(lineal) '+str(10*np.log10(max_spec))+'(dB)')
    time.sleep(0.001)

if sel == 3:
  while True:
    spec = get_data('Spec')
    spec = spec[:64]
    mean_spec.append(max(spec))
    if len(mean_spec)>200:
      break
    print('Promedio Peaks: '+str(round(np.mean(mean_spec),0))+'(lineal) '+str(round(10*np.log10(np.mean(mean_spec)),2))+'(dB)')
    time.sleep(0.001)
```

B.2. ROACH 2

Código B.5: Grafica espectros en tiempo real

```
1
2  import corr
3  import numpy as np
4  import time
5  import matplotlib.pyplot as plt
6  import matplotlib.animation as animation
7
8  global line
9  """
10 Detector v1 - ROACH 2
11
12 Obtiene y grafica los datos de:
13   - Espectro actual (FFT)
14   - Espectro dedispersado ie. salida del dedispersor
15   - Espectro FRB ie. espectro fijo cuando se detecta un FRB
16
17 sergio@mwl-calan
18 """
19 # Create fpga object (sleep for stability)
20 fpga = corr.katcp_wrapper.FpgaClient('192.168.1.14')
21 time.sleep(1)
22
23 # Upload bof file to fpga
24 fpga.upload_program_bof('detector_roach2_v1.2.bof',60000)
25 time.sleep(1)
26
27 # read_ram: Reads a single ram block of user-defined bytesize
28 def read_ram(ram_name, bytesize):
29   raw_data = fpga.read(ram_name,bytesize)
30   ram_data = np.fromstring(raw_data,dtype='>u8')
31   return ram_data
32
33 # get_data: Return a data vector concatenating data from 8-rams registers
34 def get_data(reg_name):
35    ram0_data = read_ram(reg_name+str(1),2**9*8)
36    ram1_data = read_ram(reg_name+str(2),2**9*8)
37    ram2_data = read_ram(reg_name+str(3),2**9*8)
38    ram3_data = read_ram(reg_name+str(4),2**9*8)
39    ram4_data = read_ram(reg_name+str(5),2**9*8)
40    ram5_data = read_ram(reg_name+str(6),2**9*8)
41    ram6_data = read_ram(reg_name+str(7),2**9*8)
42    ram7_data = read_ram(reg_name+str(8),2**9*8)
43
44    data_vector = []
45
46    for i in range(len(ram0_data)):
47       data_vector.append(ram0_data[i])
48       data_vector.append(ram1_data[i])
49       data_vector.append(ram2_data[i])
50       data_vector.append(ram3_data[i])
51       data_vector.append(ram4_data[i])
52       data_vector.append(ram5_data[i])
```

```
53        data_vector.append(ram6_data[i])
54        data_vector.append(ram7_data[i])
55
56    return data_vector
57
58 # init: init function for animation
59 def init():
60   line[0].set_data([],[])
61   # ~ line[1].set_data([],[])
62   # ~ line[2].set_data([],[])
63   return line
64
65 # Create figure
66 fig = plt.figure()
67 ax1 = fig.add_subplot(1,1,1)
68
69 # Axis limit
70 ax1.set_xlim(-20,560)
71 ax1.set_ylim(60,150)
72
73 # Create arrays. Will contain data to be plotted
74 line1, = ax1.plot([],[],lw=2, marker = '.', linestyle='-') # Spec
75 # ~ line2, = ax2.plot([],[],lw=2, marker = '.', linestyle='-') # Dedisp
76 # ~ line3, = ax3.plot([],[],lw=2, marker = '.', linestyle='-') # FRB
77 # ~ lines = [line1, line2, line3]
78 line = [line1]
79
80 # Write registers
81 fpga.write_int('acc_len',20000)
82 fpga.write_int('cnt_rst',1)
83 fpga.write_int('cnt_rst',0)
84
85 # Create frequency array
86 freq = np.linspace(0,540,64,endpoint=False)
87
88 # Set detector limit
89 fpga.write_int('theta',2^32-1) # Max value 2^32-1
90
91 def animate(i):
92   # Obtain data from registers
93   spec = np.array(get_data('Spec'))
94   # ~ ddisp = np.array(get_data('Dedisp'))
95   # ~ frb = np.array(get_data('FRB'))
96   # Take Only 128 Valid Values
97   spec = spec[0:64]
98   # ~ ddisp = ddisp[1:64]
99   # ~ frb = frb[1:64]
100  # Calculate data in dB
101  spec_db = 10*np.log10(spec+1)
102  # ~ ddisp_db = 10*np.log10(ddisp+1)
103  # ~ frb_db = 10*np.log10(frb+1)
104  # ~ print(max(spec_db[2:]))
105  #Asign data to arrays
106  line[0].set_data(freq,spec_db)
107  #line4[3].set_data(freq[1:],len(freq)*[100])
108  # ~ line4[1].set_data(freq[1:],ddisp_db)
```

```
  # ~ line4[2].set_data(freq[1:],frb_db)
  return line

anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
    =init,repeat=True)

fig.canvas.set_window_title('Parallel Detector v2.0')

fig.suptitle('Espectros para deteccion de FRB')

ax1.title.set_text('Espectro en tiempo real')

ax1.set_ylabel('Potencia (dB)')

ax1.set_xlabel('Frecuencia (Mhz)')

plt.show()
```

Código B.6: Grafica señal de acumulación

```

import corr
import numpy as np
import time
import matplotlib.pyplot as plt
import matplotlib.animation as animation

"""
Detector ACC v1 - ROACH 2

Obtiene y grafica en tiempo real la se al de acumulaci n
"""

global line
global acc_lim

t_d        = 7.5E-3              # Dedispersed pulse duration
f_fpga     = 67.5E6*2            # Clock freq of fpga
fft_chans  = 64                 # Real fft channels
par_chans  = 8                  # Number of parallel streams of fft
number_acc = t_d*f_fpga*par_chans/fft_chans  # Number of Accumulations
factor = 62/64.0                    # Relative number of channels to the
    total spectrum of simulated FRB
t_dd     = t_d*fft_chans*factor        # Dispersed pulse duration
frec_awg = 1/(t_dd)
print('Tiempo pulso dedispersado(ms): ' + str(t_d*1000))
print('Numero de acumulaciones: ' + str(number_acc))
print('Tiempo sweep FRB (ms): ' + str(t_dd*1000))
print('Frecuencia AWG: ' + str(frec_awg))

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.14')
time.sleep(1)

# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
```

```
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

# init: init function for animation
def init():
  line[0].set_data([],[])
  line[1].set_data([],[])
  return line

# Write registers
fpga.write_int('acc_len',201076)
fpga.write_int('cnt_rst',1)
fpga.write_int('cnt_rst',0)
time.sleep(1)

# Get some samples from ACC register to get avg and std
samp = 20     #Number of samples
acc_array = [1]*samp
for i in range(samp):
  acc = read_ram('ACC',2**9*8)
  acc_array[i] = acc

acc_avg = np.mean(acc_array)
acc_std = np.std(acc_array)
acc_lim = acc_avg + 5*acc_std

