Universidad Nacional de Mar del Plata

Facultad de Ingeniería

Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva

Detector de Energía
Informe de Proyecto Final

Autor:
Lopez, Cristian Hernán

Director:
Dra. Ing. De Micco, Luciana

Co-Director:
Dr. Ing. Antonelli, Maximiliano

Marzo 2023

Universidad Nacional de Mar del Plata

Facultad de Ingeniería

Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva

Detector de Energía
Informe de Proyecto Final

Autor:
Lopez, Cristian Hernán

Director:
Dra. Ing. De Micco, Luciana

Co-Director:
Dr. Ing. Antonelli, Maximiliano

Marzo 2023

A mis padres, Patricia y Marcelo.
A mi novia, Florencia.
A mi abuela, Rita.
A la memoria de mis abuelos, Mario, Ana y Horacio

Agradecimientos

A mis directores Luciana y Maximiliano por darme la oportunidad de trabajar en esta investigación, ayudarme y apoyarme con sus conocimientos.

A mi familia por el apoyo incondicional que recibí de cada uno de ellos para concluir con esta etapa.

Resumen

Las redes de Radio Cognitiva (RC) permiten una utilización eficiente del espectro de potencias. Un aspecto clave para su buen funcionamiento es el sensado espectral de los canales. El detector de energía es el método más usado para esta tarea por las ventajas que presenta, sin embargo, su efectividad se degrada rápidamente cuando la potencia de ruido en el canal no es bien estimada.

Este trabajo de investigación y desarrollo pretende presentar una implementación real de este sensor y estudiar las consideraciones necesarias para una buena estimación del ruido. Se presentará un informe detallado acerca de los recursos empleados y resultados de simulación y posterior implementación en una FPGA. Además, se implementarán distintos bancos de pruebas que logren evaluar el método investigado para su posterior comparación con los nuevos métodos que se están desarrollando en el laboratorio.

Índice general

Capítulo 1

Introducción

Ante la saturación de los canales de comunicaciones debida al aumento acelerado de aplicaciones inalámbricas surge la técnica de Radio Cognitiva. Esta técnica permite a usuarios sin licencia (usuarios secundarios) utilizar temporalmente bandas del espectro con licencia, mientras los usuarios con licencia (usuarios primarios) estén inactivos. De aquí surge la necesidad de algoritmos y métodos de detección para sondear el espectro de radio y determinar su estado (ocupado o libre). Una de las técnicas utilizadas en la actualidad es la llamada "Detección de Energía", si bien se trata de una de las técnicas más comunes con algunas desventajas, surge la necesidad de su implementación en FPGA para su posterior comparación con los algoritmos que se encuentran desarrollando actualmente en el grupo de investigación de Mecánica estadística y Sistemas no lineales del Laboratorio de Sistemas Caóticos perteneciente al ICYTE (Conicet-UNMDP). Este paso es imprescindible para la evaluación de los nuevos algoritmos, tanto en determinar la máxima frecuencia de operación, los recursos necesarios, y la performance presentada.

El grupo de Sistemas Caóticos en el que se está inserto como becario trabaja en forma conjunta con el Laboratorio de Comunicaciones perteneciente al mismo instituto y tienen el proyecto en común "Diseño y desarrollo de sistemas de comunicaciones basados en Software Defined Radio (SDR)" (PIP 2017-2019). Actualmente se encuentran desarrollando nuevos algoritmos de sensado, se busca que sean rápidos, adaptativos y fiables, y estén basados en cuantificadores provenientes de la teoría de la información (entropía, complejidad, desequilibrio, medidas de Fischer, etc.) y de herramientas de análisis de sistemas no lineales (dimensión de correlación, exponentes de Lyapunov, etc).

Este proyecto final tiene por objetivo la caracterización de la técnica de sensado espectral por detección de energía. Para ello, en primera instancia, se investiga la técnica como tal y la óptima estimación de la potencia de ruido. Por otro lado se desarrolla un diseño e implementación del sistema en FPGA y se simula el mismo. Por último, se toman mediciones de parámetros tales como recursos necesarios, la frecuencia máxima de operación permitida, la latencia, entre otros, en el circuito implementado en una placa ZedBoard de la empresa Xilinx. Estos datos son informados y entregados al Laboratorio de Sistemas Caóticos para su posterior comparación con nuevos algoritmos desarrollados en el mismo, junto con la implementación.

Capítulo 2

Anteproyecto

El proyecto se basa en el estudio e implementación de la técnica de detección de energía para Radio Cognitiva. Se divide en tres partes fundamentales: la investigación enfocada en la estimación de la potencia de ruido, el diseño de un detector de energía, la implementación en FPGA del detector junto con una interfaz y bancos de prueba para su posterior comparación con otros algoritmos investigados en el laboratorio de Sistemas Caóticos de la Facultad de Ingeniería.

A continuación, se presentan secciones que detallan requerimientos y especificaciones, y una introducción al plan de proyecto. Si se precisan más detalles de lo mencionado se invita a leer el **Apéndice A** en donde se encuentra detallado el plan de proyecto y el **Apéndice B** donde se detallan las especificaciones de requerimientos.

2.1. Requerimientos

Se busca implementar en una placa Zedboard de la empresa Xilinx, el algoritmo de sensado de espectro por detección de energía. Se requiere que el sistema desarrollado permita variar los siguientes parámetros:

- N: Número de muestras tomadas en el canal para evaluar su estado, por ejemplo N=16, 32, 64,128.
- Precisión empleada en cálculos internos (cantidad de bits y arquitectura empleada).
- Con respecto al cálculo de nivel de comparación o umbral $z_{th} = (Q^{-1}_{(P_{FA})}\sqrt{2N} + N)\sigma^2_w$:
    - P_{FA}: Probabilidad de falsa alarma permitido empleado para el cálculo del umbral.
    - N_{th}: Número de muestras empleadas para la estimación de la potencia de ruido.

Para poder realizar múltiples pruebas se requiere que la salida "Canal libre/ocupado" se implemente como un contador que sume la cantidad de veces que se detecta el canal ocupado.

También se requiere que se implementen los bloques que permitan verificar el funcionamiento, y realizar los siguientes TESTs:

- TEST 1: Primera aproximación: Se considera a la señal modulada muestreada como una variable aleatoria Gaussiana.
- TEST 2: Segunda aproximación: Se simula la salida del ADC con un generador de señal modulada en BPSK más ruido Gaussiano.

2.2. Introducción al plan de proyecto

El proyecto arrancó a principios de 2021 como parte de una beca de investigación para estudiante avanzado otorgada por la Universidad Nacional de Mar del Plata. Como parte de la beca se realizó una inserción al Laboratorio de Sistemas Caóticos perteneciente al ICYTE (Conicet-UNMDP), con el fin de realizar una investigación acerca de algoritmos de sensado espectral para Radio Cognitiva.

Ese año se realizaron cursos y se dio comienzo a una investigación centrada en el estudio de la estimación de la potencia de ruido para la técnica de detección de energía. Se comenzó a trabajar mediante simulaciones con el fin de llevar un diseño optimizado a una placa Zedboard proporcionada por el laboratorio. A fines del 2021 se presentó un artículo con el trabajo de investigación en el Congreso Argentino de Sistemas Embebidos [1].

La fecha estimada de finalización del proyecto se correspondía con la fecha de finalización de la beca en marzo del 2022 pero se decidió agregar algunos aspectos al proyecto que no estaban pensados en un principio, como una interfaz de usuario para realizar pruebas. Esto atrasó algunos meses la finalización del mismo.

Capítulo 3

Proyecto

3.1. Investigación

Durante la investigación, en paralelo con el diseño, la simulación y la implementación del detector de energía se realizaron distintos cursos con el fin de interiorizarse en el uso de la FPGA y su programación. Se realizaron los cursos 'Joint ICTP-IAEA School on FPGA-based SoC and its Applications for Nuclear and Related Instrumentation' dictado por investigadores del Centro internacional de Física Teórica y 'Diseño digital avanzado' dictado por investigadores de la Universidad Nacional de Córdoba y la Fundación Fulgor.

Por otro lado se realizó un estudio del funcionamiento de la técnica de detección de energía como tal. Se investigó principalmente la estimación de la potencia de ruido en esta técnica. A continuación se realiza un resumen de lo investigado. Si se requieren más detalles acerca de la investigación se invita a leer el artículo publicado en el Congreso Argentino de Sistemas Embebidos titulado 'Análisis de estimación de potencia de ruido e implementación de Detector de Energía para Radio Cognitiva'[1]. Una versión extendida del artículo será enviada a una revista internacional.

3.1.1. Detector de Energía

En la técnica por detección de energía, la señal recibida por el usuario secundario y_n contiene ruido Gaussiano aditivo (AWGN) w_n y puede poseer, o no, señal del usuario primario (PU, por sus siglas en inglés) s_n.

$$y_n = \begin{cases} w_n & : H_0 \\ s_n + w_n & : H_1 \end{cases} \tag{3.1}$$

El detector de energía realiza la siguiente operación:

$$z_n = \frac{1}{N} \sum_{i}^{N} (y_n)^2 \tag{3.2}$$

Las dos hipótesis posibles son que el canal esté libre (H_0) o que el canal esté ocupado (H_1), como se ve en la Ec. 3.1. En el caso de H_0, la señal recibida es sólo ruido Gaussiano, al ser muestreada y afectada por el cuantificador resulta una variable aleatoria con distribución χ^2 con N grados de libertad. En el caso de H_1, la señal de información muestreada puede ser aproximada a una distribución Gaussiana ya que la frecuencia de muestreo no tiene ninguna correlación con la frecuencia de portadora ni la velocidad de señalización. Con esta consideración la señal muestreada

será la suma de dos variables aleatorias Gaussianas. A la salida del cuantificador se obtiene nuevamente una variable aleatoria con distribución χ^2 con N grados de libertad. En los dos casos, H_0 y H_1, para un valor de N suficientemente alto, las dos distribuciones χ^2 pueden ser aproximadas a distribuciones Gaussianas, cada una con su valor medio y dispersión [2, 3].

$$z_n = \begin{cases} \sim \mathcal{N}(N\sigma_w^2,\, 2N\sigma_w^4)\,. & : H_0 \\ \sim \mathcal{N}(N(\sigma_w^2+\sigma_s^2),\, 2N(\sigma_w^2+\sigma_s^2)^2)\,. & : H_1 \end{cases} \tag{3.3}$$

De esta forma, se define la probabilidad de falsa alarma (P_{FA}) como la probabilidad de detectar el canal como ocupado estando libre (bajo H_0), y la probabilidad de detección (P_d) como la probabilidad de que se detecte el canal como ocupado, estando ocupado (bajo H_1), como se muestra en la figura 3.1. Luego, la curva ROC (Receiver Operation Characteristics) [4], que grafica P_{FA} versus P_d, es empleada usualmente para cuantificar y comparar el desempeño que presentan estos sensores.

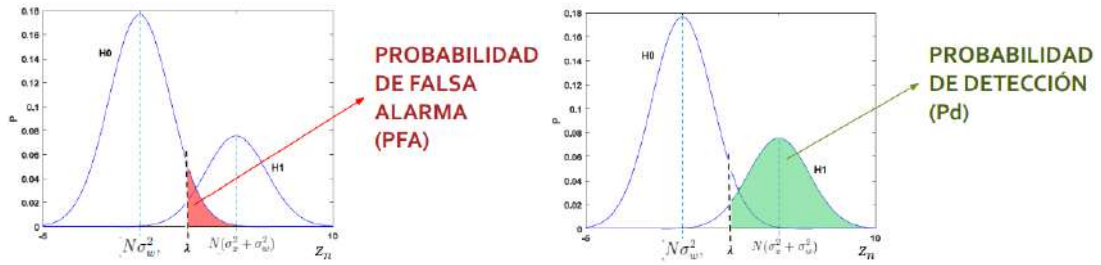


Figura 3.1: Definiciones de probabilidad de falsa alarma y de detección a partir de las hipótesis.

3.1.2. Estimación de la potencia de ruido para el cálculo del umbral

Considerando las aproximaciones realizadas, se determina la P_{FA} admitida, y se define analíticamente el valor umbral (z_{th}) por encima del cual se tomará el canal como ocupado y por debajo, como libre (criterio de Neyman-Pearson [5]).

$$z_{th} = \left(\frac{Q^{-1}_{(P_{FA})}}{\sqrt{N}} + 1 \right) \sigma_w^2 \tag{3.4}$$

Donde $Q^{-1}_{()}$ representa la inversa de la función de distribución acumulativa Gaussiana Q. En la Ec. 3.4 se ve que el cálculo del umbral requiere conocer el valor de la potencia de ruido en el canal. Conocer este valor de forma exacta es imposible [6], por lo que este requisito es una desventaja del método. En una aplicación real, esta potencia es estimada mediante la toma de N_{th} muestras en un canal que se sepa libre de transmisiones, antes de comenzar el sensado.

La figura 3.2 muestra el efecto sobre el desempeño del sensor para distinta cantidad de muestras de ruido tomadas en la estimación de su potencia. Se muestran cinco corridas para estimaciones de ruido mediante diferentes N_{th}, las cinco corridas mantienen el color de las curvas. Además, en negro se puede ver la curva analítica, ésta se corresponde con el caso (irreal) en el que se tuviera un conocimiento exacto de la potencia de ruido en el canal. Podemos ver que esta curva presenta un comportamiento cercano al ideal en el que señal y ruido pueden separase perfectamente y resulta en una línea horizontal en $P_d = 1$. Podemos ver que para valores de N_{th}

pequeños la curva se acerca a la diagonal (peor desempeño) y sobre todo se puede ver que las curvas de cada corrida tienen una gran dispersión. Para $N_{th} = 1000$ podemos decir que, si bien la *performance* no es la ideal, es relativamente buena y, sobre todo, las cinco corridas convergen en una sola curva.

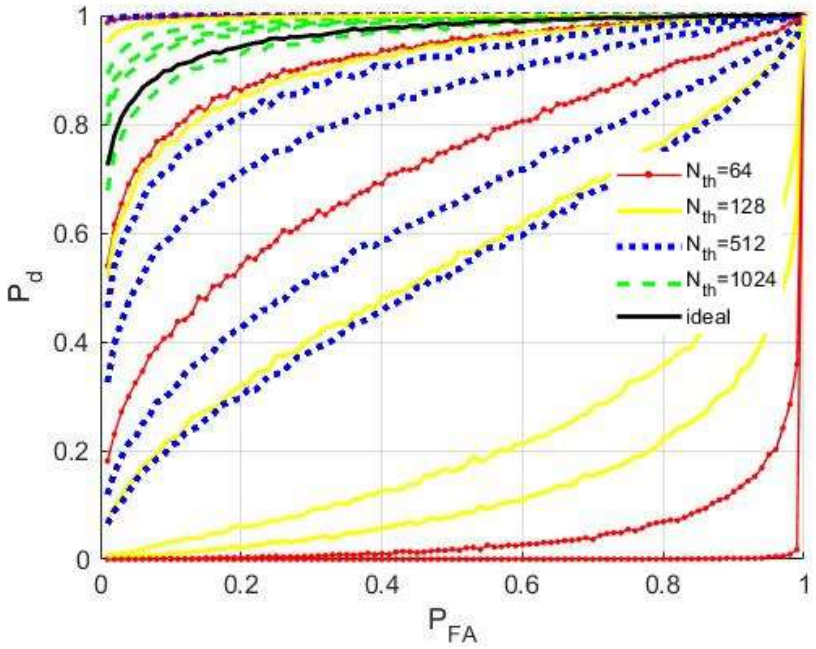


Figura 3.2: Curva ROC para N=1024, relación señal a ruido (SNR) -7dB, y modulación bpsk para distintos valores de N_{th}.

3.2. Diseño

El dispositivo se diseñó en distintos módulos que se encuentran técnicamente detallados en la especificación técnica contenida en el **Apéndice D**, paralelamente se realizaban pruebas y simulaciones para constatar que el mismo funcionara correctamente. Se utilizó la herramienta Vivado de Xilinx para realizar el diseño del circuito, mientras que para el código del microcontrolador embebido se utilizó la herramienta SDK de Xilinx. La simulación se realizó mediante el simulador propio de la herramienta con testbenchs y la herramienta System Generator de Xilinx y Simulink que brinda una interfaz de testeo más simple. El detalle de las simulaciones realizadas se pueden ver en la **sección 3.3**.

En la figura 3.3 se puede observar un diagrama general simplificado del diseño realizado. En él se pueden apreciar los distintos módulos diseñados e implementados en la FPGA, por otro lado la interfaz de testeo y la entrada de señal por parte de un SDR o bien un generador de señales empleado en las simulaciones. Por otro lado, se puede observar cómo se realiza la interconexión con los distintos componentes del sistema.

Para el diseño del circuito se utilizó lenguaje de descripción de hardware VHDL. Se hizo uso de componentes propietarios incluidos en la herramienta y a su vez se debieron diseñar íntegramente algunos de ellos necesarios para la solución final. Por otro lado, para el diseño de la interfaz gráfica de testeo se utilizó lenguaje de programación Python.

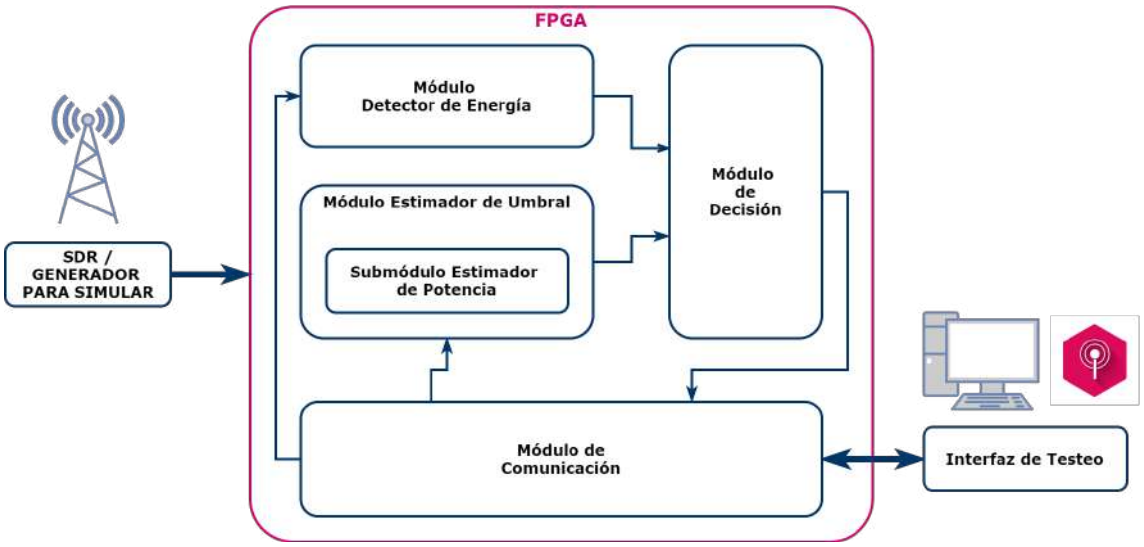


Figura 3.3: Diagrama simplificado del diseño.

Para el diseño de la interfaz se pensó en realizar una plataforma simple y de código abierto que permita realizar testeos personalizados con el circuito y además funcionara para evaluar otros sensores. En la figura 3.4 se puede observar la pantalla principal de la plataforma diseñada, se creó además un manual de usuario y soporte [7] donde se detallan cada pantalla y la lógica detrás del código.

Figura 3.4: Interfaz de testeo.

3.3. Simulaciones

3.3.1. Matlab - Simulink - System Generator

Para validar su correcto funcionamiento, se probó el comportamiento del circuito implementado mediante simulaciones sobre el propio diseño. Para ello, se importó el circuito al entorno System Generator de Xilinx-Simulink, lo que permitió que el diseño interactuara con los bloques de simulación de la herramienta. En la figura 3.5 se puede observar el circuito importado en la plataforma de simulación.

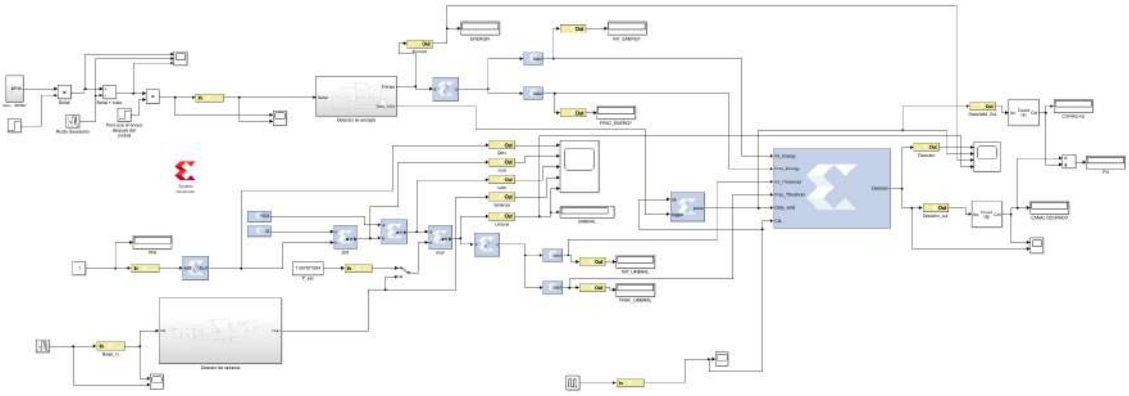
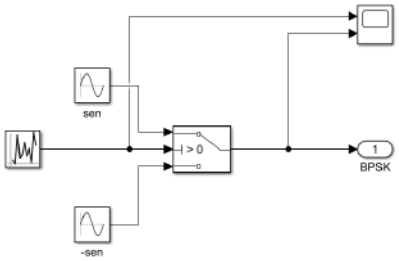
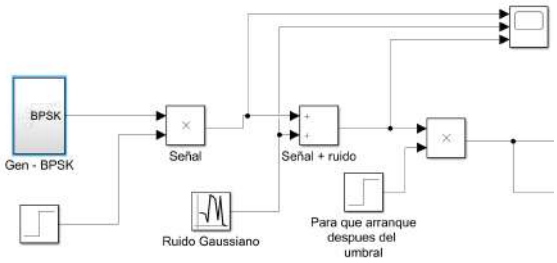


Figura 3.5: Diseño importado en el entorno System Generator.

Para la generación de la señal de ruido se utilizó un generador de ruido Gaussiano integrado en la herramienta de simulink. Para el generador de señal contaminada con ruido se simuló un generador con modulación bpsk, como se muestra en la figura 3.6, al que se le sumó ruido blanco Gaussiano con una SNR de 7dB.



(a) Generador BPSK



(b) Generador BPSK contaminado con ruido

Figura 3.6: Simulación de generador BPSK.

En la figura 3.7 se puede observar la simulación realizada del sistema completo en System Generator. El gráfico superior indica cada vez que se detecta un valor de energía mayor al umbral, el gráfico del medio corresponde al disparador generado para evaluar la energía y el umbral y tomar una decisión. Por último el gráfico inferior corresponde a los valores de energía y umbral. Se puede observar cómo el umbral es calculado una sola vez al principio y como luego la energía se mantiene cercana al mismo.