# ~ acc_lim = 10**(90.0/10) # Fix acc_lim to 90 dB

print('Average acc (dB): '+str(10*np.log10(acc_avg)))
print('std acc (dB): '+str(10*np.log10(acc_std)))
print('Acc lim (dB): '+str(10*np.log10(acc_lim)))
print('Acc lim : '+str(acc_lim))

# Create figure
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)

# Axis limit
ax1.set_xlim(-5,517)
ax1.set_ylim(10*np.log10(min(acc)*0.97),10*np.log10(acc_lim*1.08))

# Create arrays. Will contain data to be plotted
line1, = ax1.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC
line2, = ax1.plot([],[],lw=2, marker = None, linestyle='-', label= "Umbral
    : "+str(round(10*np.log10(acc_lim),2))+" dB" ) # Threshold
line = [line1, line2]

# Create x-axis array
acc_x = np.linspace(0,511,512,endpoint=True)

# Set detection limit
# ~ fpga.write_int('theta',int(acc_lim))
fpga.write_int('theta',2^32-1)

time.sleep(2)

```

```
92 def animate(i):
93   # Obtain data from registers
94   acc = read_ram('ACC',2**9*8)
95   # Data in dB
96   acc_db = 10*np.log10(acc)
97   # print(max(acc_db))
98   # ~ if(max(acc)>acc_lim):
99     # ~ print('Detected!')
100  #Asign data to arrays
101  line[0].set_data(acc_x,acc_db)
102  line[1].set_data(acc_x,len(acc_x)*[10*np.log10(acc_lim)])
103  return line
104
105 anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
       =init,repeat=True)
106
107 fig.canvas.set_window_title('Parallel Detector v2.0')
108
109 fig.suptitle('Espectros para deteccion de FRB')
110
111 ax1.title.set_text('Potencia Acumulada')
112
113 ax1.set_ylabel('Potencia (dB)')
114 ax1.set_xlabel('Muestras')
115 plt.legend()
116 plt.show()
```

Código B.7: Grafica señal de acumulación de 10 detectores en paralelo

```
1  # -*- coding: utf-8 -*-
2  # v2.2 para 10 detectores en paralelo
3
4  import corr
5  import numpy as np
6  import time
7  import matplotlib.pyplot as plt
8  import matplotlib.animation as animation
9
10 """
11 Detector ACC v2.2 - ROACH 2
12
13 Obtiene la se al de acumulaci n de 10 detectores en paralelo y las
       grafica en tiempo real
14 Pueden seleccionarse los DMs de cada se al y mostrar  los tiempos y
       retardos respectivos,
15 as   como la frecuencia que debe colocar en el AWG
16
17 """
18
19 global line
20
21 # Par metros FRB
22 k       = 4.15E6            # K_DM (MHz^2pc^-1cm^3ms)
23 BW      = 540               # Ancho de Banda (MHz)
24 f_cen   = 1100              # Frecuencia central (MHz)
25 f_low   = f_cen-BW/2        # L mite inferior de detecci n (MHz)
26 f_high  = f_cen+BW/2        # L mite superior de detecci n (MHz)
```

```
27
28 # Par metros del modelo
29 f_fpga      = 67.5E6*2          # Frecuencia de reloj
30 fft_chans   = 64                # Canales de la FFT
31 par_chans   = 8                 # N mero de streams en paralelo de la FFT
32 factor      = 62/64.0          # Relative number of channels to the total
       spectrum of simulated FRB
33
34 parameters  = [f_low,f_high,k,fft_chans,factor,f_fpga,par_chans]
35
36 # FRB_param: Funci n para obtener los par metros de tiempo y
       acumulaciones de un FRB dado un DM
37 def FRB_param(number,DM,param):
38     t_dd    = DM*((param[0]**-2-param[1]**-2)*param[2])*1E-3     #
       Duraci n del pulso dispersado (factor 1E3 pues k est  en ms)
39     f_awg   = 1/(t_dd)                                          #
       Frecuencia del AWG
40     t_d     = t_dd/(param[3]*param[4])                          #
       Duraci n del pulso dedispersado
41     n_acc   = t_d*param[5]*param[6]*param[4]/param[3]            # N mero
       de acumulaciones
42     print('*** FRB'+str(number)+' ***')
43     print('DM: '+str(DM)+' (pc*cm^-3)')
44     print('Tiempo de barrido: '+str(int(t_dd*1E3))+' (ms)')
45     print('N mero de acumulaciones: '+str(round(n_acc,2)))
46     print('Frecuencia AWG: '+str(round(f_awg,7))+' (Hz)')
47     return t_dd,f_awg,t_d,n_acc
48
49 # FRB 1
50 DM1 = 200
51 t_dd1,f_awg1,t_d1,number_acc1 = FRB_param(1,DM1,parameters)
52 # FRB 2
53 DM2 = 250
54 t_dd2,f_awg2,t_d2,number_acc2 = FRB_param(2,DM2,parameters)
55 # FRB 3
56 DM3 = 300
57 t_dd3,f_awg3,t_d3,number_acc3 = FRB_param(3,DM3,parameters)
58 # FRB 4
59 DM4 = 350
60 t_dd4,f_awg4,t_d4,number_acc4 = FRB_param(4,DM4,parameters)
61 # FRB 5
62 DM5 = 400
63 t_dd5,f_awg5,t_d5,number_acc5 = FRB_param(5,DM5,parameters)
64 # FRB 6
65 DM6 = 450
66 t_dd6,f_awg6,t_d6,number_acc6 = FRB_param(6,DM6,parameters)
67 # FRB 7
68 DM7 = 500
69 t_dd7,f_awg7,t_d7,number_acc7 = FRB_param(7,DM7,parameters)
70 # FRB 8
71 DM8 = 550
72 t_dd8,f_awg8,t_d8,number_acc8 = FRB_param(8,DM8,parameters)
73 # FRB 9
74 DM9 = 600
75 t_dd9,f_awg9,t_d9,number_acc9 = FRB_param(9,DM9,parameters)
76 # FRB 10
```

```
DM10 = 650
t_dd10,f_awg10,t_d10,number_acc10 = FRB_param(10,DM10,parameters)

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.14')

# Upload bof file to fpga
# fpga.upload_program_bof('detector_roach2_v2.2.bof',60000)
# print('fpga programada, puede comentar esta secci n')
time.sleep(2)

# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

# acc_lim: Get some samples from ACC# register to get detection limit
def acc_lim(acc_name):
    samp = 30   # Number of samples
    acc_array = [1]*samp
    for i in range(samp):
        acc = read_ram(acc_name,2**9*8)
        acc_array[i] = acc
    acc_avg = np.mean(acc_array)
    acc_std = np.std(acc_array)
    acc_limit = acc_avg + 8*acc_std

    print('FRB '+acc_name[3:]+':')
    print('Avg ACC (dB): '+str(10*np.log10(acc_avg)))
    print('Std ACC (dB): '+str(10*np.log10(acc_std)))
    print('ACC Lim (dB): '+str(10*np.log10(acc_limit)))
    return acc_limit,acc_std

# init: init function for animation
def init():
    line[0].set_data([],[])
    line[1].set_data([],[])
    line[2].set_data([],[])
    line[3].set_data([],[])
    line[4].set_data([],[])
    line[5].set_data([],[])
    line[6].set_data([],[])
    line[7].set_data([],[])
    line[8].set_data([],[])
    line[9].set_data([],[])
    line[10].set_data([],[])
    line[11].set_data([],[])
    line[12].set_data([],[])
    line[13].set_data([],[])
    line[14].set_data([],[])
    line[15].set_data([],[])
    line[16].set_data([],[])
    line[17].set_data([],[])
    line[18].set_data([],[])
    line[19].set_data([],[])
```

```
    return line

# Write registers
fpga.write_int('acc_len1',number_acc1)
fpga.write_int('acc_len2',number_acc2)
fpga.write_int('acc_len3',number_acc3)
fpga.write_int('acc_len4',number_acc4)
fpga.write_int('acc_len5',number_acc5)
fpga.write_int('acc_len6',number_acc6)
fpga.write_int('acc_len7',number_acc7)
fpga.write_int('acc_len8',number_acc8)
fpga.write_int('acc_len9',number_acc9)
fpga.write_int('acc_len10',number_acc10)
# fpga.write_int('cnt_rst',1)
# fpga.write_int('cnt_rst',0)
time.sleep(2)

# Create figure
fig = plt.figure()
ax1 = fig.add_subplot(5,2,1)
ax2 = fig.add_subplot(5,2,2)
ax3 = fig.add_subplot(5,2,3)
ax4 = fig.add_subplot(5,2,4)
ax5 = fig.add_subplot(5,2,5)
ax6 = fig.add_subplot(5,2,6)
ax7 = fig.add_subplot(5,2,7)
ax8 = fig.add_subplot(5,2,8)
ax9 = fig.add_subplot(5,2,9)
ax10 = fig.add_subplot(5,2,10)

# Calculate detection limits for DMs
acc_lim1,std1 = acc_lim('ACC1')
acc_lim2,std2 = acc_lim('ACC2')
acc_lim3,std3 = acc_lim('ACC3')
acc_lim4,std4 = acc_lim('ACC4')
acc_lim5,std5 = acc_lim('ACC5')
acc_lim6,std6 = acc_lim('ACC6')
acc_lim7,std7 = acc_lim('ACC7')
acc_lim8,std8 = acc_lim('ACC8')
acc_lim9,std9 = acc_lim('ACC9')
acc_lim10,std10 = acc_lim('ACC10')
# Detection limits to dB
acc_lim1_db = 10*np.log10(acc_lim1)
acc_lim2_db = 10*np.log10(acc_lim2)
acc_lim3_db = 10*np.log10(acc_lim3)
acc_lim4_db = 10*np.log10(acc_lim4)
acc_lim5_db = 10*np.log10(acc_lim5)
acc_lim6_db = 10*np.log10(acc_lim6)
acc_lim7_db = 10*np.log10(acc_lim7)
acc_lim8_db = 10*np.log10(acc_lim8)
acc_lim9_db = 10*np.log10(acc_lim9)
acc_lim10_db = 10*np.log10(acc_lim10)

# x-Axis limit
ax1.set_xlim(-5,517)
ax2.set_xlim(-5,517)
```

```
ax3.set_xlim(-5,517)
ax4.set_xlim(-5,517)
ax5.set_xlim(-5,517)
ax6.set_xlim(-5,517)
ax7.set_xlim(-5,517)
ax8.set_xlim(-5,517)
ax9.set_xlim(-5,517)
ax10.set_xlim(-5,517)

# Set how many deviations from acc_lim will plot data
low = 15
high = 5

# y-Axis limit
ax1.set_ylim(10*np.log10(acc_lim1-low*std1),10*np.log10(acc_lim1+high*std1
    ))
ax2.set_ylim(10*np.log10(acc_lim2-low*std2),10*np.log10(acc_lim2+high*std2
    ))
ax3.set_ylim(10*np.log10(acc_lim3-low*std3),10*np.log10(acc_lim3+high*std3
    ))
ax4.set_ylim(10*np.log10(acc_lim4-low*std4),10*np.log10(acc_lim4+high*std4
    ))
ax5.set_ylim(10*np.log10(acc_lim5-low*std5),10*np.log10(acc_lim5+high*std5
    ))
ax6.set_ylim(10*np.log10(acc_lim6-low*std6),10*np.log10(acc_lim6+high*std6
    ))
ax7.set_ylim(10*np.log10(acc_lim7-low*std7),10*np.log10(acc_lim7+high*std7
    ))
ax8.set_ylim(10*np.log10(acc_lim8-low*std8),10*np.log10(acc_lim8+high*std8
    ))
ax9.set_ylim(10*np.log10(acc_lim9-low*std9),10*np.log10(acc_lim9+high*std9
    ))
ax10.set_ylim(10*np.log10(acc_lim10-low*std10),10*np.log10(acc_lim10+high*
    std10))

# Create arrays. Will contain data to be plotted
line1, = ax1.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC1
line2, = ax2.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC2
line3, = ax3.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC3
line4, = ax4.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC4
line5, = ax5.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC5
line6, = ax6.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC6
line7, = ax7.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC7
line8, = ax8.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC8
line9, = ax9.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC9
line10,= ax10.plot([],[],lw=2, marker = '.', linestyle='-')  # ACC10
line11, = ax1.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim1),2))+" dB" )
line12, = ax2.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim2),2))+" dB" )
line13, = ax3.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim3),2))+" dB" )
line14, = ax4.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim4),2))+" dB" )
line15, = ax5.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim5),2))+" dB" )
```

```
line16, = ax6.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim6),2))+" dB" )
line17, = ax7.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim7),2))+" dB" )
line18, = ax8.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim8),2))+" dB" )
line19, = ax9.plot([],[],lw=2, marker = None, linestyle='-',label= "Umbral
    : "+str(round(10*np.log10(acc_lim9),2))+" dB" )
line20, = ax10.plot([],[],lw=2, marker = None, linestyle='-',label= "
    Umbral: "+str(round(10*np.log10(acc_lim10),2))+" dB" )

line = [line1,line2,line3,line4,line5,line6,line7,line8,line9,line10,
line11,line12,line13,line14,line15,line16,line17,line18,line19,line20]