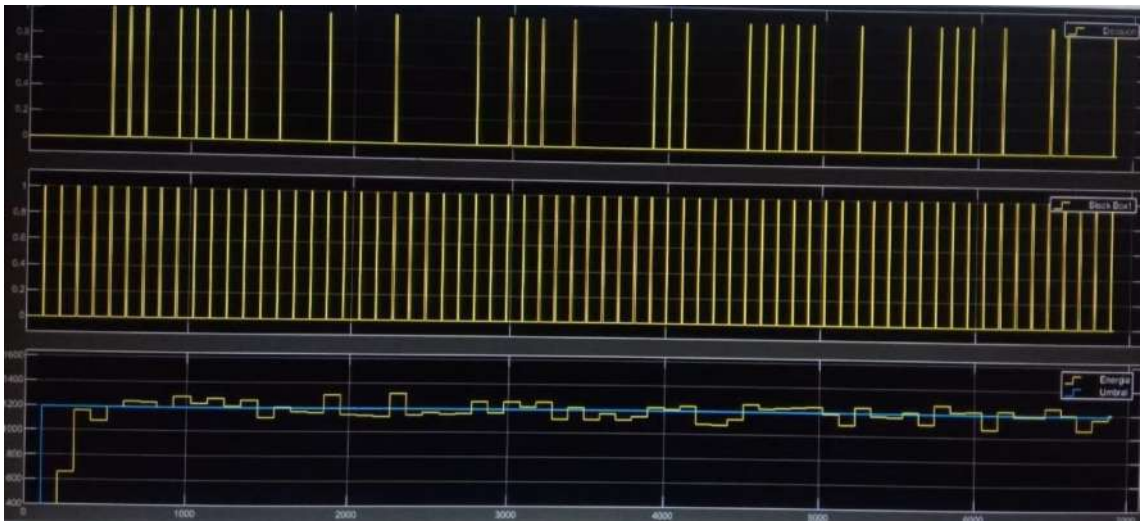
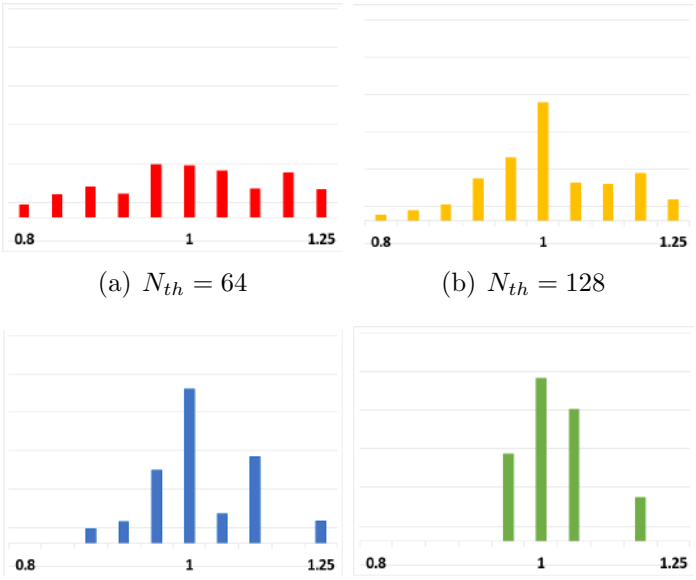


Figura 3.7: Simulación del diseño en System Generator.

Para evaluar el comportamiento del bloque estimador de potencia implementado para diferentes números de muestras, se calculó la potencia de ruido 550 veces por cada N_{th}. Las potencias estimadas obtenidas se trazaron en los histogramas que se muestran en la figura 3.8. Se puede ver que para valores más bajos de N_{th}, la potencia medida tiene una amplia dispersión y no está centrada en el valor real de la potencia (1). Es decir que a medida que aumentan las muestras tomadas (N_{th}), la varianza disminuye tendiendo al valor real. Esta situación cambia a medida que se aumentan las muestras y para $N_{th} = 1024$ se puede observar que en el histograma aparecen solo cuatro barras.



(a) $N_{th} = 64$ (b) $N_{th} = 128$

(c) $N_{th} = 512$ (d) $N_{th} = 1024$

Figura 3.8: Histograma de la estimación de potencia de ruido en el bloque del estimador de potencia.

3.3.2. Testbench - Vivado

La figura 3.9 muestra la simulación de todo el detector al realizar una operación de detección, la misma se realizó a partir de un testbench diseñado para este fin. Se

utilizó en este caso la herramienta de simulación incluida en el software de diseño Vivado.

La señal *Signal_In* simula la señal entrante a través de la antena y recibida por el sensor. Inicialmente, se compone solo de ruido y, después de unos microsegundos, se convierte en una señal sinusoidal que simula una portadora de usuario principal.

Primero, el bloque que calcula el umbral toma las muestras de solo ruido, estima la potencia y calcula el umbral. Cuando finaliza, su valor se muestra en señal *Int_Thresholds* y *Frac_Treshold*, parte entera y parte fraccionaria respectivamente. En ese momento, la señal *Threshold_end* genera un pulso. A partir de este momento se inicia el sensado del canal, para determinar la presencia o no de un usuario primario. El sensor toma N muestras para calcular la potencia (z_n) (mostrada en señales *Int_Energy* y *Frac_Energy*, parte entera y parte fraccionaria respectivamente). La señal *Data_valid* genera un pulso cada vez que finaliza el cálculo de energía, en ese momento si el valor de energía supera el umbral, la señal *Decision* se pondrá a uno, de lo contrario a cero.

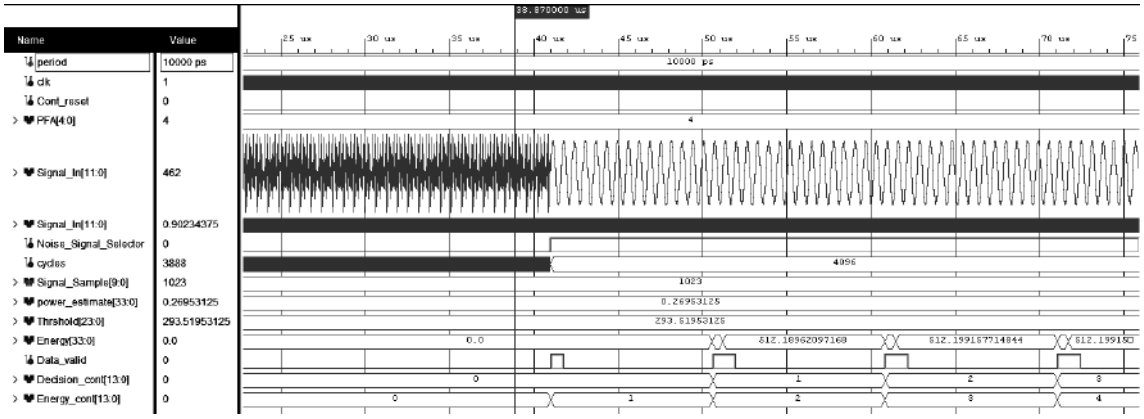


Figura 3.9: Simulación a partir de testbench.

3.3.3. Performance de interfaz

3.4. Implementación

El circuito fue implementado en una placa Zedboard de la empresa Xilinx. Consta de dos bloques principales, uno estima la potencia de ruido y calcula el umbral, se activa antes de que comience la detección (la antena debe sintonizar un canal de señal conocido libre de PU). El segundo es el propio cuantificador que se activa durante la detección (la antena debe sintonizar el canal de interés). En la figura 3.10 se puede observar la utilización de celdas de la FPGA luego de realizar la implementación.

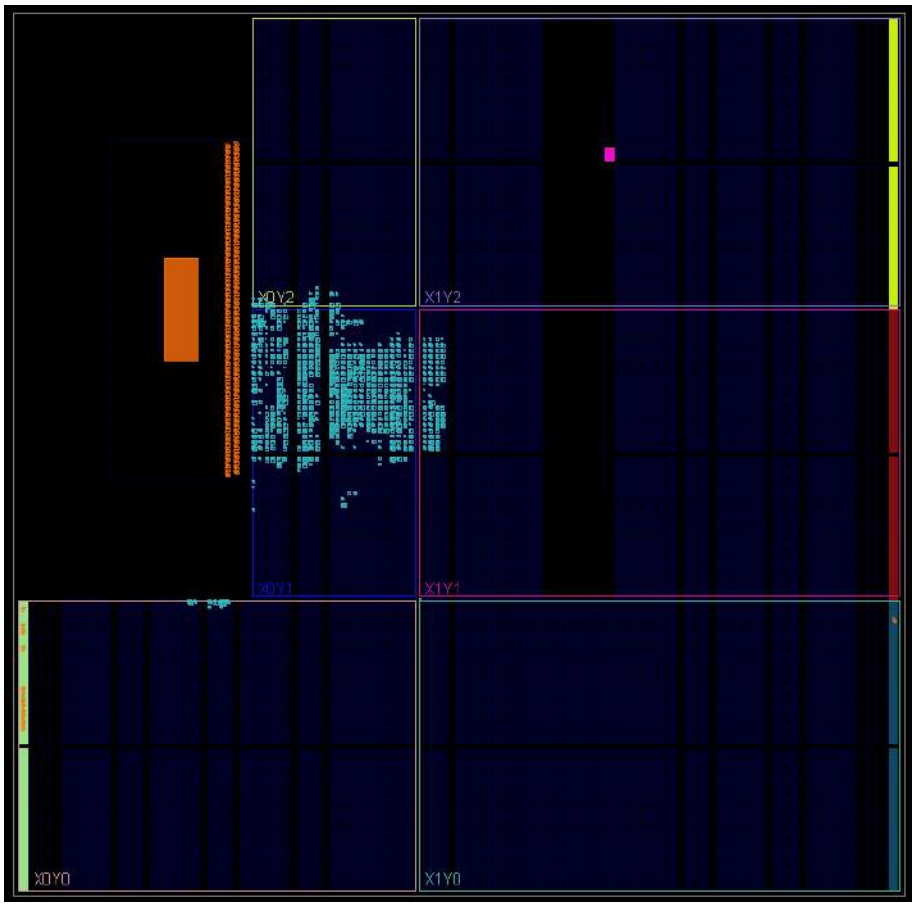


Figura 3.10: Diagrama de implementación en celdas de FPGA.

La tabla 3.4 resume los recursos utilizados en el circuito. Se puede observar que el diseño implementado utiliza menos del 10 % de los recursos disponibles en la FPGA. El circuito utiliza $3,4$ % del total de LUTs (4 para memoria y 1.820 para implementar lógica), $0,5$ % de los registros y 4 % de los Slices disponibles en la FPGA. La tabla también muestra que la frecuencia máxima permitida en esta implementación es de 182 MHz, esto podría mejorarse ya que en esta primera implementación no se priorizó la velocidad.

LUTs (53.200)	**Registros** (106400)	**Slice** (13.300)	**LUTs como memoria** (17.400)	**LUTs como lógica** (53.200)	f_{max} **[MHz]**
1,654	999	584	1599	55	182

Tabla 3.1: Recursos utilizados en la implementación en Zedboard Xilinx

Al realizar un reporte de la potencia estimada que consumirá el circuito se puede observar una potencia total de 1.688 W, donde casi en su totalidad se trata de la potencia del microcontrolador embebido en la FPGA. En la figura 3.11 se observa el reporte completo de potencia, donde *PS7* corresponde al microcontrolador embebido.

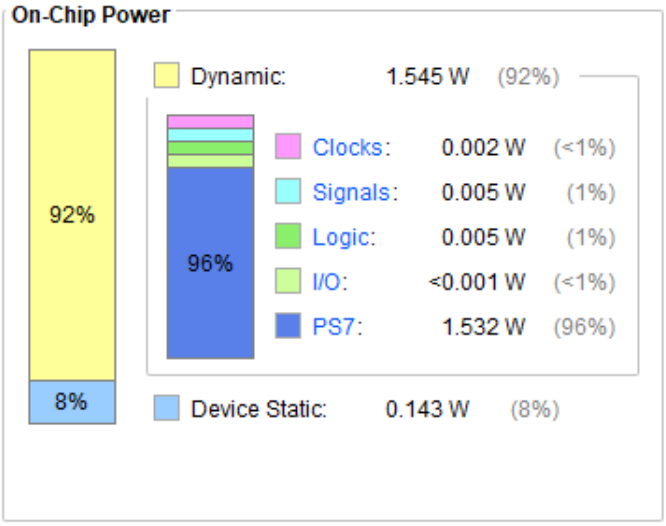


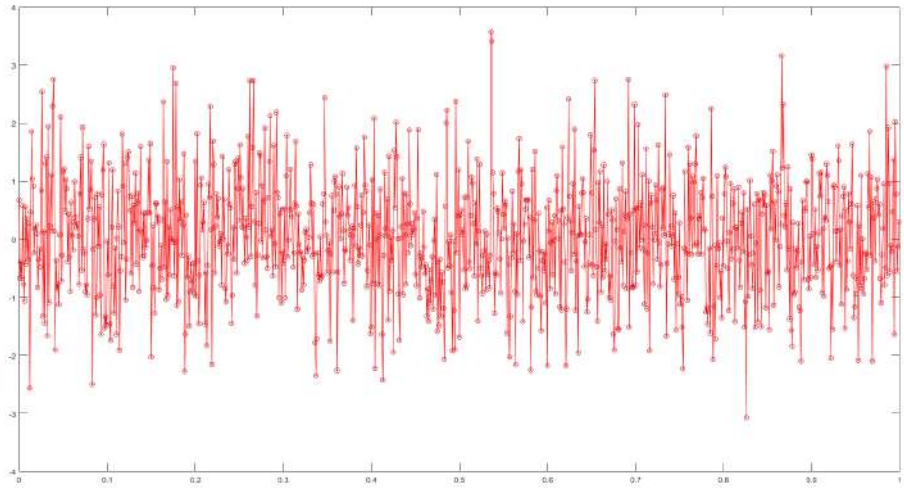
Figura 3.11: Reporte de potencia.

3.5. Testeos

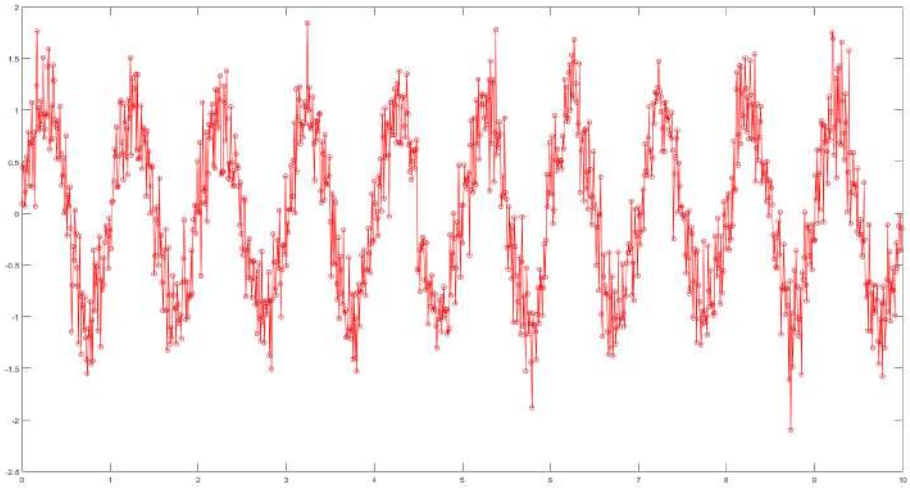
3.5.1. Generadores de prueba

Para la realización de testeos se simuló un generador de ruido Gaussiano y un generador de señal contaminada con ruido con una placa Arduino. De esta manera las señales ingresan a la FPGA ya digitalizadas como si hubiesen sido tomadas analógicamente y muestreadas digitalmente. En la placa Arduino se guardaron las señales como vectores de bits. Luego una función implementada en el microcontrolador, recorre estos vectores y genera en los pines digitales los bits necesarios para las distintas señales, a una frecuencia especificada por la configuración de las interrupciones del mismo.

Se realizaron dos códigos de Matlab para generar los datos cuantificados y formateados correctamente que se guardan en la memoria del microcontrolador. El primer programa genera los bits necesarios para formar una señal de ruido blanco Gaussiano. El segundo programa se encarga de generar una señal sinusoidal contaminada con ruido, con una SNR configurable. En la figura 3.12 se muestran las señales generadas por los distintos códigos.



(a) Ruido Gaussiano



(b) Sinusoidal contaminada con ruido. SNR = 7 dB

Figura 3.12: Señales generadas con códigos de Matlab.

Las señales generadas por el microcontrolador sin embargo, tuvieron que ser adecuadas para un correcto funcionamiento. Esto se debió a que las salidas digitales de la placa Arduino entregan valores de 0V para el valor lógico 0, y 5V para el valor lógico 1. Mientras que la placa de desarrollo Zedboard, que contiene la FPGA, recibe en sus entradas Pmod valores de señal de 0 y 3.3V para valores lógicos de 0 y 1 respectivamente.

Para realizar dicha adecuación de señal se barajaron varias opciones. La primer opción que se estudió fue la realización de un divisor resistivo por bit de señal, sin embargo esta solución fue descartada debido a que es necesario trabajar con altas frecuencias y las resistencias comienzan a tener efectos de capacitancias e inductancias parásitas.

La segunda opción que se analizó fue la utilización de circuitos conversores con optoacopladores como el que se observa en la figura 3.13. Esta opción también fue descartada, ya que al analizarla se observa que la entrada de la FPGA agrega una capacidad en paralelo al circuito, muy pequeña. Esta capacidad parásita agrega

un polo en altas frecuencias que deforma la señal de salida del Arduino. Por lo tanto, se debería agregar un capacitor de compensación al circuito para eliminar el efecto. Como este debe ser demasiado pequeño, sería necesario agregar una capacidad externa a la entrada de la FPGA para, luego, utilizar un capacitor de compensación de un valor mayor a este último. Todo esto debería diseñarse para tener una respuesta lo mas plana posible a la frecuencia de trabajo.

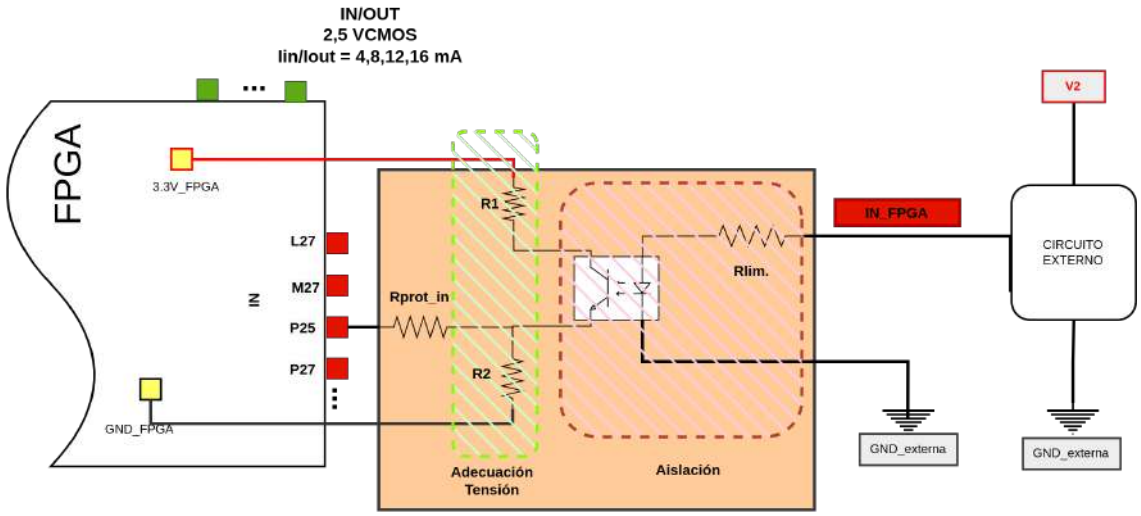


Figura 3.13: Conversor con optoacopladores.

Por último la opción que se analizó y se optó por utilizar, se trata de unos módulos comerciales conversores de niveles lógicos como el que se muestra en la figura 3.14. Estos módulos están basados en transistores mosfet BSS138. Esta opción cuenta con varias ventajas respecto a las otras, en principio el componente principal del módulo (BSS138) es un mosfet creado exclusivamente para la conversión de señales lógicas. Por otro lado, el hecho de utilizar estos componentes permite trabajar con frecuencias de hasta 30 MHz aproximadamente y no preocuparse por las capacidades parásitas que puedan aparecer. Por último, el hecho de que el módulo ya se encuentre comercializado brinda la ventaja de la robustez en cuanto al armado de la placa que no se obtiene si es diseñada por uno mismo.

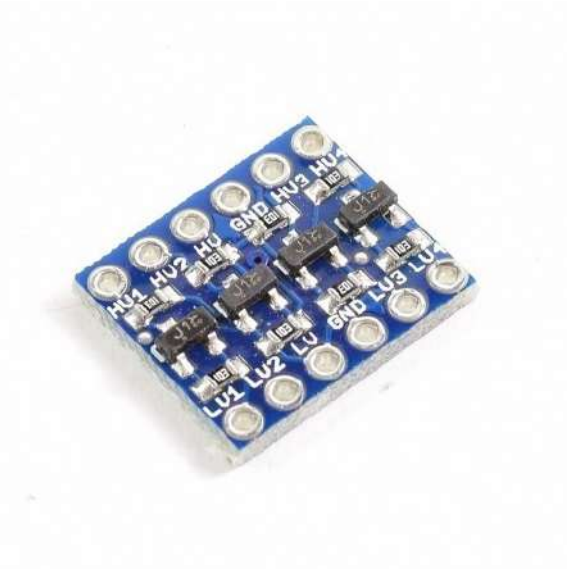
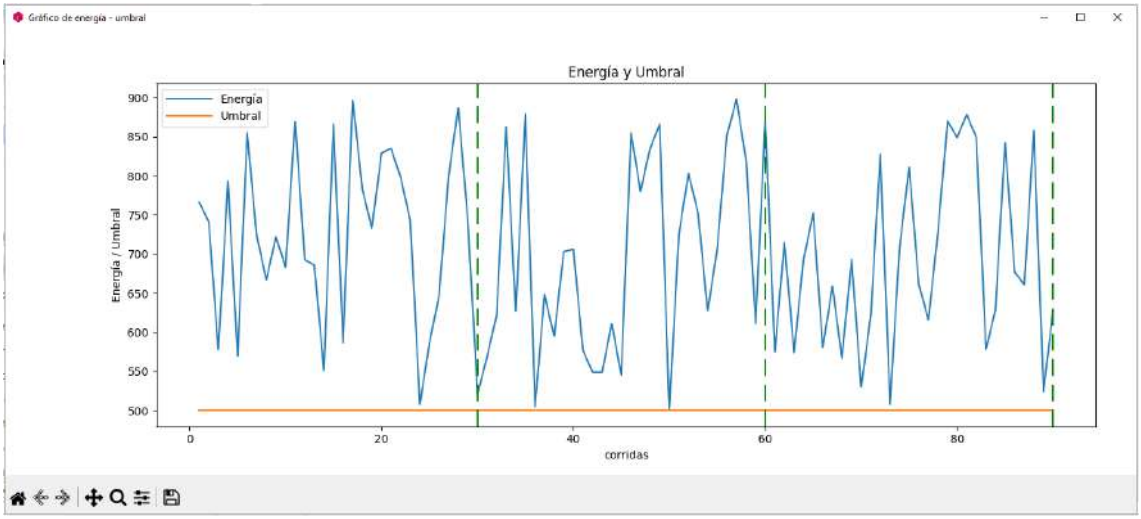


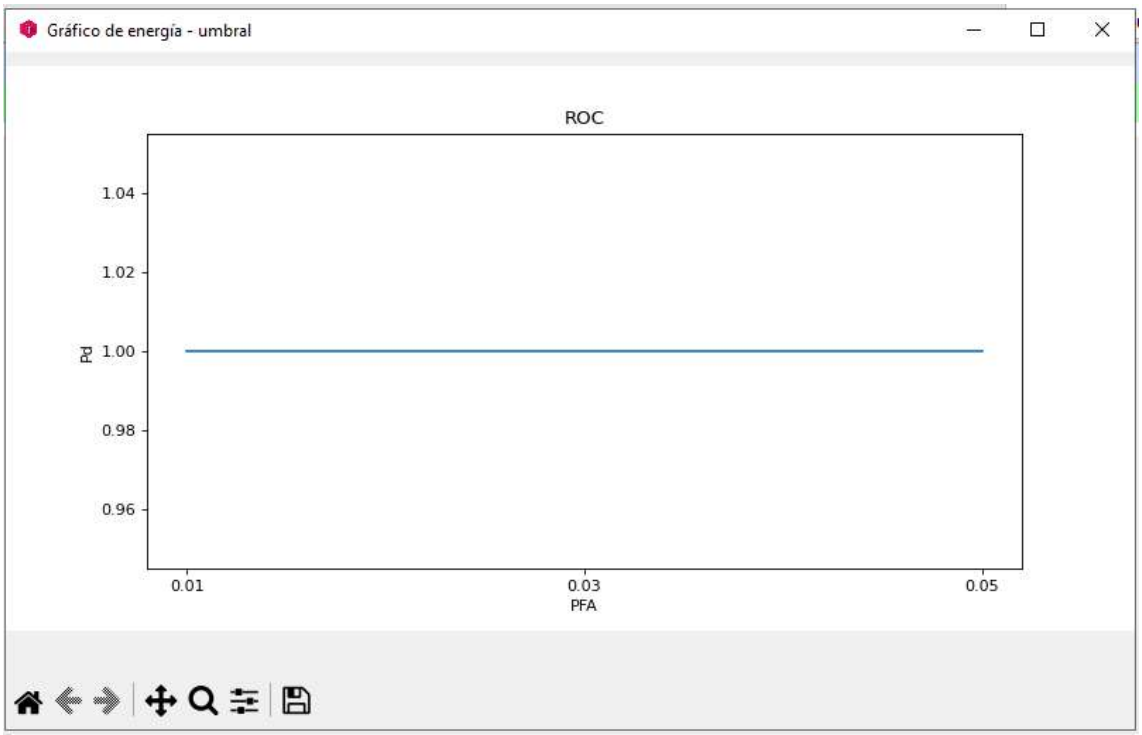
Figura 3.14: Conversor comercial basado en BSS138

3.5.2. Prueba de comunicación

Para realizar un testeo de la comunicación entre la interfaz y la FPGA, se creó un código en el sistema de procesamiento embebido de la FPGA que envía respuestas aleatorias a las consultas de la interfaz, sin utilizar la parte lógica. De esta forma se pudo emular un funcionamiento particular del circuito y comprobar las distintas respuestas de la interfaz. En la figura 3.15 se observan los gráficos de Energía-Umbral y ROC obtenidos.



(a) Gráfico de Energía-Umbral



(b) Gráfico de ROC

Figura 3.15: Gráficos obtenidos de pruebas de comunicación.

3.5.3. Prueba integral

Al realizar el testeo, la interfaz se comunica con la FPGA a través de la UART y se realiza una secuencia de procesos como se muestra en la figura 3.16. En la misma se puede observar la interacción entre la interfaz escrita en Python y las dos partes de la FPGA. Por un lado el microcontrolador embebido ($PS - FPGA$) y por otro la lógica programable ($PL - FPGA$), donde se encuentra embebido el circuito del Detector de Energía.

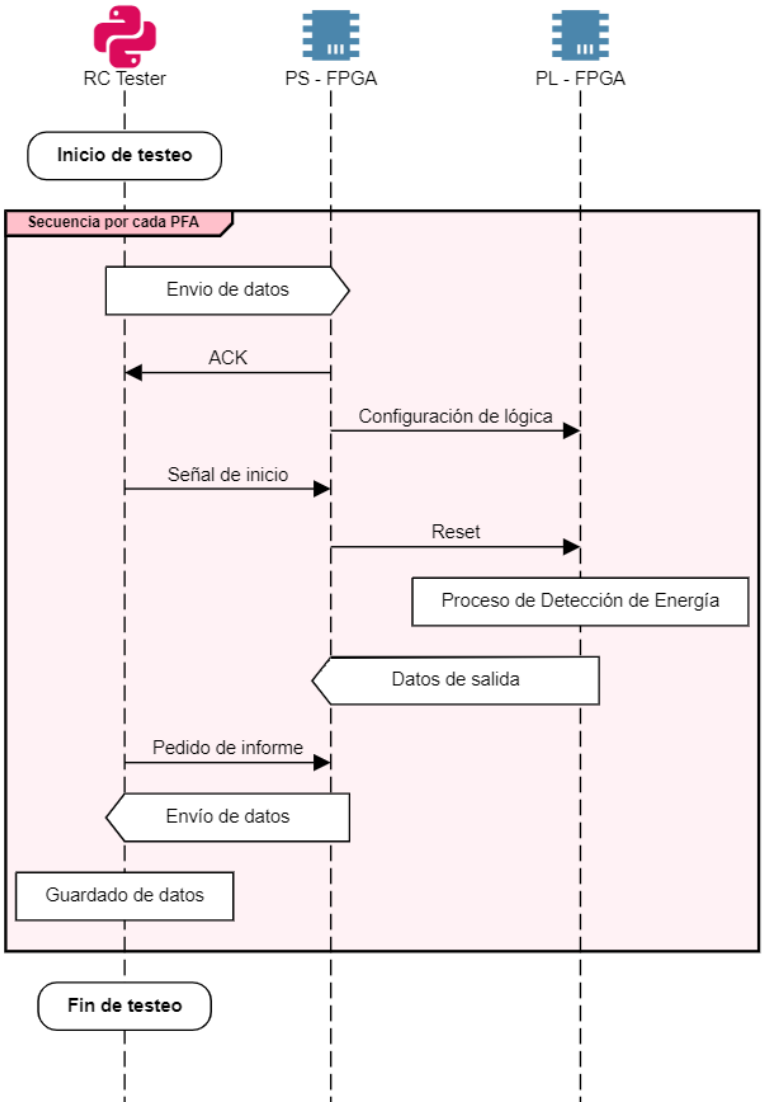


Figura 3.16: Secuencia de testeo.

En la figura 3.17 se puede observar un diagrama de conexión de la FPGA, tanto con la PC que contiene la interfaz de testeo, como con el generador de señales. Por otro lado, en la figura 3.18 se observa una imagen del dispositivo conectado y funcionando.

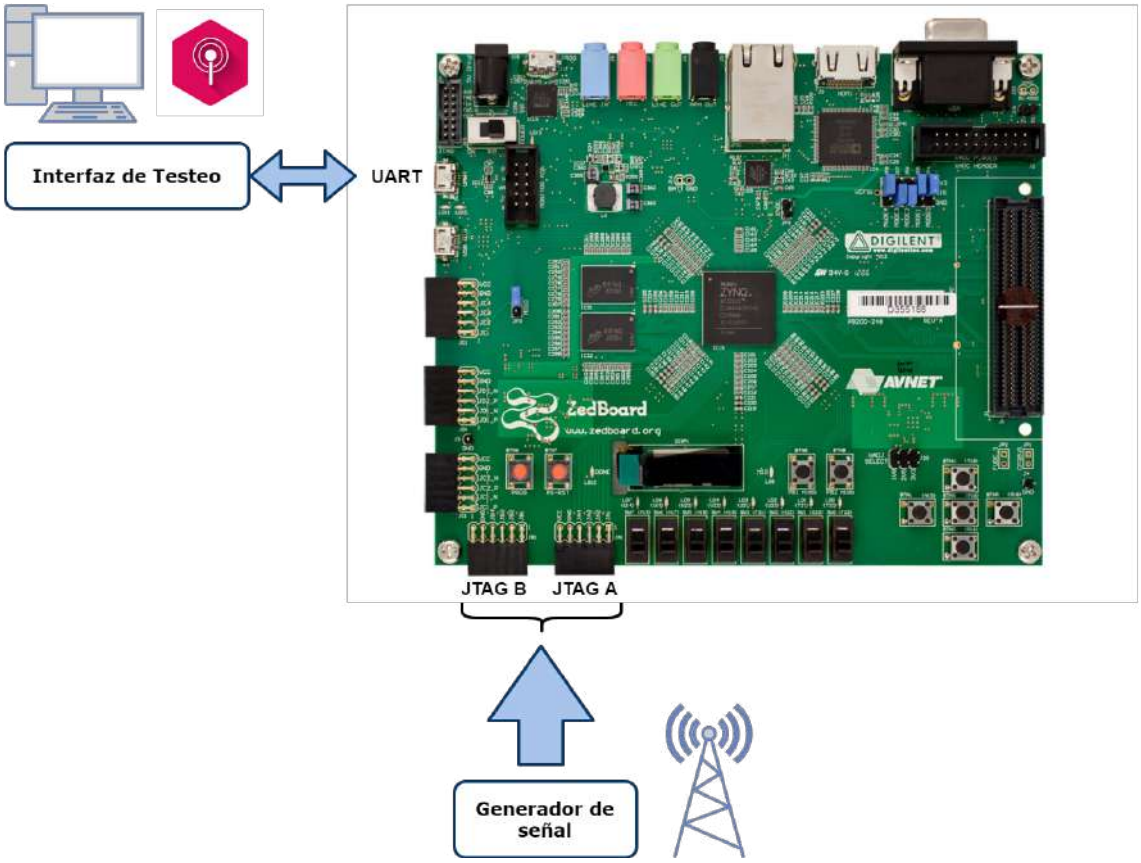


Figura 3.17: Diagrama de conexión de FPGA.

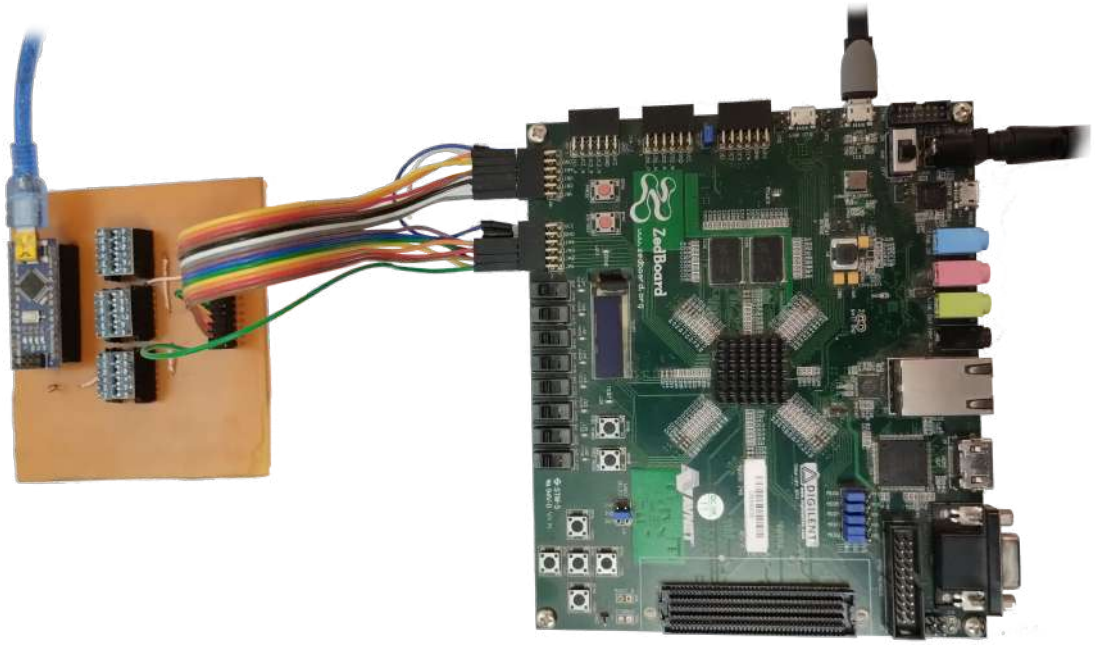


Figura 3.18: Dispositivo conectado.

Se realizaron varios testeos con distintas señales de pruebas. En particular se realizaron los dos testeos detallados en la Especificación de Requerimientos del **Apéndice B**.

TEST 1: Primera aproximación

Para este testeo se considera a la señal modulada muestreada como una variable aleatoria Gaussiana, es decir que ambas señales (ruido y modulada) son representadas por ruido Gaussiano. Para este caso se realizaron tres pruebas consecutivas y se graficaron las curvas ROC resultantes de cada prueba en un único gráfico mostrado en la figura 3.19.

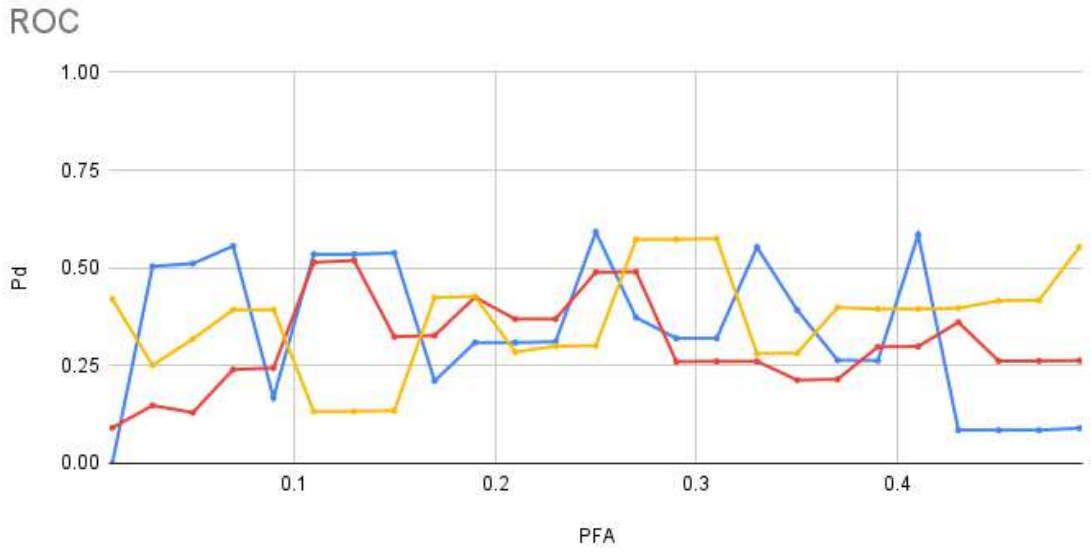


Figura 3.19: ROC TEST 1

Se puede observar cómo la probabilidad de detección (Pd) se mantiene en valores bajos para todas las probabilidades de falsa alarma.

TEST 2: Segunda aproximación

Para este testeo se considera a la señal modulada muestreada como una señal sinusoidal contaminada con ruido. En este caso se realizaron distintas series de pruebas cambiando la relación señal a ruido de la señal modulada. Para cada serie se realizaron tres pruebas consecutivas y se graficaron las curvas ROC resultantes de cada una en un único gráfico mostrado en la figura 3.20. En el caso particular de la imagen se observa la serie de pruebas realizadas con una señal modulada con $SNR = -7dB$, este caso es significativo ya que se utilizó la misma SNR que para las simulaciones realizadas para la entrega del trabajo para el CASE [1].

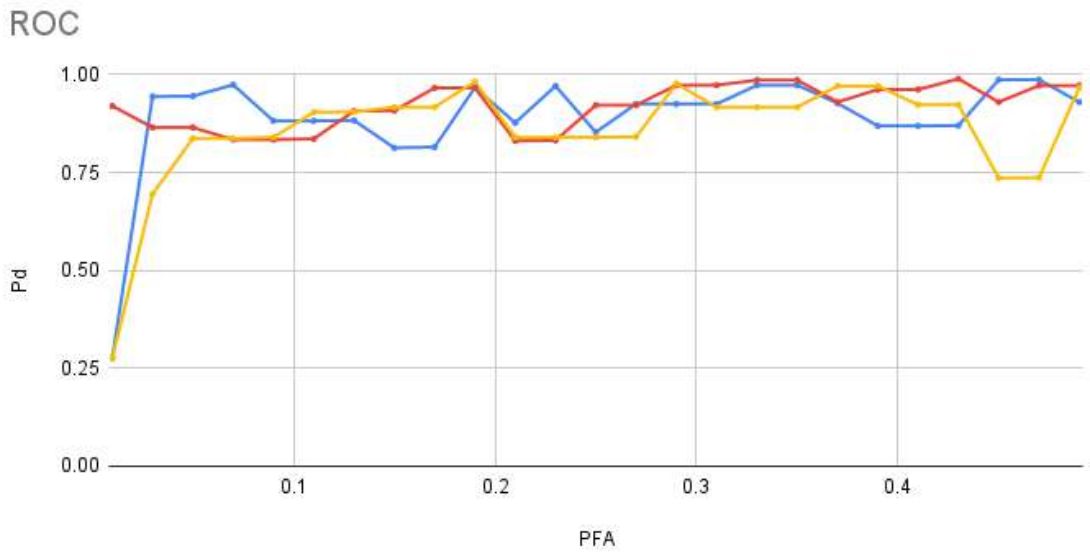


Figura 3.20: ROC TEST 2. Señal modulada con $SNR = -7dB$

Se puede observar cómo la probabilidad de detección (Pd) aumenta significativamente con el aumento de la probabilidad de falsa alarma, comportamiento similar al obtenido teóricamente.

Capítulo 4

Conclusiones

4.1. Resultados de la investigación

Gracias al estudio de las ecuaciones, la implementación del detector de energía y la simulación del mismo mediante distintas herramientas, se logró dar con una cantidad de muestras de señal de ruido óptima para una buena estimación del umbral. Se llegó a la conclusión que con una cantidad de 1024 muestras de ruido, es posible realizar una estimación del umbral lo suficientemente buena como para no generar una gran dispersión entre las distintas curvas ROC.

El estudio realizado se presentó en un artículo titulado 'Análisis de estimación de potencia de ruido e implementación de Detector de Energía para Radio Cognitiva' en el Congreso Argentino de Sistemas Embebidos [1]. Por otro lado en el mismo congreso se realizó una presentación del trabajo y se evacuaron dudas de otros participantes.

4.2. Resultados del instrumento

Se logró implementar un detector de energía parametrizable para realizar testeos y comparaciones con otros sistemas, cumpliendo con las especificaciones encargadas en la Especificación de Requerimientos detallada en el **Apéndice B** y los requerimientos funcionales C.3.1, C.3.2 y C.3.3 . Durante el proceso de diseño del circuito e implementación se enfrentaron varias dificultades que tuvieron que ver principalmente con el uso de componentes propietarios o el diseño de componentes propios en VHDL. Se intento siempre utilizar los componentes óptimos para cada etapa.

En cuanto a la implementación final y con los datos obtenidos de área, potencia y velocidad detallados en la sección 3.4 se puede decir que se cumplió con el requerimiento no funcional C.4.1. En este caso se superó dicho requerimiento que pedía una frecuencia de trabajo de hasta 100MHz mientras que el instrumento alcanza una frecuencia máxima de 182MHz.

Se diseñó e implementó una interfaz en lenguaje Python amigable con el usuario que automatiza los testeos y registra los resultados para crear gráficas y realizar análisis. De esta forma se cumplió con el requerimiento funcional C.3.4. Junto con la interfaz se generó un manual de usuario y soporte en donde se especifica cada bloque de código con su funcionalidad. La plataforma de testeo se pensó para ser de código abierto, de forma que sea utilizada y modificada por cualquier persona.

En cuanto a los testeos, los resultados fueron consistentes con los esperados. Se lograron realizar ambos testeos especificados en los requerimientos del proyecto y ambos dieron resultados similares a las simulaciones realizadas. Se puede observar en

la figura 3.19 que la probabilidad de detección cuando ambas señales son variables aleatorias Gaussianas es muy baja. Mientras que en la figura 3.20 se puede ver como la curva de probabilidad de detección se aproxima al valor 1 al aumentar la probabilidad de falsa alarma, testeandose con una señal modulada contaminada con ruido.

4.3. Gestión del proyecto

Se presentaron varias situaciones durante el desarrollo del proyecto que llevaron a un retraso de casi un año en la finalización del mismo. El primer punto a destacar es que, al iniciar el proyecto y pensar el plan del mismo, no se tuvieron en cuenta los tiempos de estudio que llevarían las materias del último año. Esto derivó en que durante la época de cursado no se pudiera avanzar todo lo planeado.

Por otro lado, con el correr del proyecto, se introdujo la necesidad de tener una interfaz de usuario para realizar los testeos con el instrumento. Eso agregó una nueva etapa, en donde diseñó e implementó esta interfaz.

Además se realizó el curso de Diseño Digital Avanzado, entre Agosto y Diciembre del año 2021, que no estaba previsto en el plan de proyecto inicial. Esto generó mayores conocimientos pero a su vez consumió tiempo de dedicación a otras tareas planeadas para esas fechas.

Otra causa del retraso en la entrega tuvo que ver con que en Marzo del 2022 se comenzó a trabajar en una empresa. Esto tuvo como consecuencia que se tuviera que dedicar al trabajo final luego del horario laboral, acortando significativamente la cantidad de horas de dedicación que se podían aportar.

Por último se subestimó ampliamente el tiempo de pruebas. Durante las pruebas fueron cuando surgieron los principales problemas y errores de diseño, los cuales tuvieron que ser corregidos para reintentar las mismas. Estos errores tenían que ver principalmente con la implementación física del instrumento, con lo que no era posible detectarlos mediante simulaciones. En particular tomó mucho tiempo lograr estabilizar el código C que contiene el sistema de procesamiento embebido de la FPGA para comunicar la interfaz de testeo y la parte lógica.

4.4. Conocimientos adquiridos

En cuanto a conocimientos adquiridos en primera instancia es interesante destacar el gran aprendizaje sobre Radio Cognitiva. La investigación desde un principio propuso sumergirse en un ambiente desconocido por completo. Aprender desde cero el concepto de Radio Cognitiva y sensado espectral. Se estudiar1on diferentes métodos de sensado espectral y en particular se centró en la detección de energía. Se investigó este método desde sus ecuaciones mas específicas, hasta las implementaciones previamente realizadas. Se realizaron simulaciones y se estudió cómo afecta el cálculo de la potencia de ruido en la estimación del umbral. Se logró conocer en profundidad el método, de manera de implementarlo físicamente en un circuito.

Por otro lado, se profundizaron conocimientos de FPGA y VHDL. Para ello se realizaron dos cursos detallados en la sección 3.1. En ambos cursos se aprendió a optimizar y hacer mejores códigos VHDL para sintetizar circuitos. Particularmente en el curso dictado por el ICTP se adquirieron conocimientos mas orientados al sistema de procesamiento embebido de la FPGA. De allí se pudo utilizar el componente

'commblock', creado por el mismo instituto, para realizar la comunicación entre la parte lógica y el sistema de procesamiento de la FPGA.

En el curso de Diseño digital avanzado se abordaron temas relacionados con los lenguajes de programación de hardware VHDL y Verilog. Estos conocimientos fueron muy útiles a la hora de diseñar componentes. Por otro lado, en dicho curso se conoció mas en profundidad la herramienta Vivado para el diseño y la programación de la FPGA. Esto fue de gran ayuda a la hora de generar reportes y realizar simulaciones con la herramienta.