# Create x-axis array
acc_x = np.linspace(0,511,512,endpoint=True)

# Set detection limit
fpga.write_int('theta1',2^32-1)
fpga.write_int('theta2',2^32-1)
fpga.write_int('theta3',2^32-1)
fpga.write_int('theta4',2^32-1)
fpga.write_int('theta5',2^32-1)
fpga.write_int('theta6',2^32-1)
fpga.write_int('theta7',2^32-1)
fpga.write_int('theta8',2^32-1)
fpga.write_int('theta9',2^32-1)
fpga.write_int('theta10',2^32-1)
time.sleep(2)

# Animation function
def animate(i):
    # Obtain data from registers
    acc1 = np.array(read_ram('ACC1',2**9*8))
    acc2 = np.array(read_ram('ACC2',2**9*8))
    acc3 = np.array(read_ram('ACC3',2**9*8))
    acc4 = np.array(read_ram('ACC4',2**9*8))
    acc5 = np.array(read_ram('ACC5',2**9*8))
    acc6 = np.array(read_ram('ACC6',2**9*8))
    acc7 = np.array(read_ram('ACC7',2**9*8))
    acc8 = np.array(read_ram('ACC8',2**9*8))
    acc9 = np.array(read_ram('ACC9',2**9*8))
    acc10 = np.array(read_ram('ACC10',2**9*8))
    # Data in dB
    acc_db1 = 10*np.log10(acc1)
    acc_db2 = 10*np.log10(acc2)
    acc_db3 = 10*np.log10(acc3)
    acc_db4 = 10*np.log10(acc4)
    acc_db5 = 10*np.log10(acc5)
    acc_db6 = 10*np.log10(acc6)
    acc_db7 = 10*np.log10(acc7)
    acc_db8 = 10*np.log10(acc8)
    acc_db9 = 10*np.log10(acc9)
    acc_db10 = 10*np.log10(acc10)
    #Asign data to arrays
    line[0].set_data(acc_x,acc_db1)
```

```
    line[1].set_data(acc_x,acc_db2)
    line[2].set_data(acc_x,acc_db3)
    line[3].set_data(acc_x,acc_db4)
    line[4].set_data(acc_x,acc_db5)
    line[5].set_data(acc_x,acc_db6)
    line[6].set_data(acc_x,acc_db7)
    line[7].set_data(acc_x,acc_db8)
    line[8].set_data(acc_x,acc_db9)
    line[9].set_data(acc_x,acc_db10)
    line[10].set_data(acc_x,len(acc_x)*[acc_lim1_db])
    line[11].set_data(acc_x,len(acc_x)*[acc_lim2_db])
    line[12].set_data(acc_x,len(acc_x)*[acc_lim3_db])
    line[13].set_data(acc_x,len(acc_x)*[acc_lim4_db])
    line[14].set_data(acc_x,len(acc_x)*[acc_lim5_db])
    line[15].set_data(acc_x,len(acc_x)*[acc_lim6_db])
    line[16].set_data(acc_x,len(acc_x)*[acc_lim7_db])
    line[17].set_data(acc_x,len(acc_x)*[acc_lim8_db])
    line[18].set_data(acc_x,len(acc_x)*[acc_lim9_db])
    line[19].set_data(acc_x,len(acc_x)*[acc_lim10_db])
    return line

anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
   =init,repeat=True)

fig.canvas.set_window_title('Parallel Detector v2.0')

fig.suptitle('Espectros para deteccion de FRB')

ax1.title.set_text('DM '+str(DM1))
ax1.set_ylabel('Potencia (dB)')
ax1.grid()
ax1.legend()

ax2.title.set_text('DM '+str(DM2))
ax2.set_ylabel('Potencia (dB)')
ax2.grid()
ax2.legend()

ax3.title.set_text('DM '+str(DM3))
ax3.set_ylabel('Potencia (dB)')
ax3.grid()
ax3.legend()

ax4.title.set_text('DM '+str(DM4))
ax4.set_ylabel('Potencia (dB)')
ax4.grid()
ax4.legend()

ax5.title.set_text('DM '+str(DM5))
ax5.set_ylabel('Potencia (dB)')
ax5.grid()
ax5.legend()

ax6.title.set_text('DM '+str(DM6))
ax6.set_ylabel('Potencia (dB)')
ax6.grid()
```

```
ax6.legend()

ax7.title.set_text('DM '+str(DM7))
ax7.set_ylabel('Potencia (dB)')
ax7.grid()
ax7.legend()

ax8.title.set_text('DM '+str(DM8))
ax8.set_ylabel('Potencia (dB)')
ax8.grid()
ax8.legend()

ax9.title.set_text('DM '+str(DM9))
ax9.set_ylabel('Potencia (dB)')
ax9.grid()
ax9.legend()

ax10.title.set_text('DM '+str(DM10))
ax10.set_ylabel('Potencia (dB)')
ax10.grid()
ax10.legend()

plt.show()
```

Código B.8: Utilidad para el SNR de la señal de acumulación

```
# -*- coding: utf-8 -*-
import corr
import numpy as np
import time
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

"""
Detector SNR ACC - ROACH 2

Obtiene el SNR para el espectro dedispersado (flux) ACC
"""

# Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.14')
time.sleep(1)

# Upload bof file to fpga
# fpga.upload_program_bof('detector_roach2_v1.2.bof',60000)
# time.sleep(1)

# read_ram: Reads a single ram block of user-defined bytesize
def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

# Write registers
fpga.write_int('acc_len',201076) # Frecuencia AWG: 1.3113 (Hz) DM: 200 (pc
    *cm^-3)
fpga.write_int('cnt_rst',1)
```

```
fpga.write_int('cnt_rst',0)

# Set detector limit
fpga.write_int('theta',2^32-1) # Max value 2^32-1

# In this case we don't need to define a baseline, because we sum over all
    spectra

# Read ACC RAM
acc = 1.0*np.array(read_ram('ACC',2**9*8))
samp = np.linspace(0,len(acc),len(acc),endpoint=False)

while True:
    print('Para obtener un nuevo ACC presione 1')
    print('Para calcular SNR de ACC presione 5')
    num = input()

    if num == 1:

        # Read ACC RAM
        acc = 1.0*np.array(read_ram('ACC',2**9*8))

        print('Muestra finalizada')

    if num == 5:

        # Calculate RMS value of signal to define threshold required by '
    find_peaks'
        acc_rms = np.sqrt(np.mean(acc**2))
        acc_avg = np.mean(acc)
        y_lim = acc_rms*1.0012

        peaks,_ = find_peaks(acc,height=y_lim)    # returns indexes of
    peaks

        # Calculate average of peaks
        pk_avg = np.mean(acc[peaks])
        pk_num = len(peaks)

        # Extend peaks indexes to +/-3
        for i in peaks:
            peaks = np.append(peaks,i+3)
            peaks = np.append(peaks,i+2)
            peaks = np.append(peaks,i+1)
            peaks = np.append(peaks,i-1)
            peaks = np.append(peaks,i-2)
            peaks = np.append(peaks,i-3)
        # Noise values
        acc_noise = np.delete(acc,peaks)