Se requirió adquirir conocimientos en programación de microcontroladores para realizar un código que vinculara el circuito lógico con la interfaz de testeo. En este sentido el aprendizaje de este lenguaje y su utilización fue el más trabajoso. En este caso se tuvo que arrancar desde lo básico el aprendizaje. Sin embargo, la mayor dificultad se produjo debido a que se tuvo que aprender a programar específicamente el sistema de procesamiento de la FPGA, con sus funciones específicas.

La utilización del lenguje C también se hizo presente en la programación de los emuladores de generador de señal de Arduino. Esto se debió a que, como la frecuencia a la que se debía trabajar era mucho mayor a la soportada por las funciones predeterminadas de Arduino, fue necesario utilizar los temporizadores del microcontrolador. De esta forma se lograron las velocidades de refresco necesarias para las salidas digitales del Arduino, pero fue necesaria la utilización de código C puro.

Por último, para la interfaz de testeo creada se utilizó lenguaje de programación Python, elegido por su gran versatilidad y disponibilidad de información. En este caso ya se tenía conocimiento previo del lenguaje. La iniciativa en el uso de Python se debe a la utilización del mismo en el trabajo diario. Específicamente para este proyecto se reforzaron conocimientos sobre la librería 'Tkinter', para el diseño de la interfaz de usuario, y la librería 'Threading', para la generación de hilos de trabajo en paralelo. La utilización de este lenguaje no presentó mayores dificultades.

Para concluir, se puede destacar que las capacidades adquiridas a lo largo de la carrera fueron sumamente importantes. La habilidad en la resolución de problemas se notó significativamente con el transcurso del proyecto final. Este proyecto se comenzó a pensar cuando se estaba terminando el 4to año de la carrera y se concluyó un año después de cursar el 5to año, en las etapas finales del proyecto se notaba una gran diferencia en la manera de afrontar y resolver los problemas.

Bibliografía

[1] Maximiliano Antonelli y Luciana De Micco Cristian Hernán Lopez. "Análisis de estimación de potencia de ruido e implementación de Detector de Energía para Radio Cognitiva". En: *CASE* (2021).

[2] Miguel López-Benıtez y Fernando Casadevall. "Improved energy detection spectrum sensing for cognitive radio". En: *IET communications* 6.8 (2012), págs. 785-796.

[3] Pratik Vijay Yadav. "An efficient hardware implementation of an energy detection-based spectral estimator for cognitive radios". Tesis doct. San Diego State University, 2016.

[4] Christopher D. Brown y Herbert T. Davis. "Receiver operating characteristics curves and related decision measures: A tutorial". En: *Chemometrics and Intelligent Laboratory Systems* 80.1 (2006), págs. 24-38. ISSN: 0169-7439. DOI: https://doi.org/10.1016/j.chemolab.2005.05.004.

[5] H Vincent Poor. *An introduction to signal detection and estimation*. Springer Science & Business Media, 2013.

[6] Rahul Tandra y Anant Sahai. "Fundamental limits on detection in low SNR under noise uncertainty". En: *2005 International Conference on Wireless Networks, Communications and Mobile Computing* 1 (2005), págs. 464-469. DOI: 10.1109/WIRLES.2005.1549453.

[7] Cristian Hernán Lopez. "Manual de usuario y soporte - Radio Cognitiva Tester". En: (2022).

Apéndices

Apéndice A

Plan de proyecto

IMPLEMENTACIÓN EN FPGA DE ALGORITMOS DE SENSADO ESPECTRAL PARA RADIO COGNITIVA

REPONSABLE DEL PROYECTO: LÓPEZ, CRISTIAN HERNÁN

Director:	Dra. Ing. De Micco, Luciana
Co-Director:	Dr. Ing. Maximiliano Antonelli
Laboratorio:	Sistemas Caóticos – ICYTE
Fechas de inicio y fin del proyecto:	26/01/2021 – 21/03/2022
Progreso:	30% (Tareas realizadas)
Tareas:	35
Recursos:	6
Riesgos:	4

Ante la saturación de los canales de comunicaciones debida al aumento acelerado de aplicaciones inalámbricas surge la técnica de Radio Cognitiva. Esta técnica permite a usuarios sin licencia (usuarios secundarios) utilizar temporalmente bandas del espectro con licencia, mientras los usuarios con licencia (usuarios primarios) estén inactivos. De aquí surge la necesidad de algoritmos y métodos de detección para sondear el espectro de radio y determinar su estado (ocupado o libre). Una de las técnicas utilizadas en la Actualidad es la llamada “Detección de Energía”, si bien se trata de una de las técnicas más comunes con algunas desventajas, se propone su implementación en FPGA para su posterior comparación con los algoritmos que se encuentran desarrollando actualmente. Este paso es imprescindible para la evaluación de los nuevos algoritmos, tanto en determinar la máxima frecuencia de operación, los recursos necesarios, y la performance presentada.

Tareas

Nombre	Recursos	Fecha de inicio	Fecha de fin
Etapa 0: Definición de proyecto	RH**: Cristian Lopez (Rol: Escritura de documentos)	14/04/21	14/06/21
Primera reunión con directores		14/04/21	14/04/21
Definición y esquematización de trabajo		14/04/21	04/05/21
Estudio Preliminar	Software de gestión de proyectos	03/05/21	03/05/21
Plan de proyecto	Software de gestión de proyectos	06/05/21	11/05/21
Especificación de requerimientos	Software de gestión de proyectos	11/05/21	13/05/21
Presentación de carta formal		15/06/21	15/06/21
Etapa 1: Capacitación y estudio de los sistemas	RH: Cristian Lopez (Rol: Investigación)	26/01/21	10/12/21
Curso en ICTP	Bibliografía, FPGA, Software: Vivado	26/01/21	22/02/21
Lectura de bibliografía específica sobre Radio Cognitiva (RC)	Bibliografía específica sobre RC	03/05/21	10/12/21
Etapa 2: Implementación y prueba de Detector de Energía	RH: Cristian Lopez (Rol: Diseño y testeo)	03/05/21	05/07/21
Estudio de simulaciones Matlab de Detector de Energía	Software: Matlab	03/05/21	07/05/21
Armado de código VHDL de Detector de Energía	Software: Vivado	07/05/21	27/05/21
Armado de Testbench para prueba en simulación	Software: Vivado	25/05/21	25/05/21
Simulación de Detector de Energía	Software: Vivado	27/05/21	02/06/21
Optimización de Detector de Energía	Software: Vivado	02/06/21	29/06/21
Presentación de resultados parciales a directores		04/06/21	04/06/21
Preparación de presentación de resumen para CASE	RH: Cristian Lopez, Directores	07/06/21	30/06/21
Presentación de resumen en CASE		06/07/21	06/07/21
Etapa 3: Implementación y prueba de TEST 1	RH: Cristian Lopez (Rol: Diseño y testeo)	30/06/21	24/08/21

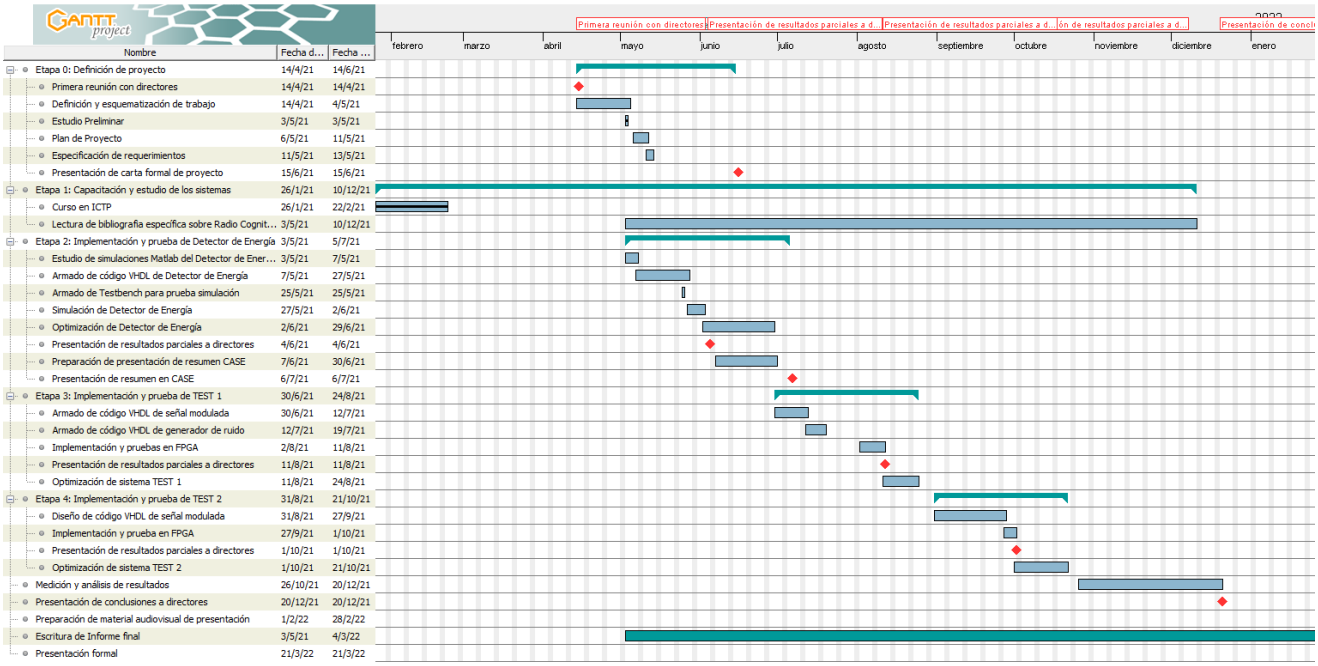
Armado de código VHDL de señal modulada	Software: Vivado	30/06/21	12/07/21
Armado de código VHDL de generador de ruido	Software: Vivado	12/07/21	19/07/21
Implementación y prueba en FPGA	FPGA: Xilinx Zedboard	02/08/21	11/08/21
Presentación de resultados parciales a directores		11/08/21	11/08/21
Optimización del sistema TEST 1	Software: Vivado	11/08/21	24/08/21
Etapa 4: Implementación y prueba de TEST 2	RH: Cristian Lopez (Rol: Diseño y testeo)	31/08/21	21/10/21
Diseño de código VHDL de señal modulada	Software: Vivado	31/08/21	27/09/21
Implementación y prueba en FPGA	FPGA: Xilinx Zedboard	27/09/21	01/10/21
Presentación de resultados parciales a directores		01/10/21	01/10/21
Optimización de sistema TEST 2	Software: Vivado	01/10/21	21/10/21
Medición y análisis de resultados	RH: Cristian Lopez (Rol: Testeo, análisis)	26/10/21	20/12/21
Presentación de conclusiones a directores		20/12/21	20/12/21
Preparación de material audiovisual de presentación	RH: Cristian Lopez (Rol: Escritura de documentos)	01/02/22	28/02/22
Escritura de informe final	RH: Cristian Lopez (Rol: Escritura de documentos)	03/05/21	04/03/22
Presentación formal		21/03/22	21/03/22

*En rojo son marcados los hitos del proyecto.

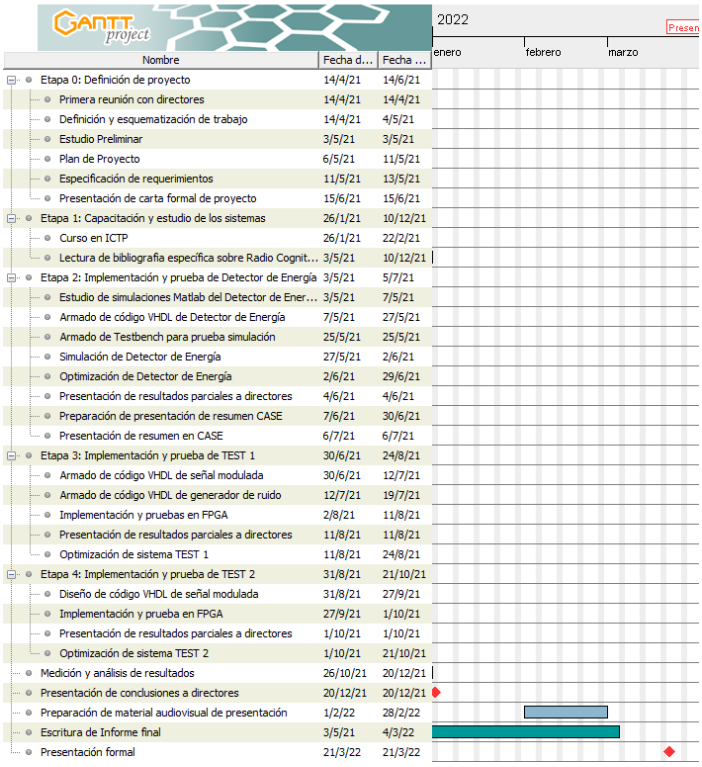
**RH: Recurso humano

Al tratarse de un proyecto de investigación y desarrollo cabe destacar que es posible que la denominada “Etapa 1” del proyecto es independiente de las demás etapas y abarca todo el proceso del proyecto. En tanto que para las etapas 2, 3 y 4 es importante que se cumplan en el orden establecido, aunque siempre se tiene en cuenta la posibilidad de realizar mejoras o cambios en una etapa anterior habiendo pasado a la siguiente.

Diagrama de Gantt



Nombre	Fecha d...	Fecha ...
Etapa 0: Definición de proyecto	14/4/21	14/6/21
Primera reunión con directores	14/4/21	14/4/21
Definición y esquematización de trabajo	14/4/21	4/5/21
Estudio Preliminar	3/5/21	3/5/21
Plan de Proyecto	6/5/21	11/5/21
Especificación de requerimientos	11/5/21	13/5/21
Presentación de carta formal de proyecto	15/6/21	15/6/21
Etapa 1: Capacitación y estudio de los sistemas	26/1/21	10/12/21
Curso en ICTP	26/1/21	22/2/21
Lectura de bibliografía específica sobre Radio Cognit...	3/5/21	10/12/21
Etapa 2: Implementación y prueba de Detector de Energía	3/5/21	5/7/21
Estudio de simulaciones Matlab del Detector de Ener...	3/5/21	7/5/21
Armado de código VHDL de Detector de Energía	7/5/21	27/5/21
Armado de Testbench para prueba simulación	25/5/21	25/5/21
Simulación de Detector de Energía	27/5/21	2/6/21
Optimización de Detector de Energía	2/6/21	29/6/21
Presentación de resultados parciales a directores	4/6/21	4/6/21
Preparación de presentación de resumen CASE	7/6/21	30/6/21
Presentación de resumen en CASE	6/7/21	6/7/21
Etapa 3: Implementación y prueba de TEST 1	30/6/21	24/8/21
Armado de código VHDL de señal modulada	30/6/21	12/7/21
Armado de código VHDL de generador de ruido	12/7/21	19/7/21
Implementación y pruebas en FPGA	2/8/21	11/8/21
Presentación de resultados parciales a directores	11/8/21	11/8/21
Optimización de sistema TEST 1	11/8/21	24/8/21
Etapa 4: Implementación y prueba de TEST 2	31/8/21	21/10/21
Diseño de código VHDL de señal modulada	31/8/21	27/9/21
Implementación y prueba en FPGA	27/9/21	1/10/21
Presentación de resultados parciales a directores	1/10/21	1/10/21
Optimización de sistema TEST 2	1/10/21	21/10/21
Medición y análisis de resultados	26/10/21	20/12/21
Presentación de conclusiones a directores	20/12/21	20/12/21
Preparación de material audiovisual de presentación	1/2/22	28/2/22
Escritura de Informe final	3/5/21	4/3/22
Presentación formal	21/3/22	21/3/22



Nombre	Fecha d...	Fecha ...
Etapa 0: Definición de proyecto	14/4/21	14/6/21
Primera reunión con directores	14/4/21	14/4/21
Definición y esquematización de trabajo	14/4/21	4/5/21
Estudio Preliminar	3/5/21	3/5/21
Plan de Proyecto	6/5/21	11/5/21
Especificación de requerimientos	11/5/21	13/5/21
Presentación de carta formal de proyecto	15/6/21	15/6/21
Etapa 1: Capacitación y estudio de los sistemas	26/1/21	10/12/21
Curso en ICTP	26/1/21	22/2/21
Lectura de bibliografía específica sobre Radio Cognit...	3/5/21	10/12/21
Etapa 2: Implementación y prueba de Detector de Energía	3/5/21	5/7/21
Estudio de simulaciones Matlab del Detector de Ener...	3/5/21	7/5/21
Armado de código VHDL de Detector de Energía	7/5/21	27/5/21
Armado de Testbench para prueba simulación	25/5/21	25/5/21
Simulación de Detector de Energía	27/5/21	2/6/21
Optimización de Detector de Energía	2/6/21	29/6/21
Presentación de resultados parciales a directores	4/6/21	4/6/21
Preparación de presentación de resumen CASE	7/6/21	30/6/21
Presentación de resumen en CASE	6/7/21	6/7/21
Etapa 3: Implementación y prueba de TEST 1	30/6/21	24/8/21
Armado de código VHDL de señal modulada	30/6/21	12/7/21
Armado de código VHDL de generador de ruido	12/7/21	19/7/21
Implementación y pruebas en FPGA	2/8/21	11/8/21
Presentación de resultados parciales a directores	11/8/21	11/8/21
Optimización de sistema TEST 1	11/8/21	24/8/21
Etapa 4: Implementación y prueba de TEST 2	31/8/21	21/10/21
Diseño de código VHDL de señal modulada	31/8/21	27/9/21
Implementación y prueba en FPGA	27/9/21	1/10/21
Presentación de resultados parciales a directores	1/10/21	1/10/21
Optimización de sistema TEST 2	1/10/21	21/10/21
Medición y análisis de resultados	26/10/21	20/12/21
Presentación de conclusiones a directores	20/12/21	20/12/21
Preparación de material audiovisual de presentación	1/2/22	28/2/22
Escritura de Informe final	3/5/21	4/3/22
Presentación formal	21/3/22	21/3/22

Riesgos

Nombre	Probabilidad	Severidad	Plan de respuesta	Prioridad
Dificultad en implementación de cálculo de nivel de comparación.	**Media**	**Alta**	Búsqueda de bibliografía específica.	1
Baja disponibilidad de placa Zedboard para pruebas	**Baja**	**Media**	Pruebas y conclusiones mediante simulación.	2
Falla del sistema implementado	**Media**	**Alta**	Probar el sistema por etapas.	1
Falla en generadores implementados para TEST 1 y TEST 2	**Media**	**Media**	Rever códigos y simulaciones.	3

Los riesgos que se identifican corresponden en gran medida a la etapa de implementación del diseño realizado. En este sentido tiene mayor relación con la arista de desarrollo del proyecto más que con la de investigación.

La severidad de estos riesgos se adjudica en base a la dificultad para la búsqueda de una solución para dicho riesgo. Luego, una severidad ALTA corresponde a un riesgo el cuál se prevé de una complejidad tal como para que la búsqueda de una solución a dicho problema sea relevante. Una severidad MEDIA refiere a los riesgos que no conllevan el detenimiento del proyecto para la búsqueda de su solución, sino que puede ejecutarse el plan de respuesta mientras se avanza en otras tareas del mismo.

Apéndice B

Especificación de requerimientos

Fecha	Versión	Descripción	Autor/a
1/7/2021	1.0	Versión inicial	Lopez, Cristian Hernán
17/8/2021	2.0	Segunda versión	Lopez, Cristian Hernán

B.1. Introducción

Este documento corresponde a la definición del tema de investigación del Proyecto "Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva", que se desarrolla en el marco del trabajo final de grado y la beca de investigación para estudiante avanzado otorgada por la Universidad Nacional de Mar del Plata. Se realiza en conjunto con el grupo de investigación Mecánica estadística y Sistemas no lineales del Laboratorio de Sistemas Caóticos perteneciente al ICYTE (Conicet) y la Universidad Nacional de Mar del Plata.

B.2. Proyecto

B.2.1. Personal involucrado

Nombre	Lopez, Cristian Hernán
Rol	Encargado del proyecto
Categoría profesional	Estudiante Avanzado
Responsabilidad	Desarrollo, implementación y testeo de algoritmos.
Información de contacto	contacto cristian.lopez@alumnos.fi.mdp.edu.ar

Nombre	De Micco, Luciana
Rol	Director del proyecto
Categoría profesional	Doctora en Ingeniería orientación Electrónica
Responsabilidad	Asistencia técnica en desarrollo e implementación.
Información de contacto	contacto ldemicco@fi.mdp.edu.ar

Nombre	Antonelli, Maximiliano
Rol	Co-Director del proyecto
Categoría profesional	Doctor en Ingeniería orientación Electrónica
Responsabilidad	Asistencia técnica en desarrollo e implementación.
Información de contacto	contacto maxanto@fi.mdp.edu.ar

B.2.2. Definiciones, acrónimos y abreviaturas

Nombre	Descripción
RF	Requerimiento funcional
RC	Radio Cognitiva
DE	Detector de Energía
SDR	Radio Definida por Software
FPGA	Arreglo de Compuertas Lógicas Programables en Campo
LSC	Laboratorio de Sistemas Caóticos
P_{FA}	Probabilidad de Falsa Alarma
N	Número de muestras de ruido
ROM	Memoria de solo lectura
IP Core	Núcleo de propiedad intelectual
RTL	Nivel de transferencia de registros
PS	Sistema de procesamiento
PL	Lógica programable
UART	Transmisor Receptor Asincrónico Universal
ICTP	Centro Internacional de Física Teórica
Pd	Probabilidad de detección

B.2.3. Referencias

Título del documento
J. Mitola, “Cognitive radio,” Ph.D. dissertation, Institutionen for telein- ¨ formatik, 2000.
M. Lopez-Ben ´ ´ıtez and F. Casadevall, “Improved energy detection spectrum sensing for cognitive radio,” IET communications, vol. 6, no. 8, pp. 785–796, 2012.
M. Z. Alom, T. K. Godder, M. N. Morshed, and A. Maali, “Enhanced spectrum sensing based on energy detection in cognitive radio network using adaptive threshold,” in 2017 International Conference on Networking, Systems and Security (NSysS). IEEE, 2017, pp. 138–143.
D. M. M. Plata and A. G. A. Re ´ atiga, “Evaluation of energy detection for ´ spectrum sensing based on the dynamic selection of detection-threshold,” Procedia Engineering, vol. 35, pp. 135–143, 2012.
P. V. Yadav, “An efficient hardware implementation of an energy detection-based spectral estimator for cognitive radios,” Ph.D. dissertation, San Diego State University, 2016.
C. D. Brown and H. T. Davis, “Receiver operating characteristics curves and related decision measures: A tutorial,” Chemometrics and Intelligent Laboratory Systems, vol. 80, no. 1, pp. 24 – 38, 2006.
H. V. Poor, An introduction to signal detection and estimation. Springer Science & Business Media, 2013.

B.2.4. Resumen

Las redes de Radio Cognitiva (RC) permiten una utilización eficiente del espectro de potencias. Un aspecto clave para su buen funcionamiento es el sensado espectral de los canales. El detector de energía es el método más usado para esta tarea por las ventajas que presenta, sin embargo, su efectividad se degrada rápidamente cuando

la potencia de ruido en el canal no es bien estimada. Este trabajo de investigación y desarrollo pretende presentar una implementación real de este sensor y estudiar las consideraciones necesarias para una buena estimación del ruido. Se presentará un informe detallado acerca de los recursos empleados y resultados de simulación y posterior implementación en una FPGA. Además, se implementarán distintos bancos de pruebas que logren evaluar el método investigado para su posterior comparación con los nuevos métodos que se están desarrollando en el laboratorio.

B.3. Problema de investigación

B.3.1. Surgimiento de la idea de investigación

Ante la saturación de los canales de comunicaciones debida al aumento acelerado de aplicaciones inalámbricas surge la técnica de Radio Cognitiva. Esta técnica permite a usuarios sin licencia (usuarios secundarios) utilizar temporalmente bandas del espectro con licencia, mientras los usuarios con licencia (usuarios primarios) estén inactivos. De aquí surge la necesidad de algoritmos y métodos de detección para sondear el espectro de radio y determinar su estado (ocupado o libre). Una de las técnicas utilizadas en la actualidad es la llamada "detección de energía", si bien se trata de una de las técnicas más comunes con algunas desventajas, surge la necesidad de su implementación en FPGA para su posterior comparación con los algoritmos que se encuentran desarrollando actualmente en el grupo de investigación de Mecánica estadística y Sistemas no lineales del Laboratorio de Sistemas Caóticos perteneciente al ICYTE (Conicet). Este paso es imprescindible para la evaluación de los nuevos algoritmos, tanto en determinar la máxima frecuencia de operación, los recursos necesarios, y la performance presentada.

El grupo de Sistemas Caóticos en el que se está inserto como becario trabaja en forma conjunta con el Laboratorio de Comunicaciones perteneciente al mismo instituto y tienen el proyecto en común "Diseño y desarrollo de sistemas de comunicaciones basados en Software Defined Radio (SDR)" (PIP 2017-2019). Actualmente se encuentran desarrollando nuevos algoritmos de sensado, se busca que sean rápidos, adaptativos y fiables, y estén basados en cuantificadores provenientes de la teoría de la información (entropía, complejidad, desequilibrio, medidas de Fischer, etc.) y de herramientas de análisis de sistemas no lineales (dimensión de correlación, exponentes de Lyapunov, etc).

B.3.2. Preguntas de la investigación

Las preguntas de la investigación se basan en la caracterización de la técnica de sensado espectral por DE. Se espera obtener respuesta a las siguientes preguntas:

- ¿Cuántos recursos son necesarios para implementar esta técnica?
- ¿Cuál es la frecuencia máxima posible del algoritmo?
- ¿Cuál es la latencia de la implementación de la técnica?
- ¿Qué debería mejorarse en el desarrollo de una nueva técnica de sensado espectral?

B.3.3. Objetivos de la investigación

Esta investigación tiene por objetivo la caracterización de la técnica de sensado espectral por DE. Para ello, en primera instancia, se desarrollará una implementación del sistema en FPGA y se simulará el mismo. Luego se desarrollarán otras dos implementaciones correspondientes a bancos de prueba para el sistema implementado anteriormente. Por último, se tomarán mediciones de parámetros tales como recursos necesarios, la frecuencia máxima de operación permitida, la latencia, entre otros en el circuito implementado en una placa ZedBoard de la empresa Xilinx. Estos datos serán informados y entregados al LSC para su posterior comparación con nuevos algoritmos desarrollados en el mismo.

B.3.4. Justificación y viabilidad de la investigación

La implementación en hardware es un paso crítico para la evaluación de los nuevos algoritmos propuestos. Contar con los sistemas en hardware permitirá evaluar la performance real de los mismos, que, por ahora y como un primer paso, se realizan en forma simulada. También permitirá evaluar los recursos necesarios, la frecuencia máxima de operación permitida, la latencia, entre otros.

B.3.5. Enfoque

La investigación se enfoca en caracterizar la técnica de sensado espectral por DE. Se tienen como referencias algunos trabajos realizados en distintas universidades del mundo acerca de la simulación del algoritmo. Es decir que ya se conoce como funciona teóricamente la técnica, y que resultados se esperan obtener. Sin embargo, no se encuentra mucha información acerca de su implementación y prueba física. El enfoque de la investigación se centra en dicha implementación así como en la extracción de conclusiones de las pruebas realizas sobre el sistema físico.

B.3.6. Alcance

Se busca implementar en una placa Zedboard de la empresa Xilinx, el algoritmo de sensado de espectro por detección de energía. Se requiere que el sistema desarrollado permita variar los siguientes parámetros:

- N (Número de muestras de la señal, por ejemplo N=16, 32, 64,128)
- Precisión empleada cálculos internos (cantidad de bits y arquitectura empleada).
- Del cálculo de nivel de comparación $zth = (Q^{-1}_{(P_{FA})}\sqrt{2N} + N)\sigma^2_w$:
  - Probabilidad de falsa alarma permitido (P_{FA}) empleado para el cálculo del umbral.
  - Número de muestras empleadas para la estimación de ruido (N_{th}).

Para poder realizar múltiples pruebas se requiere que la salida “Canal libre/ocupado” se implemente como un contador que sume la cantidad de veces que se detecta el canal ocupado.

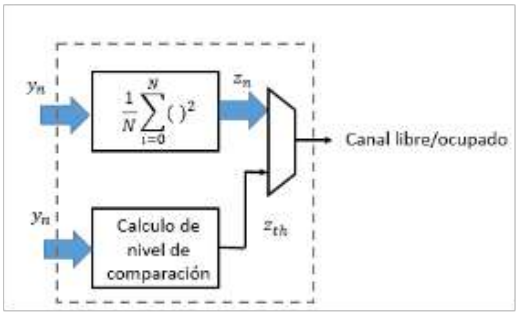


Figura B.1: Diagrama general

También se requiere que se implementen los bloques que permitan verificar el funcionamiento, y realizar los siguientes TESTs:

- TEST 1: Primera aproximación: Se considera a la señal modulada muestreada como una variable aleatoria Gaussiana.

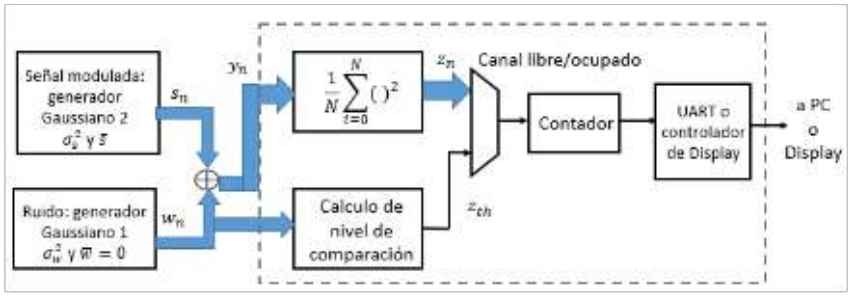


Figura B.2: Test 1

- TEST 2: Segunda aproximación: Se simula la salida del ADC con un generador de señal modulada en BPSK más ruido Gaussiano.

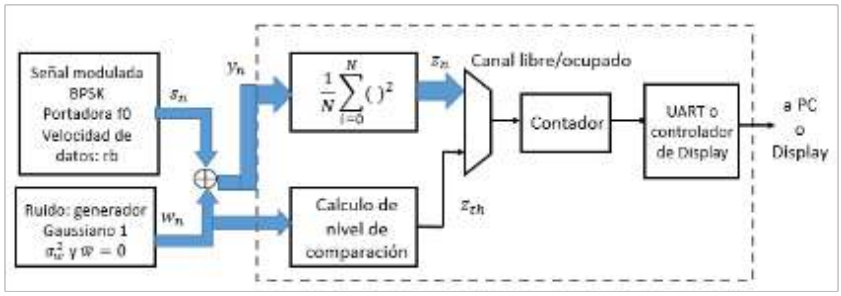


Figura B.3: Test 2

Apéndice C

Especificación funcional

Fecha	Versión	Descripción	Autor/a
05/09/2022	1.1	Corrección de ambigüedades en requerimientos funcionales.	Lopez, Cristian Hernán
17/05/2022	1.0	Versión inicial	Lopez, Cristian Hernán

C.1. Introducción

Este documento corresponde a la Especificación Funcional para la solución "Implementación de técnica por Detección de Energía". Esta especificación se ha estructurado basándose en la información mencionada en el documento de Especificación de Requerimientos (ER) ID "Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva".

C.1.1. Propósito del documento

El presente documento tiene como propósito proveer información detallada de cómo funcionará el sistema, cuáles serán sus comportamientos deseados y cómo se deberá construir, con base en los requerimientos anteriormente definidos en la ER.

Está dirigido a los directores del proyecto y demás integrantes del grupo de investigación de Mecánica estadística y Sistemas no lineales del Laboratorio de Sistemas Caóticos del ICYTE (Conicet), quienes utilizarán la solución como parte de su investigación para la comparación de esta técnica con otras desarrolladas en el marco de otras investigaciones.

C.1.2. Alcance del proyecto

Se busca implementar en una placa Zedboard de la empresa Xilinx, el algoritmo de sensado de espectro por detección de energía. Se requiere que el sistema desarrollado permita variar los siguientes parámetros:

- N (Número de muestras de la señal, por ejemplo N=16, 32, 64,128)
- Precisión empleada cálculos internos (cantidad de bits y arquitectura empleada).
- Del cálculo de nivel de comparación $zth = (Q^{-1}_{(P_{FA})}\sqrt{2N} + N)\sigma^2_w$:
  - Probabilidad de falsa alarma permitido (P_{FA}) empleado para el cálculo del umbral.
  - Número de muestras empleadas para la estimación de ruido (N_{th}).

Para poder realizar múltiples pruebas se requiere que la salida "Canal libre/ocupado" se implemente como un contador que sume la cantidad de veces que se detecta el canal ocupado.

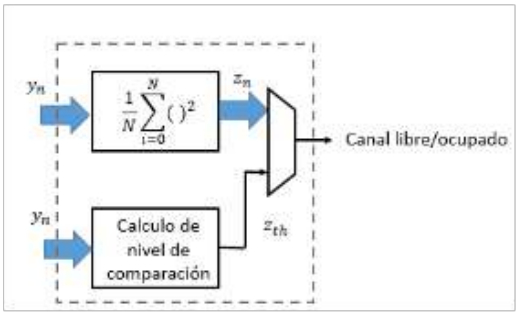


Figura C.1: Diagrama general

También se requiere que se implementen los bloques que permitan verificar el funcionamiento, y realizar los siguientes TESTs:

- TEST 1: Primera aproximación: Se considera a la señal modulada muestreada como una variable aleatoria Gaussiana.

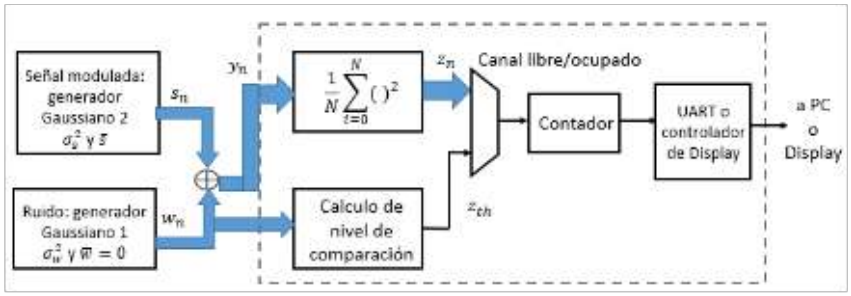


Figura C.2: Test 1

- TEST 2: Segunda aproximación: Se simula la salida del ADC con un generador de señal modulada en BPSK más ruido Gaussiano.

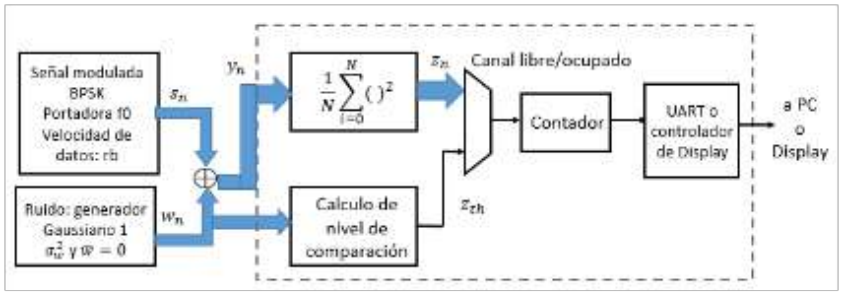


Figura C.3: Test 2

C.1.3. Personal involucrado

Nombre	Lopez, Cristian Hernán
Rol	Encargado del proyecto
Categoría profesional	Estudiante Avanzado
Responsabilidad	Desarrollo, implementación y testeo de algoritmos.
Información de contacto	contacto cristian.lopez@alumnos.fi.mdp.edu.ar

Nombre	De Micco, Luciana
Rol	Director del proyecto
Categoría profesional	Doctora en Ingeniería orientación Electrónica
Responsabilidad	Asistencia técnica en desarrollo e implementación.
Información de contacto	contacto ldemicco@fi.mdp.edu.ar

Nombre	Antonelli, Maximiliano
Rol	Co-Director del proyecto
Categoría profesional	Doctor en Ingeniería orientación Electrónica
Responsabilidad	Asistencia técnica en desarrollo e implementación.
Información de contacto	contacto maxanto@fi.mdp.edu.ar

C.1.4. Definiciones, acrónimos y abreviaturas

Nombre	**Descripción**
RF	Requerimiento funcional
RC	Radio Cognitiva
DE	Detector de Energía
SDR	Radio Definida por Software
FPGA	Arreglo de Compuertas Lógicas Programables en Campo
LSC	Laboratorio de Sistemas Caóticos
P_{FA}	Probabilidad de Falsa Alarma
N	Número de muestras de ruido
ROM	Memoria de solo lectura
IP Core	Núcleo de propiedad intelectual
RTL	Nivel de transferencia de registros
PS	Sistema de procesamiento
PL	Lógica programable
UART	Transmisor Receptor Asincrónico Universal
ICTP	Centro Internacional de Física Teórica
Pd	Probabilidad de detección

C.1.5. Referencias

Título del documento
Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva – Especificación de requerimientos

C.1.6. Resumen

Las redes de Radio Cognitiva (RC) permiten una utilización eficiente del espectro de potencias. Un aspecto clave para su buen funcionamiento es el sensado espectral de los canales. El detector de energía es el método más usado para esta tarea por las ventajas que presenta, sin embargo, su efectividad se degrada rápidamente cuando la potencia de ruido en el canal no es bien estimada. Este trabajo de investigación y desarrollo pretende presentar una implementación real de este sensor y estudiar las consideraciones necesarias para una buena estimación del ruido. Se presentará

un informe detallado acerca de los recursos empleados y resultados de simulación y posterior implementación en una FPGA. Además, se implementarán distintos bancos de pruebas que logren evaluar el método investigado para su posterior comparación con los nuevos métodos que se están desarrollando en el laboratorio.

C.2. Descripción del dispositivo

El dispositivo consiste en un conjunto de cinco bloques orientados a cumplir con los requerimientos especificados en la ER como se muestra en el siguiente esquema.

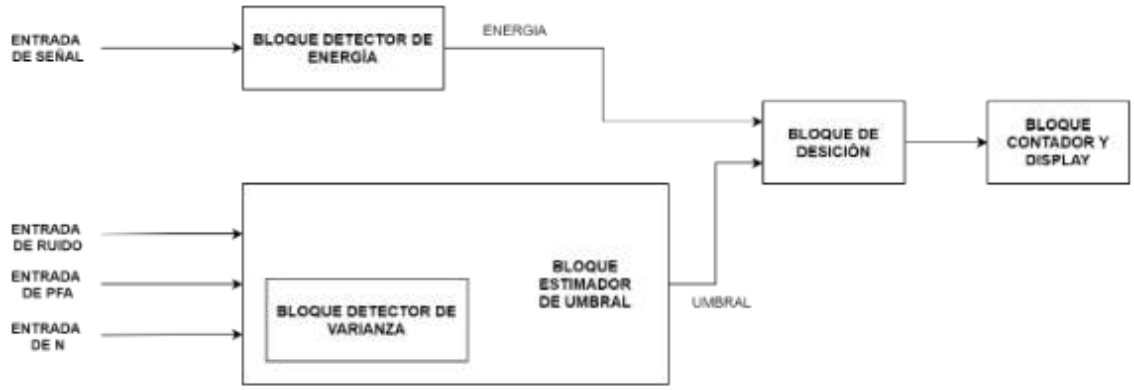


Figura C.4: Diagrama general

Debe, a partir de una señal de ruido tomada de un canal que se conozca libre, un número de muestras predeterminado (N) y una probabilidad de falsa alarma (P_{FA}) dada, calcular un valor umbral con el que se compara la energía de un canal de estado desconocido. Debe determinarse, a partir de una investigación que también forma parte de producto entregable, la cantidad de muestras (N) óptimas para una buena estimación de la potencia de ruido (varianza) utilizada en el cálculo del umbral.

La energía del canal que se quiere sensar se debe detectar y calcular a partir de una cierta cantidad de muestras a determinar para una buena detección.

Si la energía del canal sensado es mayor que el umbral se entiende que este está ocupado y se envía un pulso al contador para realizar la cuenta. Luego de un número considerable de corridas (1000 en principio), se debe mostrar el número de veces que se detectó el canal ocupado. Se debe entregar al Laboratorio de Sistemas Caóticos de la Facultad de Ingeniería:

- Diseño del dispositivo en lenguaje VHDL
- Diseño de simulación del dispositivo de System Generator – Simulink
- Investigación
- Hoja de datos con especificaciones y resultados de pruebas

C.2.1. Bloque detector de energía

Este bloque es el encargado de recibir muestras de la señal sensada del medio y calcular su energía. Para ello debe realizar la siguiente operación:

$$\frac{1}{N}\sum_{n=0}^{N-1} y(n)^2 \tag{C.1}$$

En concreto, este bloque calcula el promedio de la energía de cada una de las muestras de señal sensadas (yn). Para ello debe elevar al cuadrado el valor de la muestra, acumular una cierta cantidad (N) y luego dividir el resultado por ese valor.

C.2.2. Bloque estimador de umbral

El bloque encargado de estimar el umbral debe realizar la siguiente operación:

$$z_{th} = \left(\frac{Q^{-1}_{(P_{FA})}}{\sqrt{N}} + 1\right)\sigma_w^2 \tag{C.2}$$

Donde σ_w^2 es el resultado del bloque detector de varianza que se encuentra en su interior, explicado en detalle en la siguiente sección del documento. El resto de los parámetros de la operación son entradas al bloque ingresadas por el usuario del dispositivo. Entre las entradas se encuentra la Probabilidad de Falsa Alarma requerida (P_{FA}) y el número de muestras de ruido (N) que se tomarán para realizar la estimación del umbral. La función $Q^{-1}_{(P_{FA})}$ representa la inversa de la función Q Gaussiana, cuyos valores se encuentran tabulados en diversas bibliografías.

C.2.3. Bloque detector de varianza

Este bloque es el encargado de estimar la potencia de ruido del canal, para ello debe realizar el cálculo de la varianza de las muestras de ruidos, operación similar a la del detector de energía. En este caso, la operación a realizar es la siguiente:

$$\frac{1}{N_{th}}\sum_{n=0}^{N_{th}-1} (n(n) - \bar{n(n)})^2 \tag{C.3}$$

Donde N_{th} se refiere a la cantidad de muestras de ruido, $n(n)$ cada muestra de ruido y $\bar{n(n)}$ la media de las muestras de ruido. Como se trata de ruido aleatorio blanco Gaussiano se puede simplificar dicha expresión, asumiendo que el valor medio de las muestras es nulo, luego la operación que debe realizar el bloque es:

$$\frac{1}{N_{th}}\sum_{n=0}^{N_{th}-1} n(n)^2 \tag{C.4}$$

De esta forma el bloque detector de varianza es similar al bloque detector de energía para el caso particular de trabajar con ruido blanco Gaussiano.

C.2.4. Bloque de decisión

Este bloque recibe el valor del cálculo de la energía y del estimador de umbral y los compara. En caso que la energía sea mayor al nivel de umbral se trata de un canal ocupado y por lo tanto debe entregar un “1” lógico a la salida. En caso que

la energía sea menor al nivel de umbral se trata de un canal detectado como libre y por lo tanto debe entregar un "0" lógico a la salida.

C.2.5. Bloque contador y display

El último bloque comprende la interfaz con el usuario, es necesario que este bloque este diseñado para contar la cantidad de veces que se a detectado el canal como ocupado y mostrarlo a través de un display o enviarlo por un puerto serie a una PC que realice el procesamiento de dicha información. Sabiendo la cantidad de veces que el canal ha sido detectado como ocupado y la cantidad total de muestras evaluadas se puede tomar una decisión certera sobre el estado real del canal, y con ello saber si es posible o no su utilización temporaria.

C.3. Especificaciones funcionales

C.3.1. RF01: Variación de número de muestras de ruido

El dispositivo debe admitir la variación de la cantidad de muestras de ruido que ingresan al estimador de umbral. Para los estudios que se necesitan realizar es preciso que pueda elegirse que el dispositivo adquiera una cantidad de muestras dentro del rango de $8(2^3)$ a $4096(2^{12})$.

C.3.2. RF02: Variación de la P_{FA}

Se debe admitir la variación de la probabilidad de falsa alarma con la que se trabaja. La variación debe ser en el rango de 0.01 a 0.5, es necesario que dicha variación sea en pasos de 0.005.

C.3.3. RF03: Precisiones para los cálculos internos

Se debe diseñar cada bloque con una cantidad de bits variable para las señales de entrada y salida, para de esta manera poder manejar la precisión de los cálculos a gusto del usuario. Teniendo en cuenta que mayor precisión implica mayor tiempo de procesamiento y área de la FPGA utilizada, la longitud de las palabras de entrada y salida no debe superar los 12 bits. Por otro lado, para que el usuario tenga mayor control sobre la precisión de los cálculos es necesario el uso de un sistema numérico en punto fijo.

C.3.4. RF04: Interfaz de usuario

Se debe proporcionar una salida numérica mínimamente con la siguiente información:

- cantidad de corridas realizadas
- cantidad de veces detectadas
- probabilidad de detección (cantidad de veces detectadas/cantidad de corridas)

Para ello es necesario contar con una interfaz serie que se conecte a una PC y envíe los datos para que estos puedan ser observados mediante una lectura por terminal o bien un software de procesamiento ya disponible.

C.4. Requerimientos no funcionales

C.4.1. RNF01: Rendimiento del diseño

El diseño debe tener en cuenta la mínima latencia posible, si bien es preciso almacenar una cierta cantidad de muestras para calcular la energía en cada corrida, es necesario que el procesamiento de esas muestras sea lo más rápido posible (idealmente menor a 10ns, permitiendo el procesamiento de señales hasta 100 MHz). Disminuir la latencia de los componentes que realizan el procesamiento (o bien realizar el cálculo de la energía de la señal con la menor cantidad de componentes posible) es un requerimiento importante.

Apéndice D

Especificación técnica

Fecha	Versión	Descripción	Autor
17/10/2022	1.0	Versión Inicial	Lopez, Cristian Hernán

D.1. Introducción

Este documento corresponde a la Especificación Técnica para la solución "Implementación de técnica por Detección de Energía". Esta especificación se ha estructurado basándose en la información mencionada en el documento de Especificación Funcional (EF) ID "Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva".

D.1.1. Propósito del documento

Este documento tiene como finalidad describir cómo está compuesta la solución, definir el diseño de los distintos módulos y la conexión entre ellos. Para ello se utilizarán diagramas en bloques, diagramas de flujo y pseudocódigo para la correcta comprensión del trabajo a desarrollar.