        # Noise Statistics
        acc_noise_rms = np.sqrt(np.mean(acc_noise**2))
        acc_noise_avg = np.mean(acc_noise)
        acc_noise_std = np.std(acc_noise)

        print('Se han detectado '+str(pk_num)+' peaks')
```

```
84          print('El valor promedio de los peaks es: '+str(10*np.log10(pk_avg
    )))
85          print('El valor RMS del ruido es: '+str(10*np.log10(acc_noise_rms)
    ))
86          print('La STD del ruido es: '+str(10*np.log10(acc_noise_std)))
87          print('SNR: '+str((pk_avg-acc_noise_rms)/acc_noise_std))
88
89          print('y_lim esta un '+str(round((y_lim/acc_noise_rms-1)*100,2)) +
    '% sobre el valor RMS del ruido')
90
91          # Draw Figure
92          fig = plt.figure()
93          ax1 = fig.add_subplot(2,1,1)
94          ax2 = fig.add_subplot(2,1,2)
95          ax2.plot(10*np.log10(acc_noise), marker= '.')
96          ax1.plot(samp,10*np.log10(acc),color='black')
97          ax1.plot(peaks,10*np.log10(acc[peaks]),marker='D',linestyle='None'
    ,color='green')
98          ax1.plot(samp, len(samp)*[10*np.log10(acc_rms)], label="Valor RMS:
     "+str(round(10*np.log10(acc_rms),2)), color = "blue")
99          ax1.plot(samp, len(samp)*[10*np.log10(y_lim)], label="Limite Peaks
    " + str(round(10*np.log10(y_lim),2)), color = "red")
100         ax1.set(xlabel='Slot Memoria', ylabel='Amplitud (dB)', title='
    Espectro Acumulado')
101         ax2.set(xlabel='Slot Memoria', ylabel='Amplitud (dB)', title='
    Ruido')
102         ax1.grid()
103         ax1.legend()
104         ax2.grid()
105         plt.show()
```

Código B.9: Utilidad para el SNR del espectro en tiempo real

```
1  # -*- coding: utf-8 -*-
2
3  # Detector SNR para el espectro en tiempo real (spec)
4  # El valor de la se al ha sido colocado en el canal 28, arbitrariamente
5  # Esto corresponde a 5.85(V) en el generador arbitrario de funciones
6  # Versi n sin animation y R/S automatizado
7
8  import corr
9  import numpy as np
10 import time
11 import matplotlib.pyplot as plt
12 import pyvisa
13
14 # Nivel del Generador R&S
15 level_RS = '-74'  # Valor en dBm
16
17 # Create fpga object (sleep for stability)
18 fpga = corr.katcp_wrapper.FpgaClient('192.168.1.14')
19 time.sleep(1)
20
21 # Create object for pyvisa and generator
22 rm = pyvisa.ResourceManager('@py')
23 inst = rm.open_resource('TCPIP::192.168.1.33::INSTR')
24 print(inst.query("*IDN?")) # Check connection
```

```
25
26 # Upload bof file to fpga
27 # fpga.upload_program_bof('detector_roach2_v1.2.bof',60000)
28 # time.sleep(1)
29
30 # read_ram: Reads a single ram block of user-defined bytesize
31 def read_ram(ram_name, bytesize):
32   raw_data = fpga.read(ram_name,bytesize)
33   ram_data = np.fromstring(raw_data,dtype='>u8')
34   return ram_data
35
36 # get_data: Return a data vector concatenating data from 8-rams registers
37 def get_data(reg_name):
38    ram0_data = read_ram(reg_name+str(1),2**9*8)
39    ram1_data = read_ram(reg_name+str(2),2**9*8)
40    ram2_data = read_ram(reg_name+str(3),2**9*8)
41    ram3_data = read_ram(reg_name+str(4),2**9*8)
42    ram4_data = read_ram(reg_name+str(5),2**9*8)
43    ram5_data = read_ram(reg_name+str(6),2**9*8)
44    ram6_data = read_ram(reg_name+str(7),2**9*8)
45    ram7_data = read_ram(reg_name+str(8),2**9*8)
46
47    data_vector = []
48
49    for i in range(len(ram0_data)):
50       data_vector.append(ram0_data[i])
51       data_vector.append(ram1_data[i])
52       data_vector.append(ram2_data[i])
53       data_vector.append(ram3_data[i])
54       data_vector.append(ram4_data[i])
55       data_vector.append(ram5_data[i])
56       data_vector.append(ram6_data[i])
57       data_vector.append(ram7_data[i])
58
59    return data_vector
60
61 # Write registers
62 fpga.write_int('acc_len',603229)
63 fpga.write_int('cnt_rst',1)
64 fpga.write_int('cnt_rst',0)
65
66 # Set detector limit
67 fpga.write_int('theta',2^32-1) # Max value 2^32-1
68
69 # Turn off R&S
70 inst.write('outp off')
71 print('RS OFF')
72 time.sleep(1)
73
74 # Baseline definition
75 samp = 30       # Number of samples
76 n_ch = 64       # Number of channels
77 spec_base = np.array([0]*n_ch)
78
79 for i in range(samp):
80     spec = np.array(get_data('Spec'))
```

```
    spec = spec[0:64]
    for j in range(n_ch):
        spec_base[j] = spec_base[j]+(1.0/samp)*spec[j]

spec_base_db = 10*np.log10(spec_base+1)

# Create frequency array
freq = np.array(np.linspace(0,540,64,endpoint=False))

# Plot current baseline
# plt.plot(freq,spec_base_db)
# print('Linea base definida, cierre el grafico para continuar')
# plt.show()

# Turn on R&S and select level
inst.write('outp on')
print('RS ON')
inst.write('pow '+level_RS +' dbm')
print('RS Power set to: '+level_RS+' dbm')
time.sleep(1)

freq_signal = freq[28]
freq = np.delete(freq,28)

snr_array = []

for i in range(100):
    # Obtain data from registers
    spec = np.array(get_data('Spec'))
    # Take Only 64 Valid Values
    spec = spec[0:64]
    spec = spec-spec_base
    spec_signal = spec[28]
    spec_noise = np.delete(spec,28)
    spec_std = np.std(spec_noise)
    snr = spec_signal/spec_std
    snr_array.append(snr)

snr_std = np.std(snr_array)
snr_mean = np.mean(snr_array)

print(snr_mean)
print(snr_std)
print(10*np.log10(snr_mean))
print(10*np.log10(snr_std))

# plt.plot(snr_array)
# plt.show()
```

B.3. Tutoriales ROACH

Código B.10: Tutorial Osciloscopio: Muestra señal del ADC en tiempo real

```
"""
Created on Tue Oct 01 2019

@author: sergio@mwl-calan
"""

import corr
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fpga = corr.katcp_wrapper.FpgaClient('192.168.1.11')
time.sleep(1)

# run just once to program the fpga
# ~ fpga.progdev('snapshot_tutorial.bof')
# ~ time.sleep(1)
# ~ print('FPGA Programada, puede comentar esta seccion')

# vector length (max 8192)

v_len = 4000
x = np.linspace(0,v_len,v_len)

# create figure
fig = plt.figure()
ax = plt.axes(xlim=(0,v_len),ylim=(-100,100))
line, = ax.plot([],[],lw=2)

def init():
  line.set_data([],[])
  return line,

def animate(i):
  raw_data = fpga.snapshot_get('snapshot', man_trig=True, man_valid=True)[
    'data']
  snap_data = np.fromstring(raw_data, dtype='>i1')
  val = snap_data[0:v_len:1]
  line.set_data(x,val)
  return line,

anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
   =init,repeat=True)

plt.title('ADC Signal')
plt.ylabel('ADC Output')
plt.grid(True)
plt.show()
```

Código B.11: Tutorial Espectrómetro: Muestra señal del espectro en tiempo real

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-
#
#  tutorial2_osc_v3.py
#
#  sergio@mwl-roach

import corr
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

"""
Created on Tue Oct 1

Registers List (from tut_spec.slx):

From PC
  > acc_len
  > cnt_rst

From FPGA
  > Shared_BRAM
  > Shared_BRAM1
  > Shared_BRAM2
  > Shared_BRAM3
  > sync_cnt
  > acc_cnt

@author: sergiosaavedra@mwl-calan
"""

#Create fpga object (sleep for stability)
fpga = corr.katcp_wrapper.FpgaClient('192.168.1.11')
time.sleep(1)

# Run just once to program the fpga
fpga.progdev('tut_specv0.3.bof')
time.sleep(1)
print('Roach programmed, you can comment this section now')

def read_ram(ram_name, bytesize):
  raw_data = fpga.read(ram_name,bytesize)
  ram_data = np.fromstring(raw_data,dtype='>u8')
  return ram_data

def get_data():
   ram0_data = read_ram('Shared_BRAM',2**9*8)
   ram1_data = read_ram('Shared_BRAM1',2**9*8)
   ram2_data = read_ram('Shared_BRAM2',2**9*8)
   ram3_data = read_ram('Shared_BRAM3',2**9*8)

   data_vector = []

   for i in range(len(ram0_data)):
      data_vector.append(ram0_data[i])
```

```
58        data_vector.append(ram1_data[i])
59        data_vector.append(ram2_data[i])
60        data_vector.append(ram3_data[i])
61
62     return data_vector
63
64 def init():
65   line.set_data([],[])
66   return line,
67
68 # create figure
69 fig = plt.figure()
70 ax = plt.axes(xlim=(-20,500),ylim=(-10,10))
71 line, = ax.plot([],[],lw=2)
72
73
74 fpga.write_int('acc_len',1000)
75 fpga.write_int('cnt_rst',1)
76 fpga.write_int('cnt_rst',0)
77
78 freq = np.linspace(0,480,2048,endpoint=False)
79
80 def animate(i):
81   data = get_data()
82   data_db = 10*np.log10(data)
83   data_max = max(data_db)
84   print('data_max:')
85   print(data_max)
86   data_min = min(data_db)
87   y_min, y_max = ax.get_ylim()
88   print('y_max:')
89   print(y_max)
90   if data_max >= y_max:
91     ax.set_ylim(data_min*0.9,data_max*1.1)
92
93   line.set_data(freq,data_db)
94   return line,
95
96 anim = animation.FuncAnimation(fig,animate,blit=True,interval=10,init_func
     =init,repeat=True)
97
98 plt.title('Espectrometro AKUENTA')
99 plt.show()
```

Apéndice C

Versiones Detector

C.1. Roach 1

- **v2.2**: Detector con tolerancia shifted 32 bits
- **v2.3**: Detector con thetha registro de 32 bits casteado a 64 bits. Esto es valor máximo es $2^{32} - 1 = 4294967295$
- **v2.4**: Contador de FRBs - cuenta cuando se ha superado el umbral de detección (rising edge). Sin embargo, el sistema se corta cuando detecta un FRB.
- **v2.5**: Mitigación por hardware de componente DC y ruido del ADC (por problemas de calibración).
- **v2.5.1**: Sistema no se corta cuando detecta un FRB.

C.2. Roach 2

- **v1.0**: Funciones normales. Migración del modelo de ROACH 1 versión 2.2.
- **v1.1**: Se elimina el canal DC y el ruido del ADC
- **v1.2**: Se arregla la señal ACC que entregaba un diente de sierra. Se cambia el "shift" del límite de detección por un "cast".
- **v2.0**: Dos detectores en paralelo
- **v2.1**: Paralelismo de 7 detectores
- **v2.2**: Paralelismo de 10 detectores