Está dirigido a los directores del proyecto y demás integrantes del grupo de investigación de Mecánica estadística y Sistemas no lineales del Laboratorio de Sistemas Caóticos del ICYTE (Conicet-UNMDP), quienes utilizarán la solución como parte de su investigación para la comparación de esta técnica con otras desarrolladas en el marco de otras investigaciones.

D.1.2. Alcance del proyecto

Se busca implementar en una placa Zedboard de la empresa Xilinx, el algoritmo de sensado de espectro por DE. Se requiere que el sistema desarrollado permita variar los siguientes parámetros:

- N (Número de muestras de la señal, por ejemplo N=16, 32, 64,128)
- Precisión empleada cálculos internos (cantidad de bits y arquitectura empleada).
- Del cálculo de nivel de comparación $z_{th} = (Q^{-1}_{(P_{FA})}\sqrt{2N} + N)\sigma^2_w$:
  - Probabilidad de falsa alarma permitido (P_{FA}) empleado para el cálculo del umbral.
  - Número de muestras empleadas para la estimación de ruido (N_{th}).

Para poder realizar múltiples pruebas se requiere que la salida "Canal libre/ocupado" se implemente como un contador que sume la cantidad de veces que se detecta el canal ocupado.

También se requiere que se implementen los bloques que permitan verificar el funcionamiento, y realizar los siguientes TESTs:

- TEST 1: Primera aproximación: Se considera a la señal modulada muestreada como una variable aleatoria Gaussiana.
- TEST 2: Segunda aproximación: Se simula la salida del ADC con un generador de señal modulada en BPSK más ruido Gaussiano.

D.1.3. Personal involucrado

Nombre	Lopez, Cristian Hernán
Rol	Encargado del proyecto
Categoría profesional	Estudiante Avanzado
Responsabilidad	Desarrollo, implementación y testeo de algoritmos.
Información de contacto	contacto cristian.lopez@alumnos.fi.mdp.edu.ar

Nombre	De Micco, Luciana
Rol	Director del proyecto
Categoría profesional	Doctora en Ingeniería orientación Electrónica
Responsabilidad	Asistencia técnica en desarrollo e implementación.
Información de contacto	contacto ldemicco@fi.mdp.edu.ar

Nombre	Antonelli, Maximiliano
Rol	Co-Director del proyecto
Categoría profesional	Doctor en Ingeniería orientación Electrónica
Responsabilidad	Asistencia técnica en desarrollo e implementación.
Información de contacto	contacto maxanto@fi.mdp.edu.ar

D.1.4. Definiciones, acrónimos y abreviaturas

Nombre	**Descripción**
RF	Requerimiento funcional
RC	Radio Cognitiva
DE	Detector de Energía
SDR	Radio Definida por Software
FPGA	Arreglo de Compuertas Lógicas Programables en Campo
LSC	Laboratorio de Sistemas Caóticos
P_{FA}	Probabilidad de Falsa Alarma
N	Número de muestras de ruido
ROM	Memoria de solo lectura
IP Core	Núcleo de propiedad intelectual
RTL	Nivel de transferencia de registros
PS	Sistema de procesamiento
PL	Lógica programable
UART	Transmisor Receptor Asincrónico Universal
ICTP	Centro Internacional de Física Teórica
Pd	Probabilidad de detección

D.1.5. Referencias

Título del documento
Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva Especificación funcional
Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva Especificación de requerimientos
Divider Generator v5.1 - LogiCORE IP Product Guide - Xilinx
CORDIC v6.0 - LogiCORE IP Product Guide - Xilinx
The ComBlock User Guide - Rodrigo Alejandro Melo

D.1.6. Resumen del documento

El documento se divide en siete secciones. La primera de ellas se corresponde con la introducción al mismo. En segundo lugar se encuentra una visión general de la solución en donde se puede observar un diagrama general del diseño con todos los módulos y componentes que lo integran, además de una explicación de los componentes que interconectan módulos entre sí.

A partir de la tercer sección en adelante se comienzan a repasar uno por uno los módulos que componen la solución. Arrancando con una explicación detallada del Módulo Detector de Energía. En la cuarta sección se encuentra un detalle del Módulo Estimador de Umbral, que a su vez se divide en tres subsecciones correspondientes a dos componentes importantes en el diseño y por otro lado el Submódulo Detector de Potencia.

La quinta sección corresponde a una explicación detallada del módulo de decisión. Por otro lado la sexta sección corresponde al módulo de comunicación y control, el cual muestra las interconexiones entre la lógica programable (PL) de la FPGA y el sistema de procesamiento (PS).

Por último, la séptima sección se basa en la interfaz de control y testeo. En esta se detalla un diagrama de flujo de la lógica para la interfaz de usuario y el script de testeo que se ejecuta por detrás.

D.2. Visión general de la solución

En la Figura D.1 se muestran los distintos componentes que integran la solución:

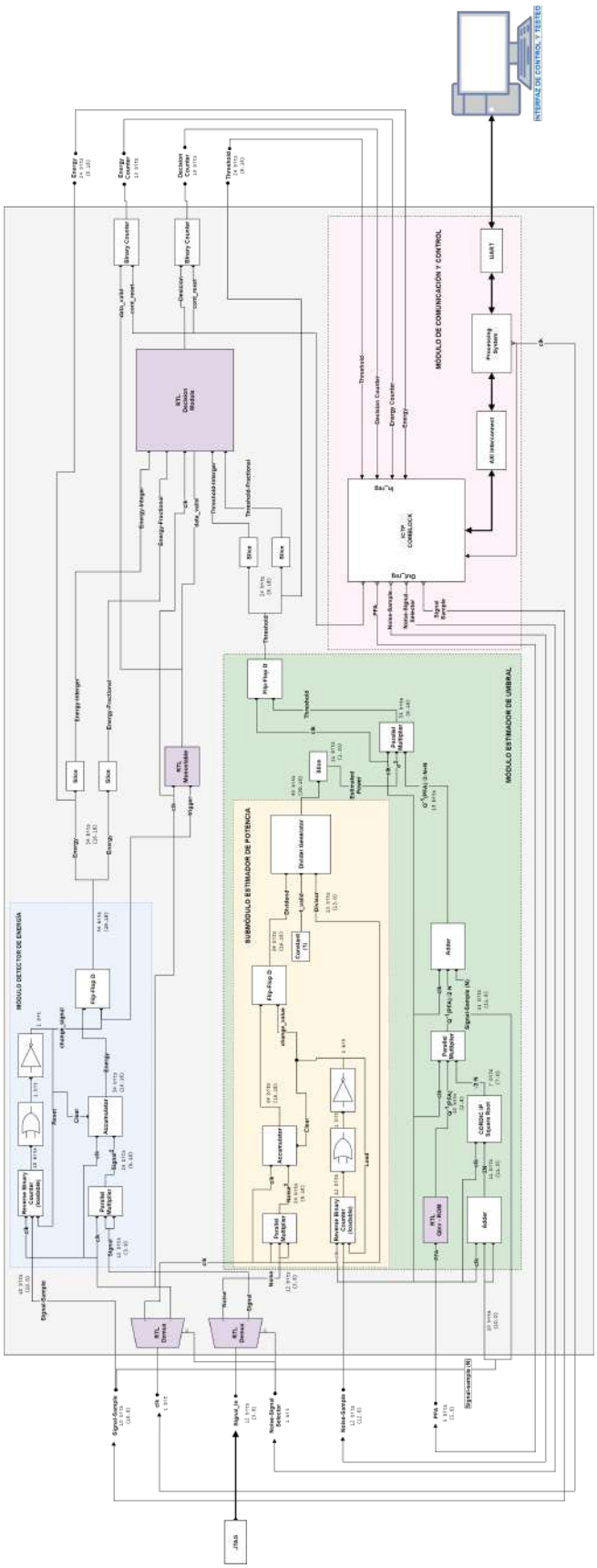


Figura D.1: Diagrama completo de la solución

En la Figura anterior, se pueden observar los distintos módulos y componentes que conforman la solución. Todos los componentes identificados en color violeta y con la sigla RTL (Nivel de transferencia de registros) corresponden a componentes diseñados exclusivamente para esta solución, mientras que los bloques identificados en blanco corresponden a componentes propietarios de Xilinx, que deberán ser correctamente configurados para los fines específicos como se explica en las siguientes secciones.

La solución se compone de cuatro módulos y un submódulo:

- Módulo Detector de Energía (ver sección D.3)
- Módulo Estimador de Umbral (ver sección D.4)
- Submódulo Detector de potencia (ver sección D.4.3)
- Módulo de decisión (ver sección D.5)
- Módulo de comunicación y control (ver sección D.6)

Para la comunicación entre los distintos módulos también contiene los siguientes componentes:

- **RTL DEMUX:** Este componente tiene como finalidad la derivación de las señales de reloj y entrada a los diferentes módulo en el momento en que sea necesario. La entrada de selección permite elegir entre la utilización de los módulos correspondientes al cálculo de la energía o los correspondientes al cálculo del umbral. Esta capacidad de elección se agregó ya que no es necesario utilizar todos los módulos a la vez, por lo que puede 'apagarse' una parte del circuito mientras no se utiliza para lograr eficiencia en la potencia requerida. En la sección D.2.1 se detalla el diseño VHDL de este componente.
- **Slice:** El objetivo de los rebanadores (slice) es el de obtener una cierta cantidad de bits de un bus de entrada. En el circuito se utilizan en diversas oportunidades, por lo general para extraer los bits correspondientes a la parte entera y fraccionaria de un resultado.
- **RTL Monostable:** Este componente tiene como finalidad obtener un pulso de una cierta cantidad de ciclos de reloj cuando le ingresa el flanco de finalización de cuenta del módulo **Detector de Energía**. Es decir, es el indicador de que existe un valor de energía disponible a la salida del módulo. La salida de este componente es útil por un lado como entrada al módulo de decisión para advertir que hay un valor de energía disponible y por otro como reloj del contador de salida de Energía. Este componente es necesario para obtener un pulso de mayor duración que el entregado por el módulo Detector de Energía. En la sección D.2.2 se detalla su diseño.

D.2.1. Diseño de DEMUX

Los demultiplexores tienen la finalidad de elegir hacia qué parte del circuito enviar la información. Se cuenta con dos de ellos a la entrada del circuito. Uno deriva la información relacionada con la señal de entrada, la cual puede ser, el ruido del canal destinado a calcular el umbral o la señal del canal que se quiere sensar. Por otro lado el segundo demux es el encargado de derivar la señal de reloj, hacia

los bloques encargados del cálculo del umbral o hacia los bloques encargados del cálculo de la energía.

Para el diseño de los demultiplexores se seguirá el pseudocódigo que se muestra a continuación:

```
1    out_1 <= Signal_In when Sel = '1';
2    out_0 <= Signal_In when Sel = '0';
```

D.2.2. Diseño de Monoestable

El circuito monoestable es el indicado para generar un pulso de una determinada longitud en el tiempo a partir de un flanco en la entrada proveniente, en este caso, del pequeño pulso generado al cumplirse las N muestras en el detector de energía. En la Figura D.2 se observa un esquema del componente y las formas de onda de entrada y salida que ejemplifican su funcionamiento.

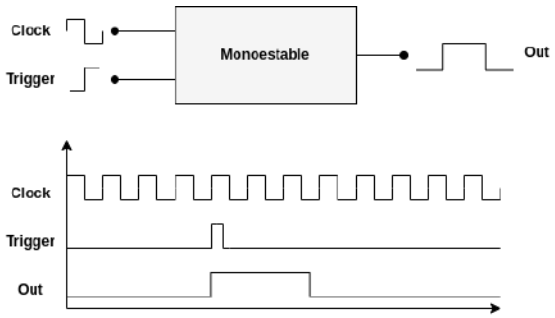


Figura D.2: Circuito monoestable

El componente se diseñará siguiendo el pseudocódigo que se encuentra a continuación.

```
1    if trigger = '1' then
2        count <= 75;
3    elsif rising_edge(clk) then
4        if count > 0 then
5            pulse <= '1';
6            count <= count - 1;
7        else
8        pulse <= '0';
9        end if;
10   end if;
```

D.3. Módulo Detector de Energía

La tarea principal de este módulo se basa en el cálculo de la energía de la señal. Para ello es necesario la síntesis de la siguiente ecuación:

$$\frac{1}{N}\sum_{n=0}^{N-1} y(n)^2 \tag{D.1}$$

Para la simplificación del circuito se decide eliminar la normalización por N. Esta normalización requiere la utilización de circuitos divisores en el diseño, los cuales es sabido que representan una dificultad circuital. Esta simplificación debe ser tenida

en cuenta en el módulo estimador de umbral, puesto que es necesario modificar la ecuación correspondiente a ese Módulo. Luego la ecuación que se sintetizará es:

$$\sum_{n=0}^{N-1} y(n)^2 \tag{D.2}$$

Donde N representa la cantidad de muestras de señal que se tomarán para realizar el cálculo. Esta cantidad de muestras es seteable en el rango de 0 a 1024. La justificación de esta elección se basa en que si bien es sabido que cuanto mayor cantidad de muestras se tomen, mejor es la precisión, luego del estudio mediante simulaciones se llegó a la conclusión que 1024 muestras es suficientemente representativa de la población para las pruebas necesarias.

En la Figura D.3 se muestra una vista detallada del módulo detector de energía:

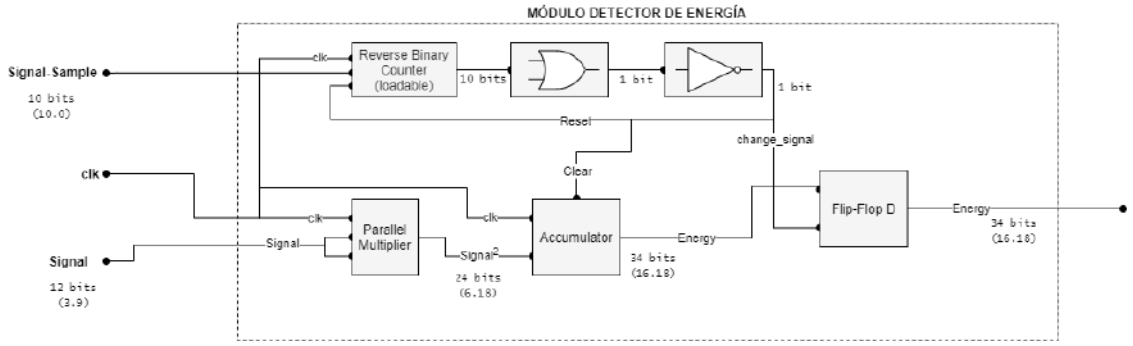


Figura D.3: Detector de Energía

Como se puede observar el módulo se compone de los siguientes bloques:

- **Reverse Binary Counter:** Contador binario reverso. Su función principal es contar la cantidad de muestras de señal que se están procesando para que, al llegar al número seteado, se resetéen el resto de los componentes.
- **Compuertas AND y NOT:** Su función principal es formatear la salida del contador cuando llega al valor cero, al llegar los 10 bits de salida del contador a cero, la salida de la compuerta AND fija un valor igual a cero de un bit. Luego, el negador convierte este valor en una señal lógica alta (1). El bit de salida de este bloque es el que activa el Flip Flop (FF) de la salida y los reset del módulo contador y acumulador para preparar el módulo para un nuevo set de datos. En la Figura D.4 se muestra un esquema del funcionamiento de este bloque.

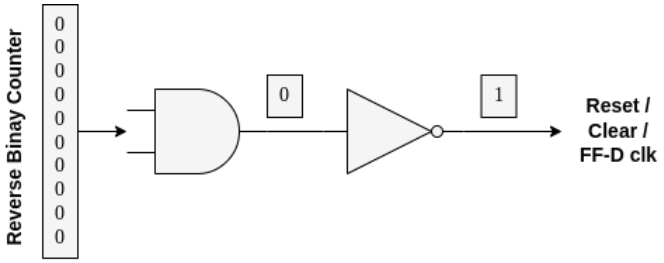


Figura D.4: Funcionamiento de bloque de compuertas AND-NOT

- **Parallel Multiplier:** Multiplicador paralelo. Este bloque realiza la operación de multiplicación de la señal por si misma, en otras palabras eleva al cuadrado la señal para obtener un valor de potencia.
- **Accumulator:** Acumulador. Su función principal es realizar la operación de suma acumulativa del valor de potencia de la señal. Al terminar de acumular las N muestras, se resetea con la señal de salida del bloque de compuertas AND y NOT.
- **Flip-Flop D:** La función principal de este bloque es fijar el valor de energía a la salida del módulo detector. De esta forma la salida del módulo solamente cambiará de valor luego de N muestras de señal. La señal de reloj de este bloque se encuentra conectada a la salida del bloque de compuertas AND y NOT.

D.4. Módulo Estimador de Umbral

La función principal del módulo Estimador de Umbral es la de calcular el valor con el que se comparará la energía calculada en el módulo **Detector de Energía**, para decidir si el canal se encuentra libre u ocupado. Esto lo realiza mediante la síntesis de la siguiente ecuación:

$$z_{th} = \left(\frac{Q^{-1}_{(P_{FA})}}{\sqrt{N}} + 1 \right) \sigma_w^2 \tag{D.3}$$

Donde Q^{-1} representa la inversa de la función Gaussiana acumulada, P_{FA} la probabilidad de falsa alarma, N el número de muestras de señal a evaluar y σ_w^2 la potencia de ruido calculada en el submódulo detector de potencia.

De igual manera que la ecuación D.1 fue simplificada para hacer más fácil la implementación del circuito, la ecuación D.3 también debe ser trabajada algebraicamente para que no se generen inconsistencias entre la energía calculada y el umbral. De esta manera, la ecuación simplificada para el umbral resulta:

$$zth = (Q^{-1}_{(P_{FA})}\sqrt{2N} + N)\sigma_w^2 \tag{D.4}$$

En la Figura D.5 se muestra una vista detalla del módulo Estimador de Umbral:

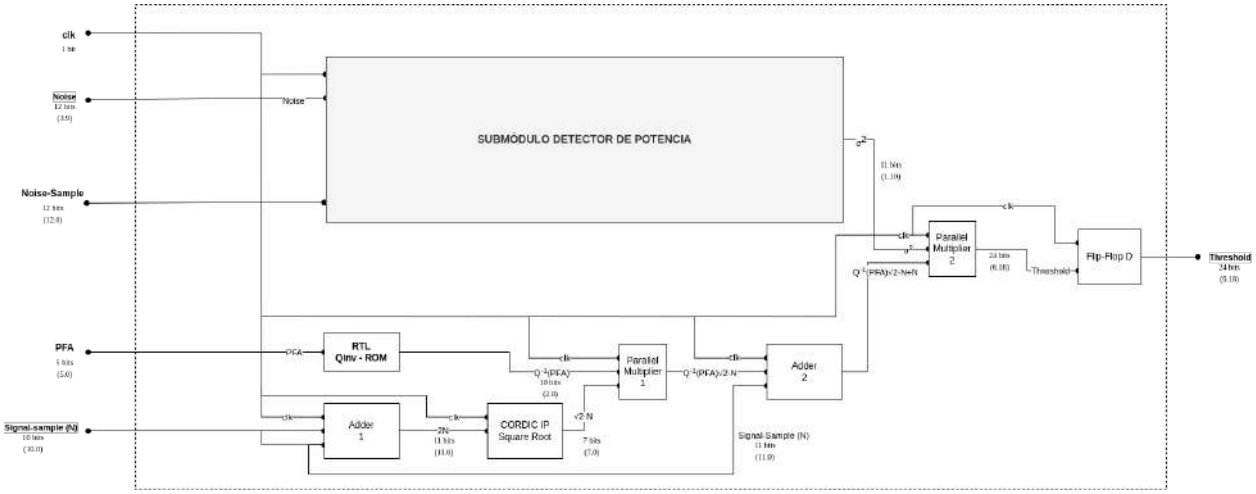


Figura D.5: Estimador de Umbral

Este módulo esta compuesto por los siguientes componentes:

- **RTL Qinv - ROM:** Este bloque es el encargado de realizar la cuenta $Q^{-1}_{(P_{FA})}$ correspondiente a la función Q Gaussiana acumulada inversa de la Probabilidad de Falsa Alarma (P_{FA}). Este bloque se diseñará en VHDL y corresponde a una memoria de solo lectura (ROM) en donde cada posición de memoria corresponde al resultado para cada P_{FA}. Se decide utilizar este diseño donde los resultados son tabulados y no uno en donde se realice la cuenta porque la función Q Gaussiana inversa representa una complejidad de diseño circuital que no tiene sentido abordar en el diseño integral que se está realizando. El detalle del diseño de este módulo se encuentra en la sección D.4.1.

- **Adder 1:** Sumador 1. Es el encargado de realizar la cuenta $2N$ y se sintetizará con un IP Core integrado de la plataforma de diseño Vivado. Se opta por realizar esta cuenta con un sumador y no con un multiplicador ya que realizar un suma circuitalmente tiene menor complejidad que realizar una operación de multiplicación.

- **CORDIC IP - Square Root:** Raiz Cuadrada. Este bloque tiene como tarea realizar la operación $\sqrt{2N}$. Para ello se utiliza un IP Core propietario llamado Cordic IP, el cual entre otra funciones posibles permite realizar la operación de raíz cuadrada. En la sección D.4.2 se detallan las especificaciones del núcleo propietario de Xilinx.

- **Parallel Multiplier 1:** Multiplicador paralelo 1. Este bloque tiene como objetivo realizar la operación $Q^{-1}_{(P_{FA})}\sqrt{2N}$. Para ello se utilizará un núcleo propietario multiplicador integrado en el software de diseño.

- **Adder 2:** Sumador 2. Es el bloque encargado de realizar la operación ($Q^{-1}_{(P_{FA})}\sqrt{2N}+N$). Para sintetizarlo se utilizará el mismo IP Core utilizado para en el bloque **Adder 1**.

- **Parallel Multiplier 2:** Multiplicador paralelo 2. Este bloque tiene como objetivo multiplicar la salida del sumador 2 (correspondiente al primer término de la ecuación para calcular el umbral) y la salida del **submódulo Estimador de Potencia** (correspondiente a la potencia de la señal de ruido). La salida de este bloque corresponde al umbral calculado.

- **Flip Flop D:** La función principal de este bloque es fijar el valor de umbral a la salida del módulo estimador de umbral. De esta forma se realiza un latcheo de la salida de importancia para obtener una mejora en la frecuencia máxima del diseño.

- **Submódulo detector de potencia:** La función principal de este submódulo es calcular la potencia de la señal de ruido que ingresa. Será detallado en la sección D.4.3.

D.4.1. Diseño de Qinv - ROM

Este componente corresponde a una memoria ROM diseñada en lenguaje VHDL, en la cual se guarden los resultados de la operación $Q^{-1}_{(P_{FA})}$ para valores de P_{FA} entre 0 y 0.5 en pasos de 0.02. Para obtener estos valores se utiliza la función de Matlab $qfuncinv(y)$, la cual devuelve el argumento de entrada de la función $Q_{()}$ para el que

el valor de salida de dicha función es y. En la Figura D.6 se muestra una gráfica de la función $Q_{()}$ donde para obtener el valor de la operación es necesario ingresar al gráfico por el eje de ordenadas y obtener el resultado del eje de abscisas.