Apéndice D

Uso de Recursos ROACH 2

```
Release 14.7 Map P.20131013 (lin64)
Xilinx Map Application Log File for Design 'system'

Design Information
------------------
Command Line   : map -timing -detail -ol high -xe n -mt 2 -register_duplication
-o system_map.ncd -w -pr b system.ngd system.pcf
Target Device  : xc6vsx475t
Target Package : ff1759
Target Speed   : -1
Mapper Version : virtex6 -- $Revision: 1.55 $
Mapped Date    : Wed Jan 22 16:38:04 2020

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
INFO:Security:56 - Part 'xc6vsx475t' is not a WebPack part.
WARNING:Security:42 - Your software subscription period has lapsed. Your current
version of Xilinx tools will continue to function, but you no longer qualify for
Xilinx software updates or new releases.
----------------------------------------------------------------------
Mapping design into LUTs...
WARNING:MapLib:701 - Signal aux_clk_p connected to top level port aux_clk_p has
   been removed.
WARNING:MapLib:701 - Signal aux_clk_n connected to top level port aux_clk_n has
   been removed.
Writing file system_map.ngm...
Running directed packing...
Running delay-based LUT packing...
Updating timing models...
INFO:Map:215 - The Interim Design Summary has been generated in the MAP Report
   (.mrp).
Running timing-driven placement...
Total REAL time at the beginning of Placer: 9 mins 27 secs
Total CPU  time at the beginning of Placer: 9 mins 24 secs

Phase 1.1  Initial Placement Analysis
Phase 1.1  Initial Placement Analysis (Checksum:59be6cda) REAL time: 10 mins 23 secs

Phase 2.7  Design Feasibility Check
Phase 2.7  Design Feasibility Check (Checksum:59be6cda) REAL time: 10 mins 37 secs

Phase 3.31  Local Placement Optimization
Phase 3.31  Local Placement Optimization (Checksum:9d37e438) REAL time: 10 mins 37 secs

Phase 4.37  Local Placement Optimization
Phase 4.37  Local Placement Optimization (Checksum:9d37e438) REAL time: 10 mins 37 secs

Phase 5.2  Initial Placement for Architecture Specific Features

......


There are 18 clock regions on the target FPGA device:
|-----------------------------------------|-----------------------------------------|
| CLOCKREGION_X0Y8:                       | CLOCKREGION_X1Y8:                       |
|   4 BUFRs available, 0 in use           |   2 BUFRs available, 0 in use           |
|   6 Regional Clock Spines, 0 in use     |   6 Regional Clock Spines, 0 in use     |
|   4 edge BUFIOs available, 0 in use     |   0 edge BUFIOs available, 0 in use     |
|   4 center BUFIOs available, 0 in use   |   4 center BUFIOs available, 0 in use   |
|                                         |                                         |
|-----------------------------------------|-----------------------------------------|
| CLOCKREGION_X0Y7:                       | CLOCKREGION_X1Y7:                       |
|   4 BUFRs available, 0 in use           |   2 BUFRs available, 0 in use           |
|   6 Regional Clock Spines, 0 in use     |   6 Regional Clock Spines, 0 in use     |
|   4 edge BUFIOs available, 0 in use     |   0 edge BUFIOs available, 0 in use     |
|   4 center BUFIOs available, 0 in use   |   4 center BUFIOs available, 0 in use   |
|                                         |                                         |
|-----------------------------------------|-----------------------------------------|
| CLOCKREGION_X0Y6:                       | CLOCKREGION_X1Y6:                       |
|   4 BUFRs available, 0 in use           |   2 BUFRs available, 0 in use           |
|   6 Regional Clock Spines, 1 in use     |   6 Regional Clock Spines, 0 in use     |
```

```
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|
| CLOCKREGION_X0Y5:                        | CLOCKREGION_X1Y5:                        |
|   4 BUFRs available, 1 in use            |   2 BUFRs available, 0 in use            |
|   6 Regional Clock Spines, 1 in use      |   6 Regional Clock Spines, 0 in use      |
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|
| CLOCKREGION_X0Y4:                        | CLOCKREGION_X1Y4:                        |
|   4 BUFRs available, 0 in use            |   2 BUFRs available, 0 in use            |
|   6 Regional Clock Spines, 1 in use      |   6 Regional Clock Spines, 0 in use      |
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|
| CLOCKREGION_X0Y3:                        | CLOCKREGION_X1Y3:                        |
|   4 BUFRs available, 0 in use            |   2 BUFRs available, 0 in use            |
|   6 Regional Clock Spines, 0 in use      |   6 Regional Clock Spines, 0 in use      |
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|
| CLOCKREGION_X0Y2:                        | CLOCKREGION_X1Y2:                        |
|   4 BUFRs available, 0 in use            |   2 BUFRs available, 0 in use            |
|   6 Regional Clock Spines, 0 in use      |   6 Regional Clock Spines, 0 in use      |
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|
| CLOCKREGION_X0Y1:                        | CLOCKREGION_X1Y1:                        |
|   4 BUFRs available, 0 in use            |   2 BUFRs available, 0 in use            |
|   6 Regional Clock Spines, 0 in use      |   6 Regional Clock Spines, 0 in use      |
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|
| CLOCKREGION_X0Y0:                        | CLOCKREGION_X1Y0:                        |
|   4 BUFRs available, 0 in use            |   2 BUFRs available, 0 in use            |
|   6 Regional Clock Spines, 0 in use      |   6 Regional Clock Spines, 0 in use      |
|   4 edge BUFIOs available, 0 in use      |   0 edge BUFIOs available, 0 in use      |
|   4 center BUFIOs available, 0 in use    |   4 center BUFIOs available, 0 in use    |
|                                          |                                          |
|------------------------------------------|------------------------------------------|


Clock-Region: <CLOCKREGION_X0Y5>
  key resource utilizations (used/available): edge-bufios - 0/4; center-bufios - 0/4; bufrs - 1/4;
regional-clock-spines - 1/6
|-------------------------------------------------------------------------------------------------------------------------------
|       |    clock    | BRAM |     |    |        |        |       |       |       |      |      |      |      |
|       |
|       |    region   | FIFO | DCM | GT | ILOGIC | OLOGIC |   FF  |  LUTM |  LUTL | MULT | EMAC | PPC |
PCIe | <- (Types of Resources in Clock Region)
|-------|-------------|------|-----|----|--------|--------|-------|-------|-------|------|------|-----|------|
|       | Upper Region| 168  |  0  |  0 |   80   |   80   | 34560 | 13440 | 21120 | 112  |   0  |  0  |
0  | <- Available resources in the upper region
|-------|-------------|------|-----|----|--------|--------|-------|-------|-------|------|------|-----|------|
|       |CurrentRegion| 168  |  0  |  0 |   80   |   80   | 30720 | 13440 | 17280 | 112  |   0  |  0  |
0  | <- Available resources in the current region
|-------|-------------|------|-----|----|--------|--------|-------|-------|-------|------|------|-----|------|
|       | Lower Region| 168  |  0  |  0 |   80   |   80   | 30720 | 13440 | 17280 | 112  |   0  |  0  |
0  | <- Available resources in the lower region
|-------|-------------|------|-----|----|--------|--------|-------|-------|-------|------|------|-----|------|
| clock |    region
|
-----------------------------------------------
|  type |  expansion
```

```
|                                                                                      | <IO/Regional
clock Net Name>
|-------|-------------|------|-----|----|--------|--------|-------|-------|-------|------|------|-----|------|
|  BUFR | Upper/Lower |   0  |  0  |  0 |    0   |    0   |     0 |     0 |     0 |   0  |   0  |  0  |
0  | "detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div"
|-------|-------------|------|-----|----|--------|--------|-------|-------|-------|------|------|-----|------|



######################################################################################
# REGIONAL CLOCKING RESOURCE DISTRIBUTION UCF REPORT:
#
# Number of Regional Clocking Regions in the device: 18  (6 clock spines in each)
# Number of Regional Clock Networks used in this design: 1 (each network can be
# composed of up to 3 clock spines and cover up to 3 regional clock regions)
#
######################################################################################

# Regional-Clock "detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div" driven
by "BUFR_X1Y11"
INST "detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/DIVBUF" LOC = "BUFR_X1Y11" ;
NET "detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div" TNM_NET =
"TN_detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div" ;