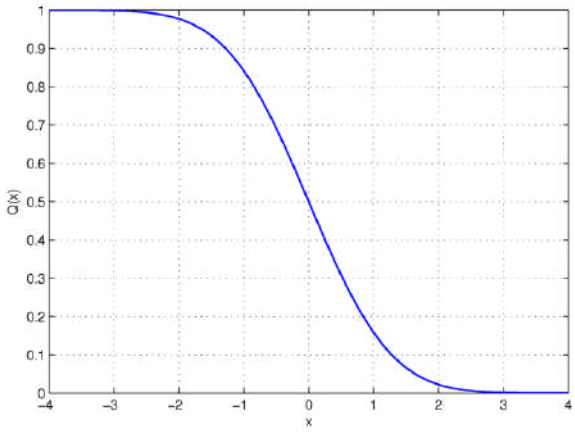


Figura D.6: Función Q

A continuación se presenta el pseudocódigo correspondiente a este componente:

```
    signal PFA_int: INTEGER RANGE 0 TO words-1;

    TYPE vector_array IS ARRAY (0 TO words-1) OF
        STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
        CONSTANT memory: vector_array := ("1001010011",
                                          "0111100001",
                                          "0110100101",
                                          "0101111001",
                                          "0101010111",
                                          "0100111001",
                                          "0100100000",
                                          "0100001001",
                                          "0011110100",
                                          "0011100000",
                                          "0011001110",
                                          "0010111101",
                                          "0010101100",
                                          "0010011100",
                                          "0010001101",
                                          "0001111110",
                                          "0001110000",
                                          "0001100010",
                                          "0001010100",
                                          "0001000111",
                                          "0000111010",
                                          "0000101101",
                                          "0000100000",
                                          "0000010011",
                                          "0000000110");
    BEGIN
       PFA_int <= to_integer(unsigned(PFA))-1;
       Q_inv <= memory(PFA_int);
```

En la tabla D.1 se indica el diseño de la memoria ROM, donde se especifica a qué valor corresponde cada posición.

ROM-POS	PFA	PFA-IN	Qinv	Qinv-ROM
0	0.01	00000	2.3263	1001010011
1	0.03	00001	1.8808	0111100001
2	0.05	00010	1.6449	0110100101
3	0.07	00011	1.4758	0101111001
4	0.09	00100	1.3408	0101010111
5	0.11	00101	1.2265	0100111001
6	0.13	00110	1.1264	0100100000
7	0.15	00111	1.0364	0100001001
8	0.17	01000	0.9542	0011110100
9	0.19	01001	0.8779	0011100000
10	0.21	01010	0.8064	0011001110
11	0.23	01011	0.7388	0010111101
12	0.25	01100	0.6745	0010101100
13	0.27	01101	0.6128	0010011100
14	0.29	01110	0.5534	0010001101
15	0.31	01111	0.4959	0001111110
16	0.33	10000	0.4399	0001110000
17	0.35	10001	0.3853	0001100010
18	0.37	10010	0.3319	0001010100
19	0.39	10011	0.2793	0001000111
20	0.41	10100	0.2275	0000111010
21	0.43	10101	0.1764	0000101101
22	0.45	10110	0.1257	0000100000
23	0.47	10111	0.0753	0000010011
24	0.49	11000	0.0251	0000000110

Tabla D.1: Diseño de memoria ROM.

D.4.2. Núcleo CORDIC

Este núcleo IP Xilinx LogiCORE implementa un algoritmo de computadora digital rotacional de coordenadas generalizadas (CORDIC). Entre otras funciones posee la posibilidad de realizar la operación de raiz cuadrada necesaria en el diseño del módulo **Estimador de Umbral**. En la Figura D.7 se observa un esquema de las entradas y salidas que posee el componente.

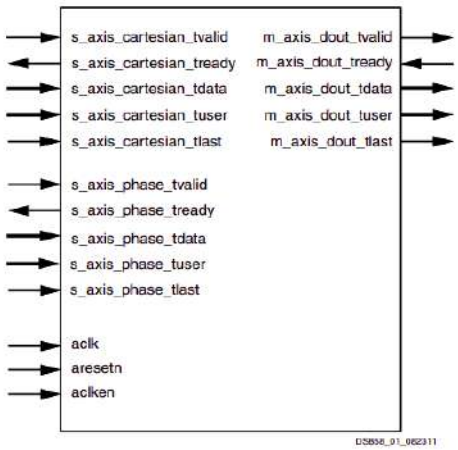


Figura D.7: Pin-out de CORDIC IP

Cuando se selecciona la configuración funcional de raíz cuadrada, se utiliza un algoritmo simplificado para calcular la raíz cuadrada positiva de la entrada. La entrada, X_{IN}, y la salida, X_{OUT}, son siempre positivas y ambas se expresan como fracciones sin signo o enteros sin signo. Para este caso particular se utilizará el formato de datos en entero sin signo, entonces X_{IN} se limita al rango: $0 \leq X_{IN} < 2^{Input_{Width}}$.

El bloque CORDIC decide automáticamente el ancho de salida según el ancho de entrada. Además, la función de rotación aproximada no es necesaria para el cálculo de la raíz cuadrada, por lo que está deshabilitada para esta aplicación en el componente. Como la entrada es de tipo entero, la salida es de forma entera pura con redondeo al techo (si es necesario).

D.4.3. Submódulo Detector de Potencia

La función principal de este módulo es la de calcular la potencia de la señal de ruido recibida, para ello es necesario realizar la siguiente operación circuitalmente:

$$\frac{1}{N_{th}} \sum_{n=0}^{N_{th}-1} (n(n) - \bar{n(n)})^2 \tag{D.5}$$

Donde N_{th} se refiere a la cantidad de muestras de ruido, $n(n)$ cada muestra de ruido y $\bar{n(n)}$ la media de las muestras de ruido. Sin embargo, como el ruido se asume Gaussiano, la media de las muestras de ruido se asume cero y por lo tanto se puede simplificar la ecuación de la siguiente manera:

$$\frac{1}{N_{th}} \sum_{n=0}^{N_{th}-1} n(n)^2 \tag{D.6}$$

Con esta simplificación la ecuación tiene un significativo parecido a la ecuación que sintetiza el módulo **Detector de Energía** por lo que los componentes que se utilizan son similares. En este caso no es posible la simplificación de la normalización por N_{th} ya que es sumemente importante para el cálculo de la potencia. En este caso se debe utilizar un bloque divisor, el cual es provisto en el catálogo de IP Cores del software Vivado.

Es importante para el estudio del sistema que el valor de la cantidad de muestras de ruido sea seteable externamente. Por lo tanto, se implementará una entrada de

12 bits (hasta 4096) para dejar a elección del usuario la cantidad de muestras de ruido que quiera utilizar para realizar el cálculo del umbral.

En la Figura D.8 se muestra una vista detalla del submódulo Detector de Potencia:

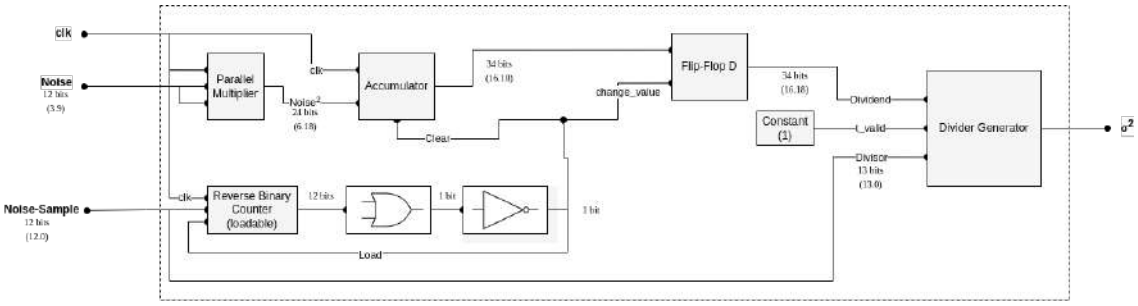


Figura D.8: Estimador de potencia

Este submódulo esta integrado por los siguientes componentes:

- **Reverse Binary Counter:** Contador binario reverso. Su función principal es contar la cantidad de muestras de señal de ruido que se están procesando para que al llegar al número seteado se resetéen el resto de los componentes.
- **Compuertas AND y NOT:** Su función principal es formatear la salida del contador cuando llega al valor cero, al llegar los 12 bits de salida del contador a cero, la salida de la compuerta AND fija un valor igual a cero pero de un bit, luego el negador convierte este valor en una señal lógica alta (1). El bit de salida de este bloque es el que activa el Flip Flop (FF) de la salida y los reset del módulo contador y acumulador para preparar el módulo para un nuevo set de datos.
- **Parallel Multiplier:** Multiplicador paralelo. Este bloque realiza la operación de multiplicación de la señal de ruido por si misma, en otras palabras eleva al cuadrado la señal. A este bloque ingresa una señal de 12 bits de entrada, mientras que en su salida obtiene una señal de 24 bits.
- **Accumulator:** Acumulador. Su función principal es realizar la operación de suma acumulativa del valor de potencia de la señal. Al terminar de acumular las N_{th} muestras, se resetea el acumulador con la señal de salida del bloque de compuertas AND y NOT. La entrada de este bloque es de 24 bits, mientras que la salida es de 34 bits. La cantidad de bits de salida se justifican con la cantidad de bits que se agregan de acumular una señal de 24 bits en el peor de los casos (con N_{th} máximo).
- **Flip-Flop D:** La función principal de este bloque es fijar el valor de energía a la salida del módulo detector. De esta forma la salida del módulo solamente cambiará de valor luego de N_{th} muestras de señal. La señal de reloj de este bloque se encuentra conectada a la salida del bloque de compuertas AND y NOT.
- **Divider Generator:** Este componente tiene la función de realizar la normalización por N_{th} de la señal de energía de ruido que se tiene a la salida del Flip-Flop D. Para este bloque se utilizará el núcleo propietario de Xilinx "Divider generator"detallado en la sección D.4.3.

- **Constant:** Constante igual a uno. Este bloque se utiliza para fijar la entrada t_{valid} del núcleo Divider Generator en un valor lógico alto. De esta manera, el módulo siempre está preparado para realizar la operación de división.

Núcleo Divider Generator

El núcleo IP Xilinx LogiCORE Divider Generator crea un circuito para la división de enteros basado en la división sin restauración Radix-2 o la división HighRadix con preescalado. El algoritmo Radix-2 explota la lógica FPGA para lograr una gama de opciones de rendimiento que incluye un ciclo único, y el algoritmo High Radix aprovecha los segmentos DSP con un rendimiento más bajo, pero con reutilización para reducir los recursos. En la Figura D.9 se muestra un diagrama de pines del componente.

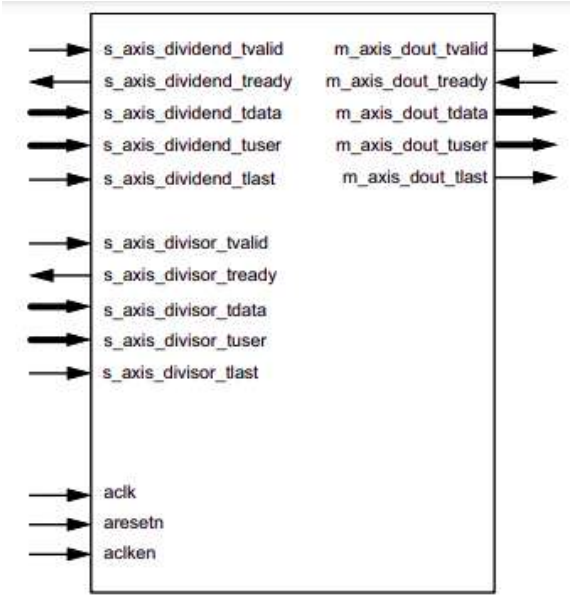


Figura D.9: Pin-out de Divider Generator

Para la solución se utiliza el algoritmo Radix-2 de división, el cual resuelve un bit del cociente por ciclo usando suma y resta. El diseño está completamente canalizado y puede lograr un rendimiento de una división por ciclo de reloj. Como se configurará la cantidad de ciclos de reloj por división en 1, el núcleo está completamente canalizado, por lo que tiene un rendimiento máximo de una división por ciclo de reloj, pero utiliza la mayoría de los recursos. En este caso se está poniendo por encima la velocidad ante la utilización de recursos.

La latencia (número de ciclos de reloj habilitados requeridos antes de que el núcleo genere la primera salida válida) para un divisor completamente encauzado es una función del ancho de bits del dividendo. Como se va utilizar el modo de resto entero para aplicación, la latencia es del orden M, donde M es el ancho del cociente.

El algoritmo Radix-2 permite proveer la solución parametrizada donde divide un dividendo variable de M bits de ancho por un divisor variable de N bits de ancho. La salida consiste en el cociente y un resto entero, donde el resultado de la división es un campo de M bits de ancho para el cociente con un campo de N bits de ancho para el resto entero, como se muestra en la ecuación D.7

$$Dividendo = Cociente \cdot Divisor + Resto \tag{D.7}$$

Es importante la correcta interpretación de los bits de entrada y salida del bloque divisor. Los campos dentro de una interfaz AXI4-Stream siguen una nomenclatura de nombres específica. En este núcleo, los operandos se pasan hacia o desde el núcleo en el puerto de *tdata* del canal. Para facilitar la interoperabilidad con los protocolos orientados a bytes, cada subcampo dentro de *tdata* que podría usarse de forma independiente primero se amplía, si es necesario, para que se ajuste a un campo de bits que es un múltiplo de 8 bits, completando con ceros en los bits más significativos. Para el canal *DOUT* de salida, los campos de resultado se extienden hasta el límite del byte. Los bits agregados son ignorados por el núcleo y no dan como resultado el uso de recursos adicionales.

Los canales de entrada Dividend y Divisor llevan sus operandos en su campo *tdata*. Para cada uno, el operando ocupa los bits menos significativos. El ancho del puerto *tdata* en sí mismo es el múltiplo mínimo de bytes de ancho requerido para contener el operando. En la Figura D.10 se observa la estructura de las entradas de datos, donde *PAD* son los bits que se agregan a los datos. Esta cantidad de bits agregados se calcula como la diferencia entre los anchos de los canales *S_AXIS_DIVIDENT_TDATA* y *DIVIDENT* o *DIVISOR*. Esta cantidad de bits es fija y se replica a la salida.

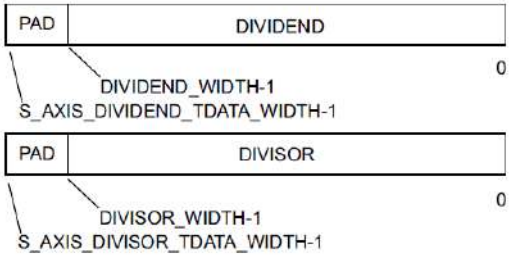


Figura D.10: Estructura de entrada de datos

La estructura de *m_axis_dout_tdata* (bus de salida de datos) es más compleja. Este puerto contiene salidas de cociente y, si las hay, de resto.Las dos salidas (cociente y resto) se consideran separadas y, por lo tanto, están orientadas a bytes antes de concatenarse para generar la señal *m_axis_dout_tdata*. En la Figura D.11 se observa la estructura de la salida de datos.

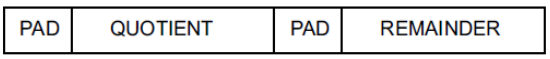


Figura D.11: Estructura de salida de datos

D.5. Módulo de decisión

Este módulo tiene como función principal comparar los valores de umbral y energía, cuando la energía es mayor que el umbral debe enviar un valor lógico alto a la salida, mientras que si la energía es menor que el umbral la salida debe tener un valor lógico bajo. En la Figura D.12 se observa un esquema del componente que se sintetizará.

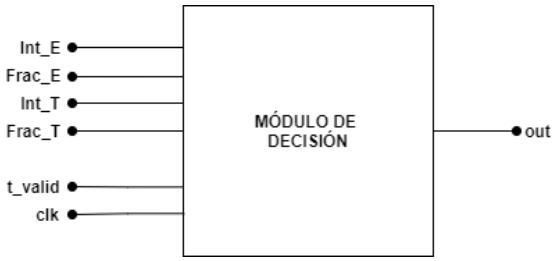


Figura D.12: Pin-out del módulo del decisión

En la Figura D.13 se muestra un diagrama de flujo de la operación del módulo de decisión:

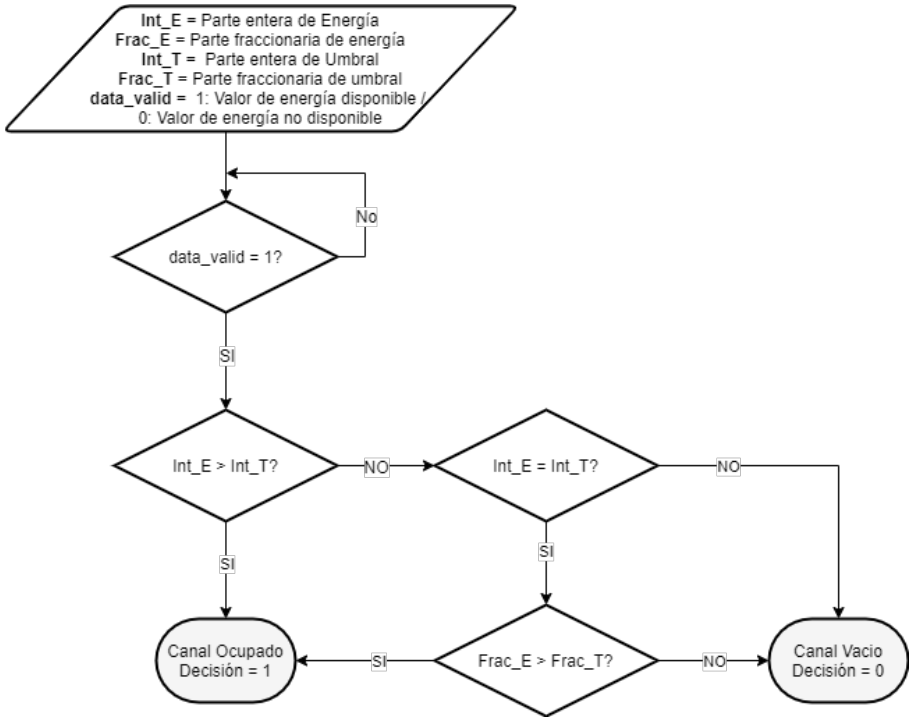


Figura D.13: Flujo del módulo de decisión

Se realizará un componente en lenguaje VHDL que cumpla con dicha función siguiendo el presente pseudocódigo.

```
1   if Data_valid = '1' then
2     if rising_edge(clk) then
3     if (unsigned(Int_Energy)>unsigned(Int_Threshold)) then
4      Decision <= '1';
5     else if (unsigned(Int_Energy)=unsigned(Int_Threshold)
6        and unsigned(Frac_Energy)>unsigned(Frac_Threshold)) then
7            Decision <='1';
8         else
9            Decision <= '0';
10        end if;
11    end if;
12
13  end if;
14
15  else
16    Decision <= '0';
17  end if;
```

D.6. Módulo de comunicación y control

Este módulo tiene como principal objetivo ofrecer una vía de comunicación entre el circuito en el interior de la FPGA y una PC con la interfaz de control y testeo. La familia de FPGAs Zynq-7000 consta de un sistema de procesamiento integrado (PS) de estilo system-on-chip (SoC) y una unidad de lógica programable (PL), que proporciona una solución SoC extensible y flexible. En este caso es necesario la integración en el diseño de bloques correspondiente al PS con el PL. Esto permite que a través de la interfaz UART (Transmisor Receptor Asincrónico Universal) de la FPGA se pueda enviar la información necesaria hacia las entradas del circuito y recibir la información de la salida para ser procesada. En la Figura D.14 se observa un diagrama de detallado con los componentes que integran en este módulo.

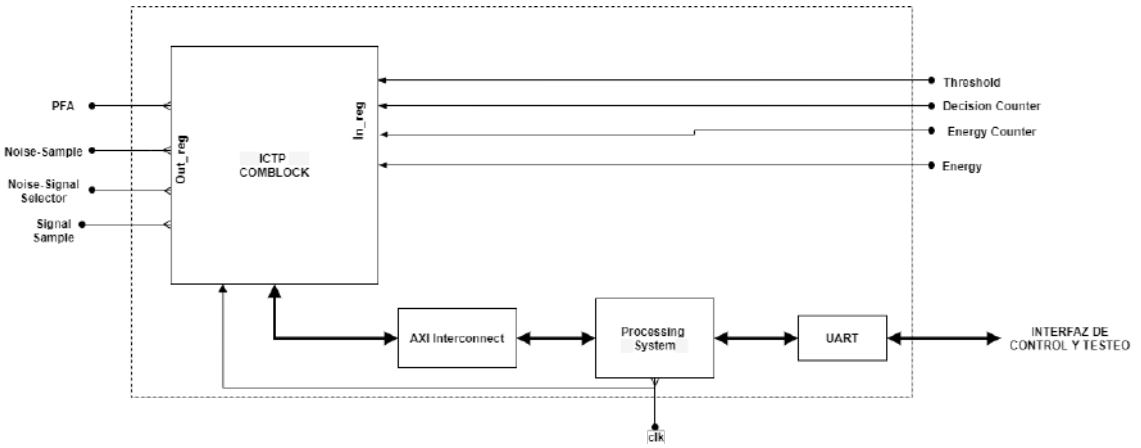


Figura D.14: Módulo de comunicación y control

El módulo esta integrado por los siguientes componentes:

- **ICTP Comblock:** Este componente es el encargado de realizar la comunicación interna entre el PL y el PS de la FPGA. Fue desarrollado por el Centro Internacional de Física Teórica (ICTP). En la sección D.6.1 se detallan las configuraciones necesarias para este componente.
- **AXI Interconnect:** IP de interconexión AXI. Conecta uno o más dispositivos maestros asignados en memoria AXI a uno o más dispositivos esclavos asignados en memoria. En este caso es el encargado de conectar los pines AXI del bloque ICTP Comblock con los del Procesing System.
- **Processing System:** El sistema de procesamiento IP es la interfaz de software del sistema de procesamiento Zynq-7000.
- **UART:** Transmisor Receptor Asincrónico Universal. Es la interfaz física de la FPGA por la cual se conectará el sistema de procesamiento de la FPGA con la interfaz de control y testeo en una PC.

D.6.1. ICTP Comblock

El COMmunication BLOCK (ComBlock) fue creado para proporcionar interfaces conocidas (registros, RAM y FIFOs) a un usuario de la Lógica Programable (PL), evitando la complejidad del bus provisto por el Sistema Procesador (PS), que es

AXI en caso de la versión Vivado. En la Figura D.15 se observa un diagrama del componente.

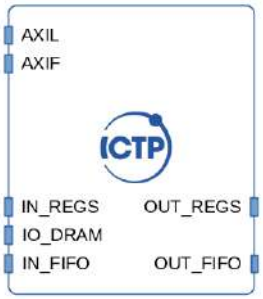


Figura D.15: ICTP ComBlock

Proporciona 5 interfaces para el usuario en el lado FPGA:

- IN_{REGS}: registros de entrada.
- OUT_{REGS}: registros de salida.
- IO_{DRAM}: RAM de puerto dual verdadero de entrada/salida.
- IN_{FIFO}: entrada FIFO.
- OUT_{FIFO}: FIFO de salida.

Por otro lado, también presenta 2 interfaces para el control en el lado del Procesador:

- $AXIL$: AXI4 Lite con los registros y FIFOs.
- $AXIF$: AXI4 Full con la RAM.