TIMEGRP "TN_detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div" AREA_GROUP =
"CLKAG_detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div" ;
AREA_GROUP "CLKAG_detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/adc_clk_div" RANGE =
CLOCKREGION_X0Y5, CLOCKREGION_X0Y6, CLOCKREGION_X0Y4;


Phase 5.2  Initial Placement for Architecture Specific Features (Checksum:4275c046) REAL time: 12 mins 21
secs

Phase 6.36  Local Placement Optimization
Phase 6.36  Local Placement Optimization (Checksum:4275c046) REAL time: 12 mins 21 secs

Phase 7.30  Global Clock Region Assignment
Phase 7.30  Global Clock Region Assignment (Checksum:4275c046) REAL time: 12 mins 21 secs

Phase 8.3  Local Placement Optimization
Phase 8.3  Local Placement Optimization (Checksum:4275c046) REAL time: 12 mins 23 secs

Phase 9.5  Local Placement Optimization
Phase 9.5  Local Placement Optimization (Checksum:4275c046) REAL time: 12 mins 26 secs

Phase 10.8  Global Placement
.......................................
.............................................................................................................
.............................................................................................................
.............................................................................................................
......................
................................
Phase 10.8  Global Placement (Checksum:756e2a13) REAL time: 33 mins 14 secs

Phase 11.5  Local Placement Optimization
Phase 11.5  Local Placement Optimization (Checksum:756e2a13) REAL time: 33 mins 28 secs

Phase 12.18  Placement Optimization
Phase 12.18  Placement Optimization (Checksum:2be74c5c) REAL time: 36 mins 57 secs

Phase 13.5  Local Placement Optimization
Phase 13.5  Local Placement Optimization (Checksum:2be74c5c) REAL time: 37 mins

Phase 14.34  Placement Validation
Phase 14.34  Placement Validation (Checksum:e9a869fa) REAL time: 37 mins 6 secs

Total REAL time to Placer completion: 37 mins 17 secs
Total CPU  time to Placer completion: 40 mins 24 secs
Running physical synthesis...
```

```
Physical synthesis completed.
Running post-placement packing...
Writing output files...
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM2_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM1_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM20_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM4_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM16_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM15_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM3_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM5_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM10_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM11_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM14_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM19_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM8_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM13_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM22_RAMB_D1_D
   PO> is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM22_RAMC_D1_D
   PO> is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM22_RAMD_D1_O
```

```
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM18_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM9_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM6_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM21_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM12_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM7_RAMD_D1_O>
   is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <detector_roach2_v2_2_asiaa_adc5g/detector_roach2_v2_2_asiaa_adc5g/FIFO/BU2/U0/grf.rf/mem/gdm.dm/
Mram_RAM17_RAMD_D1_O
   > is incomplete. The signal does not drive any load pins in the design.

Design Summary
--------------

Design Summary:
Number of errors:      0
Number of warnings:   26
Slice Logic Utilization:
  Number of Slice Registers:              177,092 out of 595,200   29%
    Number used as Flip Flops:            177,090
    Number used as Latches:                     2
    Number used as Latch-thrus:                 0
    Number used as AND/OR logics:               0
  Number of Slice LUTs:                   158,731 out of 297,600   53%
    Number used as logic:                  44,190 out of 297,600   14%
      Number using O6 output only:         32,361
      Number using O5 output only:          1,629
      Number using O5 and O6:              10,200
      Number used as ROM:                       0
    Number used as Memory:                102,296 out of 122,240   83%
      Number used as Dual Port RAM:           168
        Number using O6 output only:           16
        Number using O5 output only:            2
        Number using O5 and O6:               150
      Number used as Single Port RAM:           0
      Number used as Shift Register:      102,128
        Number using O6 output only:      100,698
        Number using O5 output only:            0
        Number using O5 and O6:             1,430
    Number used exclusively as route-thrus: 12,245
      Number with same-slice register load:  3,184
      Number with same-slice carry load:     9,061
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:               49,683 out of  74,400   66%
  Number of LUT Flip Flop pairs used:     180,474
    Number with an unused Flip Flop:       14,579 out of 180,474    8%
    Number with an unused LUT:             21,743 out of 180,474   12%
    Number of fully used LUT-FF pairs:    144,152 out of 180,474   79%
    Number of unique control sets:          1,039
```

```
    Number of slice register sites lost
      to control set restrictions:          1,712 out of 595,200    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
  one Flip Flop within a slice.  A control set is a unique combination of
  clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.
  OVERMAPPING of BRAM resources should be ignored if the design is
  over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:
  Number of bonded IOBs:                      144 out of     840   17%
    Number of LOCed IOBs:                     144 out of     144  100%
    IOB Flip Flops:                            96

Specific Feature Utilization:
  Number of RAMB36E1/FIFO36E1s:               104 out of   1,064    9%
    Number using RAMB36E1 only:               104
    Number using FIFO36E1 only:                 0
  Number of RAMB18E1/FIFO18E1s:               144 out of   2,128    6%
    Number using RAMB18E1 only:               144
    Number using FIFO18E1 only:                 0
  Number of BUFG/BUFGCTRLs:                     6 out of      32   18%
    Number used as BUFGs:                       6
    Number used as BUFGCTRLs:                   0
  Number of ILOGICE1/ISERDESE1s:               96 out of   1,080    8%
    Number used as ILOGICE1s:                  64
    Number used as ISERDESE1s:                 32
  Number of OLOGICE1/OSERDESE1s:               32 out of   1,080    2%
    Number used as OLOGICE1s:                  32
    Number used as OSERDESE1s:                  0
  Number of BSCANs:                             0 out of       4    0%
  Number of BUFHCEs:                            0 out of     216    0%
  Number of BUFIODQSs:                          0 out of     108    0%
  Number of BUFRs:                              1 out of      54    1%
    Number of LOCed BUFRs:                      1 out of       1  100%
  Number of CAPTUREs:                           0 out of       1    0%
  Number of DSP48E1s:                         224 out of   2,016   11%
  Number of EFUSE_USRs:                         0 out of       1    0%
  Number of FRAME_ECCs:                         0 out of       1    0%
  Number of GTXE1s:                             0 out of      36    0%
  Number of IBUFDS_GTXE1s:                      0 out of      18    0%
  Number of ICAPs:                              0 out of       2    0%
  Number of IDELAYCTRLs:                        2 out of      27    7%
  Number of IODELAYE1s:                        32 out of   1,080    2%
  Number of MMCM_ADVs:                          2 out of      18   11%
  Number of PCIE_2_0s:                          0 out of       2    0%
  Number of STARTUPs:                           1 out of       1  100%
  Number of SYSMONs:                            0 out of       1    0%
  Number of TEMAC_SINGLEs:                      0 out of       4    0%

Average Fanout of Non-Clock Nets:               2.29

Peak Memory Usage:  6035 MB
Total REAL time to MAP completion:  48 mins 2 secs
Total CPU time to MAP completion (all processors):   51 mins 8 secs

Mapping completed.
See MAP report file "system_map.mrp" for details.
```

Apéndice E

Modelo Simulink ROACH 2