Para esta solución solamente se utilizarán los registros de entrada y salida que posee el componente, para ello es necesario realizar las configuraciones mostradas en la Figura D.16.

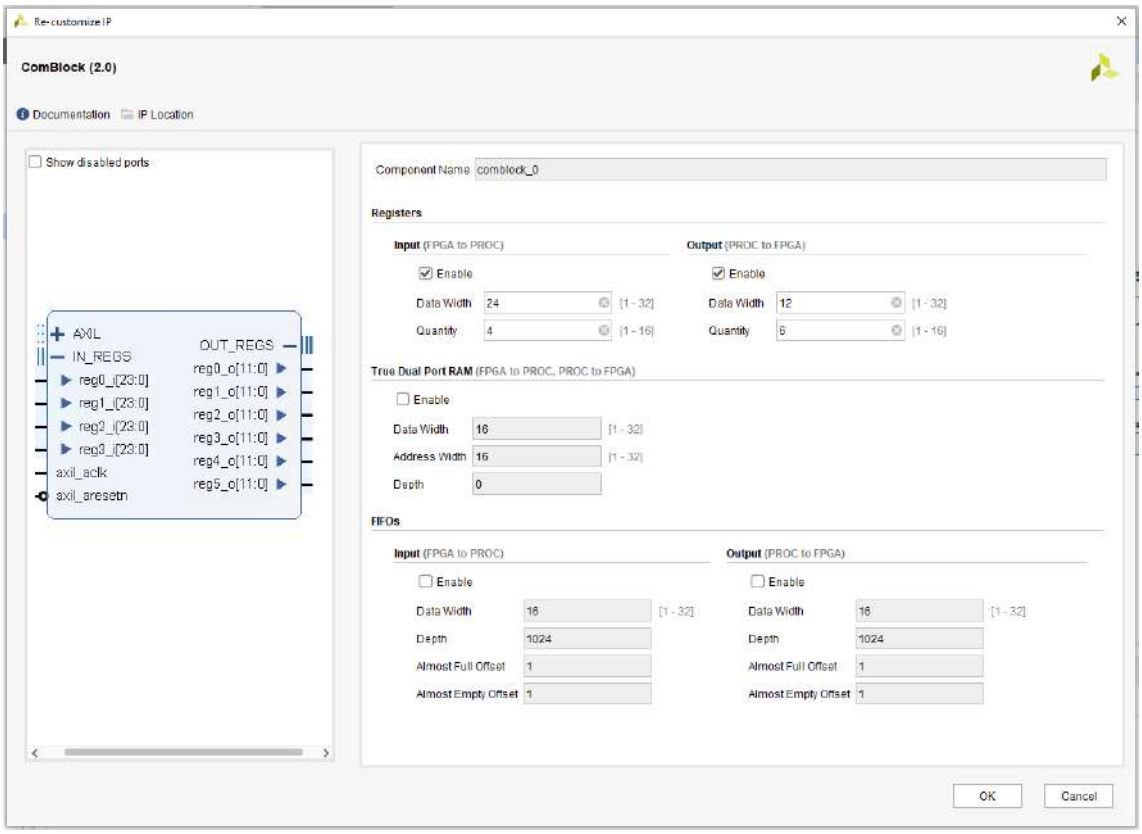


Figura D.16: ConFiguración del ComBlock

Por otro lado es necesario programar el componente en lenguaje C, para ello se proporciona un controlador C para usar con el proyecto. Luego es necesario agregar la librería que incluye ciertas definiciones necesarias para el control de los registros del componente:

```
1    #include "comblock.h"
```

D.7. Interfaz de control y testeo

La interfaz de control y testeo tiene como objetivo la generación de un ambiente automatizado para realizar los tests requeridos en las especificaciones funcionales de la solución. Como la solución se corresponde a un sistema el cual será utilizado por el Laboratorio de Sistemas Caóticos de la Facultad de Ingeniería para realizar pruebas de rendimiento y obtención de resultados para posteriores comparaciones con nuevos algoritmos, se piensa en una interfaz que pueda automatizar las pruebas y elaborar informes de resultados.

Esta plataforma será programada en lenguaje Python debido a la disponibilidad de librerías para comunicación serie con la FPGA y por las ventajas que trae este lenguje en cuanto a su baja complejidad de programación. En la Figura D.17 se muestra un diagrama de flujo de la operación de la interfaz para realizar el testeo del circuito.

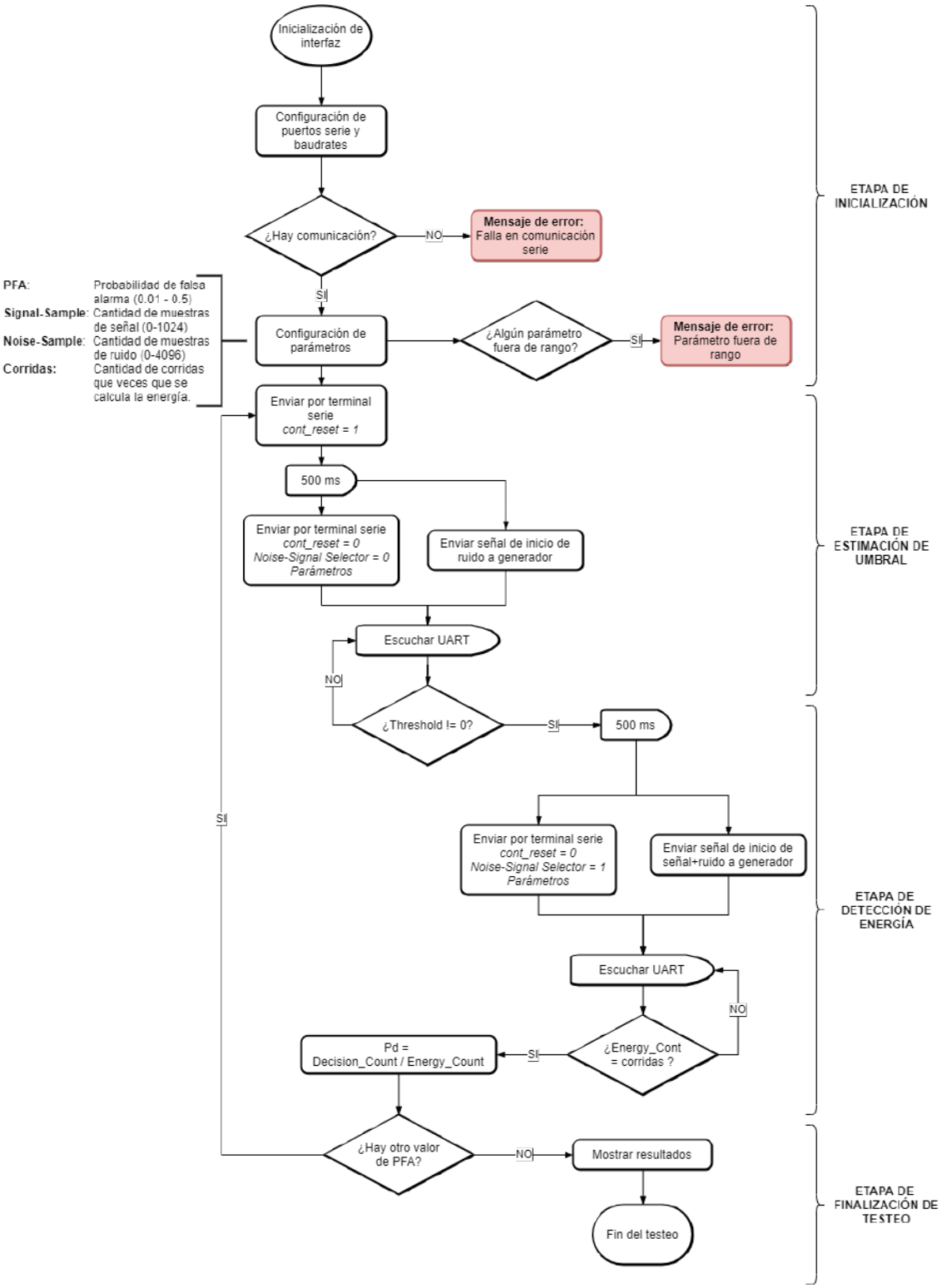


Figura D.17: Lógica de interfaz de control y testeo

Como se puede observar en el diagrama la interfaz contará con 4 etapas de procesamiento:

- **Etapa de inicialización**: Corresponde al inicio de la interfaz, en principio se debe configurar y testear la interfaz serie que conecta con la UART de la FPGA, si esta interfaz no funciona se debe advertir con un error. Para testearla se enviará un mensaje que la FPGA debe responder con un OK. Una vez que la interfaz serie se encuentre correctamente conectada es necesario cargar los

parámetros de entrada para el circuito estos son:

- **PFA:** Será posible elegir una P_{FA} o varias para que el script las recorra, por ejemplo para generar una curva ROC. Este valor puede variar entre 0.01 y 0.5.
- **Signal-Sample:** Será posible elegir la cantidad de muestras de señal que se muestrearán para calcular la energía. Este valor puede variar entre 0 y 1024.
- **Noise-Sample:** Será posible elegir la cantidad de muestras de ruido que se muestrearán para calcular la potencia estimada. Este valor puede variar entre 0 y 4096.
- **Corridas:** Será posible elegir la cantidad de corridas que se realizaran para el cálculo de la energía. Es decir, cuantas veces se calculará la energía de la señal con el mismo valor de umbral. Luego este dato servirá para calcular la Probabilidad de detección (Pd).

Si alguno de los parámetros se encuentra fuera del rango establecido se enviará un mensaje de error advirtiendo la situación.

- **Etapa de estimación de umbral:** Corresponde a los pasos necesarios para la estimación del umbral. El primer paso es resetear el contador de salida, luego se deben enviar los parámetros a la FPGA y la señal de inicio al generador de ruido. Luego se debe estar escuchando constantemente la terminal serie para que al momento que el umbral se haga distinto de cero se cambie de etapa.

- **Etapa de detección de energía:** Corresponde a los pasos necesarios para la detección de energía. Es necesario enviar los parámetros a la FPGA y la señal de inicio de señal y ruido al generador. Luego se debe estar escuchando constantemente la terminal serie para que al momento que la salida del contador de energía sea igual a la cantidad de corridas configuradas el script calcule el la probabilidad de detección como se muestra en la ecuación D.8. Donde $Count_{Decisión}$ corresponde a la cantidad de veces que se detectó el canal como ocupado, y $Count_{Energy}$ refiere a la cantidad de corridas.

$$P_d = \frac{Count_{Decision}}{Count_{Energy}} \tag{D.8}$$

- **Etapa de finalización de testeo:** Una vez completadas la cantidad de corridas, se debe consultar si existe otra P_{FA} para la cual realizar el proceso. Si la respuesta es afirmativa entonces es necesario volver a comenzar el proceso con los nuevos parámetros, es decir ir a la **etapa de estimación de umbral**. Si por el contrario, no existe ninguna P_{FA} más con la que realizar testeos entonces se genera un informe completo con los resultados y se dá por finalizado el script.

Apéndice E

Plan de pruebas

Universidad Nacional de Mar del Plata
Facultad de Ingeniería

Carrera: Ingeniería Electrónica

Proyecto: Implementación en FPGA de algoritmos de sensado espectral para Radio Cognitiva

PLAN DE PRUEBAS

Alcance
El siguiente plan de pruebas abarca desde las pruebas unitarias de los módulos, hasta las pruebas finales de homologación del instrumento como un conjunto.

Ambientes de prueba
Se realizan 3 ambientes de prueba:
- PC con python
- PC con python + FPGA
- PC con python + FPGA + Arduino

Recursos, herramientas e instrumentos
- Placa de desarrollo Zedboard de Xilinx
- Placa de desarrollo Arduino con microcontrolador ATMega328P
- PC con python + Matlab + Vivado

Políticas de trabajo
Las pruebas que se puedan realizar solo con el uso de la PC y/o la placa Arduino se podrán realizar tanto en el Laboratorio de Sistemas Caóticos como en el hogar del estudiante.
El resto de ellas serán desarrolladas en las instalaciones del Laboratorio, ya que requieren del uso de la placa FPGA.

Estrategia de Comunicación
Los resultados de las pruebas se comunicarán en reuniones con los directores de trabajo final Luciana De Micco y Maximiliano Antonelli.

Módulo	ID	Prueba	Tipo de Prueba	Procedimiento	Instrumental	Resultado Esperado	Fecha de Prueba	Resultado Obtenido	Observaciones
Detector de Energía	1	Detección de energía.	Unitaria	- Ingresar con señal senoidal conocida generada en Matlab. - Sensar la salida del módulo para 1024 muestras.	Vivado FPGA Matlab	El valor de energía a la salida del módulo debe coincidir con el valor calculado con Matlab.	25/12/2023	Resultado de acuerdo al esperado.	
Estimador de Umbral	2	Estimación de umbral	Unitaria	- Ingresar con señal de ruido gaussiano generada en Matlab. - Sensar la salida del módulo para 1024 muestras.	Vivado FPGA Matlab	El valor de umbral a la salida del módulo debe coincidir con el valor calculado con Matlab.	25/12/2023	Resultado de acuerdo al esperado.	
Detector de potencia	3	Estimación de potencia de ruido	Unitaria	- Ingresar con señal de ruido gaussiano generada en Matlab. - Sensar la salida del módulo para 1024 muestras.	Vivado FPGA Matlab	El valor de potencia de ruido a la salida del módulo debe coincidir con el valor calculado con Matlab.	25/12/2023	Resultado de acuerdo al esperado.	
Decisión	4	Toma de decisión	Unitaria	- Ingresar al módulo con distintos valores de energía y umbral, simulados mediante un archivo testbench. - Sensar la salida del módulo.	Vivado	La salida debe estar en 1 cuando el valor de energía es mayor al valor de umbral y en 0 cuando el valor de umbral es mayor al de energía.	15/10/2022	Resultados de acuerdo a los esperados.	
Comunicación y Control	5	Comunicación serie	Unitaria	- Iniciar la comunicación serie entre la FPGA y la PC, utilizar el modo debug de SDK. - Abrir una terminal serie en la PC para observar los mensajes. - Enviar mensajes a traves de la terminal.	Vivado SDK FPGA Terminal	- En la terminal de la PC se debe observar que la FPGA se comunica con la misma enviando mensajes. - Al enviar un mensaje hacia la FPGA esta debe interpretarlos, si se trata de un mensaje aleatorio debe responder un "NACK" anunciando que recibió el mensaje y no lo pudo interpretar. Si se trata de un mensaje mapeado en la comunicación para testeo debe devolver un "ACK" anunciando que recibió el mensaje y lo pudo interpretar.	8/11/2022 - 14/11/2022	Resultados de acuerdo a los esperado. La FPGA y la PC se comunican correctamente ante un mensaje aleatorio envia NACK, ante un mensaje dentro del código envia "ACK".	La prueba se repitió varias veces entre las fechas especificadas, llegando al resultado esperado. En un principio se tuvo algunas dificultades con los datos que se representan con mas de un byte y son enviados hacia la FPGA, este inconveniente se solucionó agregando lógica al código que se ejecuta en el microprocesador embebido.
	6	Control	Unitaria	- Programar mediante SDK valores fijos al sistema de procesamiento. - Iniciar la comunicación entre la lógica programable y el sistema de procesamiento de la FPGA. - Abrir terminal serie en la PC. - Observar si la lógica programable responde a las órdenes del sistema de procesamiento.	Vivado SDK FPGA Terminal	En la terminal de la PC se debe observar que la FPGA responde a las ordenes que se encuentran fijadas en el sistema de procesamiento de la misma.	2/2/2023 - 20/2/2023	Resultados de acuerdo a los esperados.	La prueba se repitió varias veces entre las fechas especificadas, llegando al resultado esperado. Se solucionaron varios inconvenientes relacionados con la interpretación y los tipos de datos que envía y recibe la FPGA. Por otro lado se realizaron algunos cambios en la lógica programable que tuvieron que ver principalmente con el agregado de instancias de reseteo de algunos componentes.
Interfaz de testeo	7	Configuración de UART	Unitaria	- Ingresar valores en la sección de configuración de UART.	PC con Python	El sistema debe verificar los valores ingresados y dar un error en caso de que no pueda comunicarse con la interfaz serie que se especificó.	30/10/2022	Resultados de acuerdo a los esperados.	
	8	Configuración de parámetros	Unitaria	- Ingresar valores en la sección de configuración de parámetros de testeo.	PC con Python	El sistema debe verificar los parámetros ingresados, si alguno se encuentra fuera de los rangos estipuladores debe dar error, sino debe escribir un archivo en donde se indiquen todos los parámetros seteados.	30/10/2022	Restados de acuerdo a los esperados	
	9	Exportación de datos	Unitaria	- Presionar el botón de exportar datos. - Presionar el botón de salir y elegir la opción de exportar antes de salir.	PC con Python	En ambos casos se debe generar una carpeta de exportación en donde se copien los datos de configuración en un archivo .txt y además los datos de Energía y Umbral en función del tiempo y de Probabilidad de detección en función de la probabilidad de falta alarma.	30/10/2022	Resultados de acuerdo a los esperados.	
	10	Graficas	Unitaria	- Iniciar la interfaz de testeo. - En los archivos de datos: SN.csv y ROC.csv cargar datos aleatoreos.	PC con Python	En la interfaz de testeo se debe ver como las graficas se actualizan automáticamente y se muestran los valores cargados en los archivos.	30/10/2022	Los resultados no fueron los esperados, la interfaz se quedaba tildada mientras se realizaban los calculos.	Se rediseñó la sección de gráficos de la interfaz y se llevó el procesamiento durante el testeo a un subproceso para que no se tildara el resto de la interfaz mientras se realizaba el mismo.
	11	Comunicación con UART	Unitaria	- Iniciar una comunicación serie con una interfaz serie conectada conectada a una placa arduino.	PC con Python Arduino	En la placa arduino se debe observar que se reciben los datos enviados por la interfaz y a su vez la interfaz puede recibir datos de la placa y procesarlos correctamente.	30/10/2022	Resultados de acuerdo a los esperados.	
	12	Simulación de testeo satisfactorio	Unitaria	- Conectar una placa Arduino que simule ser la FPGA a la interfaz serie. - Iniciar la interfaz de testeo y configurar la UART y los parámetros. - Iniciar testeo.	PC con Python Arduino	- Cuando la placa arduino reciba las distintas ordenes debe enviar los valores que enviaría la FPGA para simular un testeo real. - La interdaz debe recibir los datos, procesarlos y armar los gráficos.	30/10/2022	Resultados de acuerdo a los esperados.	

Módulo	ID	Prueba	Tipo de Prueba	Procedimiento	Instrumental	Resultado Esperado	Fecha de Prueba	Resultado Obtenido	Observaciones
	13	Simulación de testeo fallido	Unitaria	- Conectar una placa Arduino que simule ser la FPGA a la interfaz serie. - Iniciar la interfaz de testeo y configurar la UART y los parámetros. - Iniciar testeo. - Durante el testeo desconectar la UART o configurarla en otra terminal para interrumpir la comunicación.	PC con Python Arduino	- Cuando la placa arduino reciba las distintas ordenes debe enviar los valores que enviaría la FPGA para simular un testeo real. - La interdaz debe recibir los datos, procesarlos y armar los gráficos. - Al cortar la comunicación el sistema debe fallar pero realizar los reintentos programados, si los reintentos fallan debe consultar sobre el problema y si se desea volver a intentar.	30/10/2022	Resultados de acuerdo a los esperados.	
Implementación completa (Circuito en FPGA e interfaz de testeo)	14	Simulación Simulink	Integral	- Realizar una simulación integral del diseño previo a sintetizarse con la herramienta System Generator - Simulink de Matlab y Vivado. - Utilizar señal de entrada de ruido gaussiano para la detección de umbral y luego una señal modulada en BPSK contaminada con ruido para la detección de la energía. - Graficar curvas ROC (Probabilidad de detección vs Probabilidad de Falsa Alarma) para PFA entre 0,1 y 0,5.	System Generator Matlab Vivado Simulink	- Cuando se ingresa con señal de ruido al sistema debe calcular el umbral. - Cuando se ingresa con señal puramente senoidal al sistema debe detectar siempre. - Cuando se ingresa con señal contaminada con ruido no debe detectar nunca. - Observar que curvas ROC se correspondan con las teóricas.	4/12/2022	Resultados de acuerdo a los esperados.	
	15	Simulación testbench	Integral	- Realizar una simulación integral del diseño una vez sintetizado con la herramienta Vivado y la utilización de un archivo testbench. - Utilizar señal de entrada de ruido gaussiano para la detección de umbral y luego una señal modulada en BPSK contaminada con ruido para la detección de la energía. - Graficar curvas ROC (Probabilidad de detección vs Probabilidad de Falsa Alarma) para PFA entre 0,1 y 0,5.	Vivado	- Cuando se ingresa con señal de ruido al sistema debe calcular el umbral. - Cuando se ingresa con señal puramente senoidal al sistema debe detectar siempre. - Cuando se ingresa con señal contaminada con ruido no debe detectar nunca. - Observar que curvas ROC se correspondan con las teóricas.	10/12/2022 - 20/01/2023	Resultados de acuerdo a los esperados.	Se realizaron diversas pruebas con distintos escenarios de señal y ruido. En el rango de fechas de pruebas hubo un receso de dos semanas en los que no se trabajó sobre el proyecto.
	16	TEST 1: Primera aproximación	Homologación	- Simular una entrada que sea la suma de dos variables aleatorias Gausianas, ruido y señal. Estas señales son programadas y enviadas a la FPGA por un microncontrolador ATMega328 mediante un placa Arduino. - Iniciar interfaz de testeo. - Configurar UART y parámetros. - Iniciar testeo. - Observar gráficas.	FPGA Vivado Arduino PC con Python	- Se debe observar en las gráficas que hay pocas o ninguna detección ya que se trata de dos señales de ruido. - En la curva ROC la probabilidad de detección debe dar número muy bajos.	20/02/2023 - 10/03/2023	Resultados de acuerdo a los esperados. Se observan bajos valores en la probabilidad de detección.	Se hicieron varias pruebas consecutivas con distintos valores de ruido y señal, ambos como variables aleatorias gaussianas. Se graficaron las ROC de todas las pruebas en un mismo gráfico para una mejor comparación.
	17	TEST 2: Segunda aproximación	Homologación	- Simular una entrada que sea la suma de una variable aleatoria Gausiana (ruido) y una señal modulada en BPSK con f0 y rb variable. Estas señales son programadas y enviadas a la FPGA por un microncontrolador ATMega328 mediante un placa Arduino. - Iniciar interfaz de testeo. - Configurar UART y parámetros. - Iniciar testeo. - Observar gráficas.	FPGA Vivado Arduino PC con Python	- Se debe observar en las gráficas que las detecciones aumentan cuando aumenta la Probabilidad de Falsa Alarma. - La curva ROC final debe tener similitud con la teorica y la simulada.	20/02/2023 - 10/03/2023	Resultados de acuerdo a los esperados. Se observa que la probabilidad de detección aumenta a medida que aumenta la probilidad de falsa alarma.	Se hicieron varias pruebas cambiando la relación señal a ruido (SNR) de la señal senoidal, y cambiando la potencia de ruido. Para cada SNR y potencia de ruido se realizaron tres pruebas consecutivas que se graficaron en un mismo gráfico para una mejor comparación.